Savitribai Phule Pune University

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

(CBCS 2019 Pattern)

Core Java, MongoDB / Python

CA-506: Lab Book

Student Name: _____

College Name:

Roll No.: **Division:** **Seat No:**

Academic Year:

CERTIFICATE

This is to certify that Mr /Ms _____

Seat Number _____ of T Y B B A (C A) Sem V has successfully completed Laboratory course (Core Java, Mongo DB/ Python in the year _____ He/She has scored _____mark out of 10 (For Lab Book)

Subject Teacher **H.O.D./Coordinator**

Internal Examiner **External Examiner**

Introduction

1. **About the work book:**
   This workbook is intended to be used by T.Y.B.B.A. (C.A.) Semester V students for Core java, MongoDB and Python Practical assignments. This workbook is designed by considering all the practical topics mentioned in syllabus.

2. **The objectives of this workbook are:**
   - Defining the scope of the course
   - To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   - To have continuous assessment of the course and students.
   - Providing ready reference for the students during practical implementation.
   - Provide more options to students so that they can have good practice before facing the examination.
   - Catering to the demand of slow and fast learners and accordingly providing the practical assignments to them.

3. **How to use this workbook:**
   The workbook is divided into two sections. Section I is related to Core Java assignments. Section II is related to MongoDB assignments or Python assignments.

   Section I: Core java is divided into five assignments.
   Section II: MongoDB is divided into five assignments.
   OR
   Section II: Python is divided into eight assignments.

   Students have to perform practical assignments of selected elective subject from Section II.

   Each assignment of all sections has three SETs A, B and C. It is mandatory for students to complete SET A and SET B in lab. It also includes practice set which are expected to be solved by students as home assignments and to be evaluated by subject teachers.

4. **Instructions to the students:**
   Please read the following instructions carefully and follow them during practical.
   - Students are expected to carry this workbook every time they come to the lab for computer practice.
   - Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference and the concepts taught in class.
   - 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 all workbook assignments if needed.
   - 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. Instruction to the Instructors:

Make sure that students should follow above instructions

- Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software
- Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box
- The value should also be entered on assignment completion page of the respective Lab course

6. Instructions to the 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 are as given below

- Operating System – Windows
- Python 3.0
- MongoDB Community Edition
- JDK

Assignment Completion Sheet

Section 1: Core Java

Sr. No.	Assignment Name	Marks (out of 5)	Teacher's Sign
1	Introduction to Java		
2	Classes, Objects and Methods		
3	Inheritance, Package and Collection		
4	File and Exception Handling		
5	Applet, AWT, Event & Swing Programming		
Total (Out of 25)			
Total (Out of 5)			

Instructor Signature:

Section II: MongoDB

Sr. No.	Assignment Name	Marks (out of 5)	Teacher's Sign
1	MongoDB Basics		
2	MongoDB Operators		
3	Update and Delete operation in MongoDB		
4	MongoDB Cursor		
5	MongoDB Index and Aggregation		
Total (Out of 25)			
Total (Out of 5)			

'OR'

Section II: Python

Sr. No.	Assignment Name	Marks (out of 5)	Teacher's Sign
1	Introduction to Basic Python		
2	Working with Strings and List		
3	Working with Tuples, Sets and Dictionaries		
4	Working with Functions, Modules and Packages		
5	Python Classes and Objects		
6	Inheritance		
7	Exception Handling		
8	Python GUI Programming using Tkinter		
Total (Out of 40)			
Total (Out of 5)			

Instructor Signature:

Section – I

Core Java

Assignment No. 1: Introduction to Java

JDK (Java Development Kit) Tools:

Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK), and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development Kit (JDK): The JDK comes with a set of tools that are used for developing and running Java programs. It includes

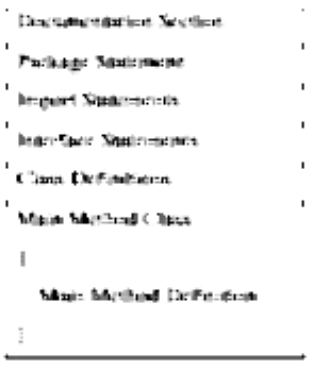
1) **appletviewer**: It is used for viewing the applet.
2) **javac**: It is a Java Compiler.
3) **java**: It is a Java interpreter.
4) **javap**: It is Java Disassembler, which converts byte code into program description.
5) **javah**: It is for gcc C Header Files.
6) **javadoc**: It is for creating HTML documents.
7) **jdb**: It is Java debugger.

Data Types:

Type	Description(Range)
boolean	These have values of either true or false.
byte	8 bit 2's complement integer with values between -27 and 27-1 (-128 to 127)
short	16 bit 2's complement integer with values between -215 and 215-1 (-32,768 to 32,767)
char	16 bit Unicode characters. Chars are the same as ASCII with the high byte set to 0. The numerical values are unsigned 16 bit values between 0 and 65535.
int	32 bit 2's complement integer with values between -231 and 231-1 (-2,147,483,648 to 2,147,483,647).
long	64 bit 2's complement integer with values between -263 and 263-1 (-9223372036854775808 to 9223372036854775807).
Float	32 bit single precision floating point numbers using the IEEE 754-1985 standard (+/- about 10^38).
double	64 bit double precision floating point numbers using the IEEE 754-1985 standard (+/- about 10^308).

Structure of Java Program:

A Java program may contain many classes of which only one class defines a main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements.



Command Line Arguments:

In Java, The command line arguments allow the programmers to pass the arguments during the execution of a program. The users can pass the arguments during the execution by passing the command line arguments inside the main () method.

For Example:

```
class Commandline
{
        public static void main(String args[ ])
        {
                System.out.println("The command line arguments are:" );
                for (int i = 0; i < args.length; i++)
                        System.out.println( args[ i ]);
        }
}
```

Steps to run the above program

To compile and run a given program on command prompt

1. Save the program as Commandline.java
2. Open the command prompt window and compile the program using
   javac Commandline.java
3. After a successful compilation of the program, execute the program using
   java Commandline Arguments list

For example: java Commandline JavaDemo Hello
Press Enter and you will get the desired output.
Output: Hello

Java Array

Array is a collection of similar type of elements that have contiguous memory location. Java array is an object that contains elements of similar data type. Only fixed set of elements can be stored in a java array.

Syntax

There are two sections in the syntax of an array:

Declaration

```
dataType[] arr_list;
dataType [ ]arr_list;
dataType arr[ ];
```

Instantiation

```
arrayRefVar=new datatype[size];
```

Use of length property of an array:

To calculate size of an array

Size=arrayname.length;

For Example: Java Program for demonstration of an array using command line arguments

```
class Demo
{
public static void main (String args [])
{
   int i,n;
   n=args.length;
   int a[]=new int[n];
   for(i=0;i<n;i++)
   {
      a[i]=Integer.parseInt(args[i]);
   }
   //printing array
    for (int i=0;i<n;i++)
      System.out.println (a[i]);
}}
```

String

The set of characters are collectively called String. It is Wrapper class. Anything, if we declare with String class, java compiler considered it as an object. It is immutable.

The List of Functions of String

Sr. No	Methods with Description
1	**char charAt(int index)** Returns the character at the specified index
2	**int compareTo(Object o)** Compares this String to another Object

1	**int compareTo(String anotherString)** Compares two strings lexicographically.
4	**int compareToIgnoreCase(String str)** Compares two strings lexicographically, ignoring case differences.
5	**String concat(String str)** Concatenates the specified string to the end of this string.
6	**boolean contentEquals(StringBuffer sb)** Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	**static String copyValueOf(char[] data)** Returns a String that represents the character sequence in the array specified.
8	**static String copyValueOf(char[] data, int offset, int count)** Returns a String that represents the character sequence in the array specified.
9	**boolean endsWith(String suffix)** Tests if this string ends with the specified suffix.
10	**boolean equals(Object anObject)** Compares this string to the specified object.
11	**boolean equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.
12	**byte getBytes()** Translates this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	**byte[] getBytes(String charsetName)** Translates this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Copies characters from this string into the destination character array.
15	**int hashCode()** Returns a hash code for this string.
16	**int indexOf(int ch)** Returns the index within this string of the first occurrence of the specified character.
17	**int indexOf(int ch, int fromIndex)** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	**int indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
19	**int indexOf(String str, int fromIndex)** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	**String intern()** Returns a canonical representation for the string object.
21	**int lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.
22	**int lastIndexOf(int ch, int fromIndex)**

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index

23 **int lastIndexOf(String str)**

Returns the index within this string of the rightmost occurrence of the specified substring

24 **int lastIndexOf(String str, int fromIndex)**

Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index

---

25 **int length()**

Returns the length of this string

26 **boolean matches(String regex)**

Tells whether or not this string matches the given regular expression.

---

27 **boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)**

Tests if two string regions are equal.

28 **boolean regionMatches(int toffset, String other, int ooffset, int len)**

Tests if two string regions are equal.

29 **String replace(char oldChar, char newChar)**

Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar

---

30 **String replaceAll(String regex, String replacement)**

Replaces each substring of this string that matches the given regular expression with the given replacement.

31 **String replaceFirst(String regex, String replacement)**

Replaces the first substring of this string that matches the given regular expression with the given replacement.

32 **String[] split(String regex)**

Splits this string around matches of the given regular expression.

---

33 **String[] split(String regex, int limit)**

Splits this string around matches of the given regular expression.

34 **boolean startsWith(String prefix)**

Tests if this string starts with the specified prefix

---

35 **boolean startsWith(String prefix, int toffset)**

Tests if this string starts with the specified prefix beginning a specified index

36 **CharSequence subSequence(int beginIndex, int endIndex)**

Returns a new character sequence that is a subsequence of this sequence.

37 **String substring(int beginIndex)**

Returns a new string that is a substring of this string.

38 **String substring(int beginIndex, int endIndex)**

Returns a new string that is a substring of this string.

39 **char[] toCharArray()**

Converts this string to a new character array.

40 **String toLowerCase()**

Converts all of the characters in this String to lower case using the rules of the default locale.

43	**String toLowerCase(Locale locale)** Converts all of the characters in this String to lower case using the rules of the given Locale.
44	**String toString()** This object (which is already a string!) is itself returned.
45	**String toUpperCase()** Converts all of the characters in this String to upper case using the rules of the default locale.
46	**String toUpperCase(Locale locale)** Converts all of the characters in this String to upper case using the rules of the given Locale.
47	**String trim()** Returns a copy of the string, with leading and trailing whitespace omitted.
48	**static String valueOf(primitive data type x)** Returns the string representation of the passed data type argument.

Example: Java Program to display the files having extension .java (Use Command Arguments).

```
class FileDisp
{
    public static void main(String args[])
    {
        int i=0;
        for(;i<args.length;i++)
        {
            if(args[i].endsWith(".java"))
            {
                System.out.println(args[i]);
            }
        }
    }
}
```

Built In Packages

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are as follows:

1) **Language Support Package:**
   A collection of classes and methods required for implementing basic features of Java.
2) **Utilities Package:**
   It is a collection of classes to provide utility functions such as date and time functions.
3) **Input / Output Package:**
   It is a collection of classes required for input/output manipulations.
4) **Networking Package:**
   It is a collection of classes for communicating with other computers via Internet.
5) **AWT Package:**

It is the Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface.

5) **Applet Package**

This package is set of classes that is used to create Java applets.

For adding the packages in an application, import statement is used.

Syntax

```
import package_Name;
```

Example 1: Java program to accept the data and display it.(User Defined Class)

```
import java.util.*;
class Emp
{
  public static void main(String args[])
  {
    int e;
    String en;
    float s;
    Scanner obj=new Scanner(System.in);
    System.out.println("Enter Employee and Salary");
    e=obj.nextInt();
    en=obj.next();
    s=obj.nextFloat();
    System.out.println("Emp No is " + e);
    System.out.println("Emp Name:" + en);
    System.out.println("Salary is " + s);
  }
}
```

Example 2: Java Program to display date and time of a system.

```
import java.io.*;
class Date_Time
{
public static void main(String args[])
{
    Date d=new Date();
    System.out.println("Date and Time of a System is:" + d);
}
}
```

Practice Set

1. Write a java Program to check whether given String is Palindrome or not
2. Write a Java program which accepts three integer values and prints the maximum and minimum.
3. Write a Java program to accept a number from command prompt and generate multiplication table of a number
4. Write a Java program to display 5 Fibonacci series
5. Write a Java program to calculate sum of digits of a number
6. Write a Java program to accept a year and check if it is leap year or not
7. Write a Java program to display characters from A to Z using loop
8. Write a Java program to accept two numbers using command line argument and calculate addition, subtraction, multiplication and division
9. Write a java Program to calculate the sum of first and last digit of a number
10. Write a java program to calculate the sum of even numbers from an array

Set A

1. Write a java Program to check whether given number is Prime or Not
2. Write a java Program to display all the perfect numbers between 1 to n
3. Write a java Program to accept employee name from a user and display it in reverse order
4. Write a java program to display all the even numbers from an array (Use Command Line arguments)
5. Write a java program to display the vowels from a given string

Set B

1. Write a java program to accept a city name and display digits if available
2. Write a java program to accept n numbers from a user store only Armstrong numbers in an array and display it
3. Write a java program to search given name into the array. If it is found then display its index otherwise display appropriate message
4. Write a java program to display following pattern

   5

   4 5

   3 4 5

   2 3 4 5

   1 2 3 4 5
5. Write a java program to display following pattern

   1

   0 1

   0 1 0

   1 0 1 0

Set C

1. Write a java program to count the frequency of each character in a given string
2. Write a java program to display each word in reverse order from a string array
3. Write a java program for union of two integer array
4. Write a java program to display characters of given range
5. Write a java program to display alternate characters from a given string

Assignment Evaluation

0 Not Done [ ] 1 Incomplete [ ] 2 Late Complete [ ]
3 Needs Improvement [ ] 4 Complete [ ] 5 Well Done [ ]

Signature of Instructor

Assignment No. 2: Classes, Objects and Methods

Class: Collection of objects is called class.

Syntax to create a Class:

A keyword class is used to create a class

```
class Classname
{
        //Variable declaration
        //Method declaration
}
```

For Example: Number.java

```
class Number
{
        int x = 0;                  //variable declaration
}
```

Object: Objects have states and behaviours. It is defined as an instance of a class. An object contains an address and takes up some space in memory. Classes create objects and objects use methods to communicate between them.

Syntax to create an Object:

An object is created for a class by using the class name and new keyword.

For Example:

ClassName objectName=new ClassName();

Number Obj=new Number();

Instance variables and methods are accessed by using objects.

```
objectname.variableName;   //Accessing variable
objectname.methodName();   // Accessing a class method
```

For Example: Java Program to demonstrate Class & Object

```
class Number
{
        int a,b;
        void disp()
        {
                System.out.println(a * * * b);
        }
}
class CDemo              // Main Class
{
        public static void main(String args[])
        {
```

```
        Number obj=new Number();
        obj.a=10;
        obj.b=20;
        obj.disp();
    }
}
```

Constructor:

A constructor in Java is a **special method** that is used to initialize objects of the class. The constructor is called when an object of a class is created. Constructor has same name as the class itself. Constructors have no explicit return type.

Types of Constructor:

a. **Default/No argument constructor**: A constructor that has no parameter is known as default constructor.

b. **Parameterized constructor**: A constructor that takes parameter is known as parameterized constructor.

Constructor Overloading:

Constructor overloading is a technique of having more than one constructor with different parameter lists. The compiler differentiates these constructors depending on the number of parameters in the list and their types.

For Example: Java Program to demonstrate Constructor overloading

```
class Number
{
    int num;
    Number()                //Default Constructor
    {
        num=10;
    }
    Number(int a)           //Parameterized Constructor
    {
        num=a;
    }
    void Display()
    {
            System.out.println("num = "+num);
        }
}
```

```
class CDemo                    // Main Class
{
        public static void main(String args[])
        {
                Number obj1 =new Number();          //invokes Default constructor
                Number obj2=new Number(50);         //invokes Parameterized constructor
                obj1.Display();
                obj2.Display();
        }
}
```

Method Overloading

A class having two or more methods with the same name but different parameters is called as method overloading. It is used to perform similar task but on different input parameters.

