INSTITUTE OF
TECHNOLOGY
NAVI MUMBAI

Department of Computer Engineering

Lab Manual

Second Year Semester-III

Subject: Skill base Lab course: Object Oriented Programming with Java

Odd Semester

Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

Our Quality Policy

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodology.

Our Motto: If it is not of quality, it is NOTRAIT!

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

- To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering.
- To provide the diverse platforms of sports, technical, co-curricular and extracurricular activities for the overall development of student with ethical attitude.
- To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service.
- To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Educational Objectives (PEOs)

1. **Learn and Integrate**
To provide Computer Engineering students with a strong foundation in the mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problems and to prepare them for graduate studies.

2. **Think and Create**
To develop an ability to analyze the requirements of the software and hardware, understand the technical specifications, create a model, design, implement and verify a computing system to meet specified requirements while considering real-world constraints to solve real world problems.

3. **Broad Base**
To provide broad education necessary to understand the science of computer engineering and the impact of it in a global and social context.

4. **Techno-leader**
To provide exposure to emerging cutting edge technologies, adequate training & opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities.

5. **Practice citizenship**
To provide knowledge of professional and ethical responsibility and to contribute to society through active engagement with professional societies, schools, civic organizations or other community activities.

6. **Clarify Purpose and Perspective**
To provide strong in-depth education through electives and to promote student awareness on the life-long learning to adapt to innovation and change, and to be successful in their professional work or graduate studies.

Departmental Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences..

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1: To build competencies towards problem solving with an ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

PSO2: To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.

PSO3: To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.

Index

List of Experiments

Sr. No.	Experiments Name
1.	The grading system describes how well students have achieved the learning objectives or goals established for a class of course of study. This system helps to categorize the students according to their grades. Design a system tht reads marks obtained by a student in a test of 100 marks and assign the grade.
2.	Prime numbers are important because the security of many encryption algorithms are based on the fact that it is very fast to multiply two large numbers and get the result, while it is extremely computer-intensive to do th reverse. Enlist all the prime numbers between 1 and 1000 to create a base for cryptography.
3.	The vendor provide a rubber material that provides the protection to the edges of rectangular object and paper material for protection of front and back of rectangular object. So a vendor needs a application to calculate the amount of rubber and paper material required for covering rectangular object. Create an application with the methods to calculate area and perimeter on provision of the length and breadth of rectangular object to get the exact amount of rubber and paper required. Also calculate the the paper required to cover square object using method overloading.
4.	An electrical engineer needs a complex number calculator for performing the operation of addition of alternating current represented using complex number. Create an application that takes two complex numbers as parameters and returns the resulting complex number, which is the addition of two complex numbers.
5.	Programmers can define their own packages to bundle group of related classes/interfaces as per their requirement. Create your own package named as your “first name” and encapsulate the Product class in it for recording and displaying the product information. Also import the Product class to demonstrate the use of packages.
6.	2D array is used in many real-life applications where we need to organize data in tabular/matrix format. Hence a matrix manipulator is required with functionality of reading, displaying and flipping data from the matrix. Generate the methods, for the functionality mentioned above for creation the matrix manipulator.
7.	A character sequence is to be read as an input and result need to declare as “yes” or “no” by investigating the fact that traversing the characters sequence backwards and forwards results in same sequence. Write a program for the same using StringBuffer.
8.	Admission section needs an application for storing the student information whenever new student come for admission. Information such as name, roll no, age, contact no. etc. is required to be stored for each student and at the end of the day the list need to be displayed on screen for verification. Make use of the vector class method for adding new student record.
9.	Write a program to calculate volume of sphere using multilevel inheritance. The base class method will accept the radius from user. A class will be derived from the above mentioned class that will have a method to find the area of a circle derived from this will have method to calculate and display the volume of the sphere.

10.	The purpose of an abstract class is to define some common behaviour that can be inherited by multiple subclasses, without implementing the entire class. Create an abstract class with common behaviour of different shapes.
11.	Consider a university where students who participate in the National Games or Olympics are given some grace marks. The grace marks provided are fixed and same for every student. Create an application that keeps student's academic marks and Sports grace marks separate and generate total of marks considering academics and sports both. Also invoke methods of base class & interface using reference. (Hint: Make use of Interface).
12.	Through Custom exception user can raise application-specific error code. You are required to calculate a square of even number provided as input by user. However, if a user provides an odd number as input, then an exception must be thrown explicitly with message indicating the input number must be even number.
13.	Divide your program into two parts: One to read a number and the other to calculate its square. Provide a simultaneous execution of both parts of a program to maximum utilize the CPU time.
14.	Create a GUI application that runs in a browser and displays the name and roll no of student provided as specific attribute for displaying on browser.
15.	You need to create an interactive GUI based form for reading student information such as name, date of birth, gender, department, contact no., address etc. using AWT components and handle events such as mouse movement or button click to store information.

Course Objective, Course Outcome & Experiment Plan

Course Objective:

1	To learn the object oriented programming concepts.
2	To study various java programming concept like multithreading, exception handling, packages etc.
3	To explain components of GUI based programming.

