



Savitribai Phule Pune University

T.Y.B.C.A. (Science) Semester-V

(2019 Pattern)

BCA 356

DSE I Lab

Programming in Java Laboratory

Work Book

**To be implemented from
Academic Year 2021–2022**

Name: _____

College Name: _____

Roll No.: _____ Seat No: _____

Academic Year: _____

From the Chairman's Desk

It gives me a great pleasure to present this workbook prepared by the Board of studies in Computer Applications.

The workbook has been prepared with the objectives of bringing uniformity in implementation of lab assignments across all affiliated colleges, act as a ready reference for both fast and slow learners and facilitate continuous assessment using clearly defined rubrics.

The workbook provides, for each of the assignments, the aims, pre-requisites, related theoretical concepts with suitable examples wherever necessary, guidelines for the faculty/lab administrator, instructions for the students to perform assignments and a set of exercises divided into three sets.

I am thankful to the Chief Editor and the entire team of editors appointed. I am also thankful to the members of BOS. I thank everyone who have contributed directly or indirectly for the preparation of the workbook.

Constructive criticism is welcome and to be communicated to the Chief Editor. Affiliated colleges are requested to collect feedbacks from the students for the further improvements.

I am thankful to Hon. Vice Chancellor of Savitribai Phule Pune University Prof. Dr. Nitin Karmalkar and the Dean of Faculty of Science and Technology Prof. Dr. M G Chaskar for their support and guidance.

Prof. Dr. S S Sane
Chairman, BOS in Computer Applications
SPPU, Pune

Editors:

Teacher Name	College Name
Mrs. Aparna Gohad	MES Abasaheb Garware College of Arts and Science, Pune
Mrs. Vandana Nemane	PAD. DR.D.Y.Patil Arts, Commerce & Science College, Pimpri, Pune
Mrs. Deepashree Mehendale	PAD. DR.D.Y.Patil Arts, Commerce & Science College, Pimpri, Pune
Mrs. Sheetal Patil	P.E.S.'S Modern College OF Arts, Commerce & Science College Shivajinagar Pune
Mrs. Jyoti Mahatme	KRT Arts, BH Commerce and AM Science (KTHM) College, Nashik
Mrs. Akshada Kulkarni	MIT Arts Commerce and Science College, Alandi(D), Pune

Compiled By:

Mrs. Aparna Gohad
MES Abasaheb Garware College of Arts and Science, Pune
Chairman, DSE I Lab Programming in Java laboratory

Reviewed By:

1. Dr. Mr. M.N. Shelar
K.R.T. Arts, B.H. Commerce and A.M. Science College, Nashik
BOS, BCA(Science)
2. Dr. Mrs. Pallavi Bulakh
PES Modern College, Ganeshkhind, Pune
BOS, BCA(Science)

Introduction

1. About the work book:

This workbook is intended to be used by T.Y.B.C.A. (Science) students for the BCA 356 DSE I Programming in Java Laboratory Assignments in Semester–V. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

2. The objectives of this workbook are:

- 1) Defining the scope of the course.
- 2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
- 3) To have continuous assessment of the course and students.
- 4) Providing ready reference for the students during practical implementation.
- 5) Provide more options to students so that they can have good practice before facing the examination.
- 6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. How to use this workbook:

- 1) The Java workbook is divided into six assignments.
- 2) Each Java assignment has three SETs.
- 3) It is mandatory for students to complete the SET A and SET B in given slot.
- 4) Set C prepares the students for the viva in the subject. Students should spend additional time either at home or in the Lab and solve these problems so that they get a deeper understanding of the subject.

4. Instructions to the students

- 1) Students are expected to carry this book every time they come to the lab for practical.
- 2) Students should prepare oneself beforehand for the Assignment by reading the relevant material.
- 3) Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible given in this work book.
- 4) Students will be assessed for each exercise on a scale from 0 to 5.

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

5. Guidelines for Instructors

- 1) Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating on Projector.
- 2) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- 3) The value should also been written on assignment completion page of the respective Lab course.

6. Guidelines for Lab administrator

You have to ensure appropriate hardware and software is made available to each student.

The operating system and software requirements on server side and also client-side areas given below:

- Operating system: Linux
- Editor: Any linux based editor like vi, gedit, eclipse etc.
- Compiler: javac (Note : JAVA 8 and above version)
- Apache tomcat webserver: Tomcat 5.5 and above version

INDEX

Assignment No	Assignment Name	No. of Sessions	Page nos.
1	Basics of Java, Classes and Objects	2	8
2	Inheritance and Interface	2	17
3	Collection, Exception handling and I/O	2	24
4	Swing	3	43
5	Database Programming	3	60
6	Servlets and JSP	3	69

Assignment Completion Sheet

DSE I Lab			
Programming in Java			
Assignment No	Assignment Name	Marks (out of 5)	Teachers Sign
1	Basics of Java, Classes and Objects		
2	Inheritance and Interface		
3	Collection, Exception handling and I/O		
4	Swing		
5	Database Programming		
6	Servlets and JSP		
Total (Out of 30)			
Total (Out of 15)			

CERTIFICATE

This is to certify that Mr./Ms. _____

has successfully completed **TYBCA (Science) BCA356 DSE I**

Lab Programming in Java Semester V course in year

_____ and his/her seat no. is _____.

He / She have scored _____ marks out of 15.

Instructor

HOD/Coordinator

Internal Examiner

External Examiner

Assignment No.	1	No. of Sessions:	02
Assignment Name :	Basics of Java, Classes and Objects		

Prerequisites:-

- OOP's Concepts
- Keywords used in Java
- Structure of Java Program

Java is extensively used programming language. It is been used by many programmers from beginners to experts. If we look into the phases of program development, we come across many phases such as:

Step 1: Creation of a Java program:-

Creating a Java program means writing of a Java program on any editor or IDE. After creating a Java program provide .java extension to file. It signifies that the particular file is a Java source code. Also, whatever progress we do from writing, editing, storing and providing .java extension to a file, it basically comes under creating of a Java program.

Step 2: Compiling a Java Program

Our next step after creation of program is the compilation of Java program. Generally Java program which we have created in step 1 with a .java extension, it is been compiled by the compiler. Suppose if we take example of a program say Welcome.java, when we want to compile this we use command such as javac. After opening command prompt or shell console we compile Java program with .java extension as:

javac Welcome.java

After executing the javac command, if no errors occur in our program we get a .class file which contains the bytecode. It means that the compiler has successfully converted Java source code to bytecode. Generally bytecode is been used by the JVM to run a particular application. The final bytecode created is the executable file which only JVM understands. As Java compiler is invoked by javac command, the JVM is been invoked by java command. On the console window you have to type:

java Welcome

This command will invoke the JVM to take further steps for execution of Java program.

Step 3: Program loading into memory by JVM:-

JVM requires memory to load the .class file before its execution. The process of placing a program in memory in order to run is called as Loading. There is a class loader present in the JVM whose main functionality is to load the .class file into the primary memory for the execution. All the .class files required by our program for the execution is been loaded by the class loader into the memory just before the execution.

Step 4: Bytecode Verification by JVM:-

In order to maintain security of the program JVM has bytecode verifier. After the classes are loaded in to memory, bytecode verifier comes into picture and verifies bytecode of the loaded class in order to maintain security. It check whether bytecodes are valid. Thus it prevent our computer from malicious viruses and worms.

Step 5: Execution of Java program: -

Whatever actions we have written in our Java program, JVM executes them by interpreting bytecode. JVM uses JIT compilation unit to which we even call just-in-time compilation.

JVM (Java Virtual Machine):-

JVM, i.e., Java Virtual Machine. JVM is the engine that drives the Java code. Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine. When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform. Bytecode is an intermediary language between Java source and the host system. It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

A Simple Java Program

```
public class Test
{
    public static void main(String args[])
    {
        System.out.println("Hello students");
    }
}
```

class keyword is used to declare a class in Java.

- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main () method is executed by the JVM, so it doesn't require creating an object to invoke the main () method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.

- **String[] args** or **String args[]** is used for command line argument.
System.out.println() is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

To compile the program the command is

```
javac Test.java
```

To execute the program the command is

```
java Test
```

Different ways of reading input from console in Java -

1. Using command line argument:

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input.

```
public class Test
{
    public static void main(String[] args)
    {
        int a,b,c;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
        c = a+b;
        System.out.println("Addition:"+c);
    }
}
javac Test.java
java Test 10 20
```

2. Using Buffered Reader Class

This is the Java classical method to take input. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

```
import java.io.*;
public 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. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however it also can be used to read input from the user in the command line.

```

import java.util.Scanner;
public 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();
    }
}

```

Class

Classes are the fundamental building blocks of any object-oriented language.

When you define a class, you declare its exact form and nature. You do this by specifying the data that it contains and the code that operates on that data. While very simple classes may contain only code or only data, most real-world classes contain both. A class is declared by use of the class keyword. A simplified general form of a class definition is shown here:

```

accessspecifier class classname
{
    DataType instancevariable1;

```

```

        DataType instancevariable 2;
        DataType instancevariable n;
        returntype methodname1 (parameter list)
        {
            Body of the method;
        }
        returntype methodname2 (parameter list)
        {
            Body of the method;
        }
    }

```

The data, or variables, defined within a class are called **instance variables**. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class. Variables defined within a class are called instance variables because each instance of the class contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another.

Objects

When you create a class, you are creating a new data type. You can use this type to declare objects of that type. However, obtaining objects of a class is a two-step process.

First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object. Second, you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the new operator. The new operator dynamically allocates memory for an object and returns a reference to it.

This reference is, essentially, the address in memory of the object allocated by new. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

Syntax for declaring Object:

```
Student stud= new Student();
```

This statement combines the two steps just described.

It can be rewritten like this to show each step more clearly:

```
Student stud;// declare reference to object
```

```
Stud = new Student (); // allocate a object
```

The first line declares stud as a reference to an object of type Student. At this point, stud does not yet refer to an actual object. The next line allocates an object and assigns a reference to it to stud. After the second line executes, you can use stud as if it were a Student object. But in reality, stud simply holds, in essence, the memory address of the actual Student object.

Array of Objects: - An array that contains **class type elements** are known as an **array of objects**. It stores the reference variable of the object. Before creating an array of objects, we must create an instance of the class by using the new keyword. Suppose, we have created a class named Student. We want to keep marks of 20 students. In this case, we will not create 20 separate objects of Student class. Instead of this, we will create an array of objects, as follows:

```
Student s[] = new Student[20];
```

```
public class Student
{
    int marks;
}
public class ArrayOfObjects
{
    public static void main(String args[])
    {
        Student std[] = new Student[3]; // array of reference variables of Student
        std[0] = new Student(); // convert each reference variable into Student object
        std[1] = new Student();
        std[2] = new Student();
        std[0].marks = 40; // assign marks to each Student element
        std[1].marks = 50;
        std[2].marks = 60;
        System.out.println("\n students average marks:"
            +(std[0].marks+std[1].marks+std[2].marks)/3);
    }
}
```

Constructors: Constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
[accesssmodifier] <class_name>()
{
}
```

```

public class Student
{
    private int id;
    private String name;
    public Student() //Default constructor
    {
        id=0;
        name="";
    }
    public void display()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[])
    {
        Student s1=new Student();
        s1.display();
    }
}

```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```

public class Student
{
    private int id;
    private String name;
    public Student(int i, String n)    //creating a parameterized constructor
    {
        id = i;
        name = n;
    }
}

```

```

public void display () //method to display the values
{
    System.out.println(id+" "+name);
}
public static void main(String args[])
{
    //creating objects and passing values
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
}
}

```

this keyword: This keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```

import java.io.*;
public class Student
{
    private int rollno;
    private String name;
    private float fee;
    public Student(int rollno,String name,float fee)
    {
        this.rollno=rollno;
        this.name=name;
        this.fee=fee;
    }
    public void display()
    {
        System.out.println(rollno+" "+name+" "+fee);
    }
    public static void main(String args[])
    {
        Student s1=new Student(111,"ankit",5000);
        s1.display();
    }
}

```


Lab Assignment

SET A

1. Write a Java program to accept a number from command prompt and generate multiplication table of a number. Accept number using BufferedReader class.
2. Write a Java Program to Reverse a Number. Accept number using command line argument.
3. Write a Java program to print the sum of elements of the array. Also display array elements in ascending order.
4. Write a Java program create class as MyDate with dd,mm,yy as data members. Write default and parameterized constructor. Display the date in dd-mm-yy format.(Use this keyword)

SET B

1. Define a class MyNumber having one private integer data member. Write a default constructor initialize it to 0 and another constructor to initialize it to a value. Write methods isNegative, isPositive, isOdd, iseven. Use command line argument to pass a value to the object and perform the above tests.
2. Write a java program which define class Employee with data member as name and salary. Program store the information of 5 Employees. (Use array of object)
3. Write a java program to create class Account (accno, accname, balance). Create an array of “n” Account objects. Define static method “sortAccount” which sorts the array on the basis of balance. Display account details in sorted order.

SET C

1. Define a class Person (id, name, dateofbirth). For dateofbirth create an object of MyDate class which is created in SET A 4. Define default and parameterized constructor. Also define accept and display function in Person and MyDate class. Call constructor and function of MyDate class from Person class for dateofbirth. Generate id automatically using static variable. Accept the details of n person and display the details.

Sample code:

```
public class Person
{
    private int id;
    private String name;
    private MyDate dob;
    private static int cnt=1;
    -----
    -----
}
```

Signature of the instructor: _____ **Date:** _____

Assignment Evaluation

0: Not Done []

1: Incomplete []

2: Late Complete []

3: Needs Improvement []

4: Complete []

5: Well done []

Assignment No.	2	No. of Sessions:	02
Assignment Name :	Inheritance and Interface		

Prerequisites:-

- Constructor/ Destructor
- Concept of Overloading and overriding

Inheritance

Inheritance is one of the feature of Object Oriented Programming System (OOPs) it allows the child class to acquire the properties (the data members) and functionality (the member functions) of parent class.

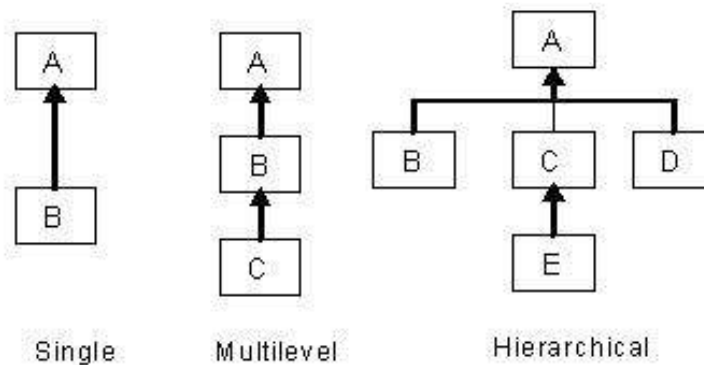
What is child class?

A class that inherits another class is known as child class, it is also known as derived class or subclass.

What is parent class?

The class that is being inherited by other class is known as parent class, super class or base class.

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 use of super keyword

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

Example:

```
public class A
{
    protected int num;
    public A(int num)
    {
        this.num = num;
    }
}
public class B extends A
{
    private int num;
    public B(int a, int b)
    {
        super(a); //should be the first line in the subclass constructor
        this.num = b;
    }
    public 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.

Example:

```
public class A
{
    public void display(int num) { //code }
}
public class B extends A
{
    public void display(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.display(10); //calls method of class A  
ref = new B();  
ref.display(20); //calls method of class B
```

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. final keyword can be for:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If any variable as final, value of final variable cannot change the (It will be constant).

2) Java final method

If any method as final, that method cannot be overridden in subclass.

3) Java final class

If any class as final, then that class cannot be extended ie creating subclass of final class is not allowed.

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 accessSpecifier class ClassName  
{  
...  
}
```

Abstract method

An abstract method is a method which has no definition. The definition is provided by the subclass.
abstract accessSpecifier returnType method(arguments);

Interface

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve* abstraction

There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

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

Example:

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

Package

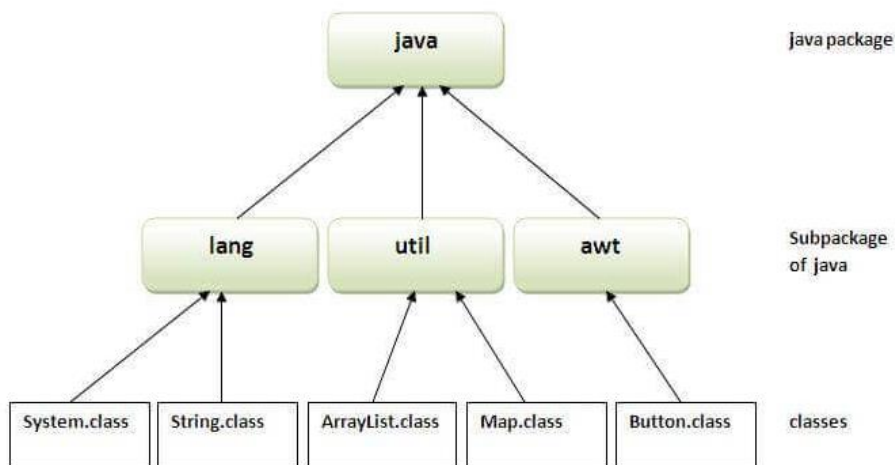
A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.



Simple example of java package

The package keyword is used to create a package in java.