For Example: Java Program to implement Method Overloading concept

```
class Overload
   {
        int add(int a, int b)
        {
                int sum=a+b;
                System.out.println("Result= "+sum);
        }
        int add(int a, int b, int c)
        {
                int sum=a+b+c;
                System.out.println("Result= "+sum);

        }
}
class MethodOverloading
{
        public static void main (String args[])
        {
                Overload obj = new Overload();
                obj.add(10,20);
                obj.add(10, 20, 30);
        }
}
```

Recursion:

A method that calls itself is known as a recursive method and this process is known as recursion. In java Recursion is a process in which a method calls itself continuously

```
class RecFibonacci
{
    int fib(int n)
    {
        if(n==0)
            return 0;
        else if(n==1)
            return 1;
        else
            return fib(n-1)+fib(n-2);
    }

    public static void main(String args[])
    {
        int n;
        n=Integer.parseInt(args[0]);
        RecFibonacci R=new RecFibonacci();
        for(int c=0;c<=n;c++)
        {
            System.out.println(R.fib(c));
        }

    }
}
```

Passing Object as Parameter:

When primitive type is passed to a method, it is passed by value. But when an object is passed to a method, then it is known as call by reference. The updation done with in method will be reflected in the object.

Syntax:

```
methodName(Object objy)
```

Returning Objects:

In java, a method can return any type of data, including objects

Syntax:

```
ObjectName methodName()
```

The new operator

The new operator is used in Java to create new objects. It can also be used to create an array object.

Syntax to create object using new operator

```
ClassName object=new ClassName();
```

The Array of Objects

Syntax :

```
Class_name [] objArray;
        (or)
Class_name objArray[];
```

Example:

```
Student[] S;
   (or)
Student S[];
```

Declare and instantiate the array of objects

```
Class obj[]= new Class[array_length];
```

Ex. Student S[] = new student[2];

It will create an array of objects 'S' with 2 elements/object references.

Imp Note that once an array of objects is instantiated, the individual elements of the array of objects need to be created using new

For Example: Java Program to Demonstrate Array of Object

```
import java.util.*;
class Student
{

        int rol;
        String Sname;
        void accept(int rl, String name)
        {

                rol=rl;
                Sname=name;
        }
        void display()
        {

                System.out.println("Rol - " +rol);
                System.out.println("Sname- " + Sname);
        }
}
class ArrayObject1 {

  public static void main(String args[])
```

```
{
        Scanner S=new Scanner(System.in);
        Student S1[]=new Student[3];

        for(int x=0; x< 3; x++)
        {
                S1[x]=new Student();
                System.out.println("Enter x");
                S1[x].x=S.nextInt();
                System.out.println("Enter Sname");
                S1[x].Sname=S.next();
                S1[x].accept(S1[x].x, S1[x].Sname);
        }
        for(int x=0; x< 3; x++)
        {
                S1[x].display();
        }
}
```

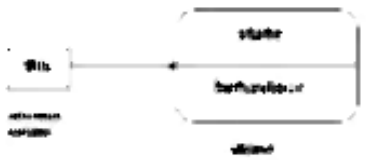
this keyword:

In java, this is a **reference variable** that refers to the current object. If there are many objects and you don't know which object is currently active then this keyword is used to refer current object.

Syntax:

```
this.varName;
```

where, varName is the name of an instance variable



For Example: Java Program to demonstrate this keyword.

```
public class Number
{
        int x;
        public Number (int x)
        {
                this.x = x;
        }
```

```
    public static void main(String[] args)
    {
        Number Obj = new Number (5);
        System.out.println("Value of x = " + Obj.x);
    }
}
```

Static Keyword:

The static keyword is used for memory management. It is applied with variables, methods, blocks and nested classes. Static keyword belongs to the class than an instance of the class.

The static is used with variable and method.

Static Variable:

Static variable is declared using static keyword. The static variable gets memory only once in the class area at the time of class loading and it is shared by all objects of its class.

Syntax:

```
static datatype varName;
```

For Example: Java Program to demonstrate static variable

```
class Number
{
    static int count=0;

    Demo()
    {
        count++;
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Number N1=new Number ();
        Number N2=new Number ();
        Number N3=new Number ();
    }
}
```

Static Method

Static method is declared using static keyword. It is called using class name. A static method can access static data members and can change the value of it.

For Example: Java Program to get the cubes of a given number using the static method

```
class Number
{
        static int cube(int x)
        {
                return x*x*x;
        }

        public static void main(String args[])
        {
                int result;
                result = Number.cube(5);
                System.out.println(result);
        }
}
```

finalize() method:

finalize() is the method of Object class(java.lang.Object). This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks. finalize() method informs system resources before the garbage collector runs for a specific object. JVM allows finalize() to be invoked only once per object.

Syntax:

```
protected void finalize() throws Throwable
```

For Example: Java Program to demonstrate finalize() method

```
public class Number
{
        public static void main(String[] args)
        {
                Number obj = new Number();
                System.out.println(obj.hashCode());
                obj = null;
                System.gc();                //calling garbage collector
                System.out.println("end of garbage collection");
        }
        protected void finalize()
        {
                System.out.println("finalize method called");
        }
}
```

Nested class:

A class defined within another class is called as nested class. A nested class is a member of its Outer class. Nested Inner class can access all the members of outer class including private data members and methods. Outer class does not have access to the members of the nested/inner class

Syntax:

```
class Outer{ //
{
        class Nested{ //
        {



        }
}
```

For Example: Java Program to demonstrate nested class concept

```
class Outer
{
        int x =10;
        class Inner
        {
              int y=20;
              void display()
              {
                System.out.println("x= " + x);
                System.out.println("y= " + y);
              }
        }
        void show()
        {
                Inner I =new Inner();
                System.out.println("Outer Show ");
                I.display();
        }
}
class NestedEx
{
        public static void main(String args[])
        {
                Outer O =new Outer();
                O.show();
        }
}
```

Inner class:

A non static class that is created inside a class but outside a method is called member inner class. Inner class can access the private data members & methods of outer class directly. It is used to group classes and interfaces in one place.

For Example: Java Program to demonstrate Inner class concept

```
class Outer
{
        int x = 30;
        class Inner
        {
                void disp()
                {
                    System.out.println("Inner class " + x);          //x is accessed directly
                }
        }
      void show()
      {
                Inner o=new Inner();
                o.disp();
      }
}
class InnerDemo
{
        public static void main(String args[])
        {
                Outer obj=new Outer();
                obj.show();
                Outer.Inner obj2=new Inner();
                obj2.disp();
        }
}
```

Anonymous Classes:

A class that has no name is known as anonymous inner class. Anonymous class is always inner class. It is defined inside another class. It is always child class of Some Parent class. They can extend a class or implements an interface. It can be used to override method of class or interface.

For Example: Java Program to demonstrate Anonymous class concept

```
class Number
{
        public void display()
```

```
        {
                System.out.println("I am in Number class");
        }
}

class AnonymousDemo
{
        public void create()
        {
                Number N = new Number()
                {
                        public void display()
                        {
                                System.out.println("I am in anonymous class");
                        }
                };
        N.display();
        }
}
class Demo
{
        public static void main(String[] args)
        {
                AnonymousDemo obj = new AnonymousDemo();
                obj.create();
        }
}
```

Practice Set:

1. Write a Java program to for the implementation of reference variable
2. Write a Java program to keep the count of object created of a class. Display the count each time when the object is created.
3. Write a Java program to convert integer primitive data. (use toString())
4. Write a Java program to calculate sum of digits of a number using Recursion
5. Write a Java program to for the implementation of this keyword.

Set A:

1. Write a Java program to calculate power of a number using recursion
2. Write a Java program to display Fibonacci series using function.
3. Write a Java program to calculate area of Circle, Triangle & Rectangle (Use Method Overloading)
4. Write a Java program to Copy data of one object to another Object

5. Write a Java program to calculate factorial of a number using recursion.

Set B:

1. Define a class person(int pid,pname,age,gender). Define Default and parameterized constructor. Override the constructor. Accept the 5 person details and display it.(use this keyword)
2. Define a class product(int pid,pname price). Write a function to accept the product details, to display product details and to calculate total amount. (use array of Objects)
3. Define a class Student(rollno,name,per). Create n objects of the student class and Display it using toString(). (Use parameterized constructor)
4. Define a class MyNumber having one private integer data member. Write a default constructor to initialize it to 0 and another constructor to initialize it to a value. Write methods isNegative, isPositive. Use command line argument to pass a value to the object and perform the above tests.

Set C:

1. Define class Student(rno, name, mark1, mark2). Define Result class(total, percentage) outside the student class. Accept the student details & display the mark sheet with (rno, name, mark1, mark2, total, percentage) (Use inner class concept)
2. Write a java program to accept n employee names from user. Sort them in ascending order and Display them. (Use array of object and Static keyword)
3. Write a java program to accept details of 11 cricket players(pid, pname, totalRuns, inningsPlayed, NotOutInnings). Calculate the average of all the players. Display the details of player having maximum average.
4. Write a java program to accept details of n books. And Display the quantity of given book.

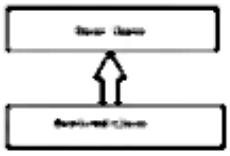
Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 3: Inheritance, Package and Collection

The mechanism of deriving a new class from an old class is called as Inheritance. Old class is called as Superclass or Base class and the new derived class is called as Subclass or Derived class. It is also defined as the process where one class acquires the properties (methods and fields) of another class. The keyword **extends** is used to inherit the properties of a base/super class in derived/sub class.



Syntax:

```
class Superclass
{
        // Superclass data variables
        // Superclass member functions
}
class Subclass extends Superclass
{
        // Subclass data variables
        // Subclass member functions
}
```

Types of inheritance:

A) Single Inheritance:

One subclass is derived from only one superclass is called as single inheritance.

Syntax:

```
class ClassA
{

}
class ClassB extends ClassA
{
```

}

B) Multilevel Inheritance:

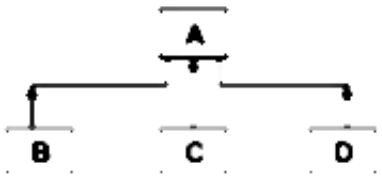
A subclass is derived from another derived class is called as Multilevel inheritance

Syntax:

```
class ClassA
{
}
class ClassB extends ClassA
{
}
class ClassC extends ClassB
{
}
```

C) Hierarchical Inheritance:

More than one classes are derived from a single superclass is called as Hierarchical Inheritance



Syntax:

```
class ClassA
{
}
class ClassB extends ClassA
{
}
class ClassC extends ClassA
{
}
class ClassD extends ClassA
{
}
```

Inheritance in constructor:

In Java, constructor of base class with no argument gets automatically called in derived class constructor. The parameterized constructor of parent class is called explicitly by using

super keyword. Base class constructor call must be the first line in derived class.

For example: Java Program to demonstrate Inheritance in Constructor

```
class Superclass
{
        int x;
        Superclass (int m)
        {
          x = m;
        }
}
class Subclass extends Superclass
{
        int y;
        Subclass (int m, int n)
        {
                super(m);
                y = n;
        }
        void Display()
        {
                System.out.println("x = " + x + " y = " + y);
        }
}
public class Mix
{
        public static void main(String[] args)
        {
                Subclass obj = new Subclass (10, 20);
                obj.Display();
        }
}
```

Method Overriding in Java

The subclass contains the same method name and type as that of the Superclass is called as method overriding. It is used to provide specific implementation of a particular method of superclass. It helps to achieve runtime polymorphism.

For example: Java Program to demonstrate overriding

```
class Superclass
{
        int a;
        Superclass(int x)
        {
                a=x;
        }
        void Display()
        {
                System.out.println("a= " +a);
        }
}
class Subclass extends Superclass
{
        int b;
        Subclass(int x, int y)
        {
                super(x);
                b = y;
        }
        void Display()
        {
                System.out.println("a= " + a + "b= " + b);
        }
}
class InheritanceOverDemo
{
        public static void main(String arg[])
        {
                Subclass obj = new Subclass(10, 20);
                obj.Display();
        }
}
```

Use of super

Super keyword is used by subclass to refer its immediate superclass.

Super keyword is used

1. **To invoke the superclass variables:** To access the data members of super class when both superclass and subclass contains member with same name.

   Syntax: super.variableName;

   For example: Java Program to invoke the superclass variable

```
class SuperClass
{
        int x = 100;
}
```

```
class SubClass extends SuperClass
{
        int x=200;
        void display()
        {
                System.out.println("Super Class x = " + super.x);
                System.out.println("Sub Class x = " + x);
        }
}
class SuperVariable
{
        public static void main(String args[])
        {
                SubClass S2= new SubClass();
                S2.display();
        }
}
```

2. **To invoke the superclass methods**: To access the method of superclass when subclass has overridden that method.
   Syntax: `super.methodName(arguments);`

   For example: Java Program to invoke the superclass method.

```
class SuperClass
{
        int x=100;
        void display()
        {
                System.out.println("Super Class x = " + x);
        }
}
class SubClass extends SuperClass
{
        int x=200;
        void display()
        {
                super.display();
                System.out.println("Sub Class x = " + x);
        }
}
```

```
class SuperMethod
{
    public static void main(String args[])
    {
        Subclass S2= new Subclass();
        S2.display();
    }
}
```

1. **To invoke the superclass constructor**: To explicitly call the constructor of superclass
   Syntax: super constructorName(arguments);

   For example: Java Program to invoke the superclass constructor

```
class Superclass
{
    int a;
    Superclass(int x)
    {
        a=x;
    }
    void Display()
    {
        System.out.println("a= " +a);
    }
}
class Subclass extends Superclass
{
    int b;
    Subclass(int x, int y)
    {
        super(x);
        b = y;
    }
    void Display()
    {
        System.out.println("a= " + a + "b= " + b);
    }
}
class SuperConstructor
{
    public static void main(String arg[])
    {
```

```
        Subclass obj = new Subclass(10, 20);
        obj.Display();
    }
}
```

final keyword:

In java final keyword is used with

1. Variable: If any variable is declared as final, the value of final variable cannot be changed throughout the program. It works as constant.
   For Example: final int SIZE = 10;
2. Method: If any method is declared final, it cannot be overridden in subclass.
   For Example: final void Display() { }
3. Class: If any class is declared as final, it cannot be extended/inherited.
   For Example: final class SuperClass { }

For example: Java Program to demonstrate final variable

```
class Test1VarDemo
{
 public static void main(String args[])
 {
        final int x = 10;
        System.out.println("First: x= " + x);
        x=20;          // Error: It will cause an error because final variable cannot be
                       reinitialized
        System.out.println("First: x= " + x);
 }
}
```

For example: Java Program to demonstrate final method.

```
class Superclass
{
        final void display()
        {
                System.out.println("Super Class: Can cannot be overridden");
        }
}
class Subclass extends Superclass
{
        void display1()
        {
                System.out.println("Sub Class");
        }
}
```

```
class FinalMethodDemo
{
        public static void main(String args[])
        {
                Subclass obj=new Subclass();
                obj.display();
                obj.display1();
        }
}
```

For example: Java Program to demonstrate final class

```
final class Superclass
{
        void display()
        {
                System.out.println("This is a final method");
        }
}
class Subclass extends Superclass
{
        void display()
        {
                System.out.println("The final method is overridden");
        }
}
class FinalClass
{
        public static void main(String args[])
        {
                Subclass obj = new Subclass();
                obj.display();
        }
}
```

Output will be:

```
C:\Program Files\Java\jdk1.8.0_221\bin>javac FinalClass.java
FinalClass.java:7: error: cannot inherit from final Superclass
class Subclass extends Superclass
                       ^
1 error
```

Abstract class:

An abstract class is a class in which one or more methods are declared, but not defined; or it contains abstract methods. Abstract class cannot be instantiated. An abstract class can contain variables, methods.

For example: Java Program to demonstrate abstract class

```
abstract class SuperClass
{
        abstract void display();        //abstract method
}

class SubClass extends SuperClass
{
        SubClass ()
        {
                System.out.println("Sub Class Constructor");
        }
        void display()
        {
                System.out.println("SubClass");
        }
}
class AbstractClassDemo
{
        public static void main(String args[])
        {
                SubClass obj=new SubClass();
                obj.display();
        }
}
```

Interface:

An interface looks like a class but it is not a class. It is a collection of only abstract methods and static final variables. Interface cannot be instantiated. An interface does not contain constructor. Interface is used to achieve **Multiple** inheritance & abstraction.

