Savitribai Phule Pune University

सावित्रीबाई फुले पुणे विद्यापीठ

T. Y. B. Sc.

(Computer Science)

Laboratory Course II

Programming in Java - CS348

Semester I

(From Academic Year 2015)

Name ______________________________ **Roll No.** _______

College ______________________________ **Division** _______

Academic Year _____________

PREPARED BY:

PROF. MS. POONAM PONDE (NOWROSJEE WADIA COLLEGE)

PROF. JEEVAN LIMAYE (FERGUSSON COLLEGE)

FIRST EDITION AUTHORS:

Ms. Poonam Ponde

Ms. Seema Jawale

Ms. Kalpana Joshi

Ms. Jayshri Patil

Ms. Ranjana Shevkar

ABOUT THE WORK BOOK

- **OBJECTIVES OF THIS BOOK**

This lab-book is intended to be used by T.Y.B.Sc(Computer Science) students for Laboratory course – II (Programming in Java) , Semester I.
The objectives of this book are

a. Covers the complete scope of the syllabus.

b. Bringing uniformity in the way course is conducted across different colleges.

c. Continuous assessment of the students.

d. Providing ready references for students while working in the lab.

- **How to use this book?**

This book is mandatory for the completion of the laboratory course. It is a measure of the performance of the student in the laboratory for the entire duration of the course.

- **Instructions to the students**

1. Students should carry this book during practical sessions.

2. Print outs of source code and outputs is optional

3. Student should read the topics mentioned in **Reading section** of this book before coming for the practical session.

4. Students should solve those exercises which are selected by Practical in-charge as a part of journal activity. However, students are free to solve additional exercises for more practice.

5. Each assignment will be assessed on a scale of 0 to 5 as indicated below.

i) Not done	0
ii) Incomplete	1
iii) Late Complete	2
iv) Needs improvement	3
v) Complete	4
vi) Well Done	5

- **Difficulty Levels**

Self Activity: Students should solve these exercises for practice only.
SET A - Easy: All exercises are compulsory.
SET B - Medium: All exercises are compulsory.

- **Instruction to the Instructors**

1) Make sure that students follow the instruction as given above.

2) After a student completes a specific set, the instructor has to verify the outputs and sign in the space provided after the activity.

3) Evaluate each assignment on a scale of 5 as specified above by ticking appropriate box.

4) The value should also be entered on assignment completion page of the respective Lab course.

5) Students should be encouraged to use an IDE like Eclipse for their assignments and project work.

Roll No :	
Name :	

Assignment Completion Sheet

Sr. No	Assignment Name	Marks
1	Java Tools and IDE, Simple java programs	
2	Array of Objects and Packages	
3	Inheritance and Interfaces	
4	Exception Handling	
5	I/O and File Handling	
6	GUI Designing, Event Handling and Applets	
	Total out of 30	
	Total out of 5	

Signature of Incharge:

Examiner I:

Examiner II:

Date:

Assignment 1: Java Tools and IDE, Simple Java programs

Objectives

- **Introduction to the java environment**
- **Use of java tools like java, javac, jdb and javadoc**
- **Use of IDE – Eclipse (demo)**
- **Defining simple classes and creating objects.**

Reading

You should read the following topics before starting this exercise

1. Creating, compiling and running a java program.
2. The java virtual machine.
3. Java tools like javac, java, javadoc, javap and jdb.
4. Java keywords
5. Syntax of class.

Ready Reference

Java Tools

(1) javac:- javac is the java compiler which compiles .java file into .class file(i.e. bytecode). If the program has syntax errors, javac reports them. If the program is error-free, the output of this command is one or more .class files.

Syntax:

```
javac  fileName.java
```

(2) java:- This command starts Java runtime environment, loads the specified .class file and executes the main method.

Syntax:

```
java fileName
```

(3) javadoc:- javadoc is a utility for generating HTML documentation directly from comments written in Java source code.Javadoc comments have a special form but seems like an ordinary multiline comment to the compiler.

Syntax of the comment:

*/***
A sample doc comment
**/*

Syntax:

```
javadoc [options] [packagenames ] [ sourcefiles ] [@files ]
```

Where,

packagenames: A series of names of packages, separated by spaces
sourcefiles: A series of source file names, separated by spaces
@files: One or more files that contain packagenames and sourcefiles in any order, one name per line.

Javadoc creates the HTML documentation on the basis of the javadoc tags used in the source code files. These tags are described in the table below:

Tag	Syntax	Description
@see	@see reference	Allows you to refer to the documentation in other classes.
@author	@author author-information	Author-information contains author name, and / or author email address or any other appropriate information.
@version	@version version-information	Specifies the version of the program
@since	@since version	This tag allows you to indicate the version of this code that began using a particular feature.

@param	@param name description	This is used for method documentation. Here, name is the identifier in the method parameter list, and description is text that can describes the parameter.
@return	@return description	This describes the return type of a method.
@throws	@throws classname description	This is used when we handle Exceptions. It describes a particular type of exception that can be thrown from the method call.
@deprecated	@deprecated description	The deprecated tag suggests that this feature is no longer supported. A method that is marked @deprecated causes the compiler to issue a warning if it is used.

(4) jdb: -

jdb helps you find and fix bugs in Java language programs. This debugger has limited functionality.

Syntax:

```
jdb [ options ] [ class ] [ arguments ]
```

options : Command-line options.
class : Name of the class to begin debugging.
arguments : Arguments passed to the main() method of class.

After starting the debugger, the jdb commands can be executed. The important jdb commands are:

i. *help, or?:* The most important **jdb** command, `help` displays the list of recognized commands with a brief description.
ii. *run:* After starting **jdb**, and setting any necessary breakpoints, you can use this command to start the execution the debugged application.
iii. *cont:* Continues execution of the debugged application after a breakpoint, exception, or step.
iv. *print:* Displays Java objects and primitive values. For variables or fields of primitive types, the actual value is printed. For objects, a short description is printed.
   Examples:
   ```
   print MyClass.myStaticField
   print myObj.myInstanceField
   print i + j + k
   print myObj.myMethod()//if myMethod returns non-null
   ```
v. *dump:* For primitive values, this command is identical to `print`. For objects, it prints the current value of each field defined in the object. Static and instance fields are included.
vi. *next:* The `next` command advances execution to the next line in the current stack frame.
vii. *step:* The `step` commands advances execution to the next line whether it is in the current stack frame or a called method.

Breakpoints can be set in jdb at line numbers, constructors, beginning of a method.
Example:

`stop at MyClass:10` //sets breakpoint at instruction at line 10 of the source file containing MyClass
`stop in MyClass.display` // sets breakpoint at beginning of method display in MyClass
`stop in MyClass.<init>` //sets breakpoint at default constructor of MyClass
`stop in MyClass.<init(int)>` //sets breakpoint at parameterized constructor with int as parameter

(4) javap: -

The javap tool allows you to query any class and find out its list of methods and constants.

```
   javap [ options ] class
```

Example: javap java.lang.String
It is a disassembler which allows the bytecodes of a class file to be viewed when used with a classname and the –c option.

```
   javap -c class
```

Setting CLASSPATH

The classpath is the path that the Java runtime environment searches for classes and other resource files. The class path can be set using either the –classpath option or by setting the CLASSPATH environment variable.
The -classpath option is preferred because you can set it individually for each application without affecting other applications and without other applications modifying its value. The default value of the class path is ".", meaning that only the current directory is searched. Specifying either the CLASSPATH variable or the -cp command line switch overrides this value.

```
javac –classpath \myProg\myPackage; \myProg\otherclasses
                        Or
CLASSPATH= classpath1;classpath2...
export CLASSPATH
```

Example

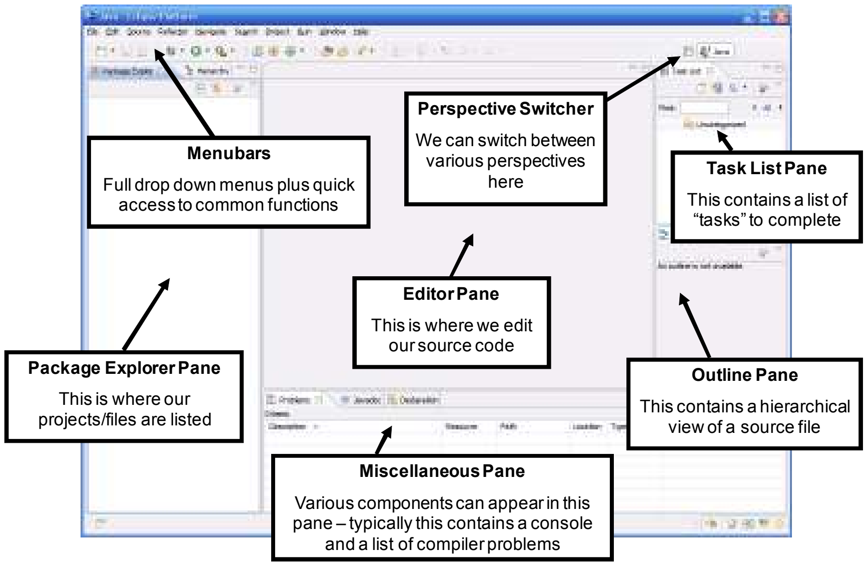
```
 CLASSPATH=.:/usr/local/classes.jar:/home/user1/myclasses
export CLASSPATH
```

To retain the existing classpath setting, use $CLASSPATH in the new list.

```
export CLASSPATH=$CLASSPATH:/home/user1/myclasses
```

About Eclipse

Eclipse is a popular IDE (Integrated Development Environment) for java programming. It contains a base workspace and an extensible plug-in system for customizing the environment. The latest Eclipse version 4.5 was released in 2015. The Eclipse IDE is also available as an IDE for other languages, ranging from C, C++ to Lua, Python, Perl and PHP. It provides an editor, debugger, source control and other tools.
The GUI looks as shown:


Menubars
Full drop down menus plus quick access to common functions
Perspective Switcher
We can switch between various perspectives here
Task List Pane
This contains a list of "tasks" to complete
Editor Pane
This is where we edit our source code
Package Explorer Pane
This is where our projects/files are listed
Outline Pane
This contains a hierarchical view of a source file
Miscellaneous Pane
Various components can appear in this pane – typically this contains a console and a list of compiler problems

Steps to run a java program using Eclipse:
1. Select workspace
2. Create a new project
3. Project name appears in package explorer, src folder contains source code files
4. Create class (New->Class)
5. Run your program (right click on the class and select Run As -> Java Application)

Self Activity (using IDE and editor)

1. Sample program

```
/* Program to generate documentation*/
/**
  This program demonstrates javadoc
*/
public class MyClass {
int num;
  /**
   Default constructor
  */
 public MyClass() {
     num=0;
 }
 /**
   Member function
   @param x Represents the new value of num
   @return void No return value
 */
 public void assignValue(int x) {
  num = x;
 }
```

Type the following command: javadoc MyClass.java. See the HTML documentation file MyClass.html

2. Sample program

```
/* Program to define a class and an object of the class* /
public class MyClass {
int num;
public MyClass() {
     num=0;
 }
public MyClass(int num) {
     this.num = num;
 }
public static void main(String[] args) {
    MyClass m1 = new MyClass();
    if(args.length > 0)
    {
      int n = Integer.parseInt(args[0]);
      MyClass m2 = new MyClass(n);
      System.out.println(m1.num);
      System.out.println(m2.num);
    }
    else
       System.out.println("Insufficient arguments");
 }
}
```

Pass one command line argument to the above program and execute it.

Answer the following questions :

1. How will you pass command line argument using IDE?

____

____

2. Write the output if the command line argument passed is “ABC”.

____

Lab Assignments

SET A

1. Using javap, view the methods of the following classes from the lang package: java.lang.Object , java.lang.String and java.util.Scanner.

2. Compile sample program 2. Type the following command and view the bytecodes.

```
javap -c MyClass
```

SET B

1. Write a java program to display the system date and time in various formats shown below:

```
Current date is  : 31/07/2015
Current date is  : 07-31-2015
Current date is  : Friday July 31 2015
Current date and time is  : Fri July 31 16:25:56 IST 2015
Current date and time is  :  31/07/15 16:25:56 PM +0530
Current time is  :   16:25:56
Current week of year is : 31
Current week of month : 5
Current day of the year is : 212
```

Note: Use java.util.Date and java.text.SimpleDateFormat class

2. Define a class MyNumber having one private int data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value (Use this). Write methods isNegative, isPositive, isZero, isOdd, isEven. Create an object in main. Use command line arguments to pass a value to the object (Hint : convert string argument to integer) and perform the above tests. Provide javadoc comments for all constructors and methods and generate the html help file.

Signature of the instructor		Date	

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 2:   Array of Objects and Packages

Objectives

- **Defining a class.**
- **Creating an array of objects.**
- **Creating a package. (Using package command)**
- **Using packages (Using import command)**

Reading

You should read the following topics before starting this exercise:
1. Structure of a class in java.
2. Declaring class reference.
3. Creating an object using new.
4. Declaring an array of references.
5. Creating an array of objects.
6. Syntax of the package and import command.

Ready Reference

General form of a class
```
class classname {
   type instance-variable1;
   type instance-variable2;
   // ...
   type instance-variableN;

   type methodname1(parameter-list) {
     // body of method
   }
   type methodname2(parameter-list) {
     // body of method
   }
   // ...
   type methodnameN(parameter-list) {
     // body of method
   }
}
```
Example
```
class Student{
   private int rollNumber;   private String name;
   Student() //constructor
   {
     rollNumber = 0; name = null;
   }
   Student(int rollNumber, String name)
   {
      this.rollNumber = rollNumber; this.name = name;
   }
   void display()
   {
     System.out.println("Roll number = "  + rollNumber);
     System.out.println(" Name = "  + name);
   }
}
```

Creating objects:

```
ClassName referenceName;
referenceName = new ClassName();
      OR
ClassName referenceName = new ClassName();
```

Example:

```
Student s1 = new Student();
Student s2 = new Student(10, "ABC");
```

Overriding toString method of the Object class:

The toString method gives a string representation of an object. To over-ride the toString method for a user defined class, use the syntax:

```
public String toString()
{
  // return a string representation of the object
}
```

Example

```
class Student{
   private int rollNumber;
   private String name;
   public String toString() {
     return "Roll Number = "  + rollNumber + "Name = "+name;
   }
}
```

Declaring an array of references:

```
ClassName[] arrayName = new ClassName[size];
```

Example:

```
Student[] studentArray = new Student[10];
```

Creating an array of objects:

```
for each reference in the array
{
  Create an object using new
}
```

Example:

```
Student[] studentArray = new Student[10];
for(i=0; i<10; i++)
     studentArray[i] = new Student();
```

To convert the argument from String to any type, use Wrapper classes.

Method	Purpose
Byte.parseByte	Returns byte equivalent of a String
Short.parseShort	Returns the short equivalent of a String
Integer.parseInt	Returns the int equivalent of a String
Long.parseLong	Returns the long equivalent of a String
Float.parseFloat	Returns the float equivalent of a String
Double.parseDouble	Returns the double equivalent of a String

Simple I/O

To read a String from the console, use the following code:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
```

Or

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

For this, you will have write the following statement at the beginning:

```
import java.io.*;
```

Packages:

A package is a collection of related classes and interfaces. It provides a mechanism for compartmentalizing classes. The Java API is organized as a collection of several predefined packages. The java.lang package is the default package included in all java programs. The commonly used packages are:

java.lang	Language support classes such as Math, Thread, String
java.util	Utility classes such as LinkedList, Vector , Date.
java.io	Input/Output classes
java.awt	For graphical user interfaces and event handling.
javax.swing	For graphical user interfaces
java.net	For networking
java.applet	For creating applets.

Creating a package

To create a user defined package, the package statement should be written in the source code file. This statement should be written as the first line of the program. Save the program in a directory of the same name as the package.

```
package packageName;
```

Accessing a package

To access classes from a package, use the import statement.

```
import packageName.*; //imports all classes
import packageName.className; //imports specified class
```

Note that the package can have a hierarchy of subpackages. In that case, the package name should be qualified using its parent packages. *Example:* project.sourcecode.java

Here, the package named project contains one subpackage named sourcecode which contains a subpackage named java.

Access Rules

The access rules for members of a class are given in the table below.

Accessible to	public	protected	none	private
Same class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	Yes	No
Subclass (in other package)	Yes	Yes	No	No
Non subclass in Other package	Yes	No	No	No

Self Activity

1. Sample program to create objects , demonstrate use of toString and static keyword.

```
class Student {
   int rollNumber;
   String name;
   static String classTeacher;

   Student(int r, String n)   {
        rollNumber = r;  name = n;
   }
   static void assignTeacher(String name) {
       classTeacher = name;
   }
   public String toString()   {
      return "[ " + rollNumber + "," + name + "," + classTeacher +"
]";
   }
   public static void main(String[] args)   {
     Student s1 = new Student(1,"A");
     Student s2 = new Student(2,"B");
     Student.assignTeacher("ABC");
     System.out.println(s1);
     System.out.println(s2);
   }
}
```

2. Sample program to read Student roll number and name from the console and display them (Using BufferedReader).

```
import java.io.*;
class ConsoleInput
{
  public static void main(String[] args)    throws IOException
  {
    int rollNumber;
    String name;
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Enter the roll number: ");
    rollNumber = Integer.parseInt(br.readLine());
    System.out.println(" Enter the name: ");
    name = br.readLine();
    System.out.println(" Roll Number = "  + rollNumber);
    System.out.println(" Name = "  + name);
  }
}
```

3. Sample program to read Student roll number and name from the console and display them (Using Scanner class).

```
import java.util.Scanner;
class ScannerTest{
     public static void main(String args[])throws Exception
     {
       Scanner sc=new Scanner(System.in);
       System.out.println("Enter your rollno and name :");
       int rollno=sc.nextInt();
       String name=sc.next();
       System.out.println("Rollno:"+rollno+" Name:"+name);
       sc.close();
     }
}
```

4. Sample program to create and use packages

```
//A.java
package P1;
public class A {
      public void display() {
            System.out.println("In display of A");
      }
}
//B.java
package P1.P2;  //P2 is a subpackage of P1
```

```
public class B {
      public void display() {
            System.out.println("In display of B");
      }
}
//PackageTest.java
import P1.*;
import P1.P2.*;
class PackageTest {
      public static void main(String args[]){
            A obj1 = new A();
            obj1.display();
            B obj2 = new B();
            obj2.display();
     }
}
```

Create folder P1. Save A.java in folder P1. Create folder P2 inside P1. Save B.java in P2.

Lab Assignments

SET A

1. Define a Student class (roll number, name, percentage). Define a default and parameterized constructor. Keep a count of objects created. Create objects using parameterized constructor and display the object count after each object is created. (Use static member and method). Also display the contents of each object.

2. Modify the above program to create n objects of the Student class. Accept details from the user for each object. Define a static method "sortStudent" which sorts the array on the basis of percentage.

SET B

1. Create a package named **Series** having three different classes to print series:
   a. Prime numbers   b. Fibonacci series   c. Squares of numbers
Write a program to generate 'n' terms of the above series.

2. Write a Java program to create a Package "SY" which has a class SYMarks (members – ComputerTotal, MathsTotal, and ElectronicsTotal). Create another package TY which has a class TYMarks (members – Theory, Practicals). Create n objects of Student class (having rollNumber, name, SYMarks and TYMarks). Add the marks of SY and TY computer subjects and calculate the Grade ('A' for >= 70, 'B' for >= 60 'C' for >= 50 , Pass Class for > =40 else 'FAIL') and display the result of the student in proper format.

Signature of the instructor		Date	

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 3: Inheritance and Interfaces

Objectives

- **To implement inheritance in java.**
- **To define abstract classes.**
- **To define and use interfaces.**
- **Use predefined interfaces like Cloneable**

Reading

You should read the following topics before starting this exercise:

1. Concept of inheritance.
2. Use of extends keyword.
3. Concept of abstract class.
4. Defining an interface.
5. Use of implements keyword.

Ready Reference

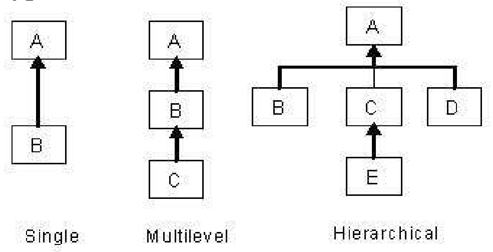
Inheriting a class : The syntax to create a subclass is :

```
class SubClassName extends SuperClassName
{
     //class body
}
```

Example:

```
class Manager extends Employee
{  //code  }
```

Types of Inheritance



Access in subclass

The following members can be accessed in a subclass:
i) public or protected superclass members.
ii) Members with no specifier if subclass is in same package.

The "super" keyword

It is used for three purposes:

i) Invoking superclass constructor - `super(arguments)`
ii) Accessing superclass members – `super.member`
iii) Invoking superclass methods – `super.method(arguments)`

Example:

```
class A
{ protected int num;
  A(int num) { this.num = num; }
}
```

```
class B extends A
{
   int num;
   B(int a, int b) {
     super(a); //should be the first line in the subclass constructor
     this.num = b;
    }
    void display() {
      System.out.println("In A, num = " + super.num);
      System.out.println("In B, num = "  + num);
    }
}
```

Overriding methods

Redefining superclass methods in a subclass is called overriding. The signature of the subclass method should be the same as the superclass method.

