Table of Contents

Lesson 1

JAVA FEATURES

This handout is a traditional introduction to any language features. You might not be able to comprehend some of the features fully at this stage but don't worry, you'll get to know about these as we move on with the course

Design Goals of Java

The massive growth of the Internet and the World-Wide Web leads us to a completely new way of looking at development of software that can run on different platforms like Windows, Linux and Solaris etc.

Right Language, Right Time

Java came on the scene in 1995 to immediate popularity.

Before that, C and C++ dominated the software development

1. compiled, no robust memory model, no garbage collector causes memory leakages, not great support of built in libraries

Java brings together a great set of "programmer efficient" features

2. Putting more work on the CPU to make things easier for the programmer.

Java – Buzzwords (Vocabulary)

From the original Sun Java whitepaper: "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high- performance, multi-threaded, and dynamic language."

Here are some original java buzzwords...

Java -- Language + Libraries

Java has two parts...

1. The core language -- variables, arrays, objects
   - The Java Virtual Machine (JVM) runs the core language
   - The core language is simple enough to run on small devices -- phones, smart cards, PDAs.
2. The libraries
   - Java includes a large collection of standard library classes to provide "off the shelf" code. (Useful built-in classes that comes with the language to perform basic tasks)
   - Example of these classes is String, ArrayList, HashMap, StringTokenizer (to break string into substrings), Date ...
   - Java programmers are more productive in part because they have access to a large set of standard, well documented library classes.

Simple

Very similar C/C++ syntax, operators, etc.

The core language is simpler than C++ -- no operator overloading, no pointers, and no multiple inheritance

The way a java program deals with memory is much simpler than C or C++.

Object-Oriented

Java is fundamentally based on the OOP notions of classes and objects.

Java uses a formal OOP type system that must be obeyed at compile-time and run-time.

This is helpful for larger projects, where the structure helps keep the various parts consistent. Contrast to Perl, which has a more anything-goes feel.

Distributed / Network Oriented

Java is network friendly -- both in its portable, threaded nature, and because common networking operations are built-in to the Java libraries.

Robust / Secure / Safe

- Java is very robust
  - Both, vs. unintentional errors and vs. malicious code such as viruses.
  - Java has slightly worse performance since it does all this checking. (Or put the other way, C can be faster since it doesn't check anything.)
- The JVM "verifier" checks the code when it is loaded to verify that it has the correct structure -- that it does not use an uninitialized pointer, or mix int and pointer types. This is one-time "static" analysis -- checking that the code has the correct structure without running it.
- The JVM also does "dynamic" checking at runtime for certain operations, such as pointer and array access, to make sure they are touching only the memory they should. You will write code that runs into
- As a result, many common bugs and security problems (e.g. "buffer overflow") are not possible in java. The checks also make it easier to find many common bugs easy, since they are caught by the runtime checker.
- You will generally never write code that fails the verifier, since your compiler is smart enough to only generate correct code. You will write code that runs into the runtime checks all the time as you debug -- array out of bounds, null pointer.
- Java also has a runtime Security Manager can check which operations a particular piece of code is allowed to do. As a result, java can run un-trusted code in a "sandbox" where, for example, it can draw to the screen but cannot access the local file system.

Portable

"Write Once Run Anywhere", and for the most part this works.

Not even a recompile is required -- a Java executable can work, without change, on any Java enabled platform.

Support for Web and Enterprise Web Applications

Java provides an extensive support for the development of web and enterprise applications Servlets, JSP, Applets, JDBC, RMI, EJBs and JSF etc. are some of the Java technologies that can be used for the above mentioned purposes.

High-performance

The first versions of java were pretty slow.

Java performance has gotten a lot better with aggressive just-in-time-compiler (JIT) techniques.

Java performance is now similar to C -- a little slower in some cases, faster in a few cases. However memory use and startup time are both worse than C.

Java performance gets better each year as the JVM gets smarter. This works, because making the JVM smarter does not require any great change to the java language, source code, etc.

Multi-Threaded

Java has a notion of concurrency wired right in to the language itself.

This works out more cleanly than languages where concurrency is bolted on after the fact.

Dynamic

Class and type information is kept around at runtime. This enables runtime loading and inspection of code in a very flexible way.

Java Compiler Structure

The source code for each class is in a .java file. Compile each class to produce
".class" file.

Sometimes, multiple .class files are packaged together into a .zip or .jar "archive"
file.

On unix or windows, the java compiler is called "javac". To compile all the .java files in a directory use "javac *.java".

Java: Programmer Efficiency

Faster Development
Building an application in Java takes about 50% less time than in C or C++. So, faster time to market
Java is said to be "Programmer Efficient".

OOP

Java is thoroughly OOP language with robust memory system

Memory errors largely disappear because of the safe pointers and garbage collector. The lack of memory errors accounts for much of the increased programmer productivity.

Libraries

Code re-use at last -- String, ArrayList, Date, ... available and documented in a standard way

Microsoft vs. Java

Microsoft hates Java, since a Java program (portable) is not tied to any particular operating system. If Java is popular, then programs written in Java might promote non-Microsoft operating systems. For basically the same reason, all the non- Microsoft vendors think Java is a great idea.

Microsoft's C# is very similar to Java, but with some improvements, and some questionable features added in, and it is not portable in the way Java is. Generally it is considered that C# will be successful in the way that Visual Basic is: a nice tool to build Microsoft only software.

Microsoft has used its power to try to derail Java somewhat, but Java remains very popular on its merits.

Java Is For Real

Java has a lot of hype, but much of it is deserved. Java is very well matched for many modern problems

Using more memory and CPU time but less programmer time is an increasingly appealing tradeoff.

Robustness and portability can be very useful features

A general belief is that Java is going to stay here for the next 10-20 years

References

Majority of the material in this handout is taken from the first handout of course cs193j at Stanford.
The Java™ Language Environment, White Paper, by James Gosling & Henry McGilton

Java's Sun site: http://java.sun.com
Java World: www.javaworld.com

Lesson 2

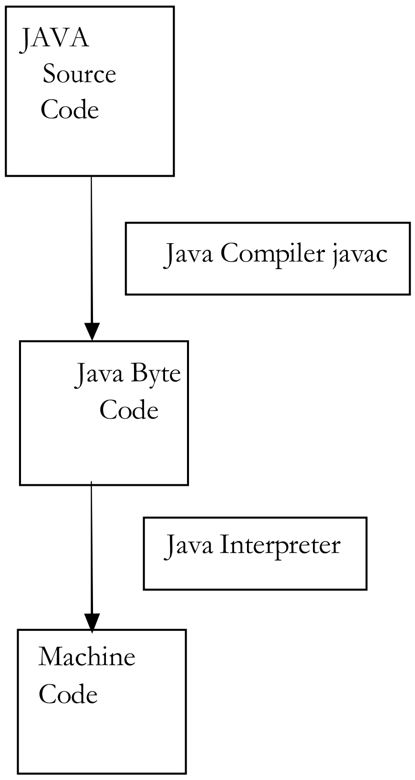
Java Virtual Machine & Runtime Environment

Basic Concept

When you write a program in C++ it is known as source code. The C++ compiler converts this source code into the machine code of underlying system (e.g. Windows) If you want to run that code on Linux you need to recompile it with a Linux based compiler. Due to the difference in compilers, sometimes you need to modify your code.

Java has introduced the concept of WORA (write once run anywhere). When you write a java program it is known as the source code of java. The java compiler does not compile this source code for any underlying hardware system; rather it compiles it for a software system known as JVM (This compiled code is known as byte code). We have different JVMs for different systems (such as JVM for Windows, JVM for Linux etc). When we run our program the JVM interprets (translates) the compiled program into the language understood by the underlying system. So we write our code once and the JVM runs it everywhere according to the underlying system.

This concept is discussed in detail below

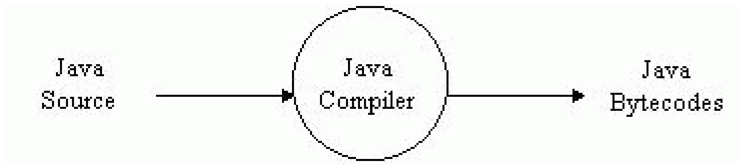


Bytecode

Java programs (Source code) are compiled into a form called Java bytecodes.

The Java compiler reads Java language source (.java) files, translates the source into Java bytecodes, and places the bytecodes into class (.class) files.

The compiler generates one class file for each class contained in java source file.



Java Virtual Machine (JVM)

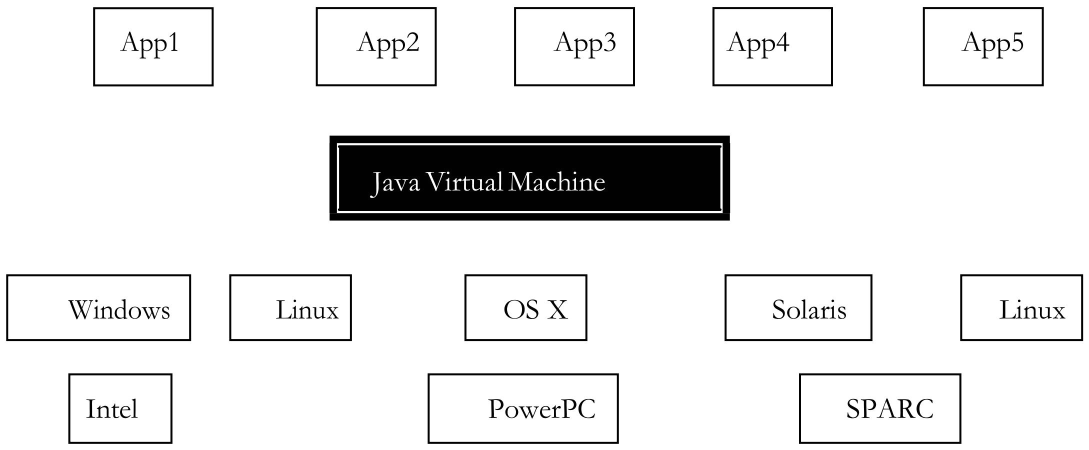
The central part of java platform is java virtual machine

Java bytecode executes by special software known as a "virtual machine".

Most programming languages compile source code directly into machine code, suitable for execution

The difference with Java is that it uses bytecode - a special type of machine code.

The JVM executes Java bytecodes, so Java bytecodes can be thought of as the machine language of the JVM.



- JVM are available for almost all operating systems.
- Java bytecode is executed by using any operating system's JVM. Thus achieve portability.

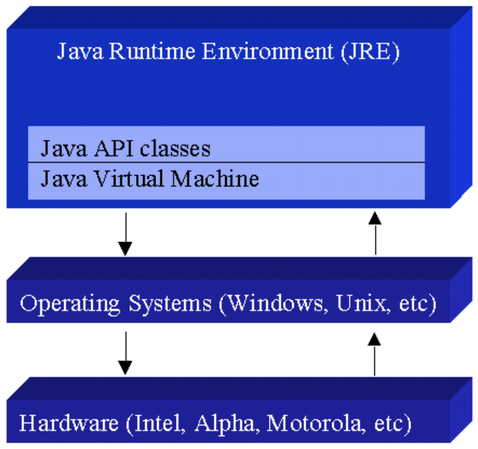
Java Runtime Environment (JRE)

The Java Virtual Machine is a part of a large system i.e. Java Runtime Environment (JRE).

Each operating system and CPU architecture requires different JRE.

The JRE consists of set of built-in classes, as well as a JVM.

Without an available JRE for a given environment, it is impossible to run Java software.


Java Runtime Environment (JRE)
Java API classes
Java Virtual Machine
Operating Systems (Windows, Unix, etc)
Hardware (Intel, Alpha, Motorola, etc)

References

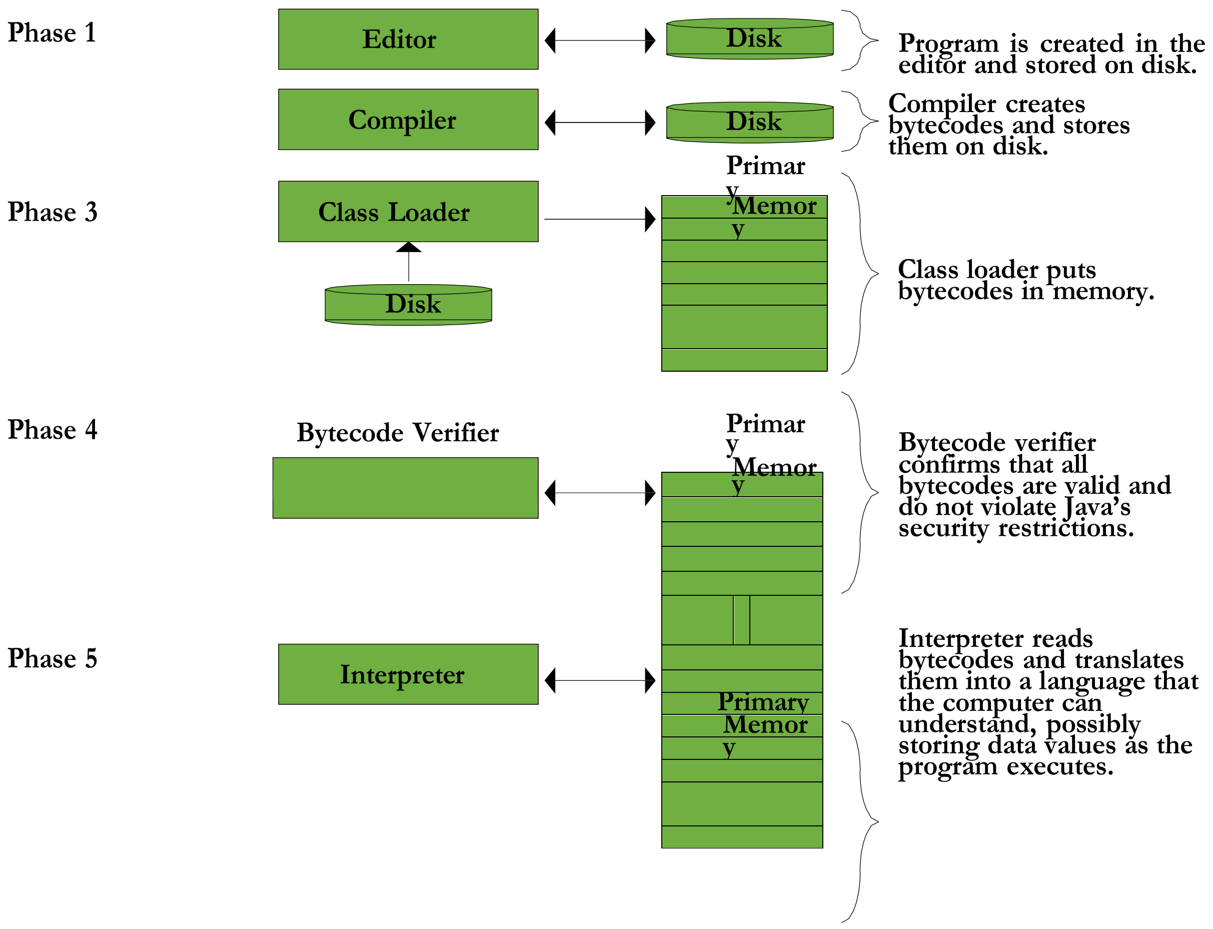
Java World: http://www.javaworld.com
Inside Java: http://www.javacoffeebreak.com/articles/inside_java

Java Program Development and Execution Steps

Java program normally go through five phases. These are

1. Edit,
2. Compile,
3. Load,
4. Verify and
5. Execute

We look over all the above mentioned phases in a bit detail. First consider the following figure that summarizes the all phases of a java program.


Phase 1
Editor
Disk
Program is created in the editor and stored on disk.
Compiler
Disk
Compiler creates bytecodes and stores them on disk.
Phase 3
Class Loader
Disk
Primary Memory
Class loader puts bytecodes in memory.
Phase 4
Bytecode Verifier
Primary Memory
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.
Phase 5
Interpreter
Primary Memory
Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

Phase 1: Edit

Phase 1 consists of editing a file. This is accomplished with an editor program. The programmer types a java program using the editor like notepad, and make corrections if necessary.

When the programmer specifies that the file in the editor should be saved, the program is stored on a secondary storage device such as a disk. Java program file name ends with a
.java extension.

On Windows platform, notepad is a simple and commonly used editor for the beginners. However java integrated development environments (IDEs) such as NetBeans, Borland JBuilder, JCreator and IBM's Ecllipse have built-in editors that are smoothly integrated into the programming environment.

Phase 2: Compile

In Phase 2, the programmer gives the command javac to compile the program. The java compiler translates the java program into bytecodes, which is the language understood by the java interpreter.

To compile a program called Welcome.java, type

```
javac Welcome.java
```

at the command window of your system. If the program compiles correctly, a file called Welcome.class is produced. This is the file containing the bytecodes that will be interpreted during the execution phase.

Phase 3: Loading

In phase 3, the program must first be placed in memory before it can be executed. This is done by the *class loader*, which takes the .class file (or files) containing the bytecodes and transfers it to memory. The .class file can be loaded from a disk on your system or over a network (such as your local university or company network or even the internet).

Applications (Programs) are loaded into memory and executed using the *java interpreter*
via the command java. When executing a Java application called Welcome, the command

```
Java Welcome
```

Invokes the interpreter for the Welcome application and causes the class loader to load information used in the Welcome program.

Phase 4: Verify

Before the bytecodes in an application are executed by the java interpreter, they are verified by the *bytecode verifier* in Phase 4. This ensures that the bytecodes for class that are loaded form the internet (referred to as *downloaded classes*) are valid and that they do not violate Java's security restrictions.

Java enforces strong security because java programs arriving over the network should not be able to cause damage to your files and your system (as computer viruses might).

Phase 5: Execute

Finally in phase 5, the computer, under the control of its CPU, interprets the program one bytecode at a time. Thus performing the actions specified by the program.

Programs may not work on the first try. Each of the preceding phases can fail because of various errors. This would cause the java program to print an error message. The programmer would return to the edit phase, make the necessary corrections and proceed through the remaining phases again to determine id the corrections work properly.

References:

Java™ How to Program 5th edition by Deitel & Deitel
Sun Java online tutorial: http://java.sun.com/docs/books/tutorial/java/index.html

Installation and Environment Setting

Installation

- Download the latest version j2se5.0 (java 2 standard edition) from http://java.sun.com or get it from any other source like CD.

  Note: j2se also called jdk (java development kit). You can also use the previous versions like jdk 1.4 or 1.3 etc. but it is recommended that you use either jdk1.4 or jdk5.0

- Install j2se5.0 on your system

  Note: For the rest of this handout, assume that j2se is installed in C:\Program Files\Java\jdk1.5.0

Environment Setting

Once you successfully installed the j2se, the next step is environment or path setting. You can accomplish this in either of two ways.

- **Temporary Path Setting**

  Open the command prompt from Start Æ Programs Æ Accessories Æ Comman Prompt. The command prompt screen would be opened in front of you.

  Write the command on the command prompt according to the following format

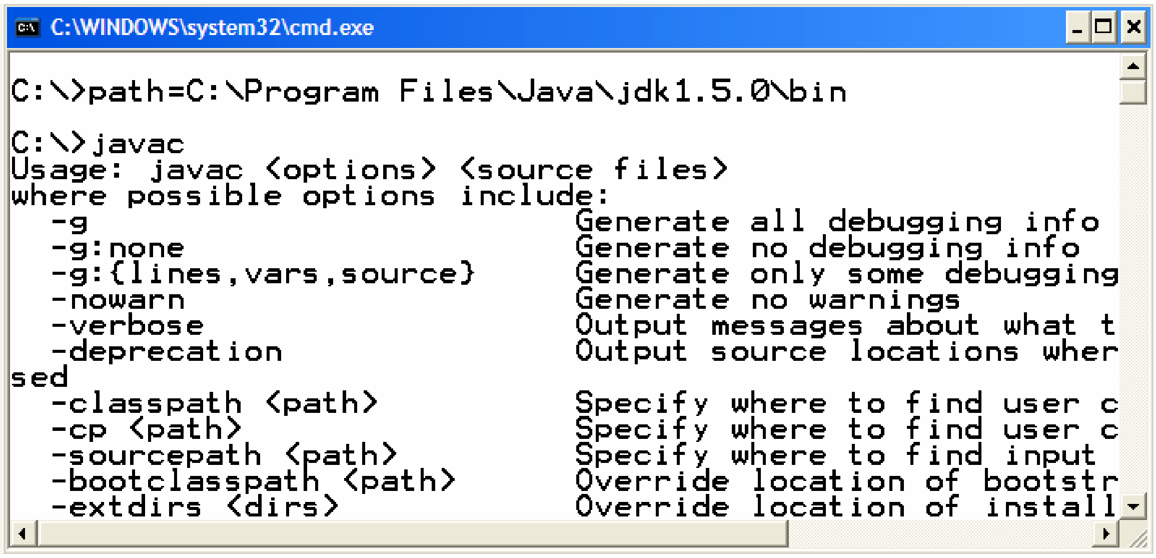
  ```
  path = < java installation directory\bin >
  ```

  So, according to handout, the command will look like this

  ```
  path = C:\Program Files\Java\jdk1.5.0\bin
  ```

  To Test whether path has been set or not, write javac and press ENTER. If the list ofn b options displayed as shown in the below figure means that you have successfully completed the steps of path setting.

The above procedure is illustrates in the given below picture.

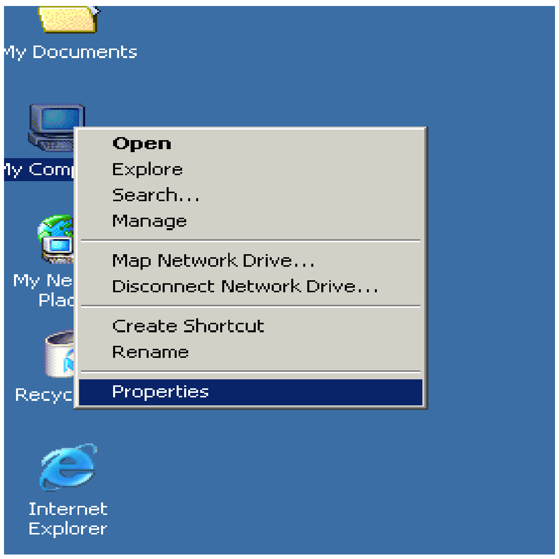


```
C:\>path=C:\Program Files\Java\jdk1.5.0\bin

C:\>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what t
  -deprecation               Output source locations wher
sed
  -classpath <path>          Specify where to find user c
  -cp <path>                 Specify where to find user c
  -sourcepath <path>         Specify where to find input
  -bootclasspath <path>      Override location of bootstr
  -extdirs <dirs>            Override location of install
```

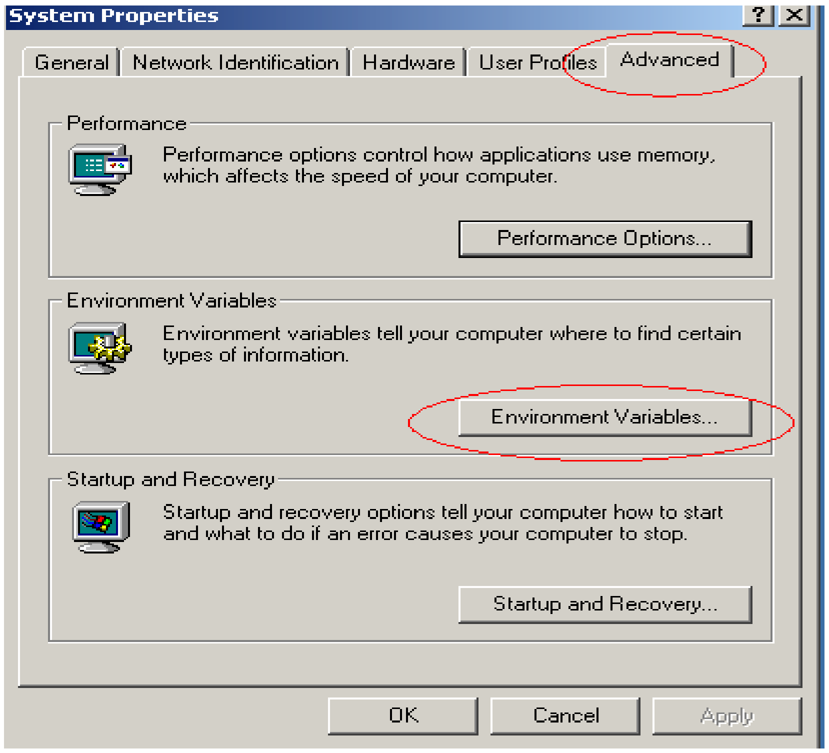
Note: The issue with the temporary path setting is you have to repeat the above explained procedure again and again each time you open a new command prompt window. To avoid this overhead, it is better to set your path permanently

- **Permanent Path Setting**

In Windows NT (XP, 2000), you can set the permanent environment variable.
Right click on **my computer** icon click on properties as shown below

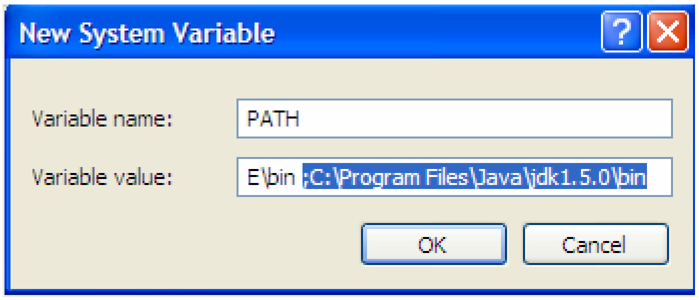


A System Properties frame would appeared as shown in the picture


System Properties
General
Network Identification
Hardware
User Profiles
Advanced
Performance
Performance options control how applications use memory, which affects the speed of your computer.
Performance Options...
Environment Variables
Environment variables tell your computer where to find certain types of information.
Environment Variables...
Startup and Recovery
Startup and recovery options tell your computer how to start and what to do if an error causes your computer to stop.
Startup and Recovery...
OK
Cancel
Apply

Select the advanced tab followed by clicking the Environment Variable button. The Environment variables frame would be displayed in front of you

Locate the Path variable in the System or user variables, if it is present there, select it by single click. Press Edit button. The following dialog box would be appeared.



Variable name: PATH

Variable value: E\bin ;C:\Program Files\Java\jdk1.5.0\bin

- Write; C:**\Program Files\Java\jdk1.5.0\bin** at the end of the value field. Press OK button. Remember to write semicolon (;) before writing the path for java installation directory as illustrate in the above figure

- If Path variable does not exist, click the New button. Write variable name "PATH", variable value **C:\Program Files\Java\jdk1.5.0\bin** and press OK button.

- Now open the command prompt and write **javac**, press enter button. You see the list of options would be displayed.

- After setting the path permanently, you have no need to set the path for each new opened command prompt.

References

Entire material for this handout is taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

First Program in Java

Like any other programming language, the java programming language is used to create applications. So, we start from building a classical "Hello World" application, which is generally used as the first program for learning any new language.

HelloWorldApp

1. Open notepad editor from *Start Æ ProgarmFiles Æ AccessoriesÆ Notepad.*
2. Write the following code into it.

Note: Don't copy paste the given below code. Probably it gives errors and you can't able to remove them at the beginning stage.

```
1.  /* The HelloWorldApp class implements an application that
2.      simply displays "Hello World!" to the standard output.
3.  */
4.  public class HelloWorldApp {
5.     public static void main(String[] args) {
6.              //Display the string. No global main
7.              System.out.println("Hello World");
8.     }
9.  }
```

3. To save your program, move to File menu and choose save as option.
4. Save your program as "HelloWorldApp.java" in some directory. Make sure to add double quotes around class name while saving your program. For this example create a folder known as "examples" in D: drive

   Note: Name of file must match the name of the public class in the file (at line 4). Moreover, it is case sensitive. For example, if your class name is MyClass, than file name must be MyClass. Otherwise the Java compiler will refuse to compile the program.

For the rest of this handout, we assume that program is saved in D:\examples directory.

HelloWorldApp Described

Lines 1-3

Like in C++, You can add multiple line comments that are ignored by the compiler.

Lines 4

Line 4 declares the class name as HelloWorldApp. In java, every line of code must reside inside class. This is also the name of our program (HelloWorldApp.java). The compiler creates the HelloWorldApp.class if this program successfully gets compiled.

Lines 5

Line 5 is where the program execution starts. The java interpreter must find this defined exactly as given or it will refuse to run the program. (However you can change the name of parameter that is passed to main. i.e. you can write String[] argv or String[] some Param instead of String[] args)

Other programming languages, notably C++ also use the main( ) declaration as the starting point for execution. However the main function in C++ is global and reside outside of all classes where as in Java the main function must reside inside a class. In java there are no global variables or functions. The various parts of this main function declaration will be covered at the end of this handout.

Lines 6

Again like C++, you can also add single line comment

Lines 7

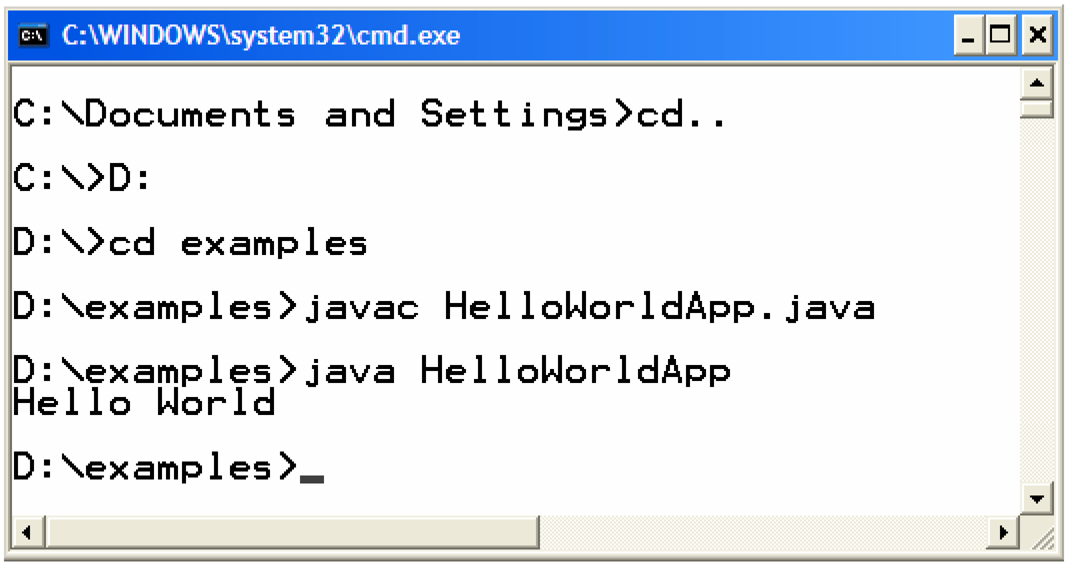
Line 7 illustrates the method call. The println( ) method is used to print something on the console. In this example println( ) method takes a string argument and writes it to the standard output i.e. console.

Lines 8-9

Line 8-9 of the program, the two braces, close the method main( ) and the class HelloWorldApp respectively.

Compiling and Running HelloWorldApp

1. Open the command prompt from *Start Æ Program Files Æ Accessories.* OR alternatively you can write *cmd* in the run command window.
2. Write *cd.*. to came out from any folder, and cd [folder name] to move inside the specified directory. To move from one drive to another, use [Drive Letter]: See figure given below
3. After reaching to the folder or directory that contains your source code, in our case HelloWorldApp.java.
4. Use **"javac"** on the command line to compile the source file (".java" file).
   ```
   D:\examples> javac HelloWorld.java
   ```
5. If program gets successfully compiled, it will create a new file in the same directory named HelloWorldApp.class that contains the byte-code.
6. Use "java" on the command line to run the compiled .class file. Note ".class" would be added with the file name.
   ```
   D:\examples> java HelloWorld
   ```
7. You can see the *Hello World* would be printed on the console. Hurrah! You are successful in writing, compiling and executing your first program in java ☺



```
C:\Documents and Settings>cd..

C:\>D:

D:\>cd examples

D:\examples>javac HelloWorldApp.java

D:\examples>java HelloWorldApp
Hello World

D:\examples>_
```

Points to Remember

- Recompile the class after making any changes
- Save your program before compilation
- Only run that class using java command that contains the main method, because program executions always starts form main

An Idiom Explained

You will see the following line of code often:

- public static void main(String args[]) { …}

- About main()

  "main" is the function from which your program starts

  Why public?

  Since main method is called by the JVM that is why it is kept public so that it is accessible from outside. Remember private methods are only accessible from within the class

  Why static?

  Every Java program starts when the JRE (Java Run Time Environment) calls the main method of that program. If main is not static then the JRE have to create an object of the class in which main method is present and call the main method on that object (In OOP based languages method are called using the name of object if they are not static). It is made static so that the JRE can call it without creating an object.
  Also to ensure that there is only one copy of the main method per class

  Why void?

  - Indicates that main ( ) does not return anything.

  What is String args[] ?

  Way of specifying input (often called command-line arguments) at startup of application. More on it latter

References

Entire material for this handout is taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose with out the consent of author.

Lesson 3

Learning Basics

Strings

A *string* is commonly considered to be a sequence of characters stored in memory and accessible as a unit. Strings in java are represented as objects.

String Concatenation

"+" operator is used to concatenate strings

– System.out.pritln("Hello" + "World") will print Hello World on console

String concatenated with any other data type such as int will also convert that datatype to String and the result will be a concatenated String displayed on console. For example,

– int i = 4;
– int j = 5;

System.out.println ("Hello" + i)
will print *Hello 4* on screen

– However
System,.out..println(i+j) ;
will print 9 on the console because both i and j are of type int.

Comparing Strings

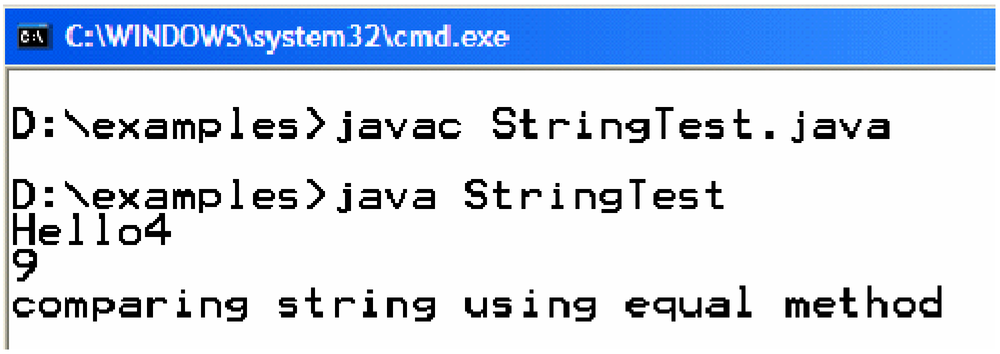
For comparing Strings never use == operator, use *equals* method of String class.

– == operator compares addresses (shallow comparison) while equals compares values (deep comparison)
E.g string1.equals(string2)

Example Code: String concatenation and comparison

```
public class StringTest {
 public static void main(String[] args) {
   int i = 4;
   int j = 5;

   System.out.println("Hello" + i); // will print Hello4
   System.out.println(i + j); // will print 9

   String s1 = new String ("pakistan");
   String s2 = "pakistan";
   if (s1 == s2) {
     System.out.println("comparing string using == operator");

   }



  if (s1.equals( s2) ) {
   System.out.println("comparing string using equal method");
  }
 }
}
```

On execution of the above program, following output will produce


C:\WINDOWS\system32\cmd.exe

```
D:\examples>javac StringTest.java

D:\examples>java StringTest
Hello4
9
comparing string using equal method
```

Taking in Command Line Arguments

In Java, the program can be written to accept *command-line-arguments*.

Example Code: command-line arguments

```
/* This Java application illustrates the use of Java command-line arguments. */

public class CmdLineArgsApp {

  public static void main(String[] args){ //main method

     System.out.println(”First argument ” + args[0]);
     System.out.println(”Second argument ” + args[1]);

  }//end main
}//End class.
```

To execute this program, we pass two arguments as shown below:

```
public void someMethod( ) {
  int x; //local variable
 System.out.println(x); // compile time error
```

- These parameters should be separated by space. .
- The parameters that we pass from the command line are stored as Strings inside the “args” array. You can see that the type of “args” array is String.

Example Code: Passing any number of arguments

In java, array knows their size by using the length property. By using, length property we can determine how many arguments were passed. The following code example can accept any number of arguments

```
/* This Java application illustrates the use of Java
  command-line arguments. */

public class AnyArgsApp {
  public static void main(String[] args){ //main method
   for(int i=0; i < args.length; i++)
   System.out.println(“Argument:” + i + “value” +args[i]);
   }//end main

}//End class.
```

Output

```
C:\java AnyArgsApp i can pass any number of arguments

Argument:0 value i Argument:1 value can
Argument:2 value pass Argument:3 value
any Argument:4 value number Argument:5
value of Argument:6 value arguments
```

Primitives vs Objects

- Everything in Java is an "Object", as every class by default inherits from class "Object" , except a few primitive data types, which are there for efficiency reasons.

- Primitive Data Types
  Primitive Data types of java

boolean, byte	1 byte
char, short	2 bytes
int, float	4 bytes
long, double	8 bytes

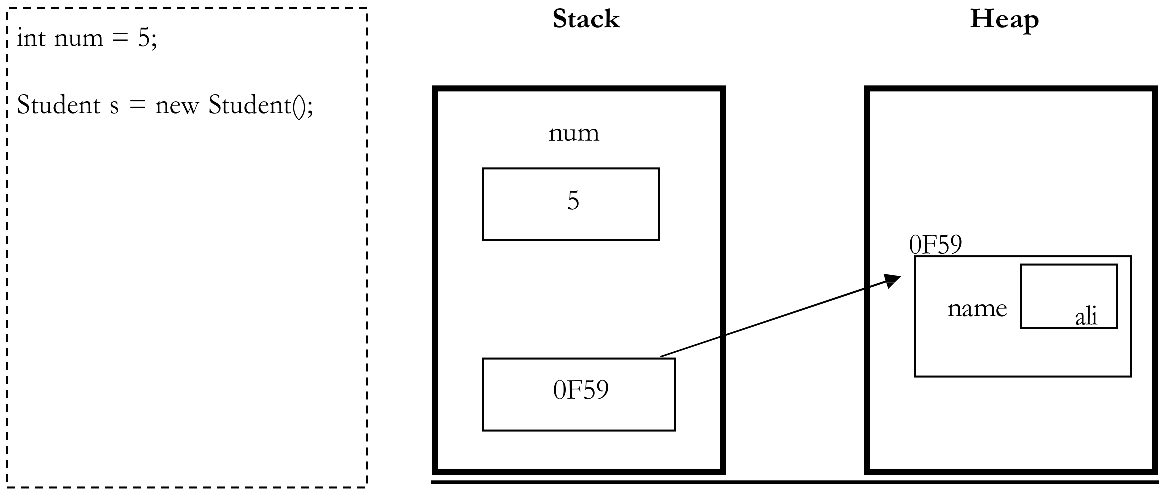
- Primitive data types are generally used for local variables, parameters and instance variables (properties of an object)

- Primitive datatypes are located on the stack and we can only access their value, while objects are located on heap and we have a reference to these objects

- Also primitive data types are always passed by value while objects are always passed by reference in java. There is no C++ like methods
  - void someMethod(int &a, int & b) // not available in java

Stack vs. Heap

Stack and heap are two important memory areas. Primitives are created on the stack while objects are created on heap. This will be further clarified by looking at the following diagram that is taken from Java Lab Course.



Wrapper Classes

Each primitive data type has a corresponding object (wrapper class). These wrapper classes provides additional functionality (conversion, size checking etc.), which a primitive data type cannot provide.

Primitive Data Type	Corresponding Object Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Wrapper Use

You can create an object of Wrapper class using a String or a primitive data type

- Integer num = new Integer(4); or
- Integer num = new Integer("4");

Note: num is an object over here not a primitive data type

You can get a primitive data type from a Wrapper using the corresponding value function

- int primNum = num.intValue();

Converting Strings to Numeric Primitive Data Types

To convert a string containing digits to a primitive data type, wrapper classes can help. *parseXxx* method can be used to convert a String to the corresponding primitive data type.

```
String value = "532";
int  d = Integer.parseInt(value);

String value = "3.14e6";
double d = Double.parseDouble(value);
```

The following table summarizes the parser methods available to a java programmer.

Data Type	Convert String using either …
byte	Byte.parseByte(*string*) new Byte(*string*).byteValue()
short	Short.parseShort(*string*) new Short(*string*).shortValue()
int	Integer.parseInteger(*string*) new Integer(*string*).intValue()
long	Long.parseLong(*string*) new Long(*string*).longValue()
float	Float.parseFloat(*string*) new Float(*string*).floatValue()
double	Double.parseDouble(*string*) new Double(*string*).doubleValue()

Example Code: Taking Input / Output

So far, we learned how to print something on console. Now the time has come to learn how to print on the GUI. Taking input from console is not as straightforward as in C++. Initially we'll study how to take input through GUI (by using JOPtionPane class).

The following program will take input (a number) through GUI and prints its square on the console as well on GUI.

```
1. import javax.swing.*;

2. public class InputOutputTest {

3.  public static void main(String[] args) {

4.    //takes input through GUI
5.    String input = JOptionPane.showInputDialog("Enter number");

6.    int number = Integer.parseInt(input);
7.    int square = number * number;

8.    //Display  square on console
9.    System.out.println("square:" + square);

10.    //Display square on GUI
11.    JOptionPane.showMessageDialog(null, "square:"+ square);

12.    System.exit(0);

13.  }
14. }
```

On line 1, swing package was imported because it contains the *JOptionPane* class that will be used for taking input from GUI and displaying output to GUI. It is similar to header classes of C++.

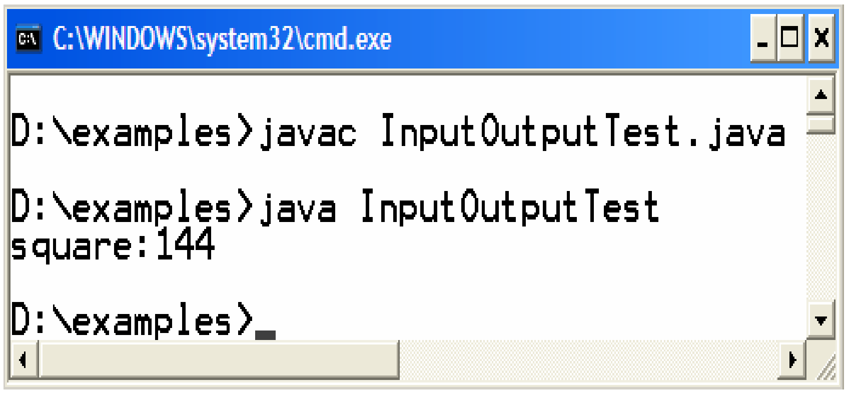
On line 5, showInputDialog method is called of JOptionPane class by passing string argument that will be displayed on GUI (dialog box). This method always returns back a String regardless of whatever you entered (int, float, double, char) in the input filed.

Our task is to print square of a number on console, so we first convert a string into a number by calling parseInt method of Integer wrapper class. This is what we done on line number 6.

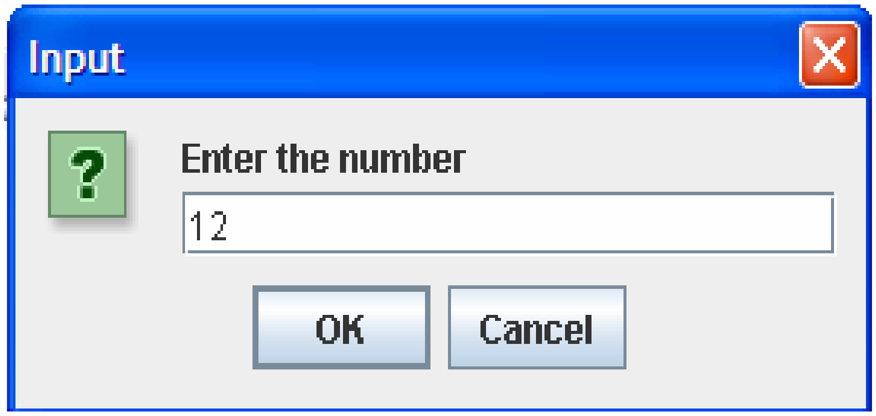
Line 11 will display square on GUI (dialog box) by using showMessageDialog method of JOptionPane class. The first argument passed to this method is null and the second argument must be a String. Here we use string concatenation.

Line 12 is needed to return the control back to command prompt whenever we use JoptionPane class.

Compile & Execute

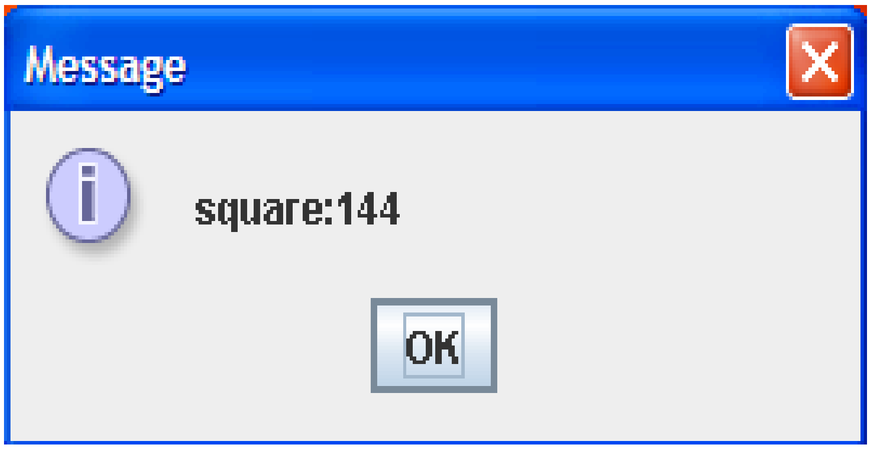


```
D:\examples>javac InputOutputTest.java

D:\examples>java InputOutputTest
square:144

D:\examples>_
```



Enter the number

12



square:144

Selection & Control Structure

The if-else and switch selection structures are exactly similar to we have in C++. All relational operators that we use in C++ to perform comparisons are also available in java with same behavior. Likewise for, while and do-while control structures are alike to C++.

Reference:

1- Java tutorial: http://www.dickbaldwin.com/java

2- Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 4

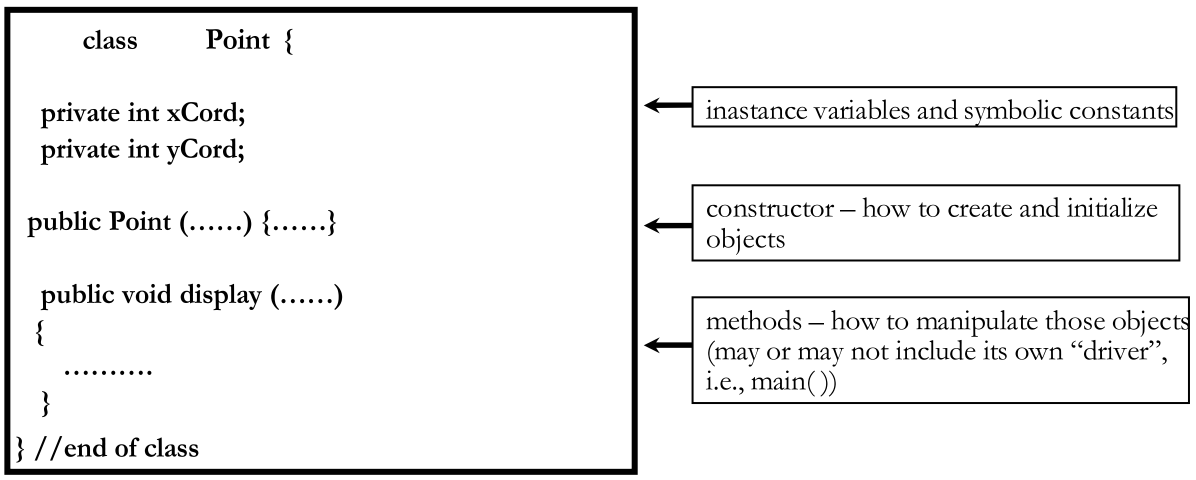
Object Oriented Programming

Java is fundamentally object oriented. Every line of code you write in java must be inside a class (not counting import directives). OOP fundamental stones Encapsulation, Inheritance and Polymorphism etc. are all fully supported by java.