Syntax to define an interface in java:

```
interface InterfaceName
{

        //members declaration
        //method declaration

}
```

Syntax to implement interface:

```
class className implements InterfaceName
{

        body of the class

}
```

"**implements**" keyword is used to inherit interface in interface class.

For example: Java Program to demonstrate interface

```
interface I1
{
        void display();
}

class Demo implements I1
{
        public void display()
        {
                System.out.println("interface Demo    ");
        }
}

class InterfaceDemo
{
        public static void main(String args[])
        {
                Demo D=new Demo();
                D.display();
        }
}
```

Interface inheritance

Interface inheritance is used when a class wants to implement more than one interface. List of interfaces are separated using comma implemented by the class. Multiple inheritance is possible using interfaces but not by using class.

Syntax to implement multiple interfaces:

```
class className implements InterfaceName1 extends (interfaceName2, InterfaceName)
{

        body of the class

}
```

For example: Java Program to demonstrate Interface inheritance

```
interface Interface1
{
        public void Display1();
}

interface Interface2
{
        public void Display2();
}

interface Interface3 extends Interface1,Interface2
{
  public void Display3();
}

public class InfiInheritanceDemo implements Interface3
{
        public void Display1()
        {
                System.out.println("Display1");
        }
        public void Display2()
        {
                System.out.println("Display2");
        }
        public void Display3()
        {
                System.out.println("Display3");
        }
        public static void main(String args[])
        {
                Interface3 obj = new InfiInheritanceDemo();
                obj.Display1();
                obj.Display2();
                obj.Display3();
        }
}
```

Dynamic method dispatch:

Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at run time instead of compile time. This is an important concept because of how Java implements run time polymorphism.

Java uses the principle of "a superclass reference variable can refer to a subclass object" to resolve calls to overridden methods at run time. When a superclass reference is used to call an overridden method, Java determines which version of the method to execute based on the type of the object being referred to at the time call.

For Example: Java Program to demonstrate Dynamic Method Dispatch

```
class A
{
        void Display()
        {
                System.out.println("I am in A's Display method");
        }
}

class B extends A
{
        void Display()          // overriding Display()
        {
                System.out.println("I am in B's Display method");
        }
}

class C extends A
{
        void Display()                  // overriding Display()
        {
                System.out.println("I am in C's Display method");
        }
}

class DispatchDemo
{
        public static void main(String args[])
        {
                A a = new A();
                B b = new B();
                C c = new C();

                A ref;                  // obtain a reference of type A
                ref = a;                // ref refers to an A object
                ref.Display();          // calling A's version of Display()

                ref = b;                // now ref refers to a B object
                ref.Display();          // calling B's version of Display()

                ref = c;                // now ref refers to a C object
                ref.Display();          // calling C's version of Display()
        }
}
```

Access Control:

Access modifiers define the scope of the class and its members (data and methods)

	Private	No Modifier	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Packages:

A java package is a group of similar types of classes, interfaces and sub-packages in Java. Package is categorized in two forms

a. User-defined package
b. Predefined package

User-defined package: The package created by user is called user-defined package. To create user-defined package, Java uses a file system directory to store them just like folders on your computer

```
C:\myjava
    mca\bca
        mypackage
            MyPackageClass.java
```

First we create a directory mypackage (name should be same as the name of the package). Then create the MyPackageClass.java inside the directory with the first statement being the package names. To create a package, package keyword is used

Syntax:

```
package packagename;
```

For example:

```
package mypackage;
class MyPackageClass
{
    public static void main(String[] args)
    {
        System.out.println("This is my package");
    }
}
```

Keyword **import** is used to use a class or a package from the library

Syntax:

```
import package_name.Class;      // Import a single class
import package_name.*;          // Import the whole package
```

For Example: Java Program to demonstrate package concept

```
package abc;
public class A
{
        public void disp()
        {
                System.out.println("A Class");
        }
}

package abc;
public class B
{
        public void disp()
        {
                System.out.println("B Class");
        }
}

import abc.*;
class PackageDemo
{
        public static void main(String args[])
        {
                A obj1 =new A();
                obj1.disp();
                B obj2 =new B();
                obj2.disp();
        }
}
```

Predefined packages: Predefined packages in java are developed by Sun Microsystem. It is also called as built-in packages. It contains large number of predefined classes, interfaces, and methods that are used by the programmer to perform any task on the program.

Key points about predefined Packages:

1. Java predefined supports a group of packages that contains a group of classes and interfaces. These classes and interfaces consist of a group of methods.
   For example, Java language contains a package called java.lang which contains string class, StringBuffer class, StringBuilder class, all wrapper classes, runnable interface, etc. String class contains a number of methods such as length(), toUpperCase(), toLowerCase() etc.
2. Java contains 14 predefined packages which are main packages. These 14 predefined packages contain nearly 150 sub-packages that consist of a maximum of 7 thousand classes. These 7 thousand classes contain approx 7 lakhs methods.

Collection:

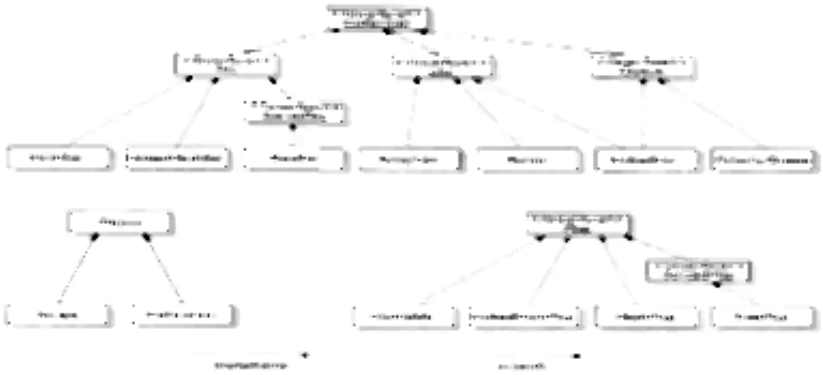
A collection is an object that groups multiple elements into a single unit or single unit of objects. Collections are used to store, retrieve, manipulate, and communicate aggregate data

Collection Framework:

The collection framework is the collection of classes and interfaces

A Java collection framework provides architecture to store and manipulate a group of objects. A Java collection framework includes the following

1. Interfaces
2. Classes
3. Algorithm



The List of interfaces

- Collection
- Set
- List
- Map
- SortedMap
- Enumeration

The List of classes

- ArrayList
- LinkedList
- HashSet
- TreeSet

Algorithm: Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces.

Interfaces: Collection, List, Set

Collection Interface:

The group of data or the set of data which is belong to a unit an object form that unit is called Collection. Collection is an interface present at topmost position in collection framework.

For the implementation of collection interface any class can be used which is at last level as leaf node in collection framework.

For Example:

```
Collection c=new HashSet();
Collection c=new ArrayList();
Collection c=new Vector();
```

Sr. No.	Methods with Description
1.	**boolean add(Object obj)** Adds obj to the invoking collection. Returns true if obj is added to the collection. Returns false if obj is already a member of the collection or if the collection does not allow duplicates.
2.	**boolean addAll(Collection c)** Adds all the elements of c to the invoking collection. Returns true if the operation succeeds (i.e. the elements were added). Otherwise, returns false.
3.	**void clear( )** Removes all elements from the invoking collection.
4.	**boolean contains(Object obj)** Returns true if obj is an element of the invoking collection. Otherwise, returns false.
5.	**boolean containsAll(Collection c)** Returns true if the invoking collection contains all elements of c. Otherwise, returns false.
6.	**boolean equals(Object obj)** Returns true if the invoking collection and obj are equal. Otherwise, returns false.
7.	**int hashCode( )** Returns the hash code for the invoking collection.
8.	**boolean isEmpty( )** Returns true if the invoking collection is empty. Otherwise, returns false.
9.	**Iterator iterator( )** Returns an iterator for the invoking collection.
10.	**boolean remove(Object obj)**

Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.

11. **boolean removeAll(Collection c)**

Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.

12. **boolean retainAll(Collection c)**

Removes all elements from the invoking collection except those in c. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.

13. **int size( )**

Returns the number of elements held in the invoking collection.

14. **Object[ ] toArray( )**

Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.

15. **Object[ ] toArray(Object array[ ])**

Returns an array containing only those collection elements whose type matches that of array.

For Example: Java Program to accept n city names from user and display it

```
import java.util.*;
public class CollDemo
{
    public static void main(String args[])
    {
        int i,n;
        String s;
        Collection c=new ArrayList();
        Scanner obj=new Scanner(System.in);
        System.out.println("How Many");
        n=obj.nextInt();
        for(i=1;i<=n;i++)
        {
            System.out.println("City Name");
            s=obj.next();
            c.add(s);
        }
        System.out.println("Elements Data is " + c);
    }
}
```

List interface:

The List interface extends Collection and describes the behavior of a collection that stores a sequence of elements. Elements can be inserted or accessed by their position in the list, using a zero-based index. A list may contain duplicate elements.

Sr. No.	Methods with Description
1.	**void add(int index, Object obj)** Inserts obj into the invoking list at the index passed in index. Any pre existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
2.	**boolean addAll(int index, Collection c)** Inserts all elements of c into the invoking list at the index passed in index. Any pre existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
3.	**Object get(int index)** Returns the object stored at the specified index within the invoking collection.
4.	**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.
5.	**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.
6.	**ListIterator listIterator( )** Returns an iterator to the start of the invoking list.
7.	**ListIterator listIterator(int index)** Returns an iterator to the invoking list that begins at the specified index.
8.	**Object remove(int index)** Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one.
9.	**Object set(int index, Object obj)** Assigns obj to the location specified by index within the invoking list.
10.	**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.

List interface has been implemented in various classes like ArrayList or LinkedList, etc.

For Example: Java Program to demonstrate List interface

```
import java.util.*;
class ListDemo
```

```
public static void main(String args[])
{
        List l1=new ArrayList();

        l1.add("aa");
        l1.add("bb");
        l1.add("cc");
        l1.add(2,"ee");

        System.out.println("");
        System.out.println("get  - " + l1.get(2));
        System.out.println("IndexOf  - " + l1.indexOf("ee"));
        System.out.println("Last IndexOf  - " + l1.lastIndexOf("ee"));
        l1.remove(2);
        System.out.println("After Remove element at 4" +l1);

        l1.set(1,"xx");
        System.out.println("After replace " +l1);
        System.out.println("Size " + l1.size());
        System.out.println(l1.subList(2,3));
}
}
```

Set interface:

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Sr. No.	Methods with Description
1.	**add( )** Adds an object to the collection
2.	**clear( )** Removes all objects from the collection
3.	**contains( )** Returns true if a specified object is an element within the collection
4.	**isEmpty( )** Returns true if the collection has no elements
5.	**iterator( )** Returns an Iterator object for the collection which may be used to retrieve an object
6.	**remove( )** Removes a specified object from the collection
7.	**size( )** Returns the number of elements in the collection

Set have its implementation in various classes like HashSet, TreeSet, LinkedHashSet.

For Example: Java Program to demonstrate Set Interface

```
import java.util.*;
class SetDemo
{
        public static void main(String[] args)
        {
                Set C1 = new HashSet();
                C1.add(2);
                C1.add(3);
                System.out.println("Set1 " + C1);

                Set C2 = new HashSet();
                C2.add(1);
                C2.add(2);
                System.out.println("Set2 " + C2);

                C2.addAll(C1);
                System.out.println("Union o " + C2);
   }
}
```

Navigation: Enumeration, Iterator, ListIterator

Enumeration:

The enumeration() method of java.util.Collections class is used to return an enumeration over the specified collection. This method takes the collection o as a parameter for which an enumeration is to be returned. This method returns an enumeration over the specified collection.

Syntax:

```
public static Enumeration enumeration(Collection C)
```

For Example: Java program to demonstrate enumeration() method

```
import java.util.*;
public class EnumDemo
{
  public static void main(String[] args) throws Exception
  {
    try
      {
                List<Integer> arrlist = new ArrayList<Integer>();
                arrlist.add(20);
                arrlist.add(30);
                arrlist.add(40);
                System.out.println("List " + arrlist);
                Enumeration<Integer> e = Collections.enumeration(arrlist);
```

```
            System.out.println("\nElements in enumeration:\n");
            while (e.hasMoreElements())
                System.out.println("Value is " + e.nextElement());
        }
        catch (IllegalArgumentException e)
        {
            System.out.println("Exception thrown " + e);
        }
        catch (NoSuchElementException e)
        {
            System.out.println("Exception thrown " + e);
        }
    }
}
```

Iterator interface:

Iterator is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection. java.util package has public interface Iterator and contains following methods:

Sr. No.	Methods with Description
1.	**boolean hasNext()** It returns true if Iterator has more element to iterate.
2.	**Object next()** It returns the next element in the collection until the hasNext() method return true. This method throws 'NoSuchElementException' if there is no next element.
3.	**void remove()** It removes the current element in the collection. This method throws 'IllegalStateException' if this function is called before next() is invoked.

For Example: Java program to accept n employee names through command line and display them.

```
import java.util.*;
class IteratorDemo
{
    public static void main(String args[])
    {
        int i=0;
        String str;
        Integer obj;
        Collection c=new ArrayList();
```

```
        a=args.length;
        for(i=0;i<a;i++)
        {
                l.add(args[i]);
        }
        Iterator itr=l.iterator();
        while(itr.hasNext())
        {
                str=(String)itr.next();
                System.out.println(str);
        }
    }
}
```

ListIterator:

Iterator interface is used to traverse List or Set interface in forward direction only. ListIterator interface traverse List or Set interface in both the directions (backward and forward). ListIterator interface is inherited from Iterator interface.

Sr. No.	Methods with Description
1.	**void add(E e):** Inserts the specified element into the list
2.	**boolean hasNext():** Returns true if this ListIterator has more elements when traversing the list in the forward direction. It returns the next element in the collection until the hasNext() method return true. This method throws NoSuchElementException if there is no next element.
3.	**boolean hasPrevious():** Returns true if this ListIterator has more elements when traversing the list in the reverse direction.
4.	**E next():** Returns the next element in the list and advances the cursor position.
5.	**int nextIndex():** Returns the index of the element that would be returned by a subsequent call to next().
6.	**E previous():** Returns the previous element in the list and moves the cursor position backwards. **int previousIndex():** Returns the index of the element that would be returned by a subsequent call to previous().

7.	**void remove():** Removes from the list the last element that was returned by next() or previous()
8.	**void set(E e):** Replaces the last element returned by next() or previous() with the specified element
9.	**E next():** Returns the next element in the list and advances the cursor position

For Example: Java program to accept N students names from user and display them in reverse order

```
import java.util.*;
class LinkedDemo
{
        public static void main(String args[])
        {
                        int n;
                        String str;
                        Scanner obj=new Scanner(System.in);
                        List l=new LinkedList();
                        System.out.println("How Many");
                        n=obj.nextInt();
                        for(int i=1;i<=n;i++)
                        {
                                System.out.println("Student Name:");
                                str=obj.next();
                                l.add(str);
                        }
                        ListIterator li=l.listIterator();
                        while(li.hasPrevious())
                        {
                                str=(String)li.previous();
                        }
        }
}
```

Classes: LinkedList, ArrayList, Vector, HashSet

LinkedList

The LinkedList class extends AbstractSequentialList and implements the List interface. It provides a linked list data structure

The LinkedList class supports two constructors

- LinkedList( ) : This is an empty linked list

- LinkedList(Collection c) : Builds a linked list that is initialized with the elements of the collection c

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size())
2.	**boolean add(Object o)** Appends the specified element to the end of this list
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null
4.	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null
5.	**void addFirst(Object o)** Inserts the given element at the beginning of this list
6.	**void addLast(Object o)** Appends the given element to the end of this List
7.	**void clear()** Removes all of the elements from this list
8.	**Object clone()** Returns a shallow copy of this LinkedList
9.	**boolean contains(Object o)** Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e))
10.	**Object get(int index)** Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())
11.	**Object getFirst()** Returns the first element in this list. Throws NoSuchElementException if this list is empty
12.	**Object getLast()** Returns the last element in this list. Throws NoSuchElementException if this list is empty
13.	**int indexOf(Object o)** Returns the index in this list of the first occurrence of the specified element,

or -1 if the List does not contain this element

14. **int lastIndexOf(Object o)**
Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element

15. **ListIterator listIterator(int index)**
Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())

16. **Object remove(int index)**
Removes the element at the specified position in this list. Throws NoSuchElementException if this list is empty

17. **boolean remove(Object o)**
Removes the first occurrence of the specified element in this list. Throws NoSuchElementException if this list is empty. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())

18. **Object removeFirst()**
Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty

19. **Object removeLast()**
Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty

20. **Object set(int index, Object element)**
Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())

21. **int size()**
Returns the number of elements in this list

22. **Object[] toArray()**
Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null

23. **Object[] toArray(Object[] o)**
Returns an array containing all of the elements in this list in the correct order, the runtime type of the returned array is that of the specified array

For Example: Java program to demonstrate LinkedList interface

```
import java.util.*;
class LinkedList1Demo
{

    public static void main(String args[])
```

```
        {
                int n;
                String str;
                Scanner obj=new Scanner(System.in);
                List l=new LinkedList();
                System.out.println("How Many");
                n=obj.nextInt();
                for(int i=0;i<n;i++)
                {
                        System.out.println("Enter Data");
                        str=obj.next();
                        l.add(str);
                }
                System.out.println("Elements Data" + l);
        }
}
```

ArrayList:

The ArrayList class extends AbstractList and implements the List interface. ArrayList supports dynamic arrays that can grow as needed. Standard Java arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

The ArrayList class supports three constructors.