```
class A
{
  void method1(int num) {
    //code
  }
}
class B extends A
{
  void method1(int x) {
    //code
  }
}
```

Dynamic binding

When over-riding is used, the method call is resolved during run-time i.e. depending on the object type, the corresponding method will be invoked.

Example:

```
A ref;
ref = new A();
ref.method1(10); //calls method of class A
ref = new B();
ref.method1(20); //calls method of class B
```

Abstract class

An abstract class is a class which cannot be instantiated. It is only used to create subclasses. A class which has abstract methods must be declared abstract. An abstract class can have data members, constructors, method definitions and method declarations.

```
      abstract class ClassName
      {
        ...
      }
```

Abstract method

An abstract method is a method which has no definition. The definition is provided by the subclass.

```
      abstract returnType method(arguments);
```

Interface

An interface is a pure abstract class i.e. it has only abstract methods and final variables. An interface can be implemented by multiple classes.

```
      interface InterfaceName
      {
        //abstract methods
        //final variables
      }
```

Example:

```
interface MyInterface
{
  void method1();
  void method2();
  int size= 10; //final and static
}
class MyClass implements MyInterface {
  //define method1 and method2
}
```

Self Activity

1. Sample program to demonstrate inheritance and interfaces

```
interface Shape
{
  double area();
}
class Circle implements Shape
{
  double radius;
  Circle(double radius)  {
     this.radius=radius;
  }
  public double area()  {
      return java.util.Math.PI * radius* radius;
  }
}
class Cylinder extends Circle
{
   double height;
   Cylinder(double radius, double height)   {
     super(radius);
     this.height=height;
  }
  public double area()  //overriding
  {
      return java.util.Math.PI * radius* radius *height;
  }
}
public class Test {
  public static void main(String[] args)
  {
  Shape s;
  s = new Circle(5.2);
  System.out.println("Area of circle = " + s.area());
  s = new Cylinder(5, 2.5);
  System.out.println("Area of cylinder = " + s.area());
  }
}
```

Lab Assignments

SET A

1. Define a class Employee having private members – id, name, department, salary. Define default and parameterized constructors. Create a subclass called “Manager” with private member bonus. Define methods accept and display in both the classes. Create n objects of the Manager class and display the details of the manager having the maximum total salary (salary+bonus)

2. Create an abstract class Shape with methods calc_area and calc_volume. Derive three classes Sphere(radius) , Cone(radius, height) and Cylinder(radius, height), Box(length, breadth, height) from it. Calculate area and volume of all. (Use Method overriding).

3. Write a Java program to create a super class **Vehicle** having members Company and price. Derive 2 different classes LightMotorVehicle (members – mileage) and HeavyMotorVehicle (members – capacity-in-tons). Accept the information for n vehicles and display the information in appropriate form. While taking data, ask the user about the type of vehicle first.

SET B

1. Define an abstract class "Staff" with members name and address. Define two sub-classes of this class – "FullTimeStaff" (department, salary) and "PartTimeStaff" (number-of-hours, rate-per-hour). Define appropriate constructors. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user's choice. Display details of all "FullTimeStaff" objects and all "PartTimeStaff" objects.

2. Create an interface "CreditCardInterface" with methods : viewCreditAmount(), useCard(), payCredit() and increaseLimit(). Create a class SilverCardCustomer (name, cardnumber (16 digits), creditAmount – initialized to 0, creditLimit - set to 50,000) which implements the above interface. Inherit class GoldCardCustomer from SilverCardCustomer having the same methods but creditLimit of 1,00,000. Create an object of each class and perform operations. Display appropriate messages for success or failure of transactions. (Use method overriding)

i. useCard() method increases the creditAmount by a specific amount upto creditLimit
ii. payCredit() reduces the creditAmount by a specific amount.
iii. increaseLimit() increases the creditLimit for GoldCardCustomers (only 3 times, not more than 5000Rs. each time)

Signature of the instructor		Date	

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 4: Exception Handling

Objectives

- **Demonstrate exception handling mechanism in java**
- **Defining user defined exception classes**

Reading

You should read the following topics before starting this exercise:

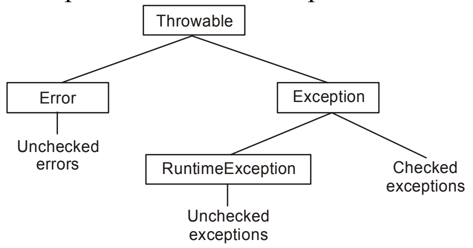
1. Concept of Exception
2. Exception class hierarchy.
3. Use of try, catch, throw, throws and finally keywords
4. Defining user defined exception classes

Ready Reference

Exception: An *exception* is an abnormal condition that arises in a code at run time.
When an exception occurs,

1. An object representing that exception is created.
2. The method may handle the exception itself.
3. If the method cannot handle the exception, it "throws" this exception object to the method which called it.
4. The exception is "caught" and processed by some method or finally by the default java exception handler.

Predefined Exception classes
Java provides a hierarchy of Exception classes which represent an exception type.



Exception Handling keywords
Exception handling in java is managed using 5 keywords: **try , catch , throw, throws, finally**
Syntax

```
try
{
      // code that may cause an exception
}
catch (ExceptionType1 object)
{
      // handle the exception
}
catch (ExceptionType2 object)
{
      // handle the exception
}
finally
{
      // this code is always executed
}
```

Example:
```
try
{
   int a = Integer.parseInt(args[0]);
    ...
}
catch(NumberFormatException e)
{
    System.out.println("Caught" ) ;
}
```
Note: try-catch blocks can be nested.

throw keyword:
The throw keyword is used to throw an exception object or to rethrow an exception.
```
      throw exceptionObject;
```
Example:
```
catch(NumberFormatException e)
{
    System.out.println("Caught and rethrown" ) ;
    throw e;
}
```
We can explicitly create an exception object and throw it. For example:
```
throw new NumberFormatException();
```

throws keyword:
If the method cannot handle the exception, it must declare a list of exceptions it may cause. This list is specified using the throws keyword in the method header. All checked exceptions muct be caught or declared.
Syntax:
```
returnType methodName(arguments) throws ExceptionType1
[,ExceptionType2...]
{
  //method body
}
```
Example:
```
void acceptData() throws IOException
{
   //code
}
```
Exception Types:
There are *two* types of Exceptions, **Checked exceptions** and **Unchecked exceptions**. Checked exceptions must be caught or rethrown. Unchecked exceptions do not have to be caught.

Unchecked Exceptions:

Exception	**Meaning**
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.

NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

Checked Exceptions:

Exception	**Meaning**
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

User Defined Exceptions:

A user defined exception class can be created by extending the Exception class.

```
      class UserDefinedException extends Exception
      {
            //code
      }
```

When that exception situation occurs, an object of this exception class can be created and thrown. For example, if we are accepting an integer whose valid values are only positive, then we can throw an “InvalidNumberException” for any negative value entered.

```
class NegativeNumberException extends Exception
{
      NegativeNumberException(int n){
            System.out.println("Negative input " + n);
       }
}
...
public static void main(String[] args)
{
   int num = Integer.parseInt(args[0]);
   if( num < 0)
       throw new NegativeNumberException(num);
  else
     //process num
}
```

Self Activity

1. Sample program to demonstrate exceptions

```
class NegativeNumberException extends Exception
{
   NegativeNumberException(int n){
     System.out.println(“Negative input : “ + n);
   }
}
public class ExceptionTest
     {
        public static void main( String args[] )
        {
          int num, i, sum=0;
          try {
           num = Integer.parseInt(args[0]);
           if(num < 0)
              throw new NegativeNumberException(num);
           for(i=0; i<num; i++)
               sum = sum+i;
          }
          catch(NumberFormatException e){
            System.out.println("Invalid format");
          }
          catch(NegativeNumberException e){    }
          finally  {
             System.out.println("The sum is : "+sum);
          }
       } // end main
    } // end class
```
Compile and run the program for different inputs like abc, -3 and 10

Lab Assignments

SET A

1. Define a class CricketPlayer (name, no_of_innings, no_times_notout, total_runs, bat_avg). Create an array of n player objects. Calculate the batting average for each player using a static method avg(). Handle appropriate exception while calculating average. Define a static method "sortPlayer" which sorts the array on the basis of average. Display the player details in sorted order.
2. Define a class SavingAccount (acNo, name, balance). Define appropriate constructors and operations withdraw(), deposit() and viewBalance(). The minimum balance must be 500. Create an object and perform operations. Raise user defined InsufficientFundsException when balance is not sufficient for withdraw operation.

SET B

1. Define a class MyDate (day, month, year) with methods to accept and display a MyDate object. Accept date as dd, mm, yyyy. Throw user defined exception "InvalidDateException" if the date is invalid.
   Examples of invalid dates : 12 15 2015, 31 6 1990, 29 2 2001

Signature of the instructor		Date	

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 5: I/O and File Handling

Objectives

- **Performing Input/Output operations using console and files.**

Reading

You should read the following topics before starting this exercise:

1. Concept of streams
2. Types of streams
3. Byte and Character stream classes.
4. The File class

Ready Reference

java.io.File class

This class supports a platform-independent definition of file and directory names. It also provides methods to list the files in a directory, to check the existence, readability, writeability, type, size, and modification time of files and directories, to make new directories, to rename files and directories, and to delete files and directories.

Constructors:

```
    public File(String path);
public File(String path, String name);
public File(File dir, String name);
```

Example

File f1=new File("/home/java/a.txt");

Methods

1. boolean canRead()- Returns True if the file is readable.
2. boolean canWrite()- Returns True if the file is writeable.
3. String getName()- Returns the name of the File with any directory names omitted.
4. boolean exists()- Returns true if file exists
5. String getAbsolutePath()- Returns the complete filename. Otherwise, if the File is a relative file specification, it returns the relative filename appended to the current working directory.
6. String getParent()- Returns the directory of the File. If the File is an absolute specification.
7. String getPath()- Returns the full name of the file, including the directory name.
8. boolean isDirectory()- Returns true if File Object is a directory
9. boolean isFile()- Returns true if File Object is a file
10. long lastModified()- Returns the modification time of the file (which should be used for comparison with other file times only, and not interpreted as any particular time format).
11. long length()- Returns the length of the file.
12. boolean delete()- deletes a file or directory. Returns true after successful deletion of a file.
13. boolean mkdir ()- Creates a directory.
14. boolean renameTo (File dest)- Renames a file or directory. Returns true after successful renaming

Example 1:- Checking file existance

```
import java.io.File;
class FileTest
{
public static void main(String args[ ])
{
      File f1=new File ("data.txt");
      if (f1.exists())
            System.out.println ("File Exists");
      else
            System.out.println ("File Does Not Exists");
      }
}
```

Directories

A directory is a File that contains a list of other files & directories. When you create a File object & it is a directory, the isDirectory() method will return true. In this case list method can be used to extract the list of other files & directories inside.

The forms of list() method is-

```
      public String[ ] list()
       public String[ ] list(FilenameFilter filter)
```

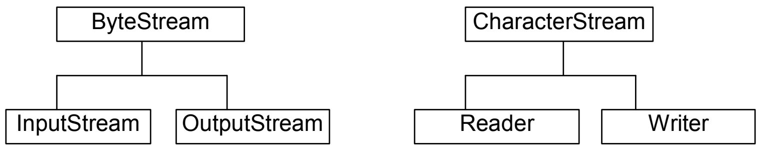
Example :- Using a list( )

```
String dirname="/javaprg/demo";
File f1=new File (dirname);
if (f1.isDirectory())
{
      String s[]= f1.list();
      for (int i=0; i<s.length; i++)
      {
            File f=new File (dirname+"/"+s[i]);
            if (f.isDirectory())
                  System.out.println(s[i]+ " is a directory");
            else
                  System.out.println(s[i]+ " is a File");
      }
}
```

Streams

A stream is a sequence of bytes. When writing data to a stream, the stream is called an output stream. When reading data from a stream, the stream is called an input stream. If a stream has a buffer in memory, it is a buffered stream. Binary Streams contain binary data. Character Streams have character data and are used for storing and retrieving text.

The two main types of Streams are ByteStream and CharacterStream.



There are four top level abstract stream classes: InputStream, OutputStream, Reader, and Writer.

1. InputStream. A stream to read binary data.
2. OutputStream. A stream to write binary data.
3. Reader. A stream to read characters.
4. Writer. A stream to write characters.

ByteStream Classes

a. InputStream Methods-

1. **int read ()-** Returns an integer representation of next available byte of input.-1 is returned at the stream end.
2. **int read (byte buffer[ ])-** Read up to buffer.length bytes into buffer & returns actual number of bytes that are read. At the end returns –1.
3. **int read(byte buffer[ ], int offset, int numbytes)-** Attempts to read up to numbytes bytes into buffer starting at buffer[offset]. Returns actual number of bytes that are read. At the end returns –1.
4. **void close()-** to close the input stream
5. **void mark(int numbytes)-** places a mark at current point in input stream & remain valid till number of bytes are read.
6. **void reset()-** Resets pointer to previously set mark/ goes back to stream beginning.
7. **long skip(long numbytes)-** skips number of bytes.
8. **int available()-** Returns number of bytes currently available for reading.

b. OutputStream Methods-

1. **void close() -** to close the OutputStream
2. **void write (int b) -** Writes a single byte to an output stream.
3. **void write(byte buffer[ ]) -** Writes a complete array of bytes to an output stream.
4. **void write (byte buffer[ ], int offset, int numbytes) -** Writes a sub range of numbytes bytes from the array buffer, beginning at buffer[offset].
5. **void flush() -** clears the buffer.

The following table lists the Byte Stream classes

Stream Class	**Meaning**
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An input stream that contains methods for reading the Java standard data types
DataOutputStream	An output stream that contains methods for writing the Java standard data types
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that writes to a file
FilterInputStream	Implements **InputStream**
FilterOutputStream	Implements **OutputStream**
InputStream	Abstract class that describes stream input
OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe
PipedOutputStream	Output pipe
PrintStream	Output stream that contains **print( )** and **println( )**
PushbackInputStream	Input stream that supports one-byte “unget,” which returns a byte to the input stream
RandomAccessFile	Supports random access file I/O
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read

CharacterStream Classes

1. Reader : Reader is an abstract class that defines Java's method of streaming character input. All methods in this class will throw an **IOException.**
Methods in this class-

1. **int read ()-** Returns an integer representation of next available character from invoking stream. -1 is returned at the stream end.
2. **int read (char buffer[ ])-** Read up to buffer.length chacters to buffer & returns actual number of characters that are successfully read. At the end returns –1.
3. **int read(char buffer[ ], int offset, int numchars)-** Attempts to read up to numchars into buffer starting at buffer[offset]. Returns actual number of characters that are read. At the end returns –1.
4. **void close()-** to close the input stream
5. **void mark(int numchars)-** places a mark at current point in input stream & remain valid till number of characters are read.
6. **void reset()-** Resets pointer to previously set mark/ goes back to stream beginning.
7. **long skip(long numchars)-** skips number of characters.
8. **int available()-** Returns number of bytes currently available for reading.

b. Writer : Is an abstract class that defines streaming character output. All the methods in this class returns a **void** value & throws an **IOException.** The methods are-

1. **void close() -** to close the OutputStream
2. **void write (int ch) -** Writes a single character to an output stream.
3. **void write(char buffer[ ]) -** Writes a complete array of characters to an output stream.
4. **void write (char buffer[ ], int offset, int numchars) -** Writes a sub range of numchars from the array buffer, beginning at buffer[offset].
5. **void write(String str)-** Writes str to output stream.
6. **void write(String str, int offset, int numchars)-** Writes a subrange of numchars from string beginning at offset.
7. **void flush() -** clears the buffer.

The following table lists the Character Stream classes

Stream Class	Meaning
BufferedReader	Buffered input character stream
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array
FileReader	Input stream that reads from a file
FileWriter	Output stream that writes to a file
FilterReader	Filtered reader
FilterWriter	Filtered writer
InputStreamReader	Input stream that translates bytes to characters
LineNumberReader	Input stream that counts lines
OutputStreamWriter	Output stream that translates characters to bytes
PipedReader	Input pipe
PipedWriter	Output pipe
PrintWriter	Output stream that contains **print( )** and **println( )**
PushbackReader	Input stream that allows characters to be returned to the input stream
Reader	Abstract class that describes character stream input
StringReader	Input stream that reads from a string
StringWriter	Output stream that writes to a string
Writer	Abstract class that describes character stream output

RandomAccessFile

Random access files permit nonsequential, or random, access to a file's contents. To access a file randomly, you open the file, seek a particular location, and read from or write to that file. When opening a file using a RandomAccessFile, you can choose whether to open it read-only or read write

```
RandomAccessFile (File file, String mode) throws FileNotFoundException
RandomAccessFile (String filePath, String mode) throws FileNotFoundExcep
tion
```

The value of mode can be one of these:

"r" Open **for** reading only.
"rw" Open **for** reading and writing.

Methods:

1. `position` – Returns the current position
2. `position(long)` – Sets the position
3. `read(ByteBuffer)` – Reads bytes into the buffer from the stream
4. `write(ByteBffer)` – Writes bytes from the buffer to the stream
5. `truncate(long)` – Truncates the file (or other entity) connected to the stream

Example:

```
File f = new File("data.dat");
//Open the file for both reading and writing
RandomAccessFile rand = new RandomAccessFile(f,"rw");
rand.seek(f.length());  //Seek to end of file
rand.writeBytes("Append this line at the end");  //Write end of file
rand.close();
System.out.println("Write-Successful");
```

Self Activity

1. Sample program

```
/* Program to count occurrences of a string within a text file*/
import java.io.*;
import java.util.*;
public class TextFileReadApp
{
  public static void main (String arg[]) {

    File f = null;
    // Get the file from the argument line.
    if (arg.length > 0)
          f = new File (arg[0]);
    if (f == null || !fe.exists ()) {
      System.exit(0);
    }

    String string_to_find = arg[1];
    int num_lines = 0;
    try {
      FileReader file_reader = new FileReader (f);
      BufferedReader buf_reader = new BufferedReader (file_reader);
      // Read each line and search string
      do {
         String line = buf_reader.readLine ();
         if (line == null) break;
         if (line.indexOf(string_to_find) != -1) num_lines++;
      } while (true);
      buf_reader.close ();
    }
    catch (IOException e) {
        System.out.println ("IO exception =" + e );
```

```
     }
     System.out.println ("No of lines containing " + string_to_find +
" = " + num_lines);
   } // main
} //class TextFileReadApp
```

Compile this program and pass two command line arguments: filename and string to search.

2. Sample program