//save as Simple.java

```
package mypack;
public class Simple
{
    public static void main(String args[])
    {
        System.out.println("Welcome to package");
    }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program -

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

Lab Assignment

SET A

1. Define a “Point” class having members – x,y(coordinates). Define default constructor and parameterized constructors. Define two subclasses “ColorPoint” with member as color and subclass “Point3D” with member as z (coordinate). Write display method to display the details of different types of Points.
2. Create a package named “Series” having three different classes to print series:
 - a. Fibonacci series
 - b. Cube of numbers
 - c. Square of numbers

Write a java program to generate “n” terms of the above series. Accept n from user.

3. Define a class Employee having members – id, name, salary. Define default constructor. 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 worker having the maximum total salary (salary + bonus).
4. Create a package “utility”. Define a class CapitalString under “utility” package which will contain a method to return String with first letter capital. Create a Person class (members – name, city) outside the package. Display the person name with first letter as capital by making use of CapitalString.

SET B

1. Define an interface “Operation” which has methods area(),volume(). Define a constant PI having a value 3.142. Create a class circle (member – radius), cylinder (members – radius, height) which implements this interface. Calculate and display the area and volume.
2. Write a Java program to create a super class Employee (members – name, salary). Derive a sub-class as Developer (member – projectname). Derive a sub-class Programmer (member – proglanguage) from Developer. Create object of Developer and display the details of it. Implement this multilevel inheritance with appropriate constructor and methods.s
3. Define an abstract class Staff with members name and address. Define two sub-classes of this class – FullTimeStaff (members - department, salary, hra - 8% of salary, da – 5% of salary) and PartTimeStaff (members - number-of-hours, rate-per-hour). Define appropriate constructors. Write abstract method as calculateSalary() in Staff class. Implement this method in subclasses. 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 along with their salary.

SET C

1. Create an interface — CreditCardInterface with method: viewCreditAmount(), useCard(), payCredit() and increaseLimit(). Create a class SilverCardCustomer (name, cardnumber (16digits), 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 []	1: Incomplete []	2: Late Complete []
3: Needs Improvement []	4: Complete []	5: Well done []

Assignment No.	3	No. of Sessions:	02
Assignment Name :	Collection, Exception handling and I/O		

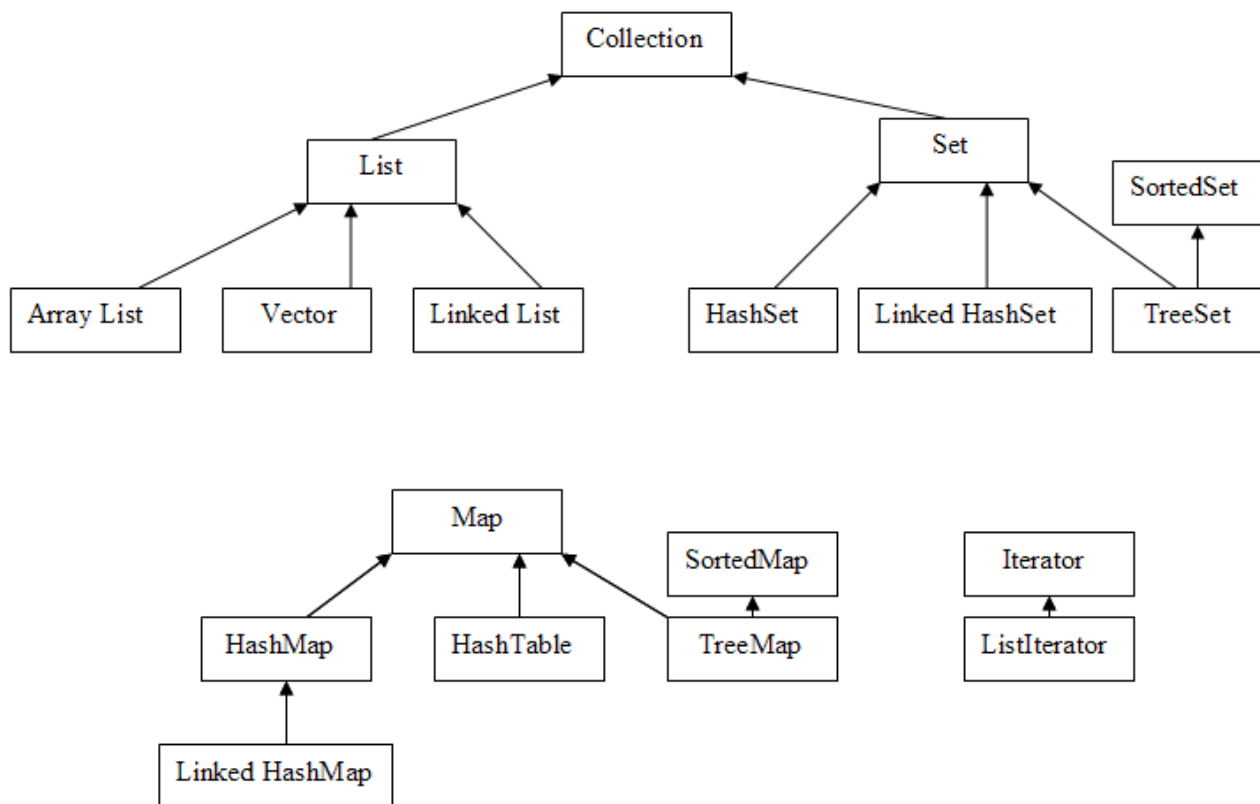
Prerequisite:

- Concept of Inheritance and Interface
- Basic concept of File handling

Java Collection

Collection: 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:



Interfaces

Interface Name	Description
Collection	This enables you to work with groups of objects; it is at the top of the collection's hierarchy.
List	This extends Collection and an instance of List stores an ordered collection of elements.
Set	This extends Collection to handle sets, which must contain unique elements.
SortedSet	This extends Set to handle sorted sets.
Map	This maps unique keys to values.
SortedMap	This extends Map so that the keys are maintained in an ascending order.

Collection Interface methods

Method	Description
boolean add(Object obj)	Adds obj to the invoking collection.
boolean addAll(Collection c)	Adds all the elements of c to the invoking collection.
void clear()	Removes all elements from the invoking collection.
boolean contains(Object obj)	Returns true if obj is an element of the invoking collection.
boolean containsAll(Collection c)	Returns true if the invoking collection contains all elements of c.
boolean equals(Object obj)	Returns true if the invoking collection and obj are equal.
int hashCode()	Returns the hash code for the invoking collection.

boolean isEmpty()	Returns true if the invoking collection is empty.
Iterator iterator()	Returns an iterator for the invoking collection.
boolean remove(Object obj)	Removes one instance of obj from the invoking collection. Returns true if the element was removed.
boolean removeAll(Collection c)	Removes all elements of c from the invoking collection.
boolean retainAll(Collection c)	Removes all elements from the invoking collection except those in c.
int size()	Returns the number of elements held in the invoking collection.
Object[] toArray()	Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
Object[] toArray(Object array[])	Returns an array containing only those collection elements whose type matches that of array.

List Interface

Stores sequence of elements may contain duplicate elements. It allows to store heterogeneous data.

Method	Description
void add(int index, Object obj)	Inserts obj into the invoking list at the index passed in the index. Any pre-existing elements at or beyond the point of insertion are shifted up.
boolean addAll(int index, Collection c)	Inserts all elements of c into the invoking list at the index passed in the index. Any pre-existing elements at or beyond the point of insertion are shifted up.
Object get(int index)	Returns the object stored at the specified index within the invoking collection.
int indexOf(Object obj)	Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, -1 is returned.

int lastIndexOf(Object obj)	Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, -1 is returned.
ListIterator listIterator()	Returns an iterator to the start of the invoking list.
ListIterator listIterator(int index)	Returns an iterator to the invoking list that begins at the specified index.
Object remove(int index)	Removes the element at position index from the invoking list and returns the deleted element.
Object set(int index, Object obj)	Assigns obj to the location specified by index within the invoking list.
List subList(int start, int end)	Returns a list that includes elements from start to end.1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

Set Interface and SortedSet Interface

Set Interface and SortedSet Interface is a collection that contains no duplicate elements. It contains the methods of the Collection interface(add (), clear(),contains() etc). A SortedSet is a Set that maintains its elements in ascending order. It allows to store homogeneous data. It adds the following methods:

Methods	Description
Comparator comparator()	Returns the invoking sorted set's comparator. If the natural ordering is used for this set, null is returned.
Object first()	Returns the first element in the invoking sorted set.
SortedSet headSet(Object end)	Returns a SortedSet containing those elements less than end that are contained in the invoking sorted set. Elements in the returned sorted set are also referenced by the invoking sorted set.
Object last()	Returns the last element in the invoking sorted set.
SortedSet subSet(Object start, Object end)	Returns a SortedSet that includes those elements between start and end.1. Elements in the returned collection are also referenced by the invoking object.
SortedSet tailSet(Object start)	Returns a SortedSet that contains those elements greater than or equal to start that are contained in the sorted set. Elements in the returned set are also referenced by the invoking object.

Map Interface

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

Method	Description
void clear()	Removes all key/value pairs from the invoking map.
boolean containsKey(Object k)	Returns true if the invoking map contains k as a key.
boolean containsValue(Object v)	Returns true if the map contains v as a value.
Set entrySet()	Returns a Set that contains the entries in the map. The set contains objects of type Map.Entry. This method provides a set-view of the invoking map.
boolean equals(Object obj)	Returns true if obj is a Map and contains the same entries.
Object get(Object k)	Returns the value associated with the key k.
int hashCode()	Returns the hash code for the invoking map.
boolean isEmpty()	Returns true if the invoking map is empty.
Set keySet()	Returns a Set that contains the keys in the invoking map. This method provides a set-view of the keys in the invoking map.
Object put(Object k, Object v)	Puts an entry in the invoking map, overwriting any previous value associated with the key.
void putAll(Map m)	Puts all the entries from m into this map.
Object remove(Object k)	Removes the entry whose key equals k.
int size()	Returns the number of key/value pairs in the map.
Collection values()	Returns a collection containing the values in the map. This method provides a collection-view of the values in the map.

List Implementation

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.
3. **Vector:** Implements Dynamic array. It is similar to ArrayList, but with two differences –
 1. Vector is synchronized.
 2. Vector contains many legacy methods that are not part of the collections framework.

Vector Constructors:

- Vector() : Default Vector with initial size zero.
- Vector(int size):Vector with initial capacity as size.
- Vector(int size, int increment): Constructs a vector whose initial capacity is specified by size and whose increment is specified by incr.
- Vector(Collection c):Construct a vector that contains the elements of collection c.

Set Implementation

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:** It 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.

Map Implementation

There are three general-purpose Map implementations — HashMap, LinkedHashMap, TreeMap

1. **HashMap:** The HashMap class uses a hashtable to implement the Map interface. This allows the execution time of basic operations, such as get() and put(), to remain constant even for large sets.
2. **LinkedHashMap:** This class extends HashMap and maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.

- 3. TreeMap:** The TreeMap class implements the Map interface by using a tree. A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.

Iterator

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

Methods by Iterator

Method	Description
boolean hasNext()	Returns true if there are more elements. Otherwise, returns false.
Object next()	Returns the next element. Throws NoSuchElementException if there is not a next element.
void remove()	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next().

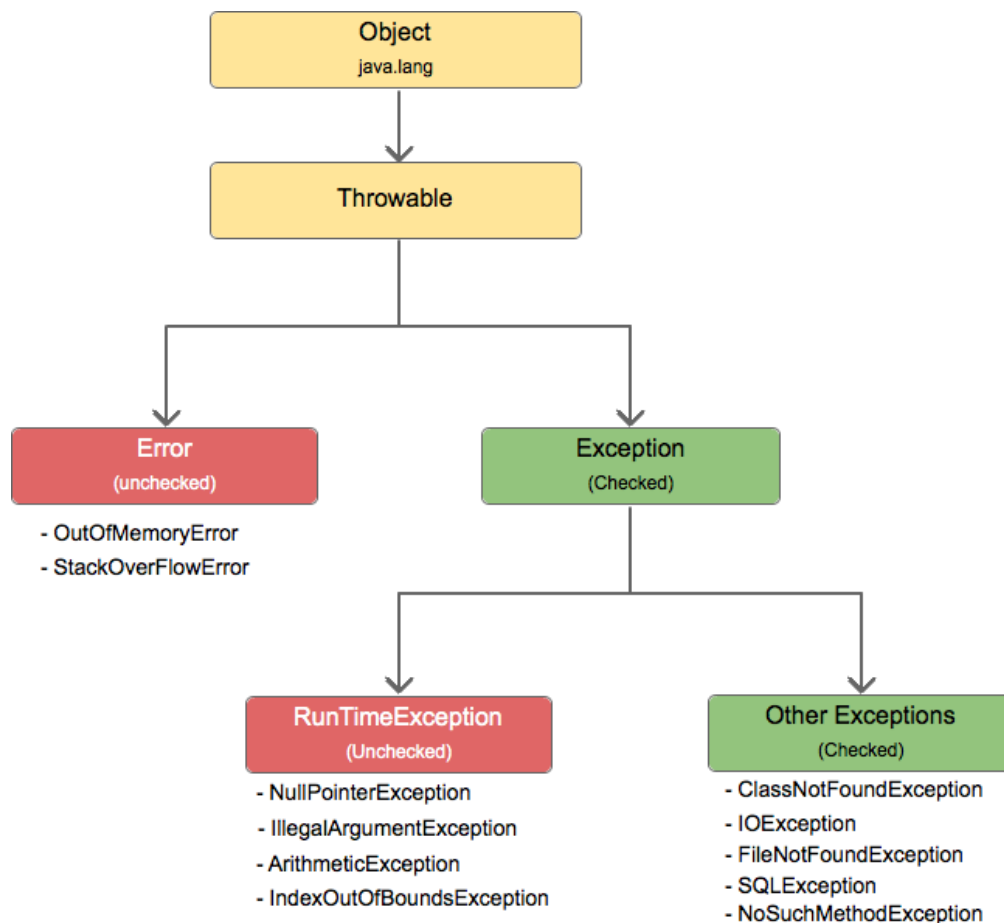
The Methods Declared by ListIterator

Methods	Description
boolean hasNext()	Returns true if there is a next element. Otherwise, returns false.
boolean hasPrevious()	Returns true if there is a previous element. Otherwise, returns false.
Object next()	Returns the next element. A NoSuchElementException is thrown if there is not a next element.
int nextIndex()	Returns the index of the next element. If there is not a next element, returns the size of the list.
Object previous()	Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
int previousIndex()	Returns the index of the previous element. If there is not a previous element, returns -1.
void remove()	Removes the current element from the list. An IllegalStateException is thrown if remove() is called before next() or previous() is invoked.

void set(Object obj)	Assigns obj to the current element. This is the element last returned by a call to either next() or previous().
----------------------	---

Exception Handling

- Exception Handling is the mechanism to handle runtime malfunctions. We need to handle such exceptions to prevent abrupt termination of program. The term exception means exceptional condition, it is a problem that may arise during the execution of program. A bunch of things can lead to exceptions, including programmer error, hardware failures, files that need to be opened cannot be found, resource exhaustion etc.
- Exception:** A Java Exception is an object that describes the exception that occurs in a program. When an exceptional event occurs in java, an exception is said to be thrown. The code that's responsible for doing something about the exception is called an exception handler.
- Exception class Hierarchy:** All exception types are subclasses of class Throwable, which is at the top of exception class hierarchy.



- Exception class is for exceptional conditions that program should catch. This class is extended to create user specific exception classes.
- RuntimeException is a subclass of Exception. Exceptions under this class are automatically defined for programs.

- Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

1. Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. Checked exceptions are checked **at compile-time**.

Checked Exceptions:

- Exception
- IOException
- FileNotFoundException
- ParseException
- ClassNotFoundException
- CloneNotSupportedException
- InstantiationException
- InterruptedException
- NoSuchMethodException
- NoSuchFieldException

2. Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. Unchecked exceptions are not checked at compile-time, but they are **checked at runtime**.

Unchecked Exceptions:

- ArrayIndexOutOfBoundsException
- ClassCastException
- IllegalArgumentException
- IllegalStateException
- NullPointerException
- ArithmeticException

3. Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Exception Handling Mechanism

In java, exception handling is done using five keywords,

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.

throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.
--------	---

Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.

Syntax of Java try-catch