<u>OOP Vocabulary Review</u>

- **Classes**
  - Definition or a blueprint of a user-defined datatype
  - Prototypes for objects
  - Think of it as a map of the building on a paper
- **Objects**
  - Nouns, things in the world
  - Anything we can put a thumb on
  - Objects are instantiated or created from class
- **Constructor**
  - A special method that is implicitly invoked. Used to create an Object (that is, an Instance of the Class) and to initialize it.
- **Attributes**
  - Properties an object has.
- **Methods**
  - Actions that an object can do

Defining a Class



```
      class        Point  {

   private int xCord;
   private int yCord;

 public Point (……) {……}

   public void display (……)
  {
     ……….
   }
} //end of class
```

inastance variables and symbolic constants

constructor – how to create and initialize objects

methods – how to manipulate those objects (may or may not include its own "driver", i.e., main( ))

Comparison with C++

Some important points to consider when defining a class in java as you probably noticed from the above given skeleton are

- There are no global variables or functions. Everything resides inside a class. Remember we wrote our main method inside a class. (For example, in HelloWorldApp program)

- Specify access modifiers (public, private or protected) for each member method or data members at every line.

    - *public:* accessible anywhere by anyone
    - *private:* Only accessible within this class
    - *protect:* accessible only to the class itself and to it's subclasses or other classes in the same package.
    - *default:* default access if no access modifier is provided. Accessible to all classes in the same package.

- There is no semicolon (;) at the end of class.

- All methods (functions) are written inline. There are no separate header and implementation files.

- Automatic initialization of class level data members if you do not initialize them

    Primitives

    - Numeric (int, float etc) with zero

- o Char with null
- o Boolean with false

Object References
- – With null

Note: Remember, the same rule is not applied to local variables (defined inside method body). Using a local variable without initialization is a compile time error

```
Public void someMethod( ) {
  int x; //local variable
  System.out.println(x); // compile time error
}
```

- – Constructor
  - – Same name as class name
  - – Does not have a return type
  - – No initialization list
  - – JVM provides a *zero argument* (default) constructor only if a class doesn't define it's own constructor

- – Destructors
  - – Are not required in java class because memory management is the responsibility of JVM.

<u>Task – Defining a Student class</u>

The following example will illustrate how to write a class. We want to write a "Student" class that

- should be able to store the following characteristics of student
  - Roll No
  - Name
- Provide default, parameterized and copy constructors
  - Provide standard getters/setters (discuss shortly) for instance variables
    - Make sure, roll no has never assigned a negative value i.e. ensuring the correct state of the object
    - Provide print method capable of printing student object on console

Getters / Setters

The attributes of a class are generally taken as private or protected. So to access them outside of a class, a convention is followed knows as getters & setters. These are generally public methods. The words *set* and *get* are used prior to the name of an attribute. Another important purpose for writing getter & setters to control the values assigned to an attribute.

<u>Student Class Code</u>

```
// File Student.java

public class Student {

    private String name;
    private int rollNo;
      // Standard Setters
    public void setName (String name) {
         this.name = name;
     }

     // Note the masking of class level variable rollNo
     public void setRollNo (int rollNo) {
         if (rollNo > 0) {
             this.rollNo = rollNo;
         }else {
             this.rollNo = 100;
         }
     }
   // Standard Getters
    public String getName ( ) {
         return name;
     }
     public int getRollNo ( ) {
         return rollNo;
     }

     // Default Constructor public Student() {
         name = “not set”;
         rollNo = 100;
     }

     // parameterized Constructor for a new student
     public Student(String name, int rollNo) {
         setName(name);      //call to setter of name
         setRollNo(rollNo);  //call to setter of rollNo
     }

     // Copy Constructor for a new student
     public Student(Student s) {
        name = s.name;
      rollNo = s.rollNo;
     }

         // method used to display method on console

     public void print () {
      System.out.print("Student name: " +name);
      System.out.println(", roll no: " +rollNo);
     }
} // end of class
```

Using a Class

Objects of a class are always created on heap using the "new" operator followed by constructor

- Student s = new Student (); // no pointer operator "*" between Student and s
- Only String constant is an exception

  String greet = "Hello" ; // No new operator
- However you can also use

  String greet2 = new String("Hello");

Members of a class (member variables and methods also known as instance variables/methods) are accessed using "." operator. There is no "Æ" operator in java

s.setName("Ali");

sÆsetName("Ali") is incorrect and will not compile in java

Note: Objects are always passed by reference and primitives are always passed by value in java.

<u>Task - Using Student Class</u>

Create objects of student class by calling default, parameterize and copy constructor
Call student class various methods on these objects

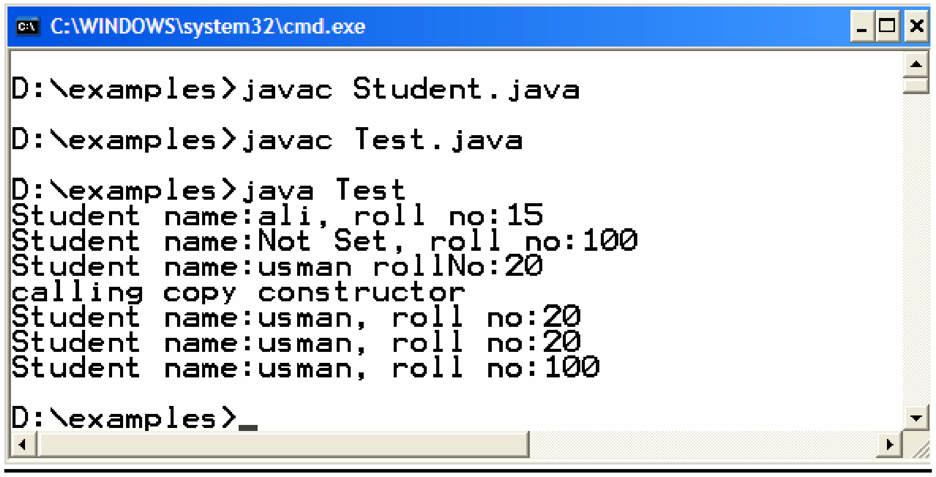
<u>Student client code</u>

```
// File Test.java

/* This class create Student class objects and demonstrates
  how to call various methods on objects
*/

public class Test{

  public static void main (String args[]){

   // Make two student obejcts
   Student s1 = new Student("ali", 15);
   Student s2 = new Student(); //call to default costructor

   s1.print(); // display ali and 15
   s2.print(); // display not set and 100


   s2.setName("usman");
   s2.setRollNo(20);

   System.out.print("Student name:" + s2.getName());
   System.out.println(" rollNo:" + s2.getRollNo());
```

```
        System.out.println("calling copy constructor");
          Student s3 = new Student(s2); //call to copy constructor

          s2.print();
          s3.print();

          s3.setRollNo(-10); //Roll No of s3 would be set to 100

          s3.print();

          /*NOTE: public vs. private
            A statement like "b.rollNo = 10;" will not compile in a
            client of the Student class when rollNo is declared
            protected or private
          */

        } //end of main
      } //end of class
```

Compile & Execute

Compile both classes using **javac** commad. Run Test class using **java** command.



C:\WINDOWS\system32\cmd.exe

```
D:\examples>javac Student.java

D:\examples>javac Test.java

D:\examples>java Test
Student name:ali, roll no:15
Student name:Not Set, roll no:100
Student name:usman rollNo:20
calling copy constructor
Student name:usman, roll no:20
Student name:usman, roll no:20
Student name:usman, roll no:100

D:\examples>_
```

More on Classes

Static

A class can have static variables and methods. Static variables and methods are associated with the class itself and are not tied to any particular object. Therefore statics can be accessed without instantiating an object. Static methods and variables are generally accessed by class name.

The most important aspect of statics is that they occur as a single copy in the class regardless of the number of objects. Statics are shared by all objects of a class. Non static methods and instance variables are not accessible inside a static method because no this reference is available inside a static method.

We have already used some static variables and methods. Examples are

System.**out**.println("some text"); -- out is a static variable
JOptionPane.**showMessageDialog**(null, "some text"); -- showMessageDialog is a static method

Garbage Collection & Finalize

Java performs garbage collection and eliminates the need to free objects explicitly. When an object has no references to it anywhere except in other objects that are also unreferenced, its space can be reclaimed.

Before an object is destroyed, it might be necessary for the object to perform some action. For example: to close an opened file. In such a case, define a *finalize()* method with the actions to be performed before the object is destroyed.

finalize

When a finalize method is defined in a class, Java run time calls *finalize()* whenever it is about to recycle an object of that class. It is noteworthy that a garbage collector reclaims objects in any order or never reclaims them. We cannot predict and assure when garbage collector will get back the memory of unreferenced objects.

The garbage collector can be *requested* to run by calling *System.gc()* method. It is not necessary that it accepts the request and run.

Example Code: using static & finalize ()

We want to count exact number of objects in memory of a Student class the one defined earlier. For this purpose, we'll modify Student class.

Add a static variable *countStudents* that helps in maintaining the count of student objects.

Write a getter for this static variable. (Remember, the getter also must be static one. Hoping so, you know the grounds).

In all constructors, write a code that will increment the countStudents by one.

Override *finalize()* method and decrement the countStudents variable by one.

Override *toString()* method.

Class *Object* is a superclass (base or parent) class of all the classes in java by default. This class has already *finalize()* and *toString()* method (used to convert an object state into string). Therefore we are actually overriding these methods over here. (We'll talk more about these in the handout on inheritance).

By making all above modifications, student class will look like

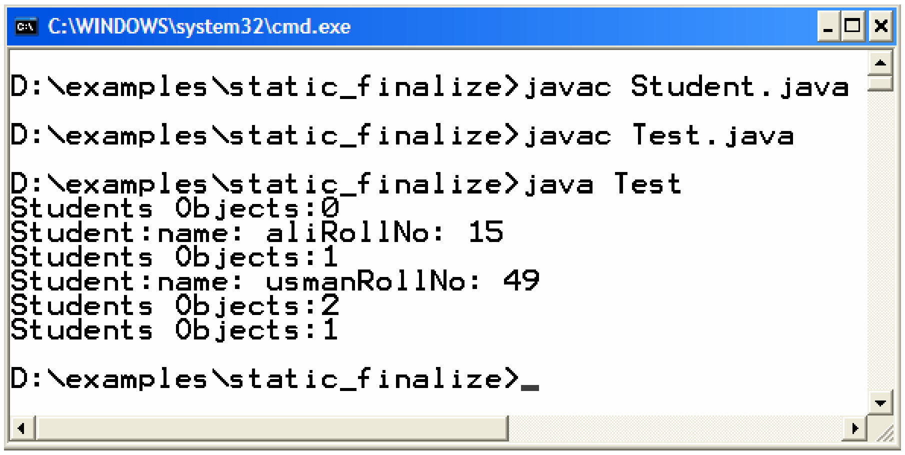
```
// File Student.java
public class Student {

 private String name;
    private int rollNo;
     private static int countStudents = 0;

       // Standard Setters
    public void setName (String name) {
          this.name = name;
     }

     // Note the masking of class level variable rollNo
     public void setRollNo (int rollNo) {
          if (rollNo > 0) {
              this.rollNo = rollNo;
          }else {
              this.rollNo = 100;
          }
     }
    // Standard Getters
    public String getName ( ) {
          return name;
    }
    public int getRollNo ( ) {
          return rollNo;
    }
    // gettter of static countStudents variable
    public static int getCountStudents(){
        return countStudents;
    }

    // Default Constructor public Student() {
        name = “not set”;
        rollNo = 100;
        countStudents += 1;
    }

    // parameterized Constructor for a new student
    public Student(String name, int rollNo) {
        setName(name);      //call to setter of name
        setRollNo(rollNo);  //call to setter of rollNo
        countStudents += 1;
    }

    // Copy Constructor for a new student
    public Student(Student s) {
       name = s.name;
       rollNo = s.rollNo;
       countStudents += 1;
    }

        // method used to display method on console
    public void print () {
       System.out.print("Student name: " +name);
       System.out.println(", roll no: " +rollNo);
    }

    // overriding toString method of java.lang.Object class
    public String toString(){
        return “name: ” + name + “ RollNo: ” + rollNo;
    }

    // overriding finalize method of Object class
    public void finalize(){
        countStudents -= 1;
    }

} // end of class
```

Next, we’ll write driver class. After creating two objects of student class, we deliberately loose object’s reference and requests the JVM to run garbage collector to reclaim the memory. By printing countStudents value, we can confirm that. Coming up code is of the Test class.

// File Test.java

```
public class Test{

  public static void main (String args[]){

   int numObjects;

   // printing current number of objects i.e 0
   numObjs = Student.getCountStudents();
   System.out.println("Students Objects" + numObjects);

   // Creating first student object & printing its values
   Student s1 = new Student("ali", 15);
   System.out.println("Student: " + s1.toString());

   // printing current number of objects i.e. 1
   numObjs = Student.getCountStudents();
   System.out.println("Students Objects" + numObjects);

   // Creating second student object & printing its values
   Student s2 = new Student("usman", 49);

   // implicit call to toString() method
   System.out.println("Student: " + s2);

   // printing current number of objects i.e. 2
   numObjs = Student.getCountStudents();
   System.out.println("Students Objects" + numObjects);

   // loosing object reference
   s1 = null

   // requesting JVM to run Garbage collector but there is
   // no guarantee that it will run
   System.gc();

   // printing current number of objects i.e. unpredictable
   numObjs = Student.getCountStudents();
   System.out.println("Students Objects" + numObjects);

  } //end of main
} //end of class
```

The compilation and execution of the above program is given below. Note that output may be different one given here because it all depends whether garbage collector reclaims the memory or not. Luckily, in my case it does.



```
D:\examples\static_finalize>javac Student.java

D:\examples\static_finalize>javac Test.java

D:\examples\static_finalize>java Test
Students Objects:0
Student:name: aliRollNo: 15
Students Objects:1
Student:name: usmanRollNo: 49
Students Objects:2
Students Objects:1

D:\examples\static_finalize>_
```

<u>Reference:</u>

Sun java tutorial: http://java.sun.com/docs/books/tutorial/java
Thinking in java by Bruce Eckle
Beginning Java2 by Ivor Hortan

Example code, their explanations and corresponding execution figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 5

Inheritance

In general, inheritance is used to implement a "is-a" relationship. Inheritance saves code rewriting for a client thus promotes reusability.

In java parent or base class is referred as *super* class while child or derived class is known as *sub* class.

Comparison with C++

- Java only supports single inheritance. As a result a class can only inherit from one class at one time.
- Keyword *extends* is used instead of "**:**" for inheritance.
- All functions are virtual by default
- All java classes inherit from Object class (more on it later).
- To explicitly call the super class constructor, use *super* keyword. It's important to remember that call to super class constructor must be first line.
- Keyword super is also used to call overridden methods.

Example Code: using inheritance

We'll use three classes to get familiar you with inheritance. First one is Employee class. This will act as super class. Teacher class will inherit from Employee class and Test class is driver class that contains main method. Let's look at them one by one

```
class Employee{

  protected int id;
  protected String name;

  //parameterized constructor
  public Employee(int id, String name){
    this.id = id;
    this.name = name;
  }
  //default constructor
  public Employee(){

     // calling parameterized constructor of same (Employee)
     // class by using keyword this

     this (10, “not set”);
  }
  //setters
  public void setId (int id) {
    this.id = id;
  }
  public void setName (String name) {
    this.name = name;
  }
  //getters
  public int getId () {
    return id;
  }

  public String getName () {
    return name;
  }
  // displaying employee object on console
  public void display(){
    System.out.println(“in employee display method”);

    System.out.println("Employee id:" + id + " name:" + name);
  }

  //overriding object's class toString method
  public String toString() {
    System.out.println(“in employee toString method”);
    return "id:" + id + "name:" + name;
 }

}//end class
```

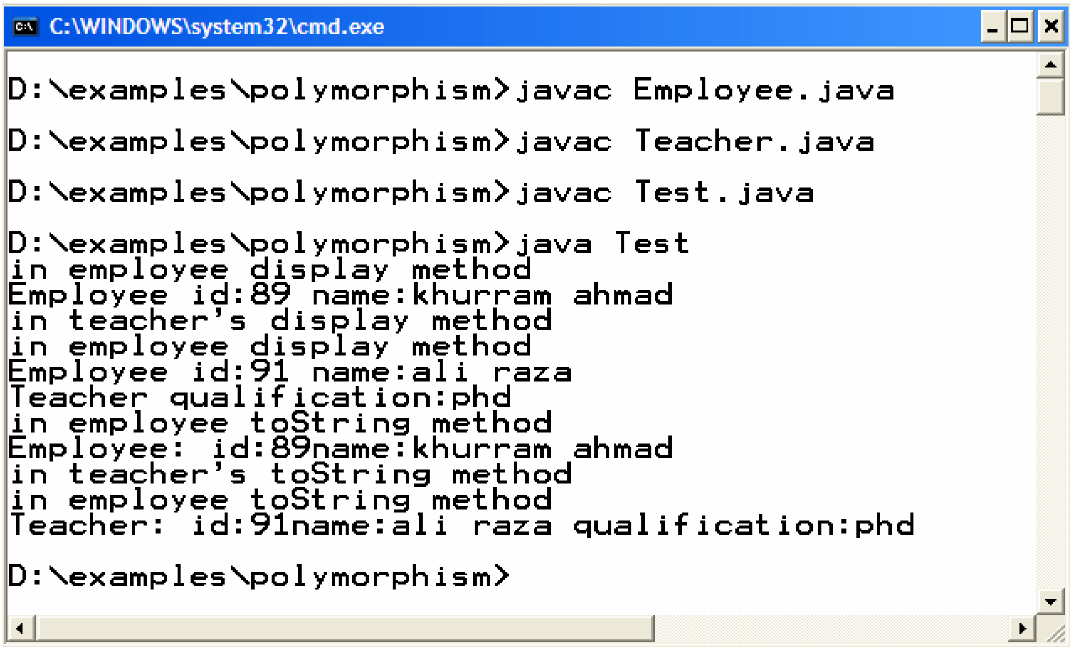
The Teacher class extends from Employee class. Therefore Teacher class is a subclass of Employee. The teacher class has an additional attribute i.e. qualification.

```
class Teacher extends Employee{

 private String qual;

 //default constructor
 public Teacher () {
   //implicit call to superclass default construct
   qual = "";
 }

 //parameterized constructor
 public Teacher(int i, String n, String q){

   //call to superclass param const must be first line
        super(i,n);
   qual = q;
 }

 //setter
 public void setQual (String qual){
   this.qual = qual;
 }
 //getter
 public String getQual(){
   return qual;
 }

 //overriding display method of Employee class
 public void display(){

   System.out.println("in teacher's display method");

        super.display(); //call to superclass display method

   System.out.println("Teacher qualification:" + qual);
 }

 //overriding toString method of Employee class
 public String toString() {

   System.out.println("in teacher's toString method");

   String emp = super.toString();

   return emp +" qualification:" + qual;
 }
}//end class
```

Objects of Employee & Teacher class are created inside main method in Test class. Later calls are made to display and toString method using these objects.

```
class Test{

  public static void main (String args[]){

     System.out.println("making object of employee");
     Employee e = new Employee(89, "khurram ahmad");

     System.out.println("making object of teacher");
     Teacher t = new Teacher (91, "ali raza", "phd");

     e.display(); //call to Employee class display method
     t.display(); //call to Teacher class display method

     // calling employee class toString method explicitly
     System.out.println("Employee: " +e.toString());

     // calling teacher class toString implicitly
          System.out.println("Teacher: " + t);

  } //end of main

}//end class
```

Output

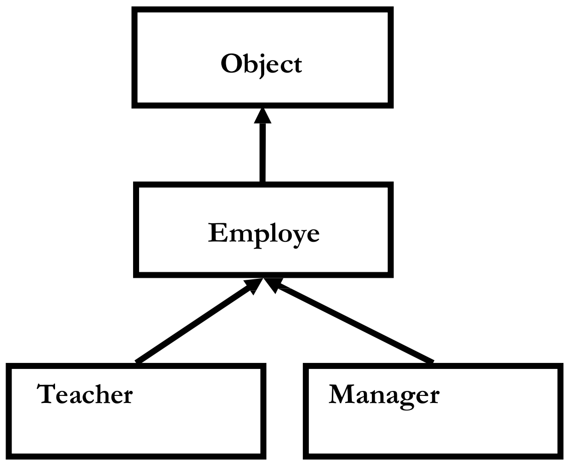


```
D:\examples\polymorphism>javac Employee.java

D:\examples\polymorphism>javac Teacher.java

D:\examples\polymorphism>javac Test.java

D:\examples\polymorphism>java Test
in employee display method
Employee id:89 name:khurram ahmad
in teacher's display method
in employee display method
Employee id:91 name:ali raza
Teacher qualification:phd
in employee toString method
Employee: id:89name:khurram ahmad
in teacher's toString method
in employee toString method
Teacher: id:91name:ali raza qualification:phd

D:\examples\polymorphism>
```

Object – The Root Class

The Od Java classes. For user defined classes, its not necessary to mention the Object class as a super class, java doesbject class in Java is a superclass for all other classes defined in Java's class libraries, as well as for user-define it automatically for you.

The class Hierarchy of Employee class is shown below. Object is the super class of Employee class and Teacher is a subclass of Employee class. We can make another class Manager that can also extends from Employee class.



Polymorphism

"Polymorphic" literally means "of multiple shapes" and in the context of OOP, polymorphic means "having multiple behavior".

A parent class reference can point to the subclass objects because of is-a relationship. For example a Employee reference can point to:

- o Employee Object
- o Teacher Object
- o Manager Object

A polymorphic method results in different actions depending on the object being referenced

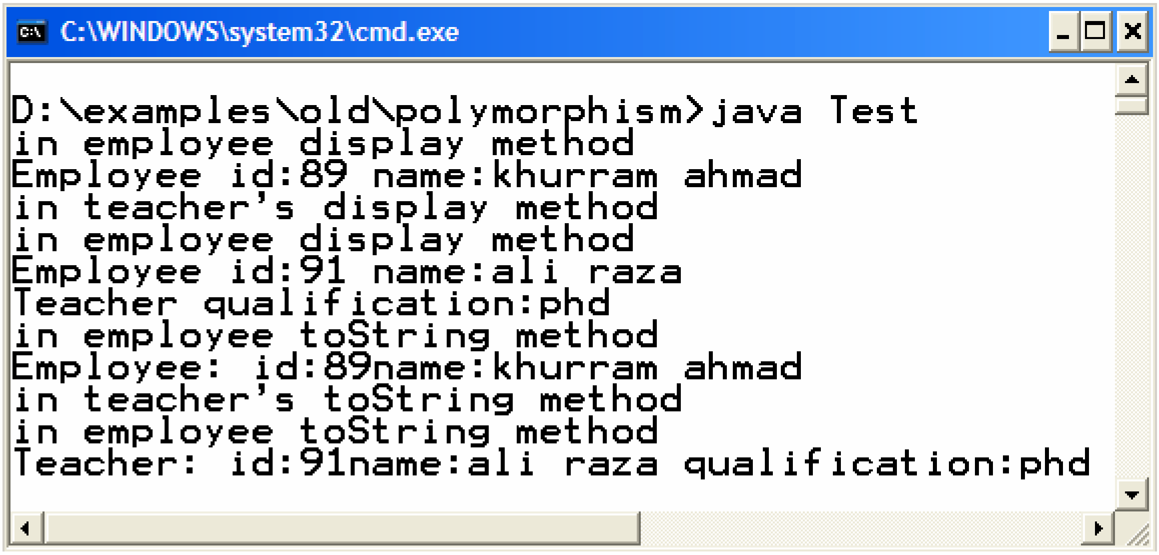
- o Also known as *late binding* or *run-time binding*

Example Code: using polymorphism

This Test class is the modification of last example code. Same Employee & Teacher classes are used. Objects of Employee & Teacher class are created inside main methods and calls are made to display and toString method using these objects.

```
class Test{
  public static void main (String args[]){
    // Make employee references
    Employee ref1, ref2;

    // assign employee object to first employee reference
    ref1 = new Employee(89, "khurram ahmad");

    // is-a relationship, polymorphism
    ref2 = new Teacher (91, "ali raza", "phd");

    //call to Employee class display method
    ref1.display();

    //call to Teacher class display method
    ref2.display();

    // call to Employee class toString method
    System.out.println("Employee: " +ref1.toString());
    // call to Teacher class toString method
    System.out.println("Teacher: " + ref2.toString());

  } //end of main

}//end class
```

Output



```
D:\examples\old\polymorphism>java Test
in employee display method
Employee id:89 name:khurram ahmad
in teacher's display method
in employee display method
Employee id:91 name:ali raza
Teacher qualification:phd
in employee toString method
Employee: id:89name:khurram ahmad
in teacher's toString method
in employee toString method
Teacher: id:91name:ali raza qualification:phd
```

Type Casting

In computer science, **type conversion** or **typecasting** refers to changing an entity of one datatype into another. Type casting can be categorized into two types

1. **Up-casting**

   Converting a smaller data type into bigger one

   Implicit – we don't have to do something special

   No loss of information

   Examples of

   — **Primitives**

   ```
   int a = 10;
   double  b = a;
   ```

   — **Classes**

   ```
   Employee e = new Teacher( );
   ```

2. **Down-casting**

   Converting a bigger data type into smaller one

   Explicit – need to mention

   Possible loss of information

   Examples of

   — **Primitives**

   ```
   double a = 7.65;
   int  b = (int) a;
   ```

   — **Classes**

   ```
   Employee e = new Teacher( );  // up-casting
   Teacher t  = (Teacher) e;    // down-casting
   ```

References:

Java tutorial: http://java.sun.com/docs/books/tutorial/java/javaOO/
Stanford University

Example code, their explanations and corresponding figures for handout 5-1,5-2 are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 6

Collections

A collection represents group of objects know as its elements. Java has a built-in support for collections. Collection classes are similar to STL in C++. An advantage of a collection over an array is that you don't need to know the eventual size of the collection in order to add objects to it. The java.util package provides a set of collection classes that helps a programmer in number of ways.

Collections Design

All classes almost provides same methods like get(), size(), isEmpty() etc. These methods will return the object stored in it, number of objects stored and whether collection contains an object or not respectively.

Java collections are capable of storing any kind of objects. Collections store references to objects. This is similar to using a void* in C. therefore down casting is required to get the actual type. For example, if string in stored in a collection then to get it back, we write

```
String element = (String)arraylist.get(i);
```

Collection messages

Some basic messages (methods) are:

Constructor
- creates a collection with no elements

int size()
- returns the number of elements in a collection

boolean add(Object)
- adds a new element in the collection
- returns true if the element is added successfully false otherwise

boolean isEmpty()
- returns true if this collection contains no element false otherwise

boolean contains(Object)
- returns true if this collection contains the specified element by using iterative search

boolean remove(Object)
- removes a single instance of the specified element from this collection, if it is present

ArrayList

It's like a resizable array. ArrayList actually comes as a replacement the old "Vector" collection. As we add or remove elements into or from it, it grows or shrinks over time.

Useful Methods

add (Object)
- With the help of this method, any object can be added into ArrayList because Object is the super class of all classes.
- Objects going to add will implicitly up-cast.

Object get(int index)
- Returns the element at the specified position in the list
- index ranges from 0 to size()-1
- must cast to appropriate type

remove (int index)
- Removes the element at the specified position in this list.
- Shifts any subsequent elements to the left (subtracts one from their indices).

int size( )

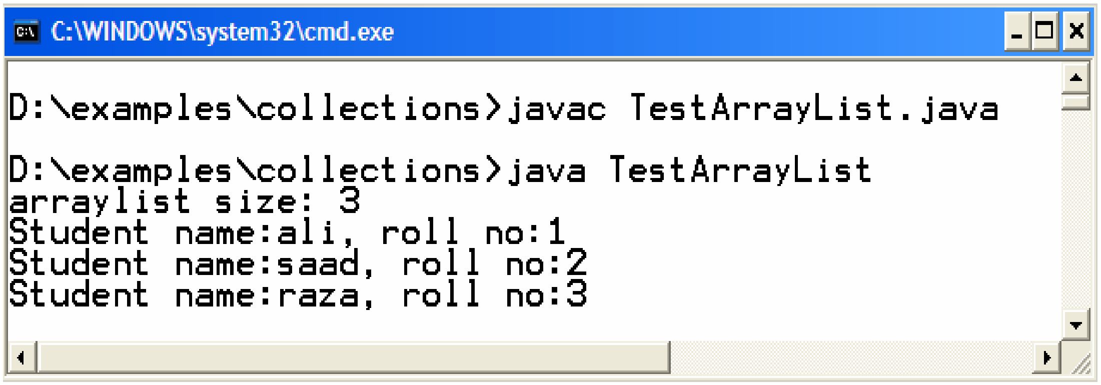
Example Code: Using ArrayList class

We'll store Student objects in the ArrayList. We are using the same student class which we built in previous lectures/handouts.

We'll add three student objects and later prints all the student objects after retrieving them from ArrayList. Let's look at the code

```
iport java.util.*;

public class ArrayListTest {

  public static void main(String[] args) {

        // creating arrayList object by calling constructor
        ArrayList al= new ArrayList();


   // creating three Student objects
        Student s1 = new Student ("ali" , 1);
        Student s2 = new Student ("saad" , 2);
        Student s3 = new Student ("raza" , 3);

  // adding elements (Student objects) into arralylist al.add(s1);
        al.add(s2);
        al.add(s3);
        // checking whether arraylist is empty or not boolean  b = al.isEmpty ();

    if (b = = true) {
            System.out.println("arraylist is empty");

    } else {
            int size = al.size();
            System.out.println("arraylist size: " + size);
    }

        // using loop to iterate. Loops starts from 0 to one
    // less than size
        for (int i=0; i<al.size(); i++ ){

      // retrieving object from arraylist
            Student s = (Student) al.get(i);

      // calling student class print method
            s.print();

        } // end for loop

   } // end main
  } // end class
```

Output



```
D:\examples\collections>javac TestArrayList.java

D:\examples\collections>java TestArrayList
arraylist size: 3
Student name:ali, roll no:1
Student name:saad, roll no:2
Student name:raza, roll no:3
```

HashMap

Store elements in the form of *key- value* pair form. A key is associated with each object that is stored. This allows fast retrieval of that object. Keys are unique.

Useful Methods

put(Object key, Object Value)
— Keys & Values are stored in the form of objects (implicit upcasting is performed).
— Associates the specified value with the specified key in this map.

— If the map previously contained a mapping for this key, the old value is replaced.

Object get(Object key)

— Returns the value to which the specified key is mapped in this identity hash map, or null if the map contains no mapping for this key.

— Must downcast to appropriate type when used

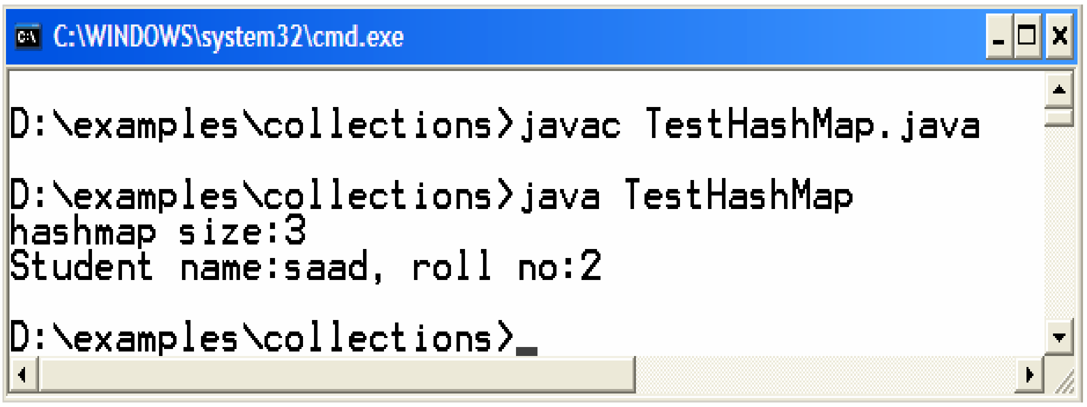
int size( )

Example Code: using HashMap class

In this example code, we'll store Student objects as values and their rollnos in the form of strings as keys. Same Student class is used. The code is;

```
iport java.util.*;
public class HashMapTest {
  public static void main(String[] args) {
         // creating HashMap object
         HashMap h= new HashMap();

    // creating Student objects
         Student s1 = new Student (“ali” , 1); Student s2 = new Student (“saad” ,
         2); Student s3 = new Student (“raza” , 6);

        // adding elements (Student objects) where roll nos
    // are stored as keys and student objects as values
         h.put(“one” , s1);
         h.put(“two” , s2);
         h.put(“six”,  s3);
         // checking whether hashmap is empty or not boolean  b = h.isEmpty ();

    if (b == true) {
           System.out.println(“hashmap  is  empty”);
    } else {
           int size = h.size();
           System.out.println(“hashmap  size: ” + size);
    }

     // retrieving student object against rollno two and
    // performing downcasting
         Student s = (Student)h.get(“two”);

  // calling student's class print method s.print();
  } // end main
} // end class
```

Output



```
D:\examples\collections>javac TestHashMap.java

D:\examples\collections>java TestHashMap
hashmap size:3
Student name:saad, roll no:2

D:\examples\collections>_
```

<u>**References:**</u>

J2SE 5.0 new features: http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html
Technical Article: http://java.sun.com/developer/technicalArticles/releases/j2se15/
Beginning Java2 by Ivor Horton
Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Address Book

Warning: It is strongly advised that you type the code given in this example yourself. Do not **copy/paste** it; most probably you will get unexpected errors that you have never seen. Some bugs are deliberately introduced as well to avoid copy- pasting. TAs will not cooperate with you in debugging such errors☺.

Problem

We want to build an address book that is capable of storing name, address & phone number of a person.

Address book provides functionality in the form of a JOptionPane based menu. The feature list includes

- Add – to add a new person record
- Delete – to delete an existing person record by name
- Search – to search a person record by name
- Exit – to exit from application

The Address book should also support persistence for person records

Approach for Solving Problem

Building a small address book generally involves 3 steps. Let us briefly discuss each step and write a solution code for each step

Step1 – Make PersonInfo class

First of all you need to store your desired information for each person. For this you can create a user-defined data type (i.e. a class). Make a class PersonInfo with name, address and phone number as its attributes.

Write a parameterized constructor for this class.

Write print method in Person class that displays one person record on a message dialog box.

The code for **PersonInfo** class is given below.

```
import javax.swing.*;

 class PersonInfo {

    String name;
    String address;
    String phoneNum;


    //parameterized constructor
    public PersonInfo(String n, String a, String p) {

      name = n;
      address = a;
      phoneNum = p;

    }

    //method for displaying person record on GUI
    public void print( ) {

      JOptionPane.showMessageDialog(null, "name: " + name +
             "address:" +address + "phone no:" + phoneNum);

    }
 }
```

Note: **N**ot declaring attributes as private is a bad approach but we have done it to keep things simple here.

Step2 – Make AddressBook class

Take the example of daily life; generally address book is used to store more than one person records and we don't know in advance how many records are going to be added into it.

So, we need some data structure that can help us in storing more than one PersonInfo objects without concerning about its size.

ArrayList can be used to achieve the above functionality

Create a class Address Book with an ArrayList as its attribute. This arraylist will be used to store the information of different persons in the form of PersonInfo Objects. This class will also provide *addPerson, deletePerson* & *searchPerson* methods. These methods are used for adding new person records, deleting an existing person record by name and searching among existing person records by name respectively.

Input/Output will be performed through JOptionPane.

The code for **AddressBook** class is

```
import javax.swing.*;
import java.util.*;


 class AddressBook {

   ArrayList persons;

   //constructor
   public AddressBook ( ) {

     persons = new ArrayList();

   }

  //add new person record to arraylist after taking input
  public void addPerson( ) {

   String name =JOptionPane.showInputDialog(“Enter name”);
   String add  = JOptionPane.showInputDialog(“Enter address”);
   String pNum = JOptionPane.showInputDialog(“Enter phone no”);

   //construt new person object
   PersonInfo p = new PersonInfo(name, add, pNum);

   //add the above PersonInfo object to arraylist
   persons.add(p);

  }

  //search person record by name by iterating over arraylist
  public void searchPerson (String n) {

   for (int i=0; i< persons.size(); i++) {

     PersonInfo p = (PersonInfo)persons.get(i);

     if ( n.equals(p.name) ) {
        p.print();
     }

   } // end for

  } // end searchPerson

  //delete person record by name by iterating over arraylist
  public void deletePerson (String n) {

   for (int i=0; i< persons.size(); i++) {

     PersonInfo p = (PersonInfo)persons.get(i);

     if ( n.equals(p.name) ) {
        persons.remove(i);
     }
```

```
    }
   }
} // end class
```

The *addperson* method first takes input for name, address and phone number and than construct a PersonInfo object by using the recently taken input values. Then the newly constructed object is added to the arraylist – *persons*.

The *searchPerson* & *deletePerson* methods are using the same methodology i.e. first they search the required record by name and than prints his/her detail or delete the record permanently from the ArrayList.

Both the methods are taking string argument, by using this they can perform their search or delete operation. We used for loop for iterating the whole ArrayList. By using the size method of ArrayList, we can control our loop as ArrayList indexes range starts from 0 to one less than size.

Notice that, inside loop we retrieve each PersonInfo object by using down casting operation. After that we compare each PersonInfo object's name by the one passed to these methods using equal method since Strings are always being compared using equal method.

Inside if block of *searchPerson*, *print* method is called using PersonInfo object that will display person information on GUI. On the other hand, inside if block of *deletePerson* method, remove method of ArrayList class is called that is used to delete record from *persons* i.e. ArrayList.

Step3 – Make Test class (driver program)

This class will contain a main method and an object of AddressBook class.

Build GUI based menu by using switch selection structure

Call appropriate methods of AddressBook class

The code for **Test** class is

```
import javax.swing.*;
class Test {

    Public static void main  (String args[]) {

        AddressBook ab = new AddressBook();

        String input, s;
        int ch;

     while (true) {

        input = JOptionPane.showInputDialog("Enter 1 to add " +
                "\n Enter 2 to Search \n Enter 3 to Delete" +
                "\n Enter 4 to Exit");
ch = Integer.parseInt(input);


        switch (ch) {

          case 1:
                ab.addPerson();
                break;

          case 2:
                s = JOptionPane.showInputDialog(
                                                    "Enter name to search ");
                ab.searchPerson(s);
                break;

          case 3:
                s = JOptionPane.showInputDialog(
                                "Enter name to delete ");
                ab.deletePerson(s);
                break;
          case 4:
                System.exit(0);
        }
      }//end while
    }//end main
}
```

Note that we use *infinite* while loop that would never end or stop given that our program should only exit when user enters 4 i.e. exit option.

Compile & Execute

Compile all three classes and run Test class. Bravo, you successfully completed the all basic three steps. Enjoy! ☺.

Reference

Entire content for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose.

Lesson 7

Intro to Exceptions

Types of Errors

Generally, you can come across three types of errors while developing software. These are Syntax, Logic & Runtime errors.

1. **Syntax Errors**

   Arise because the rules of the language are not followed.

2. **Logic Errors**

   Indicates that logic used for coding doesn't produce expected output.

3. **Runtime Errors**

   Occur because the program tries to perform an operation that is impossible to complete.

   Cause exceptions and may be handled at runtime (while you are running the program)

   For example divide by zero

What is an Exception?

An exception is an event that usually signals an erroneous situation at run time

Exceptions are wrapped up as objects

A program can deal with an exception in one of three ways:

- o ignore it
- o handle it where it occurs
- o handle it an another place in the program

Why handle Exceptions?

Helps to separate error handling code from main logic (the normal code you write) of the program.

As different sort/type of exceptions can arise, by handling exceptions we can distinguish between them and write appropriate handling code for each type for example we can differently handle exceptions that occur due to division by Zero and exceptions that occur due to non-availability of a file.

If not handled properly, program might terminate.

Exceptions in Java

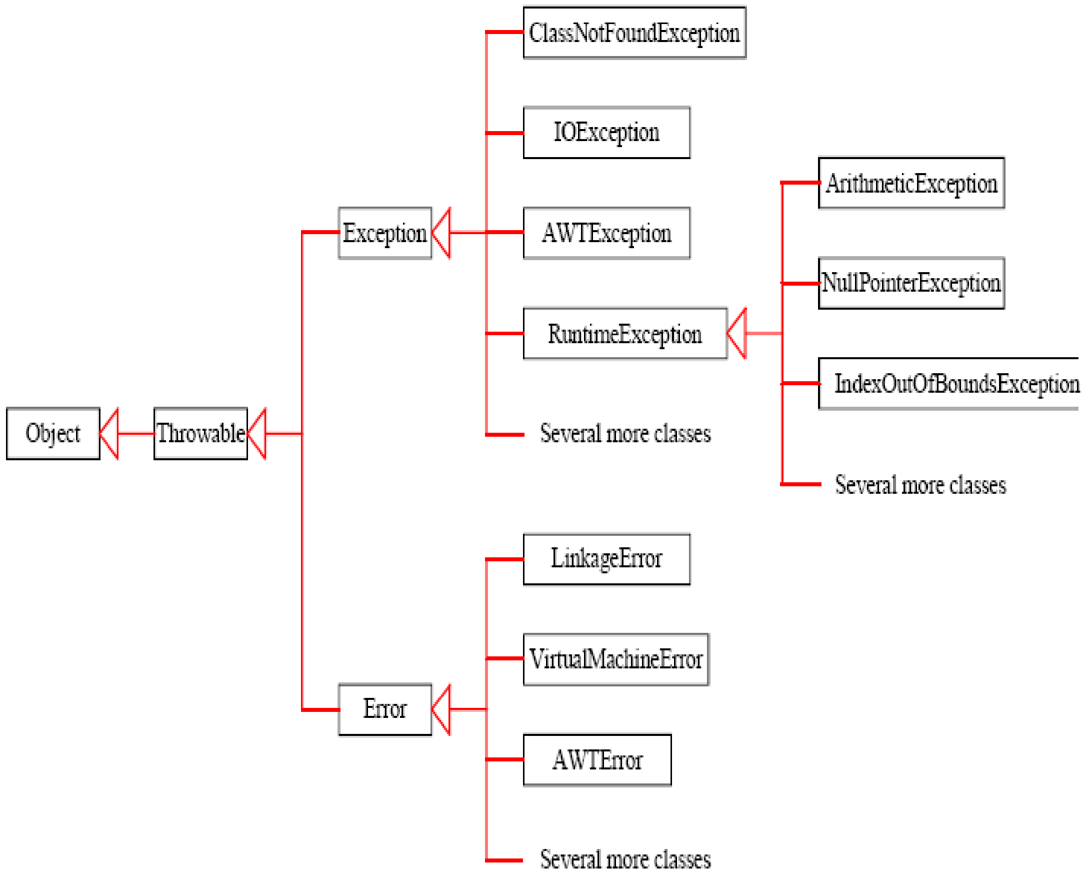
An exception in java is represented as an object that's created when an abnormal situation arises in the program. Note that an error is also represented as an object in Java, but usually represents an unrecoverable situation and should not be caught

The exception object stores information about the nature of the problem. For example, due to network problem or class not found etc.

All exceptions in java are inherited from a class know as *Throwable*.

Exception Hierarchy

Following diagram is an abridged version of Exception class hierarchy



Types of Exceptions

Exceptions can be broadly categorized into two types, Unc*hecked* & *Checked Exceptions*.

Unchecked Exceptions

- Subclasses of RuntimeException and Error.
- Does not require explicit handling
- Run-time errors are internal to your program, so you can get rid of them by debugging your code
- For example, null pointer exception; index out of bounds exception; division by zero exception; ...

Checked Exceptions

- Must be caught or declared in a throws clause
- Compile will issue an error if not handled appropriately
- Subclasses of Exception other than subclasses of RuntimeException.
- Other arrive from external factors, and cannot be solved by debugging
- Communication from an external resource – e.g. a file server or database

How Java handles Exceptions

Java handles exceptions via 5 keywords. **try**, **catch**, **finally**, **throw** & **throws.**

- **try block**
    - Write code inside this block which could generate errors
- **catch block**
    - Code inside this block is used for exception handling
    - When the exception is raised from try block, only than catch block would execute.
- **finally block**
    - This block always executes whether exception occurs or not.
    - Write clean up code here, like resources (connection with file or database) that are opened may need to be closed.

The basic structure of using try – catch – finally block is shown in the picture below:

```
try                                                          //try block
{
        // write code that could generate exceptions
} catch (<exception to be caught>)                           //catch block
{
                //write code for exception handling
}
 ……
 …….
catch (<exception to be caught>)                             //catch block
{
        //code for exception handling
} finally                                                    //finally block
{
        //any clean-up code, release the acquired resources
}
```

- **throw**
  - To manually throw an exception, keyword throw is used.

  Note: we are not covering throw clause in this handout

- **throws**
  - If method is not interested in handling the exception than it can throw back the exception to the caller method using throws keyword.
  - Any exception that is thrown out of a method must be specified as such by a **throws** clause.

References:

- Java tutorial by Sun: http://java.sun.com/docs/books/turorial
- Beginning Java2 by Ivor Hortan
- Thinking in Java by Bruce Eckle
- CS193j Stanford University

Code Examples of Exception Handling

Unchecked Exceptions

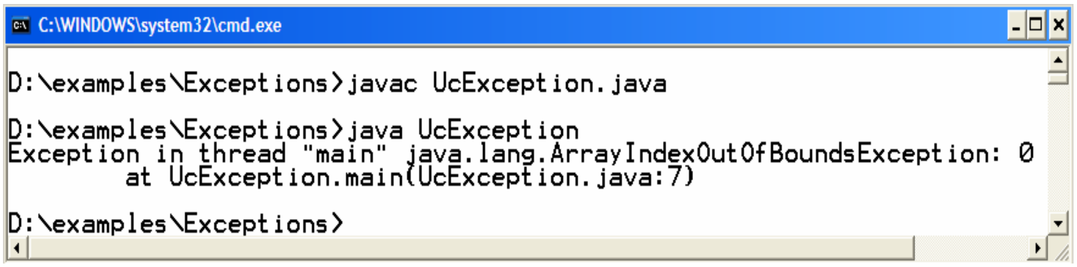
Example Code: UcException.java

The following program takes one command line argument and prints it on the console

```
// File UcException.java

public class UcException {
  public static void main (String args[ ]) {
      System.out.println(args[0]);
  }
}
```

Compile & Execute

If we compile & execute the above program without passing any command line argument, an ArrayIndexOutOfBoundsException would be thrown. This is shown in the following picture



```
D:\examples\Exceptions>javac UcException.java

D:\examples\Exceptions>java UcException
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at UcException.main(UcException.java:7)

D:\examples\Exceptions>
```

Why?

Since we have passed no argument, therefore the size of String args[ ] is zero, and we have tried to access the first element (first element has index zero) of this array.

From the output window, you can find out, which code line causes the exception to be raised. In the above example, it is

```
System.out.println(args[0]);
```

Modify UcException.java

Though it is not mandatory to handle unchecked exceptions we can still handle Unchecked Exceptions if we want to. These modifications are shown in bold.