```
/* Program to write and read primitive types to a file */
import java.io.*;
class PrimitiveTypes {
   public static void main(String args[]) throws IOException {
      FileOutputStream fos=new FileOutputStream("info.dat");
      DataOutputStream dos=new DataOutputStream(fos);
      dos.writeInt(25); dos.writeBoolean(true);
      dos.writeChar('A'); dos.writeDouble(5.45);
      fos.close();

      FileInputStream fis=new FileInputStream("info.dat")   ;
      DataInputStream dis=new DataInputStream(fis);
      int num =dis.readInt(); boolean b=dis.readBoolean();
      char ch=dis.readChar(); double dbl= dis.readDouble();
      System.out.println("Int- "+num +"\nBoolean- "+b);
      System.out.println("\nCharacter- "+ch+"\nDouble- "+dbl);
      fis.close();
   }
}
```

3. Sample program

```
/* Program to read integers from a file using Scanner class*/
import java.io.*;
import java.util.*;
class ReadIntegers {
   public static void main(String args[]) throws IOException {
      FileReader file = new FileReader("numbers.txt");
      Scanner sc = new Scanner(file);
      int sum=0, num;
      while(sc.hasNext())
      {
          num = sc.nextInt();
          System.out.println("Number = "+ num);
          sum = sum+num;
       }
       System.out.println("The sum = "+ sum);
      file.close();
   }
}
```

Lab Assignments

SET A

1. Write a program to accept a string as command line argument and check whether it is a file or directory. If it is a directory, list the contents of the directory, count how many files the directory has and delete all files in that directory having extension .txt. (Ask the user if the files have to be deleted). If it is a file, display all information about the file (path, size, attributes etc).

2. Write a menu driven program to perform the following operations on a text file "phone.txt" which contains name and phone number pairs. The menu should have options:
   i. Search name and display phone number
   ii. Add a new name-phone number pair.

SET B

1. Write a program to read item information (id, name, price, qty) in file "item.dat". Write a menu driven program to perform the following operations using Random access file:
   i. Search for a specific item by name. ii. Find costliest item. ii. Display all items and total cost

Additional Programs for practice

1. Accept the names of two files and copy the contents of the first to the second. Add Author name and Date in comments in the beginning of file. Add the comment 'end of file' at the end.

2. Write a Java program to accept an option, string and file name using command line argument. Perform following operations:
   a. If no option is passed then print all lines in the file containing the string.
   b. If the option passed is –c then print the count of lines containing the string.
   c. If the option passed is –v then print the lines not containing the string.

Signature of the instructor ☐ Date ☐

Assignment Evaluation

0: Not done ☐	2: Late Complete ☐	4: Complete ☐
1: Incomplete ☐	3: Needs improvement ☐	5: Well Done ☐

Assignment 6: GUI Designing, Event Handling and Applets

Objectives

- **To demonstrate GUI creation using Swing package and Layout managers.**
- **Understand the Event Handling mechanism in java.**
- **Using Event classes, Event Listeners and Adapters.**
- **Creating java applets which run in a web browser.**

Reading

You should read the following topics before starting this exercise

1. AWT and Swing concepts.
2. Layout managers in java
3. Containers and Components
4. Adding components to containers
5. Event sources, listeners and delegation event model
6. Adapter classes
7. Applet tag, Applet class, applet methods

Ready Reference

Graphical User Interface elements are implemented in two java packages – AWT and Swing. Swing is the newer package and swing classes are based on AWT classes.

Swing Architecture:
The design of the Swing component classes is based on the Model-View-Controller architecture, or MVC.

1. The model stores the data.
2. The view creates the visual representation from the data in the model.
3. The controller deals with user interaction and modifies the model and/or the view.

Swing Classes:
The following table lists some important Swing classes and their description.

Class	**Description**
Box	Container that uses a BoxLayout
JApplet	Base class for Swing applets
JButton	Selectable component that supports text/image display
JCheckBox	Selectable component that displays state to user
JCheckBoxMenuItem	Selectable component for a menu; displays state to user
JColorChooser	For selecting colors
JComboBox	For selecting from a drop-down list of choices
JComponent	Base class for Swing components
JDesktopPane	Container for internal frames
JDialog	Base class for pop-up subwindows
JEditorPane	For editing and display of formatted content
JFileChooser	For selecting files and directories
JFormattedTextField	For editing and display of a single line of formatted text
JFrame	Base class for top-level windows
JInternalFrame	Base class for top-level internal windows

Class	Description
JLabel	For displaying text/images
JLayeredPane	Container that supports overlapping components
JList	For selecting from a scrollable list of choices
JMenu	Selectable component for holding menu items; supports text/image display
JMenuBar	For holding menus
JMenuItem	Selectable component that supports text/image display
JOptionPane	For creating pop-up messages
JPanel	Basic component container
JPasswordField	For editing and display of a password
JPopupMenu	For holding menu items and popping up over components
JProgressBar	For showing the progress of an operation to the user
JRadioButton	Selectable component that displays state to user; included in ButtonGroup to ensure that only one button is selected
JRadioButtonMenuItem	Selectable component for menus; displays state to user; included in ButtonGroup to ensure that only one button is selected
JRootPane	Inner container used by JFrame, JApplet, and others
JScrollBar	For control of a scrollable area
JScrollPane	To provide scrolling support to another component
JSeparator	For placing a separator line on a menu or toolbar
JSlider	For selection from a numeric range of values
JSpinner	For selection from a set of values, from a list, a numeric range, or a date range
JSplitPane	Container allowing the user to select the amount of space for each of two components
JTabbedPane	Container allowing for multiple other containers to be displayed; each container appears on a tab
JTable	For display of tabular data
JTextArea	For editing and display of single-attributed textual content
JTextField	For editing and display of single-attributed textual content on a single line
JTextPane	For editing and display of multi-attributed textual content
JToggleButton	Selectable component that supports text/image display; selection triggers component to stay "in"
JToolBar	Draggable container
JToolTip	Internally used for displaying tool tips above components
JTree	For display of hierarchical data
JViewport	Container for holding a component too big for its display area
JWindow	Base class for pop-up windows

Layout Manager

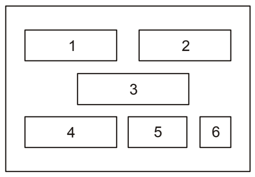
The job of a layout manager is to arrange components on a container. Each container has a layout manager associated with it. To change the layout manager for a container, use the **setLayout()** method.

Syntax

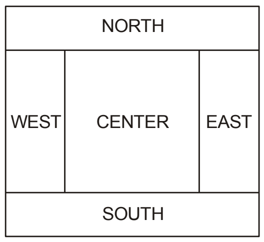
```
setLayout(LayoutManager obj)
```

The predefined managers are listed below:

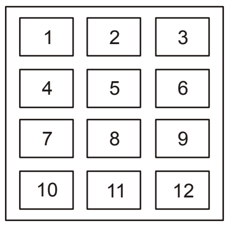
1.	FlowLayout	2.BorderLayout	3.GridLayout
4.	BoxLayout	5.CardLayout	6.GridBagLayout



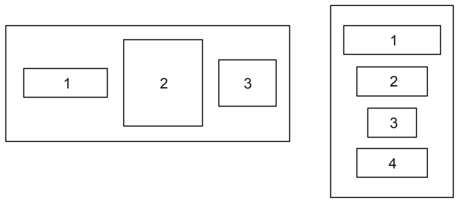
FlowLayout



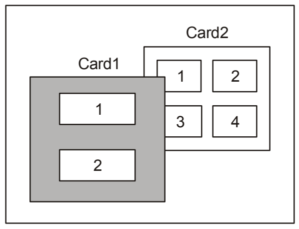
BorderLayout



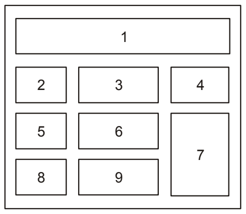
GridLayout



BoxLayout



CardLayout



GridBagLayout

Examples:
```
JPanel p1 = new JPanel()
p1.setLayout(new FlowLayout());
p1.setLayout(new BorderLayout());
p1.setLayout(new GridLayout(3,4));
```

Important Containers:

<u>1. JFrame</u> – This is a top-level container which can hold components and containers like panels.

Constructors

```
JFrame()
JFrame(String title)
```

Important Methods

setSize(int width, int height)	-Specifies size of the frame in pixels
setLocation(int x, int y)	-Specifies upper left corner
setVisible(boolean visible)	-Set true to display the frame
setTitle(String title)	-Sets the frame title
setDefaultCloseOperation(int mode)	-Specifies the operation when frame is closed. The modes are:
	JFrame.EXIT_ON_CLOSE JFrame.DO_NOTHING_ON_CLOSE
	JFrame.HIDE_ON_CLOSE JFrame.DISPOSE_ON_CLOSE
pack()	-Sets frame size to minimum size required to hold components

<u>2. JPanel</u> – This is a middle-level container which can hold components and can be added to other containers like frame and panels.

Constructors

```
public javax.swing.JPanel(java.awt.LayoutManager, boolean);
public javax.swing.JPanel(java.awt.LayoutManager);
public javax.swing.JPanel(boolean);
public javax.swing.JPanel();
```

Important Components :

1. Label : With the JLabel class, you can display unselectable text and images.

Constructors-

```
JLabel(Icon i)                JLabel(Icon I , int n)
JLabel(String s)              JLabel(String s, Icon i, int n)
JLabel(String s, int n)       JLabel()
```

The int argument specifies the horizontal alignment of the label's contents within its drawing area; defined in the SwingConstants interface (which JLabel implements): LEFT (default), CENTER, RIGHT, LEADING, or TRAILING.

Methods-

1. Set or get the text displayed by the label.
   void setText(String)   String getText()
2. Set or get the image displayed by the label.
   void setIcon   (Icon)   Icon getIcon()
3. Set or get the image displayed by the label when it's disabled. If you don't specify a disabled image, then the look-and-feel creates one by manipulating the default image.
   void setDisabledIcon(Icon)   Icon getDisabledIcon()
4. Set or get where in the label its contents should be placed. For vertical alignment: TOP, CENTER (the default), and BOTTOM.
   void setHorizontalAlignment(int)   void setVerticalAlignment(int)
   int getHorizontalAlignment()   int getVerticalAlignment()

2. Button

A Swing button can display both text and an image. The underlined letter in each button's text shows the *mnemonic* which is the keyboard alternative.

Constructors-

```
JButton(Icon I)
JButton(String s)
JButton(String s, Icon I)
```

Methods-

void setDisabledIcon(Icon)   void setPressedIcon(Icon)
void setSelectedIcon(Icon)   void setRolloverIcon(Icon)
String getText()   void setText(String)

Event- ActionEvent

3. Check boxes

Class- *JCheckBox*

Constructors-

```
JCheckBox(Icon i)              JCheckBox(Icon i,booean state)
JCheckBox(String s)            JCheckBox(String s, boolean state)
JCheckBox(String s, Icon i)    JCheckBox(String s, Icon I, boolean state)
```

Methods-

void setSelected(boolean state)   String getText()
void setText(String s)

Event- ItemEvent

4. Radio Buttons

Class- JRadioButton

Constructors-

```
JRadioButton (String s)       JRadioButton(String s, boolean state)
JRadioButton(Icon i)          JRadioButton(Icon i, boolean state)
JRadioButton(String s, Icon i) JRadioButton(String s, Icon i, boolean
state)
JRadioButton()
```

To create a button group- ButtonGroup()
Adds a button to the group, or removes a button from the group.
void add(AbstractButton) void remove(AbstractButton)

5. Combo Boxes
Class- JComboBox
Constructors- `JComboBox()`
Methods-

void addItem(Object)	Object getItemAt(int)
Object getSelectedItem()	int getItemCount()

Event- ItemEvent

6. List
Constructor- `JList(ListModel)`
List models-

1. SINGLE_SELECTION - Only one item can be selected at a time. When the user selects an item, any previously selected item is deselected first.
2. SINGLE_INTERVAL_SELECTION- Multiple, contiguous items can be selected. When the user begins a new selection range, any previously selected items are deselected first.
3. MULTIPLE_INTERVAL_SELECTION- The default. Any combination of items can be selected. The user must explicitly deselect items.

Methods-

boolean isSelectedIndex(int)	void setSelectedIndex(int)
void setSelectedIndices(int[])	void setSelectedValue(Object, boolean)
void setSelectedInterval(int, int)	int getSelectedIndex()
int getMinSelectionIndex()	int getMaxSelectionIndex()
int[] getSelectedIndices()	Object getSelectedValue()
Object[] getSelectedValues()	

Example-
```
listModel = new DefaultListModel();
listModel.addElement("India");
listModel.addElement("Japan");
listModel.addElement("France");
listModel.addElement("Denmark");
list = new JList(listModel);
```
Event- ActionEvent

7. Text classes
All text related classes are inherited from JTextComponent class
a. JTextField
Creates a text field. The int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom document for the field.
Constructors-
```
JTextField()                    JTextField(String)
JTextField(String, int)         JTextField(int)
JTextField(Document, String, int)
```

b. JPasswordField

Creates a password field. When present, the int argument specifies the desired width in columns. The String argument contains the field's initial text. The Document argument provides a custom document for the field.

Constructors-

```
JPasswordField()               JPasswordField(String)
JPasswordField(String, int)    JPasswordField(int)
JPasswordField(Document, String, int)
```

Methods-

1. Set or get the text displayed by the text field.
   void setText(String)   String getText()
2. Set or get the text displayed by the text field.
   char[] getPassword()
3. Set or get whether the user can edit the text in the text field.
   void setEditable(boolean)   boolean isEditable()
4. Set or get the number of columns displayed by the text field. This is really just a hint for computing the field's preferred width.
   void setColumns(int);   int getColumns()
5. Get the width of the text field's columns. This value is established implicitly by the font.
   int getColumnWidth()
6. Set or get the echo character i.e. the character displayed instead of the actual characters typed by the user.
   void setEchoChar(char)   char getEchoChar()

Event- ActionEvent

c. JTextArea

Represents a text area which can hold multiple lines of text

Constructors-

```
JTextArea (int row, int cols)
JTextArea (String s, int row, int cols)
```

Methods-

void setColumns (int cols)   void setRows (int rows)
void append(String s)   void setLineWrap (boolean)

8. Dialog Boxes

Types-

1. Modal- wont let the user interact with the remaining windows of application until first deals with it. Ex- when user wants to read a file, user must specify file name before prg. can begin read operation.
2. Modeless dialog box- Lets the user enters information in both, the dialog box & remainder of application ex- toolbar.

Swing has a JOptionPane class, that lets you put a simple dialog box.
Methods in JOption Class

1. static void showMessageDialog()- Shows a message with ok button.
2. static int showConfirmDialog()- shows a message & gets users options from set of options.
3. static int showOptionDialog- shows a message & get users options from set of options.
4. String showInputDialog()- shows a message with one line of user input.

9. Menu

Creating and Setting Up Menu Bars	
Constructor or Method	**Purpose**
JMenuBar()	Creates a menu bar.
JMenu add(JMenu)	Creates a menu bar.
void setJMenuBar(JMenuBar) JMenuBar getJMenuBar()	Sets or gets the menu bar of an applet, dialog, frame, internal frame, or root pane.
Creating and Populating Menus	

JMenu()
JMenu(String)
	Creates a menu. The string specifies the text to display for the menu.

JMenuItem add(JMenuItem) JMenuItem add(Action) JMenuItem add(String)	Adds a menu item to the current end of the menu. If the argument is an Action object, then the menu creates a menu item. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.
void addSeparator()	Adds a separator to the current end of the menu.
JMenuItem insert(JMenuItem, int) JMenuItem insert(Action, int) void insert(String, int) void insertSeparator(int)	Inserts a menu item or separator into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The JMenuItem, Action, and String arguments are treated the same as in the corresponding add methods.
void remove(JMenuItem) void remove(int) void removeAll()	Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.
Implementing Menu Items	

JMenuItem()
JMenuItem(String)
JMenuItem(Icon)
JMenuItem(String, Icon)
JMenuItem(String, int)
	Creates an ordinary menu item. The icon argument, if present, specifies the icon that the menu item should display. Similarly, the string argument specifies the text that the menu item should display. The integer argument specifies the keyboard mnemonic to use. You can specify any of the relevant VK constants defined in the KeyEvent class. For example, to specify the A key, use KeyEvent.VK_A.

JCheckBoxMenuItem()
JCheckBoxMenuItem(String)
JCheckBoxMenuItem(Icon)
JCheckBoxMenuItem(String, Icon)
JCheckBoxMenuItem(String, boolean)
JCheckBoxMenuItem(String, Icon, boolean)
	Creates a menu item that looks and acts like a check box. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected (checked). Otherwise, the menu item is initially unselected.

JRadioButtonMenuItem()
JRadioButtonMenuItem(String)
JRadioButtonMenuItem(Icon)
JRadioButtonMenuItem(String, Icon)
JRadioButtonMenuItem(String, boolean)
JRadioButtonMenuItem(Icon, boolean)
JRadioButtonMenuItem(String, Icon, boolean)
	Creates a menu item that looks and acts like a radio button. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected. Otherwise, the menu item is initially unselected.

void setState(boolean) boolean getState() *(in* JCheckBoxMenuItem*)*	Set or get the selection state of a check box menu item.

void setEnabled(boolean)	If the argument is true, enable the menu item. Otherwise, disable the menu item.

Event handling is an important part of GUI based applications. Events are generated by event sources. A mouse click, Window closed, key typed etc. are examples of events.
All java events are sub-classes of **java.awt.AWTEvent** class.

Java has two types of events:

1. **Low-Level Events:** Low-level events represent direct communication from user. A low level event is a key press or a key release, a mouse click, drag, move or release, and so on. Following are low level events.

Event	**Description**
ComponentEvent	Indicates that a component object (e.g. Button, List, TextField) is moved, resized, rendered invisible or made visible again.
FocusEvent	Indicates that a component has gained or lost the input focus.
KeyEvent	Generated by a component object (such as TextField) when a key is pressed, released or typed.
MouseEvent	Indicates that a mouse action occurred in a component. E.g. mouse is pressed, releases, clicked (pressed and released), moved or dragged.
ContainerEvent	Indicates that a container's contents are changed because a component was added or removed.
WindowEvent	Indicates that a window has changed its status. This low level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, deiconified or when focus is transferred into or out of the Window.

2. High-Level Events: High-level (also called as semantic events) events encapsulate the meaning of a user interface component. These include following events.

Event	**Description**
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action occurs (such as being pressed).
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars.
ItemEvent	Indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.
TextEvent	Indicates that an object's text changed. This high-level event is generated by an object (such as TextComponent) when its text changes.

The following table lists the events, their corresponding listeners and the method to add the listener to the component.

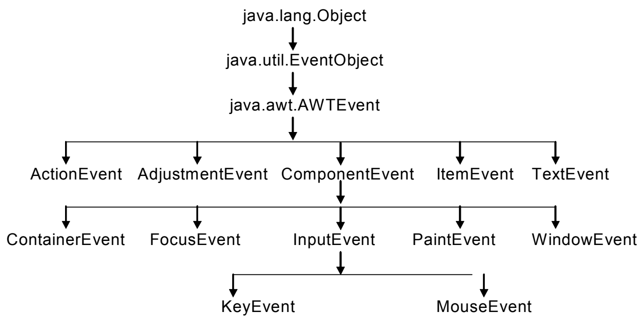
Event	**Event Source**	**Event Listener**	**Method to add listener to event source**
Low-level Events			
ComponentEvent	Component	ComponentListener	addComponentListener()
FocusEvent	Component	FocusListener	addFocusListener()
KeyEvent	Component	KeyListener	addKeyListener()

MouseEvent	Component	MouseListener
		MouseMotionListener
			addMouseListener()
			addMouseMotionListener()

ContainerEvent	Container	ContainerListener	addContainerListener()
WindowEvent	Window	WindowListener	addWindowListener()
High-level Events			
ActionEvent	Button List MenuItem TextField	ActionListener	addActionListener()

ItemEvent	Choice CheckBox CheckBoxMenuItem List	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()
TextEvent	TextField TextArea	TextListener	addTextLIstener()

Event class hierarchy



Listener Methods:

Methods	**Description**
ComponentListener	
componentResized(ComponentEvent e)	Invoked when component's size changes.
componentMoved(ComponentEvent e)	Invoked when component's position changes.
componentShown(ComponentEvent e)	Invoked when component has been made visible.
componentHidden(ComponentEvent e)	Invoked when component has been made invisible.
FocusListener	
focusGained(FocusEvent e)	Invoked when component gains the keyboard focus.
focusLost(FocusEvent e)	Invoked when component loses the keyboard focus.
KeyListener	
keyTyped(KeyEvent e)	Invoked when a key is typed.
keyPressed(KeyEvent e)	Invoked when a key is pressed.
keyReleased(KeyEvent e)	Invoked when a key is released.
MouseListener	
mouseClicked(MouseEvent e)	Invoked when a mouse button is clicked (i.e. pressed and released) on a component.
mousePressed(MouseEvent e)	Invoked when a mouse button is pressed on a component.
mouseReleased(MouseEvent e)	Invoked when a mouse button is released on a component.
mouseEntered(MouseEvent e)	Invoked when a mouse enters a component.
mouseExited(MouseEvent e)	Invoked when a mouse exits a component.
MouseMotionListener	
mouseDragged(MouseEvent e)	Invoked when a mouse button is pressed on a component and then dragged.
mouseMoved(MouseEvent e)	Invoked when a the mouse cursor is moved on to a component but mouse button is not pressed.
ContainerListener	
componentAdded(ContainerEvent e)	Invoked when a component is added to the container.
componentRemoved(ContainerEvent e)	Invoked when a component is removed from the container.
WindowListener	

windowOpened(WindowEvent e)	Invoked the first time a window is made visible
windowClosing(WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
windowClosed(WindowEvent e)	Invoked when a window has been closed as the result of calling dispose on the window.
windowIconified(WindowEvent e)	Invoked when a window is changed from a normal to a minimized state.
windowDeiconified(WindowEvent e)	Invoked when a window is changed from minimized to normal state.
windowActivated(WindowEvent e)	Invoked when the window is set to be the active window.
windowDeactivated(WindowEvent e)	Invoked when the window is no longer the active window.
ActionListener	
actionPerformed(ActionEvent e)	Invoked when an action occurs.
ComponentListsner	
itemStateChanged(ActionEvent e)	Invoked when anitem has been selected oe deselected by the user.
AdjustmentListener	
adjustmentValueChanged(ActionEvent e)	Invoked when the value of the adjustable has changed.
TextListener	
textValueChanged(ActionEvent e)	Invoked when the value of the text has changed.

Adapter Classes:

All high level listeners contain only one method to handle the high-level events. But most low level event listeners are designed to listen to multiple event subtypes (i.e. the MouseListener listens to mouse-down, mouse-up, mouse-enter, etc.). AWT provides a set of abstract "adapter" classes, which implements each listener interface. These allow programs to easily subclass the Adapters and override only the methods representing event types they are interested in, instead of implementing all methods in listener interfaces.

The Adapter classes provided by AWT are as follows:

java.awt.event.ComponenentAdapter	java.awt.event.ContainerAdapter
java.awt.event.FocusAdapter	java.awt.event.KeyAdapter
java.awt.event.MouseAdapter	java.awt.event.MouseMotionAdapter
java.awt.event.WindowAdapter	

Applet

Applets are small java programs which are executed and displayed in a java compatible web browser.

Creating an applet

All applets are subclasses of the **java.applet.Applet** class. You can also create an applet by extending the **javax.swing.JApplet** class. The syntax is:

```
class MyApplet extends Applet
{
  //applet methods
}
```

Applet methods:

Method	Purpose
init( )	Automatically called to perform initialization of the applet. Executed only once.
start( )	Called every time the applet moves into sight on the Web browser to allow the applet to start up its normal operations.
stop( )	Called every time the applet moves out of sight on the Web browser to allow the applet to shut off expensive operations.
destroy( )	Called when the applet is being unloaded from the page to perform final release of resources when the applet is no longer used
paint()	Called each time the applets output needs to be redrawn.

Running an applet

1. Compile the applet code using javac
2. Use the java tool – appletviewer to view the applet (embed the APPLET tag in comments in the code)
3. Use the APPLET tag in an HTML page and load the applet in a browser

Using appletviewer:

1. Write the HTML APPLET tag in comments in the source file.
2. Compile the applet source code using javac.
3. Use appletviewer ClassName.class to view the applet.

Using browser:

1. Create an HTML file containing the APPLET tag.
2. Compile the applet source code using javac.
3. In the web browser, open the HTML file.

The APPLET tag

```
< APPLET
      [CODEBASE   = appletURL]
      CODE        = appletClassFile
      [ALT        = alternateText]
      [ARCHIVE    = archiveFile]
      [NAME       = appletInstanceName]
      WIDTH       = pixels
      HEIGHT            = pixels
      [ALIGN            = alignment]
      [VSPACE     = pixels]
      [HSPACE     = pixels]
>
[< PARAM NAME = AttributeName VALUE = AttributeValue />]
</APPLET>
```

Attribute	Value	Meaning
align	*left* *right* *top* *bottom* *middle* *baseline*	Specifies the alignment of an applet according to surrounding elements
alt	*text*	Specifies an alternate text for an applet
archive	*URL*	Specifies the location of an archive file

code	*URL*	Specifies the file name of a Java applet
codebase	*URL*	Specifies a relative base URL for applets specified in the code attribute
height	*pixels*	Specifies the height of an applet
hspace	*pixels*	Defines the horizontal spacing around an applet
name	*name*	Defines the name for an applet (to use in scripts)
vspace	*pixels*	Defines the vertical spacing around an applet
width	*pixels*	Specifies the width of an applet

The mandatory attributes are CODE, HEIGHT and WIDTH.

Examples:
```
1. <applet code=MyApplet width=200 height=200 archive="files.jar">
   </applet>
2. <applet code=Simple.class width=100 height=200 codebase=”example/”>
   </applet>
```

Passing parameters to applets

The PARAM tag allows us to pass information to an applet when it starts running.
A parameter is a NAME – VALUE pair. Every parameter is identified by a name and it has a value.
```
< PARAM NAME = AttributeName VALUE = AttributeValue />
```

Example:
```
<APPLET     NAME = "MyApplet.class" WIDTH = 100 HEIGHT = 100>
<PARAM NAME = "ImageSource" VALUE = "project/images/">
<PARAM NAME = "BackgroundColor" VALUE = "0xc0c0c0">
<PARAM NAME = "FontColor" VALUE = "Red">
</APPLET>
```

The Applet can retrieve information about the parameters using the **getParameter()** method.
```
String getParameter(String parameterName);
```
Example:
```
String dirName = getParameter(“ImageSource”);
Color c = new Color( Integer.parseInt(getParameter(“BackgroundColor”)));
```

paint(), repaint() and update()
The paint() method redraws the applet. The repaint() method is used to force redrawing of the applet. The update() method redraws only a portion of the applet.

Self Activity

1. Sample program
```
/* Program to demonstrate Button and text field */
      import java.awt.event.*;
      import javax.swing.*;
      import java.awt.*;
      public class JButtonDemo extends JFrame implements
ActionListener
      {
            JTextField jtf; JButton jb;
               public JButtonDemo()
               {
                  setLayout(new FlowLayout());
                  jtf=new JTextField(15);
                  add (jtf);
                  jb=new JButton (“Click Me”);
                  jb.addActionListener (this);
                  add(jb);
                  setSize(200,200);
                  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  setVisible(true);
                 }
            public void actionPerformed(ActionEvent ae)
            { jtf.setText (ae.getActionCommand()); }
            public static void main(String[] args){
                  new JButtonDemo();
            }
      }
```

2. Sample program

```
/* Program to demonstrate Combobox */
import java.awt.*; import javax.swing.*;  import java.awt.event.*;
public class JCdemo extends JFrame implements ItemListener
{
      JTextField jtf; JCheckBox jcb1, jcb2;
      public JCdemo()
      {
      setLayout(new FlowLayout());
      jcb1=new JCheckBox("Swing Demos");
      jcb1.addItemListener(this); add(jcb1);
      jcb2=new JCheckBox("Java Demos");
      jcb2.addItemListener(this); add(jcb2);

      jtf=new JTextField(35); add(jtf);
      setSize(200,200);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
      public void itemStateChanged (ItemEvent ie)
      {         String text = " ";
                if(jcb1.isSelected())
                   text = text + jcb1.getText() + " ";
                if(jcb2.isSelected())
                   text = text + jcb2.getText();
              jtf.setText(text);
      }
      public static void main(String[] args){
        new JCdemo();
      }
}
```

3. Sample program

```
/* Program to demonstrate Radio Button */
import java.awt.*;  import javax.swing.*;  import java.awt.event.*;
public class JRdemo extends JFrame implements ActionListener
{
      JTextField jtf;
      JRadioButton jrb1,jrb2;  ButtonGroup bg;
      public JRdemo()
      {
            setLayout(new FlowLayout());
            bg=new ButtonGroup();

            jrb1=new JRadioButton("A");
            jrb1.addActionListener(this);
            bg.add(jrb1); add(jrb1);

            jrb2=new JRadioButton("B");
            jrb2.addActionListener(this);
            bg.add(jrb2); add(jrb2);
            jtf=new JTextField(5); add(jtf);
            setSize(200,200);
            setVisible(true);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
      public void actionPerformed (ActionEvent ae)
      { jtf.setText(ae.getActionCommand()); }
      public static void main(String[] args)
      {  new JRdemo(); }
}
```

4. Sample program

```
/* Program to handle mouse movements and key events on a frame*/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class EventTest extends JFrame
{
  JLabel l = new JLabel();
  EventTest()
  {
    setLayout(new FlowLayout());
    add(l);
    addKeyListener(new KeyAdapter()
    {
      public void keyTyped(KeyEvent ke)
      {
        l.setText("You typed " + ke.getKeyChar());
      }
    });
    addMouseMotionListener(new MouseMotionAdapter()
    {
      public void mouseMoved(MouseEvent me)
      {
        l.setText("Mouse moved : X = "+ me.getX() + "Y = " +
me.getY());
      }
    });
   setSize(200,200);
   setVisible(true);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args)
  {
    new EventTest();
  }
}
```

5. Sample program

```
/* Program to  display a message in an applet*/
import java.awt.*;
import java.applet.*;
/*
<applet code="MyApplet.class" width=200 height=100>
</applet>
```