- ArrayList() : Builds an empty array list.
- ArrayList(Collection c) : Builds an array list that is initialized with the elements of the collection c.
- ArrayList(int capacity) : Builds an array list that has the specified initial capacity. The capacity is the size of the underlying array that is used to store the elements. The capacity grows automatically as elements are added to an array list.

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).
2.	**boolean add(Object o)** Appends the specified element to the end of this list.
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.

4. **boolean addAll(int index, Collection c)**
Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null

5. **void clear()**
Removes all of the elements from this list

6. **Object clone()**
Returns a shallow copy of this ArrayList

7. **boolean contains(Object o)**
Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e))

8. **void ensureCapacity(int minCapacity)**
Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument

9. **Object get(int index)**
Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())

10. **int indexOf(Object o)**
Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element

11. **int lastIndexOf(Object o)**
Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element

12. **Object remove(int index)**
Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size())

13. **protected void removeRange(int fromIndex, int toIndex)**
Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive

14. **Object set(int index, Object element)**
Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size())

15. **int size()**
Returns the number of elements in this list

16. **Object[] toArray()**
Returns an array containing all of the elements in this list in the correct

	order. Throws NullPointerException if the specified array is null
17.	**Object[] toArray(Object[] a)** Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array
18.	**void trimToSize()** Trims the capacity of this ArrayList instance to be the list's current size

For Example: Java program to demonstrate ArrayList interface using command line arguments

```
import java.util.*;
class ArrayListDemo
{
    public static void main (String args[])
    {
        int n;
        n=args.length;
        ArrayList l=new ArrayList ();
        for (i=0; i<n; i++)
        {
            l.add(args[i]);
        }
        System.out.println (l);
    }
}
```

Vector

Vector is a class that comes in package java.util used to store the data in object form. The generic dynamic array is called Vector. Vector implements a dynamic array. It is similar to ArrayList, but with two differences

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change size over the life time of a program.

Sr. No.	Constructor
1.	**Vector( )** This constructor creates a default vector, which has an initial size of 10.
2.	**Vector(int size)** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
3.	**Vector(int size, int incr)** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.

4. **Vector(Collection c)**
   This constructor creates a vector that contains the elements of collection c

Methods:

Sr. No.	Methods with Description
1.	**void add(int index, Object element)** Inserts the specified element at the specified position in this Vector
2.	**boolean add(Object o)** Appends the specified element to the end of this Vector
3.	**boolean addAll(Collection c)** Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator
4.	**boolean addAll(int index, Collection c)** Inserts all of the elements in the specified Collection into this Vector at the specified position
5.	**void addElement(Object obj)** Adds the specified component to the end of this vector, increasing its size by one
6.	**int capacity()** Returns the current capacity of this vector
7.	**void clear()** Removes all of the elements from this vector
8.	**Object clone()** Returns a clone of this vector
9.	**boolean contains(Object elem)** Tests if the specified object is a component in this vector
10.	**boolean containsAll(Collection c)** Returns true if this vector contains all of the elements in the specified Collection
11.	**void copyInto(Object[] anArray)** Copies the components of this vector into the specified array
12.	**Object elementAt(int index)** Returns the component at the specified index
13.	**Enumeration elements()** Returns an enumeration of the components of this vector

For Example: Java program to display all the files having extension .java

import java.util.*

```
class VectDemo
{
    public static void main(String args[])
    {
        int i,n;
        n=args.length;
        Vector v=new Vector();
        String st[]=new String[n];
        for(i=0;i<n;i++)
        {
            v.addElement(args[i]);
        }
        v.copyInto(st);
        for(i=0;i<n;i++)
        {
            if(st[i].endsWith(".java"))==0)
                {
                    System.out.println(st[i]);
                }
            }
    }
}
```

Map Classes:

Following are the two main classes of the Map Class

A) HashMap:

A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class. It contains only unique elements. It may have one null key and multiple null values. It maintains no order.

For Example: Java program to demonstrate HashMap

```
import java.util.*;
class TestCollection
{
    public static void main(String args[])
    {
        HashMap<Integer,String> hm=new HashMap<Integer,String>();
        hm.put(100,"Amar");
        hm.put(101,"Vijay");
        hm.put(102,"Rahul");
        for(Map.Entry m : hm.entrySet())
```

```
                {
                        System.out.println(m.getKey()+" "+m.getValue());
                }
        }
    }
```

B) TreeMap:

A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class. It contains only unique elements. It cannot have null key but can have multiple null values. It is same as HashMap except it maintains ascending order.

For Example: Java program to demonstrate TreeMap

```
import java.util.*;
class TestCollection
{
  public static void main(String args[])
  {
        TreeMap<Integer,String> hm=new TreeMap<Integer,String>();
        hm.put(100,"Amit");
        hm.put(102,"Ravi");
        hm.put(101,"Vijay");
        hm.put(103,"Rahul");
        Set<Map.Entry> m = hm.entrySet();
        {
                System.out.println(m.getKey()+" "+m.getValue());
        }
    }
}
```

For Example: Java program to demonstrate hashtable

```
import java.util.*;

class HashtableSalary
{
  public static void main(String args[])
  {
        Hashtable<String,String> h=new Hashtable();

        h.put("Amit","10000");
        h.put("Sumit","20000");
        h.put("Akshay","30000");

        Enumeration e;
        e=h.keys();
```

```
while(!  hs.hasMoreElements())
{
        String k=(String)E.nextElement();
        System.out.println(k + "  " + ht.get(k));
}

Scanner S=new Scanner(System.in);
System.out.println("enter Employee name to search  ");
String Ename=S.next();
f=ht.exist();
while(!  hs.hasMoreElements())
{
        String s=(String)E.nextElement();

        if(Ename.equals(s))
        {
                System.out.println(s+" value="+ht.get(s));
                break;
        }
}


}
```

Practice Set:

1. Create abstract class Shape with abstract method area() Write a Java program to calculate are of Rectangle and Triangle (Inherit Shape class in classes Rectangle and Triangle)
2. Create a class Teacher(Tid, Tname, Designation, Salary, Subject) Write a Java program to accept the details of 'n' teachers and display the details of teacher who is teaching Java Subject (Use array of Object)
3. Create a class Doctor(Dno, Dname, Qualification, Specialization) Write a Java program to accept the details of 'n' doctors and display the details of doctor in ascending order by Doctor name
4. Write a Java program to accept 'n' employee names through command line. Store them in vector. Display the name of employees starting with character 'S'
5. Create a package Mathematics with two classes Maximum and Power. Write a java program to accept two numbers from user and perform the following operations on it
    a. Find Maximum of two numbers.
    b. Calculate the power(X^n).

Set A:

1. Write a java program to calculate area of Cylinder and Circle (Use super keyword)

3) Define an Interface Shape with abstract method area() Write a java program to calculate an area of Circle and Sphere (use final keyword)

3. Define an interface "Integer" with a abstract method check() Write a Java program to check whether a given number is Positive or Negative

4. Define a class Student with attributes rollno and name. Define default and parameterized constructor. Override the toString() method. Keep the count of Objects created. Create objects using parameterized constructor and Display the object count after each object is created

5. Write a java program to accept 'n' integers from the user & store them in an ArrayList collection. Display the elements of ArrayList collection in reverse order

Set B

1. Create an abstract class Shape with methods calc_area() & calc_volume(). Derive two classes Sphere(radius) & Cone(radius, height) from it. Calculate area and volume of both. (Use Method Overriding)

2. Define a class Employee having private members of, name, department, salary. Define default & parameterized constructors. Create a subclass called Manager with private member bonus. Define methods accept & display on both the classes. Create n objects of the manager class & display the details of the manager having the maximum total salary(salary+bonus)

3. Construct a Linked List containing names: CPP, Java, Python and PHP. Then extend your program to do the following.
   - Display the contents of the List using an iterator
   - Display the contents of the List in reverse order using a ListIterator

4. Create a hashtable containing employee name & salary. Display the details of the hashtable. Also search for a specific Employee and display salary of that employee

5. Write a package game which will have 2 classes Indoor & Outdoor. Use a function display() to generate the list of players for the specific game. Use default & parameterized constructor

Set C:

1. Create a hashtable containing city name & STD code. Display the details of the hashtable. Also search for a specific city and display STD code of that city

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

3. Define an abstract class Staff with members name &address. Define two sub classes FullTimeStaff(Department, Salary) and PartTimeStaff(numberOfHours, ratePerHour). Define appropriate constructors. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user's choice. Display details of FullTimeStaff and PartTimeStaff

4. Derive a class Square from class Rectangle. Create one more class Circle. Create an interface with only one method called area(). Implement this interface in all classes. Include appropriate data members and constructors in all classes. Write a java program to accept details of Square, Circle & Rectangle and display the area.
5. Create a package named Series having three different classes to print series:
   - i. Fibonacci series
   - ii. Series of numbers
   - iii. Square of numbers

   Write a java program to generate 'n' terms of the above series.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 4 : File and Exception Handling

Exception:

Exceptions are generated when an error condition occur during the execution of a method. It is possible that a statement might throw more than one kind of exception. Exception can be generated by Java runtime system or they can be manually generated by code. Error Handling becomes a necessary while developing an application to account for exceptional situations that may occur during the program execution, such as

a) Run out of memory
b) Resource allocation Error
c) Inability to find a file
d) Problems in Network connectivity

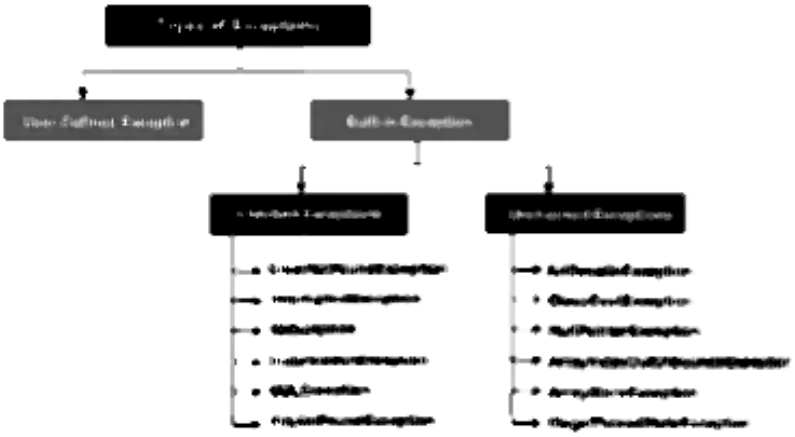
Abnormal condition of a program that terminates its execution is called Exception.

Exception Types

In Java, exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions.

Exceptions can be categorized into two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception
2. **User Defined Exceptions**



Built in Exception

Exceptions that are already available in Java libraries are referred to as built-in exception. These exceptions are able to define the error situation so that we can understand the reason of getting this error. It can be categorized into two broad categories, i.e. checked exceptions and unchecked exception.

Checked Exception

Checked exceptions are called compile time exceptions because these exceptions are checked at compile time by the compiler. The compiler ensures whether the programmer handles the exception or not. The programmer should have to handle the exception, otherwise, the system has shown a compilation error.

Unchecked Exception

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error. Usually, it occurs when the user provides bad data during the interaction with the program.

Exception Handling:

There are two ways to handle an exception:

a) One can try the "risky" code, catch the exception, and do something about it, after which the transmission of the exception come to an end.

b) One can mark that the method throws that exception, in which case the Java runtime engine will throw the exception back to the method.

So, if one uses a method in code that is marked as throwing a particular exception, the compiler will not allow that code unless user handle the exception. If the exception occurs in a try block, the JVM looks to the catch block(s) that follow to see if any of them represented the exception type. The first one that matches will be executed. If none match, then the method ends, and execution jumps to the method that called this one, at the point the call was made.

Using try...catch:

If a method is going to encounter potential exceptions internally, the bits of code that could generate the exceptions is placed inside a try block. There may be other code inside the try block, before and/or after the risky bit(s). Any code that depends upon the risky code's success should be in the try block, since it will automatically be skipped if the exception occurs.

Syntax of try:

```
try
{
    code
    risky/unsafe code
    code that depends on the risky code succeeding
}
```

There is usually at least one catch block immediately after the try block a catch block must specify what type of exception it will catch

Syntax of catch:

```
catch (ExceptionClassName exceptionObjectName)
{
    code using methods from exceptionObjectName
}
```

Example 1: Java Program to count number of valid and invalid Integers.

```
class ExcDemo
{
    public static void main (String args[])
    {
        int i, n, valid=0, invd=0,x;
        n=args.length;
        for(i=0;i<n;i++)
        {
            try
            {
                x=Integer.parseInt (args[i]);
                valid++;
            }
            catch(Exception obj)
            {
                invd++;
            }
        }
        System.out.println ("Valid Integers are " + valid);
        System.out.println ("Invalid Integers are " + invd);
    }
}
```

Example 2: Java program to accept a number from user and calculate the sum of 1st and last digit of that number.

```
import java.io.*;
class SumDemo
{
    public static void main(String args[])
    {
        int n, a, b, s=0;
        try
        {
```

```
                BufferedReader br=new BufferedReader(new InputStreamReader
                                                    (System.in));
                System.out.println("Enter the Number");
                n=Integer.parseInt(br.readLine());
                a=n%10;
                while(n>0)
                {
                        b=n%10;
                        n=n/10;
                }
                        n=n+b;
                        System.out.println("Sum is "+n);
        }
}
```

Finally Block:

To guarantee that a piece of code runs, whether an exception occurs or not, use a finally block after the try and catch blocks. The code in the finally block will almost always execute, even if an unexpected exception occurs, in fact, even if a return statement is encountered.

Syntax:

```
try
{
    Risky code/ unsafe code block
}
catch (ExceptionClassName exceptionObjectName)
{
    Code to resolve problem
}
finally
{
    Code that will always execute
}
```

Example: Java program to demonstrate finally block.

```
class FinDemo
{
    public static void main(String args[])
    {
        int a=5,b=0;
        try
        {
                a=a/b;
```

```
                    System.out.println(c);
                }
                catch ( Exception obj)
                {
                    System.out.println ("Error is " + obj);
                }
                finally
                {
                    System.out.println ("You are in finally Block");
                }
            }
}
```

throw:

It is used to throw a user defined exception. User Defined Exception can be defined by subscribing Exception class to User defined Exception class.

```
class MyException extends Exception {}
```

Syntax:

It can be thrown by using throw keyword.

```
throw ThrowableObj;
```

Example 1: Java program to check whether given name is valid or not.

```
import java.io.*;
class NameNotValidEx extends Exception{}
class ExcDemo
{
    public static void main(String args[])
    {
        int i,n,flag=0;
        String nm;
        char ch;
        try
        {
BufferedReader br=new BufferedReader(new InputStreamReader
                                        (System.in));
          System.out.println ("Enter Your Name");
          nm=br.readLine();
          n=nm.length();
          for(i=0;i<n;i++)
          {
```

```
                    ch=str.charAt(i);
                    if(Character.isLetter(ch))
                    {
                        throw new NameValExc();
                        flag=0;
                    }
                    else
                        flag=1;
                }
                if(flag==1)
                    System.out.println("Name is Valid");
            }
            catch(NameValExc obj)
            {
                System.out.println("Name is Invalid");
            }
        }
    }
```

Example 2: Java program to check given number is valid or not. If it is valid then display its factors, otherwise display appropriate message

```
import java.io.*;
class ZeroNumExc extends Exception{}
class ZeroNumDemo
{
    public static void main(String args[])
    {
        int n;
        try
        {
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter a Number");
            n=Integer.parseInt(br.readLine());
            if(n==0)
            {
                throw new ZeroNumExc();
            }
            else
            {
                for(int i=1;i<=n;i++)
                {
                    if(n%i==0)
```

```
                                        System.out.println (c);
                                }
                        }
                }
                catch (ZeroNumExc obj)
                {
                        System.out.println("Number is Zero");
                }
                catch (Exception ob)
                {
                        System.out.println ("Number is Invalid");
                }
        }
}
```

throws:

The throws statement is used by a method to specify the types of exceptions the method throws. If a method is capable of causing an exception that it does not handle, the method must specify that the exception have to be handled by the calling method. This is done using the throws statement.