Course Outcomes:

CO1	To apply fundamental programming constructs to understand object oriented programming concepts.
CO2	To illustrate the concept of packages, classes and objects.
CO3	To elaborate the concept of strings, arrays and vectors.
CO4	To understand and demonstrate the concept of inheritance and interfaces.
CO5	To interpret and apply the notion of exception handling and multithreading.
CO6	To develop GUI based application to understand event-based GUI handling

Experiment Plan:

Module No.	Week No.	Experiments Name	Course Outcome	Waightage
1.	W1	The grading system describes how well students have achieved the learning objectives or goals established for a class of course of study. This system helps to categorize the students according to their grades. Design a system tht reads marks obtained by a student in a test of 100 marks and assign the grade.	CO1	5
2.	W2	Prime numbers are important because the security of many encryption algorithms are based on the fact that it is very fast to multiply two large numbers and get the result, while it is extremely computer-intensive to do th reverse. Enlist all the prime numbers between 1 and 1000 to create a base for cryptography.	CO1	5
3.	W3	The vendor provide a rubber material that provides the protection to the edges of rectangular object and paper material for protection of front and back of rectangular object. So a vendor needs a application to calculate the amount of rubber and paper material required for covering rectangular object. Create an application with the methods to calculate area and perimeter on provision of the length and breadth of rectangular object to get the exact amount of rubber and paper required. Also calculate the the paper required to cover square object using method overloading.	CO2	3
4.	W4	An electrical engineer needs a complex number calculator for performing the operation of addition of alternating current represented using complex number. Create an application that takes two complex numbers as parameters and returns the resulting complex number, which is the addition of two complex numbers.	CO2	3
5.	W5	Programmers can define their own packages to bundle group of related classes/interfaces as per their requirement. Create your own package named as your "first name" and encapsulate the Product class in it for recording and displaying the product information. Also import the Product class to demonstrate the use of packages.	CO2	4
6	W6	2D array is used in many real-life applications where we need to organize data in tabular/matrix format. Hence a matrix manipulator is required	CO3	4

		with functionality of reading, displaying and flipping data from the matrix. Generate the methods, for the functionality mentioned above for creation the matrix manipulator.		
7.	W7	A character sequence is to be read as an input and result need to declare as "yes" or "no" by investigating the fact that traversing the characters sequence backwards and forwards results in same sequence. Write a program for the same using StringBuffer.	CO3	3
8.	W8	Admission section needs an application for storing the student information whenever new student come for admission. Information such as name, roll no, age, contact no. etc. is required to be stored for each student and at the end of the day the list need to be displayed on screen for verification. Make use of the vector class method for adding new student record.	CO3	3
9.	W9	Write a program to calculate volume of sphere using multilevel inheritance. The base class method will accept the radius from user. A class will be derived from the above-mentioned class that will have a method to find the area of a circle derived from this will have method to calculate and display the volume of the sphere.	CO4	4
10.	W10	The purpose of an abstract class is to define some common behaviour that can be inherited by multiple subclasses, without implementing the entire class. Create an abstract class with common behaviour of different shapes.	CO4	3
11.	W11	Consider a university where students who participate in the National Games or Olympics are given some grace marks. The grace marks provided are fixed and same for every student. Create an application that keeps student's academic marks and Sports grace marks separate and generate total of marks considering academics and sports both. Also invoke methods of base class & interface using reference. (Hint: Make use of Interface).	CO4	3
12.	W12	Through Custom exception user can raise application-specific error code. You are required to calculate a square of even number provided as input by user. However, if a user provides an odd number as input, then an exception must be thrown explicitly with message indicating the input number must be even number.	CO5	5

13.	W13	Divide your program into two parts: One to read a number and the other to calculate its square. Provide a simultaneous execution of both parts of a program to maximum utilize the CPU time.	CO5	5
14.	W14	Create a GUI application that runs in a browser and displays the name and roll no of student provided as specific attribute for displaying on browser.	CO6	5
15.	W15	You need to create an interactive GUI based form for reading student information such as name, date of birth, gender, department, contact no., address etc. using AWT components and handle events such as mouse movement or button click to store information.	CO6	5

CO-PO & PSO Mapping

<u>Mapping of Course outcomes with Program Outcomes:</u>

Subject Weight	**Course Outcomes**		**Contribution to Program outcomes**											
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
PRATICAL 70%	CO1	To apply fundamental programming constructs to understand object-oriented programming concepts.	1	3	3	2								1
	CO2	To illustrate the concept of packages, classes and objects.	2	2	3		1		2					
	CO3	To elaborate the concept of strings, arrays and vectors.	2	2	3		1		2					
	CO4	To understand and demonstrate the concept of inheritance and interfaces.		2	2	1	1		2					2
	CO5	To interpret and apply the notion of exception handling and multithreading.		2	2				2				2	2
	CO6	To develop GUI based application to understand event-		2	2		3				1		1	1

		based GUI handling												

Mapping of Course outcomes with Program Specific Outcomes:

Course Outcomes		Contribution to Program Specific outcomes		
		PSO1	PSO2	PSO3
CO1	To apply fundamental programming constructs to understand object-oriented programming concepts.	3	2	1
CO2	To illustrate the concept of packages, classes and objects.	3	2	2
CO3	To elaborate the concept of strings, arrays and vectors.	2	2	1
CO4	To understand and demonstrate the concept of inheritance and interfaces.	3	2	2
CO5	To interpret and apply the notion of exception handling and multithreading.	2	2	3
CO6	To develop GUI based application to understand event-based GUI handling	1	2	3

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
CSL304	Skill base Lab course: Object Oriented Programming with Java	02	02			02	--	02

Course Code	Course Name	Examination Scheme		
		Term Work	Practical & Oral	Total
CSL304	Skill base Lab course: Object Oriented Programming with Java	50	25	75

Term Work:

1. Students will submit term work in the form of journal that will include:

- At least 16-18 programs and mini project
- Two assignments covering whole syllabus

2. 50 Marks (Total Marks) = 20 marks (Experiments) + 20 marks (Mini Project) + 05 marks (Assignments) + 05 marks (Attendance).

Practical & Oral: Examination will be based on suggested practical list and entire syllabus.

OOPM JAVA LAB

Experiment No. : 1

Demonstration of Branching Statement

Experiment No.1

1. Aim: The grading system describes how well students have achieved the learning objectives or goals established for a class of course of study. This system helps to categorize the students according to their grades. Design a system that reads marks obtained by a student in a test of 100 marks and assign the grade as per the following criteria.

Marks	Grade
0 to 39	Fail
40 to 49	Pass
50 to 59	Second Class
60 to 69	First Class
70 to 100	Distinction

2. Objectives:

- To understand the concept of Object Oriented Programming.
- To understand the switch case statement

3. Outcomes:

To apply fundamental programming constructs to understand object oriented programming concepts.

4. Hardware / Software Required : JDK 1.5.0 onwards

5. Theory: The control statement that allows us to make a decision from the number of choices is called a **switch** .A switch statement allows you to test the value of an expression and, **depending** on that value, to jump directly to some location within the switch statement. Only expressions of certain types can be used. The value of the expression can be one of the primitive integer types int, short, or byte. It can be the primitive char type.

The positions that you can jump to are marked with case labels that take the form: "case constant:" This marks the position the computer jumps to when the expression evaluates to the given constant. As the final case in a switch statement you can, optionally, use the label "default:", which provides a default jump point that is used when the value of the expression is not listed in any case label.

//Syntax of switch statement :

switch (expression)

```
{
case constant1 : statement ;

break ;
case constant2 : statement ;

break;
case constant3 : statement ;

break;
default :                    statement
}
```

Sample Program :

```
class sample
{
 public static void main( String args[])
                {
int i = 1 ;
switch ( i )

{
case 1 : System.out.println( "I am in case 1") ; break;

case 2 : System.out.println ( "I am in case 2") ; break;

case 3 : System.out.println ( "I am in case 3") ; break;

default: System.out.println ( "I am in default case") ;

}
}
```

6. Algorithm:

Step 1: Initialize marks of student (between 0 to 100)

Step 2: Using switch case display grade of the student

Step 3: switch (marks/10)

Step 3.1:print the class of students according to 10 different cases.

Step 3.2 Case 0,1,2,3 : Fail

Step 3.3 Case 4 : Pass

Step 3.4 Case 5 : Second Class

Step 3.5 Case 6 : First Class

Step 3.6 Case 7,8,9,10 : Distinction

Step 4 Stop

7. **Conclusion** :

After performing this experiment we are able to use branching structures, control structure and switch case which helps in making decision by using switch cases.

8. **Viva Questions:**

- Explain switch case statement?
- What is use of default label in switch case?

9. **References:**

- Jaime Nino, Frederick A. Hosch, *‘An introduction to Programming and Object Oriented Design using Java’,* Wiley Student Edition.
- E Balgurusamy, *“Programming with JAVA”*, Tata McGraw Hill

OOPM JAVA LAB

Experiment No. : 2

Demonstration of Looping Statement

Experiment No.2

1. Aim: Prime numbers are important because the security of many encryption algorithms are based on the fact that it is very fast to multiply two large numbers and get the result, while it is extremely computer-intensive to do the reverse. Enlist all the prime numbers between 1 to 1000 to create a base for cryptography.

2. Objectives: From this experiment, the student will be able to

- Understand the basics of Control Statements
- Use the Selection (if ..else) statements and Repetition(for loop) statement.

3. Outcomes: To apply fundamental programming constructs to understand object oriented programming concepts.

4. Hardware / Software Required : Ubuntu, Java

5. Theory:

Control Statements

The control statement are used to controll the flow of execution of the program . This execution order depends on the supplied data values and the conditional logic. Java contains the following types of control statements:

1- Selection Statements

2- Repetition Statements

3- Branching Statements

Selection statements:

1. **If Statement:**

This is a control statement to execute a single statement or a block of code, when the given condition is true and if it is false then it skips **if** block and rest code of program is executed .

Syntax:

```
 if(conditional_expression){
 <statements>;
 ...;
 ...;
}
```

2. **If-else Statement:**

The **"if-else"** statement is an extension of if statement that provides another option when 'if' statement evaluates to "false" i.e. else block is executed if **"if"** statement is false.