```
*/
public class MyApplet extends Applet
{
  public void paint(Graphics g)
  {
    g.drawString("My First Applet", 20,20);
  }
}
```

Save this as MyApplet.java. Compile and Execute it using command – appletviewer MyApplet.class

6. Sample program

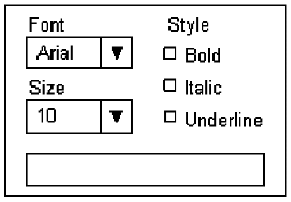
```
/* Applet with components*/
import java.awt.*;
import javax.swing.* ;
import java.applet.*;
/*
<applet code="MyApplet.class" width=200 height=100>
</applet>
*/
public class MyApplet extends Applet
{
  JPanel p;   JTextField t;   JButton b;
  public void init()
  {
    p = new JPanel();
    p.setLayout(new FlowLayout());
    t = new JTextField(20);
    b = new JButton("Click");
    p.add(t); p.add(b);
    add(p);
  }
}
```

Save this as MyApplet.java Compile the file. Execute it using command – appletviewer MyApplet.class

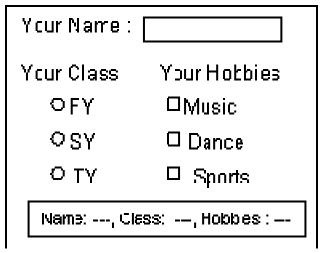
Lab Assignments

SET A

1. Write a program to create the following GUI and apply the changes to the text in the TextField.



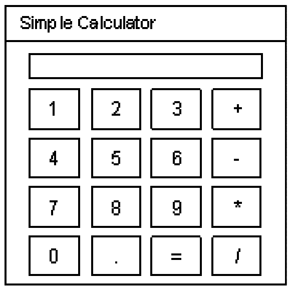
2. Create the following GUI screen using appropriate layout managers. Accept the name, class , hobbies of the user and display the selected options in a text box.



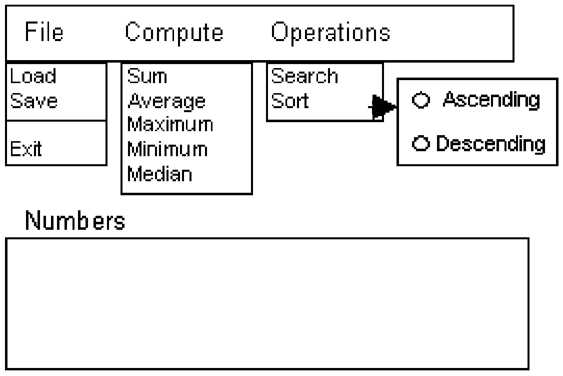
3. Create an Applet which displays a message in the center of the screen. The message indicates the events taking place on the applet window. Handle events like mouse click, mouse moved, mouse dragged, mouse pressed, and key pressed. The message should update each time an event occurs. The message should give details of the event such as which mouse button was pressed, which key is pressed etc. (Hint: Use repaint(), KeyListener, MouseListener, MouseEvent method getButton, KeyEvent methods getKeyChar)

SET B

1.Write a java program to implement a simple arithmetic calculator. Perform appropriate validations.



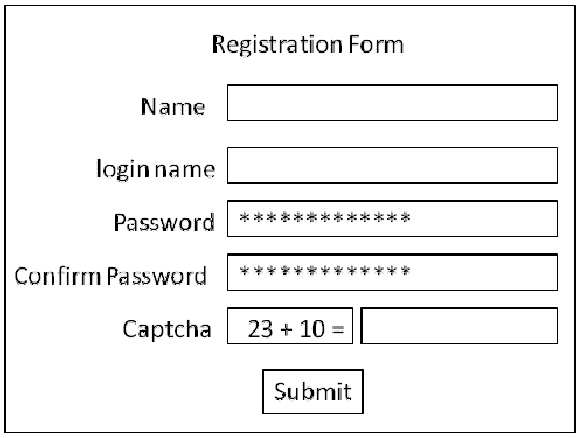
2. Write a menu driven program to perform the following operations on a set of integers. The Load operation should generate 50 random integers (2 digits) and display the numbers on the screen. The save operation should save the numbers to a file “numbers.txt”. The Compute menu provides various operations and the result is displayed in a message box. The Search operation accepts a number from the user in an input dialog and displays the search result in a message dialog. The sort operation sorts the numbers and displays the sorted data on the screen.



3. Write a java program to create the following GUI for user registration form. Perform the following validations:

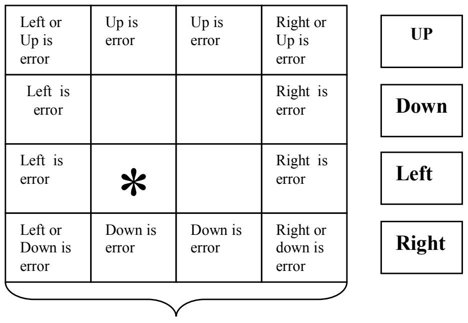
i. Password should be minimum 6 characters containing atleast one uppercase letter, one digit and one symbol.
ii. Confirm password and password fields should match.
iii. The Captcha should generate two random 2 digit numbers and accept the sum from the user.

If above conditions are met, display "Registration Successful" otherwise "Registration Failed" after the user clicks the Submit button.



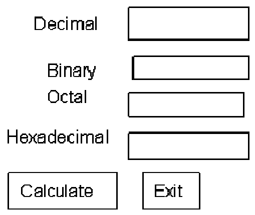
Additional programs for practice

1.Create an application in Java using swing that will move star towards up, down, left and right. Display appropriate message if it crosses the boundary. Design the screen as shown:

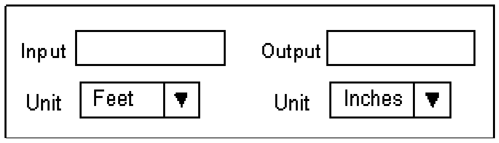


Label Field

2. Create a GUI and program for number conversion from decimal to binary, octal and hexadecimal when the user clicks on "Calculate".



3. Create a conversion applet which accepts value in one unit and converts it to another. The input and output unit is selected from a list. Perform conversion to and from Feet, Inches, Centimeters, Meters and Kilometers.



Signature of the instructor		Date	

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Savitribai Phule Pune University

सावित्रीबाई फुले पुणे विद्यापीठ

T. Y. B. Sc.

(Computer Science)

CS-348 Laboratory Course II
(Java Programming - II)

(As per new syllabus w.e.f. academic year 2015-2016)

Semester II

Name ________________________________ **Roll No.** _______

College _______________________________ **Division** _______

Academic Year _____________

CO-ORDINATOR:

PROF. MS. POONAM PONDE

NOWROSJEE WADIA COLLEGE, PUNE

EDITED BY :

Ms. Poonam Ponde

INPUTS GIVEN BY :

Mr. Jeevan Tonde

Ms. Rasika Rahalkar

PREVIOUS EDITION AUTHORS:

Ms. Poonam Ponde

Mr. Jeevan Limaye

Mr. Sachin Bhoite

Ms. Kalpana Joshi

ABOUT THE WORK BOOK

- **OBJECTIVES OF THIS BOOK**

This lab-book is intended to be used by T.Y.B.Sc(Computer Science) students for Laboratory course – II (Java programming), Semester II.

The objectives of this book are

a. Covers the complete scope of the syllabus.
b. Bringing uniformity in the way course is conducted across different colleges.
c. Continuous assessment of the students.
d. Providing ready references for students while working in the lab.

- **Instructions to the students**

1. This book is mandatory for the completion of the laboratory course. Students should carry this book during practical sessions.
2. Student should read the topics mentioned in **Reading section** of this book before coming for the practical session.
3. Students should get the assignments completed and checked in time with marks assigned by the instructor in the index page.
4. Only Set A and B programs are compulsory. Programs in Set C are only for additional practice.
5. Each assignment will be assessed on a scale of 0 to 5 as indicated below.

i) Not done	0
ii) Incomplete	1
iii) Late Complete	2

iv) Needs improvement	3
v) Complete	4
vi) Well Done	5

- **Difficulty Levels**

Self Activity : Students should solve these exercises for practice only.
SET A - Easy : All exercises are compulsory.
SET B - Medium : All exercises are compulsory.
SET C - Difficult : Not Compulsory.

- **Instruction to the Instructors**

1) Make sure that students follow the instruction as given above.
2) Evaluate each assignment on a scale of 5 as specified above by ticking appropriate box.
3) The value should also be entered on assignment completion page of the respective Lab course.
4) Students can use IDE like Eclipse for performing the assignments.
5) The database used for JDBC assignment should be postgresql.
6) The assignments should be performed on Linux operating system.

Assignment Completion Sheet

Sr. No	**Assignment Name**	**Marks**
1	Collections	
2	Database programming	
3	Servlets	
4	Java Server Pages	
5	Multithreading	
6	Networking	
	Total	

Signature of Incharge

Date:

Internal Examiner **External Examiner**

Date:

Assignment 1: Collections

Objectives

- **Study the Collections framework in java**
- **Use various collections**

Reading

You should read the following topics before starting this exercise:

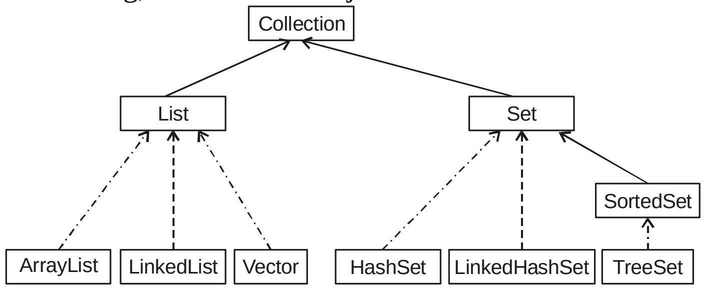
1. Concept of Collection
2. classes and interfaces in the Collections framework
3. Concept of iterator.
4. Creating and using collections objects.

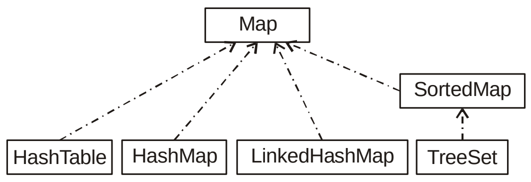
Ready Reference

A *collection* is an object that represents a group of objects. A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.
A collections framework is a unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of their representation.

It contains the following:

i. **Interfaces:** Interfaces allow collections to be manipulated independently of the details of their representation. Interfaces generally form a hierarchy.
ii. **Implementations:** These are the concrete implementations of the collection interfaces.
iii. **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on collection objects.





Interfaces:

Interface Name	Description
`Collection`	The most general collection interface type. A collection represents a group of objects known as its *elements*. The Java platform doesn't provide any direct implementations of this interface.
`List`	Represents an ordered collection. Lists can contain duplicate elements.

`Set`	Represents an unordered collection that does not permit duplicate elements.
`SortedSet`	Represents a set whose elements are maintained in a sorted order.
`Queue`	A collection used to hold multiple elements prior to processing. A Queue provides additional insertion, extraction and inspection operations.
`Map`	Represents key-value pairs. A Map cannot contain duplicate keys; each key can map to at most one value. Does not extend collection.
`SortedMap`	Represents a Map that maintains its mappings in ascending key order.

Classes:

Class name	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList.
ArrayList	Implements a dynamic array by extending AbstractList.
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
HashSet	Extends AbstractSet for use with a hash table.
TreeSet	Implements a set stored in a tree. Extends AbstractSet

The Collection interface:

This interface represents a general collection.

Method	Explanation
boolean add(Object element)	Method adds objects in the collection.
boolean remove(Object element)	Method removes objects in the collection.
The Collection interface also supports query operations:	
int size()	Returns the size of the collection.
boolean isEmpty()	Returns true if the collection is empty or false.
boolean contains(Object element)	Returns true if the collection contains the element passed in argument.
Other operations are tasks done on groups of elements or the entire collection at once:	
boolean containsAll(Collection collection)	The ***containsAll()*** method allows you to discover if the current collection contains all the elements of another collection, a *subset*.
boolean addAll(Collection collection)	ensures all elements from another collection are added to the current collection, usually a *union*.
void removeAll(Collection collection)	method is like *clear()* but only removes a subset of elements
void retainAll(Collection collection)	*This* method is similar to the *removeAll()* method but does what might be perceived as the opposite: it removes from the current

	collection those elements not in the other collection, an *intersection*.
void clear()	removes all elements from the current collection

List interface

A list stores a sequence of elements.

Method	Description
`void` **`add`**`(int index, Object element)`	Inserts the specified element at the specified position in this list (optional operation).
`Boolean` **`addAll`**`(int index, Collection c)`	Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
`Object` **`get`**`(int index)`	Returns the element at the specified position in this list.
`int` **`indexOf`**`(Object o)`	Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.
`int` **`lastIndexOf`**`(Object o)`	Returns the index in this list of the last occurrence of the specified element, or -1 if this list does not contain this element.
`ListIterator` **`listIterator`**`()`	Returns a list iterator of the elements in this list (in proper sequence).
`ListIterator` **`listIterator`**`(int index)`	Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list.
`Object` **`remove`**`(int index)`	Removes the element at the specified position in this list (optional operation).
`Object` **`set`**`(int index, Object element)`	Replaces the element at the specified position in this list with the specified element (optional operation).
`int` **`size`**`()`	Returns the number of elements in this list.
`List` **`subList`**`(int fromIndex, int toIndex)`	Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.

Set Interface and SortedSet Interface

A set is a collection that contains no duplicate elements. It contains the methods of the Collection interface. A SortedSet is a `Set` that maintains its elements in ascending order. It adds the following methods:

Method	Description
`Comparator` **`comparator()`**	Returns the comparator associated with this sorted set, or null if it uses its elements' natural ordering.
`Object first()`	Returns the first (lowest) element currently in this sorted set.
`SortedSet headset(Object toElement)`	Returns a view of the portion of this sorted set whose elements are strictly less than toElement.
`Object last()`	Returns the last (highest) element currently in this sorted set.
`SortedSet subset(Object fromElement, Object toElement)`	Returns a view of the portion of this sorted set whose elements range from fromElement, inclusive, to toElement, exclusive.

`SortedSet tailSet(Object fromElement)`	Returns a view of the portion of this sorted set whose elements are greater than or equal to fromElement.

Map Interface

A Map represents an object that maps keys to values. Keys have to be unique.

Method	Description
`void clear()`	Removes all mappings from this map
`boolean containsKey(Object key)`	Returns true if this map contains a mapping for the specified key.
`boolean containsValue(Object value)`	Returns true if this map maps one or more keys to the specified value.
`Set entrySet()`	Returns a set view of the mappings contained in this map.
`boolean equals(Object o)`	Compares the specified object with this map for equality.
`Object get(Object key)`	Returns the value to which this map maps the specified key.
`int hashCode()`	Returns the hash code value for this map.
`boolean isEmpty()`	Returns true if this map contains no key-value mappings
`Set keySet()`	Returns a set view of the keys contained in this map.
`Object put(Object key, Object value)`	Associates the specified value with the specified key in this map
`void putAll(Map t)`	Copies all of the mappings from the specified map to this map
`Object remove(Object key)`	Removes the mapping for this key from this map if it is present (optional operation).
`int size()`	Returns the number of key-value mappings in this map.
`Collection values()`	Returns a collection view of the values contained in this map

Implementations

The general-purpose implementations are summarized in the following table.

General-purpose Implementations

Interfaces	Implementations				
	Hash table	**Resizable array**	**Tree**	**Linked list**	**Hash table + Linked list**
`Set`	`HashSet`		`TreeSet`		`LinkedHashSet`
`List`		`ArrayList`		`LinkedList`	
`Map`	`HashMap`		`TreeMap`		`LinkedHashMap`

List Implementations

There are two general-purpose `List` implementations — `ArrayList` and `LinkedList`.

1. **ArrayList:** Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. Each `ArrayList` instance has a *capacity*. The capacity is the size of the array used to

store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.

2. **LinkedList:** The LinkedList class implements the `List` interface. All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

ArrayList Method name	**Description**
addAll(int index, Collection c)	Inserts all of the elements in the specified Collection into this list, starting at the specified position.
ensureCapacity(int minCapacity)	Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
removeRange(int fromIndex, int toIndex)	Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.
LinkedList Method name	**Description**
`addFirst(Object o)`	Inserts the given element at the beginning of this list.
`addLast(Object o)`	Appends the given element to the end of this list.
`Object getFirst()`	Returns the first element in the list
`Object getLast()`	Returns the last element in the list
`Object removeFirst()`	Removes and returns the first element from this list.
`Object removeLast()`	Removes and returns the last element from this list.
`ListIterator listIterator(int index)`	Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.

Set Implementations

There are three general-purpose `Set` implementations — `HashSet`, `TreeSet`, and `LinkedHashSet`.

1. TreeSet: The elements are internally stored in a search tree. It is useful when you need to extract elements from a collection in a sorted manner.
2. **HashSet: I**t creates a collection the uses a hash table for storage. The advantage of HashSet is that it performs basic operations (`add`, `remove`, `contains` and `size`) in constant time and is faster than TreeSet
3. **LinkedHashSet:** The only difference is that the LinkedHashSet maintains the order of the items added to the Set, The elements are stored in a doubly linked list.

Iterator:

The **Iterator** interface provides methods using which we can traverse any collection. This interface is implemented by all collection classes.

Methods:

`hasNext()`	`true` if there is a next element in the collection.
`next()`	Returns the next object.
`remove()`	Removes the most recent element that was returned by next()

ListIterator

`ListIterator` is implemented only by the classes that implement the `List` interface (`ArrayList`, `LinkedList`, and `Vector`). ListIterator provides the following.

Forward iteration	
`hasNext()`	`true` if there is a next element in the collection.
`next()`	Returns the next object.
Backward iteration	
`hasPrevious()`	`true` if there is a previous element.
`previous()`	Returns the previous element.
Getting the index of an element	
`nextIndex()`	Returns index of element that would be returned by subsequent call to next().
`previousIndex()`	Returns index of element that would be returned by subsequent call to previous().
Optional modification methods.	
`add(`*`obj`*`)`	Inserts obj in collection before the next element to be returned by `next()` and after an element that would be returned by `previous()`.
`set()`	Replaces the most recent element that was returned by `next` or `previous()`.
`remove()`	Removes the most recent element that was returned by `next()` or `previous()`.

- **HashTable**

This class implements a hashtable, which maps keys to values. It is similar to **HashMap**, but is synchronized. The important methods of the HashTable class are:

Method name	Description
`void clear()`	Clears this hashtable so that it contains no keys.
`boolean contains(Object value)`	Tests if some key maps into the specified value in this hashtable.
`boolean containsKey(Object key)`	Tests if the specified object is a key in this hashtable.
`containsValue(Object value)`	Returns true if this Hashtable maps one or more keys to this value.
`Enumeration elements()`	Returns an enumeration of the values in this hashtable.
`Set entrySet()`	Returns a Set view of the entries contained in this Hashtable.
`Object get(Object key)`	Returns the value to which the specified key is mapped in this hashtable.
`int hashCode()`	Returns the hash code value for this Map as per the definition in the Map interface.
`Boolean isEmpty()`	Tests if this hashtable maps no keys to values.
`Enumeration keys()`	Returns an enumeration of the keys in this hashtable.
`Set keySet()`	Returns a Set view of the keys contained in this Hashtable.
`Object put(Object key, Object value)`	Maps the specified key to the specified value in this hashtable.
`void putAll(Map m)`	Copies all of the mappings from the specified Map to this Hashtable These mappings will replace any mappings that this Hashtable had for any of the keys currently in the specified Map.
`void rehash()`	Increases the capacity of and internally reorganizes this hashtable, in order to accommodate and access its entries more efficiently.
`Object remove(Object key)`	Removes the key (and its corresponding value) from this hashtable.
`int size()`	Returns the number of keys in this hashtable.
`Collection values()`	Returns a Collection view of the values contained in this Hashtable.

For traversing a HashTable, use the **Enumeration** interface. The `Enumeration` interface has two methods, **`hasMoreElements`** and **`nextElement`** `which` are same as the `hasNext` and `next` methods of the `Iterator` interface.

Self Activity

1. Sample program

```
/* Program to demonstrate ArrayList and LinkedList */
import java.util.*;
class ArrayLinkedListDemo {
   public static void main(String args[])
   {
      ArrayList al = new ArrayList();
      LinkedList l1 = new LinkedList();
      System.out.println("Initial size of al: " + al.size());
      // add elements to the array list
      al.add("A");
      al.add("B");
      al.add("C");
      al.add(2, "AA");
      System.out.println("Contents of al: " + al);

      al.remove("B");
      al.remove(1);
      System.out.println("Contents of al: " + al);
        l1.add("A");
          l1.add("B");
          l1.add(new Integer(10));
          System.out.println("The contents of list is " + l1);
          l1.addFirst("AA");
          l1.addLast("c");
          l1.add(2,"D");
          l1.add(1,"E");
          l1.remove(3);
          System.out.println("The contents of list is " + l1);
        }
      }
```

2. Sample program

```
/* Program to demonstrate iterator */
import java.util.*;
public class IteratorDemo
{
  public static void main(String[] args)
{
     ArrayList a1 = new ArrayList();

al.add("C"); al.add("A"); al.add("E"); al.add("B"); al.add("D"); al.add("F");
  Iterator itr = a1.iterator();  //obtain iterator
  while(itr.hasNext())
  {
      String elt = (String)itr.next();
      System.out.println("Element = " + elt);
  }

  LinkedList l = new LinkedList();
  l.add("A");  l.add("B");  l.add("C");  l.add("D");
  ListIterator litr = l.listIterator();
  while(litr.hasNext())
  {
       String elt = (String)litr.next();
       System.out.println(elt);
   }
  System.out.println("Traversing Backwards : ");
  while(litr.hasPrevious())
        System.out.println(litr.previous());
}
}
```

3. Sample program

```
/* Program to demonstrate HashTable*/
import java.util.*;
public class HashtableDemo
{
  public static void main(String[] args)
{
    Hashtable hashtable = new Hashtable();
String str, name = null;
hashtable.put( "A", 75.2 ); // adding value into hashtable
hashtable.put( "B", 65.9 );
hashtable.put( "C", 95.1 );
hashtable.put( "D", 85.7 );

System.out.println("Retriving all keys from the Hashtable");
Enumeration keys = hashtable.keys();
while( keys. hasMoreElements() )
             System.out.println( keys.nextElement() );

  System.out.println("Retriving all values from the table");
Enumeration values = hashtable.elements();
while( values. hasMoreElements() )
                    System.out.println( values.nextElement() );
}
}
```

Lab Assignments

SET A

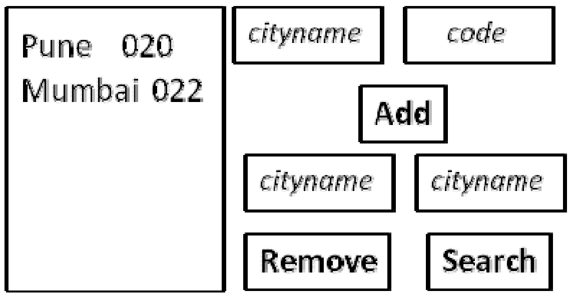
1. Accept 'n' integers from the user and store them in a collection. Display them in the sorted order. The collection should not accept duplicate elements. (Use a suitable collection). Search for an particular element using predefined search method in the Collection framework.

2. Construct a linked List containing names of colors: red, blue, yellow and orange. Then extend your program to do the following:
   i. Display the contents of the List using an Iterator;
   ii. Display the contents of the List in reverse order using a ListIterator;
   iii. Create another list containing pink and green. Insert the elements of this list between blue and yellow.

3. Create a Hash table containing student name and percentage. Display the details of the hash table. Also search for a specific student and display percentage of that student.

SET B

1. Create a java application to store city names and their STD codes using an appropriate collection. The GUI ahould allow the following operations:
   i. Add a new city and its code (No duplicates)
   ii. Remove a city from the collection
   iii. Search for a cityname and display the code



SET C

1. Read a text file, specified by the first command line argument, into a `list`. The program should then display a menu which performs the following operations on the list:
1. Insert line 2. Delete line 3. Append line 4. Modify line 5. Exit
When the user selects Exit, save the contents of the list to the file and end the program.

Signature of the instructor Date / /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 2: Database Programming

Objectives

- **To communicate with a database using java.**
- **To execute queries on tables.**
- **To obtain information about the database and tables**

Reading

You should read the following topics before starting this exercise:

1. The JDBC driver types
2. The design of JDBC
3. Statement, PreparedStatement, ResultSet
4. DatabaseMetaData and ResultSetMetaData

Ready Reference

JDBC : Java Database Connectivity

This API contains of a set of **classes** and **interfaces** to enable programmers to communicate with a database using java. These classes and interfaces are in the java.sql package.

The JDBC API makes it possible to do three things:

i. Establish a connection with a data source.
ii. Send queries and update statements to the data source.
iii. Process the results.

The classes and interfaces in the **java.sql** package are given below.

Interface Name	Description
`Array`	Maps to the SQL type ARRAY
`Blob`	Represents SQL BLOB Value
`CallableStatement`	To execute SQL stored procedures.
`Clob`	Represents SQL CLOB type
`Connection`	Represents a connection session with the database
`DatabaseMetaData`	Information about the database
`Driver`	Interface that every driver class must implement
`ParameterMetaData`	Information about parameters in PreparedStatement object
`PreparedStatement`	Represents precompiled SQL statement
`Ref`	Maps to SQL `REF` type
`ResultSet`	Table of data generated by executing a database query
`ResultSetMetaData`	Information about columns in a ResultSet
`Savepoint`	The representation of a savepoint, which is a point within the current transaction.
`SQLData`	For custom mapping of an SQL user-defined type (UDT) to a class in the Java programming language.
`SQLInput`	An input stream that contains a stream of values representing an instance of an SQL structured type.
`SQLOutput`	The output stream for writing the attributes of a user-defined type back to the database.
`Statement`	For executing a static SQL statement and returning the results it produces.

`Struct`	Maps to an SQL structured type.

Class Name	Description
`Date`	Represents an SQL `DATE` value.
`DriverManager`	The basic service for managing a set of JDBC drivers.
`DriverPropertyInfo`	Driver properties for making a connection.
`SQLPermission`	The permission for which the `SecurityManager` will check when code that is running in an applet calls the `DriverManager.setLogWriter` method or the `DriverManager.setLogStream` (deprecated) method.
`Time`	Represents an SQL `TIME` value.
`Timestamp`	Represents an SQL `TIMESTAMP` value.
`Types`	Defines constants that are used to identify generic SQL types, called JDBC types.

JDBC Drivers

To communicate with a database, you need a database driver. There are four types of drivers:

1. Type 1: JDBC-ODBC Bridge driver
2. Type 2: Native-API partly-Java driver:
3. Type 3: JDBC-Net pure Java driver:
4. Type 4: Native-protocol pure Java driver:

For postgresql, use the driver:
`org.postgresql.Driver`

To load the driver, use the following command:

```
Class.forName("driverName");
```

Example:
```
Class.forName("org.postgresql.Driver");
```

Establishing a connection

To establish a connection with the database, use the getConnection method of the DriverManager class. This method returns a Connection object.