Syntax:

```
[ access specifier ] [ modifier ] return type  method name
[ arg list ] [throws exception list ]
```

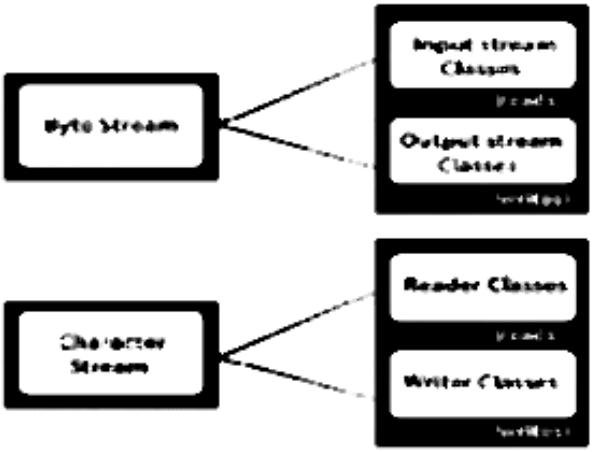
File Handling:

File: It is collection of data in byte form. File is used to store the data in proper way. File handling in Java is defined as reading and writing data to a file. The particular file class from the package called java.io allows us to handle and work with different formats of files.

In Java, a File is an abstract data type. A named location used to store related information is known as a File. There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.

Stream

A series of data is referred to as a stream. In Java, Stream is classified into two types, i.e., Byte Stream and Character Stream.



Brief classification of I/O streams

Byte Stream

Byte Stream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that ByteStream classes read/write the data of 8 bits. We can store video, audio, characters, etc., by using ByteStream classes. These classes are part of the java.io package.

The ByteStream classes are divided into two types of classes, i.e., InputStream and OutputStream. These classes are abstract and the super classes of all the Input/Output stream classes.

The List of Byte Stream classes are

Stream class	Description
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains methods for reading java standard data type.
DataOutputStream	An output stream that contain methods for writing java standard data type.
FileInputStream	Input stream that reads from a file.
FileOutputStream	Output stream that write to a file.
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contains print() and println() method.

Methods of InputStream Classes

Method	Description
read()	This method is abstract in the InputStream class, so it has to be defined in a subclass. This method returns the next byte available as stream int. Once the stream is getting towards an end, the method gives -1. An exception of type IOException will be thrown if an I/O error occurs.
read(byte[] array)	It reads byte from the successive streams to the array. The maximum of array length bytes will be read. This method will not return the data until the stream gets to the end. If the method will reach the end then it will not return the end of bytes.
read(byte[] array, int offset, int length)	It works the same as the previous methods except the length

Methods of OutputStream Classes:

Sr. No.	Method & Description
void close()	This method closes this output stream and releases any system resources associated with this stream
void flush()	This method flushes this output stream and forces any buffered output bytes to be written out
void write(byte[] b)	This method writes b.length bytes from the specified byte array to this output stream
void write(byte[] b, int off, int len)	This method writes len bytes from the specified byte array starting at offset off to this output stream
abstract void write(int b)	This method writes the specified byte to this output stream

Example 1: Java program to display the data from a file. (Use command Line argument)

```
import java.io.*;
class FileRead
{
        public static void main (String args[]) throws IOException
        {
                int b;
                FileInputStream f=new FileInputStream (args[0]);
                while((b=f.read())!= -1)
                   {
                       System.out.print ((char)b);
                   }
                   f.close();
                }
        }
```

Example 2: Java program to copy the data from one file onto another file.

```
import java.io.*;
class FileReadWrite
{
        public static void main (String args[]) throws IOException
        {
                int b;
                FileInputStream f=new FileInputStream (args[0]);
                FileOutputStream fout=new FileOutputStream (args[1]);
                while((b=f.read())!= -1)
                   {
                       fout.write(b);
                   }
                   f.close();
                   fout.close();
                }
        }
```

Character Stream Classes

The java.io package provides CharacterStream classes to overcome the limitations of ByteStream classes, which can only handle the 8 bit bytes and is not compatible to work directly with the Unicode characters. CharacterStream classes are used to work with 16 bit Unicode characters. They can perform operations on characters, char arrays and Strings.

However, the CharacterStream classes are mainly used to read characters from the source and write them to the destination. For this purpose, the CharacterStream classes are divided into two types of classes, i.e., Reader class and Writer class.

The List of Character Stream classes:

Stream class	Description
BufferedReader	Handles buffered input stream
BufferedWriter	Handles buffered output stream
FileReader	Input stream that reads from file
FileWriter	Output stream that writes to file
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte
PrintWriter	Output Stream that contain print() and println() method
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Methods of Reader classes are:

Method	Description
int read()	This method returns the integral representation of the next character present in the input. It returns -1 of the end of the input is encountered.
int read(char buffer[])	This method is used to read from the specified buffer. It returns the total number of characters successfully read. It returns -1 of the end of the input is encountered.
int read(char buffer[], int loc, int noOfChars)	This method is used to read the specified noOfChars from the buffer at the specified location. It returns the total number of characters successfully read.
void mark(int noChars)	This method is used to mark the current position on the input stream until noOfChars characters are read.
void reset()	This method is used to reset the input pointer to the previous set mark.
long skip(long noOfChars)	This method is used to skip the specified noOfChars characters from the input stream and returns the number of characters skipped.
boolean ready()	This method returns a boolean value true, if the next request of input is ready. Otherwise, it returns false.
void close()	This method is used to close the input stream. However, if the program attempts to access the input, it generates IOException.

Methods of Writer classes are:

Method	Description
void write()	This method is used to write the data to the output stream.
void write(int c)	This method is used to write a single character to the output stream.
void write(char buffer[])	This method is used to write the array of characters to the output stream
void write(char buffer [], int loc, int noOfChars)	This method is used to write the noOfChars characters to the character array from the specified location.
void close ()	This method is used to close the output stream. However, this generates the IOException if an attempt is made to write to the output stream after closing the stream
void flush ()	This method is used to flush the output stream and writes the waiting buffered characters.

Example 1: Java program to read the data from a file.

```
import java.io.*;
class FileRead
{
 public static void main(String args[])
 {
  char[] array = new char[100];
  try
  {
      FileReader input = new FileReader("input.txt");
      input.read(array);
      System.out.println("Data in the file:");
      System.out.println(array);
      input.close();
  }
  catch (Exception e)
  {
   e.getStackTrace();
  }
 }
}
```

Example 2 : Java program to write data into the file.

```
import java.io.*;
class FileWrite
{
  public static void main(String args[])
  {
     String data = "This is the data in the output file";
     try
     {
        FileWriter output = new FileWriter("output.txt");
        output.write (data);
        System.out.println("Data is written to the file ");
        output.close ();
     }
     catch (Exception e)
     {
     e.getStackTrace ();
     }
  }
}
```

File class

It is related with characteristics of a file such as name, size, location etc

Syntax:

```
File f=new File ("Path");
```

Methods of File Class

Method	Description
createTempFile(String prefix, String suffix)	It creates an empty file in the default temporary file directory, using the given prefix and suffix to generate its name
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
canWrite()	It tests whether the application can modify the file denoted by this abstract pathname String[]
canExecute()	It tests whether the application can execute the file denoted by this abstract pathname

canRead()	It tests whether the application can read the file denoted by this abstract pathname
isAbsolute()	It tests whether this abstract pathname is absolute
isDirectory()	It tests whether the file denoted by this abstract pathname is a directory
isFile()	It tests whether the file denoted by this abstract pathname is a normal file
getName()	It returns the name of the file or directory denoted by this abstract pathname
getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory
toPath()	It returns a java.nio.file.Path object constructed from the this abstract path
toURI()	It constructs a file URI that represents this abstract pathname
listFiles()	It returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname
getFreeSpace()	It returns the number of unallocated bytes in the partition named by this abstract path name
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
mkdir()	It creates the directory named by this abstract pathname

Example 1: Java Program to create a new File.

```
import java.io.*;
class FileDemo
{
  public static void main(String args[])
  {
    try
    {
      File file = new File("DYP.txt");
      if (file.createNewFile())
      {
```

```
                System.out.println("New File is created!");
            }
            else
            {
                System.out.println("File already exists");
            }
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
    }
}
```

Example 2: Java program to display details of files from a given directory

```
import java.io.*;
class FileDemo
{
    public static void main(String args[])
    {
        File dir=new File("D:\\Sai\\javaio");
        File f[]=dir.listFiles();
        for(File file:files)
        {
            System.out.println(file.getName()+" Can Write "+file.canWrite()+"
            Is Hidden "+file.isHidden()+" Length "+file.length()+" bytes");
        }
    }
}
```

Example 3: Write a java program to delete a given file

```
import java.io.*;
class FileDel
{
    public static void main (String[] args)
    {
        File file = new File("file.txt");
        boolean value = file.delete();
        if(value)
        {
                System.out.println("The File is deleted");
```

```
        }
        else
        {
            System.out.println("The File is not created");
        }
    }
}
```

Example 4: Java program to rename a given file.

```
import java.io.*;
class FileRename
{
    public static void main(String args[])
    {
        File file = new File("oldName");
        try
        {
                file.createNewFile();
        }
        catch(Exception e)
        {
            e.getStackTrace();
        }
            File newFile = new File("newName");
            boolean value = file.renameTo(newFile);
            if(value)
            {
                System.out.println("The name of the file is changed");
            }
            else
            {
                System.out.println("The name cannot be changed");
            }
    }
}
```

Practice Set:

1. Write a java program to accept the data from a user and write it into the file
2. Write a java program to display ASCII values of the characters from a file
3. Write a java program to count number of digits, spaces and characters from a file
4. Write a java program to accept a number from user. If it is non zero then check whether it is Armstrong or not, otherwise throws user defined Exception "Number is Invalid"
5. Write a java program to check whether given file is readable or not
6. Write a java program to display name and size of the given files

Set A:

1. Write a java program to count the number of integers from a given list (Use command line arguments)
2. Write a java program to check whether given candidate is eligible for voting or not. Handle user defined as well as system defined Exceptions
3. Write a java program to calculate the size of a file
4. Write a java program to accept a number from a user, if it is zero then throw user defined Exception "Number is Zero". If it is non numeric then generate an error "Number is Invalid" otherwise check whether it is palindrome or not
5. Write a java program to accept a number from user. If it is greater than 100 then throw user defined exception "Number is out of Range" otherwise do the addition of digits of that number. (Use static keyword)

Set B

1. Write a java program to copy the data from one file into another file, while copying change the case of characters in target file and replace all digits by '*' symbol
2. Write a java program to accept string from a user. Write ASCII values of the characters from a string into the file
3. Write a java program to accept a number from a user, if it less than 5 then throw user defined Exception "Number is small", if it is greater than 10 then throw user defined exception "Number is Greater", otherwise calculate its factorial
4. Write a java program to display contents of a file in reverse order
5. Write a java program to display each word from a file in reverse order

Set C:

1. Write a java program to accept list of file names through command line. Delete the files having extension .txt. Display name, location and size of remaining files
2. Write a java program to display the files having extension .txt from a given directory
3. Write a java program to count number of lines, words and characters from a given file
4. Write a java program to read the characters from a file, if a character is alphabet then reverse its case, if not then display as it is. Display all the output on the Screen. (whether it is Digit or Space)

5. Write a java program to validate PAN number and Mobile Number. If it is invalid then throw user defined Exception "Invalid Data", otherwise display it.

Assignment Evaluation

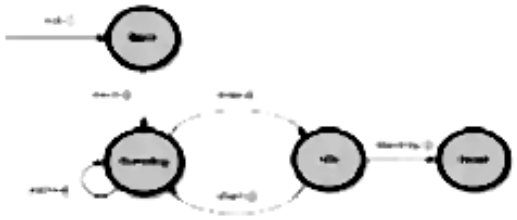
0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 5: Applet, AWT, Event and Swing Programming

Graphical User Interface elements are implemented in two java packages: AWT and Swing. Swing is the newer package and swing classes are based on AWT classes. In this assignment we will learn three important concepts of java Applet, AWT, Event and Swing Programming.

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



Applet Lifecycle

Creating an applet

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

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

Applet methods:

Method	Description	Example

init( )	Automatically called to perform initialization of the applet. Executed only once.	public void init()
		{
		//initialization
		}

start( )	Called every time the applet moves into sight on the Web browser to allow the applet to start up its normal operations.	public void start()
		{
		//Code
		}

stop( )	Called every time the applet moves out of sight on the Web browser to allow the applet to shut off expensive operations.	public void stop()
		{
		//Code
		}

destroy( )	Called when the applet is being unloaded from the page to perform final release of resources when the applet is no longer used	public void destroy()
		{
		// Code
		}

paint()	Called each time the applets output needs to be redrawn	public void
		paint(Graphics g)
		{
		// Display Statements
		}

Running an applet

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

Using appletviewer

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

Using browser

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

The APPLET tag

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

Attribute	Value	Meaning
align	left right top bottom middle baseline	Specifies the alignment of an applet according to surrounding elements
alt	text	Specifies an alternate text for an applet
archive	URL	Specifies the location of an archive file
code	URL	Specifies the file name of a Java applet
codebase	URL	Specifies a relative base URL for applets specified in the code attribute
height	pixels	Specifies the height of an applet
hspace	pixels	Defines the horizontal spacing around an applet
name	name	Defines the name for an applet (to use in scripts)
vspace	pixels	Defines the vertical spacing around an applet
width	pixels	Specifies the width of an applet

The mandatory attributes are CODE, HEIGHT and WIDTH

Examples:

1. <applet code="MyApplet.class" width="200" height="200" archive="files.jar">
   </applet>

2. <applet code="Sample.class" width="100" height="200" codebase="/examples/">
   </applet>

Passing parameters to applets

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

<PARAM NAME = AttributeName VALUE = AttributeValue />

Example:

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

Example: Using getParameter()

```
String dirName = getParameter("ImageSource");
Color c = new Color(Integer.parseInt(getParameter("BackgroundColor")));
```

Sr No	Method	Description
1	public abstract void drawString(String str, int x, int y)	Used to draw specified string.
2	public void drawRect(int x, int y, int width, int height)	Used to draw a rectangle of specified width and height.
3	public abstract void fillRect(int x, int y, int width, int height)	Used to draw a rectangle with a default colour and specified width and height.
4	public abstract void drawOval(int x, int y, int width, int height)	Used to draw oval of specified width and height.
5	public abstract void fillOval(int x, int y, int width, int height)	Used to draw oval with a default colour of specified width and height.
6	public abstract void drawLine(int x1, int y1, int x2, int y2)	Used for drawing a line between the points (x1, y1) and (x2, y2).
7	public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)	Used for drawing a specified image.
8	public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used for drawing a circular arc.
9	public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used for filling circular arc.
10	public abstract void setColor(Color c)	Used to set a colour to the object.
11	public abstract void setFont(Font font)	Used to set font.
12	public void paint(Graphics g)	The paint() method draws the applet.
13	public void repaint()	The repaint() method is used to force redrawing of the applet.
14	public void update()	The update() method redraws only a portion of the applet.
15	String getParameter(String Parameter);	The Applet can retrieve information about the parameters using the getParameter() method.