```
try{
    //code that may throw an exception
}catch(Exception_class_Name ref){ }
```

```
public class TryCatchExample
{
    public static void main(String[] args)
    {
        try
        {
            int data=50/0; //may throw exception
            // if exception occurs, the remaining statement will not execute
            System.out.println("rest of the code");
        }
        // handling the exception
        catch(ArithmeticException e)
        {
            System.out.println(e);
        }
    }
}
```

Syntax of try-finally block

```
try{
    //code that may throw an exception
}finally{ }
```

```
public class TestFinallyBlock
{
    public static void main(String args[])
    {
        try
        {
            //below code do not throw any exception
            int data=25/5;
            System.out.println(data);
        }
        //catch won't be executed
    }
}
```

```

        catch(NullPointerException e)
        {
            System.out.println(e);
        }
        //executed regardless of exception occurred or not
        finally
        {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}

```

Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

```

public class TestThrow
{
    //function to check if person is eligible to vote or not
    public static void validate(int age)
    {
        if(age<18)
        {
            //throw Arithmetic exception if not eligible to vote
            throw new ArithmeticException("Person is not eligible to vote");
        }
        else
        {
            System.out.println("Person is eligible to vote!!");
        }
    }
    //main method
    public static void main(String args[])
    {
        //calling the function
        TestThrow.validate(13);
        System.out.println("rest of the code...");
    }
}

```

User defined Exception

You can also create your own exception sub class simply by extending java Exception class. You can define a constructor for your Exception sub class (not compulsory) and you can override the toString()function to display your customized message on catch.