```
DriverManager.getConnection("url", "user", "password");
```

Example:
```
Connection conn = DriverManager.getConnection
("jdbc:postgresql://192.168.100.4/TestDB", "scott", "tiger");
```

Methods of Connection class:

`void` **`close`**`()`	Releases this `Connection` object's database and JDBC resources immediately instead of waiting for them to be automatically released.
`void` **`commit`**`()`	Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object.
`Statement` **`createStatement`**`()`	Creates a `Statement` object for sending SQL statements to the database.
`Statement` **`createStatement`**`(int`	Creates a `Statement` object that will generate `ResultSet` objects

`resultSetType, int resultSetConcurrency)`	with the given type and concurrency.
`Boolean` **`getAutoCommit`**`()`	Retrieves the current auto-commit mode for this `Connection` object.
`DatabaseMetaData` **`getMetaData`**`()`	Retrieves a `DatabaseMetaData` object that contains metadata about the database to which this `Connection` object represents a connection.
`CallableStatement` **`prepareCall`**`(String sql)`	Creates a `CallableStatement` object for calling database stored procedures.
`CallableStatement` **`prepareCall`**`(String sql, int resultSetType, int resultSetConcurrency)`	Creates a `CallableStatement` object that will generate `ResultSet` objects with the given type and concurrency.
`PreparedStatement` **`prepareStatement`**`(String sql)`	Creates a `PreparedStatement` object for sending parameterized SQL statements to the database.
`PreparedStatement` **`prepareStatement`**`(String sql, int resultSetType, int resultSetConcurrency)`	Creates a `PreparedStatement` object that will generate `ResultSet` objects with the given type and concurrency.
`void` **`rollback`**`()`	Undoes all changes made in the current transaction and releases any database locks currently held by this `Connection` object.
`void` **`setAutoCommit`**`(boolean autoCommit)`	Sets this connection's auto-commit mode to the given state.

Executing Queries

To execute an SQL query, you have to use one of the following classes:

- Statement
- PreparedStatement
- CallableStatement

A Statement represents a general SQL statement without parameters. The method **createStatement()** creates a Statement object. A PreparedStatement represents a precompiled SQL statement, with or without parameters. The method **prepareStatement(String sql)** creates a PreparedStatement object. CallableStatement objects are used to execute SQL stored procedures. The method **prepareCall(String sql)** creates a CallableStatement object.

Executing a SQL statement with the Statement object, and returning a jdbc resultSet.

To execute a query, call an `execute` method from `Statement` such as the following:

- `execute`: Use this method if the query could return one or more `ResultSet` objects.
- `executeQuery`: Returns one `ResultSet` object.

- `executeUpdate`: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using `INSERT`, `DELETE`, or `UPDATE` SQL statements.

Examples:
```
ResultSet rs = stmt.executeQuery("SELECT * FROM Student");
int result = stmt.executeUpdate("Update authors SET name = 'abc' WHERE id = 1");
boolean ans = stmt.execute("DROP TABLE IF EXISTS test");
```

ResultSet provides access to a table of data generated by executing a Statement. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The **next()** method is used to successively step through the rows of the tabular results.

Examples:
```
Statement stmt =  conn.prepareStatement();
ResultSet rs = stmt.executeQuery("Select * from student");
while(rs.next())
{
  //access resultset data
}
```
To access these values, there are getXXX() methods where XXX is a type *for example*, getString(), getInt() etc. There are two forms of the getXXX methods:

i. Using columnName: getXXX(String columnName)

ii. Using columnNumber: getXXX(int columnNumber)

Example

```
rs.getString("stuname"));
rs.getString(1); //where name appears as column 1 in the ResultSet
```

Using PreparedStatement

These are precompiled sql statements. For parameters, the SQL commands in a PreparedStatement can contain **placeholders** which are represented by '**?**' in the SQL command.

Example
```
String sql = "UPDATE authors SET name = ? WHERE id = ?";
PreparedStatement  ps = conn.prepareStatement(sql);
```

Before the sql statement is executed, the placeholders have to be replaced by actual values. This is done by calling a **setXXX(int n, XXX x)** method, where XXX is the appropriate type for the parameter *for example*, setString, setInt, setFloat, setDate etc, n is the placeholder number and x is the value which replaces the placeholder.

Example
```
String sql = "UPDATE authors SET name = ? WHERE id = ?";
PreparedStatement  ps = conn.prepareStatement(sql);
ps.setString(1,'abc'); //assign abc to first placeholder
ps.setInt(2,123); //assign 123 to second placeholder
```

ResultSet Scroll Types and Concurrency

The scroll type indicates how the cursor moves in the ResultSet. The concurrency type affects concurrent access to the resultset. The types are given in the table below.

Scroll Type

`TYPE_FORWARD_ONLY`	The result set is not scrollable.
`TYPE_SCROLL_INSENSITIVE`	The result set is scrollable but not sensitive to database changes.
`TYPE_SCROLL_SENSITIVE`	The result set is scrollable and sensitive to database changes.
Concurrency Type	
`CONCUR_READ_ONLY`	The result set cannot be used to update the database.
`CONCUR_UPDATABLE`	The result set can be used to update the database.

Example:

```
Statement stmt = conn.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

ResultSet Interface

The `ResultSet` interface provides methods for retrieving and manipulating the results of executed queries.

Method Name	Description
`beforeFirst()`	Default position. Puts cursor before 1st row of ResultSet.
`first()`	Puts cursor on 1st row of ResultSet.
`last()`	Puts cursor on last row of ResultSet.
`afterLast()`	Puts cursor after/beyond last row of ResultSet.
`absolute (int pos)`	Puts cursor at row number position where absolute (1) is a 1st row and absolute (-1) is last row of ResultSet.
`relative (int pos)`	Puts cursor at row no. position relative from current position.
`next()`	To move to the next row in ResultSet
`previous()`	To move to the previous row in ResultSet.
`void close()`	To close the ResultSet.
`deleteRow()`	Deletes the current row from the ResultSet and underlying database.
`getRow()`	Retrieves the current row number
`insertRow()`	Inserts the contents of the insert row into the ResultSet object and into the database.
`refreshRow()`	Refreshes the current row with its most recent value in the database.
`updateRow()`	Updates the underlying database with the new contents of the current row of this ResultSet object.
`getXXX(String columnName)`	Retrieves the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
`moveToInsertRow()`	Moves the cursor to the insert row.
`close()`	Disposes the ResultSet.
`isFirst()`	Tests whether the cursor is at the first position.
`isLast()`	Tests whether the cursor is at the last position
`isBeforeFirst()`	Tests whether the cursor is before the first position
`isAfterLast()`	Tests whether the cursor is after the last position
`updateXXX(int columnNumber, XXX value)`	Updates the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.
`updateXXX(String columnName, XXX value)`	Updates the value of the designated column in the current row as a corresponding type in the Java programming language. XXX represents a type: Int, String, Float, Short, Long, Time etc.

DatabaseMetaData

This interface provides methods that tell you about the database for a given connection object.

Method Name	Description
`getDatabaseProductName()`	Retrieves the name of this database product.
`getDatabaseProductVersion()`	Retrieves the version number of this database product.
`getDriverName()`	Retrieves the name of this JDBC driver.
`getDriverVersion()`	Retrieves the version number of this JDBC driver as a String.
`getUserName()`	Retrieves the user name as known to this database.
`getCatalogs()`	Retrieves the catalog names available in this database.
`getSchemas(String catalog, String schemaPattern)`	Retrieves the schema names available in this database.
`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`	Retrieves a description of the tables available in the given catalog.
`getPrimaryKeys(String catalog, String schema, String table)`	Retrieves a description of the given table's primary key columns.
`getExportedKeys(String catalog, String schema, String table)`	Retrieves a description of the foreign key columns that reference the given table's primary key columns (the foreign keys exported by a table).
`getImportedKeys(String catalog, String schema, String table)`	Retrieves a description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).
`getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)`	Retrieves a description of table columns available in the specified catalog.
`getProcedures(String catalog, String schemaPattern, String procedureNamePattern)`	Retrieves a description of the stored procedures available in the given catalog.
`getFunctions(String catalog, String schemaPattern, String functionNamePattern)`	Retrieves a description of the system and user functions available in the given catalog.

Example:
```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getTables(null, null, null,new String[] {"TABLE"});
while (rs.next())
         System.out.println( rs.getString("TABLE_NAME"));
```

ResultSetMetaData

The ResultSetMetaData interface provides information about the structure of a particular ResultSet.

Method Name	Description
`getColumnCount()`	Returns the number of columns in the current ResultSet object.
`getColumnDisplaySize(int column)`	Gives the maximum width of the column specified by the index parameter.
`getColumnLabel(int column)`	Gives the suggested title for the column for use in display and printouts.
`getColumnName(int column)`	Gives the column name associated with the column index.
`getColumnTypeName(int column)`	Gives the designated column's SQL type.
`isReadOnly(int column)`	Indicates whether the designated column is read-only.
`isWritable(int column)`	Indicates whether you can write to the designated column.

`isNullable(int column)`	Indicates the nullability of values in the designated column.

```
Example:
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
int noOfColumns = rsmd.getColumnCount();
System.out.println("Number of columns = " + noOfColumns);
for(int i=1; i<=noOfColumns; i++)
{
   System.out.println("Column No : " + i);
   System.out.println("Column Name : " + rsmd.getColumnName(i));
   System.out.println("Column Type : " + rsmd.getColumnTypeName(i));
  System.out.println("Column display size : " + rsmd.getColumnDisplaySize(i));
}
```

Self Activity

1. Sample program to display employee data (id, name, salary)

```
import java.sql.*;
import java.io.*;
class JDBCDemo
{
  public static void main(String[] args) throws SQLException
  {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    try
    {
      Class.forName("org.postgresql.Driver");
      conn =
DriverManager.getConnection("jdbc:postgresql://192.168.100.254/employeeDB","student","");
      if(conn==null)
        System.out.println("Connection failed ");
                    else
             {
             System.out.println("Connection successful..");
             stmt = conn.createStatement();
          rs = stmt.executeQuery("Select * from emp");
          while(rs.next())
          {
          System.out.print("ID = " + rs.getInt(1));
          System.out.println("Name = " + rs.getString(2));
          System.out.println("Salary = " + rs.getInt(3));
          }
          conn.close();
     }
   }
    catch(Exception e)
    { System.out.println(e);}
  }
}// end of class
```

2. Sample program to perform insert and delete operations on employee table using PreparedStatement (id, name, salary)

```
import java.sql.*;
import java.io.*;
class JDBCDemoOp
{
  public static void main(String[] args) throws SQLException
  {
Connection conn = null;
Statement stmt =  null;
ResultSet rs = null;
PreparedStatement ps1 = null, ps2=null;
int id, sal;
String name;
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
Class.forName("org.postgresql.Driver");
conn =
DriverManager.getConnection("jdbc:postgresql://192.168.100.254/employeeDB","student","");
stmt = conn.createStatement();
ps1 = conn.prepareStatement("Insert into employee values(?,?,?)");
ps2 = conn.prepareStatement("Delete employee where ID = ?");
if(conn!=null)
  System.out.println("Connection successful..");
     System.out.println("Enter the ID, name and salary to be inserted ");
     id = Integer.parseInt(br.readLine());
     name = br.readLine();
     sal = Integer.parseInt(br.readLine());
     ps1.setInt(1,id); ps1.setString(2,name);ps1.setInt(3,sal);
     ps1.executeUpdate();

     System.out.println("Enter the ID to be deleted ");
     id = Integer.parseInt(br.readLine());
     ps2.setInt(1,id);
     ps2.executeUpdate();
     conn.close();
  }
}// end of class
```

Lab Assignments

SET A

1. Create a student table with fields roll number, name, percentage. Insert values in the table. Display all the details of the student table in a tabular format on the screen (using swing).

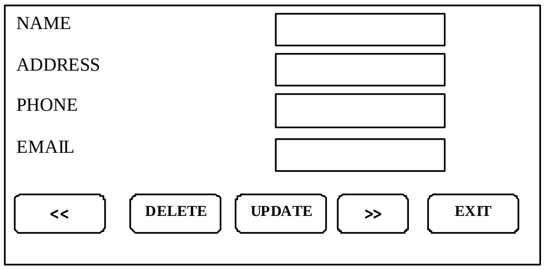
2. Write a program to display information about the database and list all the tables in the database. (Use DatabaseMetaData).

3. Write a program to display information about all columns in the student table. (Use ResultSetMetaData).

SET B

1. Write a menu driven program (Command line interface) to perform the following operations on student table.
   1. Insert 2. Modify 3. Delete 4. Search 5. View All 6. Exit

2. Design a following Phone Book Application Screen using swing & write a code for various operations like Delete, Update, Next, Previous. Raise an appropriate exception if invalid data is entered like name left blank and negative phone Number.



SET C

1. Create tables : Course (id, name, instructor) and Student (id, name). Course and Student have a many to many relationship. Create a GUI based system for performing the following operations on the tables:
   Course: Add Course, View All students of a specific course
   Student: Add Student, Delete Student, View All students, Search student

2. Design a GUI to perform the following operations on Telephone user data.
i. Add record   ii. Display current bill
Add record stores the details of a telephone user in a database. User has the following attributes: User (id, name, telephone number, number of calls, month, year).
Display current bill should Calculate and display the bill for a specific user (search by name or phone number) using the following rules. Provide button to Search user on basis of telephone number or name.
Rules: The first 100 calls are free and rent is Rs. 300)

No. Of Calls	Charge (per call)
> 100 and <=500	Rs. 1.00
> 500	Rs. 1.30

Signature of the instructor [ ]   Date [/ /]

Assignment Evaluation   **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 3: Servlets

Objectives

- **To understand server-side programming**
- **Defining and executing servlets**

Reading

You should read the following topics before starting this exercise:

1. Concept of servlet
2. Difference between applet and servlet
3. Introduction to Servlet (HTTP Servlet)
4. Lifecycle of a Servlet
5. Handling Get and Post requests (HTTP)
6. Data Handling using Servlet
7. Creating Cookies
8. Session Tracking using HTTP Servlet

Ready Reference

What are servlets?

Servlets are small programs that execute on the server side. Servlets are pieces of Java source code that add functionality to a web server

Servlet provides full support for sessions, a way to keep track of a particular user over time as a website's pages are being viewed. They also can communicate directly with a web server using a standard interface.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

Running servlets requires a server that supports the technologies. Several web servers, each of which has its own installation, security and administration procedures, support Servlets. The most popular one is the Tomcat- an open source server developed by the Apache Software Foundation in cooperation with Sun Microsystems version 5.5 of Tomcat supports Java Servlet.

Getting Tomcat

The software is available a a free download from Apache's website at the address http://jakarta.apache.org/tomcat. Several versions are available: Linux users should download the **rpm** of Tomcat.

The javax.servlet package

The important interfaces and classes are described in the table below.

Interface	Description
`Servlet`	A java servlet must implement the Servlet interface. This interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods.

`ServletConfig`	The ServletConfig interface is used by the server to pass configuration information to a servlet. Its methods are used by the servlet to retrieve this information.
`ServletRequest`	The ServletRequest interface encapsulates a client request for service. It defines a number of methods for obtaining information about the server, requester, and request.
`ServletResponse`	The ServletResponse interface is used by a servlet to respond to a request by sending information back to the client.
`ServletContext`	The ServletContext interface defines the environment in which an applet is executed. It provides methods that are used by applets to access environment information.
`SingleThreadModel`	The SingleThreadModel interface is used to identify servlets that must be thread-safe. If a servlet implements this interface, the Web server will not concurrently execute the service() method of more than one instance of the servlet.
Class	**Description**
`GenericServlet`	The GenericServlet class implements the Servlet interface. You can subclass this class to define your own servlets.
`ServletInputStream`	The ServletInputStream class is used to access request information supplied by a Web client. An object of this class is returned by the getInputStream() method of the ServletRequest interface.
`ServletOutputStream`	The ServletOutputStream class is used to send response information to a Web client. An object of this class is returned by the getOutputStream() method of the ServletResponse interface.

The javax.servlet.http package

Interface	Description
`HttpServletRequest`	The HttpServletRequest interface extends the ServletRequest interface and adds methods for accessing the details of an HTTP request.
`HttpServletResponse`	The HttpServletResponse interface extends the ServletResponse interface and adds constants and methods for returning HTTP-specific responses.
`HttpSession`	This interface is implemented by servlets to enable them to support browser-server sessions that span multiple HTTP request-response pairs. Since HTTP is a stateless protocol, session state is maintained externally using client-side cookies or URL rewriting. This interface provides methods for reading and writing state values and managing sessions.
`HttpSessionContext`	This interface is used to represent a collection of HttpSession objects that are associated with session IDs.
Class	**Description**
`HttpServlet`	Used to create HTTP servlets. The HttpServlet class extends the GenericServlet class.
`Cookie`	This class represents an HTTP cookie. Cookies are used to maintain session state over multiple HTTP requests. They are named data values that are created on the Web server and stored on individual browser clients. The Cookie class provides the method for getting and setting cookie values and attributes.

Servlet Life Cycle

A servlet's life cycle methods function similarly to the life cycle methods of applets.

- The **init(ServletConfig)** method is called automatically when a web server first begins a servlet to handle the user's request. The init() method is called only once. ServletConfig is an interface in the javax.servlet package, containing the methods to find out more about the environment in which a servlet is running.
- The servlet action is in the **service()** method. The service() method checks the HTTP request type (GET, POST, PUT, DELETE etc.) and calls doGet(), doPost(),

doPut(), doDelete() etc. methods. A GET request results from normal request for a URL or from an HTML form that has no METHOD specified. The POST request results from an HTML form that specifically lists POST as the METHOD.

- The **destroy()** method is called when a web server takes a servlet offline.

Using Servlets

One of the main tasks of a servlet is to collect information from a web user and present something back in response. Collection of information is achieved using form, which is a group of text boxes, radio buttons, text areas, and other input fields on the web page. Each field on a form stores information that can be transmitted to a web server and then sent to a Java servlet. web browsers communicate with servers by using Hypertext Transfer Protocol (HTTP).

- Form data can be sent to a server using two kinds of HTTP requests: get and post. When web page calls a server using **get** or **post**, the name of the program that handles the request must be specified as a web address, also called uniform resource locator (URL). A get request affixes all data on a form to the end of a URL. A post request includes form data as a header and sent separately from the URL. This is generally preferred, and it's required when confidential information is being collected on the form.
- Java servlets handle both of these requests through methods inherited from the HTTPServlet class: **doGet(HttpServletRequest, HttpServletResponse)** and **doPost(HttpServletRequest, HttpServletResponse).** These methods throw two kinds of exceptions: ServletException, part of javax.servlet package, and IOException, an exception in the java.io package.
- The **getparameter(String)** method is used to retrieve the fields in a servlet with the name of the field as an argument. Using an HTML document a servlet communicates with the user.
- While preparing the response you have to define the kind of content the servlet is sending to a browser. The **setContentType(String)** method is used to decide the type of response servlet is communicating. Most common form of response is written using an HTML as: **setContentType("text/html").**
- To send data to the browser, you create a servlet output stream associated with the browser and then call the **println(String)** method on that stream. The **getWriter()** method of HttpServletResponse object returns a stream. which can be used to send a response back to the client.

Example

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http.*;
public class MyHttpServlet extends HttpServlet
{
  public void doGet(HttpServletRequest req,HttpServletResponse res) throws
ServletException, IOException
{
    // Use "req" to read incoming request
    // Use "res" to specify the HTTP response status
    //Use req.getParameter(String) or getParameterValues(String) to obtain
parameters
         PrintWriter out = res.getWriter();//stream for output
    // Use "out" to send content to browser
  }
}
```

Request and Response methods

`ServletRequest` **methods**	
`String getParameter(String name )`	Obtains the value of a parameter sent to the servlet as part of a `get` or `post` request. The `name` argument represents the parameter name.
`Enumeration getParameterNames()`	Returns the names of all the parameters sent to the servlet as part of a `post` request.
`String[]getParameterValues(String name)`	For a parameter with multiple values, this method Returns an array of strings containing the values for a specified servlet parameter.
`String getProtocol()`	Returns the name and version of the protocol the request uses in the form *protocol/majorVersion.minorVersion*, for example, HTTP/1.
`String getRemoteAddr()`	Returns the Internet Protocol (IP) address of the client that sent the request.
`String getRemoteHost()`	Returns the fully qualified name of the client that sent the request.
`String getServerName()`	Returns the host name of the server that received the request.
`int getServerPort()`	Returns the port number on which this request was received.
`HttpServletRequest` **methods**	
`Cookie[] getCookies()`	Returns an array of Cookie objects stored on the client by the server.
`HttpSession getSession( boolean create )`	Returns an `HttpSession` object associated with the client's current browsing session. This method can create an `HttpSession` object (`True argument`) if one does not already exist for the client.
`String getServletPath()`	Returns the part of this request's URL that calls the servlet.
`String getMethod()`	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
`String getQueryString()`	Returns the query string that is contained in the request URL after the path.
`String getPathInfo()`	Returns any extra path information associated with the URL the client sent when it made this request.
String getRemoteUser()	Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
`ServletResponse` **methods**	
`ServletOutputStream getOutputStream()`	Obtains a byte-based output stream for sending binary data to the client.
`PrintWriter getWriter()`	Obtains a character-based output stream for sending text data (usually HTML formatted text) to the client.
`void setContentType(String type)`	Specifies the content type of the response to the browser. The content type is also known as MIME (Multipurpose Internet Mail Extension) type of the data. For examples, `"text/html"` , `"image/gif"` etc.
`String setContentLength(int len)`	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.

`HttpServletResponse` methods	
`void addCookie(Cookie cookie)`	Used to add a `Cookie` to the header of the response to the client.
`void sendError(int ec)`	Sends an error response to the client using the specified status.
`void sendError(int ec, String messg)`	Sends an error response to the client using the specified status code and descriptive message.
`void sendRedirect(Stirng url)`	Sends a temporary redirect response to the client using the specified redirect location URL.
`void setHeader(String name, String value)`	Sets a response header with the given name and value.

Writing, Compiling and Running Servlet
Type the first sample program of the self-activity section. After saving this servlet,compile it with the Java compiler as: javac SimpleServlet.java. After compilation a class file with name SimpleServlet.class is created.

To make the servlet available, you have to publish this class file in a folder on your web server that has been designated for Java servlets. Tomcat provides the classes sub-folder to deploy this servlet's class file. Copy this class file in this classes sub-folder, which is available on the path: tomcat/webapps/ WEB-INF/classes. Now edit the web.xml file available under WEB-INF sub-folder with the following lines:

```
<servlet>
	<servlet-name>SimpleServlet</servlet-name>
	<servlet-class>SimpleServlet</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>SimpleServlet</servlet-name>
	<url-pattern>/SimpleServlet</url-pattern>
</servlet-mapping>
```

Repeat the above sequence of line to run every newly created servlet. Remember, these line lines must be placed somewhere after the <web-app> tag and before the closing </web-app> tag.

After adding these lines, save web.xml file. Restart the Tomcat service and run the servlet by loading its address with a web browser as: http://localhost:8080/FirstServlet.

Using MySQL – Database Connectivity tool with servlets
Java's Servlet also provides support for data handling using MySQL database. For this you have to do few simple steps.

1. Copy the jar file mentioned in Database Connectivity assignment into the subfolder: tomcat/lib/common.
2. Edit the file .bash_profile of your login using command: vi .bash_profile.
3. Add the following line without removing any line.
   `export CLASSPATH=$CLASSPATH:/$HOME/tomcat/common/lib/<jar file> used in database connectivity assignment.`
   Example: if I have mysql-connector-java-5.1.6.jar file, I will type the line as
   `export CLASSPATH=$CLASSPATH:/$HOME/tomcat/common/lib/mysql-connector-java-5.1.6.jar`
4. Save this file. Logout from the terminal and re-login.

5. Create the table student(rno, sname) in your database. Insert few records into this table.

Session Handling

1. Using cookies
2. Using HttpSession class

1. Using Cookies

To keep the track of information about you and the features you want the site to display. This customization is possible because of a web browser features called cookies, small files containing information that a website wants to remember about a user, like username, number of visits, and other. The files are stored on the user's computer, and a website can read only the cookies on the user's system that the site has created. The default behavior of all the web browsers is to accept all cookies.

The **javax.servlet.http.Cookie** class allows us to create a cookie and send it to the browser. The methods are:

Method	Description
`int getMaxAge()`	Returns the maximum age of the cookie, specified in seconds, By default, `-1` indicating the cookie will persist until browser shutdown.
`String getName()`	Returns the name of the cookie.
`String getValue()`	Returns the value of the cookie.
`void setMaxAge(int s)`	Sets the maximum age of the cookie in seconds.
`void setValue (String value)`	Assigns a new value to a cookie after the cookie is created.

- The Cookie class in the javax.servlet.http package supports cookies. To create a cookie, call the Cookie(String,String) constructor. The first argument is the name you want to give the Cookie, and the second is the cookie's value.
- To send a cookie, call the addCookie(Cookie) method of an HttpServletResponse object. You can add more than one cookie to a response.
- In a servlet,call the getCookies() method of an HttpServletRequest object to receive an array of Cookie objects. Use getName() and getValue() methods to find out about cookie.

2.HttpSession class

Servlet can retain the state of user through **HttpSession**, a class that represents sessions. There can be one session object for each user running your servlet.

- A user's session can be created or retrieved by calling the **getSession(Boolean)** method of the servlet's request object. Use an argument true if a session should be created when one doesn't already exist for the user.

Example: HttpSession state=req.getSession(true);

```
public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
{
      HttpSession session = req.getSession(true);
// ...
}
```

- Objects held by session are called its attributes. Call the session's **setAttribute(String, Object)** method with two arguments: a name to give the attribute and the object.
- To retrieve an attribute, call the **getAttribute(String)** method with its name as the only argument. It returns the object, which must be cast from object to the desired class, or null if no attribute of that name exists.
- To remove an attribute when it's no longer needed, call **removeAttribute(String)** with its name as the argument.

Method	Description
`Object getAttribute(String name)`	Returns the object bound with the specified name in this session, or `null` if no object is bound under the name.
`Enumeration getAttributeNames()`	Returns an `Enumeration` of `String` objects containing the names of all the objects bound to this session.
`long getCreationTime()`	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
`long getLastAccessedTime()`	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
`int getMaxInactiveInterval()`	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
`void RemoveAttribute(String name)`	Removes the object bound with the specified name from this session.
`void setAttribute(String name, Object value)`	Binds an object to this session, using the name specified.
`void setMaxInactiveInterval(int seconds)`	Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.
`void invalidate()`	Invalidates this session then unbinds any objects bound to it.
`Boolean isNew()`	Returns true if it is a new session.
`String getId()`	Returns a string containing the unique identifier assigned to this session.

Self Activity

1. Sample program

/* Program for simple servlet*/

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet
{
        public void service(HttpServletRequest req, HttpServletResponse rs)
        throws ServletException, IOException
  {
                rs.setContentType("text/html");
                PrintWriter pw=rs.getWriter();
                pw.println("<html>");
                pw.println("<body>");
                pw.println("<B>Hello, Welcome to Java Servlet's");
                pw.println("</body>");
                pw.println("</html>");
                pw.close();
  }
}
```

After saving this servlet, compile it with the Java compiler as: javac SimpleServlet.java.
Run the servlet using http://server-ip:8080/SimpleServlet

2. Sample program to read two numbers and return their sum

```
// Save the following code as Sum.html
<html>
<head>
<title></title>
</head>
<body>
<form method="post" action="http://server-ip:8080/AddServlet">
Enter the Number1 <input type="text" name="No1">
Enter the Number2 <input type="text" name="No2">
<br>
<input type="Submit">
</form>
</body>
</html>

// Save the following code as AddServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddServlet extends HttpServlet{
      public void doGet(HttpServletRequest req,HttpServletResponse
res)
            throws ServletException, IOException{

            int no1=Integer.parseInt(req.getParameter("No1"));
            int no2=Integer.parseInt(req.getParameter("No2"));
            int total=no1+no2;

            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            pw.println("<h1> ans </h1> <h3>"+total+"</h3>");
            pw.close();
      }
}
```

3. Sample program for database handling using servlet

```
//Create a student table (rno, name)
//The servlet displays all records from the student table on the client machine.
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class ServJdbc extends HttpServlet{

      String qry;
      ResultSet rs;
      Statement st;
      Connection cn;
      String ename,sal;

      public void init()
      {
            try{
                  Class.forName("org.gjt.mm.mysql.Driver");

      cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/root","root","");
            }
            catch(ClassNotFoundException ce){}
            catch(SQLException se){}

      }

      public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
      {
            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            try{

                  qry="Select * from student";
                  st=cn.createStatement();
                  rs=st.executeQuery(qry);

                  while(rs.next())
                  {
                        pw.print("<table border=1>");
                        pw.print("<tr>");
                        pw.print("<td>" + rs.getInt(1) + "</td>");
                        pw.print("<td>" + rs.getString(2) + "</td>");
                        pw.print("</tr>");
                        pw.print("</table>");
                  }
            }
            catch(Exception se){}
            pw.close();
      }
}
```

Run this program as http://server-ip:8080/ServJdbc

4. Sample program for cookies

```
/*Save this program as AddCookie.java
*/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddCookie extends HttpServlet{

      public void doGet(HttpServletRequest req,HttpServletResponse
res) throws ServletException, IOException{

            Cookie c1=new Cookie("Cookie1","1");
            res.addCookie(c1);
            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            pw.print("Cookie added with value 1);
            Cookie c2=new Cookie("Cookie2","2");
            res.addCookie(c2);
            pw.print("Cookie added with value 2);
            pw.close();
      }
}
```

/* Save this program as GetCookie.java */

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetCookie extends HttpServlet{

      public void doGet(HttpServletRequest req,HttpServletResponse
res) throws ServletException, IOException{

            Cookie [] c=req.getCookies();
            res.setContentType("text/html");
            PrintWriter pw=res.getWriter();
            for(int i=0;i<c.length;i++)
                  pw.println("Cookie Name"+c[i].getName());
            pw.close();
      }
}
```

Run this program as http://server-ip:8080/AddCookie
Run this program as http://server-ip:8080/GetCookie

5. Sample program for sessions

/* Program for Session using Servlet
Save this program as SessionDemo.java
*/

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionDemo extends HttpServlet{

      String result1="success";
      String result2="failure";
      public void doGet(HttpServletRequest req,HttpServletResponse
res)
      throws ServletException, IOException{


      HttpSession hs=req.getSession(true);

      String lname=req.getParameter("txt1");
      String pwd=req.getParameter("txt2");

      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();

      if((lname.equals("BCS"))&&(pwd.equals("CS")))
      {
            pw.print("<a href=http://localhost:8080/NewInfo.html>
Login Success</a>");
            hs.setAttribute("loginID",result1);
      }
      else
      {
            pw.print("<a href=http://localhost:8080/NewInfo.html>
Kick Out</a>");
            hs.setAttribute("loginID",result2);
      }
      pw.close();
      }
}
<!—HTML File for NewInfo.html -->
<html>
<head>
<title></title>
</head>
<body>
<form method="post" action="http://localhost:8080/SessionInfo">
<input type="Submit" value=”Read Session Value”>
</form>
</body>
</html>

/*Save this program as SessionInfo.java */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionInfo extends HttpServlet{
      String readloginid;
      public void doGet(HttpServletRequest req,HttpServletResponse
res)  throws ServletException, IOException{

      HttpSession hs=req.getSession(true);
      readloginid=hs.getId();

      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();

      if(hs.getAttribute("loginID").equals("success"))
            pw.print("Your Session ID " + readloginid);
      else
            pw.print("<h1>Session Expired </h1>");
      pw.close();
      }
}
```

Create an html file for login and password and use http://server-ip:8080/SessionDemo in the Form Action tag.

Lab Assignments

SET A

1. Design a servlet that provides information about a HTTP request from a client, such as IP address and browser type. The servlet also provides information about the server on which the servlet is running, such as the operating system type, and the names of currently loaded **servlets.**

2. Write a servlet which counts how many times a user has visited a web page. If the user is visiting the page for the first time, display a welcome message. If the user is re-visiting the page, display the number of times visited. (Use cookies)

3. Design an HTML page which passes student roll number to a search servlet. The servlet searches for the roll number in a database (student table) and returns student details if found or error message otherwise.

SET B

1. Write a program to create a shopping mall. User must be allowed to do purchase from two pages. Each page should have a page total. The third page should display a bill, which consists of a page total of what ever the purchase has been done and print the total. (Use HttpSession)

2. Design an HTML page containing 4 option buttons (Painting, Drawing, Singing and swimming) and 2 buttons reset and submit. When the user clicks submit, the server responds by adding a cookie containing the selected hobby and sends a message back to the client. Program should not allow duplicate cookies to be written.

Set B: Additional Programs For Practice

1. Design the table Login(login_name, password) using MySQL. Also design an HTML login screen. Accept the login name and password from the user. The servlet accepts the login name and password and validates it from the database you have created. The servlet sends back an appropriate response. Also calculate the number of time that user has successfully performed the login activity. Use cookies.

SET C

1. Consider the following entities and their relationships
   Movie (movie_no, movie_name, release_year)
   Actor(actor_no, name)
   Relationship between movie and actor is many – many with attribute rate in Rs. Create a RDB in 3 NF answer the following:
   a) Accept an actor name and display all movie names in which he has acted along with his name on top.
   b) Accept a movie name and list all actors in that movie along with the movie name on top.

Signature of the instructor		Date	/ /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 4: Java Server Pages

Objectives

- **To demonstrate the use of JSP**

Reading

You should read the following topics before starting this exercise

1. Concept of Servlets.
2. JSP life-cycle.
3. JSP Directives
4. Scripting elements.
5. Actions in JSP.

Ready Reference

What is JSP?

JSP is Java Server Page, which is a dynamic web page and used to build dynamic websites. To run jsp, we need web server which can be tomcat provided by apache, it can also be jRun, jBoss(Redhat), weblogic (BEA) , or websphere(IBM).

JSP is dynamic file whereas Html file is static. HTML can not get data from database or dynamic data. JSP can be interactive and communicate with database and controllable by programmer. It is saved by extension of **.jsp**. Each Java server page is compiled into a servlet before it can be used. This is normally done when the first request to the JSP page is made.

A JSP contains 3 important types of elements:-

1. **Directives:-** these are messages to the JSP container that is the server program that executes JSPs.
2. **Scripting elements:-** These enables programmers to insert java code which will be a part of the resultant servlet.
3. **Actions:-** Actions encapsulates functionally in predefined tags that programmers can embedded in a JSP.

JSP Directives:-

Directives are message to the JSP container that enable the programmer to specify page setting to include content from other resources & to specify custom tag libraries for use in a JSP.

Syntax:-

<%@ name attribute1="….", attribute2="…"…%>

Directive	Description
`page`	Defines page settings for the JSP container to process.
`include`	Causes the JSP container to perform a translation-time insertion of another resource's content. The file included can be either static (HTML file) or dynamic (i.e., another tag file)
`taglib`	Allows programmers to use new tags from tag libraries that encapsulate more complex functionality and simplify the coding of a JSP.

Page Directive:-
The page directives specify global settings for the JSP in the JSP container. There can be many page directives, provided that there is only one occurrence of each attribute.

Syntax:-
```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import="{package.class | package.*}, ..." ]
  [ session="true|false" ]
  [ buffer="none|8kb|sizekb" ]
  [ autoFlush="true|false" ]
  [ isThreadSafe="true|false" ]
  [ info="text" ]
  [ errorPage="relativeURL" ]
  [ contentType="mimeType [ ; charset=characterSet ]" |
    "text/html ; charset=ISO-8859-1" ]
  [ isErrorPage="true|false" ]
  [ pageEncoding="characterSet | ISO-8859-1" ]    %>
```

Scripting Elements

1. Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file.
Syntax
```
<%! Java declaration statements %>
```
Example,
```
<%! private int count = 0; %>
<%! int i = 0; %>
```
2. Expressions
An expression element contains a java expression that is evaluated, converted to a `String`, and inserted where the expression appears in the JSP file.
Syntax
```
<%= expression %>
```
Example,
```
Your name is <%= request.getParameter("name") %>
```
3. Scriptlet
A scriptlet contains a set of java statements which is executed. A scriptlet can have java variable and method declarations, expressions, use implicit objects and contain any other statement valid in java.
Syntax
```
<% statements %>
```
Example
```
<%
   String name = request.getParameter("userName");
       out.println("Hello  " + name);
%>
```

Implicit objects used in JSP

Implicit object	Description
`application`	A javax.servlet.ServletContext object that represents the container in which the JSP executes. It allows sharing information between the jsp page's servlet and any web components with in

	the same application.
config	A javax.servlet.ServletConfig object that represents the JSP configuration options. As with servlets, configuration options can be specified in a Web application descriptor (web.xml). The method getinitparameter() is used to access the initialization parameters.
exception	A java.lang.Throwable object that represents an exception that is passed to a JSP error page. This object is available only in a JSP error page.
out	A javax.servlet.jsp.JspWriter object that writes text as part of the response to a request. This object is used implicitly with JSP expressions and actions that insert string content in a response.
page	An `Object` that represents the current JSP instance.
pageContext	A javax.servlet.jsp.PageContext object that provides JSP programmers with access to the implicit objects discussed in this table.
request	An object that represents the client request and is normally an instance of a class that implements HttpServletRequest. If a protocol other than HTTP is used, this object is an instance of a subclass of javax.servlet.Servlet-Request. It uses the getParameter() method to access the request parameter.
response	An object that represents the response to the client and is normally an instance of a class that implements HttpServletResponse (package javax.servlet.http). If a protocol other than HTTP is used, this object is an instance of a class that implements javax.servlet.ServletResponse.
session	A javax.servlet.http.HttpSession object that represents the client session information. This object is available only in pages that participate in a session.

To run JSP files: all JSP code should be copied (Deployed) into webapps folder in the tomcat server. To execute the file, type: http://server-ip:8080/Filename.jsp

Self Activity

1. Sample program : Simple display on browser

```
/* type this as first.jsp */
<html> <body>
<%
out.print("Hello World!");
%>
</body> </html>
```

2. Sample program to display current date

```
<%@ page  language="java" import="java.util.*" %>
<html> <body>
Current Date time: <%=new java.util.Date()%>
</body> </html>
```

3. Sample program to add two numbers “AddNumbers.jsp”

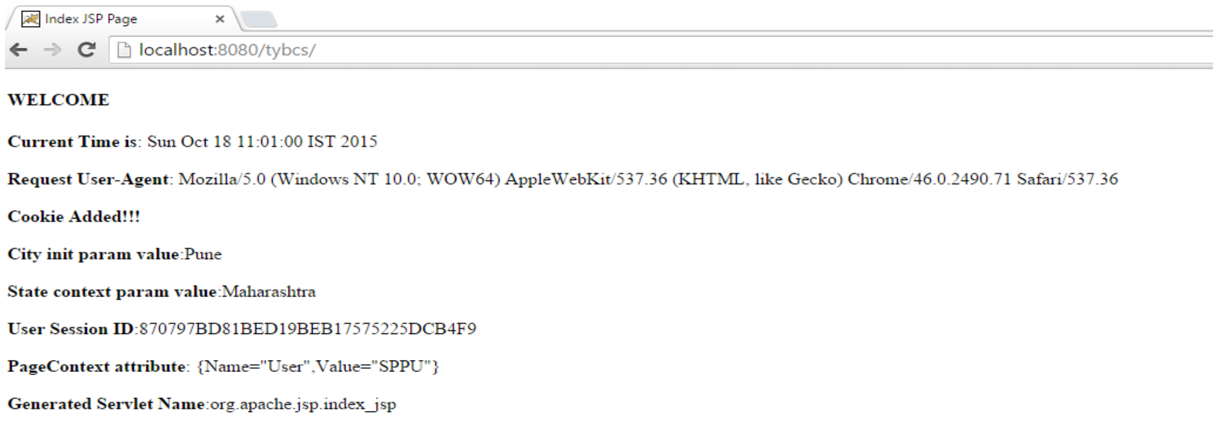
```
<%@ page language="java"%>
<html> <head>
<title>Add number program in JSP</title>
</head>
<body>
<form method = “post” action = “AddNumbers.jsp”>
Enter Number 1 <input type =”text” name = “No1”>
Enter Number 2 <input type =”text” name = “No2”>
<input type="submit" value="Get Result"/>
<%
 int a=Integer.parseInt(request.getParameter(“No1”));
 int b=Integer.parseInt(request.getParameter(“No2”));
  int result=a+b;
  out.print("Additon of a and b :"+result);
%>
```

```
</form>  </body> </html>
```

Lab Assignments

SET A

1. Write a Program to make use of following JSP implicit objects:
   i. out: To display current Date and Time.
   ii. request: To get header information.
   iii. response: To Add Cookie
   iv. config: get the parameters value defined in <init-param>
   v. application: get the parameter value defined in <context-param>
   vi. session: Display Current Session ID
   vii. pageContext: To set and get the attributes.
   viii. page: get the name of Generated Servlet



WELCOME

Current Time is: Sun Oct 18 11:01:00 IST 2015

Request User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36

Cookie Added!!!

City init param value:Pune

State context param value:Maharashtra

User Session ID:870797BD81BED19BEB17575225DCB4F9

PageContext attribute: {Name="User",Value="SPPU"}

Generated Servlet Name:org.apache.jsp.index_jsp

2. Create a JSP page which will accept the file extension and display all files in the current directory having that extension. Each filename should appear as a hyperlink on screen.

SET B

1. Create a JSP page, which accepts user name in a text box and greets the user according to the time on server side. Example: User name : ABC
Output : Good morning ABC / Good Afternoon ABC/ Good Evening ABC

2. Create a JSP page for an online multiple choice test. The questions are randomly selected from a database and displayed on the screen. The choices are displayed using radio buttons. When the user clicks on next, the next question is displayed. When the user clicks on submit, display the total score on the screen.