Syntax:

```
  if(conditional_expression){
```

```
<statements>;
...;
...;
}
 else{
<statements>;
....;
....;
 }
```

6. Algorithm:

```
Step 1: Initialize i=1, num=1000, flag=1
Step 2:  If i<=num
Step 2.1: Initialize j=2
Step 2.2: If j<i do
Step 2.2.1: If i%j==0
Step 2.2.1.1: Set flag=0
Step 2.2.1.2: break
Step 2.2.2: Increment j
Step 2.3: if flag=1
Step 2.3.1: Display i
Step 2.4: Increment i
Step 3: STOP
```

7. Conclusion:

After performing this experiment we are able to create a class and control statements.

8. Viva Questions:

- What is a control statement?
- What are the types of the Control statements?

9. References:

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 3

Demonstration of Class, Objects, Methods and Method Overloading

Experiment No.3

1. **Aim:** The vendor provide a rubber material that provides the protection to the edges of rectangular object and paper material for protection of front and back of rectangular object. So a vendor needs a application to calculate the amount of rubber and paper material required for covering rectangular object. Create an application with the methods to calculate area and perimeter on provision of the length and breadth of rectangular object to get the exact amount of rubber and paper required. Also calculate the the paper required to cover square object using method overloading.

2. **Objectives:** From this experiment, the student will be able to
- Understand basics of Object Oriented programming
- To solve the real world scenarios using top down approach.

3. **Outcomes:** To illustrate the concept of packages, classes and objects.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

Objects are the basic runtime entities in an object oriented system. A class may be thought of as a 'data type' and an object as 'variable' of that data type. The data and code of the class are accessed by object of that class.
In this program, the object of class circle is used to access the methods: getdata() & area() of this class.
Syntax:-

```
class Class_Name
    {
        public static void main(String args[])
         {
                  // Statements;
          }
    }
```

For example:-

```
class Room
    {
       float length;
        float breadth;
        void getdata(float a, float b)
        {
            length=a;
            breadth=b;
```

```
                    }
                   public static void main(String args[])
                   {
                      float area;
                       Room r1=new Room();
                       R1.getdata(20,10);
                     area=r1.length*r1.breadth;
                     System.out.println("Area= "+area);
            }
}
```

Output:-
Area=200.0

Java allows to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading.
Method Overloading:
- Two or more methods within the same class that have the same name but methods must differ in the type and/or number of their parameters.
- Method overloading is one of the ways that Java implements polymorphism
- When an overloaded method is invoked, Java uses the type and/or number of arguments to determine which version of the overloaded method to actually call.

```
  class ClassName
  {
       void Function1(data_type a, data_type b)
        {
           // Statements

         }
       void Function1(data_type a, data_type b, data_type c)
        {
           // Statements
         }
       public static void main(String args[])
        {
            ClassName obj=new ClassName();
             obj.Function1(Value1,Value2);
             obj.Function1(Value1,Value2,Value3);
           }
}
```

For example:-

```
  class  Calculation
   {
    void sum(int a,int b)
     {
System.out.println("Addition of two numbers is:"+(a+b));
 }
void sum(int a, int b, int c)
 {
   System.out.println("Addition of three numbers is:"+(a+b+c));
}
public static void main(String args[])
{
Calculation c=new Calculation();
c.sum(10,40,60);
c.sum(20,20);
}
}
```

Output:-

```
Addition of two numbers is: 110
Addition of three numbers is: 40
```

6. **Algorithm**:

```
Step 1.    Start.
Step 2.    Declare a class named Rectangle.
Step 3.    Declare data members length and breadth.
Step 4.    Define a method getDim() to take input for data members
Step 5.    Define a method area() to calculate area of Rectangle
Step 6.    Declare a method perimeter() to calculate the perimeter of Rectangle
Step 7.    Overload a method area(int) to calculate area of Square
Step 8.    Create object of Rectangle class and invoke methods of Rectangle class inside main() method.
Step 9.    Stop.
```

7. **Conclusion:**

After performing this experiment we are able to create a class and object. The classes are interacting with each other by accessing the methods and members of class using different objects.

8. **Viva Questions:**

- What is class and object? How to access class using object?
- How two objects communicate?

9. **References:**

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Grady Booch, Object Oriented Analysis and Design
- Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
- Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 4

Passing and Returning object as an Argument

Experiment No.4

1. **Aim:** An electrical engineer needs a complex number calculator for performing the operation of addition of alternating current represented using complex number. Create an application that takes two complex numbers as parameters and returns the resulting complex number, which is the addition of two complex numbers.

2. **Objectives:** From this experiment, the student will be able to

- Solve the real world scenarios using top down approach.
- Understand the concept and properties of constructor.

3. **Outcomes:** To illustrate the concept of packages, classes and objects. .

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

A constructor is a special public member method whose task is to initialize the object data members when an object is declared. The constructor of a class should have the same name as the class. For example if the name of class is Employee then the constructor is a method with the name Employee( ).