```

class NegativeNumberException extends Exception
{
    public NegativeNumberException (String str)
    {
        // Calling constructor of parent Exception
        super(str);
    }
}
public class TestUser
{
    public static void main(String args[])
    {
        try
        {
            int n = Integer.parseInt(args[0]);
            if (n<0)
            {
                // throw an object of user defined exception
                throw new NegativeNumberException (n+" is negative");
            }
        }
        catch (NegativeNumberException e)
        {
            System.out.println("Caught the exception");
            System.out.println(e.getMessage());
        }
    }
}

```

Java I/O

Java I/O (Input and Output) is used *to process the input and produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

- 1) System.out: standard output stream
- 2) System.in: standard input stream
- 3) System.err: standard error stream

String class

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. This class present in the package java.lang which contains two string classes:

- a) String class
- b) StringBuffer class

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Method	Description
char charAt(int index)	It returns char value for the particular index
int length()	It returns string length
static String format(String format, Object... args)	It returns a formatted string.
static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
String substring(int beginIndex)	It returns substring for given begin index.
String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
boolean equals(Object another)	It checks the equality of string with the given object.
boolean isEmpty()	It checks if string is empty.
String concat(String str)	It concatenates the specified string.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
String[] split(String regex)	It returns a split string matching regex.
String[] split(String regex, int limit)	It returns a split string matching regex and limit.
String intern()	It returns an interned string.
int indexOf(int ch)	It returns the specified char value index.
int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
int indexOf(String substring)	It returns the specified substring index.
int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
String toLowerCase()	It returns a string in lowercase.
String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
String toUpperCase()	It returns a string in uppercase.
String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
String trim()	It removes beginning and ending spaces of this string.
static String valueOf(int value)	It converts given type into string. It is an overloaded method.

StringBuffer Class

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

StringBuffer Methods

Method	Description
StringBuffer append(String str)	appends the specified string to this character sequence.
int capacity()	returns the current capacity.
char charAt(int index)	This method returns the char value in this sequence at the specified index.
StringBuffer delete(int start, int end)	This method removes the characters in a substring of this sequence.
StringBuffer deleteCharAt(int index)	This method removes the char at the specified position in this sequence
void ensureCapacity(int minimumCapacity)	This method ensures that the capacity is at least equal to the specified minimum.
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	This method characters are copied from this sequence into the destination character array dst.
int indexOf(String str)	This method returns the index within this string of the first occurrence of the specified substring.
int indexOf(String str, int fromIndex)	This method returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
StringBuffer insert(int offset, int i)	This method inserts the string representation of the second int argument into this sequence.
int lastIndexOf(String str)	This method returns the index within this string of the rightmost occurrence of the specified substring.
int lastIndexOf(String str, int fromIndex)	This method returns the index within this string of the last occurrence of the specified substring.
int length()	This method returns the length (character count).
StringBuffer replace(int start, int end, String str)	This method replaces the characters in a substring of this sequence with characters in the specified String.
StringBuffer reverse()	This method causes this character sequence to be replaced by the reverse of the sequence.
void setCharAt(int index, char ch)	The character at the specified index is set to ch.

void setLength(int newLength)	This method sets the length of the character sequence.
String substring(int start)	This method returns a new String that contains a subsequence of characters currently contained in this character sequence
String substring(int start, int end)	This method returns a new String that contains a subsequence of characters currently contained in this sequence.
void trimToSize()	This method attempts to reduce storage used for the character sequence.

Java File Class

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Constructor	Description
File(File parent, String child)	It creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	It creates a new File instance by converting the given pathname string into an abstract pathname.
File(String parent, String child)	It creates a new File instance from a parent pathname string and a child pathname string.
File(URI uri)	It creates a new File instance by converting the given file: URI into an abstract pathname.

File class methods

Method	Description
boolean canWrite()	It tests whether the application can modify the file denoted by this abstract pathname.String[]
boolean canExecute()	It tests whether the application can execute the file denoted by this abstract pathname.
boolean canRead()	It tests whether the application can read the file denoted by this abstract pathname.
boolean isAbsolute()	It tests whether this abstract pathname is absolute.
boolean isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String getName()	It returns the name of the file or directory denoted by this abstract pathname.
String getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
Path toPath()	It returns a java.nio.file.Path object constructed from the this abstract path.
URI toURI()	It constructs a file: URI that represents this abstract pathname.

File[] listFiles()	It returns an <u>array</u> of abstract pathnames denoting the files in the directory denoted by this abstract pathname
long getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name.
String[] list(FilenameFilter filter)	It returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
boolean mkdir()	It creates the directory named by this abstract pathname.
boolean createNewFile()	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

Java FileReader Class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

It is character-oriented class which is used for file handling in java.

Constructor	Description
FileReader(String file)	It gets filename in <u>string</u> . It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in <u>file</u> instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods of FileReader class

Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in <u>string</u> .
FileWriter(File file)	Creates a new file. It gets file name in File <u>object</u> .

Methods of FileWriter class

Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

Java DataInputStream Class

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataInputStream class Methods

Method	Description
int read(byte[] b)	to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	to read len bytes of data from the input stream.
int readInt()	to read input bytes and return an int value.
byte readByte()	to read and return the one input byte.
char readChar()	to read two input bytes and returns a char value.
double readDouble()	to read eight input bytes and returns a double value.
boolean readBoolean()	to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	to skip over x bytes of data from the input stream.
String readUTF()	to read a <u>string</u> that has been encoded using the UTF-8 format.
void readFully(byte[] b)	to read bytes from the input stream and store them into the <u>buffer array</u> .
void readFully(byte[] b, int off, int len)	to read len bytes from the input stream.

Java DataOutputStream Class

Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.

Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataOutputStream class methods

Method	Description
int size()	It is used to return the number of bytes written to the data output stream.

<code>void write(int b)</code>	It is used to write the specified byte to the underlying output stream.
<code>void write(byte[] b, int off, int len)</code>	It is used to write len bytes of data to the output stream.
<code>void writeBoolean(boolean v)</code>	It is used to write Boolean to the output stream as a 1-byte value.
<code>void writeChar(int v)</code>	It is used to write char to the output stream as a 2-byte value.
<code>void writeChars(String s)</code>	It is used to write string to the output stream as a sequence of characters.
<code>void writeByte(int v)</code>	It is used to write a byte to the output stream as a 1-byte value.
<code>void writeBytes(String s)</code>	It is used to write string to the output stream as a sequence of bytes.
<code>void writeInt(int v)</code>	It is used to write an int to the output stream
<code>void writeShort(int v)</code>	It is used to write a short to the output stream.
<code>void writeShort(int v)</code>	It is used to write a short to the output stream.
<code>void writeLong(long v)</code>	It is used to write a long to the output stream.
<code>void writeUTF(String str)</code>	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
<code>void flush()</code>	It is used to flushes the data output stream.

Lab Assignment

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 a particular element using predefined search method in the Collection framework.
2. Create a Hash table containing Employee name and Salary. Display the details of the hash table. Also search for a specific Employee and display Salary of that Employee.
3. Write a java program to accept a number from the user, if number is zero then throw user defined exception —Number is 0, otherwise check whether no is prime or not.
4. Write a java program that displays the number of characters, lines and words of a file.

SET B

1. Construct a linked List containing names of colours: 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.
2. Write a java program to accept Doctor Name from the user and check whether it is valid or not. (It should not contain digits and special symbol) If it is not valid then throw user defined Exception - Name is Invalid -- otherwise display it.

3. Write a java program to accept details of n customers (c_id, cname, address, mobile_no) from user and store it in a file (Use DataOutputStream class). Display the details of customers by reading it from file.(Use DataInputStream class)

SET C

1. Write a menu driven program to perform the following operations on a set of integers.
 1. Load: This operation should generate 10 random integers (2 digit) and display the number on screen.
 2. Save: The save operation should save the number to a file “number.txt”.
 3. Exit
2. 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 []	1: Incomplete []	2: Late Complete []
3: Needs Improvement []	4: Complete []	5: Well done []

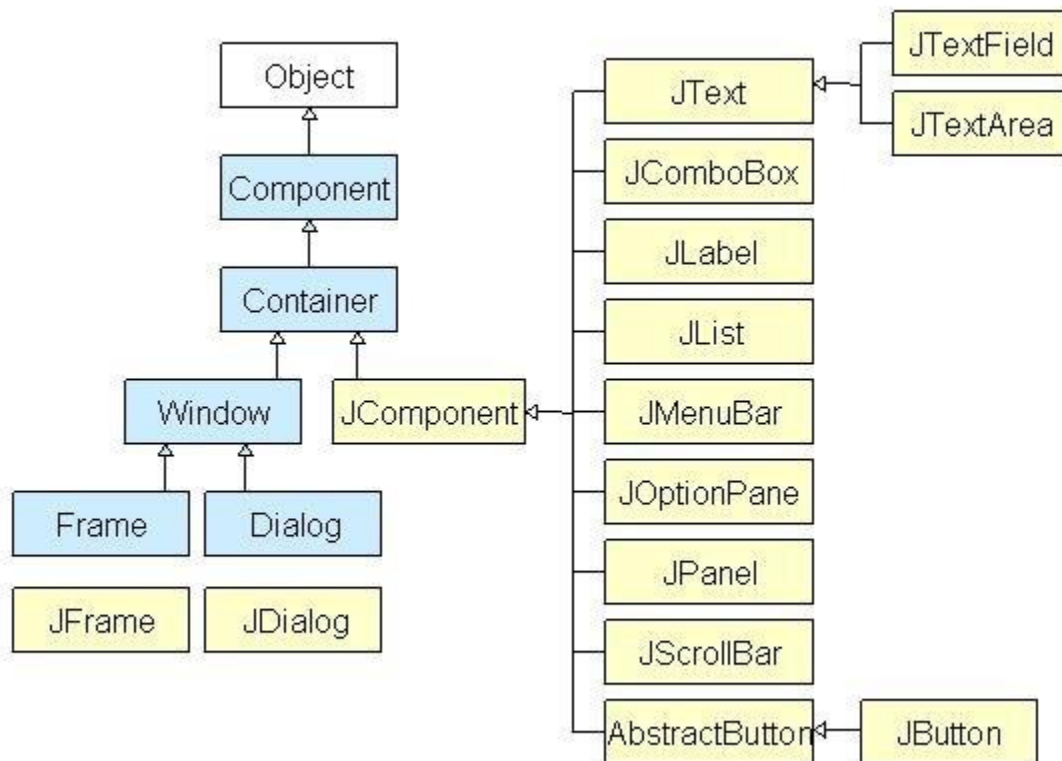
Assignment No.	4	No. of Sessions:	03
Assignment Name :	Swing		

Prerequisite:

- Concept of Inheritance and Interface
- Basics of GUI

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



MVC (Model View Controller) Architecture

Swing API architecture follows loosely based MVC architecture in the following manner.

- One component architecture is MVC - Model-View-Controller.
- The **model** corresponds to the state information associated with the component.
- The **view** determines how the component is displayed on the screen.
- The **controller** determines how the component responds to the user.
- Swing uses a modified version of MVC called "Model-Delegate". In this model the view (look) and controller (feel) are combined into a "delegate".

- Because of the Model-Delegate architecture, the look and feel can be changed without affecting how the component is used in a program.

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.
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.
TextArea	For editing and display of single-attributed textual content.
TextField	For editing and display of single-attributed textual content on a single line.
TextPane	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.

Important Containers

1. JFrame

This is a top-level container which can hold components and containers like panels.

Constructors

JFrame()

JFrame(String title)

Methods

Constructor or Method	Purpose
JFrame()	Creates a frame
JFrame(String title)	Creates a frame with title
setSize(int width, int height)	Specifies size of the frame in pixels
setSize(int width, int height)	Specifies upper left corner
setSize(int width, int height)	Set true to display the frame
setTitle(String title)	Sets the frame title
setDefaultCloseOperation(int mode)	Specifies the operation when frame is closed

Different ways to use this container JFrame:

- i. Create a frame using Swing inside main()
- ii. By creating the object of Frame class (association)
- iii. By extending Frame class (inheritance)

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class SampleTest extends JFrame
{ //inheriting JFrame
    JFrame f;
    SampleTest()
    {
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);
        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String[] args)
    {
        new Simple2();
    }
}
```

JPanel

This is a middle-level container which can hold components and can be added to other containers like frame and panels. The main task of JPanel is to organize components, various layouts can be set in JPanel which provide better organization of components, and however, it does not have a title bar.

Constructors

JPanel(): creates a new panel with a flow layout

JPanel(LayoutManager l): creates a new JPanel with specified layoutManager

JPanel(boolean isDoubleBuffered): creates a new JPanel with a specified buffering strategy

JPanel(LayoutManager l, boolean isDoubleBuffered): creates a new JPanel with specified layoutManager and a specified buffering strategy

Methods

add(Component c): Adds a component to a specified container

setLayout(LayoutManager l): sets the layout of the container to the specified layout manager

updateUI(): resets the UI property with a value from the current look and feel.

setUI(PanelsUI ui): sets the look and feel of an object that renders this component.

getUI(): returns the look and feel object that renders this component.

paramString(): returns a string representation of this JPanel.

getUIClassID(): returns the name of the Look and feel class that renders this component.

getAccessibleContext(): gets the AccessibleContext associated with this JPanel.

// Java Program to Create a Simple JPanel

// and Adding Components to it

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

public class PanelTest extends JFrame
{
    // JButton
    private JButton b, b1, b2;

    // Label to display text
    private JLabel l;

    public PanelTest()
    {
        // Creating a label to display text
        l = new JLabel("panel label");

        // Creating a new buttons
        b = new JButton("button1");
        b1 = new JButton("button2");
        b2 = new JButton("button3");

        // Creating a panel to add buttons
        JPanel p = new JPanel();

        // Adding buttons and textfield to panel
        // using add() method
        p.add(b);
        p.add(b1);
```



```

        p.add(b2);
        p.add(l);

        // setBackground of panel
        p.setBackground(Color.red);

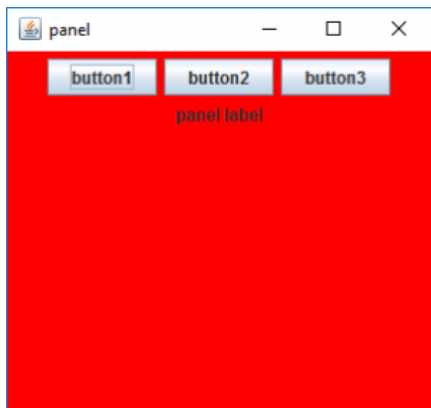
        // Adding panel to frame
        add(p);

        // Setting the size of frame
        setSize(300, 300);

        show();
    }
    public static void main(String[] args)
    {
        PanelTest p = new PanelTest();
    }
}

```

Output:



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 |



Examples:

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

Important Components

1. **JLabel :** 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-

```
void setText(String)
String getText()
void setIcon (Icon)
Icon getIcon()
void setDisabledIcon(Icon)
Icon getDisabledIcon()
void setHorizontalAlignment(int)
void setVerticalAlignment(int)
int getHorizontalAlignment()
int getVerticalAlignment()
```

2. **JButton**: 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()
```

3. JCheckBox**Constructors**

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

Methods

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

4. 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. JComboBox

Constructors-

JComboBox()

Methods

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

6. JList

Constructor

JList(ListModel)

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()

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

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()

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)

Event Handling on components:

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.