Example 1: Program to demonstrate Applet Lifecycle.

```
import java.applet.*;
```

```
public class AppletLifeCycle extends Applet
{
        public void init()
        {
                System.out.println("Applet is initialized");
        }
        public void start()
        {
                System.out.println("Applet is being Executed");
        }
        public void stop()
        {
                System.out.println("Applet execution has Stopped");
        }
        public void paint(Graphics g)
        {
                System.out.println("Painting the Applet ");
        }
        public void destroy()
        {
                System.out.println("Applet has been destroyed ");
        }
}

// AppletLifeCycle.html
<html>
<body>
<applet code ="AppletLifeCycle" width=200 height=60>
</applet>
</body>
</html>
```

> Compile the above program using
javac AppletLifeCycle.java

> Execute the applet using
appletviewer AppletLifeCycle.html

Example 2: Java Program to demonstrate sample applet.

```
import java.applet.Applet;
import java.awt.Graphics;
// HelloWorld class extends Applet
```

```
public class HelloWorldApplet extends Applet
{
        // Overriding paint() method
        public void paint (Graphics g)
        {
                g.drawString ("Hello World", 20, 20);
        }
}
```

Example 3: Java Program to show status using applet.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="StatusWindow" width=300 height=50>
</applet>
*/
public class StatusWindow extends Applet
{
        public void init()
        {
                setBackground (Color.cyan);
        }
        public void paint (Graphics g)
        {
                g.drawString ("This is in the applet window", 10, 20);
                showStatus ("This is shown in the status window.");
        }
}
```

Example 4: Sample Program Passing Parameters to Applet.

```
import java.applet.Applet;
import java.awt.Graphics;
/*
<applet code="ParamDemo" width="300" height="300">
<param name=fontName value=Welcome>
</applet>
*/
public class ParamDemo extends Applet
{
        String fontName;
        public void init()
        {
```

```
            fontName = getParameter ("paramter");
            if (fontName == null)
            {
                  fontName = "Welcome to Applet Window";
                  fontName = "Hello " + fontName;
            }
      }
      public void paint (Graphics g)
      {
            g.drawString (fontName, 0, 10);
      }
}
```

Example 5: Program to draw rectangle.

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
      int height, width;
      public void init()
      {
            height = getSize().height;
            width = getSize().width;
            setName("MyApplet");
      }
      public void paint(Graphics g)
      {
            g.drawRoundRect(10, 30, 150, 150, 5, 5);
      }
}
```

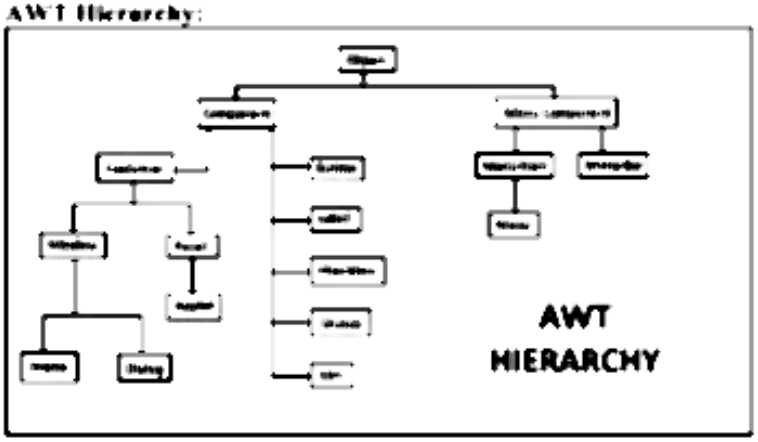
Example 6: Program to draw different Shapes.

```
import java.applet.Applet;
import java.awt.*;
public class GraphicalDemo1 extends Applet
{
 public void paint(Graphics g)
 {
      g.setColor(Color.black);
      g.drawString("Welcome to TYBBA (CA)",50,50);
      g.setColor(Color.blue);
      g.fillOval(170,200,30,30);
```

```
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
g.drawLine(21,11,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
```

}

AWT Hierarchy:



AWT Controls in Java: Controls are components that allow a user to interact with your application in various ways. The AWT supports the following types of controls:

1. **Labels:** The simplest control is a label. A label contains a string and is an object of type Label. Labels are passive controls that do not support any interaction with the user.

Creating Label : Label l = new Label(String);

Label Constructors

1. **Label() throws HeadlessException:** It creates a blank label.
2. **Label(String str) throws HeadlessException:** It creates a label that contains the string specified by str.
3. **Label(String str, int how):** It creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: **LEFT, Label.RIGHT, Label.CENTER**

Label Methods

1. **void setText(String str):** It is used to set or change the text in a label by using the setText() method. Here, str specifies the new label.
2. **String getText():** It is used to obtain the current **label** by calling getText() method. Here, the current **label** is returned.
3. **void setAlignment(int how):** It is used to set the alignment of the string within the label by calling setAlignment() method. Here, how is one of the alignment constants.

4. **int getAlignment()**: It is used to obtain the current alignment. getAlignment() is called.

2. AWT Button Control: The most widely used control is Button. A button is a component that contains a label and that generates an event when it is pressed.
Creating Button : **Button b = new Button(String label);**

Button Constructors

1. **Button() throws HeadlessException**: It creates an empty button.
2. **Button(String str) throws HeadlessException**: It creates a button that contains str as a label.

Button Methods

1. **void setLabel(String str)**: You can set its label by calling setLabel(). Here, str is the new Label for the button.
2. **String getLabel()**: You can retrieve its label by calling getLabel() method.

3. Canvas Control in java: Canvas encapsulates a blank window upon which you can draw or an application can receive inputs created by the user.
Canvas Constructor

1. **Canvas()** : Constructs a new Canvas.
2. **Canvas (GraphicsConfiguration config)** : Constructs a new Canvas given a GraphicsConfiguration object.

Canvas Methods

1. **void addNotify()**: It is used to create the peer of the canvas.
2. **void createBufferStrategy(int numBuffers)**: It is used to create a new strategy for multi-buffering on this component.
3. **BufferStrategy getBufferStrategy()**: It is used to return the BufferStrategy used by this component.

4. Checkbox Control in java: This component is used to display Checkbox.
Creating Checkbox : **Checkbox cb = new Checkbox(Label);**

Checkbox Constructor

1. **Checkbox() throws HeadlessException**: Creates a checkbox whose label is initially blank. The state of the checkbox is unchecked.
2. **Checkbox(String str) throws HeadlessException**: Creates a checkbox whose label is specified by str. The state of the checkbox is unchecked.
3. **Checkbox(String str, Boolean on) throws HeadlessException**: It allows you to have the initial state of the checkbox. If on is true, the checkbox is initially checked; otherwise, it is cleared.
4. **Checkbox(String str, Boolean on, CheckboxGroup cbGroup) throws HeadlessException or Checkbox(String str, CheckboxGroup cbGroup, Boolean on) throws HeadlessException**: It creates a checkbox whose label is specified by str and whose group is specified by cbGroup. If this checkbox isn't a part of a group, then cbGroup must be null. The value of on determines the initial state of the checkbox.

Methods of Checkbox

1. **boolean getState()**: To retrieve the present state of a checkbox.

3. **void setState(boolean on)**: To know its state, call setState(). Here, if on is true, the box is checked. If it's false, the box is cleared.
4. **String getLabel()**: you'll obtain the present label related to a checkbox by calling getLabel().
5. **void setLabel(String str)**: To have the label, call setLabel(). The string passed as str becomes the new label related to the invoking checkbox.

5. CheckboxGroup: Radio Buttons

Creating Radiobutton

```
CheckboxGroup cbg = new CheckboxGroup();
Checkbox rb = new Checkbox(Label, cbg, boolean);
```

CheckboxGroup Methods

1. **Checkbox getSelectedCheckbox()**: You can determine which checkbox in a group is currently selected by calling getSelectedCheckbox().
2. **void setSelectedCheckbox(Checkbox which)**: You can set a checkbox by calling setSelectedCheckbox(). Here, which is the checkbox that you simply want to be selected. The previously selected checkbox is going to be turned off.

6. AWT Choice Control in Java: This Component is use to create a dropdown list.
Note: Choice only defines the default constructor, which creates an empty list.
Creating Choice : Choice ch = new Choice();

Choice Methods

1. **void add(String name)**: To add a selection to the list, use add(). Here, the name is the name of the item being added.
2. **String getSelectedItem()**: It determines which item is currently selected. It returns a string containing the name of the item.
3. **int getSelectedIndex()**: It determines which item is currently selected. It returns the index of the item.
4. **int getItemCount()**: It obtains the number of items in the list.
5. **void select(int index)**: It is used to set the currently selected item with a zero-based integer index.
6. **void select(String name)**: It is used to set the currently selected item with a string that will match a name in the list.
7. **String getItem(int index)**: It is used to obtain the name associated with the item at the given index. Here, the index specifies the index of the desired items.

7. AWT List Control in Java: This component will display a group of items as a drop-down menu from which a user can select only one item. The List class provides a compact, multiple-choice, scrolling selection list.
Creating List : List l = new List(int, Boolean);
List Constructor

1. **List() throws HeadlessException**: It creates a list control that allows only one item to be selected at any one time.
2. **List(int numRows) throws HeadlessException**: Here, the value of numRows specifies the number of entries in the list that will always be visible.

3. **List(int numRows, boolean multipleSelect) throws HeadlessException**: If multipleSelect is true, then the user may select two or more items at a time. If it is false, then just one item could also be selected.

Method of Lists

1. **void add(String name)**: To add a selection to the list, use add(). Here, the name is that the name of the item being added. It adds items to the end of the list.
2. **void add(String name, int index)**: It also adds items to the list but it adds the items at the index specified by the index.
3. **String getSelectedItem()**: It determines which item is currently selected. It returns a string containing the name of the item. If more than one item is selected, or if no selection has been made yet, null is returned.
4. **int getSelectedIndex()**: It determines which item is currently selected. It returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made, -1 is returned.
5. **String[] getSelectedItems()**: It allows multiple selections. It returns an array containing the names of the currently selected items.
6. **int[] getSelectedIndexes()**: It also allows multiple selections. It returns an array containing the indexes of the currently selected items.
7. **int getItemCount()**: It obtains the number of items in the list.
8. **void select(int index)**: It is used to set the currently selected item with a zero-based integer index.
9. **String getItem(int index)**: It is used to obtain the name associated with the item at the given index. Here, the index specifies the index of the desired items.

8. **AWT Scroll Bar Control in java**: This component is used to create a Scrollbar.

Creating Scrollbar: Scrollbar sb = new Scrollbar();

Scrollbar Constructor:

1. **Scrollbar() throws HeadlessException**: It creates a vertical scrollbar.
2. **Scrollbar(int style) throws HeadlessException**: It allows you to specify the orientation of the scrollbar. If style is Scrollbar.VERTICAL, a vertical scrollbar is created. If a style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.
3. **Scrollbar(int style, int initialValue, int thumbSize, int min, int max) throws HeadlessException**: Here, the initial value of the scroll bar is passed in initialValue. The number of units represented by the width of the thumb is passed in thumbSize. The minimum and maximum values for the scroll bar are specified by min and max.

Scrollbar Methods:

1. **void setValues(int initialValue, int thumbSize, int min, int max)**: It is used to set the parameters of the constructors.
2. **int getValue()**: It is used to obtain the current value of the scroll bar. It returns the current setting.
3. **void setValue(int newValue)**: It is used to set the current value. Here, newValue specifies the new value for the scroll bar. When you set a value, the slider box inside the scroll bar is going to be positioned to reflect the new value.
4. **int getMaximum()**: It is used to retrieve the maximum values. They return the requested quantity. By default, 1 is the increment added to the scroll bar.
5. **int getMaximum()**: It is used to retrieve the maximum value. By default, 1 is the increment subtracted from the scroll bar.

9. **AWT TextComponent Control in Java:** The TextComponent class is the superclass of any component that permits the editing of some text. A text component embeddes a string of text. There are two types of TextComponent: **TextField, TextArea**

1. **TextField:** The TextField component will allow the user to enter some text. It is used to implements a single line text entry area, usually called an edit control.

   Creating TextField : TextField tf = new TextField(size);

   TextField Constructors

   1. **TextField() throws HeadlessException:** It creates a default textfield.
   2. **TextField(int numChars) throws HeadlessException:** It creates a textfield that is numChars characters wide.
   3. **TextField(String str) throws HeadlessException:** It initializes the textfield with the string contained in str.
   4. **TextField(String str, int numChars) throws HeadlessException:** It initializes a text field and sets its width.

   TextField Methods

   1. **String getText():** It is used to obtain the string currently contained in the text field.
   2. **void setText(String str):** It is used to set the text. Here, str is the new String.
   3. **void select(int startIndex, int endIndex):** It is used to select a portion of the text under program control. It selects the characters beginning at startIndex and ending at endIndex-1.
   4. **String getSelectedText():** It returns the currently selected text.
   5. **boolean isEditable():** It is used to determine editability. It returns true if the text may be changed and false if not.
   6. **void setEditable(boolean canEdit):** It is used to control whether the contents of a text field may be modified by the user. If canEdit is true, the text may be changed. If it is false, the text cannot be altered.
   7. **void setEchoChar(char ch):** It is used to disable the echoing of the characters as they are typed. This method specifies a single character that the TextField will display when characters are entered.
   8. **boolean echoCharIsSet():** By this method, you can check a text field to see if it is an this mode.
   9. **char getEchoChar():** It is used to retrieve the echo character.

2. **TextArea:** Sometimes one line of text input isn't enough for a given task. To handle these situations, the AWT includes an easy multiline editor called TextArea.

   Creating TextArea : TextArea ta = new TextArea();

   TextArea Constructor:

   1. **TextArea() throws HeadlessException:** It creates a default text area.
   2. **TextArea(int numLines, int numChars) throws HeadlessException:** It creates a text area that is numChars characters wide. Here, numLines specifies the height, in lines, of the text area.
   3. **TextArea(String str) throws HeadlessException:** It initializes the text area with the string contained in str.

4. **TextArea(String str, int numLines, int numChars) throws HeadlessException**: It instantiates a text field and sets its width. Initial text can be specified by str.
5. **TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException**: Here, you can specify the scroll bars that you want the control to have. sBars must be one of these values
    1. **SCROLLBARS_BOTH**
    2. **SCROLLBARS_NONE**
    3. **SCROLLBARS_HORIZONTAL_ONLY**
    4. **SCROLLBARS_VERTICAL_ONLY**

TextArea Methods: TextArea is a subclass of TextComponent. Therefore, it supports the **getText( ), setText( ), getSelectedText( ), select( ), isEditable( ),** and **setEditable( )** methods described in the TextField section. **TextArea** adds the following methods

1. **void append(String str)**: It appends the string specified by str to the end of the current text.
2. **void insert(String str, int index)**: It inserts the string passed in str at the specified index.
3. **voidReplaceRange(String str, int startIndex, int endIndex)**: It is used to replace the text. It replaces the characters from startIndex to endIndex-1.