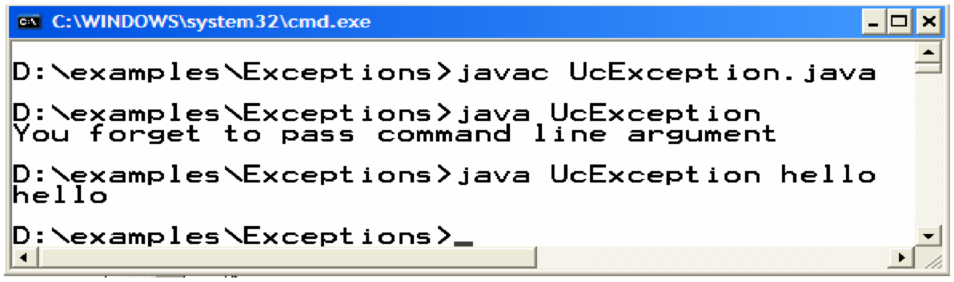
```
// File UcException.java

public class UcException {
  public static void main (String args[ ]) {
    try {
      System.out.println(args[0]);
    catch (IndexOutOfBoundsExceptoin ex) {
      System.out.println("You forget to pass command line argument");
        }
}
```

The possible exception that can be thrown is IndexOutOfBoundsException, so we handle it in the catch block.

When an exception occurs, such as IndexOutOfBoundsException in this case, then an object of type IndexOutOfBoundesException is created and it is passed to the corresponding catch block (i.e. the catch block which is capable of handling this exception). The catch block receives the exception object inside a variable which is *ex* in this case. It can be any name; it is similar to the parameter declared in the method signature. It receives the object of exception type (IndexOutOfBoundsExceptoin) it is declared.

Compile & Execute

If we execute the modified program by passing command line argument, the program would display on console the provided argument. After that if we execute this program again without passing command line argument, this time information message would be displayed which is written inside catch block.



```
D:\examples\Exceptions>javac UcException.java

D:\examples\Exceptions>java UcException
You forget to pass command line argument

D:\examples\Exceptions>java UcException hello
hello

D:\examples\Exceptions>_
```

Checked Exceptions

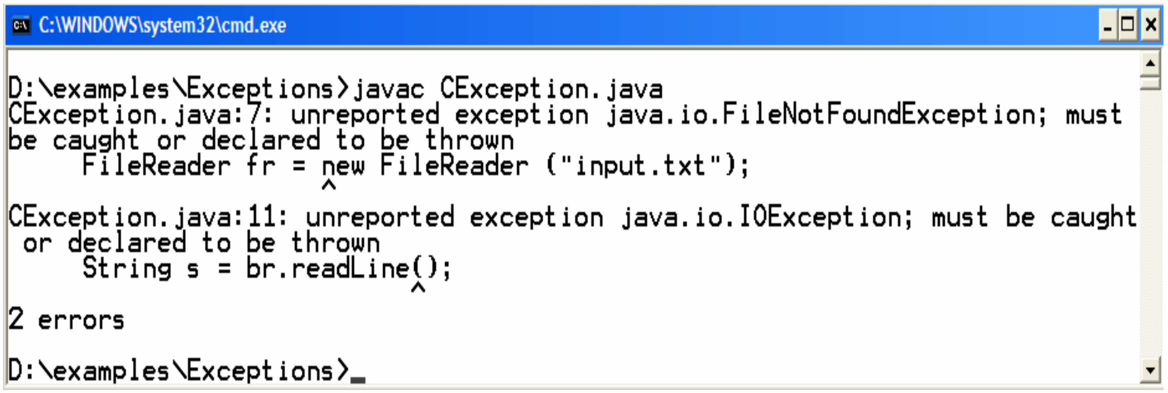
Example Code: CException.java

The following program reads a line *(hello world)* from a file and prints it on the console. The File reading code is probably new for you. We'll explain it in the coming handouts
(Streams). For now, assumed that the code written inside the main read one line from a file and prints that to console.

```
// File CException.java

import java.io.* ;

public class CException {
   public static void main (String args[ ]) {
        FileReader fr = new FileReader ("input.txt");
        BufferedReader br = new BufferedReader (fr);

        //read the line form file
        String line = br.readLine();
        System.out.println(line);
   }
}
```

Compile & Execute

If you try to compile this program, the program will not compile successfully and displays the message of unreported exception. This happens when there is code that can generate a checked exception but you have not handled that exception. Remember checked exceptions are detected by compiler. As we early discussed, without handling Checked exception, out program won't compile.


C:\WINDOWS\system32\cmd.exe

```
D:\examples\Exceptions>javac CException.java
CException.java:7: unreported exception java.io.FileNotFoundException; must
be caught or declared to be thrown
     FileReader fr = new FileReader ("input.txt");
                     ^
CException.java:11: unreported exception java.io.IOException; must be caught
 or declared to be thrown
     String s = br.readLine();
                           ^
2 errors

D:\examples\Exceptions>_
```

Modify CException.java

As we have discussed earlier, it is mandatory to handle checked exceptions. In order to compile the code above, we modify the above program so that file reading code is placed inside a try block. The expected exception (IOException) that can be raised is caught in catch block.

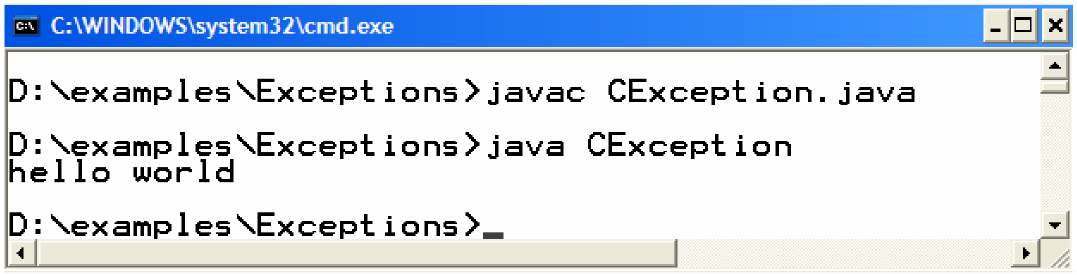
```
// File CException.java

import java.io.* ;

public class CException {
   public static void main (String args[ ]) {
    try{
       FileReader fr = new FileReader ("input.txt");
       BufferedReader br = new BufferedReader (fr);

       //read the line form file
       String line = br.readLine();
       System.out.println(line);
    catch( IOExceptoin ex) {
      System.out.println(ex);
    }
   }
}
```

The code line written inside the catch block will print the exception name on the console if exception occurs

Compile & Execute

After making changes to your program, it would compile successfully. On executing this program, *hello world* would be displayed on the console

Note: Before executing, make sure that a text file named *input.txt* must be placed in the same directory where the program is saved. Also write *hello world* in that file before saving it.



```
D:\examples\Exceptions>javac CException.java

D:\examples\Exceptions>java CException
hello world

D:\examples\Exceptions>_
```

The finally block

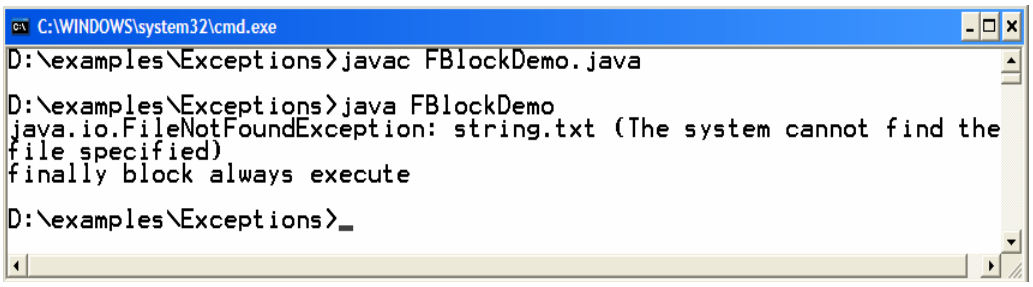
The finally block always executes regardless of exception is raised or not while as you remembered the catch block only executes when an exception is raised.

Exampel Code : FBlockDemo.java

```
// File FBlockDemo.java

import java.io.* ;

public class FBlockDemo {
  public static void main (String args[ ]) {
   try{
      FileReader fr = new FileReader ("strings.txt");
      BufferedReader br = new BufferedReader (fr);

      //read the line form file
      String line = br.readLine();
      System.out.println(line);
   catch( IOExceptoin ex) {
     System.out.println(ex);
   }
   finally {
     System.out.println("finally block always execute");
   }
  }
}
```

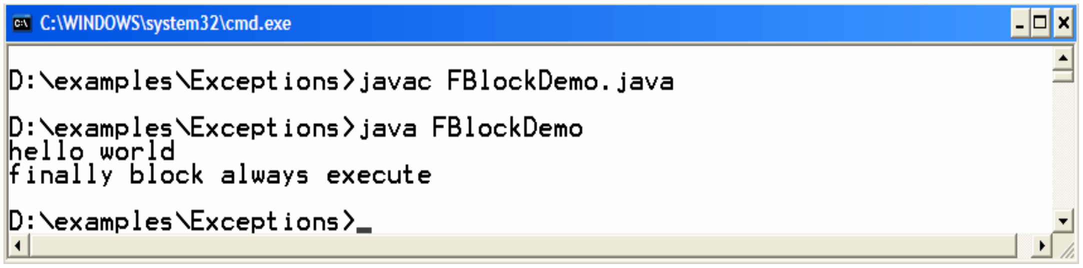
Compile & Execute

The program above, will read one line from *string.txt* file. If *string.tx* is not present in the same directory the FileNotFoundException would be raised and catch block would execute as well as the finally block.



```
D:\examples\Exceptions>javac FBlockDemo.java

D:\examples\Exceptions>java FBlockDemo
java.io.FileNotFoundException: string.txt (The system cannot find the
file specified)
finally block always execute

D:\examples\Exceptions>_
```

If *string.txt* is present there, no such exception would be raised but still finally block executes. This is shown in the following output diagram



```
D:\examples\Exceptions>javac FBlockDemo.java

D:\examples\Exceptions>java FBlockDemo
hello world
finally block always execute

D:\examples\Exceptions>_
```

Multiple catch blocks

- Possible to have multiple catch clauses for a single try statement
  - Essentially checking for different types of exceptions that may happen
- Evaluated in the order of the code
  - *Bear in mind the Exception hierarchy when writing multiple catch clauses!*
  - If you catch Exception first and then IOException, the IOException will never be caught!

Example code: MCatchDemo.java

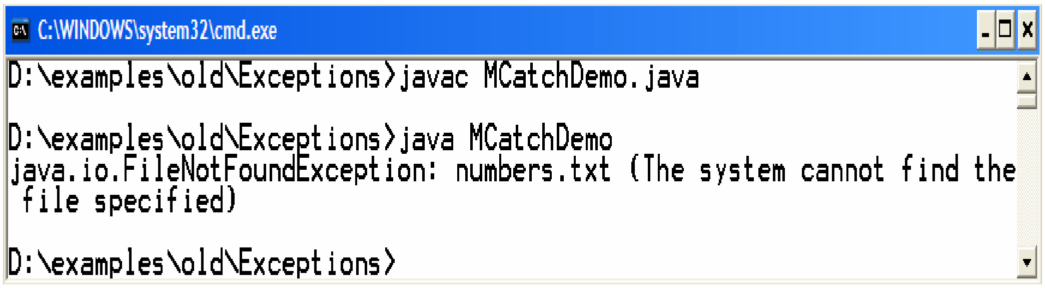
The following program would read a number form a file *numbers.txt* and than prints its square on the console

```
// File MCatchDemo.java

import java.io.* ;

public class MCatchDemo {
   public static void main (String args[ ]) {
    try{

  // can throw FileNotFound or IOException
      FileReader fr = new FileReader ("numbers.txt");
      BufferedReader br = new BufferedReader (fr);

      //read the number form file
      String s = br.readLine();

           //may throws NumberFormatException, if s is not a no.
      int number = Integer.parseInt(s);
      System.out.println(number * number);

    catch( NumberFormatExceptoin nfEx) {
      System.out.println(nfEx);
    }

   catch( FileNotFoundExceptoin fnfEx) {
        System.out.println(fnfEx);
      }
      catch( IOExceptoin ioEx) {
        System.out.println(ioEx);
      }
    }
  }
```

We read everything from a file (numbers, floating values or text) as a String. That's why we first convert it to number and than print its square on console.

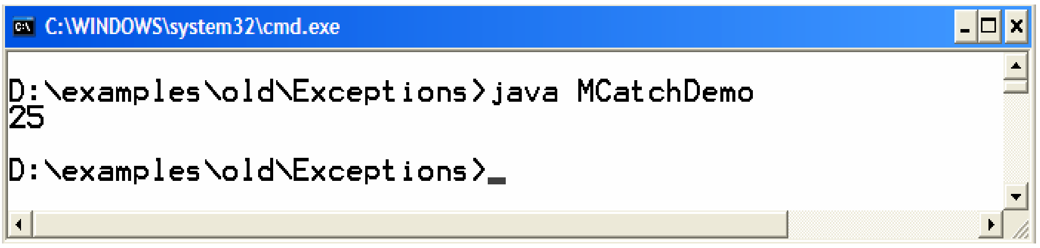
Compile & Execute

If file *numbers.txt* is not present in the same directory, the FileNotFoundException would be thrown during execution.



```
D:\examples\old\Exceptions>javac MCatchDemo.java

D:\examples\old\Exceptions>java MCatchDemo
java.io.FileNotFoundException: numbers.txt (The system cannot find the
 file specified)

D:\examples\old\Exceptions>
```

If *numbers.txt* present in the same directory and contains a number, than hopefully no exception would be thrown.



```
D:\examples\old\Exceptions>java MCatchDemo
25

D:\examples\old\Exceptions>_
```

The throws clause

The following code examples will introduce you with writing & using throws clause.

Example Code: ThrowsDemo.java
The ThrowsDemo.java contains two methods namely method1 & method2 and one main method. The main method will make call to method1 and than method1 will call method2. The method2 contains the file reading code. The program looks like one given below

```
// File ThrowsDemo.java

import java.io.* ;

public class ThrowsDemo {


// contains file reading code
public static void method2( ) {
try{
    FileReader fr = new FileReader (“strings.txt”);
    BufferedReader br = new BufferedReader (fr);

    //read the line form file
    String s = br.readLine();
    System.out.println(s);

 catch( IOExceptoin ioEx) {
   ioEx.printStackTrace();
 }
}// end method 2
//only calling method2
public static void method1( ) {
   method2();
}
public static void main (String args[ ]) {
   ThrowsDemo.method1();
}
}
```

printStackTrace method

Defined in the Throwable class – superclass of Exception & Error classes

Shows you the full method calling history with line numbers.

Extremely useful in debugging

Modify: ThrowsDemo.java

Let method2 doesn't want to handle exception by itself, so it throws the exception to the caller of method2 i.e. method1

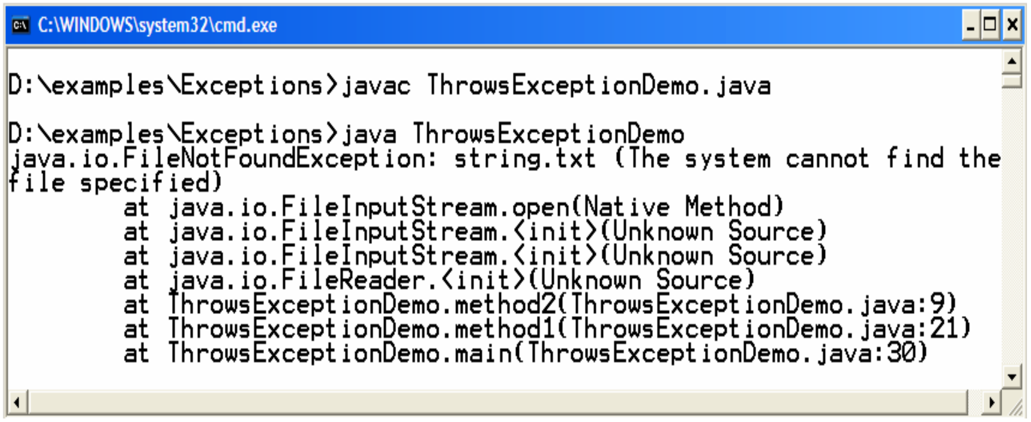
So method1 either have to handle the incoming exception or it can re-throw it to its caller i.e. main.

Let method1 is handling the exception, so method1& method2 would be modified as:

```
// File ThrowsDemo.java

 import java.io.* ;

 public class ThrowsDemo {

 // contains file reading code
 public static void method2() throws IOEception{
    FileReader fr = new FileReader ("strings.txt");
    BufferedReader br = new BufferedReader (fr);

    //read the line form file
    String s = br.readLine();
    System.out.println(s);

 }// end method 2

 // calling method2 & handling incoming exception
 public static void method1( ) {
   try {
     method2();
   catch (IOException ioEx) {
     ioEx.printStackTrace();
   }
 }

 public static void main (String args[ ]) {
     ThrowsDemo.method1();
 }
}
```

Compile & Execute

If file *strings.txt* is not present in the same directory, method2 will throw an exception that would be caught by method1 and the printStackTrace method will print the full calling history on console. The above scenario is shown in the output below:



```
D:\examples\Exceptions>javac ThrowsExceptionDemo.java

D:\examples\Exceptions>java ThrowsExceptionDemo
java.io.FileNotFoundException: string.txt (The system cannot find the
file specified)
        at java.io.FileInputStream.open(Native Method)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.io.FileInputStream.<init>(Unknown Source)
        at java.io.FileReader.<init>(Unknown Source)
        at ThrowsExceptionDemo.method2(ThrowsExceptionDemo.java:9)
        at ThrowsExceptionDemo.method1(ThrowsExceptionDemo.java:21)
        at ThrowsExceptionDemo.main(ThrowsExceptionDemo.java:30)
```

If file *strings.txt* exist there, than hopefully line would be displayed on the console.

Reference

Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 8

Streams

I/O libraries often use the abstraction of a *stream*, which represents any data source or sink as an object capable of producing or receiving pieces of data.

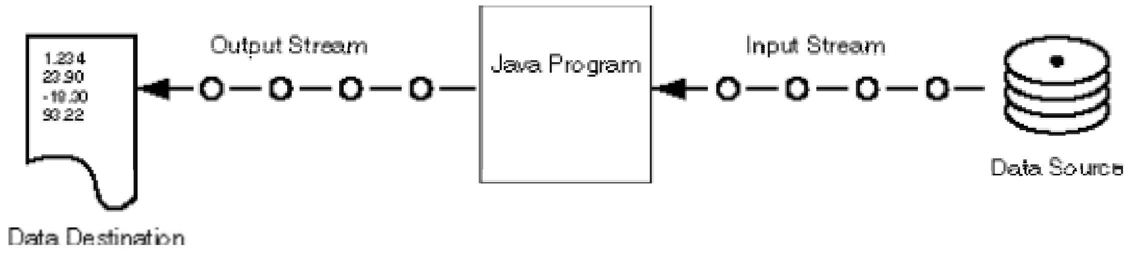
The Java library classes for I/O are divided by input and output. You need to import java.io package to use streams. There is no need to learn all the streams just do it on the need basis.

The concept of "streams"

- It is an abstraction of a data source/sink
    - We need abstraction because there are lots of different devices (files, consoles, network, memory, etc.). We need to talk to the devices in different ways (sequential, random access, by lines, etc.) Streams make the task easy by acting in the same way for every device. Though inside handling of devices may be quite different, yet on the surface everything is similar. You might read from a file, the keyboard, memory or network connection, different devices may require specialization of the basic stream, but you can treat them all as just "streams". When you read from a network, you do nothing different than when you read from a local file or from user's typing

```
//Reading from console
BufferedReader stdin =  new BufferedReader(new InputStreamReader(
                                                               System.in ));
                                   -------- ( your console)
// Reading from file
BufferedReader br=new BufferedReader(new FileReader("input.txt"));

//Reading from network
BufferedReader br = new BufferedReader(new InputStreamReader
                                   (s.getInputStream()));
                                   ---- "s" is the socket
```

- So you can consider stream as a data path. Data can flow through this path in one direction between specified terminal points (your program and file, console, socket etc.)



Stream classification based on Functionality

Based on functionality streams can be categorized as Node Stream and Filter Stream. Node Streams are those which connect directly with the data source/sick and provide basic functionality to read/write data from that source/sink

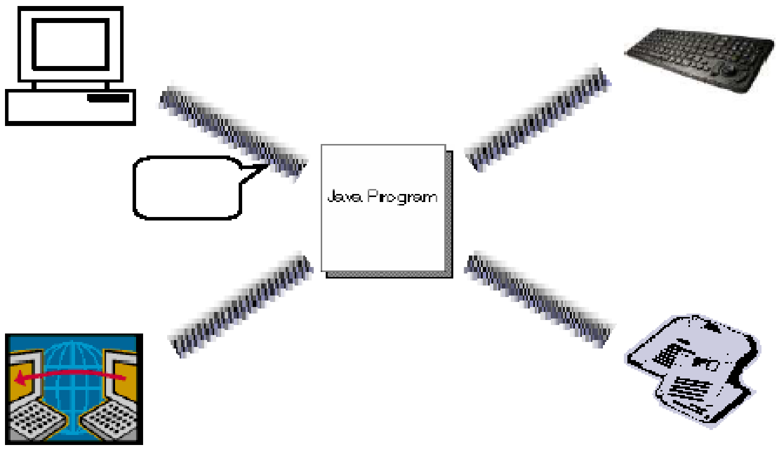
```
FileReader fr = new FileReader("input.txt");
```

You can see that FileReader is taking a data/source "input.txt" as its argument and hence it is a node stream.

FilterStreams sit on top of a node stream or chain with other filter stream and provide some additional functionality e.g. compression, security etc. FilterStreams take other stream as their input.

```
BufferedReader bt = new BufferedReader(fr);
```

BufferedReader makes the IO efficient (enhances the functionality) by buffering the input before delivering. And as you can see that BufferedReader is sitting on top of a node stream which is FileReader.

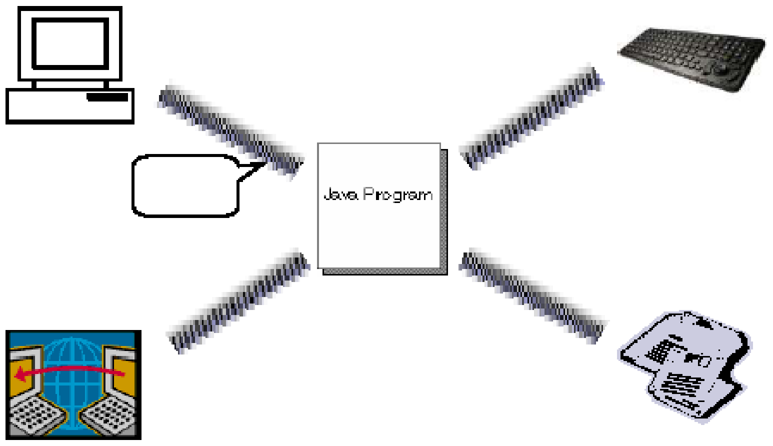


Stream classification based on data

Two type of classes exists.

Classes which contain the word stream in their name are byte oriented and are here since JDK1.0. These streams can be used to read/write data in the form of bytes. Hence classes with the word stream in their name are byte-oriented in nature. Examples of byte oriented streams are FileInputStream, ObjectOutputStream etc.

Classes which contain the word Reader/Writer are character oriented and read and write data in the form of characters. Readers and Writers came with JDK1.1. Examples of Reader/Writers are FileReader, PrintWriter etc



Example Code 8.1: Reading from File

The ReadFileEx.java reads text file line by line and prints them on console. Before we move on to the code, first create a text file (*input.txt*) using notepad and write following text lines inside it.

Text File: input.txt

```
Hello World
Pakistan is our homeland
Web Design and Development
```

```
// File ReadFileEx.java

import java.io.*;

public class ReadFileEx {
  public static void main (String args[ ]) {

   FileReader fr = null;
   BufferedReader br = null;

  try {

    // attaching node stream with data source
    fr = new FileReader("input.txt");

    // attatching filter stream over node stream
    br = new BufferedReader(fr);


    // reading first line from file
    String line = br.readLine();

    // printing and reading remaining lines
    while (line != null){
       System.out.println(line);
       line = br.readLine();
    }

    // closing streams
    br.close();
    fr.close();

  }catch(IOException ioex){
    System.out.println(ioex);
  }
 } // end main
} // end class
```

Example Code 8.2: Writing to File

The WriteFileEx.java writes the strings into the text file named "output.txt". If "output.txt" file does not exist, the java will create it for you.

```
// File WriteFileEx.java

import java.io.*;

public class WriteFileEx {

  public static void main (String args[ ]) {

   FileWriter fw  = null;
   PrintWriter pw = null;

  try {

    // attaching node stream with data source
    // if file does not exist, it automatically creates it
    fw = new FileWriter ("output.txt");

    // attatching filter stream over node stream
    pw = new PrintWriter(fw);

    String s1 = "Hello World";
    String s2 = "Web Design and Development";

    // writing first string to file
    pw.println(s1);

    // writing second string to file
    pw.println(s2);

    // flushing stream
    pw.flush();

    // closing streams
    pw.close();
    fw.close();

  }catch(IOException ioex){
    System.out.println(ioex);
  }
  } // end main
  } // end class
```

After executing the program, check the output.txt file. Two lines will be written there.

Reference

Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 9

Modification of Address Book Code

Adding Persistence Functionality

Hopefully, your address book you built previously is giving you the required results except one i.e. persistence. You might have noticed that after adding some person records in the address book; if you exit form the program next time on re-executing address book all the previous records are no more available.

To overcome the above problem, we will modify our program so that on exiting/starting of address book, all the previously added records are available each time. To achieve this, we have to provide the persistence functionality. Currently, we will accomplish this task by saving person records in some text file.

Supporting simple persistence by any application requires handling of two scenarios. These are

On start up of application – data (person records) must be read from file
On end/finish up of application – data (person records) must be saved in file

To support persistence, we have to handle the above mentioned scenarios

Scenario 1 – Start Up

Establish a data channel with a file by using streams

Start reading data (person records) from file line by line

Construct PersonInfo objects from each line you have read

Add those PersonInfo objects in arraylist persons.

Close the stream with the file

Perform these steps while application is loading up

We will read records from a text file named persons.txt. The person records will be present in the file in the following format.

```
Ali,defence,9201211
Usman,gulberg,5173940
Salman,LUMS,5272670
```

persons.txt

As you have seen, each person record is on a separate line. Person's name, address & phone number is separated using comma (,).

We will modify our AddressBook.java by adding a new method loadPersons into it. This method will provide the implementation of all the steps. The method is shown below:

```
public void loadPersons ( ){

    String tokens[] = null;
    String name, add, ph;

    try {

        FileReader fr = new FileReader("persons.txt");
        BufferedReader br = new BufferedReader(fr);

        String line = br.readLine();

        while ( line != null ) {

            tokens = line.split(",");

            name = tokens[0];
            add  = tokens[1];
            ph   = tokens[2];

            PersonInfo p =  new PersonInfo(name, add, ph);
            persons.add(p);

                    line = br.readLine();
        }

        br.close();
        fr.close();

    }catch(IOException ioEx){
      System.out.println(ioEx);
    }
}
```

First, we have to connect with the text file in order to read line by line person records from it. This task is accomplished with the following lines of code

```
FileReader fr = new FileReader("persons.txt"); BufferedReader br = new
BufferedReader(fr);
```

FileReader is a character based (node) stream that helps us in reading data in the form of characters. As we are using streams, so we have to import the java.io package in the AddressBook class.

We passed the file name persons.txt to the constructor of the FileReader.

Next we add BufferedReader (filter stream) on top of the FileReader because BufferedReader facilitates reading data line by line. (As you can recall from the lecture that filter streams are attached on top of node streams). That's why the constructor of BufferedReader is receiving the fr – the FileReader object.

The next line of code will read line from file by using readLine( ) method of BufferedReader and save it in a string variable called line.

```
String line = br.readLine( );
```

After that while loop starts. The condition of while loop is used to check whether the file is reached to end (returns null) or not. This loop is used to read whole file till the end. When end comes (null), this loop will finish.

```
while (line != null)
```

Inside loop, the first step we performed is tokenizing the string. For this purpose, we have used split method of String class. This method returns substrings (tokens) according to the regular expression or delimiter passed to it.

```
tokens = line.split(“,”);
```

The return type of this method is array of strings that’s why we have declared tokens as a String array in the beginning of this method as

```
String tokens[];
```

For example, the line contains the following string

```
Ali,defence,9201211
```

Now by calling split(“,”) method on this string, this method will return back three substrings *ali defence* and *9201211* because the delimiter we have passed to it is comma. The delimiter itself is not included in the substrings or tokens.

The next three lines of code are simple assignments statements. The tokens[0] contains the *name* of the person because the name is always in the beginning of the line, tokens[1] contains *address* of the person and tokens[2] contains the phone number of the person.

```
 name = tokens[0];
add  = tokens[1];
ph   = tokens[2];
```

The name, add and ph are of type String and are declared in the beginning of this method.

After that we have constructed the object of PersonInfo class by using parameterized constructor and passed all these strings to it.

```
PersonInfo p = new PersonInfo(name, add, ph);
```

Afterward the PersonInfo object’s p is added to the arraylist i.e. persons. persons.add(p);

The last step we have done inside loop is that we have again read a line form the file by using the readLine() method.

By summarizing the task of while loop we can conclude that it reads the line from a file, Tokenize that line into three substrings followed by constructing the PersonInfo object by using these tokens. And adding these objects to the arraylist. This process continues till the file reaches its end.

The last step for reading information from the file is ordinary one – closing the streams, because files are external resources, so it's better to close them as soon as possible.

Also observe that we used try/catch block because using streams can result in raising exceptions that falls under the checked exceptions category – that needs mandatory handling.

The last important step you have to perform is to call this method while loading up. The most appropriate place to call this method is from inside the constructor of AddressBook.java. So the constructor will now look like similar to the one given below:

```
………………
 public AddressBook () {
     Persons = new ArrayList();
     loadPersons();
 }
………………
```

AddressBook.java

Scenario 2 – End/Finish Up

Establish a datachanel(stream) with a file by using streams

Take out PersonInfo objects from ArrayList (persons)

Build a string for each PersonInfo object by inserting commas (,) between name & address and address & phone number.

Write the constructed string to the file

Close the connection with file

Perform these steps while exiting from address book.

Add another method savePersons into AddressBook.java. This method will provide the implementation of all the above mentioned steps. The method is shown below:

```
Public void savePersons ( ){

   try {

      PersonInfo p;
      String line;

      FileWriter fw = new FileWriter("persons.txt");
      PrintWriter pw = new PrintWriter(fw);

      for(int i=0; i<persons.size(); i++)
      {
        p = (PersonInfo)persons.get(i);
        line = p.name +","+ p.address +","+ p.phoneNum;


        // writes line to file (persons.txt)
        pw.println(line);

      }

      pw.flush();
      pw.close();
      fw.close();

   }catch(IOException ioEx){
     System.out.println(ioEx);
   }
}
```

As you can see, that we have opened the same file (persons.txt) again by using a set of streams.

After that we have started for loop to iterate over arraylist as we did in searchPerson and deletePerson methods.

Inside for loop body, we have taken out PersonInfo object and after type casting it we have assigned its reference to a PersonInfo type local variable p. This is achieved by the help of following line of code

```
p = (PersonInfo)persons.get(i);
```

Next we build a string and insert commas between the PersonInfo attributes and assign the newly constructed string to string's local variable line as shown in the following line of code.

```
line = p.name +","+ p.address +","+ p.phoneNum;
```

Note: Since, we haven't declare PersonInfo attributes private, therefore we are able to directly access them inside AddressBook.java.

The next step is to write the line representing one PersonInfo object's information, to the file. This is done by using println method of PrintWriter as shown below

```
pw.println(line);
```

After writing line to the file, the println method will move the cursor/control to the next line. That's why each line is going to be written on separate line.

The last step for saving information to the file is ordinary one – closing the streams but before that notice the code line that you have not seen/performed while loading persons records from file. That is

```
pw.flush();
```

The above line immediately flushes data by writing any buffered output/data to file. This step is necessary to perform or otherwise you will most probably lose some data for the reason that PrintWriter is a Buffered Stream and they have their own internal memory/storage capacity for efficiency reasons. Buffered Streams do not send the data until their memory is full.

Also we have written this code inside try-catch block.

The last important step you have to perform is to call this method before exiting from the address book. The most appropriate place to call this method is under case 4 (exit scenario) in Test.java. So the case 4 will now look like similar to the one given below:

```
case 4:
      ab.savePersons();
      System.exit(0);
```

Test.java

Compile & Execute

Now again after compiling all the classes, run the Test class. Initially we are assuming that out persons.txt file is empty, so our arraylist persons will be empty on the first start up of address book. Now add some records into it, perform search or delete operations. Exit from the address book by choosing option 4. Check out the persons.txt file. Don't get surprised by seeing that it contains all the person records in the format exactly we have seen above.

Next time you will run the address book; all the records will be available to you. Perform the search or delete operation to verify that.

Finally You have done it !!!

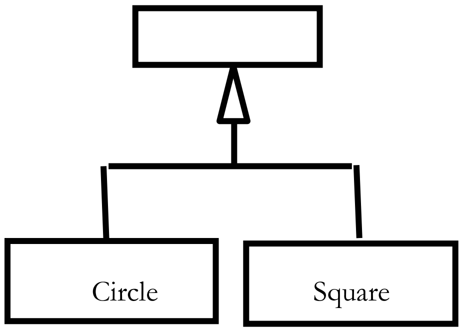
References

Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Abstract Classes and Interfaces

Problem and Requirements

Before moving on to abstract classes, first examine the following class hierarchy shown below:



- Suppose that in order to exploit polymorphism, we specify that 2-D objects must be able to compute their area.
  - All 2-D classes must respond to area() message.

- How do we ensure that?
  - Define area method in class Shape
  - Force the subclasses of Shape to respond area() message

- Java's provides us two solutions to handle such problem
  - Abstract Classes
  - Interfaces

Abstract Classes

Abstract classes are used to define only part of an implementation. Because, information is not complete therefore an abstract class cannot be instantiate. However, like regular classes, they can also contain instance variables and methods that are full implemented. The class that inherits from abstract class is responsible to provide details.

Any class with an abstract method (a method has no implementation similar to pure virtual function in C++) must be declared abstract, yet you can declare a class *abstract* that has no abstract method.

If subclass overrides all abstract methods of the super class, than it becomes a concrete (a class whose object can be instantiate) class otherwise we have to declare it as abstract or we can not compile it.

The most important aspect of abstract class is that reference of an abstract class can point to the object of concrete classes.

Code Example of Abstract Classes

The **Shape** class contains an abstract method calculateArea() with no definition.

```
public abstract class Shape{
      public abstract void calculateArea();
}
```

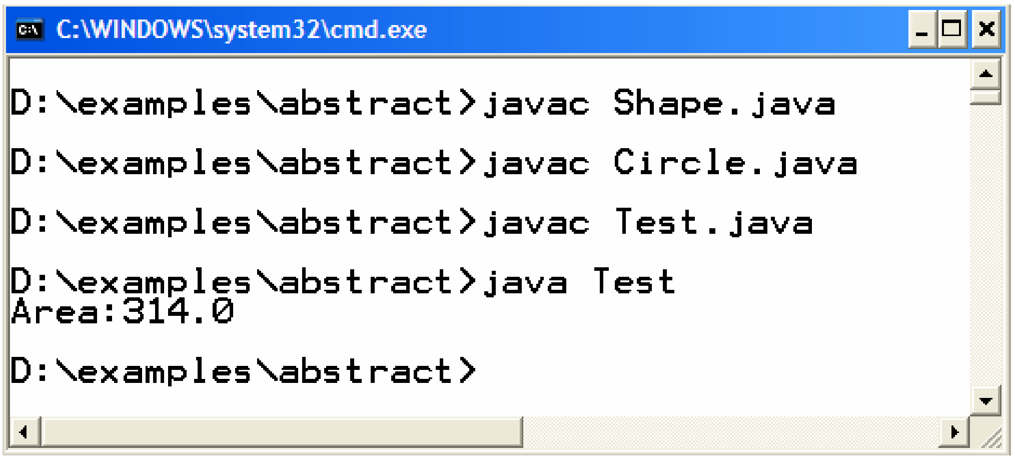
Class **Circle** extends from abstract Shape class, therefore to become concrete class it must provides the definition of calculateArea() method.

```
public class Circle extends Shape {

    private int x, y;
    private int radius;

    public Circle() {
        x = 5;
        y = 5;
        radius = 10;
    }

    // providing definition of abstract method
    public void calculateArea () {

       double area = 3.14 * (radius * radius);
       System.out.println("Area: " + area);

    }
}//end of class
```

The Test class contains main method. Inside main, a reference s of abstract Shape class is created. This reference can point to Circle (subclass of abstract class Shape) class object as it is a concrete class. With the help of reference, method calculateArea() can be invoked of Circle class. This is all shown in the form of code below

```
public class Test {
  public static void main(String args[]){
    //can only create references of A.C.
    Shape s = null;

    //Shape s1 = new Shape(); //cannot instantiate

    //abstractclass reference can point to concrete subclass
    s = new Circle();

    s.calculateArea();
  }
}//end of class
```

The compilation and execution of the above program is shown below:


C:\WINDOWS\system32\cmd.exe

```
D:\examples\abstract>javac Shape.java

D:\examples\abstract>javac Circle.java

D:\examples\abstract>javac Test.java

D:\examples\abstract>java Test
Area:314.0

D:\examples\abstract>
```

Interfaces

As we seen one possible java's solution to problem discussed in start of the tutorial. The second possible java's solution is Interfaces.

Interfaces are special java type which contains only a set of method prototypes, but doest not provide the implementation for these prototypes. All the methods inside an interface are abstract by default thus an interface is tantamount to a pure abstract class – a class with zero implementation. Interface can also contains static final constants

Defining an Interface

Keyword interface is used instead of class as shown below:

```
public interface Speaker{
      public void speak();
}
```

Implementing (using) Interface

Classes *implement* interfaces. Implementing an interface is like *signing a contract*. A class that implements an interface will have to provide the definition of all the methods that are present inside an interface. If the class does not provide definitions of all methods, the class would not compile. We have to declare it as an abstract class in order to get it compiled.

Relationship between a class and interface is equivalent to *"responds to"* while *"is a"* relationship exists in inheritance.

Code Example of Defining & Implementing an Interface

The interface **Printable** contains print() method.

```
public interface Printable{
      public void print();
}
```

Class **Student** is implementing the interface Printable. Note the use of keyword *implements* after the class name. Student class has to provide the definition of print method or we are unable to compile.

The code snippet of student class is given below:

```
public class Student implements Printable {
   private String name;
   private String address;

   public String toString () {
      return "name:"+name +" address:"+address;
   }

   //providing definition of interface's print method
   public void print() {
            System.out.println("Name:"  +name+"  address"+address);
   }
}//end of class
```

Interface Characteristics

Similar to abstract class, interfaces imposes a design structure on any class that uses the interface. Contrary to inheritance, a class can implement more than one interfaces. To do this separate the interface names with comma. This is java's way of multiple inheritance.

```
class Circle implements Drawable , Printable { ……… }
```

Objects of interfaces also cannot be instantiated.

```
Speaker s = new Speaker(); // not compile
```

However, a reference of interface can be created to point any of its implementation class. This is interface based polymorphism.

Code Example: Interface based polymorphism

Interface Speaker is implemented by three classes Politician, Coach and Lecturer. Code snippets of all these three classes are show below:

```
public class Politician implements Speaker{
    public void speak(){
        System.out.println("Politics Talks");
    }
}
```

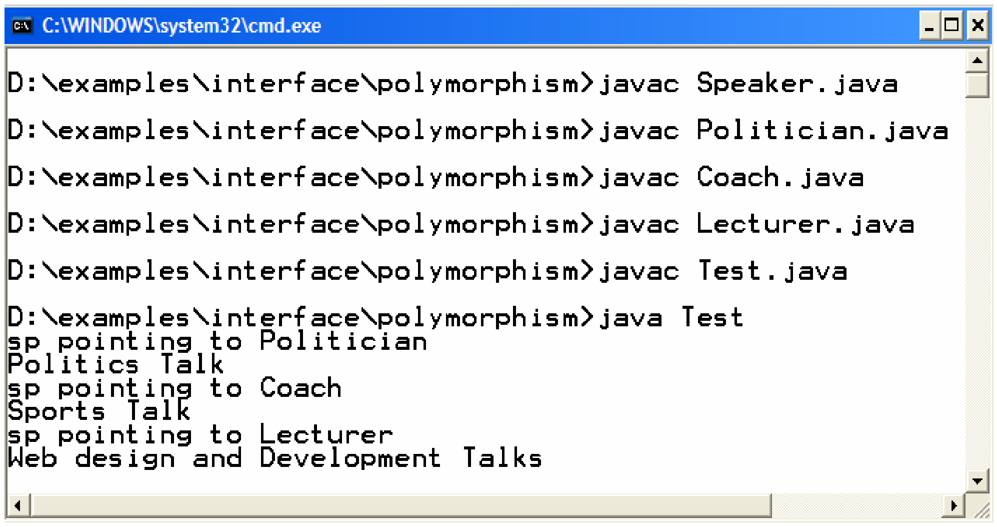
```
public class Coach implements Speaker{
    public void speak(){
        System.out.println("Sports Talks");
    }
}
```

```
public class Lecturer implements Speaker{
   public void speak(){
     System.out.println("Web Design and Development Talks");
   }
}
```

As usual, Test class contains main method. Inside main, a reference sp is created of Speaker class. Later, this reference is used to point to the objects of Politician, Coach and Lecturer class. On calling speak method with the help of sp, will invoke the method of a class to which sp is pointing.

```
public class Test{
    public static void main (String args[ ]) {

        Speaker sp = null;

        System.out.println("sp pointing to Politician");
        sp = new Politician();
        sp.speak();

        System.out.println("sp pointing to Coach");
        sp = new Coach();
        sp.speak();

        System.out.println("sp pointing to Lecturer");
        sp = new Lecturer();
        sp.speak();
    }
}
```

The compilation and execution of the above program is shown below:



```
D:\examples\interface\polymorphism>javac Speaker.java

D:\examples\interface\polymorphism>javac Politician.java

D:\examples\interface\polymorphism>javac Coach.java

D:\examples\interface\polymorphism>javac Lecturer.java

D:\examples\interface\polymorphism>javac Test.java

D:\examples\interface\polymorphism>java Test
sp pointing to Politician
Politics Talk
sp pointing to Coach
Sports Talk
sp pointing to Lecturer
Web design and Development Talks
```

References

Example code, their explanations and corresponding figures for this handout are taken from the book JAVA A Lab Course by Umair Javed. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Graphical User Interfaces

A graphical user interface is a visual interface to a program. GUIs are built from GUI components (buttons, menus, labels etc). A GUI component is an object with which the user interacts via the mouse or keyboard.

Together, the appearance and how user interacts with the program are known as the program look and feel.

Support for GUI in Java

The classes that are used to create GUI components are part of the "java.awt" or "javax.swing" package. Both these packages provide rich set of user interface components.

GUI classes vs. Non-GUI Support Classes

The classes present in the awt and swing packages can be classified into two broad categories. GUI classes & Non-GUI Support classes.

The GUI classes as the name indicates are visible and user can interact with them. Examples of these are JButton, JFrame & JRadioButton etc

The Non-GUI support classes provide services and perform necessary functions for GUI classes. They do not produce any visual output. Examples of these classes are Layout managers (discussed latter) & Event handling (see handout on it) classes etc.

java.awt package

AWT stands for "Abstract Windowing Toolkit "contains original GUI components that came with the first release of JDK. These components are tied directly to the local platform's (Windows, Linux, MAC etc) graphical user interface capabilities. Thus results in a java program executing on different java platforms (windows, Linux, Solaris etc) has a different appearance and sometimes even different user interaction on each platform.

AWT components are often called Heavy Weight Components (HWC) as they rely on the local platform's windowing system to determine their functionality and their look and feel. Every time you create an AWT component it creates a corresponding process on the operating system. As compared to this SWING components are managed through threads and are known as Light Weight Components.

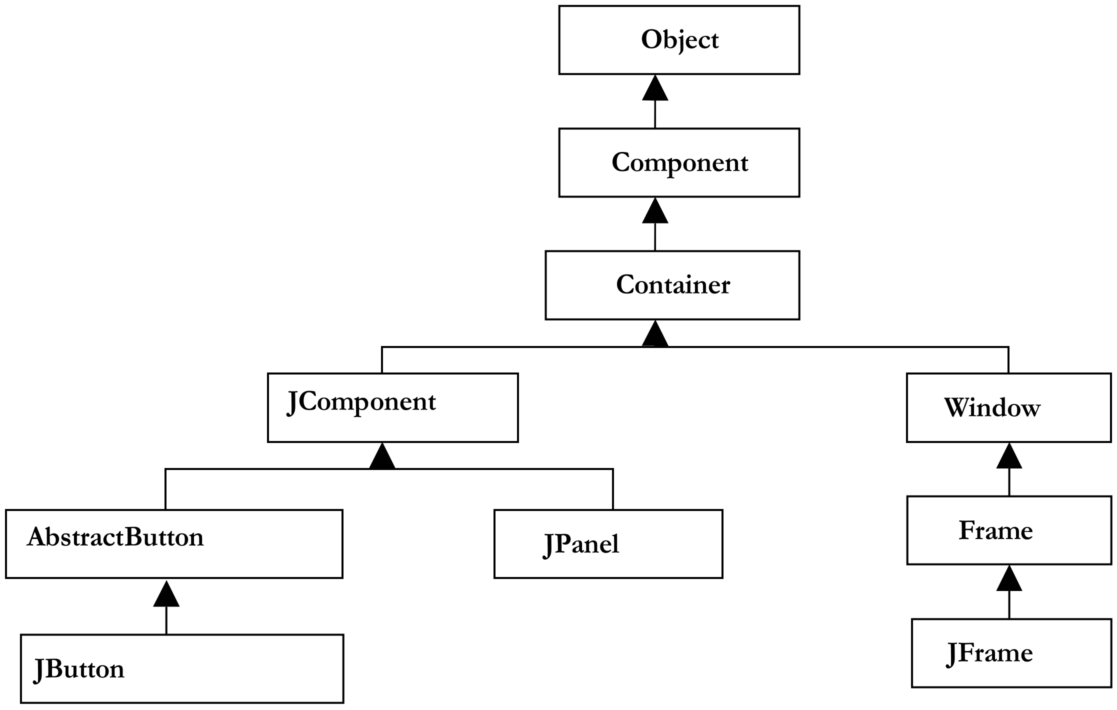
This package also provides the classes for robust event handling (see handout on it) and layout managers.

javax.swing package

These are the newest GUI components. Swing components are written, manipulated and displayed completely in java, therefore also called pure java components. The swing components allow the programmer to specify a uniform look and feel across all platforms.

Swing components are often referred to as Light Weight Components as they are completely written in java. Several swing components are still HWC. e.g. JFrame etc.

A part of the FrameWork



GUI Creation Steps

1. **import required packages**

import java.awt.* and/or javax.swing.* package.

2. **Setup the top level containers**

A container is a collection of related components, which allows other components to be nested inside it. In application with JFrame, we attatch components to the content pane – a container.

Two important methods the container class has **add** and **setLayout.**

The add method is used for adding components to the content pane while setLayout method is used to specify the layout manager.

Container are classified into two broad categories that are Top Level containers and General Purpose Containers

Top level containers can contain (add) other containers as well as basic components (buttons, labels etc) while general purpose containers are typically used to collect basic components and are added to top level containers.

General purpose containers cannot exist alone they must be added to top level containers

Examples of top level container are JFrame, Dialog and Applet etc. Our application uses one of these.

Examples of general purpose container are JPanel, Toolbar and ScrollPane etc.