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 CheckBoxM enuItemList	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()
TextEvent	TextField TextArea	TextListener	addTextListener()

Sample programs:

1. 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. 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);

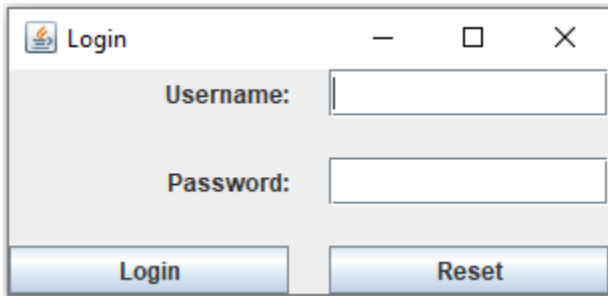
        jtf = new JTextField(35);
        add(jtf);
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void itemStateChanged (ItemEvent ie)
    {
        String text = " "; if(jcb1.isSelected())
            text = text + jcb1.getText()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

```

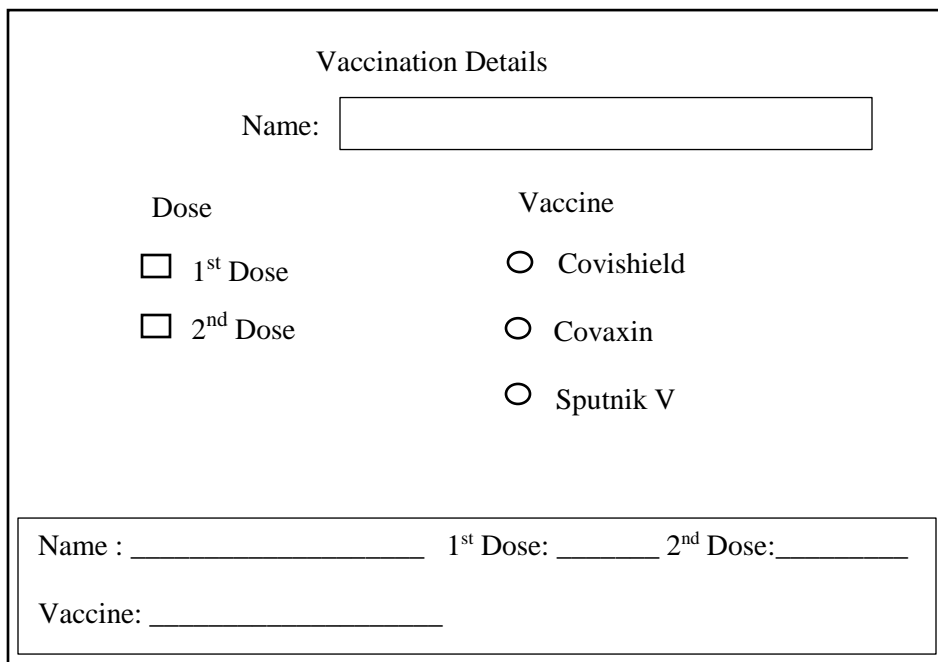

Lab Assignment

SET A

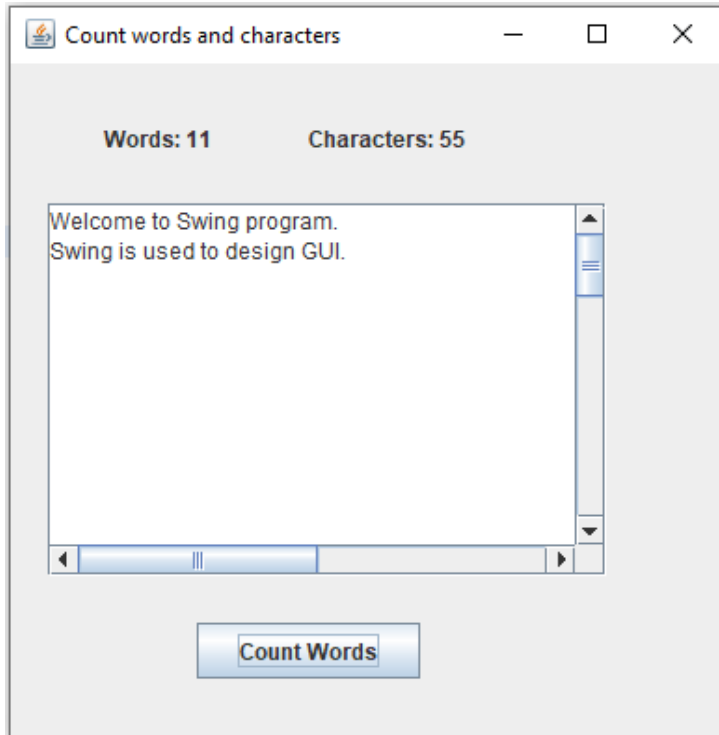
1. Write a java program to design a following GUI. Use appropriate Layout and Components.



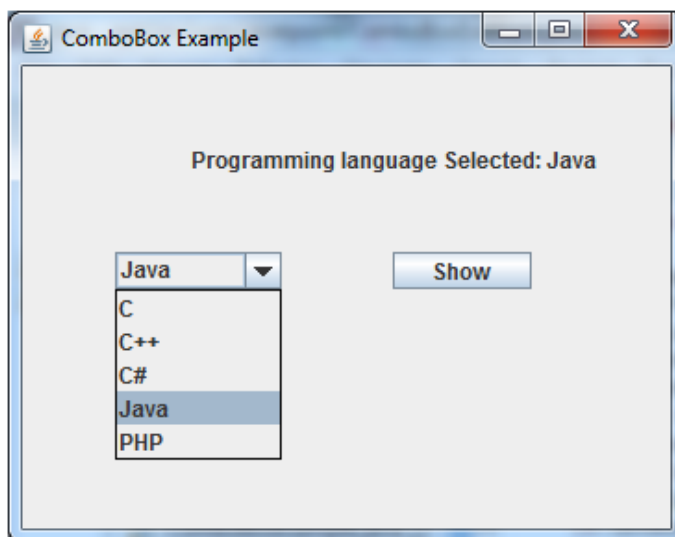
2. Write a java program to design a following GUI. Use appropriate Layout and Components.



3. Write a program to design following GUI using JTextArea. Write a code to display number of words and characters of text in JLabel. Use JScrollPane to get scrollbars for JTextArea.



4. Write a Program to design following GUI by using swing component JComboBox. On click of show button display the selected language on JLabel.



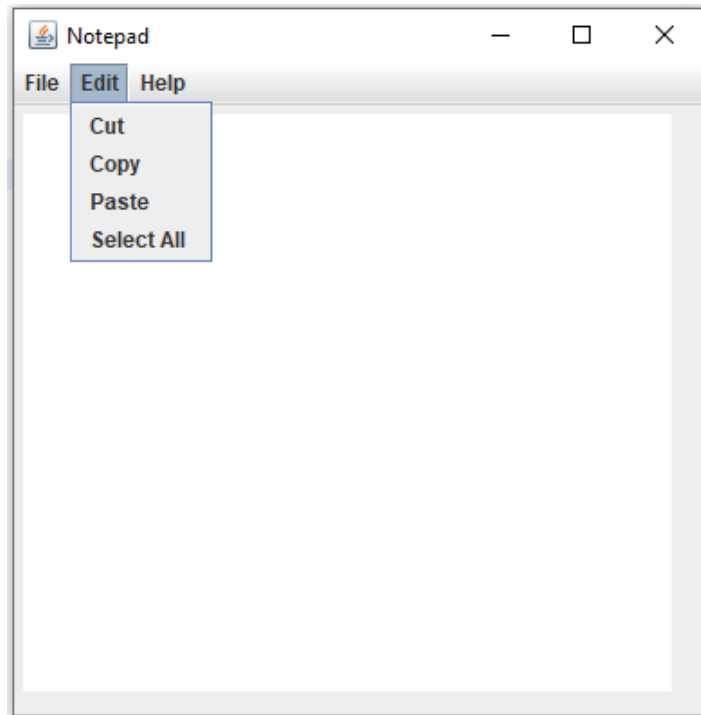
SET B

1. Implement event handling for SET A 1. Verify username and password in 3 attempts. Display dialog box "Login successful" on success or display "Username or Password is in correct". After 3 attempts display "Login Failed". On reset button clear the fields of text box.
2. Implement event handling for SET A 2. Display selected Name, Vaccine. If 1st Dose is taken then write Yes otherwise write No.
3. Write a java program to create the following GUI using Swing components.



SET C

1. Write a program to design and implement following GUI.



Signature of the instructor: _____ **Date:** _____

Assignment Evaluation

0: Not Done []

1: Incomplete []

2: Late Complete []

3: Needs Improvement []

4: Complete []

5: Well done []

Assignment No.	5	No. of Sessions:	03
Assignment Name :	Database Programming		

Prerequisite:

- Basic Concepts of Databases
- Handling database in PostgreSQL

JDBC

Java Database Connectivity (JDBC) API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver-Type 1
2. Native-API driver (partially java driver)-Type 2
3. Network Protocol driver (fully java driver)-Type 3
4. Thin driver (fully java driver)-Type 4

JDBC classes are enclosed in java.sql package. This package contains following set of classes and interfaces.

Classes/interface	Description
java.sql.BLOB	Provide support for BLOB(Binary Large Object) SQL type.
java.sql.Connection	creates a connection with specific database
java.sql.CallableStatement	Execute stored procedures
java.sql.CLOB	Provide support for CLOB(Character Large Object) SQL Type
java.sql.Date	Provide support for Date SQL type.
java.sql.Driver	create an instance of a driver with the DriverManager.
java.sql.DriverManager	This class manages database drivers.

java.sql.PreparedStatement	Used to create and execute parameterized query.
java.sql.ResultSet	It is an interface that provide methods to access the result row-by-row.
java.sql.Savepoint	Specify savepoint in transaction.
java.sql.SQLException	Encapsulate all JDBC related exception.
java.sql.Statement	This interface is used to execute SQL statements.

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

Steps to connect to the database in java:

There are 5 steps to connect any java application with the database in java using JDBC.

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

1. Register the driver Class:

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax

public static void forName(String className)throws ClassNotFoundException

2. Create the connection object:

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax

- public static Connection getConnection(String url)throws SQLException
- public static Connection getConnection(String url,String name,String password)
throws SQLException

Commonly used methods of Connection interface:

createStatement()	Creates a statement object that can be used to execute SQL queries
createStatement(int resultSetType, int resultSetConcurrency) setAutoCommit(boolean status)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency. Is used to set the commit status.By default it is true.
commit()	saves the changes made since the previous

Rollback() Close()	commit/rollback permanent Drops all changes made since the previous commit/rollback. closes the connection and Releases a JDBC resources immediately.
-----------------------	---

3. Create the Statement object:

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database

Syntax

public Statement createStatement()throws SQLException
eg. Statement stmt=con.createStatement();

4. Execute the query:

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax

public ResultSet executeQuery(String sql)throws SQLException
eg. ResultSet rs=stmt.executeQuery("select * from employee");
while(rs.next())
{
 System.out.println(rs.getInt(1)+" "+rs.getString(2));
}

5. Close the connection:

Closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax

public void close()throws SQLException

Classes and Interfaces of JDBC:

1. **DriverManager:** This class will attempt to load the driver classes referenced in the jdbc.drivers system property.
2. **Statement:** Object of this interface is used for firing a static SQL or PL/SQL queries returning the results it has produced.
3. **CallableStatement:** This interface is used to execute SQL stored procedures. The stored procedures are a group of queries with variables in it.

Defined as:

CallableStatement cs=con.prepareCall();

The developer can prepare stored procedure for multiple uses and hence it allows variable declaration also. Executing a SQL statement with the Statement object, and returning a jdbc resultSet.

To execute a query, call an execute method from Statement:

- `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.

4. **ResultSet interface**: In case of select query, it returns set of selected records, a `ResultSet` object maintains cursor pointing to its current row of data.

Methods

Method	Description
<code>boolean next()</code>	is used to move the cursor to the one row next from the current position
<code>boolean previous()</code>	is used to move the cursor to the one row previous from the current position.
<code>boolean first()</code>	is used to move the cursor to the first row in result set object
<code>boolean last()</code>	is used to move the cursor to the last row in result set object.
<code>boolean absolute(int row)</code>	is used to move the cursor to the specified row number in the <code>ResultSet</code> object.
<code>boolean relative(int row)</code>	is used to move the cursor to the relative row number in the <code>ResultSet</code> object, it may be positive or negative.

MetaData:

MetaData is data of data, i.e. we can get further information from the data.

1. **ResultSetMetaData**:

In case of `ResultSet`, metadata of retrieved `ResultSet` called as `ResultSetMetaData`. This interface provides an object that can be used to get information about the types and properties of the columns in a `ResultSet` object.

Methods:

Method	Description
int getColumnCount()	Returns the number of columns available in ResultSet object.
String getColumnType(int col_num)	Returns the SQL type of the column.
getPrecision(int col_num)	Returns the max size for the column
Boolean isAutoIncrement(int col_num)	Checks & returns whether the specified 70 column is set for auto increment or not.
String getTableName(int col_num)	Returns the table name to which the specified column belongs.
boolean isNullable()	Checks and returns whether the specified column is allowed to keep null or not.
boolean isReadOnly()	Checks and returns whether specified column is read only or not.

- 2. DatabaseMetaData:** In case of Database, metadata of database called as DatabaseMetaData. An instance of this interface provides comprehensive information about the database.

Methods:

Method	Description
boolean allProceduresAreCallable()	Retrieves whether all the stored procedures are callable or not
Boolean allTablesAreSelectable()	Retrieves whether the user can fire SELECT query on all connected tables.
int getDatabaseMinorVersion()	Retrieves the minor version of connected database.
int getDatabaseMajorVersion()	Retrieves the major version of connected database.
String getDatabaseProductName()	Retrieves the name of this database product.
String getDatabaseProductVersion()	Retrieves the product version of the connected database

1. Program to insert data into student table.