Example: Java Program to display a simple calculator using AWT

```
import java.awt.*;
import java.awt.event.*;
public class Calculator extends Frame implements ActionListener
{
 Button add,clear,sub,mul,div,end,equal;
 TextField input;
 Panel pan1,pan2;
 int val1,val2,ans;
       char oper;
       public Calculator()
       {
              pan1=new Panel();
              pan2=new Panel();
              add=new Button("+");
              sub=new Button("-");
              mul=new Button("*");
              div=new Button("/");
              equal=new Button("=");
              end=new Button("Exit");
              on=new Button("On");
              clear=new Button("Clear");
              input=new TextField();
              pan1.add(add);
              pan1.add(sub);
              pan1.add(mul);
```

```
        pan1.add(div);
        pan1.add(equal);

        pan2.add(sub);
        pan2.add(clear);
        pan2.add(end);

        setLayout(new BorderLayout());
        add(input,"North");
        add(pan1,"Center");
        add(pan2,"South");

        input.setEnabled(false);
        add.setEnabled(false);
        sub.setEnabled(false);
        mul.setEnabled(false);
        div.setEnabled(false);

        add.addActionListener(this);
        sub.addActionListener(this);
        mul.addActionListener(this);
        div.addActionListener(this);
        equal.addActionListener(this);
        end.addActionListener(this);
        clear.addActionListener(this);

        setTitle("Arithmetic Calculator");
        setSize(400,400);
        setVisible(true);
}
public static void main(String args[])
{
        new Ass5().show();

}
public void actionPerformed(ActionEvent e)
{
        Button btn=(Button)e.getSource();
        if(btn==end)
        {
                System.exit(0);
        }
        if(btn==add || btn==sub || btn==mul || btn==div)
        {
                ex=Integer.parseInt(input.getText());
                oper=btn.getLabel().charAt(0);
        }
```

```
            if(b==equal)
            {
                  no2=Integer.parseInt(input.getText());
                  switch(oper)
                  {
                        case '+':
                                    ans=no1 + no2;
                                    break;
                        case '-':
                                    ans=no1 - no2;
                                    break;
                        case '*':
                                    ans=no1 * no2;
                                    break;
                        case '/':
                                    ans=no1 / no2;
                                    break;
                  }
                  input.setText(Integer.toString(ans));
            }
            if(b==clear)
            {
                  input.setText("");
                  input.requestFocus();
            }
            if(b==on)
            {
               input.setEnabled(true);
                  add.setEnabled(true);
                  sub.setEnabled(true);
                  mul.setEnabled(true);
                  div.setEnabled(true);

            }
}
}
```

Swing Architecture:

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

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

Swing Class Hierarchy:

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)
```

Types of Layout Managers: AWT package provides following types of Layout Managers

1. Flow Layout
2. Border Layout
3. Card Layout
4. Grid Layout
5. Grid Bag Layout

Examples:

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

1. Flow Layout: This layout will display the components in sequence from left to right, from top to bottom.

Constructor:

```
FlowLayout fl = new FlowLayout();
FlowLayout fl = new FlowLayout(int align);
FlowLayout fl = new FlowLayout(int align, int hgap, int vgap);
```

For Example : Java Program to demonstrate Flow Layout Manager

```
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutDemo
{
        JFrame f;
        FlowLayoutDemo()
        {
                f = new JFrame();
                JLabel l = new JLabel("Enter Name");
                JTextField t = new JTextField(10);
                JButton b = new JButton("SUBMIT");
                f.add(l);
                f.add(t);
                f.add(b);
                f.setLayout(new FlowLayout(FlowLayout.RIGHT));
                //setting flow layout of right alignment
                f.setSize(300, 300);
                f.setVisible(true);
        }
        public static void main(String[]args)
        {
                new FlowLayoutDemo();
        }
}
```

Output:

2. Border Layout: This layout will display the components along the borders of the container. This layout contains five locations. Locations are North, South, East, west, and Center. The default region is the center.

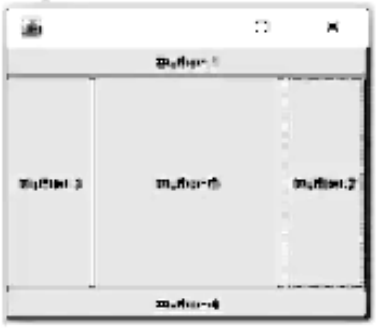
Constructor:

```
BorderLayout bl = new BorderLayout();
BorderLayout b = new BorderLayout(int vgap, int hgap);
```

For Example : Java Program to demonstrate Border Layout Manager.

```
import java.awt.*;
public class BorderLayoutDemo
{
  public static void main (String[] args)
  {
    Frame f1 = new Frame ();
    f1.setSize (250, 250);
    Button b1 = new Button ("Button1");
    Button b2 = new Button ("Button2");
    Button b3 = new Button ("Button3");
    Button b4 = new Button ("Button4");
    Button b5 = new Button ("Button5");
    f1.add (b1, BorderLayout.NORTH);
    f1.add (b2, BorderLayout.EAST);
    f1.add (b3, BorderLayout.WEST);
    f1.add (b4, BorderLayout.SOUTH);
    f1.add (b5);
    f1.setVisible (true);
  }
}
```

Output:



3.Card Layout: A card layout represents a stack of cards displayed on a container. At a time only one card can be visible and each can contain the only component.

Constructor

```
Cardlayout c1 = new Cardlayout();
Cardlayout c1 = new Cardlayout(int hgap, int vgap);
```

To add the components in Cardlayout we use add method

```
add("Constraint", Component);
```

Methods of CardLayout

1. first(Container) : It is used to flip to the first card of the given container.
2. last(Container) : It is used to flip to the last card of the given container.
3. next(Container) : It is used to flip to the next card of the given container.
4. previous(Container) : It is used to flip to the previous card of the given container.
5. show(Container, cardName) : It is used to flip to the specified card with the given name.

For Example: Java Program to demonstrate CardLayout Manager.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JButton;
import java.awt.event.*;
public class CardLayoutDemo extends JFrame implements ActionListener
{
  JButton b1, b2, b3, b4, b5;
  CardLayout cl;
  Container c;
  CardLayoutDemo()
  {
    b1 = new JButton("Button 1");
    b2 = new JButton("Button 2");
    b3 = new JButton("Button 3");
    b4 = new JButton("Button 4");
    b5 = new JButton("Button 5");
    c = this.getContentPane();
    cl = new CardLayout(10, 20);
    c.setLayout(cl);
    c.add("Card1", b1);
    c.add("Card2", b2);
    c.add("Card3", b3);
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
    setVisible(true);
    setSize(400, 400);
    setTitle("Card Layout");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent ae)
  {
    cl.next(c);
  }
```

```
    public static void main (String[] args)
    {
        new GridLayoutDemo ();
    }
}
```

Output:

4. Grid Layout: The layout will display the components in the format of rows and columns vertically. The components will be divided into a table of rows and columns.

Constructor:

GridLayout g1 = new GridLayout(int rows, int cols);

GridLayout g1 = new GridLayout(int rows, int cols, int vgap, int hgap);

For Example: Java Program to demonstrate Grid Layout Manager.

```
import java.awt.*;
import java.awt.event.*;
public class GridLayoutDemo
{
    public static void main (String[] args)
    {
        Frame f1 = new Frame ();
        f1.setSize (250, 250);
        GridLayout obj = new GridLayout (2, 2);
        f1.setLayout (obj);
        Panel p1 = new Panel ();
        Label l1 = new Label ("Enter name");
        TextField tf = new TextField(10);
        Button b1 = new Button ("Submit");
        p1.add (l1);
        p1.add (tf);
        p1.add (b1);
        f1.add (p1);
        Panel p2 = new Panel ();
        f1.add (p2);
```

```
        Panel p1 = new Panel ();
        f1.add(p1);
        Label l2 = new Label ("Welcome to Java");
        f1.add(l2);
        f1.setVisible (true);
    }
}
```

Output:

5. Grid Bag Layout: This layout is the most efficient layout that can be used for displaying components by specifying the location, the size, etc.

Constructor:

```
GridBagLayout gbl = new GridBagLayout();
```

Instance variables to manipulate the GridBagLayout object Constraints are:

Variable	Role
gridx and gridy	These contain the coordinates of the origin of the grid. They allow a at a specific position of a component positioning. By default, they have GridBagConstraint.RELATIVE value which indicates that a component can be placed to the right of previous.
gridwidth, gridheight	Define how many cells will occupy component (height and width) by The default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints.REMAINDER value specifies that the next component inserted will be the last of the line or the current column. the value GridBagConstraints.RELATIVE up the component after the last component of a row or column
fill	Defines the fate of a component smaller than the grid cell. GridBagConstraints.NONE returns the original size. Default. GridBagConstraints.HORIZONTAL expanded horizontally. GridBagConstraints.VERTICAL GridBagConstraints.BOTH expanded vertically expanded to the dimensions of the cell.
ipadx, ipady	Used to define the horizontal and vertical expansion f components. not works if expansion is engraved by fill. The default value is (0,0).

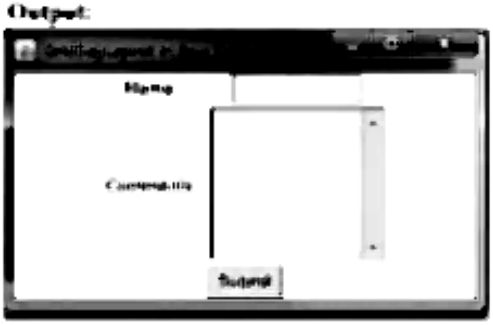
anchor	When a component is smaller than the cell in which it is contained, it can be positioned using this variable to define the side from which the empty space should be assigned within the cell. Possible variables NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST and EAST
weightx, weighty	Used to define the distribution of space in case of change of dimension

For Example: Java Program to demonstrate GridBag Layout Manager.

```
import java.awt.*;
class GridBagLayoutExample extends Frame
{
    GridBagLayoutExample()
    {
        Label lblName = new Label("Name");
        TextField txtName = new TextField(10);
        Label lblComments = new Label("Comments");
        TextArea txtComments = new TextArea(5,15);
        Button btnSubmit = new Button("Submit");
        setLayout(new GridBagLayout());
        GridBagConstraints gc = new GridBagConstraints();
        add(lblName,gc,0,0,1,1,0,0);
        add(txtName,gc,1,0,1,1,0,50);
        add(lblComments,gc,0,1,1,1,0,0);
        add(txtArea,comments,gc,1,1,1,1,0,60);
        add(btnSubmit,gc,0,2,2,1,0,50);
    }
    void add(Component comp,GridBagConstraints gc,int x,int y,int w,int h,int wx,int wy)
    {
        gc.gridx = x;
        gc.gridy = y;
        gc.gridwidth = w;
        gc.gridheight = h;
        gc.weightx = wx;
        gc.weighty = wy;
        add(comp,gc);
    }
}
class GridBagLayoutJavaExample
{
    public static void main(String args[])
    {
```

```
GridBagLayoutExample frame = new GridBagLayoutExample();
frame.setTitle("GridBagLayout in Java Example");
frame.setSize(300,300);
frame.setVisible(true);
}
```

Output:



Container in Swing:

1. **JFrame** – This is a top level container which can hold components and containers like panels.

Constructors:

JFrame()

JFrame(String title)

Methods:

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

2. JPanel This is a medium level container which can hold components and can be added to other containers like frame and panels.

Constructors:

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

Component used in Swing:

1. **Label** With the JLabel class, you can display uneditable 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

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

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

Constructors

JButton(Icon i)
JButton(String s)
JButton(String s, Icon i)

Methods

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

Event Handling: ActionEvent

3. Check boxes : This component allows the user to select multiple items from a group of items. It is used to create a CheckBox. It is used to turn an option ON or OFF.

Constructors

- **JCheckBox(Icon i)**
- **JCheckBox(Icon i, boolean state)**
- **JCheckBox(String s)**
- **JCheckBox(String s, boolean state)**
- **JCheckBox(String s, Icon i)**
- **JCheckBox(String s, Icon i, boolean state)**

Methods

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

Event Handling: ItemEvent

4. Radio Buttons : This component allows the user to select only one item from a group item. By using the JRadioButton component you can choose one option from multiple options.

Constructors

JRadioButton(): It is used to create an unselected radio button with no text.

JRadioButton(Label): It is used to create an unselected radio button with specified text.

JRadioButton(Label, boolean): It is used to create a radio button with the specified text and selected status.

Creating a ButtonGroup:

ButtonGroup(): The class is used to place multiple RadioButton into a single group. So the user can select only one value from that group. We can add RadioButtons to the ButtonGroup by using the **add** method.

Constructor: ButtonGroup bg = new ButtonGroup();

Methods:

void add(AbstractButton)	Adds a button to the group
void remove(AbstractButton)	Removes a button from the group

Example:

```
JRadioButton jrb = new JRadioButton();
ButtonGroup bg = new ButtonGroup();
add(jrb);
```

4. Combo Boxes: This component will display a group of items as a drop-down menu from which one item can be selected. At the top of the menu the choice selected by the user is shown. It basically inherits JComponent class. We can add the items to the ComboBox by using the addItem() method.

Constructor:

JComboBox(): It is used to create a JComboBox with a default data model.

JComboBox(Object[] items): It is used to create a JComboBox that contains the elements in the specified array.

JComboBox(Vector<?> items): It is used to create a JComboBox that contains the elements in the specified Vector.

Methods:

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

Event Item Event

6. List: The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

Constructor JList(ListModel)

List models

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

Methods

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

Example

```
listModel1 = new
DefaultListModel();
listModel1.addElement("India");
listModel1.addElement("Japan");
listModel1.addElement("France");
listModel1.addElement("Denmark");
list = new JList(listModel1);
```

Event: ActionEvent

7. **Text Classes**: All text related classes are inherited from JTextComponent class.

1. **JTextField**: The JTextField component allows the user to type some text on a single line. It basically inherits the JTextComponent class.

Constructors

- ✓ JTextField(): It is used to create a new TextField.
- ✓ JTextField(String text): It is used to create a new TextField initialized with the specified text.
- ✓ JTextField(String text, int columns): It is used to create a new TextField initialized with the specified text and columns.
- ✓ JTextField(int columns): It is used to create a new empty TextField with the specified number of columns.

Example:

```
JTextField jf = new JTextField();
JTextField jf= new JTextField(20);
```

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

Constructors

- ✓ JPasswordField(): It is used to construct a new JPasswordField, with a default document, null starting text string, and column width.
- ✓ JPasswordField(int columns): It is used to construct a new empty JPasswordField with the specified number of columns.
- ✓ JPasswordField(String text): It is used to construct a new JPasswordField initialized with the specified text.

Example:

```
JPasswordField jpf = new JPasswordField();
```

Methods:

void setText(String), String getText()	Set or get the text displayed by the text field.
char[] getPassword()	Set or get the text displayed by the text field.

void setEditable(boolean), boolean isEditable()	Set or get whether the user can edit the text in the text field
void setColumns(int), int getColumns()	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
int getColumnWidth()	Get the width of the text field's columns. This value is established implicitly by the font
void setEchoChar(char), char getEchoChar()	Set or get the echo character, i.e. the character displayed instead of the actual characters typed by the user

Event: ActionEvent

3. **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)

Example:

```
JTextArea ja = new JTextArea();
JTextArea ja = new JTextArea (5, 20)
```

8. Dialog Boxes:

Types:

1. **Modal** : won't let the user interact with the remaining windows of application until he/she deals with it. Ex: when user wants to read a file, user must specify the name before prg. can begin read operation.
2. **Modeless dialog box** : Lets the user enters information in both, the dialog box & remainder of application ex: toolbar

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

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

- **Menu**: Following table gives idea about Menu component

Creating and Setting Up Menu Bars	
Constructor or Method	Description
JMenuBar()	Creates a menu bar.
JMenu add(JMenu)	Creates a menu bar.
void setJMenuBar(JMenuBar) JMenuBar getJMenuBar()	Sets or gets the menu bar of an applet, dialog, frame, internal frame, or root pane
Creating and Populating Menus	
JMenu() JMenu(String)	Creates a menu. The string specifies the text to display for the menu.
JMenuItem add(JMenuItem) JMenuItem add(Action) JMenuItem add(String)	Adds a menu item to the current end of the menu. If the argument is an Action object, then the menu creates a menu item. If the argument is a string, then the menu automatically creates a JMenuItem object that displays the specified text.
void addSeparator()	Adds a separator to the current end of the menu.
JMenuItem insert(JMenuItem, int) JMenuItem insert(Action, int) void insert(String, int) void insertSeparator(int)	Inserts a menu item or separator into the menu at the specified position. The first menu item is at position 0, the second at position 1, and so on. The JMenuItem, Action, and String arguments are treated the same as in the corresponding add methods.
void remove(JMenuItem) void remove(int) void removeAll()	Removes the specified item(s) from the menu. If the argument is an integer, then it specifies the position of the menu item to be removed.
Implementing Menu Items	
JMenuItem() JMenuItem(String) JMenuItem(Icon) JMenuItem(String, Icon) JMenuItem(String, int)	Creates an ordinary menu item. The icon argument, if present, specifies the icon that the menu item should display. Similarly, the string argument specifies the text that the menu item should display. The integer argument specifies the keyboard mnemonic to use. You can specify any of the relevant VK constants defined in the KeyEvent class. For example, to specify the A key, use KeyEvent.VK_A.
JCheckBoxMenuItem() JCheckBoxMenuItem(String) JCheckBoxMenuItem(Icon) JCheckBoxMenuItem(String, Icon) JCheckBoxMenuItem(String, boolean)	Creates a menu item that looks and acts like a check box. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected (checked). Otherwise, the menu item is initially unselected.

boolean) JCheckBoxMenuItem(String, Icon, boolean)	
JRadioButtonMenuItem() JRadioButtonMenuItem(String) JRadioButtonMenuItem(Icon) JRadioButtonMenuItem(String, Icon) JRadioButtonMenuItem(String, boolean) JRadioButtonMenuItem(Icon, boolean) JRadioButtonMenuItem(String, Icon, boolean)	Creates a menu item that looks and acts like a radio button. The string argument, if any, specifies the text that the menu item should display. If you specify true for the boolean argument, then the menu item is initially selected. Otherwise, the menu item is initially unselected.
void setState(boolean) boolean getState() (or JCheckBoxMenuItem) void setEnabled(boolean)	Set or get the selection state of a check box menu item. If the argument is true, enable the menu item. Otherwise, disable the menu item.