So, take a top level container and create its instance. Consider the following code of line if JFrame is selected as a top level container

JFrame *frame* = new JFrame();

3. **Get the component area of the top level container**

Review the hierarchy given above, and observe that JFrame *is a* frame *is a* window. So, it can be interpreted as JFrame *is a* window.

Every window has two areas. System Area & Component Area

The programmer cannot add/remove components to the System Area.

The Component Area often known as Client area is a workable place for the programmer. Components can be added/removed in this area.

So, to add components, as you guessed right component area of the JFrame is required. It can be accomplished by the following code of line

```
Conntainer con = frame.getContentPane();
```

frame is an instance of JFrame and by calling *getContentPane()* method on it, it returns the component area. This component area is of type container and that is why it is stored in a variable of a Container class. As already discussed, container allows other components to be added / removed.

4. **Apply layout to component area**

The layout (size & position etc. How they appear) of components in a container is usually governed by Layout Managers.

The layout manager is responsible for deciding the layout policy and size of its components added to the container.

Layout managers are represented in java as classes. (Layout Managers are going to be discussed in detail later in this handout)

To set the layout, as already discussed use setLayout method and pass object of layout manager as an argument.

```
con.setLayout( new FlowLayout( ) );
```

We passed an object of FlowLayout to the setLayout method here.

We can also use the following lines of code instead of above. FlowLayout layout = new FlowLayout();con.setLayout(layout);

5. **Create and Add components**

Create required components by calling their constructor.

```
JButton button = new JButton ();
```

After creating all components your are interested in, the next task is to add these components into the component area of your JFrame (i.e ContentPane, the reference to which is in variable con of type Container)

Use *add* method of the Container to accomplish this and pass it the component to be added.

```
con.add(button);
```

6. **Set size of frame and make it visible**

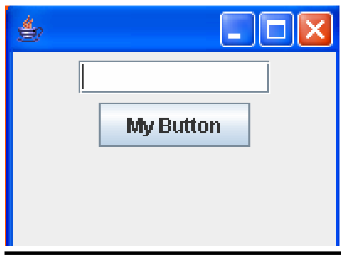
A frame must be made visible via a call to setVisible(true) and its size defined via a call setSize(rows in pixel, columns in pixel) to be displayed on the screen.

```
frame.setSize(200, 300) ;
frame.setVisible(true) ;
```

Note: By default, all JFrame's are invisible. To make visible frame visible we have passed *true* to the setVisible method.

```
frame.setVisible(false) ;
```

Making a Simple GUI



The above figured GUI contains one text field and a button. Let's code it by following the six GUI creation steps we discussed.

Code for Simple GUI

```
// File GUITest.java

//Step 1: import packages import java.awt.*;
import javax.swing.*;

public class GUITest {

  JFrame myFrame ;
  JTextField tf;
  JButton b;

  //method used for setting layout of GUI
    public void initGUI ( ) {

     //Step 2: setup the top level container
     myFrame = new JFrame();


     //Step 3: Get the component area of top-level  container
     Container c = myFrame.getContentPane();


     //Step 4: Apply layouts
     c.setLayout( new FlowLayout( ) );

    //Step 5: create & add components
    JTextField tf = new JTextField(10);
    JButton b1 = new JButton("My Button");

    c.add(tf);
    c.add(b1);

    //Step 6: set size of frame and make it visible
        myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    myFrame.setSize(200,150);
    myFrame.setVisible(true);

  } //end initGUI method


   public GUITest () {  // default constructor
     initGUI ();
   }


   public static void main (String args[ ]) {
      GUITest gui = new GUITest();
   }

} // end of class
```

Important Points to Consider

main method (from where program execution starts) is written in the same class. The main method can be in a separate class instead of writing in the same class its your choice.

Inside main, an object of GUI test class is created that results in calling of constructor of the class and from the constructor, *initGUI* method is called that is responsible for setting up the GUI.

The following line of code is used to exit the program when you close the window

```
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

If you delete this line and run your program, the desired GUI would be displayed. However if you close the window by using (X) button on top left corner of your window, you'll notice that the control doesn't return back to command prompt. The reason for this is that the java process is still running. How ever if you put this line in your code, when you exit your prompt will return.

References:

Sun java tutorial: http://java.sun.com/docs/books/tutorial/java
Thinking in java by Bruce Eckle
Beginning Java2 by Ivor Hortan
GUI creation steps are taken from the book Java A Lab Course by Umair Javed

Graphical User Interfaces - 2

Layout Managers

Layout Managers are used to form the appearance of your GUI. They are concerned with the arrangement of components of GUI. A general question is "why we can not place components at our desired location (may be using the x,y coordinate position?"

The answer is that you can create your GUI without using Layout Managers and you can also do VB style positioning of components at some x,y co-ordinate in Java, but that is generally not advisable if you desire to run the same program on different platforms

The appearance of the GUI also depends on the underlying platform and to keep that same the responsibility of arranging layout is given to the LayoutManagers so they can provide the same look and feel across different platforms

Commonly used layout managers are

1. Flow Layout
2. Grid Layout
3. Border Layout
4. Box Layout
5. Card Layout
6. GridBag Layout and so on

Let us discuss the top three in detail one by one with code examples. These top three will meet most of your basic needs

<u>1. Flow Layout</u>

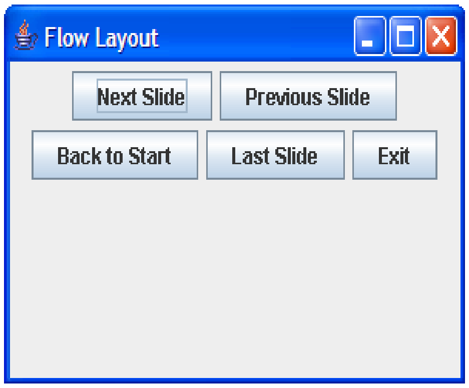
Position components on line by line basis. Each time a line is filled, a new line is started.

The size of the line depends upon the size of your frame. If you stretch your frame while your program is running, your GUI will be disturbed.

<u>Example Code</u>

```
// File FlowLayoutTest.java

import java.awt.*;
import javax.swing.*;

public class FlowLayoutTest {

  JFrame myFrame ;
  JButton b1, b2, b3, b4, b5;

  //method used for setting layout of GUI
    public void initGUI ( ) {
        myFrame = new JFrame("Flow Layout");

        Container c = myFrame.getContentPane();

            c.setLayout( new FlowLayout( ) );

        b1 =  new JButton("Next Slide");
        b2 =  new JButton("Previous Slide");
        b3 =  new JButton("Back to Start");
        b4 =  new JButton("Last Slide");
        b5 =  new JButton("Exit");

        c.add(b1);
        c.add(b2);
        c.add(b3);
        c.add(b4);
        c.add(b5);

       myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       myFrame.setSize(300,150);
       myFrame.setVisible(true);

     } //end initGUI method


     public FlowLayoutTest () {  // default constructor
        initGUI ();
     }


     public static void main (String args[ ]) {
         FlowLayoutTest flTest = new FlowLayoutTest();
     }

   } // end of class
```

Output



2. Grid Layout

Splits the panel/window into a grid (cells) with given number of rows and columns.
Forces the size of each component to occupy the whole cell. Size of each component is same

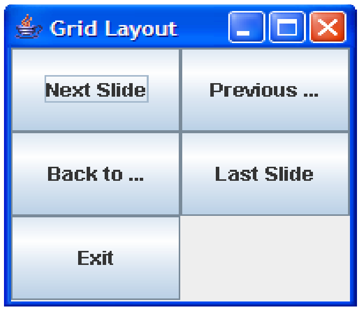
Components are added row wise. When all the columns of the first row are get filled the components are then added to the next row.
Only one component can be added into each cell.

Example Code

```
// File GridLayoutTest.java

import java.awt.*;
import javax.swing.*;

public class GridLayoutTest {

  JFrame myFrame ;
  JButton b1, b2, b3, b4, b5;

  //method used for setting layout of GUI
    public void initGUI ( ) {


      myFrame = new JFrame("Grid Layout");

      Container c = myFrame.getContentPane();

                        // rows , cols
      c.setLayout( new GridLayout( 3  , 2  ) );

      b1 =  new JButton("Next Slide");
      b2 =  new JButton("Previous Slide");
      b3 =  new JButton("Back to Start");
      b4 =  new JButton("Last Slide");
      b5 =  new JButton("Exit");

      c.add(b1);
      c.add(b2);
      c.add(b3);
      c.add(b4);
      c.add(b5);

     myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     myFrame.setSize(300,150);
     myFrame.setVisible(true);



  } //end initGUI method
```

```
    public GridLayoutTest () {  // default constructor
        initGUI ();
     }


      public static void main (String args[ ]) {
         GridLayoutTest glTest = new GridLayoutTest();
     }

    } // end of class
```

output



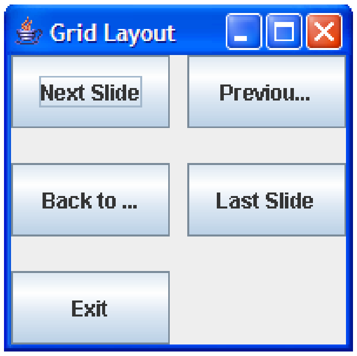
Modification

The grid layout also allows the spacing between cells. To achieve spacing between cells, modify the above program.

Pass additional parameters to the constructor of GridLayout, spaces between rows & spaces between columns as shown below

```
        c.setLayout( new GridLayout( 3 , 2   ,  10 ,  20)  );
```

The output is look similar to one given below.



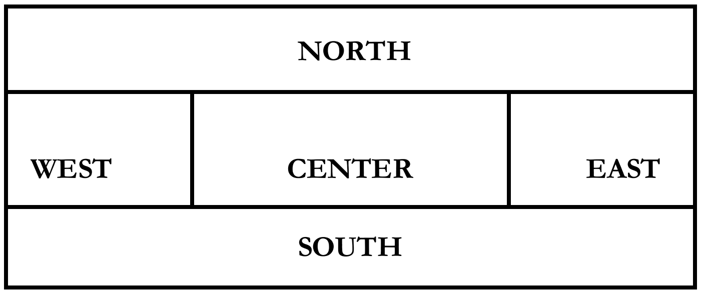
3.**Border Layout**

Divides the area into five regions. North, South, East, West and Center

Components are added to the specified region

If any region not filled, the filled regions will occupy the space but the center region will still appear as background if it contains no component.

Only one component can be added into each region.



Example Code

```
// File BorderLayoutTest.java

import java.awt.*;
import javax.swing.*;

public class BorderLayoutTest {

  JFrame myFrame ;
  JButton b1, b2, b3, b4, b5;

  //method used for setting layout of GUI
    public void initGUI ( ) {


    myFrame = new JFrame("Border Layout");

    Container c = myFrame.getContentPane();


        c.setLayout( new BorderLayout( );

    b1 =  new JButton("Next Slide");
    b2 =  new JButton("Previous Slide");
    b3 =  new JButton("Back to Start");
    b4 =  new JButton("Last Slide");
    b5 =  new JButton("Exit");
   c.add( b1 , BorderLayout.NORTH );
    c.add( b2 , BorderLayout.SOUTH );
```

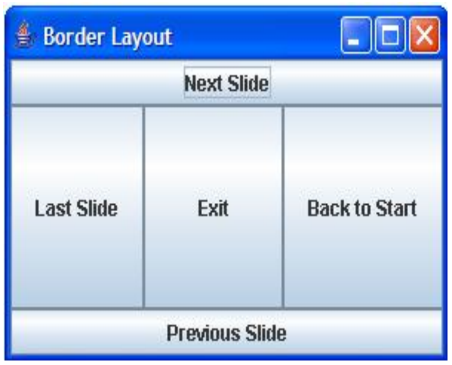
```
        c.add( b3 , BorderLayout.EAST );
          c.add( b4 , BorderLayout.WEST );

          c.add( b5 , BorderLayout.CENTER);

         myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         myFrame.setSize(300,150);
         myFrame.setVisible(true);

       } //end initGUI method


        public BorderLayoutTest () {  // default constructor
          initGUI ();
        }


        public static void main (String args[ ]) {
           BorderLayoutTest glTest = new BorderLayoutTest();
        }

     } // end of class
```

<u>Points to Remember</u>

Revisit the code of adding components, we specify the region in which we want to add component or otherwise they will not be visible.

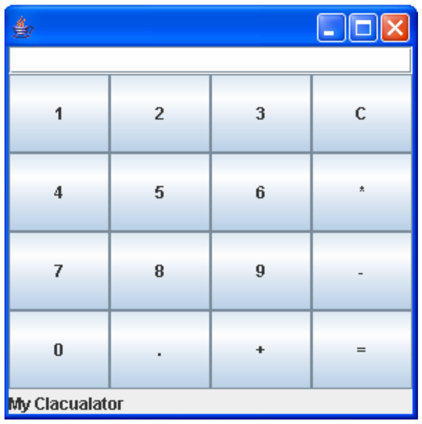
Consider the following segment of code: BorderLayout.NORTH, as you guessed correctly NORTH is a constant (final) defined in BorderLayout class public access modifier. Similarly the other ones are defined. <u>Now you understand,</u>
<u>why so much emphasis has been made on following the naming conventions</u>.

<u>Output</u>



Making Complex GUIs

From the discussion above it seems that the basic Layout Managers may not help us in constructing complex GUIs, but generally a combination of these basic layouts can do the job. So lets try to create the calculator GUI given below



This GUI has 16 different buttons each of same size and text field on the top and a label 'my calculator' on the bottom.

So, how we can make this GUI? If Border Layout is selected, it has five regions (each region can have at most one component) but here we have more than five components to add. Lets try Grid Layout, but all the components in a Grid have same size and the text field at the top and label at the bottom has different size. Flow Layout cannot be selected because if we stretch our GUI it will destroy its shape.

Can we make this GUI? Yes, we can. Making of such GUI is a bit tricky business but General Purpose Containers are there to provide the solution.

JPanel

It is general purpose container (can't exist alone, it has to be in some toplevel container) in which we can put in different components (JButton , JTextField etc even other JPanels)

JPanel has its own layout that can be set while creating JPanel instance

```
JPanel myPanel = new JPanel ( new FlowLayout( ) );
```

Add components by using *add* method like shown below.

```
myPanel.add (button );
```

Must be added to a top level container (like JFrame etc) in order to be visible as they (general purpose containers) can't exist alone.

Solution

To make the calculator GUI shown above, take JFrame (top level container) and set its layout to border. Than take JPanel (general purpose container) and set its layout to Grid with 4 rows and 4 columns.

Add buttons to JPanel as they all have equal size and JPanel layout has been set to GridLayout. Afterthat, add text field to the north region, label to the south region and panel to the center region of the JFrame's container. The east and west regions are left blank and the center region will be stretched to cover up these. So, that's how we can build our calculator GUI.

Code for Calculator GUI

```
// File CalculatorGUI.java

import java.awt.*;
import javax.swing.*;

public class CalculatorGUI {

  JFrame fCalc;

  JButton b1, b2, b3, b4, b5, b6, b7, b8, b9, b0;
  JButton bPlus, bMinus, bMul, bPoint, bEqual, bClear;

      JPanel pButtons;

  JTextField  tfAnswer;

  JLabel lMyCalc;


  //method used for setting layout of GUI
   public void initGUI ( ) {

      fCalc = new JFrame();

      b0  =  new JButton("0");
      b1  =  new JButton("1");
      b2  =  new JButton("2");
      b3  =  new JButton("3");
      b4  =  new JButton("4");
      b5  =  new JButton("5");
      b6  =  new JButton("6");

      b7  =  new JButton("7");
      b8  =  new JButton("8");
      b9  =  new JButton("9");
```

```
        bPlus   =  new JButton("+");
        bMinus  =  new JButton("-");
        bMul    =  new JButton("*");
        bPoint  =  new JButton(".");
        bEqual  =  new JButton("=");
        bClear  =  new JButton("C");

        tfAnswer = new JTextField();

            lMyCalc = new JLabel("My Clacualator");

              //creating panel object and setting its layout pButtons = new JPanel (new
              GridLayout(4,4));

              //adding components (buttons) to panel
             pButtons.add(b1);
             pButtons.add(b2);
             pButtons.add(b3);
             pButtons.add(bClear);

             pButtons.add(b4);
             pButtons.add(b5);
             pButtons.add(b6);
             pButtons.add(bMul);

             pButtons.add(b7);
             pButtons.add(b8);
             pButtons.add(b9);
             pButtons.add(bMinus);

             pButtons.add(b0);
             pButtons.add(bPoint);
             pButtons.add(bPlus);
             pButtons.add(bEqual);

         // getting componenet area of JFrame
         Container con = fCalc.getContentPane();
             con.setLayout(new BorderLayout());


             //adding  components  to  container
             con.add(tfAnswer,  BorderLayout.NORTH);
             con.add(lMyCalc,  BorderLayout.SOUTH);
             con.add(pButtons,  BorderLayout.CENTER);

         fcalc.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
             fCalc.setSize(300, 300);
             fCalc.setVisible(true);

    } //end initGUI method

    public CalculatorGUI () {  // default constructor

      initGUI ();

    }
    public static void main (String args[ ]) {

        CalculatorGUI calGUI = new CalculatorGUI ();

    }

} // end of class
```

Reference:

Sun java tutorial: http://java.sun.com/docs/books/tutorial/java
Thinking in java by Bruce Eckle
Beginning Java2 by Ivor Hortan
Java A Lab Course by Umair Javed

Lesson 11

Event Handling

One of the most important aspects of most non-trivial applications (especially UI type- apps) is the ability to respond to events that are generated by the various components of the application, both in response to user interactions and other system components such as client-server processing. In this handout we will look at how Java supports event generation and handling and how to create (and process) custom events.

GUIs generate events when the user interacts with GUI. For example,

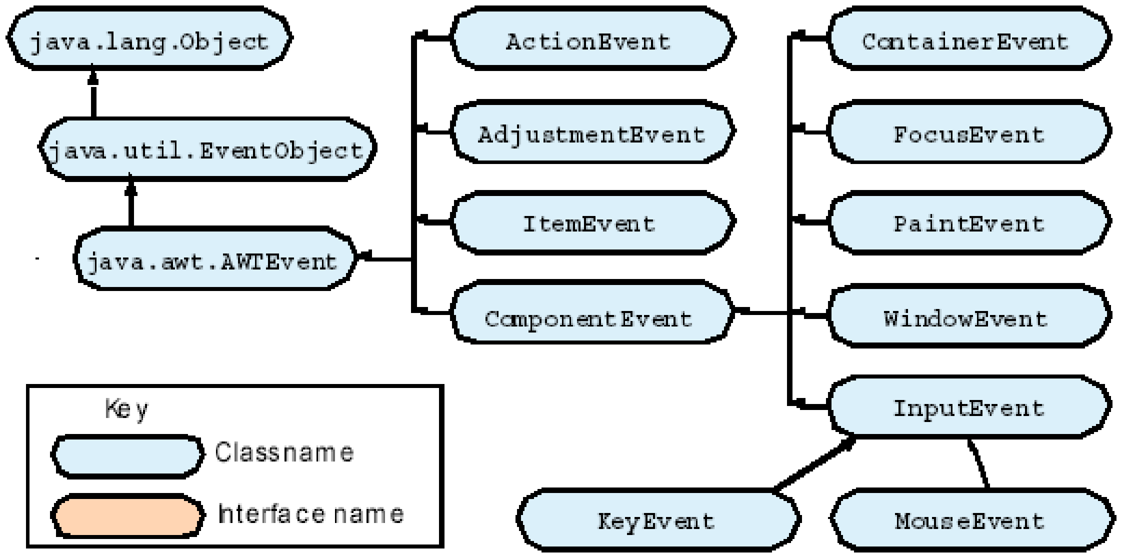
— Clicking a button
— Moving the mouse
— Closing Window etc

Both AWT and swing components (not all) generate events
— java.awt.event.*;
— javax.swing.event.*;

In java, events are represented by Objects

These objects tells us about event and its source. Examples are:
— ActionEvent (Clicking a button)
— WindowEvent (Doing something with window e.g. closing , minimizing)

Some event classes of java.awt.event are shown in diagram below



Event Handling Model

In Java both AWT and Swing components use Event Delegation Model.

- In this model processing of an event is delegated to a particular object (handlers) in the program
- It's a Publish-Subscribe model. That is, event generating component publish an event and event handling components subscribe for that event. The publisher sends these events to subscribers. Similar to the way that you subscribe for newspaper and you get the newspaper at your home from the publisher.
- This model separates UI code from program logic, it means that we can create separate classes for UI components and event handlers and hence business/program logic is separated from GUI components.

Event Handling Steps

For a programmer the event Handling is a three step process in terms of code

- **Step 1:** Create components which can generate events (Event Generators)
- **Step 2:** Build component (objects) that can handle events (Event Handlers)
- **Step 3:** Register handlers with generators

Event Handling Process

Step 1: Event Generators

The first step is that you create an event generator. You have already seen a lot of event generators like:

- Buttons
- Mouse
- Key
- Window etc

Most of GUI components can be created by calling their constructors. For example

```
JButton b1 = new JButton("Hello");
```

Now b1 can generate events

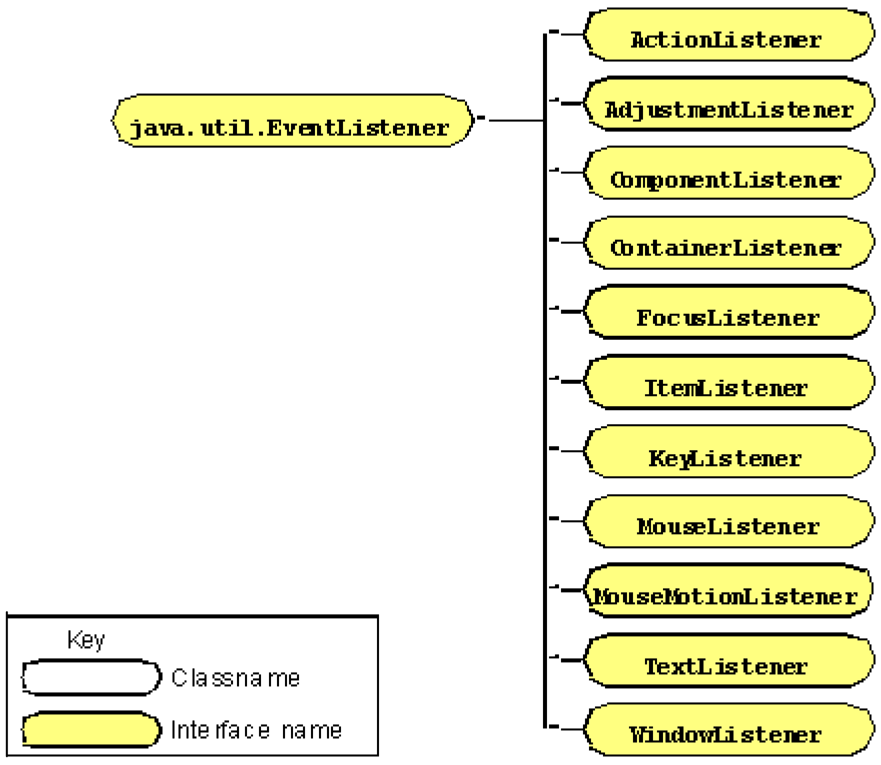
Note: We do not create Mouse/Keys etc as they are system components

Step 2: Event Handlers/ Event Listener

The second step is that you build components that can handle events

First Technique - *By Implementing Listener Interfaces*

- Java defines interfaces for every event type
- If a class needs to handle an event. It needs to implement the corresponding listener interface
- To handle "ActionEvent" a class needs to implement "ActionListener"
- To handle "KeyEvent" a class needs to implement "KeyListener"
- To handle "MouseEvent" a class needs to implement "MouseListener" and so on
- Package java.awt.event contains different event Listener Interfaces which are shown in the following figure



- Some Example Listeners, the way they are defined in JDK by Sun

```
    public interface ActionListener {
      public void actionPerformed(ActionEvent e);
  }
```

```
    public interface ItemListener {
      public void itemStateChanged(ItemEvent e);
  }
```

```
    public interface ComponentListener {
    public void componentHidden(ComponentEvent e);
    public void componentMoved(ComponentEvent e);
    public void componentResized(ComponentEvent e);
    public void componentShown(ComponentEvent e);
  }
```

- By implementing an interface the class agrees to implement all the methods that are present in that interface. Implementing an interface is like *signing a contract*.

- Inside the method the class can do what ever it wants to do with that event

- Event Generator and Event Handler can be the same or different classes

- To handle events generated by Button. A class needs to implement ActionListener interface and thus needs to provide the definition of actionPerformed() method which is present in this interface.

```
public class Test implements ActionListener{
  public void actionPerformed(ActionEvent ae) {
    // do something
  }
}
```

Step 3: Registering Handler with Generator

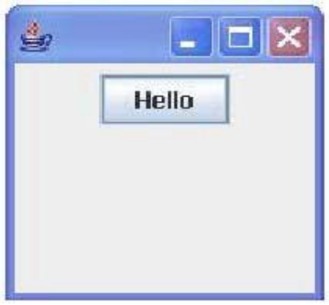
The event generator is told about the object which can handle its events

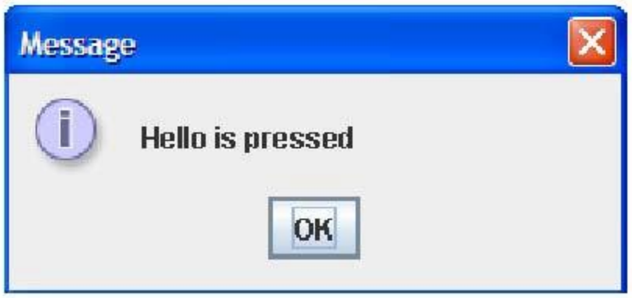
Event Generators have a method
— addXXXListener(_reference to the object of Handler class_)

For example, if b1 is JButton then
— b1.addActionListener(this); // if listener and generator are same class

Event Handling Example

Clicking the "Hello" button will open up a message dialog shown below.





Hello is pressed

We will take the simplest approach of creating handler and generator in a single class. Button is our event generator and to handle that event our class needs to implement ActionListener Interface and to override its actionPerformed method and in last to do the registration

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

    /* Implementing the interface according to the type of the event, i.e. creating event handler (first
    part of step 2 of our process) */

public class ActionEventTest implements ActionListner{

  JFrame frame;
  JButton hello;

     // setting layout components
  public void initGUI ( ) {

     frame = new JFrame();
     Container cont =  frame.getContentPane();
      cont.setLayout(new FlowLayout());

       //Creating event generator step-1 of our process
           hello = new JButton("Hello");
```

```
        /* Registering event handler with event generator.
        Since event handler is in same object that contains
        button, we have used this to pass the reference.(step
        3 of the process) */
            hello.addActionListener(this);
    cont.add(hello);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(150, 150);
    frame.setVisible(true);
 }

    //constructor
 public ActionEventTest ( ) {
     initGUI();
 }


    /* Override actionPerformed method of ActionListener's
       interfacemethod of which will be called when event
       takes place (second part of step 2 of our process) */

public void actionPerformed(ActionEvent event) {
  JOptionPane.showMessageDialog(null,"Hello is pressed");
}


public static void main(String args[]) {
   ActionEventTest aeTest = new ActionEventTest();
}

} // end class
```

How Event Handling Participants interact Behind the Scenes?

We have already seen that what a programmer needs to do handle events. Let's see what takes place behind the scenes, i.e How JVM handles event. Before doing that lets revisit different participants of Event Handling Process and briefly what they do.

1. **Event Generator / Source**

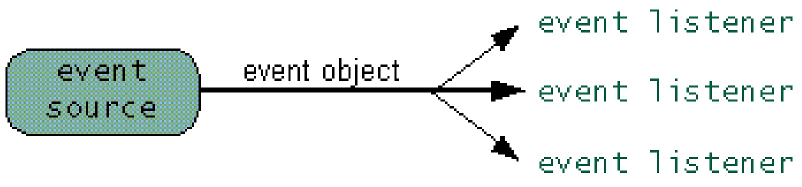
   – Swing and awt components
   – For example, JButton, JTextField, JFrame etc
   – Generates an event object
   – Registers listeners with itself

2. **Event Object**

   – Encapsulate information about event that occurred and the source of that event
   – For example, if you click a button, ActionEvent object is created

3. **Event Listener/handler**

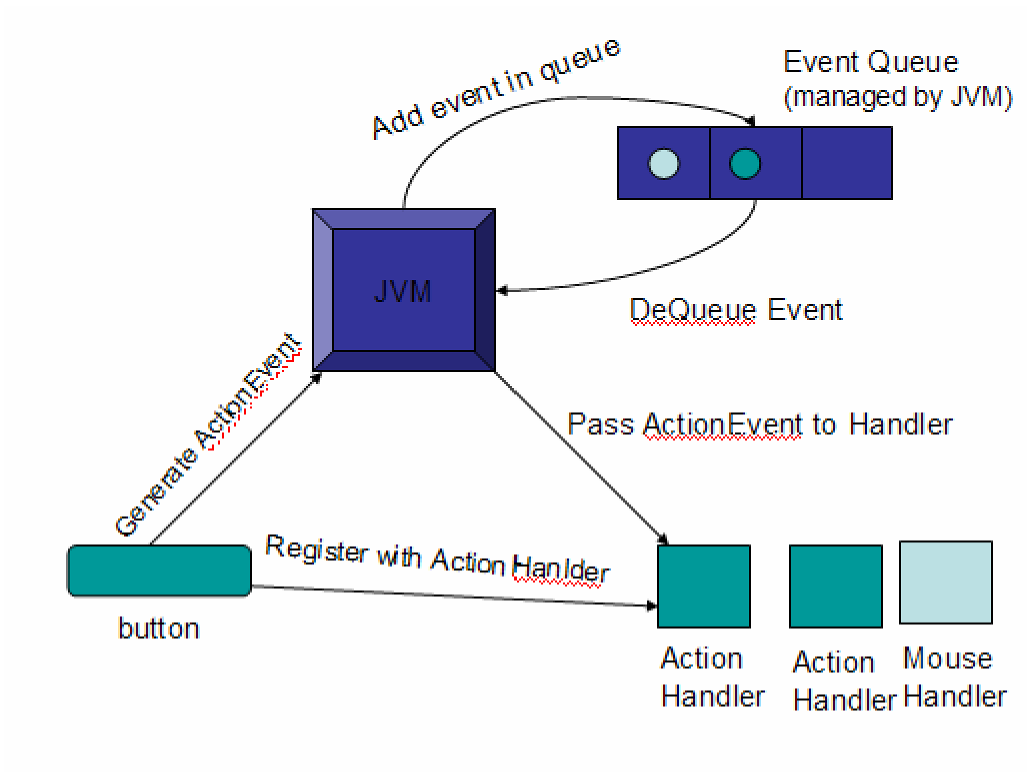
   – Receives event objects when notified, then responds
   – Each event source can have multiple listeners registered on it
   – Conversely, a single listener can register with multiple event sources



4. **JVM**

   – Receives an event whenever one is generated
   – Looks for the listener/handler of that event
   – If exist, delegate it for processing
   – If not, discard it (event).

When button generates an ActionEvent it is sent to JVM which puts it in an event queue. After that when JVM find it appropriate it de-queue the event object and send it to all the listeners that are registered with that button. This is all what we shown in the pictorial form below:



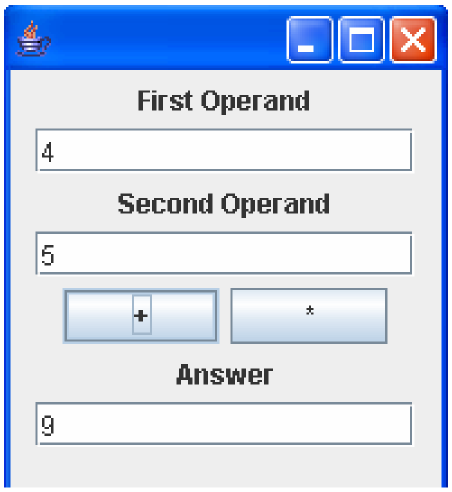
(figure from JAVA A Lab Course)

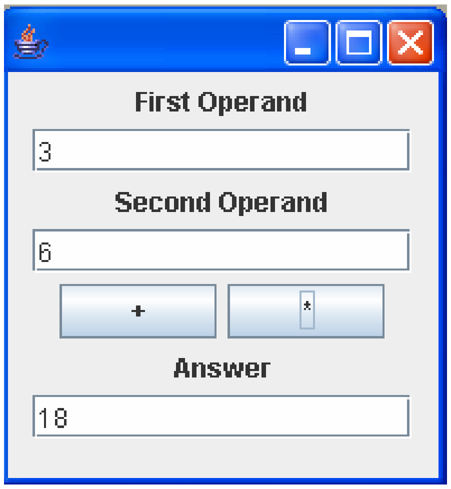
Making Small Calculator

User enters numbers in the provided fields

On pressing "+" button, sum would be displayed in the answer field

On pressing "*" button, product would be displayed in the answer field





Example Code: Making Small Calculator

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class SmallCalcApp implements ActionListener{

 JFrame frame;
 JLabel firstOperand, secondOperand, answer;
 JTextField op1, op2, ans;
 JButton plus, mul;

 // setting layout
  public void initGUI ( ) {
   frame = new JFrame();

   firstOperand  = new JLabel("First Operand");
   secondOperand = new JLabel("Second Operand");
   answer        = new JLabel("Answer");

   op1 = new JTextField (15);
   op2 = new JTextField (15);
   ans = new JTextField (15);

   plus = new JButton("+");
   plus.setPreferredSize(new   Dimension(70,25));

   mul = new JButton("*");
   mul.setPreferredSize(new   Dimension(70,25));

   Container cont =  frame.getContentPane();
   cont.setLayout(new FlowLayout());

   cont.add(firstOperand);
   cont.add(op1);

   cont.add(secondOperand);
   cont.add(op2);

   cont.add(plus);
   cont.add(mul);

   cont.add(answer);
  cont.add(ans);

         plus.addActionListener(this);
   mul.addActionListener(this);

   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setSize(200, 220);
   frame.setVisible(true);

  }

         //constructor
    public SmallCalcApp ( ) {
   initGUI();
}

public void actionPerformed(ActionEvent event) {
  String oper, result;
  int num1, num2, res;

       /* All the information regarding an event is contained
         inside the event object. Here we are calling the
         getSource() method on the event object to figure out
         the button that has generated that event.   */

   if (event.getSource() == plus) {
     oper = op1.getText();
     num1 = Integer.parseInt(oper);
     oper = op2.getText();
     num2 = Integer.parseInt (oper);
     res = num1+num2;

     result = res+"";
     ans.setText(result);
  }
   else if (event.getSource() == mul) {
     oper = op1.getText();
     num1 = Integer.parseInt(oper);
     oper = op2.getText();
     num2 = Integer.parseInt (oper);
     res = num1*num2;

     result = res+"";
     ans.setText(result);
  }


  public static void main(String args[]) {
     SmallCalcApp scApp = new SmallCalcApp();
  }
}// end class
```

Lesson 12

More Examples of Handling Events

Handling Mouse Event

Mouse events can be trapped for any GUI component that inherits from Component class. For example, JPanel, JFrame & JButton etc.

To handle Mouse events, two types of listener interfaces are available.

- MouseMotionListener
- MouseListener

The class that wants to handle mouse event needs to implement the corresponding interface and needs to provide the definition of all the methods in that interface.

MouseMotionListener interface

- Used for processing mouse motion events
- Mouse motion event is generated when mouse is moved or dragged

MouseMotionListener interfaces is defined in JDK as follows

```
public interface MouseMotionListener {
  public void mouseDragged (MouseEvent me);
  public void mouseMoved (MouseEvent me);
}
```

MouseListener interface

- Used for processing “interesting” mouse events like when mouse is:
  - Pressed
  - Released
  - Clicked (pressed & released without moving the cursor)
  - Enter (mouse cursor enters the bounds of component)
  - Exit (mouse cursor leaves the bounds of component)

MouseListener interfaces is defined in JDK as follows

```
public interface MouseListener {
  public void mousePressed (MouseEvent me);
  public void mouseClicked (MouseEvent me);
  public void mouseReleased (MouseEvent me);
  public void mouseEntered (MouseEvent me);
  public void mouseExited (MouseEvent me);
}
```

Example Code: Handling Mouse Events

Example to show Mouse Event Handling .Every time mouse is moved, the coordinates for a new place is shown in a label.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class EventsEx implements MouseMotionListener{

  JFrame frame;

  JLabel coordinates;

  // setting layout
  public void initGUI ( ) {

    // creating event generator
      frame = new JFrame();

      Container cont =  frame.getContentPane();
      cont.setLayout(new BorderLayout( ) );

      coordinates = new JLabel ();
      cont.add(coordinates,    BorderLayout.NORTH);

      // registring mouse event handler with generator
      frame.addMouseMotionListener(this);

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.setSize(350, 350);
      frame.setVisible(true);

    } // end initGUI method

    //default constructor
    public EventsEx ( ) {
      initGUI();
    }
  // MouseMotionListener event hadler handling dragging
    public void mouseDragged (MouseEvent me) {

      int x = me.getX();
      int y = me.getY();

      coordinates.setText("Dragged at [" + x + "," + y + "]");

    }

  // MouseMotionListener event handler handling motion
    public void mouseMoved (MouseEvent me) {

      int x = me.getX();
      int y = me.getY();

      coordinates.setText("Moved at [" + x + "," + y + "]");

    }

    public static void main(String args[]) {

      EventsEx ex = new EventsEx();

    }

    } // end class
```

Another Example: Handling Window Events

Task

We want to handle Window Exit event only

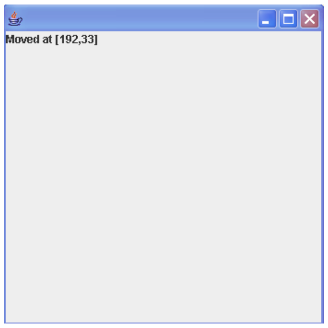
Why?

When window is closed, control should return back to command prompt.

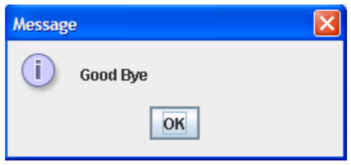
But we have already achieved this functionality through following line of code

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

But, what if we want to display some message (*Good Bye*) before exiting?


Moved at [192,33]

When user closes the window, Message would be displayed


Message

Good Bye

After pressing Ok button program will exit

How?

To handle window events, we need to implement "WindowListner" interface.

"WindowListner" interface contains 7 methods We require only one i.e. windowClosing

But, We have to provide definitions of all methods to make our class a concrete class

WindowListener interface is defined in the JDK as follows

```
        public interface WindowListener {
        public void windowActivated(WindowEvent we);
        public void windowClosed(WindowEvent we);
        public void windowClosing(WindowEvent we);
        public void windowDeactivated(WindowEvent we);
        public void windowDeiconified(WindowEvent we);
        public void windowIconified(WindowEvent we);
        public void windowOpened(WindowEvent we);
    }
```

public void windowClosing(WindowEvent we is our required method

Example Code: WindowExitHandler

This example code is modification of the last code example i.e. EventsEx.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class EventsEx implements MouseMotionListener ,
                                                   WindowListener {
  JFrame frame;

  JLabel coordinates;

// setting layout
  public void initGUI ( ) {

 // creating event generator
    frame = new JFrame();

    Container cont =  frame.getContentPane();
    cont.setLayout(new BorderLayout( ) );

    coordinates = new JLabel ();
    cont.add(coordinates,    BorderLayout.NORTH);

 // registring mouse event handler with generator
    frame.addMouseMotionListener(this);

 // registering window handler with generator
    frame.addWindowListener(this);

    frame.setSize(350, 350);
    frame.setVisible(true);

  } // end initGUI method

  //default constructor
  public EventsEx ( ) {
   initGUI();
  }

// MouseMotionListener event hadler handling dragging
public void mouseDragged (MouseEvent me) {

  int x = me.getX();
  int y = me.getY();

  coordinates.setText("Dragged at [" + x + "," + y + "]");

}
// MouseMotionListener event handler handling motion
   public void mouseMoved (MouseEvent me) {

     int x = me.getX();
     int y = me.getY();

     coordinates.setText("Moved at [" + x + "," + y + "]");
   }

  // window listener event handler
   public void windowActivated (WindowEvent we) {    }
   public void windowClosed (WindowEvent we) {  }
   public void windowClosing (WindowEvent we) {
     JOptionPane.showMessageDialog(null, "Good Bye");
     System.exit(0);
   }
   public void windowDeactivated (WindowEvent we) {   }
   public void windowDeiconified (WindowEvent we) {   }
   public void windowIconified (WindowEvent we) {   }
   public void windowOpened (WindowEvent we) {   }

   public static void main(String args[]) {
     EventsEx ex = new EventsEx();
   }
  } // end class
```

Problem in Last Code Example

Problem

– We were interested in *windowClosing()* method only

– But have to provide definitions of all the methods, Why?

– Because a class implementing an interface has to provide definitions of all methods present in that interface.

Solution

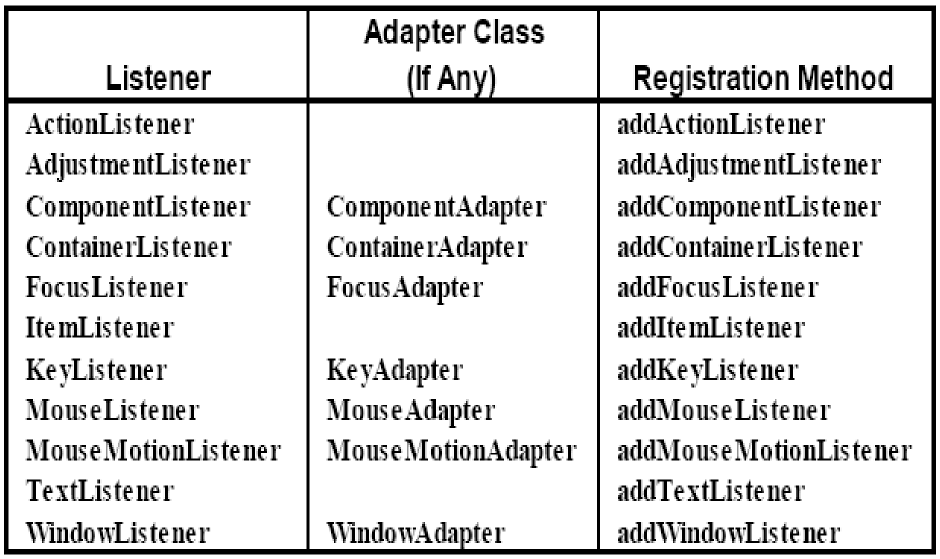
– To avoid giving implementations of all methods of an interface when we are not using these methods we use Event *Adapter* classes

Adapter Classes

- For listener interfaces containing more than one event handling methods, jdk defines adapter classes. Examples are
  - For WindowListener Æ WindowAdapter
  - For MouseMotionListener Æ MouseMotionAdapter
  - and many more
- Adapter classes provide definitions for all the methods (empty bodies) of their corresponding Listener interface
- It means that WindowAdapter class implements WindowListener interface and provide the definition of all methods inside that Listener interface
- Consider the following example of MouseMotionAdapter and its corresponding MouseMotionListener interface

```
  public interface MouseMotionListener {
      public void mouseDragged (MouseEvent me);
      public void mouseMoved (MouseEvent me);
  }


 public class MouseMotionAdapter implements MouseMotionListener{
      public void mouseDragged (MouseEvent me) { }
      public void mouseMoved (MouseEvent me)  { }
 }
```

Available Adapter classes



Listener	Adapter Class (If Any)	Registration Method
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

How to use Adapter Classes

Previously handler class need to implement interface

public class EventsEx **implements MouseMotionListener**{...}

Therefore it has to provide definitions of all the methods inside that interface

Now our handler class will inherit from adapter class

public class EventsEx **extends MouseMotionAdapter**{...}

Due to inheritance, all the methods of the adapter class will be available inside our handler class

Since adapter classes has already provided definitions with empty bodies.

We do not have to provide implementations of all the methods again

We only need to override our method of interest.

Example Code 13.1: Handling Window Events using Adapter Classes

Here we are modifying the window event code in the last example to show the use of WindowAdapter instead of WindowListener. Code related to MouseMotionListener is deleted to avoid cluttering of code.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class EventsEx extends WindowAdapter {

  JFrame frame;

  JLabel coordinates;

 // setting layout
  public void initGUI () {

  // creating event generator
    frame = new JFrame();

    Container cont =  frame.getContentPane();
    cont.setLayout(new BorderLayout() );

    coordinates = new JLabel ();
    cont.add(coordinates,    BorderLayout.NORTH);

  // registering window handler with generator
    frame.addWindowListener(this);

    frame.setSize(350, 350);
    frame.setVisible(true);

  } // end initGUI method
 //default constructor
  public EventsEx () {
  initGUI();
  }
// As you can see that we have only implemented
// our required method
 public void windowClosing (WindowEvent we) {
  JOptionPane.showMessageDialog(null, "Good Bye");
  System.exit(0);
 }

 public static void main(String args[]) {
  EventsEx ex = new EventsEx();
 }
} // end class
```

Problem in Last Code Example

We have inherited from WindowAdapter

What if we want to use MouseMotionAdpater as well ? or what if our class already inherited form some other class ?