```
import java.sql.*;
public class Demo1
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("org.postgresql.Driver");
            Connection con = DriverManager.getConnection
            ("jdbc:postgresql://127.0.0.1:5432/dbbca","postgres","123");
            Statement stmt=con.createStatement();
            stmt.executeUpdate("insert into student(rno,name)values(1,'Nandini')");
            stmt.close();
            con.close();
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }
    }
}
```

2. Program to update data from student table.

```
import java.sql.*;
public class Demo2
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("org.postgresql.Driver");
            Connection con=DriverManager.getConnection
            ("jdbc:postgresql://127.0.0.1:5432/dbbca","postgres","123");
            stmt.executeUpdate("update student set rno=5 where name='Rahul'");
            stmt.close();
            con.close();
        }
    }
}
```

```

        catch(Exception ex)
        {
            System.out.println(ex);
        }
    }
}

```

3. Program to delete data from student table

```

import java.sql.*;
public class Demo3
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("org.postgresql.Driver");
            Connection con = DriverManager.getConnection
            ("jdbc:postgresql://127.0.0.1:5432/dbbca","postgres","123");
            Statement stmt=con.createStatement();
            stmt.executeUpdate("delete from student where rno=1");
            stmt.close();
            con.close();
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }
    }
}

```

4. Program to display information of student(rno,name,marks).

```

import java.sql.*;
import java.io.*;
public class Demo4
{
    public static void main(String[] args) throws SQLException
    {
        Class.forName("org.postgresql.Driver");
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
    }
}

```

```

import java.sql.*;
import java.io.*;
public class Demo4
{
    public static void main(String[] args) throws SQLException
    {
        Class.forName("org.postgresql.Driver");
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try
        {
            Connection con=DriverManager.getConnection
            ("jdbc:postgresql://127.0.0.1:5432/dbbca","postgres","123");
            if(conn==null)
                System.out.println("Connection failed ");
            else
            {
                System.out.println("Connection successful");
                stmt = conn.createStatement();
                rs = stmt.executeQuery("Select * from student");
                while(rs.next())
                {
                    System.out.print("RollNo = " + rs.getInt(1));
                    System.out.println("Name = " + rs.getString(2));
                    System.out.println("Marks = " + rs.getInt(3));
                }
                conn.close();
            }
        }
        catch(Exception ex)s
        {
            System.out.println(ex);
        }
    }
}

```

Lab Assignment

SET A

1. Write a JDBC program to display all the details of the Person table in proper format on the screen. Create a Person table with fields as PID, name, gender, birth_year in PostgreSQL. Insert values in Person table.
2. Write a program to display information about the ResultSet like number of columns available in the ResultSet and SQL type of the column. Use Person table. (Use ResultSetMetaData).
3. Write a JDBC program to display all the countries located in West Region. Create a table Country in PostgreSQL with fields (Name, continent, Capital,Region). Insert values in the table.
4. Write a JDBC program to insert the records into the table Employee(ID,name,salary) using PreparedStatement interface. Accept details of Employees from user.

SET B

1. Write a JDBC program to perform search operation on Person table.
 1. Search all the person born in the year 1986.
 2. Search all the females born between 2000- 2005.
2. Write a JDBC program to update number_of_students of “BCA Science” to 1000. Create a table Course (Code,name, department,number_of_students). Insert values in the table.
3. Write a menu driven program to perform the following operations on District(Name, area, population) table.
 1. Insert
 2. Modify
 3. Delete
 4. Search
 5. View All
 6. Exit

SET C

1. Create a table Student with fields roll number, name, percentage using Postgresql. Insert values in the table. Write a JDBC program to display all the details of the student table in a tabular format on the screen. (Using Swing)

Signature of the instructor: _____ **Date:** _____

Assignment Evaluation

0: Not Done []

1: Incomplete []

2: Late Complete []

3: Needs Improvement []

4: Complete []

5: Well done []

Assignment No.	6	No. of Sessions:	03
Assignment Name :	Servlets and JSP		

Prerequisite:

- Designing of HTML

What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

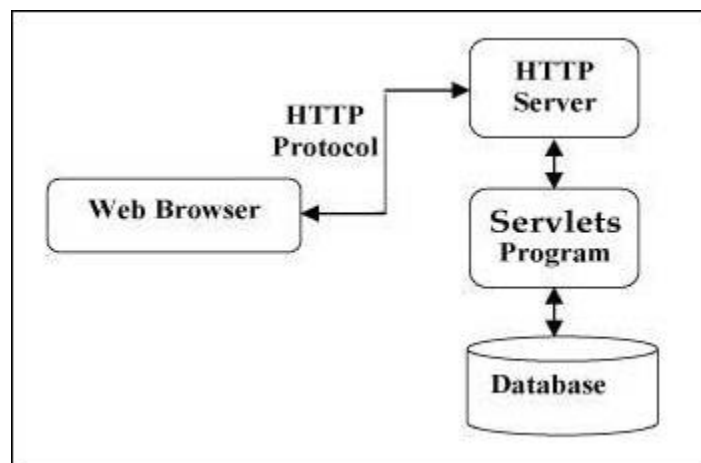
Servlets allows:

- Collect input from users through web page forms
- Present records from a database or another source
- Create web pages dynamically

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 and above of Tomcat supports Java Servlet.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

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.

How to Create a Java Servlet Using Tomcat and Linux

Install and Configure Tomcat on Linux.

Visit <http://tomcat.apache.org/> and download a binary core archive of the latest stable version. To install Tomcat extract the archive at appropriate location.

By default the Tomcat server is using port 8080.

Creating a Servlet

The servlet is a JAVA server application which dynamically process requests and construct responses.

To create a HTTP servlet follow five simple steps:

- **Create Directory Structure**

Enter Tomcat's main directory and navigate to webapps. Create a new directory for your servlet using the mkdir command.

- After this create directory WEB-INF. WEB-INF directory contains configuration information about the web application.
- classes (subdirectory of WEB-INF). The subdirectory class should include the source code of the servlet.

Follow the commands:

```
$ cd webapps/  
$ mkdir ServletTest  
$ cd ServletTest  
$ mkdir WEB-INF  
cd WEB-INF  
$ mkdir classes
```

- **Configuration**

The servlet must have a configuration XML file at directory WEB-INF. Create a text file with name **web.xml**. The file have to uses the XML schema. The name of the root tag have to be web-app and should contain subelements servlet (contains information about name and main class of the servlet) and servlet-mapping (URL pattern to execute the servlet).

Example:

```
<web-app>  
    <servlet>  
        <servlet-name>HelloWorld</servlet-name>  
        <servlet-class>HelloWorld</servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>HelloWorld</servlet-name>  
        <url-pattern>/HelloWorld</url-pattern>  
    </servlet-mapping>  
</web-app>
```

By default the Tomcat uses port 8080 and this servlet can be access with URL:
<http://192.168.100.10:8080/HelloWorld>.

3. Write Servlet Code

The name of the main class must be same as described at web.xml (configured at step2) and should be saved as a file at directory classes (created at step 1). The class should extend the abstract class HttpServlet and to override method doGet. The packages for input/output operations, servlet exception and HTTP request/responses have to be included.

HelloWorld.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out= response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("\t<title>Servlet Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<b>Hello World!</b>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

4. Build the Servlet

Before executing a servlet it has to be build using the source code, j2ee.jar and servlet-api.jar. The Servlet API is part of the Tomcat and can be found at lib (subdirectory of the Tomcat main directory). To build the class HelloWorld execute:

```
$ javac -classpath %J2EE_HOME%\lib\j2ee.jar -classpath
```

```
../../lib/servlet-api.jar HelloWorld.java
```

```
$ On success no warnings or errors will occur.
```

5. Run the Servlet

Start Tomcat server (if not started) and using a web browser open the URL configured at step 2 (example: <http://192.168.100.10:8080/HelloWorld.s>).

```
$ ./startup.sh
```


Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation:

- `getParameter()`: You call `request.getParameter()` method to get the value of a form parameter.
- `getParameterValues()`: Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- `getParameterNames()`: Call this method if you want a complete list of all parameters in the current request

Writing a servlet Program to firstname and lastname from HTML to servlet.

1. Create info.html HTML page under webapps/ServletTest folder

```
<html>
  <title>Login</title>
  <body>
    <form action="NameServlet" method="POST">
      Firstname : <input type="text" name="firstname"> <br/>
      Password : <input type="text" name="lastname"><br/>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

2. Write servlet program under webapps/ServletTest/WEB-INF/classes

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

public class NameServlet extends HttpServlet
{
    public void doPost(HttpServletRequest oRequest,HttpServletResponse
oResponse) throws ServletException,IOException
    {
        String firstname = oRequest.getParameter("firstname");
        String lastname = oRequest.getParameter("lastname");
```

```
        response.setContentType("text/html");
        PrintWriter out= response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("\t<title>Servlet Example</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("\t<b>firstname lastname</b>");
        out.println("</body>");
        out.println("</html>");

    }
}
```

3. Add tag in existing web.xml under webapps/ServletTest/WEB-INF

```
<web-app>
    <servlet>
        <servlet-name>NameServlet</servlet-name>
        <servlet-class> NameServlet </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NameServlet</servlet-name>
        <url-pattern>/NameServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

4. Compile NameServlet.java program to generate NameServlet.class file.

5. Apply URL in browser <http://192.168.100.10:8080/info.html>

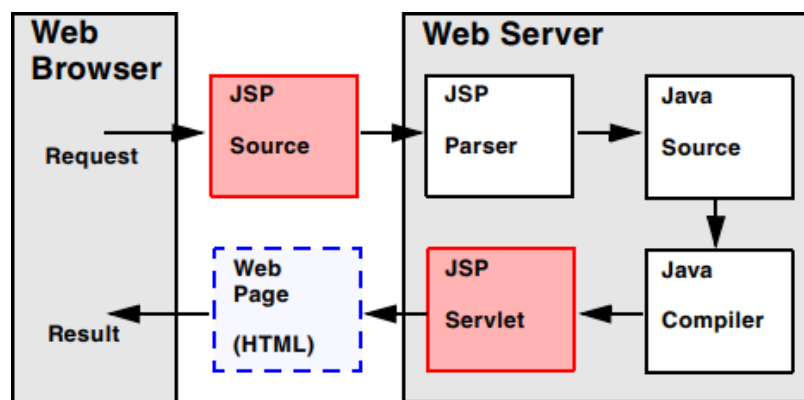
JSP

JavaServer Pages (JSPs) are similar to HTML files, but provide the ability to display dynamic content within Web pages. JSP technology was developed by Sun Microsystems to separate the development of dynamic Web page content from static HTML page design.

How JavaServer Pages work:

- JavaServer Pages are made operable by having their contents (HTML tags, JSP tags and scripts) translated into a servlet by the application server.
- This process is responsible for translating both the dynamic and static elements declared within the JSP file into Java servlet code that delivers the translated contents through the Web server output stream to the browser. Because JSPs are server-side technology, the processing of both the static and dynamic elements of the page occurs in the server.
- The architecture of a JSP/servlet-enabled Web site is often referred to as thin-client because most of the business logic is executed on the server.

The following process outlines the tasks performed on a JSP file on the first invocation of the file or when the underlying JSP file is changed by the developer:



The JSP processing life-cycle on first-time invocation

JSP Scripting elements:

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1. Scriptlet tag

A scriptlet tag is used to execute java source code in JSP.

Syntax:

```
<% java source code %>
```

e.g

Welcome.jsp

```
<html>
```

```
<body>
```

```
<% out.print("welcome to jsp"); %>
```

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

2. Expression tag

The code placed within JSP expression tag is written to the output stream of the response. So you need not write `out.print()` to write data. It is mainly used to print the values of variable or method.

Syntax:

```
<%= statement %>
```

e.g.

Welcome.jsp

```
<html>
  <body>
    <%= "welcome to jsp" %>
  </body>
</html>
```

3. Declaration tag

The JSP declaration tag is used to declare fields and methods.

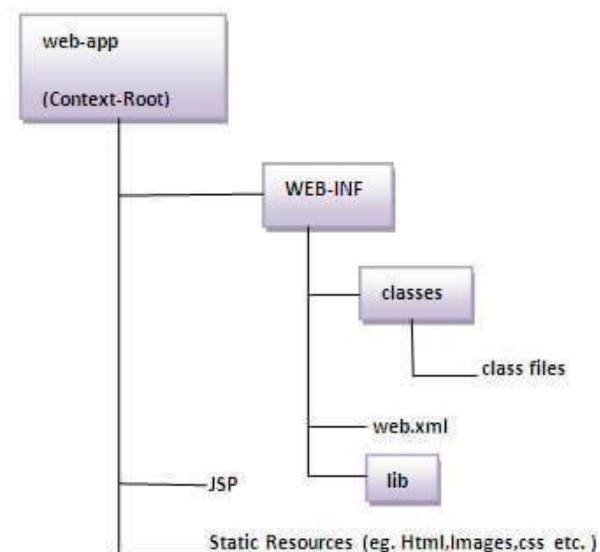
Syntax:

```
<%! field or method declaration %>
```

e.g. Test.jsp

```
<html>
  <body>
    <%! int data=50; %>
    <%= "Value of the variable is:"+data %>
  </body>
</html>
```

Use following path in tomcat to write JSP program.



JSP Program to greet the user

User.html

```
<html>
  <body>
    <form action="Greet.jsp" method="POST">
      Username : <input type="text" name="uname">
      <input type="Submit" value="Submit">
    </form>
  </body>
</html>
```

Greet.jsp

```
<%@ page import="java.util.*" %>
<html>
  <body>
    <%
      String user = request.getParameter("uname");
      out.println("<h1>Welcome : "+user+"</h1>");
      Date d = new Date();
      out.println("Todays date is : "+d);
    %>

    <%= request.getParameter("uname") %>
  </body>
</html>
```

Lab Assignment

SET A

1. Write a servlet program to display current date and time of server.
2. Design a servlet to display “**Welcome IP address of client**” to first time visitor. Display “**Welcome-back IP address of client**” if the user is revisiting the page. (Use Cookies)
(Hint: Use req.getRemoteAddr() to get IP address of client)
3. Create a JSP page to accept a number from an user and display it in words:
Example: 123 – One Two Three.
4. Write a JSP program to perform Arithmetic operations such as Addition, Subtraction, Multiplication and Division. Design a HTML to accept two numbers in text box and radio

buttons to display operations. On submit display result as per the selected operation on next page using JSP.

SET B

1. Design the table User (username, password) using Postgre Database. Design HTML login screen. Accept the user name and password from the user. Write a servlet program to accept the login name and password and validates it from the database you have created. If it is correct then display **Welcome.html** otherwise display **Error.html**.
2. Create a JSP page, which accepts user name in a text box and greets the user according to the time on server side.

Example: If user name is Admin

Output:

If it is morning then display message in **red** color as,
Good morning Admin

Today's date: dd/mm/yyyy format

Current time: hh:mm:ss format

If it is afternoon then display message in **green** color as,
Good afternoon Admin

Today's date : dd/mm/yyyy format

Current time : hh:mm:ss format

If it is evening then display message in **blue** color as,
Good evening Admin

Today's date: dd/mm/yyyy format

Current time: hh:mm:ss format

(Hint: To display date and time use GregorianCalendar and Calendar class)

3. Write a JSP program to display number of times user has visited the page. (Use cookies)

SET C

1. Write a servlet 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 whatever the purchase has been done and print the total. (Use HttpSession)
2. Write a JSP program to display details of products from database in tabular format. Design Product table in database as (pid, pname, qty, price)

Signature of the instructor: _____ Date: _____

Assignment Evaluation

0: Not Done []

1: Incomplete []

2: Late Complete []

3: Needs Improvement []

4: Complete []

5: Well done []