Example: Java JMenuItem and JMenu

```
import java.awt.*;
class Menud_example
{       JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        Menud_example()
        {
          JFrame f= new JFrame("Menu and MenuItem Example");
          JMenuBar mb=new JMenuBar();
          menu=new JMenu("Menu");
          submenu=new JMenu("Sub Menu");
          i1=new JMenuItem("Item 1");
          i2=new JMenuItem("Item 2");
          i3=new JMenuItem("Item 3");
          i4=new JMenuItem("Item 4");
          i5=new JMenuItem("Item 5");
          menu.add(i1); menu.add(i2); menu.add(i3);
          submenu.add(i4); submenu.add(i5);
          menu.add(submenu);
          mb.add(menu);
          f.setJMenuBar(mb);
```

```
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}
```

10. JTable Class: The JTable class is used to display data in tabular form. It is composed of rows and columns.

Constructors:

JTable() Creates a table with empty cells.

JTable(Object[][] rows, Object[] columns) Creates a table with the specified data.

Example:

```
import javax.swing.*;
public class TableExample
{
        JFrame f;
        TableExample()
        {
                f=new JFrame();
                String data[][]={ {"101","Amit","670000"},
                                  {"102","Jai","780000"},
                                  {"101","Sachin","700000"}
                                };
                String column[]={"ID","NAME","SALARY"};
                JTable jt=new JTable(data,column);
                jt.setBounds(30,40,200,300);
                JScrollPane sp=new JScrollPane(jt);
                f.add(sp);
                f.setSize(300,400);
                f.setVisible(true);
        }
        public static void main(String[] args)
        {
                new TableExample();
        }
}
```

Output:

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

Java has two types of events:

1. **Low Level Events:** Low level events represent direct communication from user. A low level event is a key press or a key release, a mouse click, drag, move or wheel, 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 on a component. E.g. mouse is pressed, released, 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, gains focus or when focus is transferred into or out of the Window.

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

Event	Description
ActionEvent	Indicates that a component-defined action occurred. This high-level event is generated by a component (such as Button) when the component-specific action

	actions (such as being pressed)
AdjustmentEvent	The adjustment event is emitted by Adjustable objects like scrollbars
ItemEvent	It 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	It indicates that an object's text changed. This high level event is generated by an object (such as TextComponent) when its text changes

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

Event	Event Source	Event Listener	Method to add listener to event source
Low level Events			
ComponentEvent	Component	ComponentListener	addComponentListener()
FocusEvent	Component	FocusListener	addFocusListener()
KeyEvent	Component	KeyListener	addKeyListener()
MouseEvent	Component	MouseListener MouseMotionListener	addMouseListener() addMouseMotionListener()
ContainerEvent	Container	ContainerListener	addContainerListener()
WindowEvent	Window	WindowListener	addWindowListener()
High level Events			
ActionEvent	Button List MenuItem TextField	ActionListener	addActionListener()
ItemEvent	Choice CheckBox CheckBoxMenuItem List	ItemListener	addItemListener()
AdjustmentEvent	Scrollbar	AdjustmentListener	addAdjustmentListener()

TextEvent	TextField TextArea	TextListener	addTextListener()

Listener Methods:

Methods	Description
ComponentListener	
componentResized(ComponentEvent e)	Invoked when component's size changes.
componentMoved(ComponentEvent e)	Invoked when component's position changes.
componentShown(ComponentEvent e)	Invoked when component has been made visible.
componentHidden(ComponentEvent e)	Invoked when component has been made invisible.
FocusListener	
focusGained(FocusEvent e)	Invoked when component gains the keyboard focus.
focusLost(FocusEvent e)	Invoked when component loses the keyboard focus.
KeyListener	
keyTyped(KeyEvent e)	Invoked when a key is typed.
keyPressed(KeyEvent e)	Invoked when a key is pressed.
keyReleased(KeyEvent e)	Invoked when a key is released.
MouseListener	
mouseClicked(MouseEvent e)	Invoked when a mouse button is clicked (pressed and released) on a component.
mousePressed(MouseEvent e)	Invoked when a mouse button is pressed on a component.
mouseReleased(MouseEvent e)	Invoked when a mouse button is released on a component.
mouseEntered(MouseEvent e)	Invoked when a mouse enters a component.
mouseExited(MouseEvent e)	Invoked when a mouse exits a component.
MouseMotionListener	

mouseDragged(MouseEvent e)	Invoked when a mouse button is pressed on a component and then dragged.
mouseMoved(MouseEvent e)	Invoked when the mouse cursor is moved onto a component but mouse button is not pressed.
ContainerListener	
componentAdded(ContainerEvent e)	Invoked when a component is added to the container.
componentRemoved(ContainerEvent e)	Invoked when a component is removed from the container.
WindowListener	
windowOpened(WindowEvent e)	Invoked the first time a window is made visible.
windowClosing(WindowEvent e)	Invoked when the user attempts to close the window from the window's system menu.
windowClosed(WindowEvent e)	Invoked when a window has been closed as the result of calling dispose on the window.
windowIconified(WindowEvent e)	Invoked when a window is changed from a normal to a minimized state.
windowDeiconified(WindowEvent e)	Invoked when a window is changed from minimized to normal state.
windowActivated(WindowEvent e)	Invoked when the window is set to be the active window.
windowDeactivated(WindowEvent e)	Invoked when the window is no longer the active window.
ActionListener	
actionPerformed(ActionEvent e)	Invoked when an action occurs.
ComponentListener	
componentShown(ComponentEvent e)	Invoked when a component has been made visible.
AdjustmentListener	
adjustmentValueChanged(AdjustmentEvent e)	Invoked when the value of the adjustable has changed.

e)	
TextListener	
textValueChanged(AdjustmentEvent e)	Invoked when the value of the text has changed.

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

The Adapter classes provided by AWT are as follows:

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

Example: Program to close window

```
// importing the necessary libraries
import java.awt.*;
import java.awt.event.*;

public class AdapterExample
{
   Frame f;   // object of Frame
   // class constructor
   AdapterExample()
   {
         // creating a frame with the title
         f = new Frame ("Window Adapter");
         // adding the WindowListener to the frame overriding the windowClosing() method
         f.addWindowListener(new WindowAdapter()
         {
               public void windowClosing(WindowEvent e)
               {
                     f.dispose();
               }
```

```
        });
      // setting the user layout and
      frame.setSize(400, 400);
      frame.setLayout(null);
      frame.setVisible(true);
    }

public static void main(String[] args)
{
  new Adapter1Example();
}
}
```

Example 1: Sample Program to understand JLabel Swing Control in Java:

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

public class JLabelDemo extends JFrame
{
  JLabel jl;
  JLabelDemo()
  {
    jl = new JLabel("Good Morning");
    Container c = this.getContentPane();
    c.setLayout(new FlowLayout());
    c.setBackground(Color.blue);
    Font f = new Font("serif", Font.BOLD, 34);
    jl.setFont(f);
    jl.setBackground(Color.white);
    c.add(jl);
    this.setVisible(true);
    this.setSize(400, 400);
    this.setTitle("JLabel");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[]args)
  {
    new JLabelDemo();
  }
}
```

Example 2: Java Program to understand the above discussed Swing Controls

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingDemo2 extends JFrame implements ActionListener
{
  JRadioButton eng, doc;
  ButtonGroup bg;
  JTextField jtf;
  JCheckBox bicd, ccb, acb;
  JTextArea jta;
  SwingDemo2 ()
  {
    eng = new JRadioButton ("Engineer");
    doc = new JRadioButton ("Doctor");
    bg = new ButtonGroup ();
    bg.add (eng);
    bg.add (doc);
    jtf = new JTextField (20);
    bicd = new JCheckBox ("Bike");
    ccb = new JCheckBox ("Car");
    acb = new JCheckBox ("Aeroplane");
    jta = new JTextArea (5, 20);
    Container c = this.getContentPane ();
    c.setLayout (new FlowLayout ());

    // Registering the listeners with the components
    eng.addActionListener (this);
    doc.addActionListener (this);
    bicd.addActionListener (this);
    ccb.addActionListener (this);
    acb.addActionListener (this);
    c.add (eng);
    c.add (doc);
    c.add (jtf);
    c.add (bicd);
    c.add (ccb);
    c.add (acb);
    c.add (jta);
    this.setVisible (true);
    this.setSize (400, 400);
    this.setTitle ("Swing Demo Example");
```

```
        this.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed (ActionEvent ae)
    {
        if (ae.getSource () == eng)
        {
            jtf.setText ("You are an Engineer");
        }
        if (ae.getSource () == doc)
        {
            jtf.setText ("You are an Doctor");
        }
        String str = "";
        if (bx1.isSelected ())
        {
            str += "Bike\n";
        }
        if (cb1.isSelected ())
        {
            str += "Car\n";
        }
        if (cb2.isSelected ())
        {
            str += "Aeroplane\n";
        }
        jta.setText (str);
    }
    public static void main (String[] args)
    {
        new SwingDemo2 ();
    }
}
```

Example 3: Java Program to understand JButton and Border controls.

```
import javax.swing.*;
import java.awt.*;
public class JButtonDemo extends JFrame
{
    private JButton button[];
    private JPanel panel;
    public JButtonDemo ()
    {
```

```
        setTitle ("JButton Borders");
        panel = new JPanel ();
        panel.setLayout (new GridLayout (7, 1));
        button = new JButton[7];
        for (int count = 0; count < button.length; count++)
        {
                button[count] = new JButton ("Button " + (count + 1));
                panel.add (button[count]);
        }
        button[0].setBorder (BorderFactory.createLineBorder (Color.blue));
        button[1].setBorder (BorderFactory.createRaisedBevelBorder ());
        button[2].setBorder (BorderFactory.createLoweredBevelBorder ());
        button[3].setBorder (BorderFactory.createMatteBorder (1, 5, 1, 1,
        Color.orange, Color.red, Color.blue));
        button[4].setBorder (BorderFactory.createEmptyBorder (10, 10, 10, 10));
        button[5].setBorder (BorderFactory.createEtchedBorder ());
        button[6].setBorder (BorderFactory.createTitledBorder ("Titled Border"));
        add (panel, BorderLayout.CENTER);
        setSize (400, 300);
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo (null);
        setVisible (true);
    }
    public static void main (String[]args)
    {
        new JButtonDemo ();
    }
}
```

Example 4: Sample Program to understand JComboBox control in Java

```
import javax.swing.*;
public class JComboBoxDemo
{
    JFrame f;
    JComboBoxDemo ()
    {
        f = new JFrame ("ComboBox Example");
        String country[] = {"Hyderabad", "Chennai", "Bangalore", "Mumbai", "Delhi"};
        JComboBox cb = new JComboBox (country);
        cb.setBounds (50, 50, 90, 20);
        f.add (cb);
        f.setLayout (null);
```

```
    f.setSize (400, 500);
    f.setVisible (true);
  }
  public static void main (String[] args)
  {
    new JComboBoxDemo ();
  }
}
```

Example 4: Sample Program to understand JTabbedPane control in Java

```
import javax.swing.*;
import java.awt.*;
public class JTabbedPaneDemo
{
  public static void main (String args[])
  {
    JFrame f=new JFrame ("Technologies");
    JTabbedPane tabbedPane = new JTabbedPane ();
    JPanel panel1, panel2, panel3, panel4, panel5;
    panel1 = new JPanel ();
    panel2 = new JPanel ();
    panel3 = new JPanel ();
    panel4 = new JPanel ();
    panel5 = new JPanel ();
    tabbedPane.addTab ("Cricket", panel1);
    tabbedPane.addTab ("Badminton", panel2);
    tabbedPane.addTab ("Football", panel3);
    tabbedPane.addTab ("Basketball", panel4);
    tabbedPane.addTab ("Tennis", panel5);
    frame.add (tabbedPane);
    frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
    frame.setSize (550, 350);
    frame.setVisible (true);
  }
}
```

Example 6: Sample Program to understand JPasswordField Swing control in Java

```
import javax.swing.*;
public class JPasswordFieldDemo
{
  public static void main (String[] args)
  {
```

```
        JFrame f = new JFrame ("Password Field Example");
        JPasswordField value = new JPasswordField ();
        JLabel l1 = new JLabel ("Password:");
        l1.setBounds (20, 100, 80, 30);
        value.setBounds (100, 100, 100, 30);
        f.add (value);
        f.add (l1);
        f.setSize (300, 300);
        f.setLayout (null);
        f.setVisible (true);
    }
}
```

Practice Set:

1. Develop an applet that draws a circle. The dimension of the applet should be 500 * 500 pixels the circle should be centered the applet and have a radius of 100 pixels. Display your name centered on a circle(using drawOval() method)
2. Write a program to create a frame using AWT. Implement mouseClicked(), mouseEntered() and mouseExited() events. Frame should become visible when mouse enters it.
3. Write a program to display a string in frame window with pink colour as background.
4. Write a program to create two buttons named "Red" and "Blue". When a button is pressed the background colour should be set to the colour named by the button's label.
5. Write a program which responds to KEY_TYPED event and updates the status window with message ("Typed character is X"). Use adapter class for other two events.
6. Write a program to create two buttons labelled 'GetInfo' and 'GetCGPA'. When button 'GetInfo' is pressed, it displays your personal information (Name, Course, Roll No, College) and when button 'GetCGPA' is pressed, it displays your CGPA in previous semester.

Set A:

1. Write a program that asks the user's name, and then greets the user by name. Before outputting the user's name, convert it to upper case letters. For example, if the user's name is Raj, then the program should respond "Hello, RAJ, nice to meet you!"
2. Write a program that reads one line of input text and breaks it up into words. The words should be output one per line. A word is defined to be a sequence of letters. Any characters in the input that are not letters should be discarded. For example, if the user inputs the line: He said, "That's not a good idea." then the output of the program should be:

```
He
said
Thats
not
```

a

good

idea

3. Write a program that will read a sequence of positive real numbers entered by the user and will print the same numbers in sorted order from smallest to largest. The user will input a zero to mark the end of the input. Assume that at most 100 positive numbers will be entered.
4. Create an Applet that displays the x and y position of the cursor movement using Mouse and Keyboard. (Use appropriate listener)
5. Create the following GUI screen using appropriate layout managers

Set B:

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

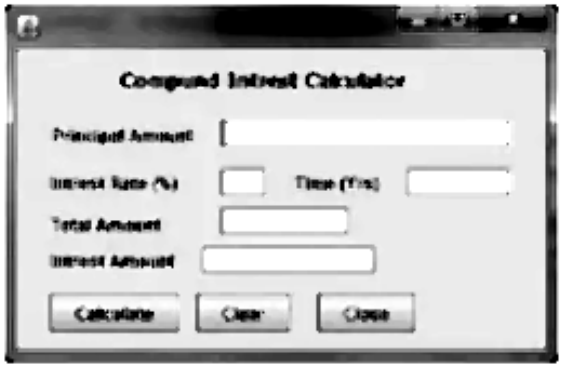
2. Write a java program to implement following. Program should handle appropriate events

3. Write an applet application to draw Triangle

4. Write an applet application to display Table lamp. The color of lamp should get change on random basis.
5. Write a java program to design email registration form. (Use appropriate Swing components in form)

Set C

1. Write a java program to accept the details of employee employee-no, name, salary and display it on next frame using appropriate event.
2. Write a java program to display at least five records of employee in JTable (Emp. Ename Sal)
3. Write a java Program to change the color of frame. If user clicks on close button then the position of frame should get change.
4. Write a java program to display following screen.



5. Write an applet application to display money and sad face

Assignment Evaluation

0. Not Done [ ]
1. Incomplete [ ]
2. Late Complete [ ]
3. Needs Improvement [ ]
4. Complete [ ]
5. Well Done [ ]

Signature of Instructor