Problem

— Java allows single inheritance

Solution

— Use *Inner* classes

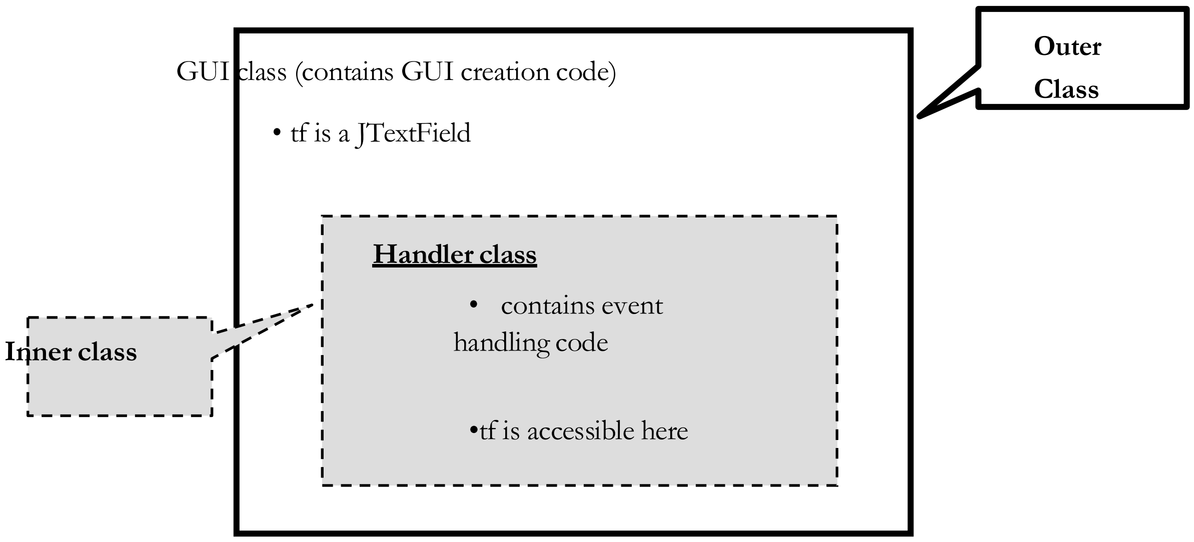
Inner Classes

A class defined inside another class

Inner class can access the instance variables and members of outer class

It can have constructors, instance variables and methods, just like a regular class

Generally used as a private utility class which does not need to be seen by others classes



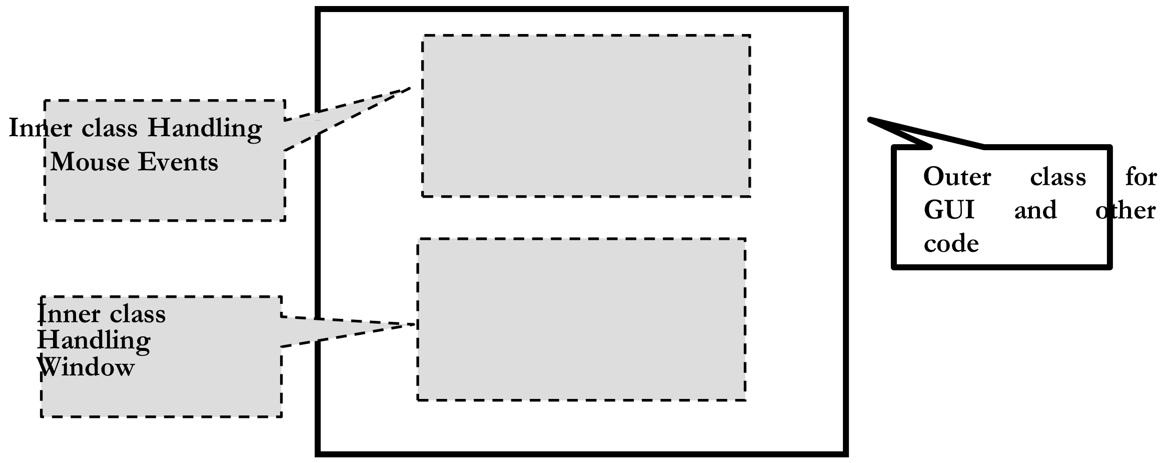
Example Code13.2: Handling Window Event with Inner Class

Here we are modifying the window event code in the last example to show the use of WindowAdapter as an inner class.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class EventEx {

  JFrame frame;
  JLabel coordinates;

  // setting layout
  public void initGUI ( ) {

    frame = new JFrame();
  Container cont = frame.getContentPane();
  cont.setLayout(new BorderLayout( ));

  coordinates = new JLabel ();
  cont.add(coordinates,   BorderLayout.NORTH);

      /* Creating an object of the class which is handling our
        window events and registering it with generator */

        WindowHandler handler = new Window Handler ();
  frame.addWindowListener(handler);

  frame.setSize(350, 350);
  frame.setVisible(true);
} // end initGUI

//default constructor
public EventEx ( ) {
   initGUI();
}

  /* Inner class implementation of window adapter. Outer
    class is free to inherit from any other class. */

  private class WindowHandler extends WindowAdapter {
  // Event Handler for WindowListener
   public void windowClosing (WindowEvent we) {
     JOptionPane.showMessageDialog(null, “Good Bye”);
     System.exit(0)
   }
} // end of WindowHandler class

public static void main(String args[]) {
   EventEx e = new EventEx();
}
} // end class
```

Example Code 13.3: Handling Window and Mouse Events with Inner Class

Here we are modifying the window event code of the last example to handle window and mouse events using inner classes. The diagram given below summarizes the approach.


Inner class Handling Mouse Events
Inner class Handling Window
Outer class for GUI and other code

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class EventEx {
  JFrame frame;
  JLabel coordinates;
  // setting layout
  public void initGUI ( ) {
     frame = new JFrame();
  Container cont =  frame.getContentPane();
  cont.setLayout(new BorderLayout( ) );
  coordinates = new JLabel ();
  cont.add(coordinates,    BorderLayout.NORTH);

   /* Creating an object of the class which is handling our
     window events and registering it with generator */
 WindowHandler whandler = new WindowHandler ();
 frame.addWindowListener(whandler);

/* Creating an object of the class which is handling our
      MouseMotion events & registering it with generator */
  MouseHandler mhandler = new MouseHandler ();
  frame.addMouseMotionListener(mhandler);
  frame.setSize(350, 350);
  frame.setVisible(true);
}
```

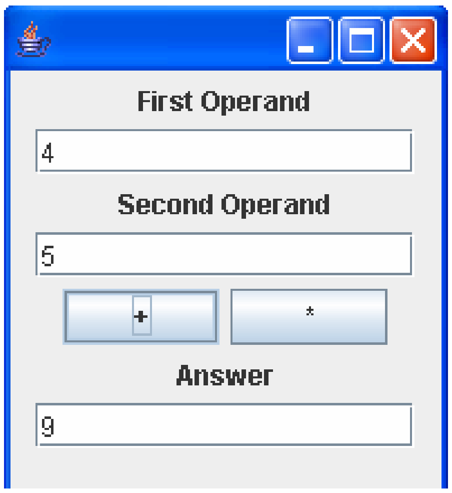
```
   //default constructor
    public EventEx ( ) {
      initGUI();
    }

     /* Inner class implementation of WindowAdapter. Outer class
        is free to inherit from any other class. */

   private class WindowHandler extends WindowAdapter {

    // Event Handler for WindowListener
     public void windowClosing (WindowEvent we) {

         JOptionPane.showMessageDialog(null, “Good Bye”);
         System.exit(0)
     }

    } // end of WindowHandler


    //Inner class implementation of MouseMotionAdapter
   private class MouseHandler extends MouseMotionAdapter {

       // Event Handler for mouse motion events
     public void mouseMoved (MouseEvent me) {
        int x = me.getX();
        int y = me.getY();

        coord.setText(“Moved at [" + x + "," + y + "]” );
     }
   } // end of MouseHandler

    public static void main(String args[]) {
      EventEx e = new EventEx();
    }

} // end class
```

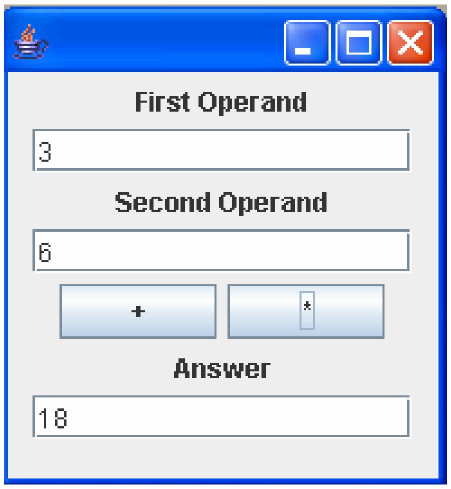
Example Code: Making Small Calculator using Inner classes

User enters numbers in the provided fields

On pressing "+" button, sum would be displayed in the answer field

On pressing "*" button, product would be displayed in the answer field





```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class SmallCalcApp implements ActionListener{

 JFrame frame;
 JLabel firstOperand, secondOperand, answer;
 JTextField op1, op2, ans;
 JButton plus, mul;

 // setting layout
 public void initGUI ( ) {
   frame = new JFrame();

   firstOperand  = new JLabel("First Operand");
   secondOperand = new JLabel("Second Operand");
   answer        = new JLabel("Answer");

   op1 = new JTextField (15);
   op2 = new JTextField (15);
   ans = new JTextField (15);

   plus = new JButton("+");
   plus.setPreferredSize(new    Dimension(70,25));

   mul = new JButton("*");
   mul.setPreferredSize(new    Dimension(70,25));

   Container cont =  frame.getContentPane();
   cont.setLayout(new FlowLayout());

   cont.add(firstOperand);
   cont.add(op1);

   cont.add(secondOperand);
   cont.add(op2);

   cont.add(plus);
   cont.add(mul);

   cont.add(answer);
   cont.add(ans);

  /* Creating an object of the class which is handling
    button events & registering it with generators */
   ButtonHandler bHandler = new ButtonHandler();
         plus.addActionListener(bHandler);
   mul.addActionListener(bHandler);

   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setSize(200, 220);
   frame.setVisible(true);

 }
//constructor
   public SmallCalcApp ( ) {
 initGUI();
}

 //Inner class implementation of ActionListener
private class ButtonHandler implements ActionListener{
  public void actionPerformed(ActionEvent event) {

    String oper, result;
    int num1, num2, res;

    if (event.getSource() == plus) {
      oper = op1.getText();
      num1 = Integer.parseInt(oper);

      oper = op2.getText();
      num2 = Integer.parseInt (oper);

        res = num1+num2;

        result = res+"";
        ans.setText(result);
    }

    else if (event.getSource() == mul) {
      oper = op1.getText();
      num1 = Integer.parseInt(oper);

      oper = op2.getText();
      num2 = Integer.parseInt (oper);

        res = num1*num2;

        result = res+"";
        ans.setText(result);
    }
 } // end actionPerformed method
} // end inner class ButtonHandler

  public static void main(String args[]) {
    SmallCalcApp scApp = new SmallCalcApp();
  }

}// end class
```

Anonymous Inner Classes

Has no name

Same as inner class in capabilities

much shorter

difficult to understand

Named vs. Anonymous Objects

Named

– String s = “hello”; System.out.println(s);

– “hello” has a named reference *s*.

Anonymous

– System.out.println(“hello”);

We generally use anonymous object when there is just a one time use of a particular object but in case of a repeated use we generally used named objects and use that named reference to use that objects again and again.

Example Code 13.4 Handling Window Event with Anonymous Inner Class

Here we are modifying the window event code of 13.3 to show the use of anonymous inner class.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class EventsEx extends WindowAdapter {

 JFrame frame;
 JLabel coordinates;

// setting layout
public void initGUI ( ) {

// creating event generator
  frame = new JFrame();

  Container cont =  frame.getContentPane();
  cont.setLayout(new BorderLayout( ) );

  coordinates = new JLabel ();
  cont.add(coordinates,    BorderLayout.NORTH);

  // registering event handler (anonymous inner class)
  // with generator by using

 frame.addWindowListener (
   new WindowAdapter ( ) {
     public void windowClosing (WindowEvent we) {
       JOptionPane.showMessageDialog(null, "Good  Bye");
       System.exit(0);

     } // end window closing

   } // end WindowAdapter
 ); // end of addWindowListener

 frame.setSize(350, 350);
 frame.setVisible(true);
} // end initGUI method

//default constructor
public EventsEx ( ) {
 initGUI();
}

public static void main(String args[]) {
  EventsEx ex = new EventsEx();
}

} // end class
```

Summary of Approaches for Handling Events

1. **By implementing Interfaces**
2. **By extending from Adapter classes**

To implement the above two techniques we can use

Same class

- putting event handler & generator in one class

Separate class

1. Outer class
    - Putting event handlers & generator in two different classes

3. Inner classes

3. Anonymous Inner classes

References

Java A Lab Course by Umair Javed

Lesson 14

Java Database Connectivity

Introduction

Java Database Connectivity (JDBC) provides a standard library for accessing databases. The JDBC API contains number of interfaces and classes that are extensively helpful while communicating with a database.

The java.sql package

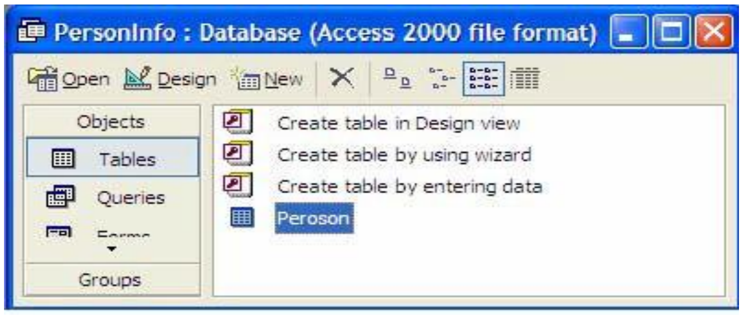
The java.sql package contains basic & most of the interfaces and classes. You automatically get this package when you download the J2SE™. You have to import this package whenever you want to interact with a relational database.

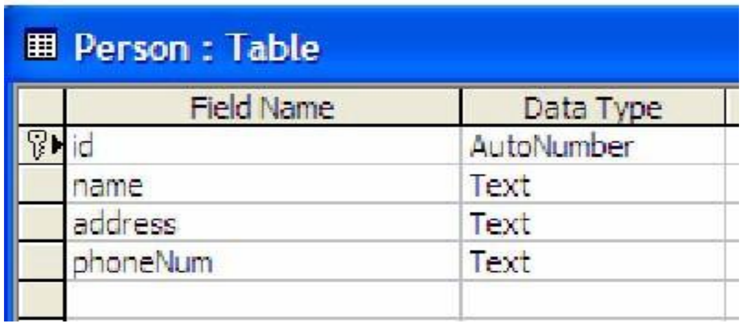
Connecting With Microsoft Access

In this handout, we will learn how to connect & communicate with Microsoft Access Database. We chooses Access because most of you are familiar with it and if not than it is very easy to learn.

Create Database

In start create a database "PersonInfo" using Microsoft Access. Create one table named "Person". The schema of the table is shown in the picture.

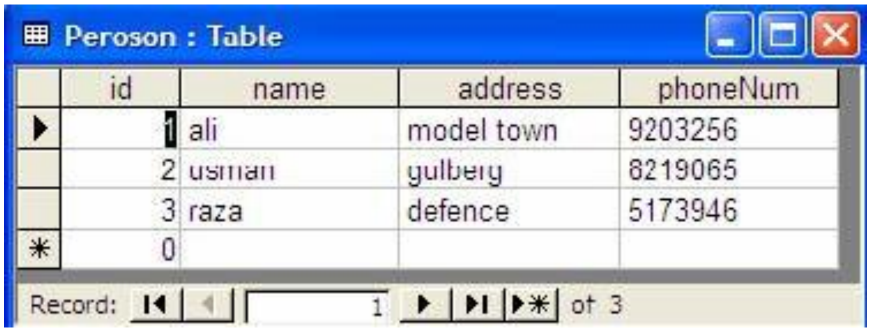

PersonInfo : Database (Access 2000 file format)
Open
Design
New
Objects
Tables
Queries
Forms
Groups
Create table in Design view
Create table by using wizard
Create table by entering data
Peroson



Person : Table

Field Name	Data Type
id	AutoNumber
name	Text
address	Text
phoneNum	Text

Add the following records into Person table as shown below.

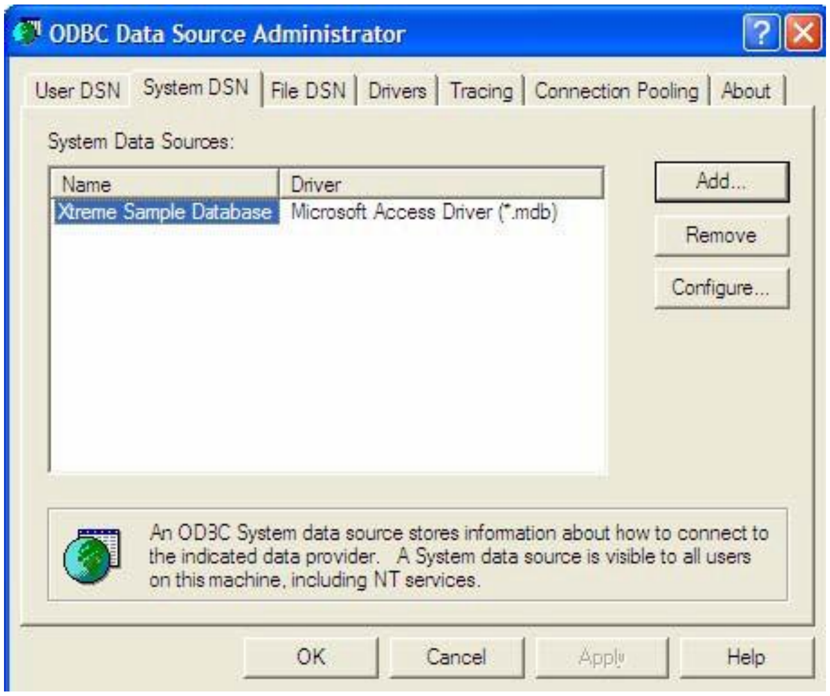


id	name	address	phoneNum
1	ali	model town	9203256
2	usman	gulberg	8219065
3	raza	defence	5173946
0			

Save the data base in some folder. (Your database will be saved as an .mdb file)

Setup System DSN

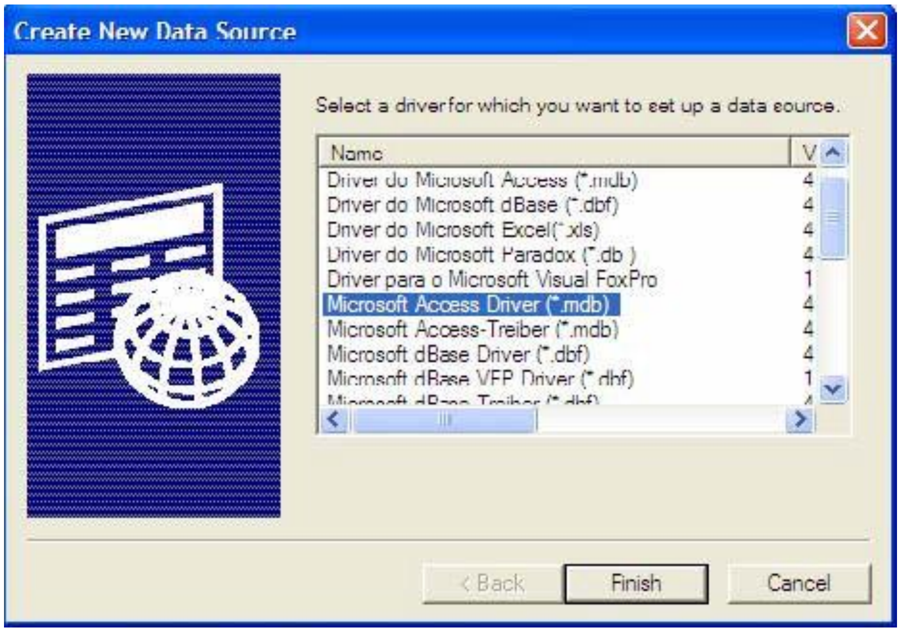
- After creating database, you have to setup a system Data Source Name (DSN). DSN is a name through which your system recognizes the underlying data source.
- Select Start Settings Control Panel Administrative Tools Data Sources (ODBC).
- The ODBC Data Source Administrator window would be opened as shown below. Select System DSN tab. (If you are unable to use System DSN tab due to security restrictions on your machine, you can use the User DSN tab)



Name	Driver
Xtreme Sample Database	Microsoft Access Driver (*.mdb)

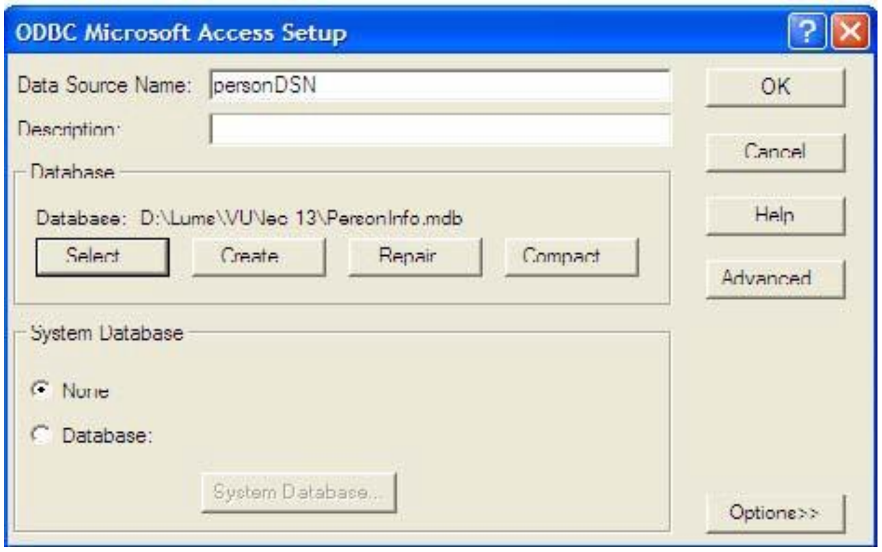
An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

- Press Add… button and choose Microsoft Access Driver (*.mdb) from Create New Data Source window and press Finish button as shown in diagram.
- After that, ODBC Microsoft Access Setup window would be opened as shown in following diagram
- Enter the Data Source Name personDSN and select the database by pressing Select button. The browsing window would be opened, select the desired folder that contains the database (The database .mdb file you have created in the first step) Press Ok button.



Select a driver for which you want to set up a data source.

Name	V
Driver do Microsoft Access (*.mdb)	4
Driver do Microsoft dBase (*.dbf)	4
Driver do Microsoft Excel(*.xls)	4
Driver do Microsoft Paradox (*.db)	4
Driver para o Microsoft Visual FoxPro	1
Microsoft Access Driver (*.mdb)	4
Microsoft Access-Treiber (*.mdb)	4
Microsoft dBase Driver (*.dbf)	4
Microsoft dBase VFP Driver (*.dbf)	1



Data Source Name: personDSN
Description:
Database
Database: D:\Lume\VU\lec 13\PersonInfo.mdb
System Database
[✓ None]
Database:

Basic Steps in Using JDBC

There are eight (8) basic steps that must be followed in order to successfully communicate with a database. Let's take a detail overview of all these one by one.

1. Import Required Package

- Import the package java.sql.* that contains useful classes and interfaces to access & work with database.

```
import java.sql.*;
```

2. **Load Driver**

- Need to load suitable driver for underlying database.
- Different drivers & types for different databases are available.
- For MS Access, load following driver available with j2se.

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

- For Oracle, load the following driver. You have to download it explicitly.

Class.forName("oracle.jdbc.driver.OracleDriver");

3. **Define Connection URL**

- To get a connection, we need to specify the URL of a database (Actually we need to specify the address of the database which is in the form of URL)
- As we are using Microsoft Access database and we have loaded a JDBC-ODBC driver. Using JDBC-ODBC driver requires a DSN which we have created earlier and named it personDSN. So the URL of the database will be

String conURL = "jdbc:odbc:personDSN";

4. **Establish Connection With DataBase**

- Use DriverManagerto get the connection object.
- The URL of the database is passed to the getConnection method. Connection con = DriverManager.getConnection(conURL);
- If DataBase requires username & password, you can use the overloaded version of getConnection method as shown below:

String usr = "umair";
String pwd = "vu";

Connection con = null;con = DriverManager.getConnection(conURL, usr, pwd);

5. **Create Statement**

- A Statement object is obtained from a Connection object.

Statement stmt = con.createStatement( );

Once you have a statement, you can use it for various kinds of SQL queries.

6. **Execute a Query**

- The next step is to pass the SQL statements & to execute them.
- Two methods are generally used for executing SQL queries. These are:

executeQuery(sql) method

- Used for SQL SELECT queries.
- Returns the ResultSET object that contains the results of the query and can be used to access the query results.

```
String sql = "SELECT * from sometable";ResultSet rs =
stmt.executeQuery(sql);
```

executeUpdate(sql)method

. This method is used for executing an update statement like INSERT, UPDATE or

7. **DELETE**

- Returns an Integer value representing the number of rows updated

```
String sql = "INSERT INTO tablename " + "(columnNames) Values (values)" ;

int count = stmt.executeUpdate(sql);
```

Process Results of the Query

- The ResultSet provides various getXXX methods that takes a column index or name and returns the data
- The ResultSet maintains the data in the form tables (rows & columns)
- First row has index 1, not 0.
- The next method of ResultSet returns true or false depending upon whether the next row is available (exist) or not and moves the cursor
- Always remember to call next() method at-least once
- To retrieve the data of the column of the current row you need to use the various getters provided by the ResultSet.
- For example, the following code snippet will iterate over the whole ResultSet and illustrates the usage of getters methods

```
while ( rs.next() ){
   //by using column name
   String name = rs.getString("columnName");
  // or by using column indexString name =
  rs.getString(1); }
```

8. **Close the Connection**

- An opening connection is expensive, postpone this step if additional database operations are expected

```
con.close();
```

__**Example Code 14.1: Retrieving Data from ResultSet**__

The JdbcEx.java demonstrates the usage of all above explained steps. In this code example, we connect with the PersonInfo database, the one we have created earlier, and then execute the simple SQL SELECT query on Person table, and then process the query results.

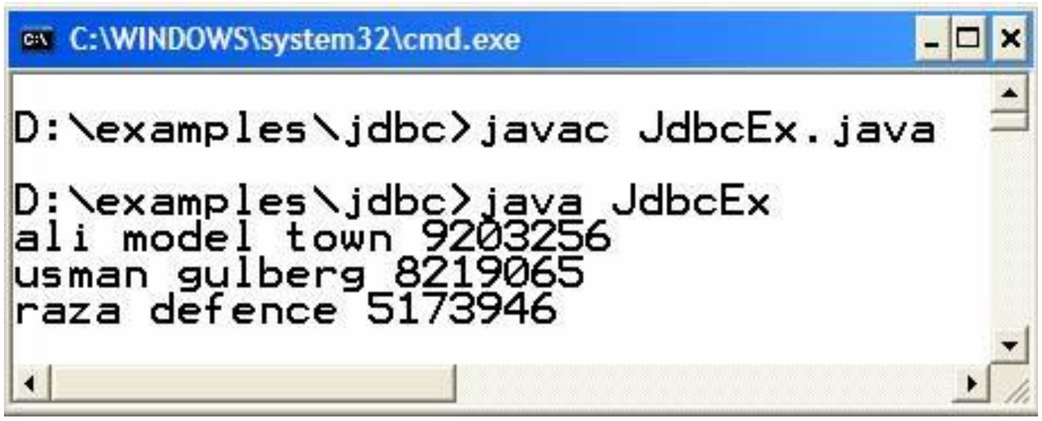
```
// File JdbcEx.java

//step 1: import packageimport java.sql.*;

public class JdbcEx {

    public static void main (String args[ ]) {

      try {

          //Step 2: load driverClass.forName("sun.jdbc.odbc.JdbcOdbcDriver");

          //Step 3: define the connection URL
          String url = "jdbc:odbc:personDSN";

          //Step 4: establish the connection
          Connection con = DriverManager.getConnection(url);

          //Step 5: create Statement
          Statement st = con.createStatement();


          //Step 6: preapare & execute the query
          String sql = "SELECT * FROM Person";

          ResultSet rs = st.executeQuery(sql);

          //Step 7: process the results
          while(rs.next()){


              // The row name is "name" in database "PersonInfo,// hence specified in the getString()
              method.

              String name = rs.getString("name");String add = rs.getString("address");String
              pNum = rs.getString("phoneNum");

              System.out.println(name + " " + add + " " + pNum);}

          //Step 8: close the connection
          con.close();


        }catch(Exception sqlEx){
            System.out.println(sqlEx);
            }

    } // end main} // end class
```

The important thing you must notice that we have put all code inside try block and then handle (in the above example, only printing the name of the exception raised) exception inside catch block.

Why? Because we are dealing with an external resource (database). If you can recall all IO related operations involving external resources in java throw exceptions. These exceptions are checked exceptions and we must need to handle these exceptions.

Compile & Execute

Since the Person table contains only three records, so the following output would be produced on executing the above program.



```
D:\examples\jdbc>javac JdbcEx.java

D:\examples\jdbc>java JdbcEx
ali model town 9203256
usman gulberg 8219065
raza defence 5173946
```

References:

. Java – A Lab Course by Umair Javed
. Java tutorial by Sun: http://java.sun.com/docs/books/turorial
. Beginning Java2 by Ivor Hortan

Lesson 15

More on JDBC

In the previous handout, we have discussed how to execute SQL statements. In this handout, we'll learn how to execute DML (insert, update, delete) statements as well some useful methods provided by the JDBC API.

Before jumping on to example, lets take a brief overview of executeUpdate()method that is used for executing DML statements.

Useful Statement Methods:

o **executeUpdate( )**

- Used to execute for INSERT, UPDATE, or DELETE SQL statements.
- This method returns the number of rows that were affected in the database.
- Also supports DDL (Data Definition Language) statements CREATE TABLE, DROP TABLE, and ALERT TABLE etc. For example,

  int num = stmt.executeUpdate("DELETE from Person WHERE id = 2");

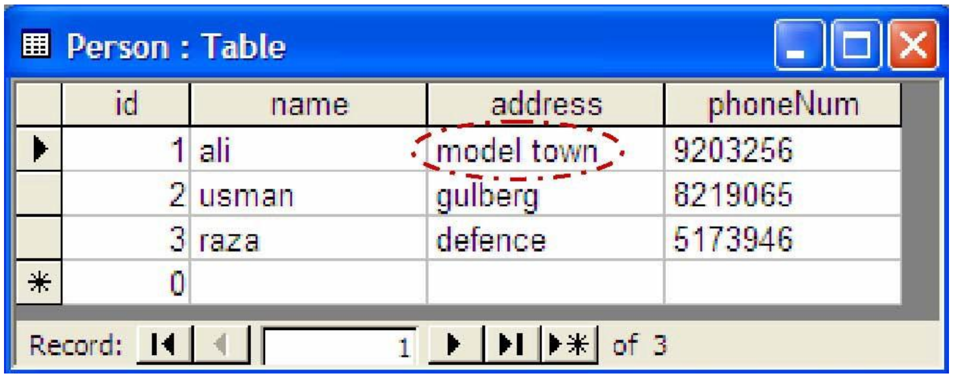
Example Code 15.1: Executing SQL DML Statements

This program will take two command line arguments that are used to update records in the database. executeUpdate( ) method will be used to achieve the purpose stated above.

```
// File JdbcDmlEx.java

  //step 1: import packageimport java.sql.*; public class
  JdbcDmlEx {public static void main (String args[ ]) { try {

        //Step 2: load driverClass.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        //Step 3: define the connection URL
        String url = "jdbc:odbc:personDSN";


        //Step 4: establish the connection
        Connection con = DriverManager.getConnection(url);


        //Step 5: create Statement
        Statement st = con.createStatement();

        // assigning first command line argument value
        String addVar = args[0];

        // assigning second command line argument value
        String nameVar = args[1];

        // preparing query – nameVar & addVar strings are embedded
        // into query within '" + string + "'
        String sql = "UPDATE Person SET address = '"+addVar+"'" +

                              " WHERE name = '"+nameVar+"' ";

        // executing query
        int num = st.executeUpdate(sql);

        // Step 7: process the results of the query
        // printing number of records affected

        System.out.println(num + " records updated");

        //Step 8: close the connection
        con.close();


      }catch(Exception sqlEx){
          System.out.println(sqlEx);
          }


   } // end main} // end class
```

Compile & Execute

The Person table is shown in the following diagram before execution of the program. We want to update first row i.e address of the person ali.

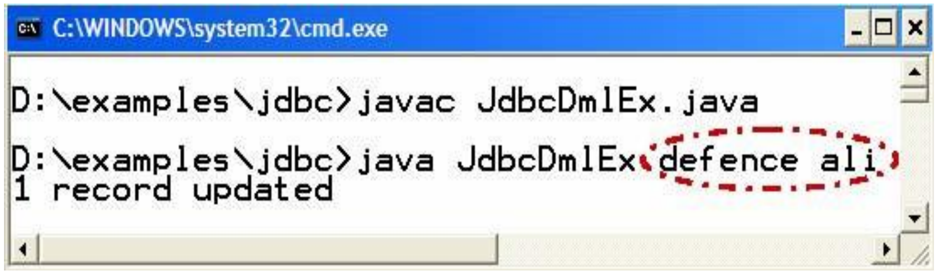


Person : Table

	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

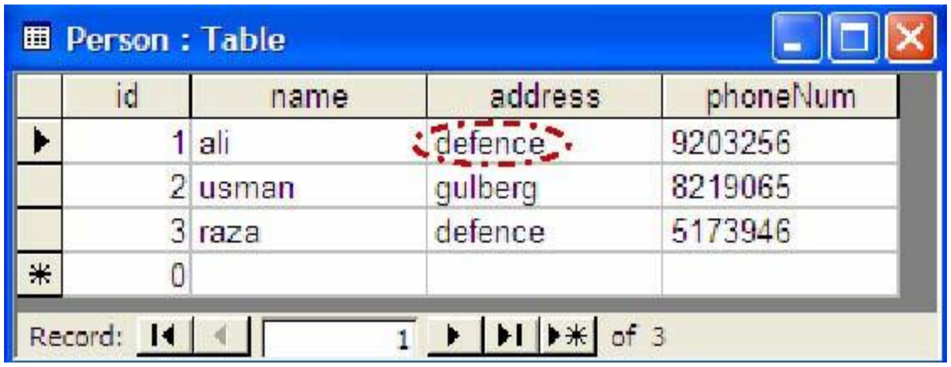
Record: 1 of 3

The next diagram shows how we have executed our program. We passed it two arguments. The first one is the address (defence) and later one is the name (ali) of the person against whom we want to update the address value.



```
D:\examples\jdbc>javac JdbcDmlEx.java

D:\examples\jdbc>java JdbcDmlEx defence ali
1 record updated
```

The Person table is shown in the following diagram after the execution of the program. Notice that address of the ali is now changed to defence.



id	name	address	phoneNum
1	ali	defence	9203256
2	usman	gulberg	8219065
3	raza	defence	5173946
0			

Note

When we execute DML statements (insert, update, delete) we have to commit it in the database explicitly to make the changes permanent or otherwise we can rollback the previously executed statements.

But in the above code, you have never seen such a statement. This is due to the fact that java will implicitly commit the changes. However, we can change this java behavior to manual commit. We will cover these in some later handout.

Useful Statement Methods (cont.):

- o **getMaxRows / setMaxRows(int)**
  - . Used for determines the number of rows a ResultSet may contain
  - . By default, the number of rows are unlimited (return value is 0), or by using setMaxRows(int), the number of rows can be specified.
- o **getQueryTimeOut / setQueryTimeOut (int)**
  - . Retrieves the number of seconds the driver will wait for a Statement object to execute.
  - . The current query time out limit in seconds, zero means there is no limit
  - . If the limit is exceeded, a SQLException is thrown

Different Types of Statements

- . As we have discussed in the previous handout that through Statement objects, SQL queries are sent to the databases.
- . Three types of Statement objects are available. These are;

1. **Statement**

   -The Statement objects are used for executing simple SQL statements. -We have already seen its usage in the code examples.

2. **PreparedStatement**

   -The PrepaeredStatement are used for executing precompiled SQL statements and passing in different parameters to it.

   - We will talk about it in detail shortly.

3. **CallableStatement**

   - Theses are used for executing stored procedures.

   -We are not covering this topic; See the Java tutorial on it if you are interested in learning it.

Prepared Statements

- What if we want to execute same query multiple times by only changing parameters.
- PreparedStatement object differs from Statement object as that it is used to create a statement in standard form that is sent to database for compilation, before actually being used.
- Each time you use it, you simply replace some of the marked parameters (?) using some setter methods.
- We can create PreparedStatement object by using prepareStatementmethod of the connection class. The SQL query is passed to this method as an argument as shown below.

```
PreparedStatement pStmt = con.prepareStatement (“UPDATE tableName SET columnName = ? ” +
                                   “WHERE columnName = ? ” );
```

- Notices that we used marked parameters **(?)** in query. We will replace them later on by using various setter methods.
- If we want to replace first **?** with String value, we use setString method and to replace second **?** with int value, we use setInt method. This is shown in the following code snippet.

```
pStmt.setString (1 , stringValue);

pStmt.setInt (2 , intValue)
```

Note: The first market parameter has index 1.

. Next, we can call executeUpdate (for INSERT, UPDATE or DELETE queries) or executeQuery (for simple SELECT query) method**.**

```
pStmt.executeUpdate();
```

Modify Example Code 15.1: Executing SQL DML using Prepared Statements

This example code is modification to the last example code (JdbcDmlEx.java).The modifications are highlighted as bold face.

```
// File JdbcDmlEx.java

//step 1: import packageimport java.sql.*;

    public class JdbcDmlEx {public static void main (String
        args[ ]) { try {

         //Step 2: load driverClass.forName("sun.jdbc.odbc.JdbcOdbcDriver");

         //Step 3: define the connection URL
         String url = "jdbc:odbc:personDSN";


         //Step 4: establish the connection
         Connection con = DriverManager.getConnection(url, "", "");



         // make query and place ? where values are to
         //be inserted later

         String sql =       "UPDATE Person SET address = ? " + " WHERE name
                            = ? ";

             // creating statement using Connection object and passing  // sql statement as parameter
         PreparedStatement pStmt = con.prepareStatement(sql);

         // assigning first command line argument valueString addVar = args[0];

         // assigning second command line argument valueString nameVar = args[1]; //
         setting first marked parameter (?) by using setString()// method to address.
         pStmt.setString(1 , addVar);

         // setting second marked parameter(?) by using setString()// method to name
         pStmt.setString(2 , nameVar);


         // suppose address is "defence" & name is "ali"
         // by setting both marked parameters, the query will look
         // like:
         // sql = "UPDATE Person SET address = "defence"
         // WHERE name = "ali" "



         // executing update statemnt

         int num = pStmt.executeUpdate();
```

```
            // Step 7: process the results of the query// printing number of records
            affectedSystem.out.println(num + " records updated");

            //Step 8: close the connection
            con.close();


         }catch(Exception sqlEx){
              System.out.println(sqlEx);
              }


     } // end main} // end class
```

Compile & Execute

Execute this code in a similar way as we showed you in execution of the last program. Don't forget to pass the address & name values as the command line arguments.

References:

Entire material for this handout is taken from the book **JAVA A Lab Course** by **Umair Javed**. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 16

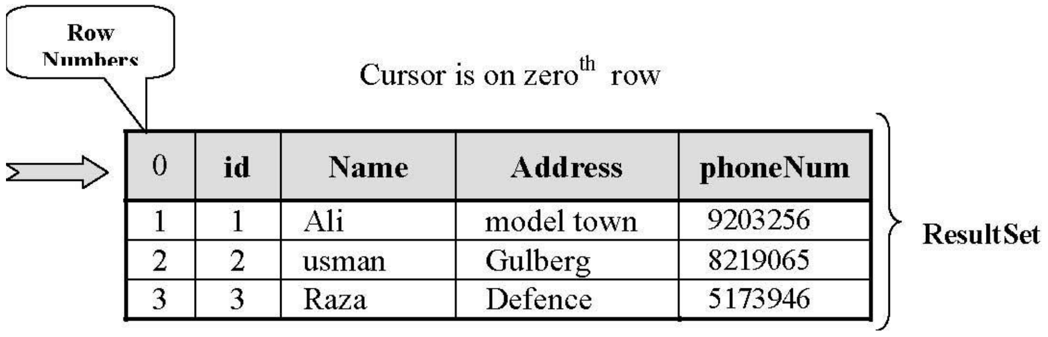
Result Set

This handout will familiarize you with another technique of inserting, updating & deleting rows. Before moving on, first we look at ResultSet.

ResultSet

– A ResultSet contains the results of the SQL query

Represented by a table with rows and columns
Maintains a cursor pointing to its current row of data.
Initially the cursor positioned before the row (0).
First row has index 1



0	id	Name	Address	phoneNum
1	1	Ali	model town	9203256
2	2	usman	Gulberg	8219065
3	3	Raza	Defence	5173946

Default ResultSet

. A default ResultSet object is not updatable and has a cursor that moves forward only.

. You can iterate over through it only once and only from the first row to last row.

. Until now, we have worked & used it in various examples.

. For a quick overview, here how we create a default ResultSet object.

String sql = "SELECT * FROM Person"; PreparedStatement pStmt = con.prepareStatement(sql); **ResultSet rs** = pStmt.executeQuery( );

Useful ResultSet's Methods

Following methods are used often to work with default ResultSet object. We already seen and used some of them in code examples.

- **next( )**

-Attempts to move to the next row in the ResultSet, if available

-The next() method returns true or false depending upon whether the next row is available (exist) or not.

-Before retrieving any data from ResultSet, always remember to call next()at least once because initially cursor is positioned before first row.

- **getters**

-To retrieve the data of the column of the current row you need to use the various getters provided by the

ResultSet

-These getters return the value from the column by specifying column name or column index.

-For example, if the column name is "Name" and this column has index 3 in the ResultSet object, then

we can retrieve the values by using one of the following methods:

```
String name = rs.getString("Name");String name =
rs.getString(3);
```

-These getter methods are also available for other types like getInt( ),getDouble( ) etc. Consult the Java

API documentation for more references.

Note: Remember that first column has an index 1, NOT zero (0).

- **close( )**

-Used to release the JDBC and database resources

-The ResultSet is implicitly closed when the associated Statement object executes a new query or closed by method call.

Updatable and/or Scrollable ResultSet

- It is possible to produce ResultSet objects that are scrollable and/or updatable (since JDK 1.2)
- With the help of such ResultSet, it is possible to move forward as well as backward with in RestultSetobject.
- Another advantage is, rows can be inserted, updated or deleted by using updatable ResultSet object.

Creating Updatable & Scrollable ResultSet

The following code fragment, illustrates how to make a ResultSet object that is scrollable and updatable.

```
String sql = "SELECT * FROM Person";
PreparedStatement pStmt =

con.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR
                                        _UPDATABLE);

ResultSet rs = pStmt.executeQuery( );
```

Two constants have been used of ResultSet class for producing a ResultSet rs that is scrollable, will not show changes made by others and will be updatable

Useful ResultSet's Methods (cont.)

The methods discussed in this section can only be used with updatable/scrollable ResultSet object.

. **previous( )**

-Moves the cursor to the previous row in the ResultSet object, if available -Returns true if

cursor is on a valid row, false it is off the result set. -Throws exception if result type is

TYPE_FORWARD_ONLY.

Example Code 16.1: Use of previous( ), next( ) & various getters methods

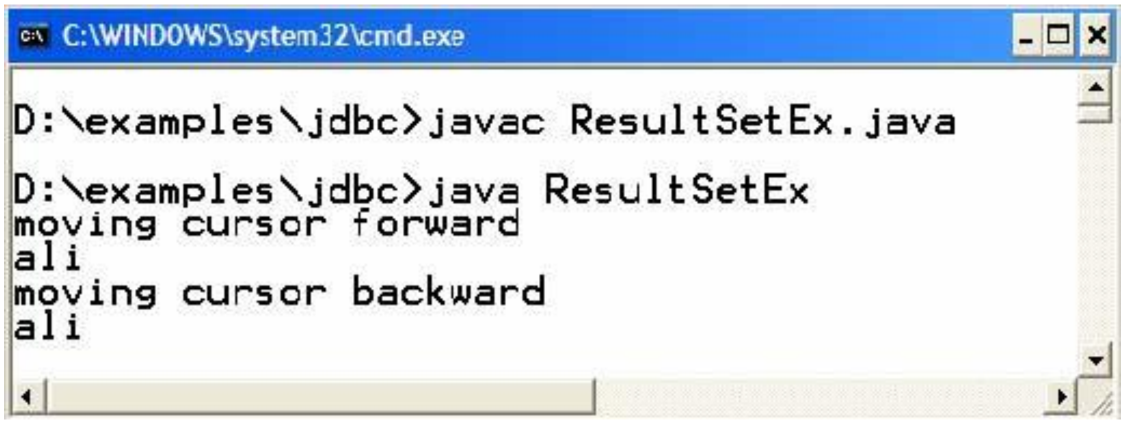
The ResultSetEx.java shows the use of previous, next and getters methods. We are using the same Person table of PersonInfo database, the one we had created earlier in this example and later on.

```
// File ResultSetEx.java
import java.sql.*;
public class ResultSetEx {
 public static void main (String args[ ]) {
 try {
 //Step 2: load driver
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 //Step 3: define the connection URL
 String url = "jdbc:odbc:personDSN";
 //Step 4: establish the connection
 Connection con = DriverManager.getConnection(url);
 //Step 5: creating PrepareStatement by passing sql and
 //ResultSet's constants so that the ResultSet that will
 //produce as a result of executing query will be
 //scrollable & updatable
 String sql = "SELECT * FROM Person";
 PreparedStatement pStmt = con.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
 //Step 6: execute the query
 ResultSet rs = pStmt.executeQuery();
```

```
        // moving cursor forward i.e. first row
       rs.next( );
        // printing column “name” value of current row (first)
        System.out.println(“moving cursor forward”);
        String name = rs.getString(“Name”);
        System.out.println(name);
        // moving cursor forward i.e. on to second row
        rs.next( );
        // moving cursor backward i.e to first row
       rs.previous( );
        // printing column “name” value of current row (first)
        System.out.println(“moving cursor forward”);
        name = rs.getString(“Name”);
        System.out.println(name);
        //Step 8: close the connection
        con.close();
        }catch(Exception sqlEx){
        System.out.println(sqlEx);
        }
        } // end main

       } // end class
```

Compile & Execute:

The sample output is given below:



```
D:\examples\jdbc>javac ResultSetEx.java

D:\examples\jdbc>java ResultSetEx
moving cursor forward
ali
moving cursor backward
ali
```

Useful ResultSet's Methods (cont.)

- **absolute(int)**

-Moves the cursor to the given row number in the ResultSetobject.

-If given row number is positive, moves the cursor forward with respect to beginning of the result set.

-If the given row number is negative, the cursor moves to the absolute row position with respect to the end of the result set.

-For example, calling absolute(-1) positions the cursor on the last row; calling absolute(-2) moves the cursor to next-to-last row, and so on.

-Throws Exception if ResultSet type is TYPE_FORWARD_ONLY

- **updaters (for primitives, String and Object)**

-Used to update the column values in the current row or in insert row (discuss later)

-Do not update the underlying database

-Each update method is overloaded; one that takes column name while other takes column index. For example String updater are available as:

updateString(String columnName, String value)
updateString(String columnIndex, String value)

- **updateRow( )**

-Updates the underlying database with new contents of the current row of this ResultSetobject

Example Code 16.2: Updating values in existing rows

The following code example updates the *Name* column in the second row of the ResultSet object rs and then uses the method updateRow to update the Person table in database.

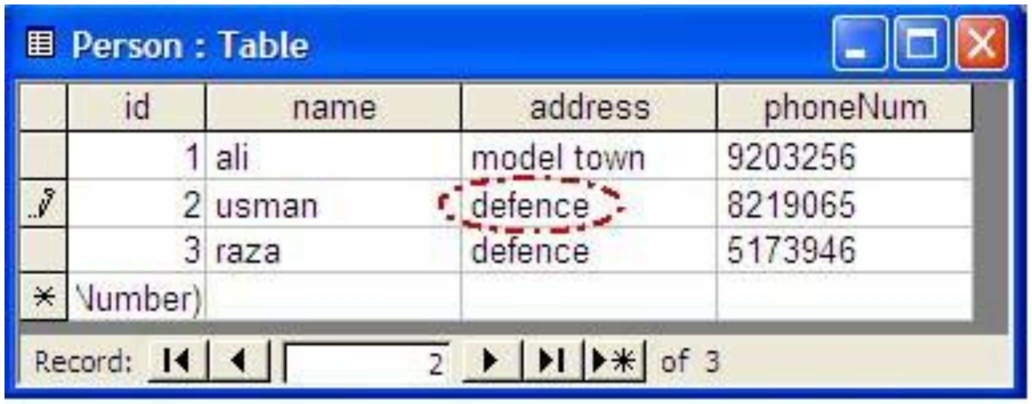
This code is the modification of the last one. Changes made are shown in bold face.