Characteristics of a constructor are :

1. They should be declared in the public section.

2. They automatically get invoked when the objects are created.

3. They do not have return types not even void and therefore they cannot return values.

4. Like other methods they can have default arguments.

Types of Constructor:

1. Default Constructor: A constructor without arguments is called a default constructor. When no constructor is created explicitly then Java first implicitly creates a constructor without any parameter and invokes it. The default constructor then initializes the instance variables to default values, i.e. zero for numeric data types, null for string type and false for Boolean.

2. Parameterized Constructor: The constructor with arguments is called a parameterized constructor. In parameterized constructors the instance variable is initialized automatically at run time according to the values passed to parameters during the object creation.

Constructor overloading means we can create and use more than one constructor in a class. A constructor can be overloaded on the basis of following facts:

1. Number of parameters
2. Data type of parameters
3. Order of parameters

When we create object of the classes, the instance variables are initialized according to the constructor signature.

For example :

```
class Product
  {
      int x, y;
      Product( );    //Default Constructor
      Product(int a, int b )      //Parameterized Constructor
  }
```

6. Algorithm :

```
Step 1: Start
Step 2: Define class complex with
Step 2.1: data members real and imaginary
Step 2.2: default constructor to initialize real and imaginary to 0
Step 2.3: parameterized constructor to initialize real and imaginary
Step 2.4: void show() method to display complex number
Step 2.5: Complex sum(Complex, Complex) method to perform addition of two complex
numbers
Step 3: Define main() method
Step 3.1: Create object A, B and C of Complex class
Step 3.2: Make a call to sum() and show() method
Step 4: Stop
```

7. Conclusion:

This experiment implements constructors and give us the clear scenario of characteristics and types of constructor along with parameters passing to constructor. We can formulate and solve real world problems.

8. Viva Questions:

- What is constructor?
- What is garbage collection?
- Enlist different types of constructors.

9. References:

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Grady Booch, Object Oriented Analysis and Design
- Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
- Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 5

Create and Import User-Defined Package

Experiment No.5

1. **Aim:** Programmers can define their own packages to bundle group of related classes/interfaces as per their requirement. Create your own package named as your "first name" and encapsulate the Product class in it for recording and displaying the product information. Also import the Product class to demonstrate the use of packages.

2. **Objectives:** From this experiment, the student will be able to

- Study java constructs.
- Study and implement packages

3. **Outcomes:** To illustrate the concept of packages, classes and objects.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

Benefits of Packages

- Packages allow us to organize classes into smaller units and make it easy to locate and use the appropriate class file.
- It helps to avoid naming conflict.
- Packages protect classes, data and methods.

Creating a Package

To create a package, you choose a name for the package and put a package statement with that name at the top of every source file that contains the types (classes, interfaces) that you want to include in the package.

The package statement for example, package graphics; must be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

Package FirstPackage

Public class Rectangle

```
{
        // class body
}
```

This file would be stored as A.java in a directory named FirstPackage. If we do not use a package statement, then that class is stored in default package (unnamed package). Generally speaking, an unnamed package is only for small or temporary applications or when you are just beginning the development process. Otherwise, classes and interfaces belong in named packages.

Using Package Members

The types that comprise a package are known as the package members.

To use a public package member from outside its package, you must do one of the following:

- Refer to the member by its fully qualified name
- Import the package member
- Import the member's entire package

Importing a Package Member

To import a specific member into the current file, put an import statement at the beginning of the file before any type definitions but after the package statement, if there is one.

For example,

import FirstPackage.Rectangle;

Now you can refer to the Rectangle class by its simple name.

Rectangle myRectangle = new Rectangle();

myRectangle is an object of class Rectangle

To import an Entire Package with all the classes, we have to use the asterisk (*) wildcard character.

import FirstPackage.*;

6. Algorithm :

(Note: Algorithm depends on the package name and class name used in this experiment.)

7. Conclusion:

A package is collection of classes, interfaces, etc. In this experiment we have implemented user defined packages and understood access modifiers.

8. Viva Questions:

- What are benefits of packages?
- How to create user defined package?

9. References:

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Grady Booch, Object Oriented Analysis and Design
- Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
- Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 6

Demonstration of 2D- Array

Experiment No.6

1. **Aim:** 2D array is used in many real-life applications where we need to organize data in tabular/matrix format. Hence a matrix manipulator is required with functionality of reading, displaying and flipping data from the matrix. Generate the methods, for the functionality mentioned above for creation the matrix manipulator.

2. **Objectives:**
- To understand the concept of Object Oriented Programming.
- To understand how to use of Array in java

3. **Outcomes:** To elaborate the concept of strings, arrays and vectors.