Signature of the instructor ☐ Date / /

Assignment Evaluation **Signature**

0: Not done	☐	2: Late Complete	☐	4: Complete	☐
1: Incomplete	☐	3: Needs improvement	☐	5: Well Done	☐

Assignment 5: Multithreading

Objectives

- **To create and use threads in java**
- **To demonstrate multithreading**

Reading

You should read the following topics before starting this exercise:
1. Thread class
2. Runnable interface
3. Thread lifecycle
4. Thread methods

Ready Reference

Introduction

Nearly every operating system supports the concept of processes -- independently running programs that are isolated from each other to some degree. Threading is a facility to allow multiple activities to coexist within a single process. Java is the first mainstream programming language to explicitly include threading within the language itself.

A process can support multiple threads, which appear to execute simultaneously and asynchronously to each other. Multiple threads within a process share the same memory address space, which means they have access to the same variables and objects, and they allocate objects from the same heap.

Every Java program uses threads:-
Every Java program has at least one thread -- the **main** thread. When a Java program starts, the JVM creates the main thread and calls the program's main() method within that thread. The JVM also creates other threads that are mostly invisible to you -- for example, threads associated with garbage collection, object finalization, and other JVM housekeeping tasks. Other facilities create threads too, such as the AWT or Swing UI toolkits, servlet containers, application servers, and RMI (Remote Method Invocation).

Thread Lifecycle:

The lifecycle of thread consist of several states which thread can be in. Each thread is in one state at any given point of time.
1. New State: - Thread object was just created. It is in this state before the start () method is invoked. At this point the thread is considered not alive.
2. Runnable or ready state:- A thread starts its life from Runnable state. It enters this state the time start() is called but it can enter this state several times later. In this state, a thread is ready to run as soon as it gets CPU time.
3. Running State:- In this state, a thread is assigned a processor and is running. The thread enters this state only after it is in the Runnable state and the scheduler assigns a processor to the thread.

4. Sleeping state: - When the sleep method is invoked on a running thread, it enters the Sleeping state.
5. Waiting State:- A thread enters the waiting state when the wait method is invoked on the thread. When some other thread issues a notify () or notifyAll (), it returns to the Runnable state().
6. Blocked State: - A thread can enter this state because it is waiting for resources that are held by another thread – typically I/O resources.
7. Dead State:- This is the last state of a thread. When the thread has completed execution that is its run () method ends, the thread is terminated. Once terminated, a thread can't be resumed.

Thread Creation
There are two ways to create thread in java;
1. Implement the Runnable interface (java.lang.Runnable)
2. By Extending the Thread class (java.lang.Thread)

1. Implementing the Runnable Interface

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

The procedure for creating threads based on the Runnable interface is as follows:
1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.
2. An object of Thread class is created by passing a Runnable object as argument to the Thread constructor. The Thread object now has a Runnable object that implements the run() method.
3. The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.
4. The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

Example:

```
class MyRunnable implements Runnable
{
  public void run() //define run method
       {
              //thread action
       }
}
...
MyRunnable r = new MyRunnable(); //create object
Thread t = new Thread(r); //create Thread object
t.start(); //execute thread
```

2. Extending java.lang.Thread class

A class can also extend the `Thread` class to create a thread. When we extend the Thread class, we should override the run method to specify the thread action.

```
class MyThread extends Thread
```

```
{
  public void run() //override run method
       {
             //thread action
       }
}
...
MyThread t = new MyThread(); //create Thread object
t.start(); //execute thread
```

Important methods of the Thread class:

Method	Description
`static int activeCount()`	Returns the number of active threads in the current thread's thread group.
`static Thread currentThread()`	Returns a reference to the currently executing thread object.
`String getName()`	Returns this thread's name.
`int getPriority()`	Returns this thread's priority.
`ThreadGroup getThreadGroup()`	Returns the thread group to which this thread belongs.
`void interrupt()`	Interrupts this thread.
`boolean isAlive()`	Tests if this thread is alive.
`boolean isDaemon()`	Tests if this thread is a daemon thread.
`boolean isInterrupted()`	Tests whether this thread has been interrupted.
`void join()`	Waits for this thread to end.
`void setName(String name)`	Changes the name of this thread.
`void setPriority(int newPriority)`	Changes the priority of this thread.
`void sleep(long mSec)`	Causes the currently executing thread to sleep.
`void start()`	Causes this thread to begin execution.
`String toString()`	Returns a string representation of this thread, including the thread's name, priority, and thread group.
`static void yield()`	Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Thread priorities

Every thread has a priority. A priority is an integer from 1 to 10 inclusive, where 10 is the highest priority, referred to as the ***maximum priority***, and 1 is the lowest priority, also known as the ***minimum priority***. The normal priority is 5, which is the default priority for each thread.

To set a thread's priority, use the setPriority() method. You can use an integer from 1 to 10, or you can use static, final variables defined in the Thread class. These variables are

i. Thread.MIN_PRIORITY: Minimum priority i.e. 1
ii. Thread.MAX_PRIORITY: Maximum priority i.e. 10
iii. Thread.NORM_PRIORITY: Default priority i.e. 5

Interthread Communication

When multiple threads are running, it becomes necessary for the threads to communicate with each other. The methods used for inter-thread communication are join(), wait(), notify() and notifyAll().

Self Activity

1. Below is a program that illustrates instantiation and running of threads using the Runnable interface.

```
class RunnableThread implements Runnable {
    Thread runner;
    public RunnableThread() {
    }
    public RunnableThread(String threadName) {
        runner = new Thread(this, threadName); // (1) Create a new
thread.
        System.out.println(runner.getName());
        runner.start(); // (2) Start the thread.
    }
    public void run() {
        //Display info about this particular thread
        System.out.println(Thread.currentThread());
    }
}

public class RunnableExample {
    public static void main(String[] args) {
        Thread thread1 = new Thread(new RunnableThread(), "thread1");
        Thread thread2 = new Thread(new RunnableThread(), "thread2");
        RunnableThread thread3 = new RunnableThread("thread3");
        //Start the threads
        thread1.start();
        thread2.start();
        try {
            //delay for one second
            Thread.currentThread().sleep(1000);
        } catch (InterruptedException e) {
        }
        //Display info about the main thread
        System.out.println(Thread.currentThread());
    }
}
```

2. Creating multiple threads using the Thread class.

```
class MyThread extends Thread
{
      String message;
      MyThread(String message)
      {
            this.message = message;
      }
      public void run()
      {
            try
            {
                  for(int i=1; i<=5; i++)
                  {
                        System.out.println(message + ”-” + i);
                        Thread.sleep(5000); //sleep for 5 seconds
                  }
            }
            catch(InterruptedException ie) { }
      }
}
public class MultipleThreadDemo
{
      public static void main( String[] args)
      {
            MyThread t1 = new MyThread("One");
    MyThread t2 = new MyThread("Two");
    System.out.println(t1);
    System.out.println(t2);
    t1.start();
    t2.start();
     }
}
```

Lab Assignments

SET A

1. Write a program that create 2 threads – each displaying a message (Pass the message as a parameter to the constructor). The threads should display the messages continuously till the user presses ctrl-c. Also display the thread information as it is running.

2. Write a java program to calculate the sum and average of an array of 1000 integers (generated randomly) using 10 threads. Each thread calculates the sum of 100 integers. Use these values to calculate average. [Use join method]

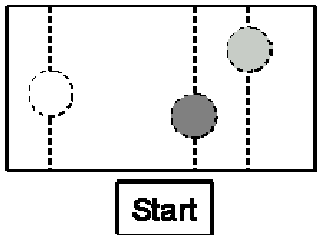
Set A Extra programs for practice

1. Define a thread called "PrintText_Thread" for printing text on command prompt for n number of times. Create three threads and run them. Pass the text and n as parameters to the thread constructor. Example:
   i. First thread prints "I am in FY" 10 times
   ii. Second thread prints "I am in SY" 20 times
   iii. Third thread prints "I am in TY" 30 times

SET B

1. Write a program for a simple search engine. Accept a string to be searched. Search for the string in all text files in the current folder. Use a separate thread for each file. The result should display the filename, line number where the string is found.

2. Define a thread to move a ball inside a panel vertically. The Ball should be created when user clicks on the Start Button. Each ball should have a different color and vertical position (calculated randomly). **Note**: Suppose user has clicked buttons 5 times then five balls should be created and move inside the panel. Ensure that ball is moving within the panel border only.



SET C

1. Write a java program to create a class called FileWatcher that can be given several filenames. The class should start a thread for each file name. If the file exists, the thread will write a message to the console and then end. If the filw does not exist, the thread will check for the existence of its file after every 5 seconds till the file gets created.

2.Write a program to simulate traffic signal using threads.

3. Write a program to show how three thread manipulate same stack , two of them are pushing elements on the stack, while the third one is popping elements off the stack.

Signature of the instructor		Date	/ /

Assignment Evaluation **Signature**

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 6: Networking

Objectives

- **Introduction to the java.net package**
- **Connection oriented transmission – Stream Socket Class**

Reading

You should read the following topics before starting this exercise

1. Networking Basics
2. The java.net package – InetAddress class, URL class, URLConnection class
3. Connection oriented communication –ServerSocket class, Socket class

Ready Reference

Introduction

One of the important features of Java is its networking support. Java has classes that range from low-level TCP/IP connections to ones that provide instant access to resources on the World Wide Web. Java supports fundamental networking capabilities through java.net package.

Networking Basics

Protocol

A protocol is a set of rules and standards for communication. A protocol specifies the format of data being sent over the Internet, along with how and when it is sent. Three important protocols used in the Internet are:
1. IP (Internet Protocol): is a network layer protocol that breaks data into small packets and routes them using IP addresses.
2. TCP (Transmission Control Protocol): This protocol is used for connection-oriented communication between two applications on two different machines. It is most reliable and implements a connection as a stream of bytes from source to destination.
3. UDP (User Datagram Protocol): It is a connection less protocol and used typically for request and reply services. It is less reliable but faster than TCP.

Addressing

1. **MAC Address**: Each machine is uniquely identified by a physical address, address of the network interface card. It is 48-bit address represented as 12 hexadecimal characters:
   For Example: 00:09:5B:EC:EE:F2
2. **IP Address**: It is used to uniquely identify a network and a machine in the network. It is referred as global addressing scheme. Also called as logical address. Currently used type of IP addresses is: Ipv4 – 32-bit address and Ipv6 – 128-bit address.
   For Example:
   Ipv4 – 192.168.16.1 Ipv6 – 0001:0BA0:01E0:D001:0000:0000:D0F0:0010
3. **Port Address:** It is the address used in the network to identify the application on the network.

Domain Name Service (DNS)

It is very difficult to remember the IP addresses of machines in a network. Instead, we can identify a machine using a “domain name” which is character based naming mechanism. Example: www.google.com. The mapping between the domain name and the IP address is done by a service called DNS.

URL

A URL (Uniform Resource Locator) is a unique identifier for any resource located on the Internet. The syntax for URL is given as:

```
<protocol>://<hostname>[:<port>][/<pathname>][/<filename>[#<section>]]
```

For Example: http://java.sun.com/j2se/1.5.0/download.jsp

Sockets

A socket represents the end-point of the network communication. It provides a simple read/write interface and hides the implementation details network and transport layer protocols. It is used to indicate one of the two end-points of a communication link between two processes. When client wishes to make connection to a server, it will create a socket at its end of the communication link.
The socket concept was developed in the first networked version of UNIX developed at the University of California at Berkeley. So sockets are also known as Berkeley Sockets.

Ports

A port number identifies a specific application running in the machine. A port number is a number in the range 1-65535.
Reserved Ports: TCP/IP protocol reserves port number in the range 1-1023 for the use of specified standard services, often referred to as “well-known” services.

Client-Server

It is a common term related to networking. A server is a machine that has some resource that can be shared. A server may also be a proxy server. Proxy server is a machine that acts as an intermediate between the client and the actual server. It performs a task like authentications, filtering and buffering of data etc. it communicates with the server on behalf of the client.
A client is any machine that wants to use the services of a particular server. The server is a permanently available resource, while the client is free to disconnect after it’s job is done.

The java.net Package

The java.net package provides classes and interfaces for implementing network applications such as sockets, network addresses, Uniform Resource Locators (URLs) etc. some important interfaces, classes and exceptions are :

InetAddress Class

This class is used to represent numerical IP addresses and domain name for that address i.e. it supports both numeric IP address and hostnames. There are no public constructors for InetAddress class. Instead, there are static methods that return InetAddress instances. Such methods are called as **factory methods**.

1. static InetAddress getLocalHost(): represents the IP address of the local host.
2. static InetAddress getByName(String hostname): represents the IP address of the host name passed to it. hostName can be "www.google.com", or hostname can be "130.95.72.134".
3. static InetAddress[] getAllByName(String hostname): represents an array of IP addresses of the specified host name.

All of these methods throw UnknownHostException.

Some of the instance methods of the InetAddress class are:

1. byte[] getAddress(): returns the raw IP address of the InetAddress object.
2. String getHostAddress(): returns the IP address string.
3. String getHostName(): Gets the host name for the IP address.

URL Class

As the name suggests, it provides a uniform way to locate resources on the web. Every browser uses them to identify information on the web. The URL class provides a simple API to access information across net using URLs.

The class has the following constructors:

1. **URL(String url_str)**: Creates a URL object based on the string parameter. If the URL cannot be correctly parsed, a MalformedURLException will be thrown.
2. **URL(String protocol, String host, String path)**: Creates a URL object with the specified protocol, host and path.
3. **URL(String protocol, String host, int port, String path)**: Creates a URL object with the specified protocol, host, port and file path.
4. **URL(URL urlObj, String urlSpecifier)**: Allows you to use an existing URL as a reference context and then create a new URL from that context.

Some important methods are given below:

Method	Description
URLConnection openConnection()	This method establishes a connection and returns a stream
Object getContent()	This method makes a connection and reads the contents
String getFile()	This method returns the filename part of the URL.
String getHost()	This method returns only the host name part from the URL.
String getPort()	This method returns the port number part from the URL. This is an optional component and returns –1 if not present.
String getProtocol()	This method returns the name of the protocol used in URL.
String getRef()	This method returns the anchor reference part from the URL.
int equals(Object)	This method compares whether two URL objects represents the same resource.
InputStream openStream()	This method establishes a connection and returns a stream for reading it.

URLConnection Class

This class is useful to actually connect to a website or resource on a network and get all the details of the website. Using the openConnection() method, we should establish a contact with the site on Internet. Method returns URLConnection object. Then using

URLConnection class methods, we can display all the details of the website and also content of the webpage whose name is given in URL.

Some important methods are:

Method	Description
void connect()	Establishes a connection between the application and the resource
Object getContent()	Reads the contents of the resource
int getContentLength()	Returns the value of the “content-length” header field, if such a field exists. Returns –1 if no content-length field was specified.
String getContentType()	Returns the value of the “content-type” header field, if such a field exists. Returns null if no content-type field was specified.
long getExpiration()	Returns the value of the “Expires” header field, expressed as the number of seconds since January 1, 1970 GMT. If no such a header field was specified, a value of zero will be returned.
InputStream getInputStream()	Returns an InputStream object that reads the contents of the resource pointed to by the URLConnection
OutputStream getOutputStream()	Returns an OutputStream object that writes to the remote connection
URL getURL()	Returns a URL object representing the location of the resource pointed by the URL connection
long getDate()	Returns the value of the “Date” header field, expressed as the number of seconds since January 1, 1970 GMT. Returns 0 if not specified.
long getLastModified()	Returns the date of the “Last-modified” header field, expressed as the number of seconds since January 1 1970 GMT. Returns 0 if not specified.

Connection Oriented Communication

Using Socket Java performs the network communication. Sockets enables to transfer data through certain port. Socket class of Java makes it easy to write the socket programs. Sockets are broken into two types:

1. Datagram sockets (UDP Socket)

Java uses java.net.DatagramSocket class to create datagram socket. The java.net.DatagramPacket represents a datagram.

2. Stream Socket (TCP Socket)

Java uses java.net.Socket class to create stream socket for client and java.net.ServerSocket for server.

ServerSocket Class

This class is used to create a server that listens for incoming connections from clients. It is possible for client to connect with the server when server socket binds itself to a specific port. The various constructors of the ServerSocket class are:

1. **ServerSocket(int port):** binds the server socket to the specified port number. If 0 is passed, any free port will be used. However, clients will be unable to access the service unless notified the port number.
2. **ServerSocket(int port, int numberOfClients)**: Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets.
3. **ServerSocket(int port, int numberOfClients, InetAddress address):** Binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets and the IP address to which this socket binds.

The various methods of this class are:

Method	Description
accept()	Listens for socket connection. This is a blocking I/O operation, and will not return until a connection is made. When a connection is established a Socket object will be returned.
getLocalSocketAddresss()	Returns local socket information.
bind()	Binds a connection to destination.
close()	Closes the connection.
getChannel()	Returns channel for connection.
getInetAddress()	Returns address of connection.
getLocalPort()	Returns local port used for connection.
isBound()	Checks whether socket is bound.
isClosed()	Checks whether socket is closed.
toString()	Converts server socket to string.

Socket Class

This class is used to represents a TCP client socket, which connects to a server socket and initiate protocol exchanges. The various constructors of this class are:

1. **Socket(InetAddress address, int port):** creates a socket connected to the specified IP address and port. Can throw an IOException.
2. **Socket(String host, int port)**: creates a socket connected to the specified host and port. Can throw an UnknownHostException or an IOException.

The various methods of this class are:

Method	Description
getLocalSocketAddresss()	Returns local socket information.
bind()	Binds a connection to destination.
close()	Closes the connection.
connect()	Makes connection to destination
getChannel()	Returns channel for connection.
getInetAddress()	Returns address of connection.
getLocalPort()	Returns local port used for connection.
getInputStream()	Returns input stream for connection.
getKeepAlive()	Returns indication of keep alive option enabled.
getLocalAddress()	Returns local address
getOutputStream()	Returns output stream for connection.
getPort()	Returns remote used for connection.
getRemoteSocketAddress()	Returns remote socket information.
isBound()	Checks whether socket is bound.
isClosed()	Checks whether socket is closed.
isConnected()	Checks whether socket is connected.
toString()	Converts server socket to string.

Self Activity

1. Sample program to find IP address of a web-site (Internet connection required)

```
/* Program to find the IPAddress of a website*/
```

```
import java.io.*;
import java.net.*;
class AddressDemo
{
      public static void main(String args[]) throws IOException
      {
            BufferedReader br=new BufferedReader(new
             InputStreamReader(System.in));
            System.out.print("Enter a website name: ");
            String site=br.readLine();
            try
            {
                  InetAddress ip=InetAddress.getByName(site);
                  System.out.print("The IP address is: " + ip);
            }
            catch(UnknownHostException ue)
            {
                  System.out.println("WebSite not found");
            }
      }
}
```

2. Sample program for client-server communication

/* Program to display socket information on client machine*/

```
import java.net.*;
import java.io.*;
public class Server
{
      public static void main(String args[])throws   UnknownHostException ,
IOException
      {
            ServerSocket ss = new ServerSocket(4444);
            System.out.println("Server Started");
            Socket s = ss.accept();
            System.out.println("Connected to client");
      }
}

public class Client
{
      public static void main(String args[]) throws  UnknownHostException,
IOException
      {
            Socket s = new Socket ("localhost", 4444);
            System.out.println (s.getInetAddress());
            System.out.println (s.getPort());
            System.out.println (s.getLocalPort());
            s.close();
      }
}
```

Lab Assignments

SET A

1. Write a client-server program which displays the server machine's date and time on the client machine.

2. Write a program which sends the name of a text file from the client to server and displays the contents of the file on the client machine. If the file is not found, display an error message.

SET B

1. Write a program to accept a list of file names on the client machine and check how many exist on the server. Display appropriate messages on the client side.

2. Write a server program which echoes messages sent by the client. The process continues till the client types "END".

SET C

1. Write a program for a simple GUI based chat application between client and server.

Signature of the instructor		Date	/ /

Assignment Evaluation				**Signature**	
0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	