```
// File ResultSetEx.java
import java.sql.*;
public class ResultSetEx {
 public static void main (String args[ ]) {
 try {
 //Step 2: load driver
 Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
 //Step 3: define the connection URL
 String url = “jdbc:odbc:personDSN”;
 //Step 4: establish the connection
 Connection con = DriverManager.getConnection(url);
 //Step 5: create PrepareStatement by passing sql and
 // ResultSet appropriate fields
 String sql = “SELECT * FROM Person”;
 PreparedStatement pStmt = con.prepareStatement(sql,
 ResultSet.TYPE_SCROLL_INSENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 //Step 6: execute the query
 ResultSet rs = pStmt.executeQuery();
```

```
    // moving cursor to second row
   rs.absolute(2);
    // update address column of 2n  row in rs
    rs.updateString("Address", "model town");
    // update the row in database
    rs.updateRow( );
    //Step 8: close the connection
    con.close();
    }catch(Exception sqlEx){
    System.out.println(sqlEx);
    }
    } // end main
   } // end class
```

Compile & Execute

Given below are two states of Person table. Notice that address of 2nd row is updated. Person table: Before execution

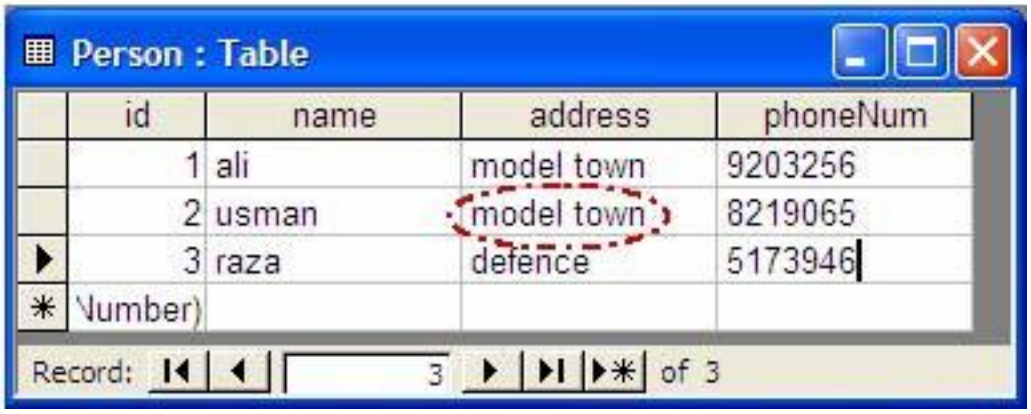


Person : Table

	id	name	address	phoneNum
	1	ali	model town	9203256
	2	usman	defence	8219065
	3	raza	defence	5173946
*	Number)			

Record: 2 of 3

Person table: After execution



Person : Table

	id	name	address	phoneNum
	1	ali	model town	9203256
	2	usman	model town	8219065
	3	raza	defence	5173946
*	Number)			

Record: 3 of 3

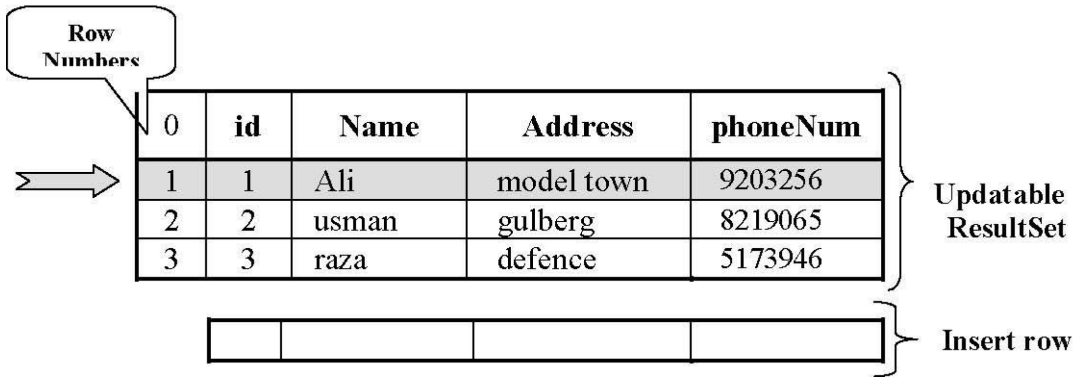
Person table: After execution

Useful ResultSet's Methods (cont.)

. **moveToInsertRow(int)**

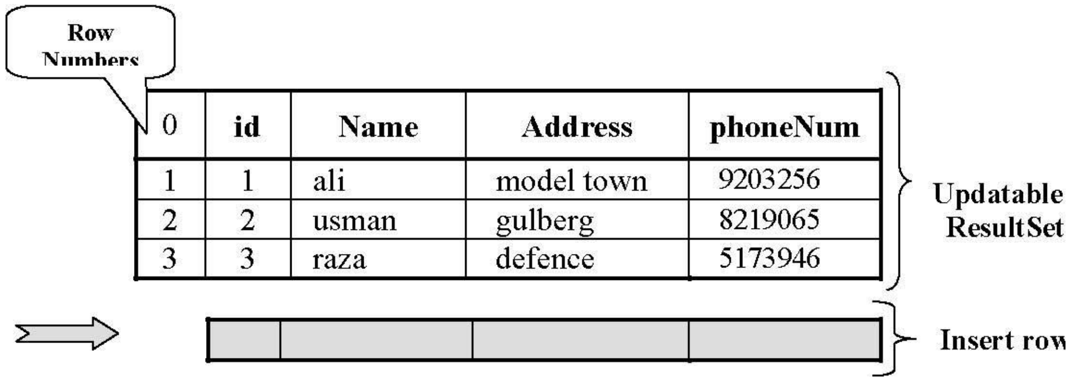
-An updatable resultset object has a special row associate with it i.e. insert row -Insert row – a buffer, where a new row may be constructed by calling updater methods. -Doesn't insert the row into a result set or into a databse.

-For example, initially cursor is positioned on the first row as shown in the diagram.



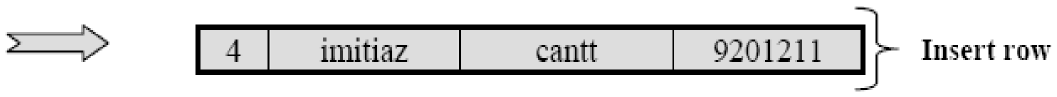
0	id	Name	Address	phoneNum
1	1	Ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946

-By calling moveToInsertRow( ), the cursor is moved to insert row as shown below.



0	id	Name	Address	phoneNum
1	1	ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	raza	defence	5173946

Now by calling various updates, we can insert values into the columns of insert row as shown below.



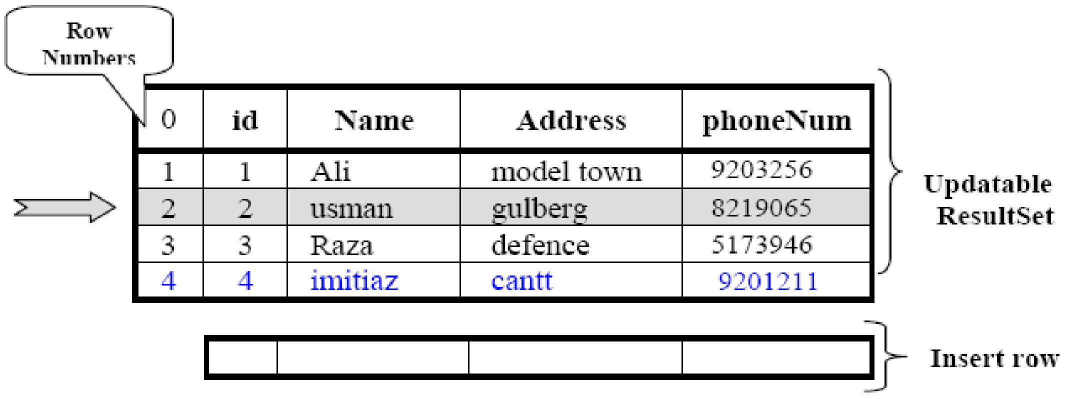
4	imitiaz	cantt	9201211

. **insertRow( )**

-Inserts the contents of the current row into this ResultSet object and into the database too.

-Moves the cursor back to the position where it was before calling moveToInsertRow()

-This is shown in the given below diagram



0	id	Name	Address	phoneNum
1	1	Ali	model town	9203256
2	2	usman	gulberg	8219065
3	3	Raza	defence	5173946
4	4	imitiaz	cantt	9201211

Note: The cursor must be on the insert row before calling this method or exception would be raised.

Example Code 16.3: Inserting new row

The following code example illustrates how to add/insert new row into the ResultSet as well into the database.

This code is the modification of the last one. Changes made are shown in bold face.

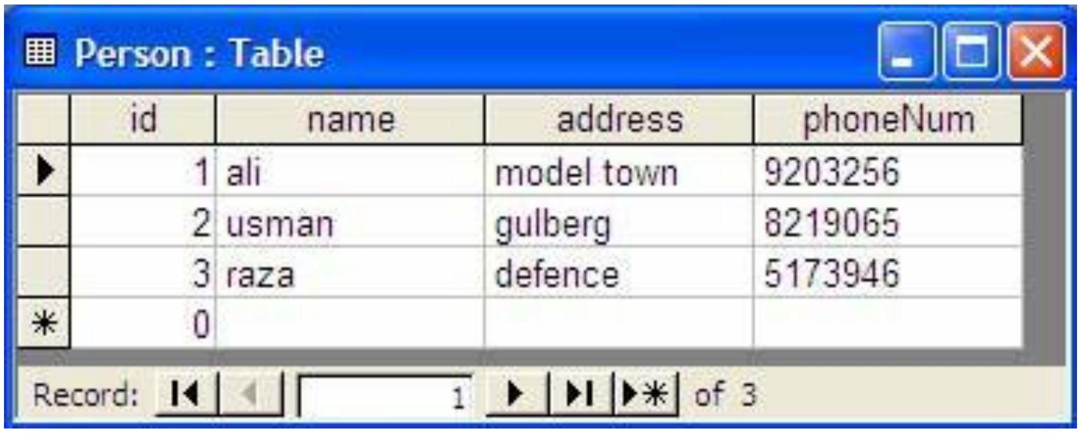
```
// File ResultSetEx.java
import java.sql.*;
public class ResultSetEx {
public static void main (String args[ ]) {
 try {
 //Step 2: load driver
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 //Step 3: define the connection URL
 String url = "jdbc:odbc:personDSN";
 //Step 4: establish the connection
 Connection con = DriverManager.getConnection(url);
 //Step 5: create PrepareStatement by passing sql and
 // ResultSet appropriate fields
 String sql = "SELECT * FROM Person";
 PreparedStatement pStmt = con.prepateStatement(sql,
 ResultSet.TYPE_SCROLL_INSENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 //Step 6: execute the query
 ResultSet rs = pStmt.executeQuery();
 // moving cursor to insert row
rs.moveToInsertRow();
 // updating values in insert row
 rs.updateString( "Name" , "imitiaz" );
 rs.updateString( "Address" , "cantt" );
 rs.updateString( "phoneNum" , "9201211" );
```

```
        // inserting row in resultset & into database
        rs.insertRow( );
        //Step 8: close the connection
        con.close();
        }catch(Exception sqlEx){
        System.out.println(sqlEx);
        }
       } // end main
        } // end class
```

Compile & Execute

Given below are two states of Person table. Note that after executing program, a newly added row is present.

Person table: Before execution

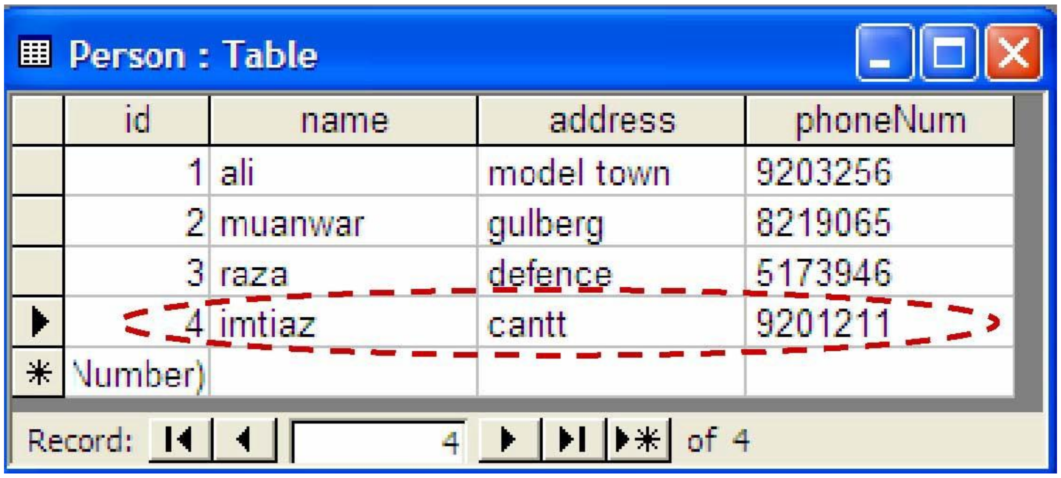


Person : Table

	id	name	address	phoneNum
▶	1	ali	model town	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
✱	0			

Record: 1 of 3

Person table: After execution



Person : Table

	id	name	address	phoneNum
	1	ali	model town	9203256
	2	muanwar	gulberg	8219065
	3	raza	defence	5173946
▶	4	imtiaz	cantt	9201211
✱	Number)			

Record: 4 of 4

Useful ResultSet's Methods (cont.)

. **last( ) & first( )**

-Moves the cursor to the last & first row of the ResultSetobject respectively. -Throws exception if the ResultSet is TYPE_FORWARD_ONLY

. **getRow( )**

-Returns the current row number

-As mentioned earlier, the first row has index 1 and so on.

. **deleteRow( )**

-Deletes the current row from this ResultSet object and from the underlying database.

-Throws exception if the cursor is on the insert row.

Example Code 16.4: Deleting existing row

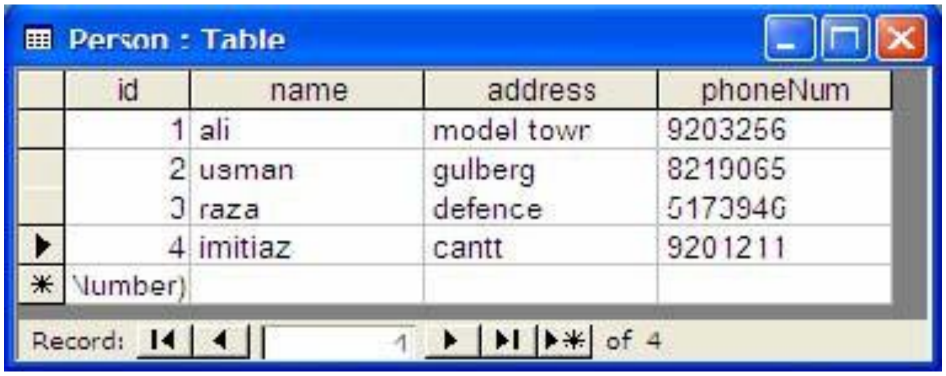
The given below example code shows the usage of last( ), getRow( ) and deleteRow( ) method.

This code is also the modification of the last one. Changes made are shown in bold face.

```
// File ResultSetEx.java
import java.sql.*;
public class ResultSetEx {
public static void main (String args[ ]) {
 try {
 //Step 2: load driver
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 //Step 3: define the connection URL
 String url = "jdbc:odbc:personDSN";
 //Step 4: establish the connection
 Connection con = DriverManager.getConnection(url);
 //Step 5: create PrepareStatement by passing sql and
 // ResultSet appropriate fields
 String sql = "SELECT * FROM Person";
 PreparedStatement pStmt = con.prepateStatement(sql,
 ResultSet.TYPE_SCROLL_INSENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 //Step 6: execute the query
 ResultSet rs = pStmt.executeQuery();
 // moves to last row of the resultset
rs.last();
 // retrieving the current row number
 int rNo = rs.getRow();
 System.out.println("current row number" + rNo);
 // delete current row from rs & db i.e. 4 because
 // previously we have called last() method
 rs.deleteRow( );
 //Step 8: close the connection
 con.close();
 }catch(Exception sqlEx){
 System.out.println(sqlEx);
 }
} // end main
} // end class
```

Compile & Execute

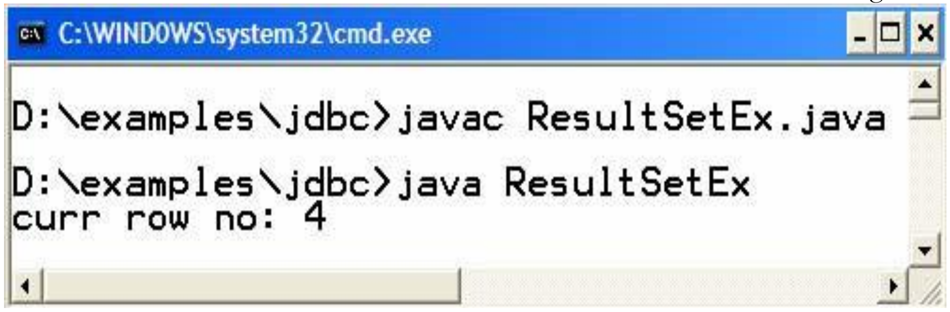
The first diagram shows the Person table before execution. Person table: Before execution


Person : Table

	id	name	address	phoneNum
	1	ali	model towr	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
▶	4	imitiaz	cantt	9201211
✳	Number)			

Record: 1 of 4

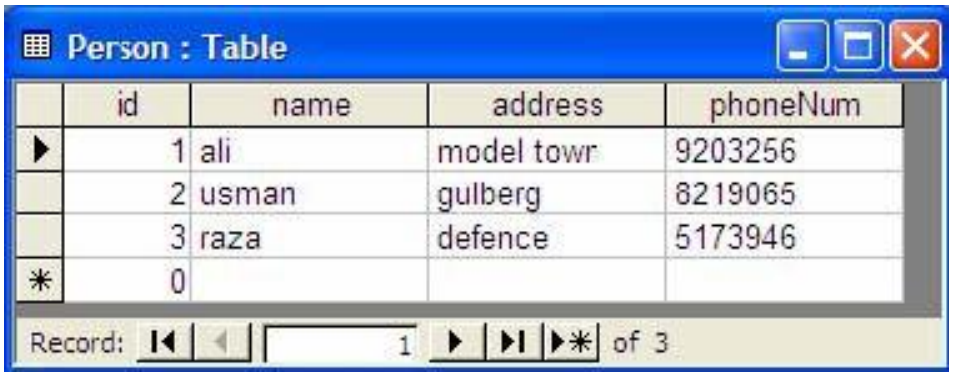
Execution program from command prompt will result in displaying current row number on console. This can be confirmed from following diagram.



```
D:\examples\jdbc>javac ResultSetEx.java

D:\examples\jdbc>java ResultSetEx
curr row no: 4
```

After execution, the last row (4) is deleted from ResultSet as well as from database. The Person table is shown after execution

Person table: After execution



Person : Table

	id	name	address	phoneNum
▶	1	ali	model towr	9203256
	2	usman	gulberg	8219065
	3	raza	defence	5173946
*	0			

Record: 1 of 3

References:

Entire material for this handout is taken from the book **JAVA A Lab Course** by **Umair Javed**. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 17

Meta Data

In simple terms, Meta Data is data (information) about data. The actual data has no meaning without existence of Meta data. To clarify this, let's look at an example. Given below are listed some numeric values

1000
2000
4000

What this information about? We cannot state accurately. These values might be representing some one's salaries, price, tax payable & utility bill etc. But if we specify Meta data about this data like shown below:

Salary
1000
2000
4000

Now, just casting a glance on these values, you can conclude that it's all about some ones salaries.

ResultSet Meta data

ResultSet Meta Data will help you in answering such questions

-How many columns are in the ResultSet?

-What is the name of given column?

-Are the column name case sensitive?

-What is the data type of a specific column?

-What is the maximum character size of a column?

-Can you search on a given column?

Creating ResultSetMetaData object

From a ResultSet (the return type of executeQuery()), derive a ResultSetMetaData object by calling getMetaData() method as shown in the given code snippet (here rsis a valid ResultSetobject):

```
ResultSetMetaData rsmd = rs.getMetaData();
```

Now, rsmd can be used to look up number, names & types of columns

Useful ResultSetMetaData methods

- **getColumnCount ( )**
  - Returns the number of columns in the result set
- **getColumnDisplaySize (int)**
  - Returns the maximum width of the specified column in characters
- **getColumnName(int) / getColumnLabel (int)**
  - The getColumnName() method returns the database name of the column
  - The getColumnLabel() method returns the suggested column label for printouts
- **getColumnType (int)**
  - Returns the SQL type for the column to compare against types in java.sql.Types

Example Code 17.1: Using ResultSetMetaData

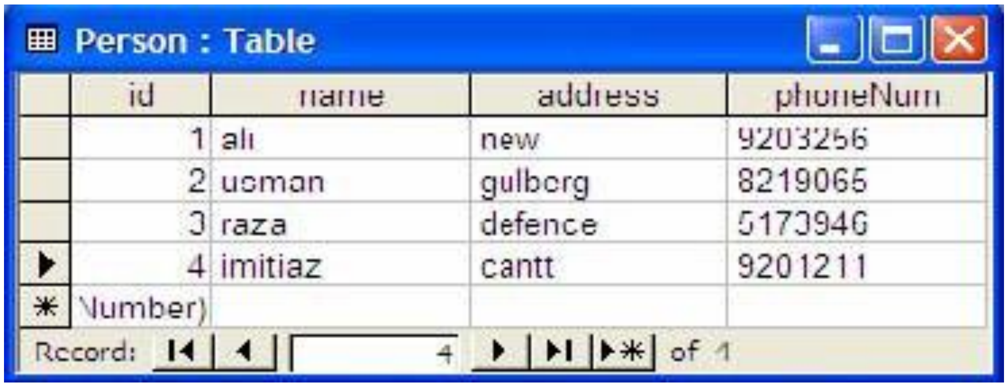
The MetaDataEx.java will print the column names by using ResultSetMetaData object and column values on console. This is an excellent example of the scenario where we have no idea about the column names in advance

Note: For this example code and for the coming ones, we are using the same database (PersonInfo) the one we created earlier and repeatedly used. Changes are shown in bold face

```
// File MetaDataEx.java
import java.sql.*;
public class MetaDataEx {
 public static void main (String args[ ]) {
 try {
 //Step 2: load driver
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 //Step 3: define the connection URL
 String url = "jdbc:odbc:personDSN";
 //Step 4: establish the connection
 Connection con = null;
 con = DriverManager.getConnection(url, "", "");
 //Step 5: create PrepareStatement by passing sql and
 // ResultSet appropriate fields
 String sql = "SELECT * FROM Person";
 PreparedStatement pStmt = con.prepateStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
 //Step 6: execute the query
 ResultSet rs = pStmt.executeQuery();
 // get ResultSetMetaData object from rs
 ResultSetMetaData rsmd = rs.getMetaData( );
 // printing no. of column contained by rs
int numColumns = rsmd.getColumnCount();
 System.out.println("Number of Columns:" + numColumns);
 // printing all column names by using for loop
 String cName;
 for(int i=1; i<= numColumns; i++) {
 cName = rsmd.getColumnName(i);
System.out.println(cName);
 System.out.println("\t");
 }
 // changing line or printing an empty string
 System.out.println(" ");
 // printing all values of ResultSet by iterating over it
 String id, name, add, ph;
 while( rs.next() )
 {
 id = rs.getString(1);
 name = rs.getString(2);
 add = rs.getString(3);
 ph = rs.getString(4);
 System.out.println(id);
 System.out.println("\t");
 System.out.println(name);
 System.out.println("\t");
 System.out.println(add);
 System.out.println("\t");
 System.out.println(ph);
 System.out.println(" ");
 }
 //Step 8: close the connection
 con.close();
 }catch(Exception sqlEx){
 System.out.println(sqlEx);
 }
 } // end main101.} // end class
```

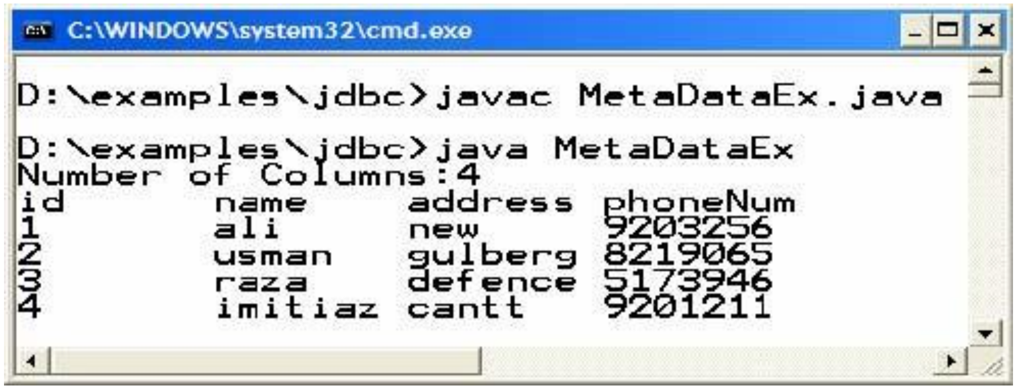
Compile & Execute:

The database contains the following values at the time of execution of this program. The database and the output are shown below:


Person : Table

id	name	address	phoneNum
1	ali	new	9203256
2	usman	gulberg	8219065
3	raza	defence	5173946
4	imitiaz	cantt	9201211
(Number)			

Record: 4 of 4



```
C:\WINDOWS\system32\cmd.exe

D:\examples\jdbc>javac MetaDataEx.java

D:\examples\jdbc>java MetaDataEx
Number of Columns:4
id      name     address  phoneNum
1       ali      new      9203256
2       usman    gulberg  8219065
3       raza     defence  5173946
4       imitiaz  cantt    9201211
```

DataBaseMetaData

DataBase Meta Data will help you in answering such questions

- What SQL types are supported by DBMS to create table?
- What is the name of a database product?
- What is the version number of this database product?
- What is the name of the JDBC driver that is used?
- Is the database in a read-only mode?

Creating DataBaseMetaData object

From a Connection object, a DataBaseMetaData object can be derived. The following code snippet demonstrates how to get DataBaseMetaDataobject.

Connection con= DriverManager.getConnection(url, usr, pwd);
DataBaseMetaData dbMetaData = con.getMeataData();

Now, you can use the dbMetaData to gain information about the database.

Useful ResultSetMetaData methods

. **getDatabaseProductName( )**
– Returns the name of the database's product name
. **getDatabaseProductVersion( )**
– Returns the version number of this database product
. **getDriverName( )**
– Returns the name of the JDBC driver used to established the connection
. **isReadOnly( )**
– Retrieves whether this database is in read-only mode
– Returns true if so, false otherwise

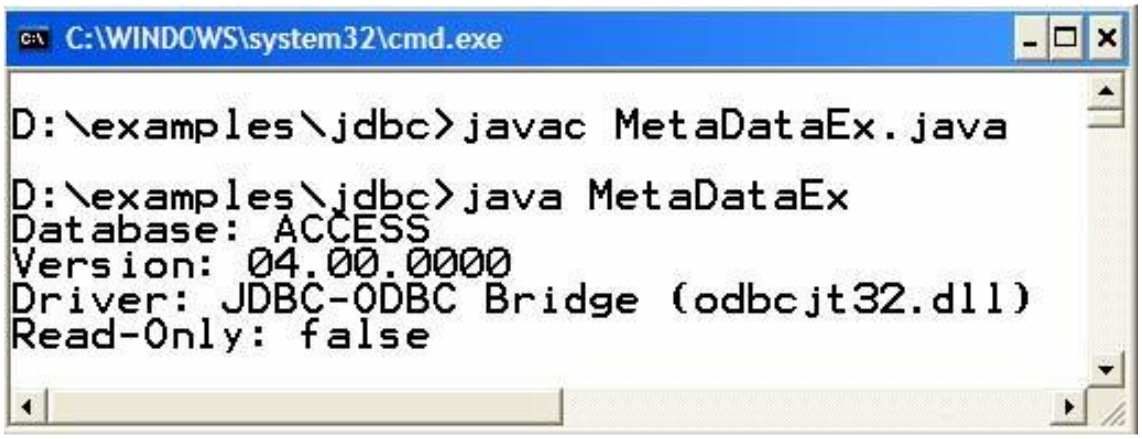
Example Code 17.2: using DataBaseMetaData

This code is modification of the example code 17.1. Changes made are shown in bold face.

```
// File MetaDataEx.java
import java.sql.*;
public class MetaDataEx {

    public static void main (String args[ ]) {
    try {
    //Step 2: load driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    //Step 3: define the connection URL
    String url = "jdbc:odbc:personDSN";

    //Step 4: establish the connection
    Connection con = null;
    con = DriverManager.getConnection(url, "", "");

    // getting DataBaseMetaDat object
    DataBaseMetaData dbMetaData = con.getMetaData();

    // printing database product name
    Sring pName = dbMetaData.getDatabaseProductName();
    System.out.println("DataBase: " + pName);

    // printing database product version
    Sring pVer = dbMetaData.getDatabaseProductVersion();
    System.out.println("Version: " + pVer);

    // printing driver name used to establish connection &
    // to retrieve data
    Sring dName = dbMetaData.getDriverName();
    System.out.println("Driver: " + dName);

    // printing whether database is read-only or not
    boolean rOnly = dbMetaData.isReadOnly();
    System.out.println("Read-Only: " + rOnly);

    // you can create & execute statements and can
    // process results over here if needed

    //Step 8: close the connection
    con.close();

    }catch(Exception sqlEx){
    System.out.println(sqlEx);
    }

    } // end main
   } // end class
```

Compile & Execute

On executing the above program, the following output will produce:



```
D:\examples\jdbc>javac MetaDataEx.java

D:\examples\jdbc>java MetaDataEx
Database: ACCESS
Version: 04.00.0000
Driver: JDBC-ODBC Bridge (odbcjt32.dll)
Read-Only: false
```

<u>JDBC Driver Types</u>

- JDBC Driver Types are divided into four types or levels.
- Each type defines a JDBC driver implementation with increasingly higher level of platform independence, performance, deployment and administration.
  The four types are:

  Type – 1: JDBC – ODBC Bridge

  Type 2: Native – API/partly Java driver

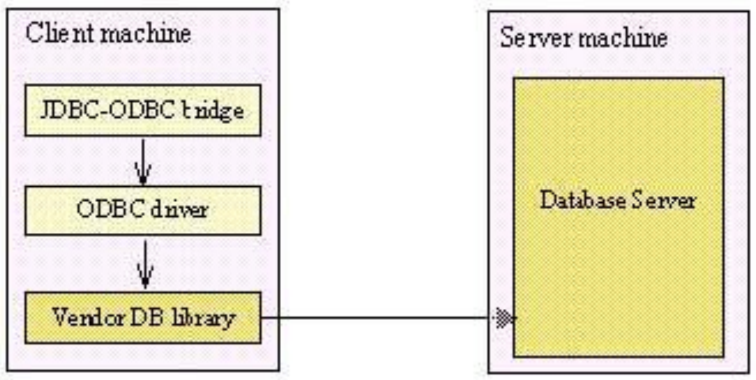
  Type 3: Net – protocol/all–Java driver

  Type 4: Native – protocol/all–Java driver

Now, let's look at each type in more detail

Type – 1: JDBC – ODBC Bridge

-Translates all JDBC calls into ODBC (Open Database Connectivity) calls and send them to the ODBC Driver - Generally used for Microsoft database. -Performance is degraded

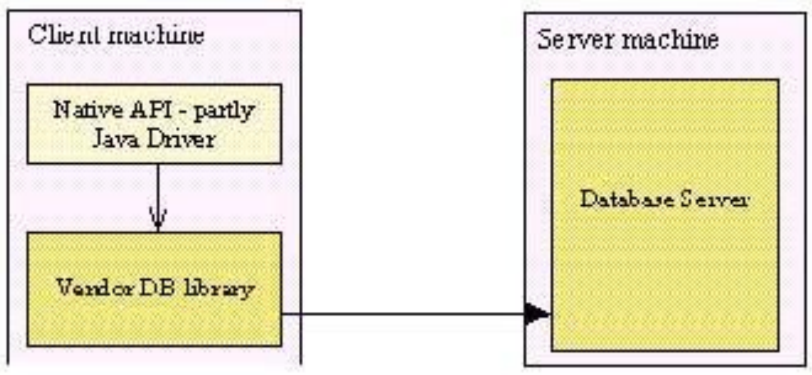


4. Type – 2: Native – API/partly Java driver

-Converts JDBC calls into database-specific calls such as SQL Server, Informix, Oracle or Sybase.

-Partly-Java drivers communicate with database-specific API (which may be in C/C++) using the Java Native Interface.

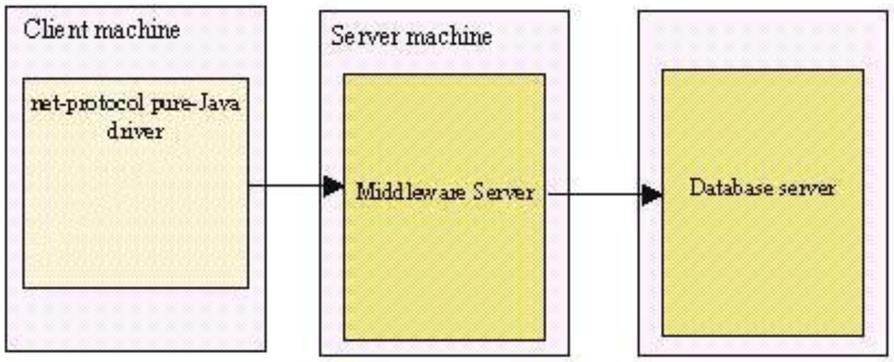
-Significantly better Performance than the JDBC-ODBC bridge



4. Type – 3: Net – protocol/all–Java driver

-Follows a three-tiered approach whereby the JDBC database requests ()are passed through the network to the middle-tier server

-Pure Java client to server drivers which send requests that are not database-
specific to a server that translates them into a database-specific protocol. . -If the middle-tier server is written in java, it can use a type 1or type 2JDBC driver
to do this

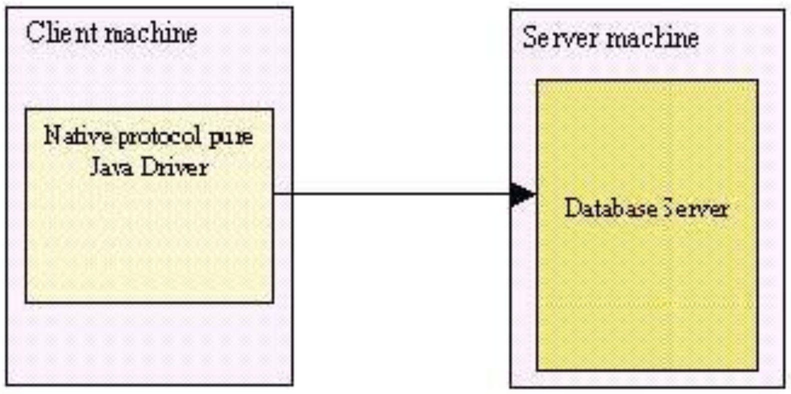


4. Type – 4: Native – protocol / all – java driver

-Converts JDBC calls into the vendor-specific DBMS protocol so that client application can communicate directly with the database server

-Completely implemented in Java to achieve platform independence and eliminate deployment issues.

-Performance is typically very good



On – Line Resources

- **Sun's JDBC Site**
  http://java.sun.com/products/jdbc/
- **JDBC Tutorial**
  http://java.sun.com/docs/books/tutorial/jdbc/
- **List of available JDBC Drivers**
  http://industry.java.sun.com/products/jdbc/drivers/
- **RowSet Tutorial**
  http://java.sun.com/developer/Books/JDBCTutorial/chapter5.html
- **JDBC RowSets Implementation Tutorial**
  http://java.sun.com/developer/onlineTraining/ Database/**jdbcrowset**s.pdf

References:

- Java API documentation 5.0
- Java – A Lab Course by Umair Javed
- JDBC drivers in the wild
  http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc_p.html

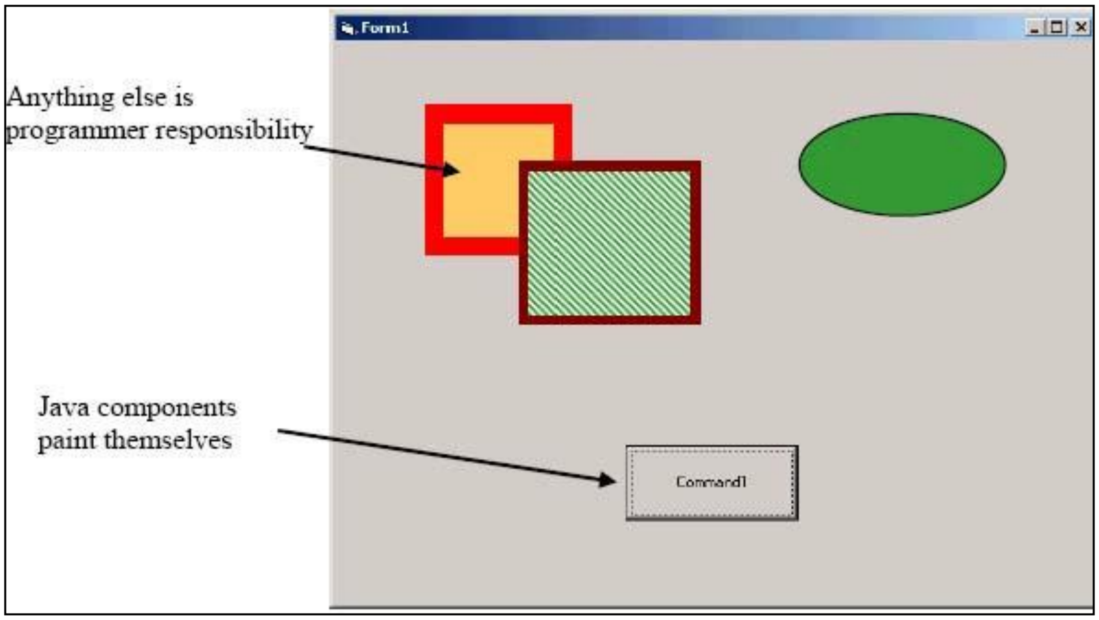
Lesson 18

Java Graphics

Painting

Window is like a painter's canvas. All window paints on the same surface. More importantly, windows don't remember what is under them. There is a need to repaint when portions are newly exposed.

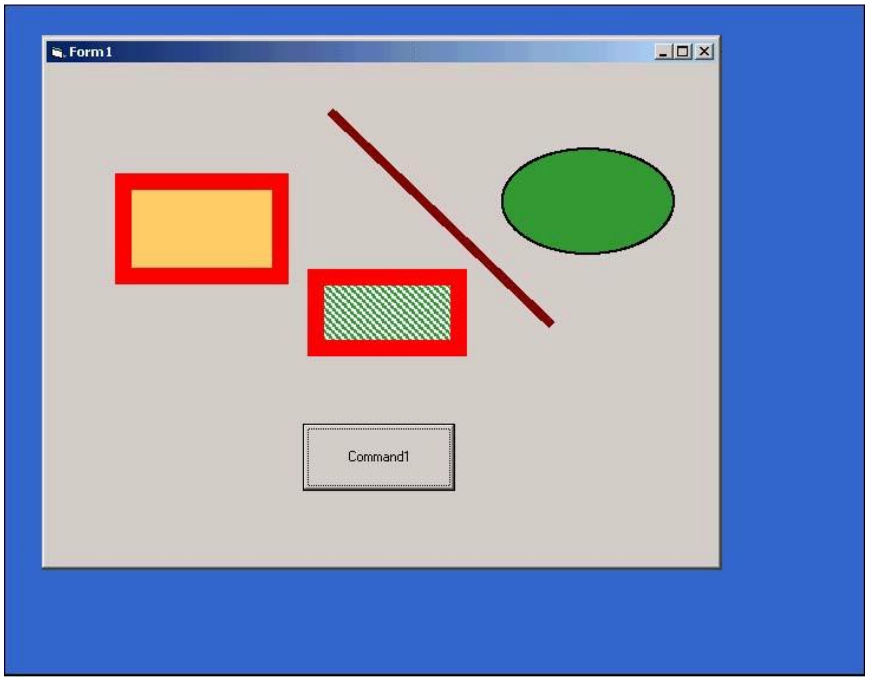
Java components are also able to paint themselves. Most of time, painting is done automatically. However sometimes you need to do drawing by yourself. Anything else is programmer responsibility


Form1
Anything else is programmer responsibility
Java components paint themselves
Command1

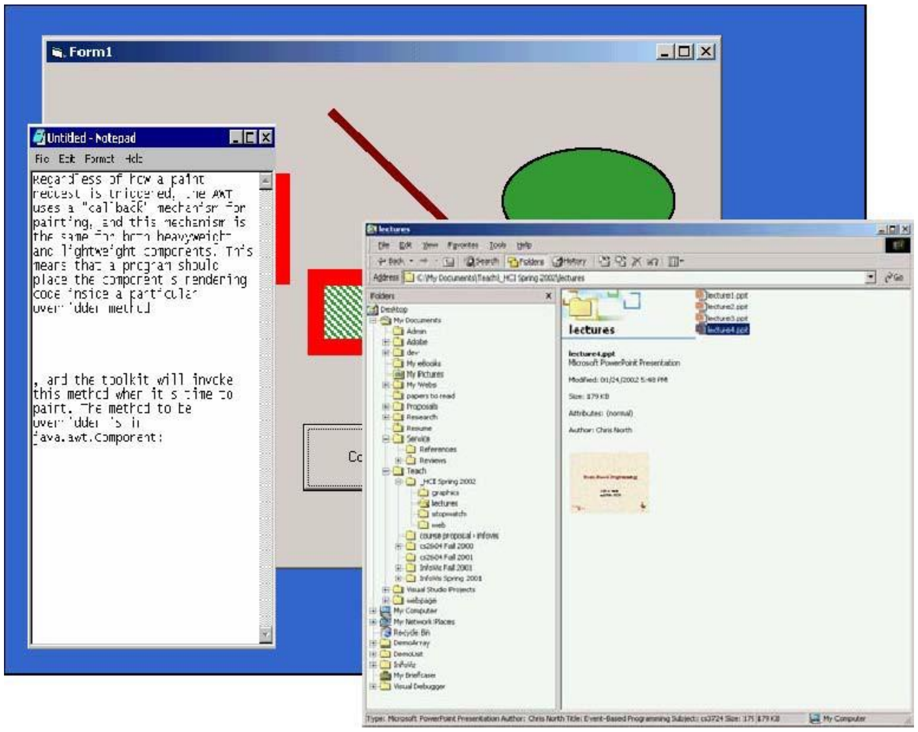
How painting works?

Let's take windows example. Consider the following diagram in which the blue area is representing the desktop. The one frame (myApp) is opened in front of desktop with some custom painting as shown below.

myApp consist of a JPanel. The JPanel contains a JButton. Two rectangles, a circle & a lines are also drawn on the JPanel.



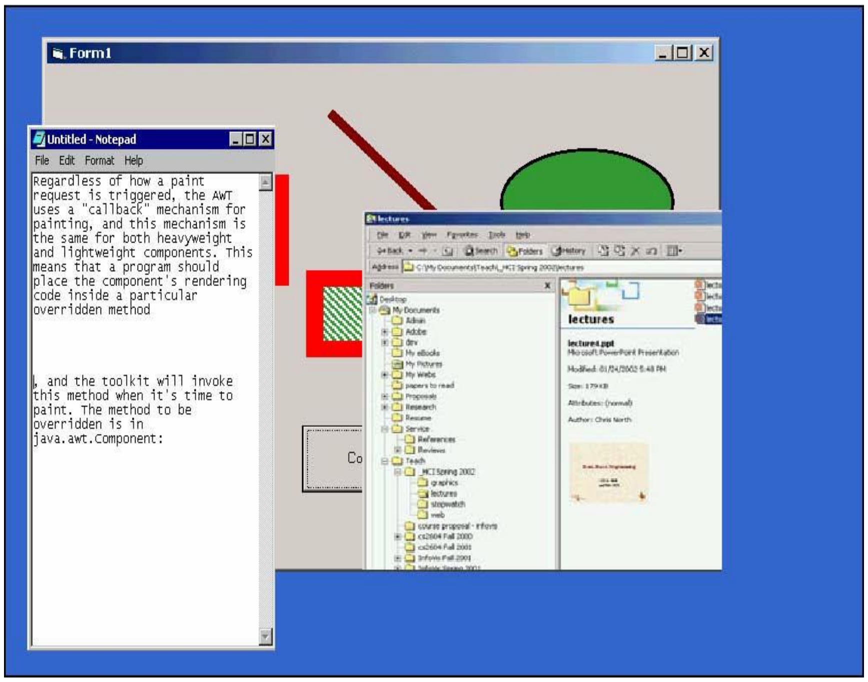
After opening notepad and windows explorer window, diagram will look like this:



Regardless of how a paint request is triggered, the AWT uses a "callback" mechanism for painting, and this mechanism is the same for both heavyweight and lightweight components. This means that a program should place the component's rendering code inside a particular overridden method

, and the toolkit will invoke this method when it's time to paint. The method to be overridden is in java.awt.Component:

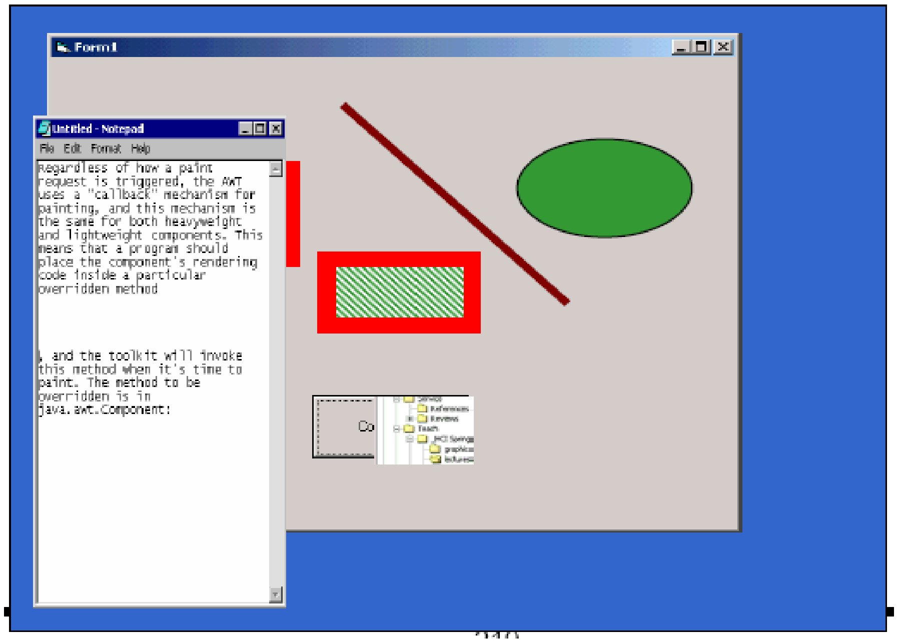
Lets shuts off the windows explorer, the repaint event is sent to desktop first and then to myApp. The figure shown below describes the situation after desktop repaint event get executed. Here you can clearly see that only desktop repaints itself and window explorer remaining part is still opened in front of myApp.



Regardless of how a paint
request is triggered, the AWT
uses a "callback" mechanism for
painting, and this mechanism is
the same for both heavyweight
and lightweight components. This
means that a program should
place the component's rendering
code inside a particular
overridden method

, and the toolkit will invoke
this method when it's time to
paint. The method to be
overridden is in
java.awt.Component:

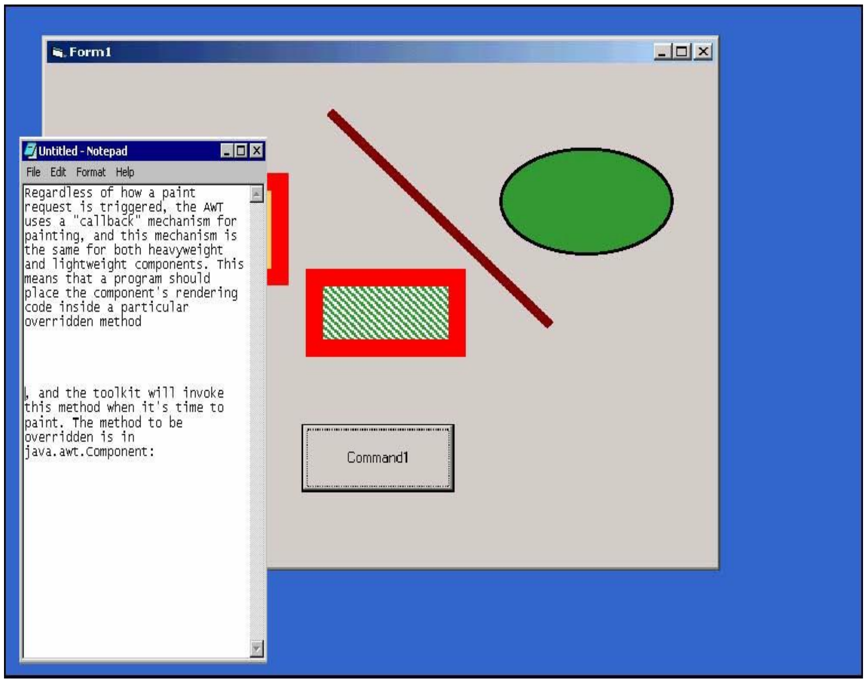
The following figure shows the situation when myApp's JPanel calls its repaint method. Notice that some portion of window explorer is still remains in front of JButton because yet not repaint event is sent to it.



Regardless of how a paint
request is triggered, the AWT
uses a "callback" mechanism for
painting, and this mechanism is
the same for both heavyweight
and lightweight components. This
means that a program should
place the component's rendering
code inside a particular
overridden method

, and the toolkit will invoke
this method when it's time to
paint. The method to be
overridden is in
java.awt.Component:

Next, JPanel forwards repaint event to JButton that causes the button to be displayed in its original form.