4. **Hardware / Software Required :** JDK 1.5.0 onwards

5. **Theory**: Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;   // preferred way.
```

Or

```
dataType arrayRefVar[];  //  works but not preferred way.
```

Note: The style dataType[] arrayRefVar is preferred. The style dataType arrayRefVar[] comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example:

The following code snippets are examples of this syntax:

```
double[] myList;         // preferred way.
```

Or

double myList[];   // works but not preferred way.

Creating Arrays:

You can create an array by using the new operator with the following syntax:

arrayRefVar = new dataType[arraySize];

The above statement does two things:

It creates an array using new dataType[arraySize];

It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

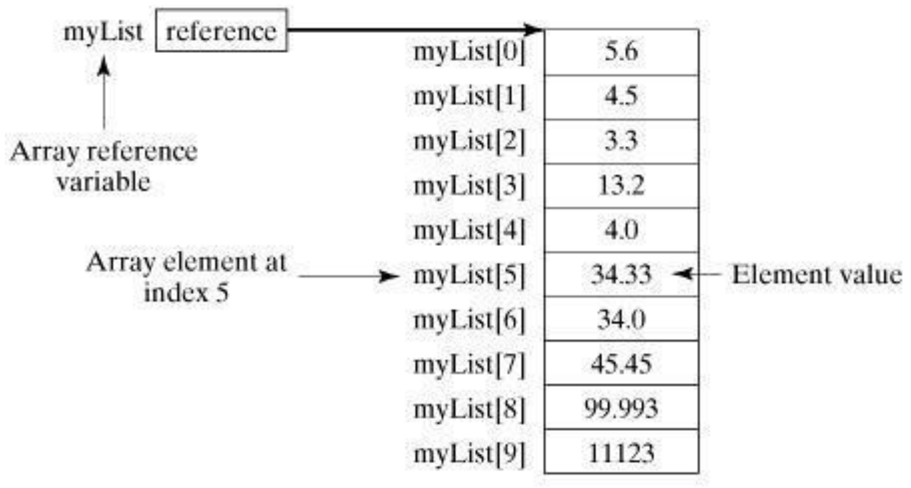
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type, and assigns its reference to myList.:

double[] myList = new double[10];

Following picture represents array myList. Here myList holds ten double values and the indices are from 0 to 9.



6. Algorithm:-

1. 1. Create a class Matrix and declare two 2D array for addition and transpose of matrix and variables r,c,i,j are used.

```
class Matrix
{
  int a[][],b[][],sum[][],t[][];
  int r,c,i,j;
```

2. Create a method read() and enter the elements for two matrices

```
 void read()
  {
     Scanner sc=new Scanner(System.in);
     System.out.println("Enter no. of rows and columns:");
     r=sc.nextInt();
     c=sc.nextInt();
                a =new int[r][c];
     b =new int[r][c];
     System.out.println("Enter First Matrix:");
     for(i=0;i<r;i++)
      {
       for(j=0;j<c;j++)
        {
         //System.out.print("Enter data:");
         a[i][j]=sc.nextInt();
        }
    }
   System.out.println("Enter Second Matrix:");
   for(i=0;i<r;i++)
    {
      for(j=0;j<c;j++)
       {
         //System.out.print("Enter data:");
         b[i][j]=sc.nextInt();
        }
    }
  } //end of read
```

3. Create a method addition() for the ddition of two matrices

```
void addition()
 {
   sum=new int[r][c];
   for(i=0;i<r;i++)
                          for(j=0;j<c;j++)
        sum[i][j]=a[i][j]+b[i][j];
  } //end of addition
```

4. Create another method display() to display the sum of matrices

```
void display()
 {
  for(i=0;i<r;i++)
    {
     for(j=0;j<c;j++)
       System.out.print(sum[i][j]+" ");
     System.out.println();
     }
 } //end of display
```

5. Create a method transpose() to transpose the matrix after the addition of two matrices.

```
void transpose()
 {
   t=new int[r][c];
   for(i=0;i<c;i++)
      for(j=0;j<r;j++)
         t[i][j]=sum[j][i];

   for(i=0;i<r;i++)
     {
      for(j=0;j<c;j++)
        System.out.print(t[i][j]+" ");
      System.out.println();
      }
  } //end of transpose

} //end of class Matrix
```

5. Stop

7. **Conclusion:-** This experiment implements transpose matrix, we also learnt how to create an array data structure which include 2Dimentional array.

8. Viva Questions:

- What is array?
- How to access array in java?

9. References:

- Jaime Nino, Frederick A. Hosch, ‘An introduction to Programming and Object Oriented Design using Java’, Wiley Student Edition.
- E Balgurusamy, “Programming with JAVA”, Tata McGraw Hill

OOPM JAVA LAB

Experiment No. : 7

Demonstration of StringBuffer methods

Experiment No.7

1. **Aim:** A character sequence is to be read as an input and result need to declare as "yes" or "no" by investigating the fact that traversing the characters sequence backwards and forwards results in same sequence. Write a program for the same using StringBuffer.

2. **Objectives:** From this experiment, the student will be able to
- Understand basic concepts in Object Oriented Programming
- Study different operations on strings and string Buffer class.

3. **Outcomes:** To elaborate the concept of strings, arrays and vectors.

4. **Hardware / Software Required :** JDK 1.5.0 onwards

5. **Theory:**

String manipulation is the most common part of many Java programs. Strings are very important in programming languages as they are used to process long textual/symbolic information. The information you provide is in form of ordered sequence of characters and symbols is called as string.

In c, c++ character arrays are used to process long textual information. Using character arrays is time consuming so in java, strings are not considered as the fundamental data type. In Java **String** class is provided to work with strings along with facility of character array.

In Java, Strings are immutable as once an object of the String class is created and initialized with some value then it cannot be changed. The Java development environment provides two classes that store and manipulate character data: **String**, for constant strings, and **StringBuffer**, for mutable strings.

Another way of making a string in java is making an object of Stringbuffer class. One major speciality of the object of this class is that it is mutable.

The size of this object can be changed again and again during runtime i.e. we can insert, append, delete etc a character from the string.

An object of StringBuffer has some advantages of extra methods provided in the class. Let us see methods supported in this class.

There are some StringBuffer methods given as below:-

- **append(String):** Append the string with original string.
- **insert(int,char):** This method is used to insert the character passed at indexed passed.
- **replace(int startIndex, int endIndex, String str):** This method is used to replace the string from specified startIndex and endIndex.
- **delete(int startIndex, int endIndex):** This method is used to delete the string from specified startIndex and endIndex.
- **reverse():** Rreverses the Original string.
- **capacity():** This method is used to return the current capacity of the StringBuffer object.
- **char charAt(int index):** This method is used to return the character at the specified position.
- **setcharAt(int,index):** The character passed is set at the index specified in the brackets.
- **length():** This method is used to return the length of the string i.e. total number of characters.
- **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

Defining string with String class :

1. String str = new String ( );
   str = "Hello";
2. String str = "Hi";

Defining string with StringBuffer class :

StringBuffer str = new StringBuffer("Hello");

Defining array of strings :

String array[] ={"str1", "str2", "str3" , "str4"};

6. Algorithm :