```
Regardless of how a paint
request is triggered, the AWT
uses a "callback" mechanism for
painting, and this mechanism is
the same for both heavyweight
and lightweight components. This
means that a program should
place the component's rendering
code inside a particular
overridden method



, and the toolkit will invoke
this method when it's time to
paint. The method to be
overridden is in
java.awt.Component:
```

This is all done automatically and we cannot feel this process cause of stunning speed of modern computers that performs all these steps in flash of eye.

Painting a Swing Component

Three methods are at the heart of painting a swing component like JPanel etc. For instance, paint() gets called when it's time to render -- then Swing further factors the paint() call into three separate methods, which are invoked in the following order:

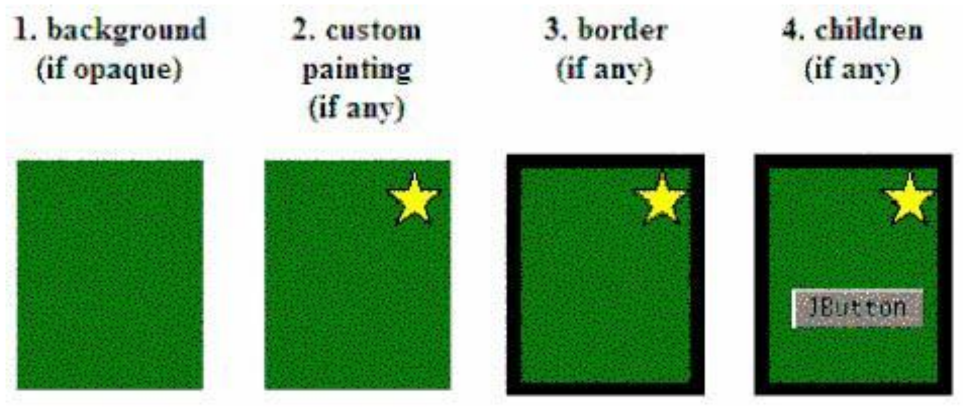
— protected void paintComponent(Graphics g)
— protected void paintBorder(Graphics g)
— protected void paintChildren(Graphics g)

Lets look at these methods in order in which they get executed

paintComponet( )
— it is a main method for painting
— By default, it first paints the background
— After that, it performs custom painting (drawing circle, rectangles etc.)
paintBorder( )
— Tells the components border (if any) to paint.
— It is suggested that you do not override or invoke this method
paintChildern( )
— Tells any components contained by this component to paint themselves
— It is suggested that you do not override or invoke this method too.

Example: Understanding methods calls

Consider the following figure



The figure above illustrates the order in which each component that inherits from

JComponent paint itself. Figure 1 to 2 --painting the background and performing custom painting is performed by the paintComponent method

In Figure 3 – paintBorder is get called And finally in figure 4 – paintChildern is called that causes the

JButton to render itself. Note: The important thing to note here is for JButton (since it is a JComponent),

all these

methods are also called in the same order.

<u>**Your Painting Strategy**</u>

You must follow the three steps in order to perform painting.

Subclass JPanel

– class MyPanel extends JPanel

– Doing so MyPanel also becomes a JPanle due to inheritance

Override the paintComponent(Graphics g) method

– Inside method using graphics object, do whatever drawing you want to do

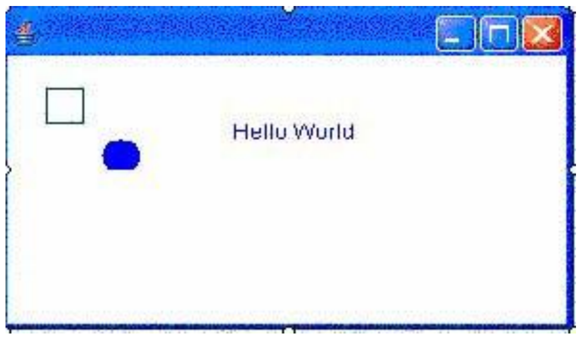
Install that JPanel inside a JFrame

– When frame becomes visible through the paintChildren() method your panel become visible

– To become visible your panel will call paintComponent() method which will do your custom drawing

Example Code 18.1:

Suppose, we want to draw one circle & rectangle and a string "Hello World".



The first step is building a class that inherits from JPanel. The following class MyPanel is fulfilling this requirement. paintComponent( ) method is also override in this class. The sample code is given below

```
// importing required packagesimport javax.swing.*;import java.awt.*;

// extending class from JPanelpublic class MyPanel extends JPanel {

   // overriding paintComponent method
   public void paintComponent(Graphics g){
        // erasing behaviour – this will clear all the// previous painting super.paintComponent(g);

        // Down casting Graphics object to Graphics2DGraphics2D g2 = (Graphics2D)g;

        // drawing rectanle
        g2.drawRect(20,20,20,20);

        // changing the color to blue
        g2.setColor(Color.blue);

        // drawing filled oval with color i.e. blueg2.fillOval(50,50,20,20);

        // drawing stringg2.drawString("Hello World", 120, 50);

   }// end paintComponent

} // end MyPanel class
```

The Test class that contains the main method as well uses MyPainel (previously built) class is given below

```
// importing required packages
import javax.swing.*;
import java.awt.*;

public class Test {

    JFrame f;

    // declaring Reference of MyPanel class

    MyPanel p;

    // parameter less constructor

    public Test(){
        f = new JFrame();
        Container c = f.getContentPane();

        c.setLayout(new BorderLayout());

        // instantiating reference

        p = new MyPanel();

        // adding MyPanel into container

        c.add(p);

        f.setSize(400,400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    } // end constructor
     // main method
     public static void main(String args[ ]){
     Test t = new Test();
     }
```

Note: Here we have used only some methods (drawRect( ) & fillOval( ) etc.) of Graphics class. For a complete list, see the Java API documentation

Lesson 19

How to Animate?

If we want to animate something like ball, moving from one place to another, we constantly need to call paintComponent( ) method and to draw the shape (ball etc.) at new place means at new coordinates.

Painting is managed by system, so calling paintComponent() directly is not recommended at all. Similarly calling paint( ) method is also not recommended. Why? Because such code may be invoked at times when it is not appropriate to paint -- for instance, before the component is visible or has access to a valid Graphics object.

Java gives us a solution in the from of repaint( ) method. Whenever we need to repaint, we call this method that in fact makes a call to paint( ) method at appropriate time.

Problem & Solution

- What to do to move the shapes present in example code 18.1 (last example) when a mouse is dragged
- First time painting is what we already have done
- When a mouse is clicked find the co-ordinates of that place and paint Rectangle at that place by requesting, using repaint() call
- Here instead of Hard-coding the position of co-ordinates uses some variables. For example mx, my
    - In the last example code, we draw a rectangle by passing hard-coded values like 20

      g.drawRect(20,20,20,20);

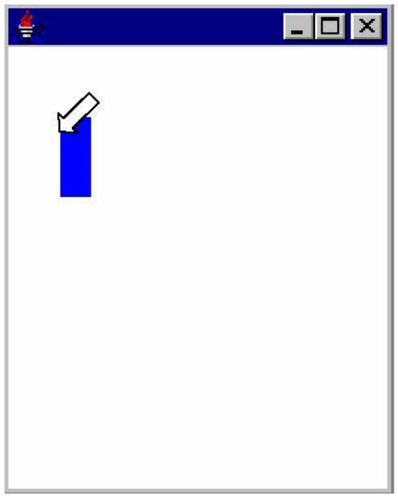
    - Now, we'll use variables so that change in a variable value causes to display a rectangle at a new location

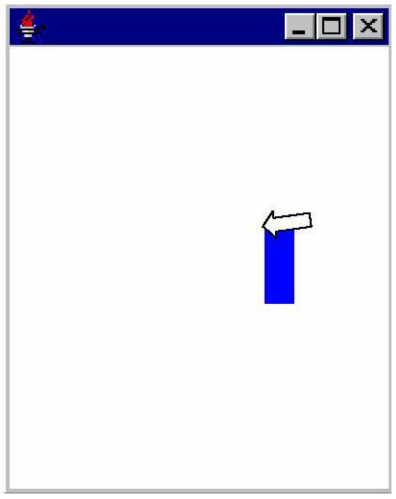
      g.drawRect(**mx**,**my**,20,20;

- Similarly, you have seen a tennis game (during lecture). Now, what to do code the paddle movement.
- In the coming up example. We are doing it using mouse, try it using mouse.

Example Code 19.1

The following outputs were produced when mouse is dragged from one location to anther





First we examine the MyPanel.java class that is drawing a filled rectangle.

```
import javax.swing.*;import java.awt.*;

// extending class from JPanelpublic class MyPanel extends JPanel {

   // variables used to draw rectangles at different//locations
   int mX = 20;
   int mY = 20;

   // overriding paintComponent method
   public void paintComponent(Graphics g){

        // erasing behaviour – this will clear all the// previous painting super.paintComponent(g);

        // Down casting Graphics object to Graphics2D
        Graphics2D g2 = (Graphics2D)g;

        // changing the color to blue
        g2.setColor(Color.blue);

        // drawing filled oval with color i.e. blue// using instance variablesg2.fillRect(mX,mY,20,20);

   }// end paintComponent
```

The Test class is given below. Additionally this class also contains the code for handling mouse events.

```
// importing required packagesimport javax.swing.*;import java.awt.*;

public class Test {
    JFrame f;

    // declaring Reference of MyPanel class
    MyPanel p;

    // parameter less constructor
    public Test(){


       f = new JFrame();
       Container c = f.getContentPane();


       c.setLayout(new BorderLayout());

       // instantiating reference
       p = new MyPanel();

       // adding MyPanel into container
       c.add(p);

       f.setSize(400,400);
       f.setVisible(true);
       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


       // creating inner class object

       Handler h = new Handler();

       // registering MyPanel to handle events
       p.addMouseMotionListner(h);

 } // end constructor

   // inner class used for handling events
  public class Handler extends MouseMotionAdapter{

    // capturing mouse dagged events
    public void mouseDragged(MouseEvent me){

        // getting the X-Position of mouse and assigning// value to instance variable mX of MyPanel
        class
        p.mX = me.getX();

        // getting the Y-Position of mouse and assigning// value to instance variable mX of MyPanel
        class
        p.mY = me.getY();



        // call to repaint causes rectangle to be drawn on// new location
```

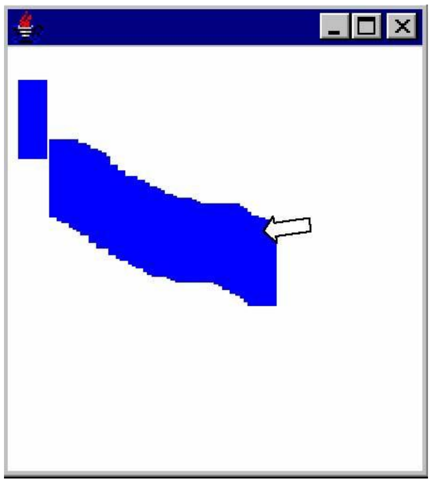
```
            p.repaint() ;

        } // end mouseDragged

        } // end Handler class

        // main method

        public static void main(String args[ ]){
              Test t = new Test();
              }

        } // end MyPanel class
```

On executing this program, when you drag mouse from one location to another, rectangle is also in sink with the movement of mouse. Notice that previously drawn rectangle is erased first.

If we exclude or comment out the following line from MyPanel class

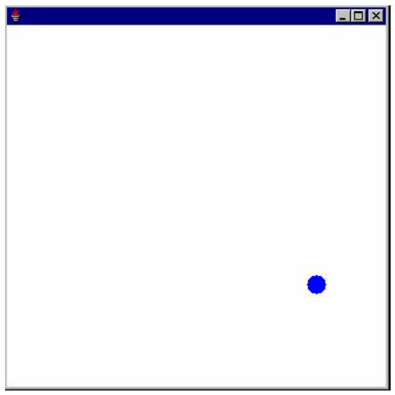
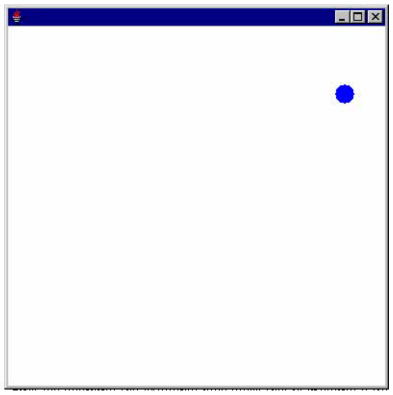
```
                    super.paintComponent(g);
```

Dragging a mouse will produce a similar kind of output shown next



Example Code 19.2: Ball Animation

The ball is continuously moving freely inside the corner of the frames. The sample outputs are shown below:

First we examine the MyPanel.java class that is drawing a filled oval.

```
import javax.swing.*;import java.awt.*;

// extending class from JPanelpublic class MyPanel extends JPanel {

   // variables used to draw oval at different locations
   int mX = 200;
   int mY = 0;

   // overriding paintComponent method
   public void paintComponent(Graphics g){
        // erasing behaviour – this will clear all the// previous painting super.paintComponent(g);

        // Down casting Graphics object to Graphics2D
        Graphics2D g2 = (Graphics2D)g;

        // changing the color to blue
        g2.setColor(Color.blue);

        // drawing filled oval with blue color
        // using instance variables
        g2.fillOval(mX,mY,20,20);

   }// end paintComponent

} end of MyPanel class
```

The Test class is given below. Additionally this class also contains the code for handling mouse events.

```
// importing required packagesimport javax.swing.*;import
java.awt.*;Import java.awt.event.*;

public class AnimTest implements ActionListener {

    JFrame f;
    MyPanel p;


    // used to control the direction of ball
    int x, y;

    public AnimTest(){

        f = new JFrame();
        Container c = f.getContentPane();
        c.setLayout(new BorderLayout());

        x = 5;
        y = 3;


        p = new MyPanel();
        c.add(p);


        f.setSize(400,400);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        // creating a Timer class object, used for firing
        // one or more action events after a specified delay
        // Timer class constructor requires time in
        // milliseconds and object of class that handles
        // action events


        Timer t = new Timer (5, this);

        // starts the timer, causing it to start sending// action events to listeners
        t.start();

    } // end constructor

    // event handler method
    public void actionPerformed(ActionEvent ae){
        // if ball reached to maximum width of frame minus// 40 since diameter of ball is 40 then
        change the// X-direction of ball
        if (f.getWidth()-40 == p.mX)

                x = -5;

   // if ball reached to maximum height of frame
   // minus 40 then change the Y-direction of ball

   if (f.getHeight()-40 == p.mY)

           y = -3;


   // if ball reached to min. of width of frame,
   // change the X-direction of ball

   if (p.mX == 0 )

           x = 5;

   // if ball reached to min. of height of frame,
   // change the Y-direction of ball


   if (p.mY == 0 )

           y = 3;

   // Assign x,y direction to MyPanel's mX & mY
   p.mX += x;
   p.mY += y;

   // call to repaint() method so that ball is drawn on// new locations
   p.repaint();

} // end actionPerformed() method

 // main method
public static void main(String args[]){


     AnimTest at = new AnimTest();

}

} // end of AnimTest class
```

References:

- Java, A Lab Course by Umair Javed
- Painting in AWT & Swing
  http://java.sun.com/products/jfc/tsc/articles/painting/index.html
- Performing Custom Painting
  http://java.sun.com/docs/books/tutorial/uiswing/14painting/index.html

Lesson 20

Applets

- A small program written in Java and included in a HTML page.
- It is independent of the operating system on which it runs
- An applet is a Panel that allows interaction with a Java program
- A applet is typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet
- For security reasons, applets run in a sandbox: they have no access to the client's file system

Applets Support

- Most modern browsers support Java 1.4 if they have the appropriate plugin
- Sun provides an application appletviewerto view applets without using browser.
- In general you should try to write applets that can be run with any browser

What an Applet is?

- You write an applet by extending the class Appletor JApplet
- Applet is just a class like any other; you can even use it in applications if you want
- When you write an applet, you are only writing part of a program
- The browser supplies the main method

The genealogy of Applet

The following figure shows the inheritance hierarchy of the JApplet class. This hierarchy determines much of what an applet can do and how, as you'll see on the next few pages.

```
java.lang.Object
   |
   +----java.awt.Component

           |
           +----java.awt.Container|+----java.awt.Panel
                                  |
                                  +----java.applet.Applet|+----javax.swing.JApplet
```

Example Code 20.1: Writing a Simple Applet

Below is the source code for an applet called HelloApplet. This displays a "Hello World" string. Note that no main method has been provided.

```
// File HelloApplet.java
```

```
//step 1: importing required packagesimport java.awt.*;import
javax.swing.*;

// extending class from JApplet so that our class also becomes an//appletpublic class HelloApplet extends
JApplet {

    // overriding paint method
    public void paint(Graphics g) {


            // write code here u want to display & draw by using// Graphics objectg.drawString("Hello
            World", 30 , 30);

    } } // end class
```

After defining the HelloApplet.java, the next step is to write .html file. Below is the source code of Test.html file. The Test.html contains the ordinary html code except one.

```
          <html>
              <head>
              <title> Simple
              Applet </title>
              </head>


        <body>

                  <!-- providing the class name of applet with
                        width &height--!>

<applet code="HelloApplet.class"

width=150 height=100>

                 </applet>
        </body>
    </html>
```

Compile & Execute

By simply double clicking on Test.html file, you can view the applet in your browser. However, you can also use the appletviewer java program for executing or running applets.

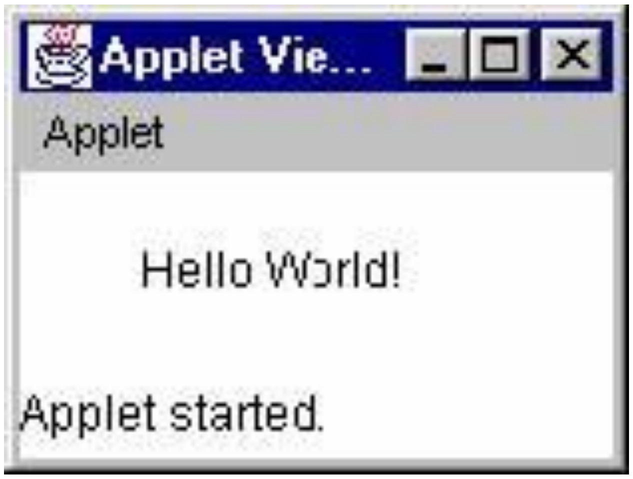
The applet viewer is invoked from the command line by the command

```
        appletviewer htmlfile
```

where htmlfile is the name of the file that contains the html document. For our example, the command looks like this:

appletviewer *Test.html*

As a result, you will see the following output



```
Hello World!
Applet started.
```

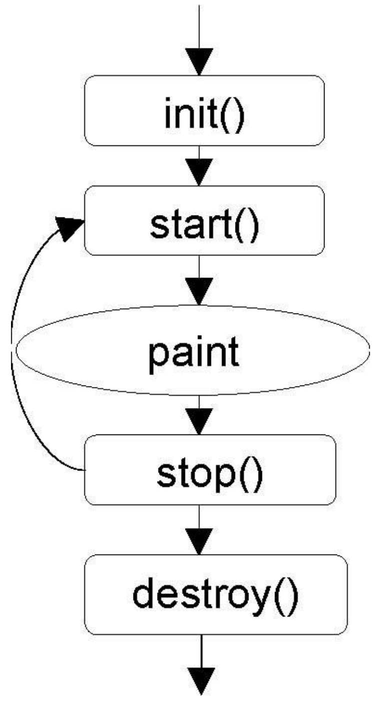
Applet Life Cycle Methods

When an applet is loaded, an instance of the applet's controlling class (an Applet subclass) is created. After that an applet passes through some stages or methods, each of them are build for specific purpose

An applet can react to major events in the following ways:

- It can **initialize** itself.
- It can **start** running.
- It can **stop** running.
- It can perform a **final cleanup**, in preparation for being unloaded

The applet's life cycle methods are called in the specific order shown below. Not every applet needs to override every one of these methods.



Let's take a look on each method in detail and find out what they do

init( )

- Is called only once.
- The purpose of init( ) is to initialize the applet each time it's loaded (or reloaded).
- You can think of it as a constructor

start( )
- To start the applet's execution
- For example, when the applet's loaded or when the user revisits a page that contains the applet
- start( ) is also called whenever the browser is maximized

paint( )
- paint( ) is called for the first time when the applet becomes visible
- Whenever applet needs to be repainted, paint( ) is called again
- Do all your painting in paint( ), or in a method that is called from paint( )

stop( )
- To stop the applet's execution, such as when the user leaves the applet's page or quits the browser.
- stop( ) is also called whenever the browser is minimized

destroy( )
- Is called only once.
- To perform a **final cleanup** in preparation for unloading

Example Code 20.2: Understanding Applet Life Cycle Methods

The following code example helps you in understanding the calling sequence of applet's life cycle methods.

These methods are only displaying debugging statements on the console.

```
// File AppletDemo.java

//step 1: importing required packagesimport java.awt.*;import
javax.swing.*;

// extending class from JApplet so that our class also becomes an//appletpublic

class AppletDemo extends JApplet {

   // overriding init method
   public void init ( ) {

         System.out.println("init() called");

   }

   // overriding start method
   public void start ( ){

         System.out.println("start() called");

   }

   // overriding paint method
   public void paint(Graphics g){

           System.out.println("paint() called");

   }

   // overriding stop method
   public void stop(){

       System.out.println("stop() called");

   }

   // overriding destroy method
   public void destroy(){

       System.out.println("destroy() called");

   }

} // end class
```

The DemoTest.html file is using this applet. The code snippet of it given below:

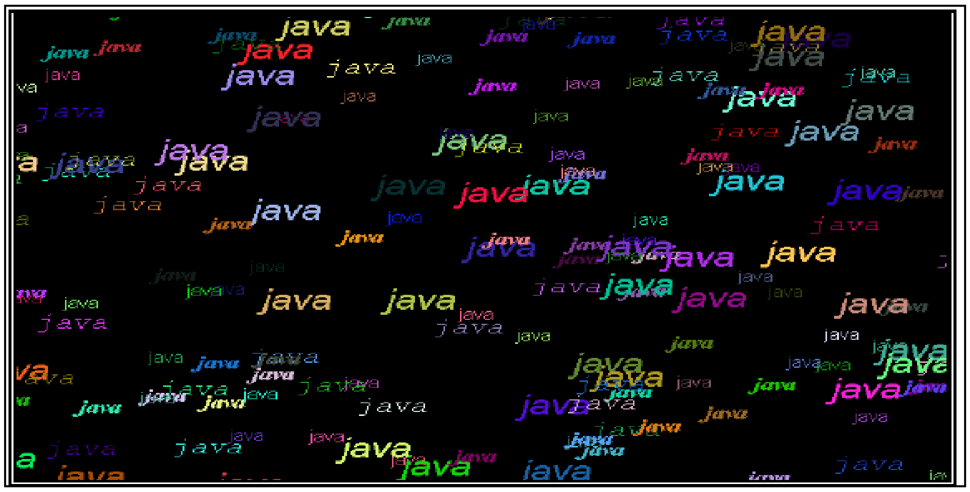
```
<html> <head> <title> Applet Life Cycle Methods
</title></head>

    <body>
```

```
<!-- providing the class name of applet with width &height--!>

<applet code="AppletDemo.class"

    width=150 height=100>

</applet></body></html>
```

Compile & Execute

To understand the calling sequence of applet life cycle methods, you have to execute it by using appletviewer command. Do experiments like maximizing, minimizing the applet, bringing another window in front of applet and keep an eye on console output.

Example Code 20.3: Animated Java Word Sample Output
The browser output of the program is given below:



Design Process

The Program in a single call of paint method

- Draws string "java" on 40 random locations
- For every drawing, it selects random font out of 4 different fonts
- For every drawing, it selects random color out of 256 * 256 * 256 RGB colors

Repaint is called after every 1000 ms.
After 10 calls to repaint, screen is cleared

Generating Random Numbers

- Use static method random of Math class
  Math.random() ;
- Returns positive double value greater than or equal to 0.0 or less than 1.0.
- Multiply the number with appropriate scaling factor to increase the range and type cast it, if needed.

  int i = (int)(Math.random() * 5);// will generate random numbers between 0 & 4.

Program's Modules

The program is build using many custom methods. Let's discuss each of them one by one that will help in understanding the overall logic of the program.

. **drawJava( )**

As name indicates, this method will be used to write String "java" on random locations. The code is given below:

```
// method drawJava

public void drawJava(Graphics2D g2) {


   // generate first number randomly. The panel width is 1000int x = (int) (Math.random() * 1000); //
   generate second number randomly. The panel height is 700

  int y = (int) (Math.random() * 700); // draw String on these randomly
  selected numebrsg2.drawString("java", x, y); }
```

. **chooseColor( )**

This method will choose color randomly out of 256 * 256 * 256 possible colors. The code snippet is given below:

```
// method chooseColor

public Color chooseColor() {

   // choosing red color value randomly


   int r = (int) (Math.random() * 255);

   // choosing green color value randomly

   int g = (int) (Math.random() * 255);

   // choosing blue color value randomly

   int b = (int) (Math.random() * 255);

   // constructing a color by providing R-G-B valuesColor c = new Color(r, g, b);

// returning color
return c;
```

}

. chooseFont( )

This method will choose a Font for text (java) to be displayed out of 4 available fonts. The code snippet is given below:

```
    // method chooseFont

    public Font chooseFont() {

        // generating a random value that helps in choosing a fontint fontChoice = (int)
        (Math.random() * 4) + 1;

        // declaring font reference
        Font f = null;


        // using switch based logic for selecting font
        switch (fontChoice) {


            case 1: f = new Font("Serif", Font.BOLD + Font.ITALIC, 20);break;
       case 2: f = new Font("SansSerif", Font.PLAIN, 17);break;

       case 3: f = new Font("Monospaced", Font.ITALIC, 23);break;

       case 4: f = new Font("Dialog", Font.ITALIC, 30);break;

      } // end switch

      // returns Font object
      return f;

      } //end chooseFont
```

. **paint( )**

The last method to be discussed here is paint( ). By overriding this method, we will print string "java" on 40 random locations. For every drawing, it selects random font out of 4 different fonts & random color out of 256 * 256 * 256 RGB colors.

Let's see, how it happens:

```
    // overriding method paint
      public void paint(Graphics g) {


        // incrementing clear counter variable.
        clearCounter++;
        // printing 40 “java” strings on different locations by// selcting random font & colorfor (int i =
        1; i <= 40; i++) {
            // choosing random color by calling chooseColor() method
            Color c = chooseColor();


            // setting color

            g2.setColor(c);


            // choosing random Font by calling chooseColor() method
            Font f = chooseFont();
            g2.setFont(f);


            // drawing string “java” by calling drawJava() method
            drawJava(g2);


        } // end for loop Graphics2D g2 = (Graphics2D) g;

// checking if paint is called 10 times then clears the// screen and set counter again to zero if
(clearCounter == 10) {

g2.clearRect(0, 0, 1000, 700);clearCounter = 0; } }

// end paint method
```

Merging Pieces

By inserting all method inside JavaAnim.java class, the program will look like one given below. Notice that it contains methods discussed above with some extra code with which you are already familiar.

```
   // File JavaAnim.java

   //step 1: importing required packages
   import java.awt.*;
   import java.awt.event.*;
   import javax.swing.*;
   public class JavaAnim extends JApplet implements ActionListener {

        // used to count how many times paint is called
       int clearCounter;
       // declaring Timer reference
       Timer t;
       // overriding init method, used to initialize variables

       public void init() {


           setBackground(Color.black);

           clearCounter = 0;

           Timer t = new Timer(1000, this);
           t.start();

           }

// overriding paint method – discussed above
public void paint(Graphics g) {

   clearCounter++;

   Graphics2D g2 = (Graphics2D) g;

   if (clearCounter == 10) {
       g2.clearRect(0, 0, 1000, 700);
       clearCounter = 0;

   }

   for (int i = 1; i <= 40; i++) {

       Color c = chooseColor();
       g2.setColor(c);

       Font f = chooseFont();
```

```
        g2.setFont(f);

        drawJava(g2);  }
                    }



// overriding actionPerformed()of ActionListener interface// called by Timer object
public void actionPerformed(ActionEvent ae) {

    repaint();

}


// chooseColor method – discussed above
public Color chooseColor() {

    int r = (int) (Math.random() * 255);
    int g = (int) (Math.random() * 255);
    int b = (int) (Math.random() * 255);


    Color c = new Color(r, g, b);
    return c;

}
    // chooseFont method – discussed above
    public Font chooseFont() {

        int fontChoice = (int) (Math.random() * 4) + 1;

        Font f = null;

        switch (fontChoice) {

            case 1: f = new Font("Serif", Font.BOLD + Font.ITALIC, 20);break;

            case 2: f = new Font("SansSerif", Font.PLAIN, 17);break;

            case 3: f = new Font("Monospaced", Font.ITALIC, 23);break;

            case 4: f = new Font("Dialog", Font.ITALIC, 30);break;

        }

        return f;
    }
    // drawJava() method – discussed above
    public void drawJava(Graphics2D g2) {

        int x = (int) (Math.random() * 1000);
        int y = (int) (Math.random() * 700);


        g2.drawString("java", x, y);

    }


} // end class
```

The AnimTest.html file is using this applet. The code snippet of it given below:

```
<html>

     <head>
          <title> Animated Java Word </title>
          </head>



   <body>

        <applet code="JavaAnim.class" width=1000 height=700> </applet>

   </body> </html>
```

Compile & Execute

You can execute it directly using browser or by using appletviewer application. For having fun, you can use "your name" instead of "java" and watch it in different colors ☺
References:

< . **Java, A Lab Course by Umair Javed**
< . **Writing Applets**

. http://java.sun.com/docs/books/tutorial/**applet/**

Lesson 21

Socket Programming

Socket

- A socket is one endpoint of a two-way communication link between two programs running generally on a network.
- A socket is a bi-directional communication channel between hosts. A computer on a network often termed as host.

Socket Dynamics

- As you have already worked with files, you know that file is an abstraction of your hard drive. Similarly you can think of a socket as an abstraction of the network.
- Each end has input stream (to send data) and output stream (to receive data) wired up to the other host.
- You store and retrieve data through files from hard drive, without knowing the actual dynamics of the hard drive. Similarly you send and receive data to and from network through socket, without actually going into underlying mechanics.
- You read and write data from/to a file using streams. To read and write data to socket, you will also use streams.

What is Port?

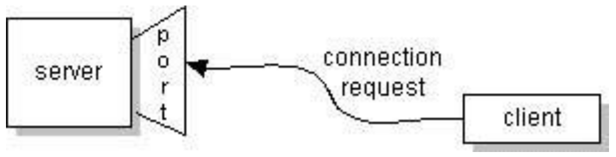
- It is a transport address to which processes can listen for connections request.
- There are different protocols available to communicate such as TCP and UDP. We will use TCP for programming in this handout.
- There are 64k ports available for TCP sockets and 64k ports available for UDP, so at least theoretically we can open 128k simultaneous connections.
- There are well-known ports which are
  - below 1024
  - provides standard services
  - Some well-known ports are:

    -FTP works on port 21

    -HTTP works on port 80 -TELNET works on port 23 etc.

How Client – Server Communicate

- Normally, a server runs on a specific computer and has a socket that is bound to a specific port number.
- The server just waits, listening to the socket for a client to make a connection request.
- On the client side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected.
- On the server side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- Note that the socket on the client side is not bound to the port number used to make contact with the server. Rather, the client is assigned a port number local to the machine on which the client is running.
- The client and server can now communicate by writing to or reading from their sockets



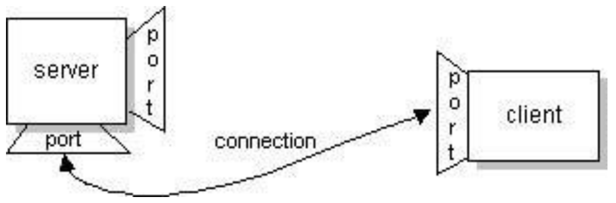
. As soon as client creates a socket that socket attempts to connect to the specified server.

. The server listens through a special kind of socket, which is named as server socket.

. The sole purpose of the server socket is to listen for incoming request; it is not used for communication.

. If every thing goes well, the server accepts the connection. Upon acceptance, the server gets a new socket, a communication socket, bound to a different port number.

. The server needs a new socket (and consequently a different port number) so that it can continue to listen through the original server socket for connection requests while tending to the needs of the connected client. This scheme is helpful when two or more clients try to connect to a server simultaneously (a very common scenario).



<u>Steps – To Make a Simple Client</u>

To make a client, process can be split into 5 steps. These are:

1. Import required package

You have to import two packages
. java.net.*;
. java.io.*;

2. Connect / Open a Socket with Server

Create a client socket (communication socket)
Socket s = new Socket("serverName", serverPort) ;

serverName : Name or address of the server you wanted to connect such as http://www.google.com or 172.2.4.98 etc. For testing if you are running client and server on the same machine then you can specify "localhost" as the name of server

· **<u>serverPort :</u>** Port number you want to connect to

The scheme is very similar to our home address and then phone number.

3. **Get I/O Streams of Socket**
   Get input & output streams connected to your socket

. For reading data from socket As stated above, a socket has input stream attached to it.

InputStream is = s.getInputStream();
// now to convert byte oriented stream into character oriented buffered reader // we use intermediary stream that helps in achieving above stated purpose
InputStreamReader isr= new InputStreamReader(is); BufferedReader br = new BufferedReader(isr);

. For writing data to socket

A socket has also output stream attached to it. Therefore,

OutputStream os = s.getOutputStream();

// now to convert byte oriented stream into character oriented print writer

// here we will not use any intermediary stream because PrintWriter constructor // directly accepts an object of OutputStream
PrintWriter pw = new PrintWriter(os, true);

Here notice that true is also passed to so that output buffer will flush.

4. **Send / Receive Message**
   Once you have the streams, sending or receiving messages isn't a big task. It's very much similar to the way you did with files
   . To send messages
   pw.println("hello world");
   . To read messages
   String recMsg = br.readLine();

5. **Close Socket**
   Don't forget to close the socket, when you finished your work
   s.close();

Steps – To Make a Simple Server

To make a server, process can be split into 7 steps. Most of these are similar to steps used in making a client. These are:

1. Import required package

You need the similar set of packages you have used in making of client

. java.net.*;
. java.io.*;

2. Create a Server Socket

In order to create a server socket, you will need to specify port no eventually on which server will listen for client requests.

ServerSocket ss = new ServerSocket(serverPort) ;

. serverPort: port local to the server i.e. a free port on the server machine. This is the same port number that is given in the client socket constructor

3. Wait for Incoming Connections

The job of the server socket is to listen for the incoming connections. This listening part is done through the accept method.

Socket s = ss.accept();

The server program blocks (stops) at the accept method and waits for the incoming client connection when a request for connection comes it opens a new communication socket (s) and use this socket to communicate with the client.

4. Get I/O Streams of Socket

Once you have the communication socket, getting I/O streams from communication socket is similar to the way did in making a client

1. For reading data from socket

InputStream is = s.getInputStream(); InputStreamReader isr= new

InputStreamReader(is); BufferedReader br = new BufferedReader(isr);

2. For writing data to socket

OutputStream os = s.getOutputStream();

PrintWriter pw = new PrintWriter(os, true);

5. Send / Receive Message

Sending and receiving messages is very similar as discussed in making of client

To send messages:

pw.println("hello world");

. To read messages

String recMsg = br.readLine();

6. Close Socket

```
s.close();
```

Example Code 21.1: Echo Server & Echo Client

The client will send its name to the server and server will append "hello" with the name send by the client. After that, server will send back the name with appended "hello".

EchoServer.java

Let's first see the code for the server

```
// step 1: importing required package

import java.net.*;

import java.io.*;

import javax.swing.*;



public class EchoServer

{

        public static void main(String args[])

        {

                try

                {

                //step 2: create a server socket

                ServerSocket ss = new ServerSocket(2222);

              System.out.println("Server started...");
          /* Loop back to the accept method of the serversocket and wait for a new connection request.
              Soserver will continuously listen for requests
          */
          while(true) {
                // step 3: wait for incoming connection

                Socket s = ss.accept();
        System.out.println("connection request recieved");
        // step 4: Get I/O streams
             InputStream is = s.getInputStream();InputStreamReader isr= new
             InputStreamReader(is);BufferedReader br = new BufferedReader(isr);
             OutputStream os = s.getOutputStream();
```

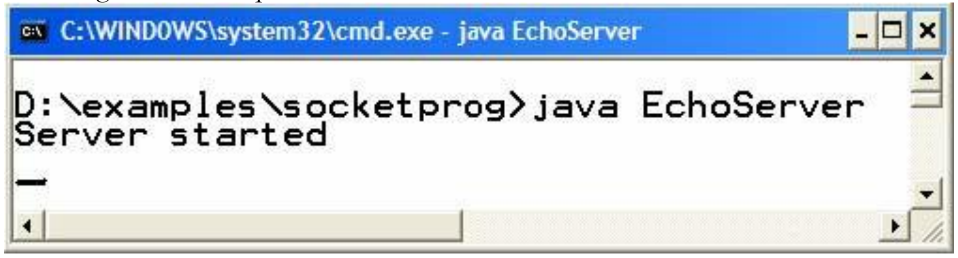
```
                PrintWriter pw = new PrintWriter(os,true);

                // step 5: Send / Receive message

                // reading name sent by clientString name = br.readLine(); // appending "hello" with the
                received name

                String msg = "Hello " + name + " from Server"; // sending back to client

                pw.println(msg);
                // closing communication sockeys.close();
            } // end while

        }catch(Exception ex){

        System.out.println(ex); } } } // end class
```

EchoClient.java
The code of the client is given below

```
// step 1: importing required package
import java.net.*;import java.io.*;import
javax.swing.*;
public class EchoClient{
     public static void main(String args[]){
        try {
           //step 2: create a communication socket
           // if your server will run on the same machine thenyou can pass "localhost" as server address
                */
           /* Notice that port no is similar to one passedwhile creating server socket */server
           Socket s = new Socket("localhost", 2222);
           // step 3: Get I/O streams
           InputStream is = s.getInputStream();InputStreamReader isr= new
           InputStreamReader(is);BufferedReader br = new BufferedReader(isr);

           OutputStream os = s.getOutputStream();PrintWriter pw = new
           PrintWriter(os,true);
           // step 4: Send / Receive message
           // asking use to enter his/her name
           String msg = JOptionPane.showInputDialog("Enter your name");
           // sending name to server
           pw.println(msg);
           // reading message (name appended with hello) from// servermsg = br.readLine();

           // displaying received messageJOptionPane.showMessageDialog(null , msg);
           // closing communication sockets.close();

       }catch(Exception ex){ System.out.println(ex); } } }

           // end class
```

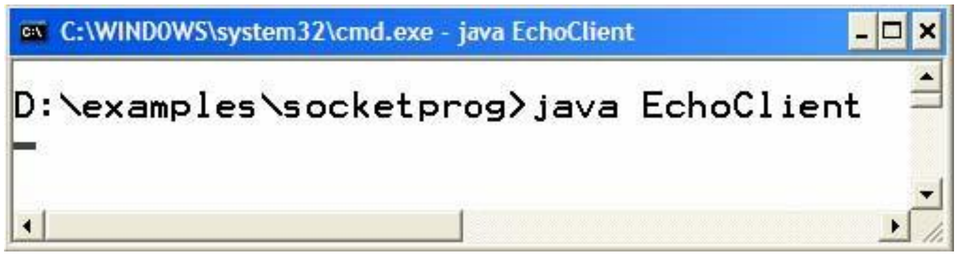
Compile & Execute

After compiling both files, run EchoServer.java first, from the command prompt window. You'll see a message of "server started" as shown in the figure below. Also notice that cursor is continuously blinking since server is waiting for client request

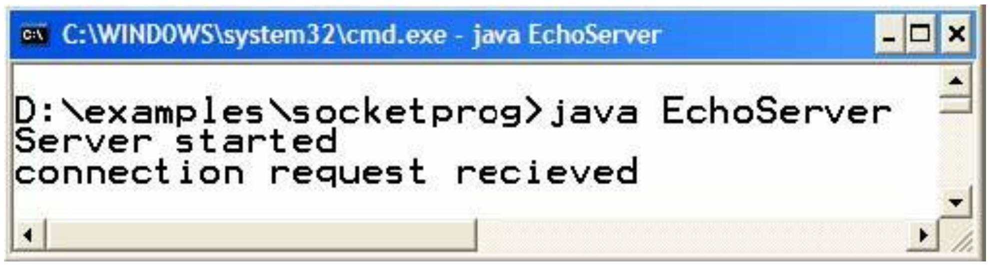

C:\WINDOWS\system32\cmd.exe - java EchoServer

```
D:\examples\socketprog>java EchoServer
Server started
_
```

Now, open another command prompt window and run EchoClient.java from it. Look at EchoServer window; you'll see the message of "request received". Sooner, the EchoClient program will ask you to enter name in input dialog box. After entering name press ok button, with in no time, a message dialog box will pop up containing your name with appended "hello" from server. This whole process is illustrated below in pictorial form:

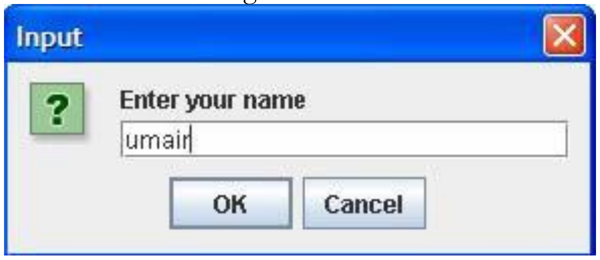

C:\WINDOWS\system32\cmd.exe - java EchoClient

```
D:\examples\socketprog>java EchoClient
_
```

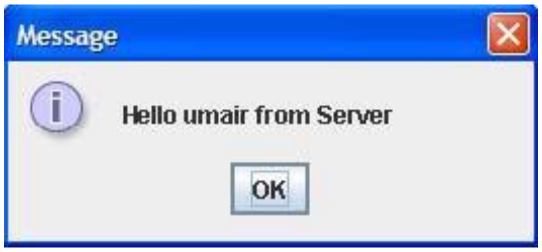

C:\WINDOWS\system32\cmd.exe - java EchoServer

```
D:\examples\socketprog>java EchoServer
Server started
connection request recieved
```

sending name to server



response from server



Hello umair from Server

Notice that server is still running, you can run again EchoClient.java as many times untill server is running. To have more fun, run the server on a different computer and client on a different. But before doing that find the IP of the computer machine on which your EchoServer will eventually run. Replace "localhost" with the new IP and start conversion over network ☺

References

Entire material for this handout is taken from the book **JAVA A Lab Course** by **Umair Javed**. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Serialization

What?

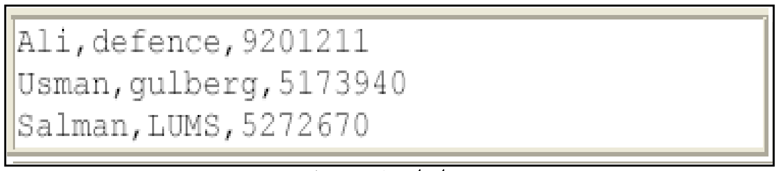
. You want to send an object to a stream.

Motivation

- A lot of code involves boring conversion from a file to memory
- As you might recall that AddressBook program reads data from file and then parses it
- This is a common problem

Revisiting AddressBook

We read record from a text file named persons.txt. The person record was present in the file in the following format.



```
Ali,defence,9201211
Usman,gulberg,5173940
Salman,LUMS,5272670
```

persons.txt

The code that was used to construct Person objects after reading information from the file is given below. Here only the part of code is shown, for complete listing, see AddressBook code in your earlier handout.

```
FileReader fr = new FileReader("persons.txt");BufferedReader br =
new BufferedReader(fr);
String line = br.readLine();

while ( line != null ) {

        tokens = line.split(",");
        name = tokens[0];

        add = tokens[1];

        ph = tokens[2];


        PersonInfo p = new PersonInfo(name, add, ph); // you can add p into arraylist, if
        needed line = br.readLine();
}
```

As you have seen a lot of parsing code is required for converting a line into PersonInfo objects. Serialization mechanism eases developer's life by achieving all above in a very simple way.

Serialization in Java

- Java provides an extensive support for serialization
- Object knows how to read or write themselves to streams
- Problem:
- As you know, objects get created on heap and have some values therefore Objects have some state in memory
- You need to save and restore that state.
- The good news is that java serialization takes care of it automatically

Serializable Interface

- By implementing this interface a class declares that it is willing to be read/written by automatic serialization machinery
- Found in java.iopackage
- Tagging interface – has no methods and serves only to identify the semantics of being serializable

Automatic Writing

- System knows how to recursively write out the state of an object to stream
- If an object has the reference of another object, the java serialization mechanism takes care of it and writes it too.

Automatic Reading

- System knows how to read the data from Stream and re-create object in memory
- The recreated object is of type "Object" therefore Down-casting is required to convert it into actual type.

Serialization: How it works?

- To write an object of PersonInfo, ObejctOutputStream and its method writeObject( ) will be used

```
PersonInfo p = new PersonInfo( );
ObejctOutputStream out;
// writing PersonInfo's object p
out.writeObject(p);
```

- To read that object back, ObejctInputStream and its method readObject()will be used

```
ObejctInputStream in;
// reading PersonInfo's object. Remember type casting// is required
  PersonInfo obj = (PersonInfo)in.readObject( );
```

Example Code 22.1: Reading / Writing PersonInfo objects

We want to send PersonInfo object to stream. You have already seen this class number of times before. Here it will also implement serializable interface.

PersonInfo.java

```
import javax.swing.*;
import java.io.*
   class PersonInfo implements Serializable{
         String name;
         String address;
         String phoneNum;

         //parameterized constructorpublic PresonInfo(String n, String a, String p) {
              name = n;
              address = a;
              phoneNm = p;

         }
         //method for displaying person record on GUIpublic void printPersonInfo( ) {
                                 JOptionPane.showMessageDialog(null,

                        "name: " + name + "address:" +address +

                        "phone no:" + phoneNum);


         }
  } // end class
```

WriteEx.java
The following class will serialize PersonInfo object to a file

```
import java.io*;
public class WriteEx{

   public static void main(String args[ ]){
      PersonInfo pWrite =new PersonInfo("ali", "defence", "9201211");
      try {
            // attaching FileOutput stream with "ali.dat"

            FileOutputStream fos =new FileOutputStream("ali.dat");
            // attaching ObjectOutput stream over FileOutput// stream

            ObjectOutputStream out =new ObjectOutputStream(fos);
            //serialization
            // writing object to 'ali.dat'

            out.writeObject(pWrite);
            // closing streams
            out.close();
            fos.close();

      } catch (Exception ex){

           System.out.println(ex)
```

```
        }
} // end class
```

ReadEx.java
The following class will read serialized object of PersonInfo from file i.e "ali.data"

```
    import java.io*; public class ReadEx{ public

    static void main(String args[ ]){ try { //

    attaching FileInput stream with "ali.dat"

        FileInputStream fin = new FileInputStream("ali.dat");

        // attaching FileInput stream over ObjectInput stream

        ObjectInputStream in = new ObjectInputStream(fis);
        //de-serialization
        // reading object from 'ali.dat'

          PersonInfo pRead = (PersoInfo)in.ReadObject( );
        // calling printPersonInfo method to confirm that// object contains same set of values
        before// serializatoion
        pRead.printPersonInfo();
      // closing streams
      in.close();
      fis.close();


     } catch (Exception ex){

           System.out.println(ex)

           }

} // end class
```

Compile & Execute
After compilation, first run the WriteEx.java file and visit the "ali.dat" file. Then run ReadEx.java from different command or same command prompt.

Object Serialization & Network

. You can read / write to a network using sockets
. All you need to do is attach your stream with socket rather than file
. The class version should be same on both sides (client & network) of the network

Example Code 22.2: Sending/Reading Objects to/from Network

We are going to use same PersonInfo class listed in example code 22.1. An object of PersonInfo class will be sent by client on network using sockets and then be read by server from network.

Sending Objects over Network

The following class ClientWriteNetEx.java will send an object on network

```
import java.net.*;import java.io.*;import
javax.swing.*;

    public class ClientWriteNetEx{ public static void

       main(String args[]){ try { PersonInfo p = new

       PersonInfo("ali", "defence", "9201211"); // create a

       communication socket

Socket s = new Socket("localhost", 2222);

      // Get I/O streams

      OutputStream is = s.getOutputStream();

// attaching ObjectOutput stream over Input stream

ObjectOutputStream oos= new ObjectOutputStream(is);

      // writing object to network

      oos.write(p);
      // closing communication socket
      s.close();

   }catch(Exception ex){

      System.out.println(ex);

      }

      } }// end class
```

Reading Objects over Network

The following class ServerReadNetEx.java will read an object of PersonInfo sent by client.

```
import java.net.*;import java.io.*;import
javax.swing.*;
```

```
    public class ServerReadNetEx{

    public static void main(String rgs[])

    {

    try

    {

        // create a server socket

ServerSocket ss = new ServerSocket(2222);

System.out.println("Server started...");
        /* Loop back to the accept method of the serversocket and wait for a new connection request.
            Soserver will continuously listen for requests
        */
        while(true) {
// wait for incoming connection

Socket s = ss.accept();
System.out.println("connection request recieved");
// Get I/O streams
InputStream is = s.getInputStream();
// attaching ObjectOutput stream over Input stream
ObjectInputStream ois = new ObjectInputStream(is);
// read PersonInfo object from network

     PersonInfo p = (PersonInfo)ois.read( );

          p.printPersonInfo();

         // closing communication socket

         s.close();

         } // end while

       }catch(Exception ex){ System.out.println(ex); } } } // end class
```

Compile & Execute

After compiling both files, run ServerReadNetEx.java first, from the command prompt window. Open another command prompt window and run ClientWriteNetEx.javafrom it.
The ClientWriteNetEx.java will send an Object of PersonInfo to ServerReadNetEx.java that displays that object values in dialog box after reading it from network.

Preventing Serialization

- Often there is no need to serialize sockets, streams & DB connections etc because they do no represent the state of object, rather connections to external resources
- To do so, **transient** keyword is used to mark a field that should not be serialized
- So we can mark them as,
  - o **transient** Socket s;
  - o **transient** OutputStream os;
  - o **transient** Connecction con;
- Transient fields are returned as nullon reading

Example Code 22 . 3: transient

Assume that we do not want to serialize phoneNum attribute of PersonInfo class, this can be done as shown below

PersonInfo.java

```
import javax.swing.*;
import java.io.*
   class PersonInfo implements Serializable{
              String name;
              String address;
              transient String phoneNum;

         public PresonInfo(String n, String a, String p) { name = n;address = a;phoneNm = p;
         }
         public void printPersonInfo( ) {
        JOptionPane.showMessageDialog(null , "name: " + name + "address:" +address +
                      "phone no:" + phoneNum);
         }
  } // end class
```

References

Entire material for this handout is taken from the book **JAVA A Lab Course** by **Umair Javed**. This material is available just for the use of VU students of the course Web Design and Development and not for any other commercial purpose without the consent of author.

Lesson 23

Multithreading

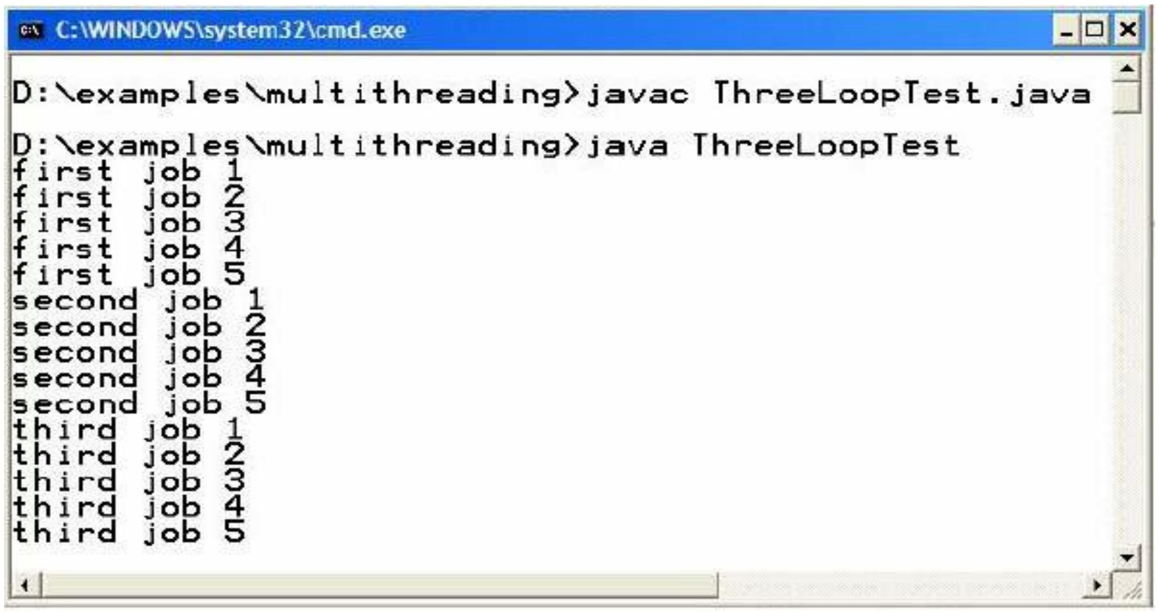
Introduction

Multithreading is the ability to do multiple things at once with in the same application. It provides finer granularity of concurrency. A thread — sometimes called an execution context or a lightweight process — is a single sequential flow of control within a program.
Threads are light weight as compared to processes because they take fewer resources then a process. A thread is easy to create and destroy. Threads share the same address space
i.e. multiple threads can share the memory variables directly, and therefore may require more complex synchronization logic to avoid deadlocks and starvation.

Sequential Execution vs Multithreading

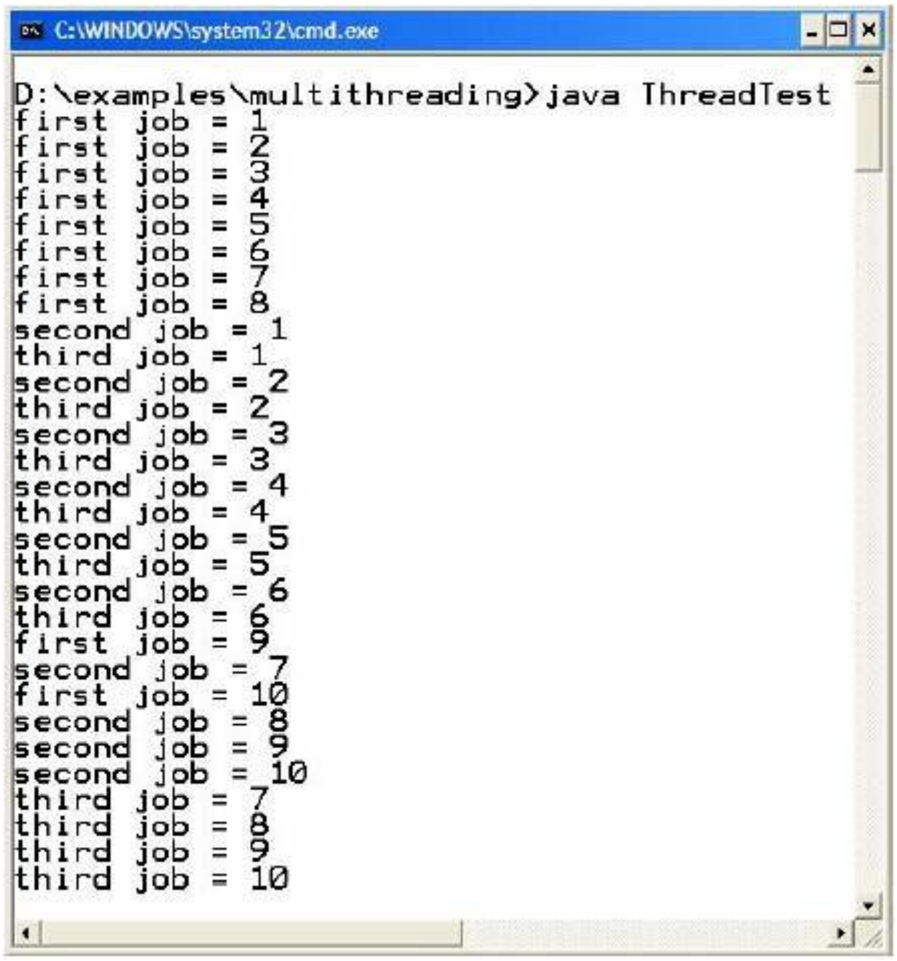
Every program has atleast one thread. Programs without multithreading executes sequentially. That is, after executing one instruction the next instruction in sequence is executed. If a function is called then until the completion of the function the next instruction is not executed. Similarly if there is a loop then instructions after loop only gets executed when the loop gets completed. Consider the following java program having three loops in it.

```
// File ThreeLoopTest.java
public class ThreeLoopTest {
          public static void main (String args[ ]) {
            //first loop
            for (int i=1; i<= 5; i++)System.out.println("first " +i);
            // second loop
            for (int j=1; j<= 5; j++)System.out.println("second " + j);
            // third loop
            for (int k=1; k<= 5; k++)System.out.println("third " + k);
     } // end main} // end class
```



```
C:\WINDOWS\system32\cmd.exe

D:\examples\multithreading>javac ThreeLoopTest.java

D:\examples\multithreading>java ThreeLoopTest
first job 1
first job 2
first job 3
first job 4
first job 5
second job 1
second job 2
second job 3
second job 4
second job 5
third job 1
third job 2
third job 3
third job 4
third job 5
```

Note: Each loop has 5 iterations in the ThreeLoopTest program.



```
D:\examples\multithreading>java ThreadTest
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
second job = 1
third job = 1
second job = 2
third job = 2
second job = 3
third job = 3
second job = 4
third job = 4
second job = 5
third job = 5
second job = 6
third job = 6
first job = 9
second job = 7
first job = 10
second job = 8
second job = 9
second job = 10
third job = 7
third job = 8
third job = 9
third job = 10
```

Note: Each loop has 10 iterations in the ThreadTest program. Your output can be different from the one given above.

Notice the difference between the outputs of the two programs. In ThreeLoopTest each loop generated a sequential output while in ThreadTest the output of the loops got intermingled i.e. concurrency took place and loops executed simultaneously

Let us code our first multithreaded program and try to learn how Java supports multithreading.

Java includes built-in support for threading. While other languages have threads bolted-on to an existing structure. i.e. threads were not the part of the original language but latter came into existence as the need arose.

All well known operating systems these days support multithreading. JVM transparently maps Java Threads to their counter-parts in the operating system i.e. OS Threads. JVM allows threads in Java to take advantage of hardware and operating system level advancements. It keeps track of threads and schedules them to get CPU time. Scheduling may be pre-emptive or cooperative. So it is the job of JVM to manage different tasks of thread. Let's see how we can create threads?

Creating Threads in Java

There are two approaches to create threads in Java.

Using Interface
Using Inheritance

Following are the steps to create threads by using Interface:

1 Create a class where you want to put some code that can run in parallel with some other code and let that class implement the Runnable interface
2 Runnable interface has the run() method therefore provide the implementation for the run() method and put your code that you want to run in parallel here
3 Instantiate Thread class object by passing Runnable object in constructor
4 Start thread by calling start() method

Following are the steps to create threads by using Intheritance:

1 Inherit a class from java.lang.Thread class
2 Override the run() method in the subclass
3 Instantiate the object of the subclass
4 Start thread by calling start() method

To write a multithreaded program using Runnable interface, follow these steps:

- **Step 1 - Implement the Runnable Interface**
  class Worker implements Runnable
- **Step 2 - Provide an Implementation of run() method**
  public void run( ){// write thread behavior// code that will be executed by the thread
- **Step 3 - Instantiate Thread class object by passing Runnable object in the constructor**
  Worker w = new Worker(“first”);
  Thread t = new Thread (w);

- **Step 4 – Start thread by calling start() method**
  t.start();

Threads Creation Steps Using Inheritance

To write a multithreaded program using inheritance from Thread class, follow these steps:

Step 1 – Inherit from Thread Class

class Worker extends Thread

Step 2 - Override run() method

```
public void run( ){
// write thread behavior
// code that will execute by thread
```

Step 3 - Instantiate subclass object
Step 4 – Start thread by calling start() method

```
Worker w = new Worker(“first”);
t.start();
```

So far we have explored:

- What is multithreading?
- What are Java Threads?
- Two ways to write multithreaded Java programs

Now we will re-write the ThreeLoopTest program by using Java Threads. At first we will use the Interface approach and then we will use Inheritance.

Code Example using Interface

```
// File Worker.java
public class Worker implements Runnable {
    private String job ;
      //Constructor of Worker class
      public Worker (String j ){
      job = j;
      }


//Implement run() method of Runnable interface
    public void run ( ) {
           for(int i=1; i<= 10; i++)System.out.println(job + " = " + i);
    }
} // end class


// File ThreadTest.java
public class ThreadTest{
    public static void main (String args[ ]){
           //instantiate three objects
           Worker first = new Worker ("first job");
           Worker second = new Worker ("second job");
           Worker third = new Worker ("third job");

           //create three objects of Thread class & passing worker
           //(runnable) to them
           Thread t1 = new Thread (first );
           Thread t2 = new Thread (second);
           Thread t3 = new Thread (third);

           //start threads to execute
           t1.start();
           t2.start();
           t3.start();

    }//end main} // end class
```

Following code is similar to the code given above, but uses Inheritance instead of interface

```
// File Worker.java
public class Worker extends Thread{
    private String job ;
      //Constructor of Worker class
      public Worker (String j ){
      job = j;
      }


//Override run() method of Thread class
    public void run ( ) {for(int i=1; i<= 10; i++)System.out.println(job + " = " + i);
    }
} // end class

// File ThreadTest.java

public class ThreadTest{
    public static void main (String args[ ]) {
        //instantiate three objects of Worker (Worker class is now
//becomes a Thread because it is inheriting from it)class
Worker first = new Worker ("first job");
Worker second = new Worker ("second job");
Worker third = new Worker ("third job");
//start threads to execute
t1.start();
t2.start();
t3.start();
}//end main} // end class
```

Threads provide a way to write concurrent programs. But on a single CPU, all the threads do not run simultaneously. JVM assigns threads to the CPU based on thread priorities. Threads with higher priority are executed in preference to threads with lower priority. A thread's default priority is same as that of the creating thread i.e. parent thread.
A Thread's priority can be any integer between 1 and 10. We can also use the following predefined constants to assign priorities.

```
Thread.MAX_PRIORITY (typically 10)
Thread.NORM_PRIORITY (typically 5)
Thread.MIN_PRIORITY (typically 1)
```

To change the priority of a thread, we can use the following method

```
setPriority(int priority)
```

It changes the priority of this thread to integer value that is passed. It throws an IllegalArgumentException if the priority is not in the range MIN_PRIORITY to MAX_PRIORITY i.e. (1–10).
For example, we can write the following code to change a thread's priority.

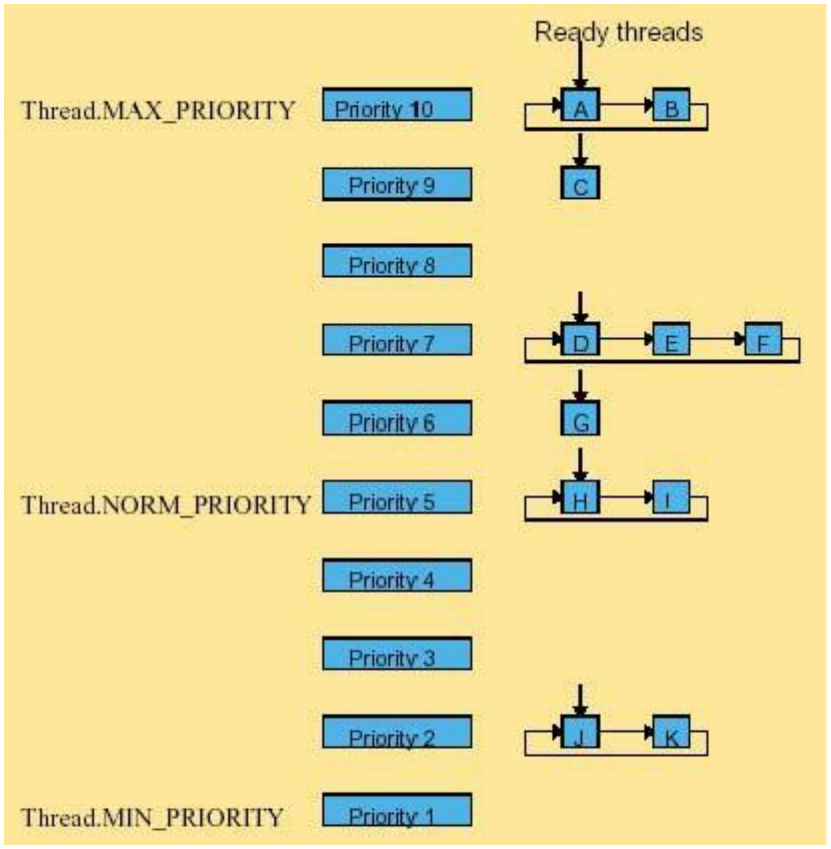
```
Thread t = new Thread (RunnableObject);
// by using predefined constantt.setPriority
(Thread.MAX_PRIORITY);
// by using integer constantt.setPriority (7);
```

Thread Priority Scheduling

The Java runtime environment supports a very simple, deterministic scheduling algorithm called fixed-priority scheduling. This algorithm schedules threads on the basis of their priority relative to other Runnable threads.
At any given time, when multiple threads are ready to be executed, the runtime system chooses for execution the Runnable thread that has the highest priority. Only when that thread stops, yields (will be explained later), or becomes Not Runnable will a lower-priority thread start executing. If two threads of the same priority are waiting for the CPU, the scheduler arbitrarily chooses one of them to run. The chosen thread runs until one of the following conditions becomes true:

- A higher priority thread becomes Runnable.
- It yields, or its run() method exits.
- On systems that support time-slicing, its time allotment has expired.

Then the second thread is given a chance to run, and so on, until the interpreter exits. Consider the following figure in which threads of various priorities are represented by capital alphabets A, B, …, K. A and B have same priority (highest in this case). J and K have same priority (lowest in this case). JVM start executing with A and B, and divides CPU time between these two threads arbitrarily. When both A and B comes to an end, it chooses the next thread C to execute.

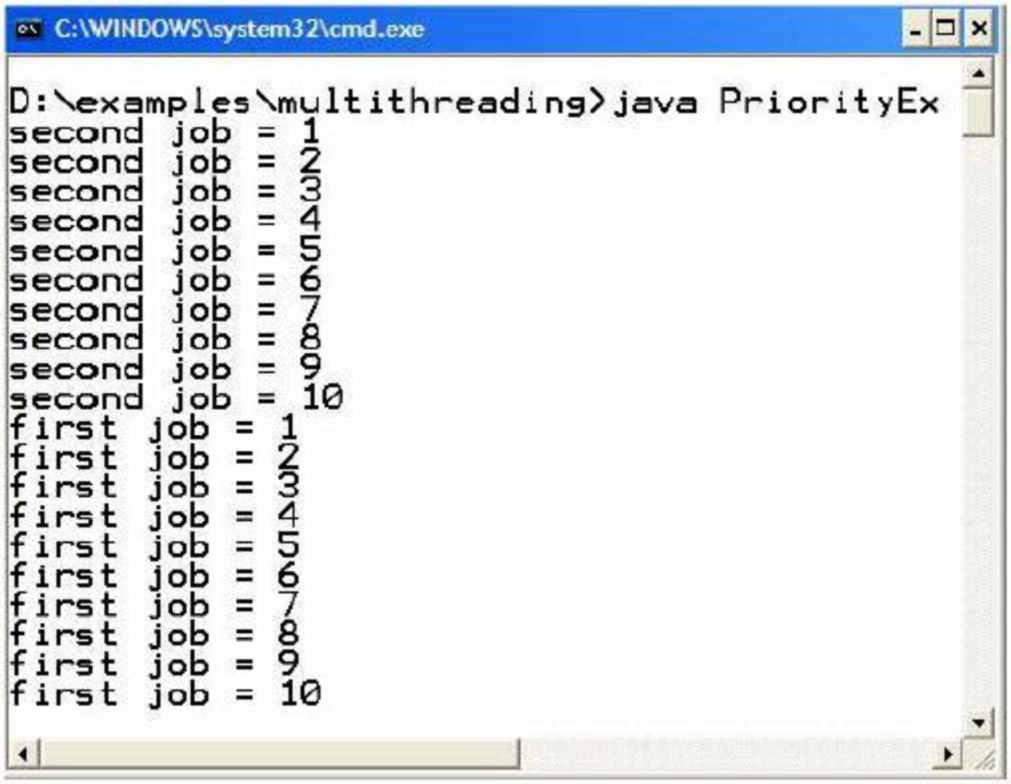


Code Example: Thread Properties

Try following example to understand how JVM executes threads based on their priorities.

```
// File PriorityEx.java
public class PriorityEx{ public static void main (String args[ ]){
        //instantiate two objects
        Worker first = new Worker ("first job");
        Worker second = new Worker ("second job");

        //create two objects
        Thread t1 = new Thread (first );
        Thread t2 = new Thread (second);

        //set thread priorities
        t1.setPriority (Thread.MIN_PRIORITY);
        t2.setPriority (Thread.MAX_PRIORITY);

        //start threads to execute
        t1.start();
        t2.start();
}//end main} // end class
```

Output



```
D:\examples\multithreading>java PriorityEx
second job = 1
second job = 2
second job = 3
second job = 4
second job = 5
second job = 6
second job = 7
second job = 8
second job = 9
second job = 10
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
first job = 9
first job = 10
```

Problems with Thread Priorities

However, when using priorities with Java Threads, remember the following two issues:
First a Java thread priority may map differently to the thread priorities of the underlying OS. It is because of difference in priority levels of JVM and underlying OS. For example

- Solaris has 2^{32} –1 priority levels
- Windows NT has only 7 user priority levels

Second, starvation can occur for lower-priority threads if the higher-priority threads never terminate, sleep, or wait for I/O indefinitely.

References:

- Java, A Practical Guide by Umair Javed
- Java How to Program by Deitel and Deitel
- CS193j handouts on Stanford

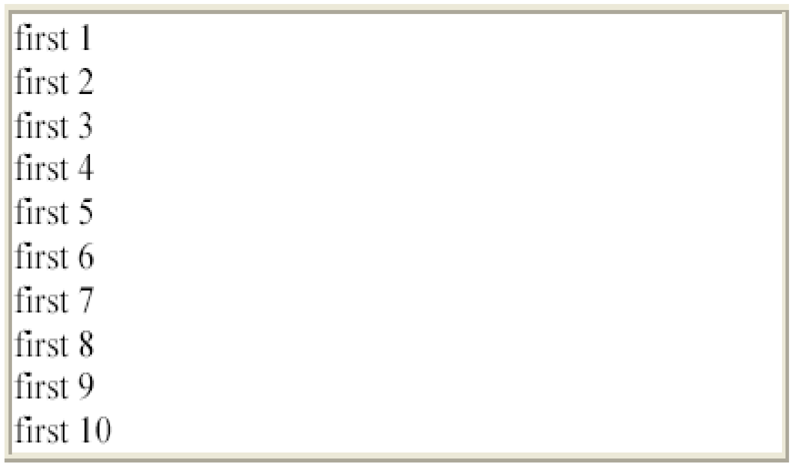
Lesson 24

More on Multithreading

In this handout, we'll cover different aspects of multithreading. Some examples are given to make you understand the topic of multithreading. First we will start with an example that reads data from two text files simultaneously.

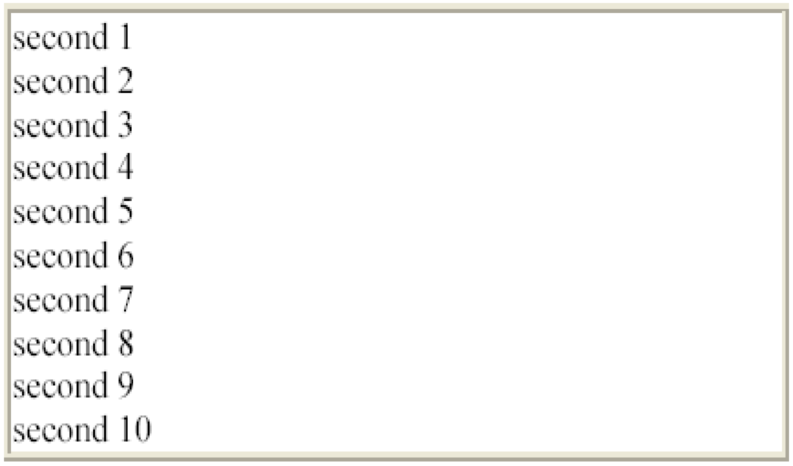
Example Code: Reading Two Files Simultaneously

The task is to read data from file "first.txt" & "second.txt" simultaneously. Suppose that files contains the following data as shown below



```
first 1
first 2
first 3
first 4
first 5
first 6
first 7
first 8
first 9
first 10
```

first.txt



```
second 1
second 2
second 3
second 4
second 5
second 6
second 7
second 8
second 9
second 10
```

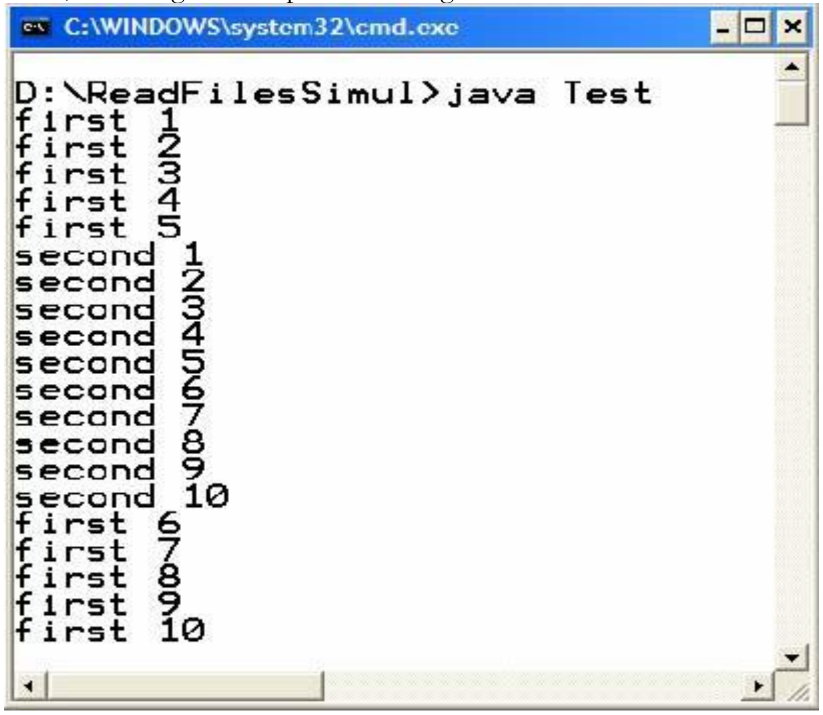
second.txt

Following is the code for ReadFile.java that implements Runable interface. The file reading code will be written inside the run( ) method.

```
// File ReadFile.java
import java.io.*;
public class ReadFile implements Runnable{
    //attribute used for name of file
    String fileName;

      // param constructor
      public ReadFile(String fn){
      fileName = fn;
      }

    // overriding run method// this method contains the code for file reading
    public void run ( ){
      try
      { // connecting FileReader with attribute fileNameFileReader fr = new
          FileReader(fileName);BufferedReader br = new BufferedReader(fr);
          String line = "";
          // reading line by line data from file// and displaying it on console
          line = br.readLine();
          while(line != null) {
              System.out.println(line);
              line = br.readLine();

          }
        fr.close();
        br.close();

        }catch (Exception e){
            System.out.println(e);
            }

    } // end run() method
} // File Test.java
public class Test {
    public static void main (String args[]){
        // creating ReadFile objects by passing file names to them
        ReadFile first = new ReadFile("first.txt");
        ReadFile second = new ReadFile("second.txt");

        // Instantiating thread objects and passing
        // runnable (ReadFile) objects to them
        Thread t1 = new Thread(first);
        Thread t2 = new Thread(second);

        // starting threads that cause threads to read data from
        // two different files simultaneously
        t1.start();
        t2.start();

    }

    }
```

Output

On executing Test class, following kind output would be generated:



C:\WINDOWS\system32\cmd.exe

```
D:\ReadFilesSimul>java Test
first 1
first 2
first 3
first 4
first 5
second 1
second 2
second 3
second 4
second 5
second 6
second 7
second 8
second 9
second 10
first 6
first 7
first 8
first 9
first 10
```

Now let's discuss some useful thread class methods.

. **sleep(int time) method**

-Causes the currently executing thread to wait for the time (milliseconds) specified

-Waiting is efficient equivalent to non-busy. The waiting thread will not occupy the processor

-Threads come out of the sleep when the specified time interval expires or when interrupted by some other thread

-Thread coming out of sleep may go to the running or ready state depending upon the availability of the processor. The different states of threads will be discussed later

-High priority threads should execute sleep method after some time to give low priority threads a chance to run otherwise starvation may occur

-sleep() method can be used for delay purpose i.e. anyone cal call Thread.sleep()method

-Note that sleep() method can throw InterruptedException. So, you need try-catch block

Example Code: Demonstrating sleep ( ) usage

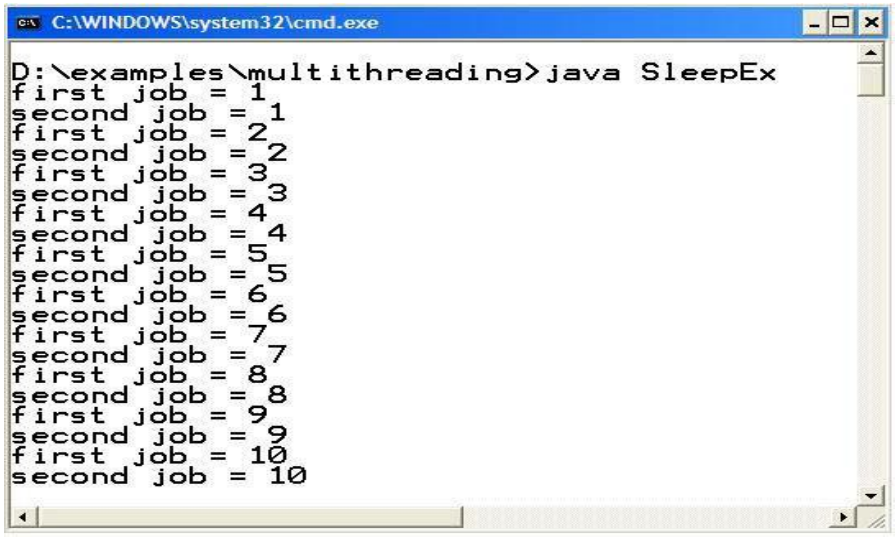
Below the modified code of Worker.java is given that we used in the previous handout.

```
// File Worker.javapublic class Worker implements Runnable {
private String job ;
    //Constructor of Worker class
    public Worker (String j ){
    job = j;
    }

     //Implement run() method of Runnable interface
     public void run ( ) {
     for(int i=1; i<= 10; i++) {

                    try { Thread.sleep(100); // go to sleep for 100
```

```
                        ms}catch (Exception
                        ex){System.out.println(ex);}
             System.out.println(job + " = " + i);} // end for} //
        end run } // end class
// File SleepEx.java
public class SleepEx {
    public static void main (String args[ ]){
        // Creating Worker objects
        Worker first = new Worker ("first job");
        Worker second = new Worker ("second job");

        // Instantiating thread class objects
        Thread t1 = new Thread (first );
        Thread t2 = new Thread (second);

        // starting thread
        t1.start();
        t2.start();

    }
} // end class
```

Output

On executing SleepEx.java, the output will be produced with exact alternations between first thread & second thread. On starting threads, first thread will go to sleep for 100 ms. It gives a chance to second thread to execute. Later this thread will also go to sleep for 100 ms. In the mean time the first thread will come out of sleep and got a chance on processor. It will print job on console and again enters into sleep state and this cycle goes on until both threads finished the run() method



```
D:\examples\multithreading>java SleepEx
first job = 1
second job = 1
first job = 2
second job = 2
first job = 3
second job = 3
first job = 4
second job = 4
first job = 5
second job = 5
first job = 6
second job = 6
first job = 7
second job = 7
first job = 8
second job = 8
first job = 9
second job = 9
first job = 10
second job = 10
```

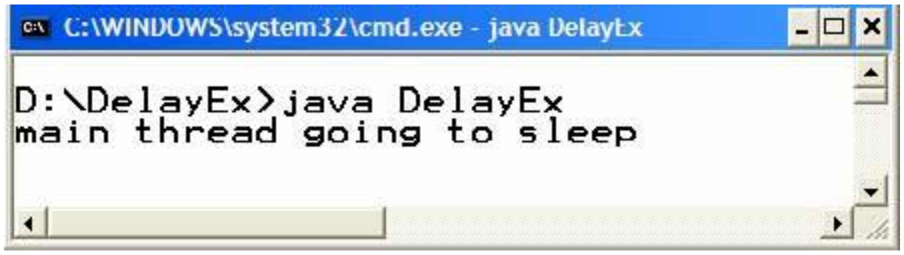
Before jumping on to example code, lets reveal another aspect about main() method. When you run a Java program, the VM creates a new thread and then sends the main(String[] args) message to the class to be run! Therefore, there is always at least one running thread in existence. However, we can create more threads

which can run concurrently with the existing default thread.
sleep() method can be used for delay purpose. This is demonstrated in the DelayEx.java given below

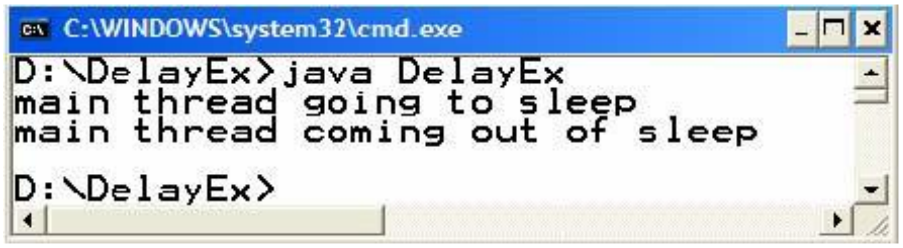
```
    // File DelayEx.java public class DelayEx { public static
    void main (String args[ ]) { System.out.println("main thread
    going to sleep");
         try {
// the main thread will go to sleep causing delay
                        Thread.sleep(100);
              }catch (Exception ex){
              System.out.println(ex)
              ;
              }

      System.out.println("main thread coming out of sleep"); } // end main()
} // end class
```

Output

On executing DelayEx class, you will experience a delay after the first statement displayed. The second statement will print when the time interval expired. This has been show below in the following two diagrams:



```
D:\DelayEx>java DelayEx
main thread going to sleep
```



```
D:\DelayEx>java DelayEx
main thread going to sleep
main thread coming out of sleep

D:\DelayEx>
```

. **yield( ) method**

-Allows any other threads of the same priority to execute (moves itself to the end of the priority queue)
-If all waiting threads have a lower priority, then the yielding thread resumes execution on the CPU
-Generally used in cooperative scheduling schemes

Example Code: Demonstrating yield ( ) usage

Below the modified code of Worker.javais given

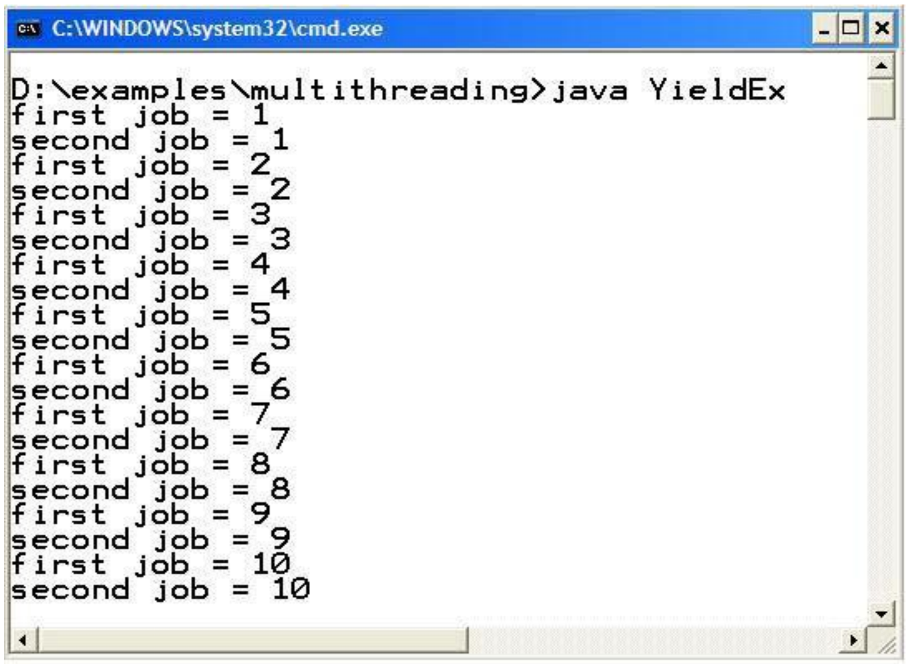
```
 // File Worker.javapublic class Worker implements
Runnable { private String job ;
```

```
//Constructor of Worker class
```

```
        public Worker (String j ){
        job = j;
        }

         //Implement run() method of Runnable interface
         public void run ( ) {
         for(int i=1; i<= 10; i++) {


// giving chance to a thread to execute of same priority

Thread.yield( );
System.out.println(job + " = " + i);
             } // end for
             } // end run
} // end class // File YieldEx.java
public class YieldEx {
    public static void main (String args[ ]){
         // Creating Worker objects
         Worker first = new Worker ("first job");
         Worker second = new Worker ("second job");

         // Instantiating thread class objects
         Thread t1 = new Thread (first );
         Thread t2 = new Thread (second);

         // starting thread
         t1.start();
         t2.start();

    }
} // end class
```

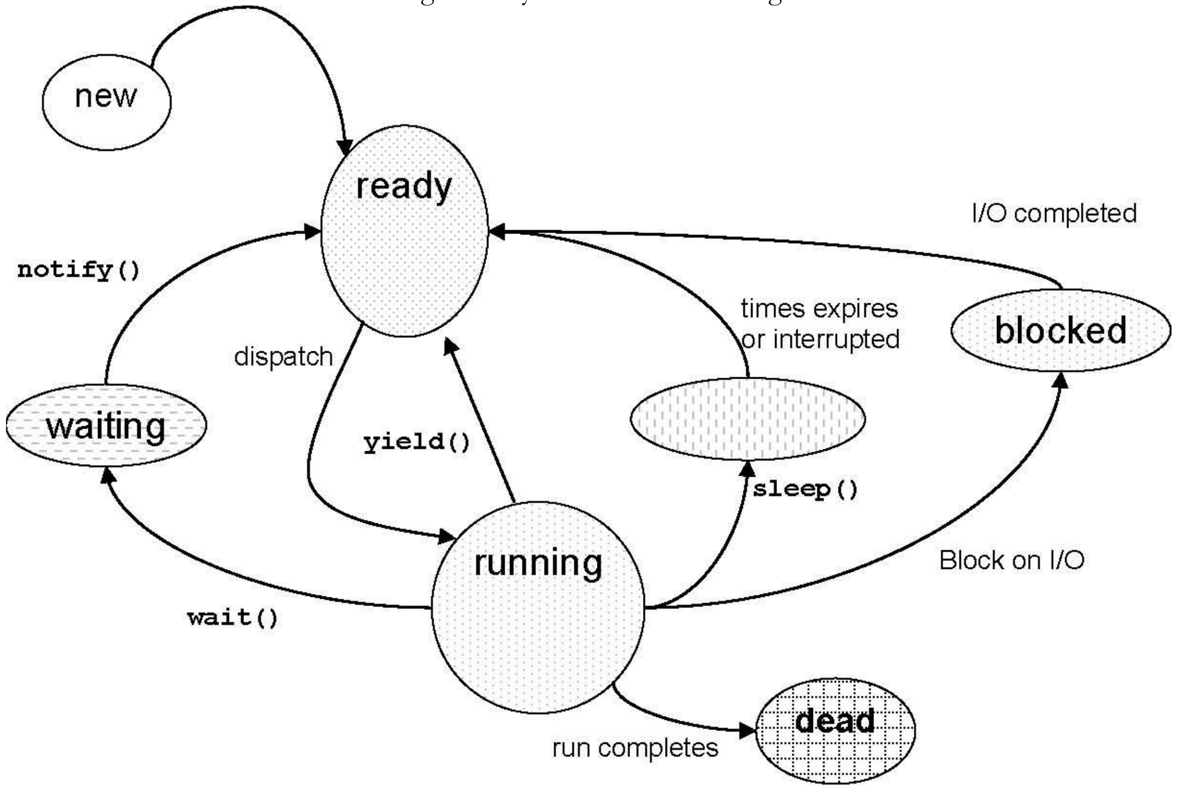
Output

Since both threads have the same priority (until we change the priority of some thread explicitly). Therefore both threads will execute on alternate basis. This can be confirmed from the output given below:



```
D:\examples\multithreading>java YieldEx
first job = 1
second job = 1
first job = 2
second job = 2
first job = 3
second job = 3
first job = 4
second job = 4
first job = 5
second job = 5
first job = 6
second job = 6
first job = 7
second job = 7
first job = 8
second job = 8
first job = 9
second job = 9
first job = 10
second job = 10
```

A thread can be in different states during its lifecycle as shown in the figure.



Some Important states are

. **New state**

-When a thread is just created

. **Ready state**

-Thread's start() method invoked -Thread can now execute -Put it into the Ready Queue of the scheduler

. **Running state**

-Thread is assigned a processor and now is running

. **Dead state**

-Thread has completed or exited

-Eventually disposed off by system

<u>Thread's Joining</u>

-Used when a thread wants to wait for another thread to complete its run( ) method

-For example, if thread2 sent the thread2.join() message, it causes the currently executing thread to block efficiently until thread2 finishes its run() method

-Calling join method can throw InterruptedException, so you must use try-catch block to handle it

Example Code: Demonstrating join( ) usage

Below the modified code of Worker.java is given. It only prints the job of the worker

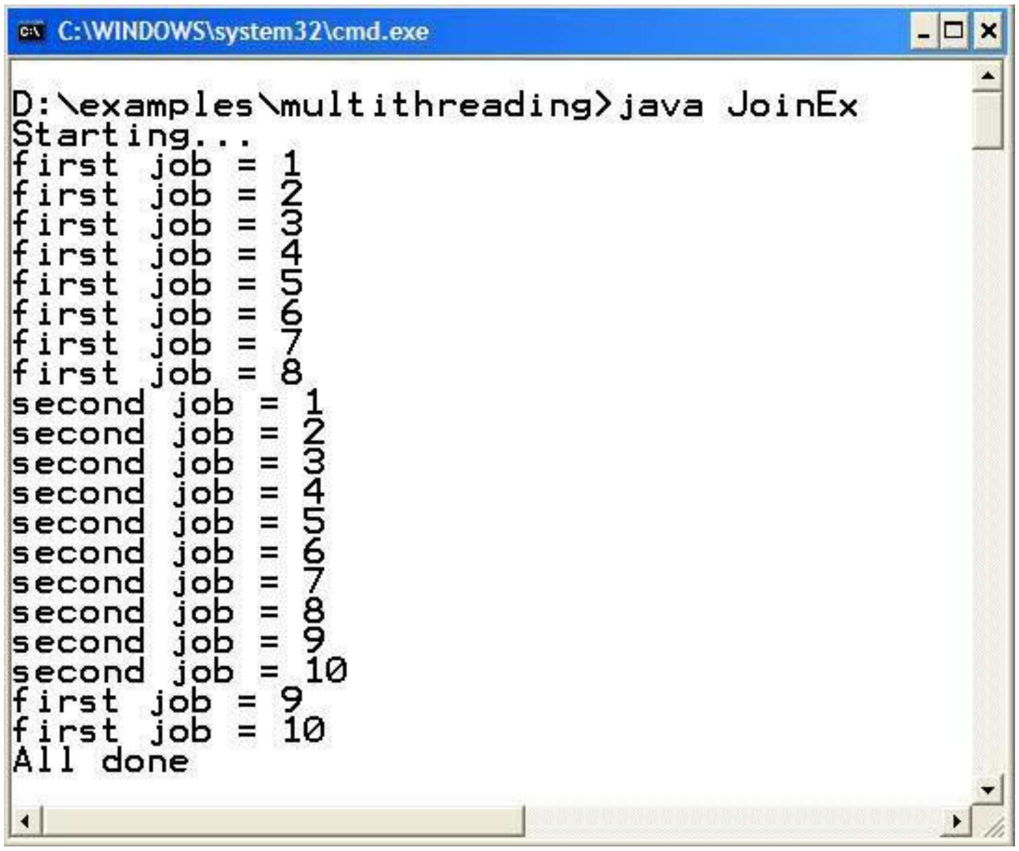
```
// File Worker.java
public class Worker implements Runnable { private String job ; public Worker (String j ){
                job = j;} public void run ( ) {for(int i=1; i<= 10; i++) {
        System.out.println(job + " = " + i); } // end for} // end run} //
        end class
```

The class JoinEx will demonstrate how current running (main) blocks until the remaining threads finished their run( )

```
// File JoinEx.java
public class JoinEx { public static void main (String args[ ]){
          Worker first = new Worker ("first job");Worker second = new Worker ("second
          job");
          Thread t1 = new Thread (first );
          Thread t2 = new Thread (second);

          System.out.println("Starting...");
          // starting threads
          t1.start();
          t2.start();

          // The current running thread (main) blocks until both //workers have finished
          try {
             t1.join();
             t2.join();

           }catch (Exception ex) {System.out.println(ex);}
          System.out.println("All done ");
     } // end main}
```

Output

On executing JoinEx, notice that "Starting" is printed first followed by printing workers jobs. Since main thread do not finish until both threads have finished their run( ). Therefore "All done" will be print on last.



```
D:\examples\multithreading>java JoinEx
Starting...
first job = 1
first job = 2
first job = 3
first job = 4
first job = 5
first job = 6
first job = 7
first job = 8
second job = 1
second job = 2
second job = 3
second job = 4
second job = 5
second job = 6
second job = 7
second job = 8
second job = 9
second job = 10
first job = 9
first job = 10
All done
```

References:

- Java, A Practical Guide by Umair Javed
- Java tutorial by Sun: http://java.sun.com/docs/books/tutorial/
- CS193j handouts on Stanford