```
Step 1: Start.
Step 2: Define class Palindrome.
Step 3: Declare 2 String variables str and rev.
Step 4: Take input from user for String str
Step 5: Declare StringBuffer variable str1 using String str as a parameter
Step 6: Reverse StringBuffer str1
Step 7: Convert StringBuffer str1 to String and store it in String rev
Step 8: if String str equals to String rev
```

Step 8.1: Display String str is a palindrome
Step 8.2: else display String str is not a palindrome
Step 9: Stop

7. Conclusion:

With this experiment we have learned difference between String and StringBuffer, different methods of String class and StringBuffer class. We have checked whether the given string is palindrome or not and also we are able to solve real time problems.

8. Viva Questions:

- What is difference between String and StringBuffer?
- What are the methods used in StringBuffer Class?

9. References:

1. Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw- Hill
2. Grady Booch, Object Oriented Analysis and Design
3. Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
4. Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 8

Demonstration of Vector

Experiment No.8

1. **Aim:** Admission section needs an application for storing the student information whenever new student come for admission. Information such as name, roll no, age, contact no. etc. is required to be stored for each student and at the end of the day the list need to be displayed on screen for verification. Make use of the vector class method for adding new student record.

2. **Objectives:**
- From this experiment, the student will be able to
- Understand basic concepts in Object Oriented Programming
- Study different operations on vector class and vector class Methods.

3. **Outcomes:** To elaborate the concept of strings , arrays and vectors.

4. **Hardware / Software Required :** JDK 1.5.0 onwards

5. **Theory:**

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 sizes over the lifetime of a program.

Following is the list of constructors provided by the vector class.

Sr.No.	Constructor & Description
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.

Apart from the methods inherited from its parent classes, Vector defines the following methods –

Sr.No.	Method & 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 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.
14	**void ensureCapacity(int minCapacity)** Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
15	**boolean equals(Object o)** Compares the specified Object with this vector for equality.
16	**Object firstElement()** Returns the first component (the item at index 0) of this vector.
17	**Object get(int index)** Returns the element at the specified position in this vector.
18	**int hashCode()** Returns the hash code value for this vector.
19	**int indexOf(Object elem)** Searches for the first occurence of the given argument, testing for equality using the equals method.
20	**int indexOf(Object elem, int index)** Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the equals method.
21	**void insertElementAt(Object obj, int index)** Inserts the specified object as a component in this vector at the specified index.

22	**boolean isEmpty()** Tests if this vector has no components.
23	**Object lastElement()** Returns the last component of the vector.
24	**int lastIndexOf(Object elem)** Returns the index of the last occurrence of the specified object in this vector.
25	**int lastIndexOf(Object elem, int index)** Searches backwards for the specified object, starting from the specified index, and returns an index to it.
26	**Object remove(int index)** Removes the element at the specified position in this vector.
27	**boolean remove(Object o)** Removes the first occurrence of the specified element in this vector, If the vector does not contain the element, it is unchanged.
28	**boolean removeAll(Collection c)** Removes from this vector all of its elements that are contained in the specified Collection.
29	**void removeAllElements()** Removes all components from this vector and sets its size to zero.
30	**boolean removeElement(Object obj)** Removes the first (lowest-indexed) occurrence of the argument from this vector.
31	**void removeElementAt(int index)** removeElementAt(int index).
32	**protected void removeRange(int fromIndex, int toIndex)** Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
33	**boolean retainAll(Collection c)** Retains only the elements in this vector that are contained in the specified Collection.
34	**Object set(int index, Object element)** Replaces the element at the specified position in this vector with the specified element.
35	**void setElementAt(Object obj, int index)** Sets the component at the specified index of this vector to be the specified object.
36	**void setSize(int newSize)** Sets the size of this vector.
37	**int size()** Returns the number of components in this vector.
38	**List subList(int fromIndex, int toIndex)**

	Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.
39	**Object[] toArray()** Returns an array containing all of the elements in this vector in the correct order.
40	**Object[] toArray(Object[] a)** Returns an array containing all of the elements in this vector in the correct order; the runtime type of the returned array is that of the specified array.
41	**String toString()** Returns a string representation of this vector, containing the String representation of each element.
42	**void trimToSize()** Trims the capacity of this vector to be the vector's current size.

6. Algorithm :

Step 1. Start.

Step 2. Define class Student.

Step 3. Declare a String variables name and integer variables rno, contactNo, age.

Step 4. Declare a method getData() which will take data from user using scanner class and assign it to declared variables.

Step 5. in Main class Create a vector object v

Step 6. Declare integer I and choice variable

Step 7. Accept the choice from user and assign it to choice variable

Step 7.1. choice no 1: create object S of student class, call getData() method using the object S, add S object to the vector v using vector class method.

Step 7.2. choice no 2: display student details i.e the element of vector v at that index position using vector class method.

(e.g System.out.println(((Student) v.elementAt(i)).rno());

Step 8. Repeat step 7 if u want to continue

Step 9. Stop

7. Conclusion:

With this experiment we have learned vector class and vector class methods, and how to add new student details and display student details using vector class.

8. Viva Questions:

- What is vector class and vector class methods?

9. References:

1. Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw- Hill
2. Grady Booch, Object Oriented Analysis and Design
3. Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
4. Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 9

Demonstration of Multilevel Inheritance

Experiment No.9

1. **Aim:** Write a program to calculate volume of sphere using multilevel inheritance. The base class method will accept the radius from user. A class will be derived from the above-mentioned class that will have a method to find the area of a circle derived from this will have method to calculate and display the volume of the sphere.

2. **Objectives:**
- To understand the concept of inheritance.
- To understand how to use method overriding.

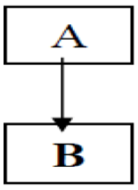
3. **Outcomes:** To understand and demonstrate the concept of inheritance and interfaces.

4. **Hardware / Software Required : JDK 1.5.0 onwards.**

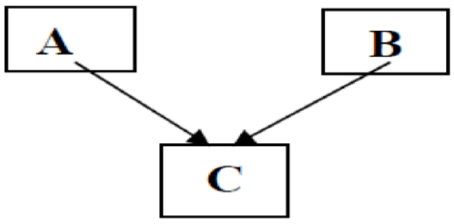
5. **Theory:**

Types of Inheritance:

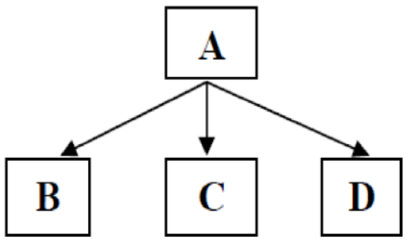
1. Single Inheritance: A derived class with only one base class is called as Single Inheritance.



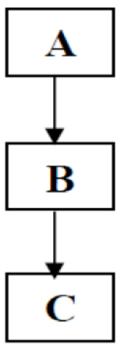
2. Multiple Inheritances: A derived class with several base classes is called Multiple Inheritance.



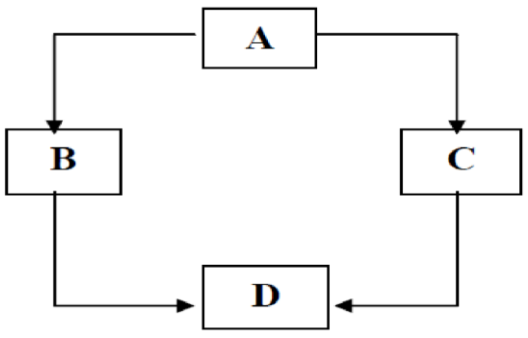
3. Hierarchical Inheritance: More than one class may inherit the features of one class this process is called as Hierarchical Inheritance**.**



4. Multilevel Inheritance: The mechanism of deriving a class from another derived class is called Multilevel Inheritance.



5. Hybrid Inheritance: There could be situations where we need to apply one or more types of inheritances to design a program this process is called Hybrid Inheritance.



```
class  A
{
}
class  B extends A
{
 }
class  C extends B
{
}
```

Overriding and Hiding Methods

Instance Methods

An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.

The ability of a subclass to override a method allows a class to inherit from a superclass whose behavior is "close enough" and then to modify behavior as needed. The overriding method has the same name, number and type of parameters, and return type as the method it overrides. An overriding method can also return a subtype of the type returned by the overridden method. This is called a *covariant return type*.

When overriding a method, you might want to use the `@Override` annotation that instructs the compiler that you intend to override a method in the superclass. If, for some reason, the compiler detects that the method does not exist in one of the superclasses, it will generate an error.

Class Methods

If a subclass defines a class method with the same signature as a class method in the superclass, the method in the subclass *hides* the one in the superclass.

The distinction between hiding and overriding has important implications. The version of the overridden method that gets invoked is the one in the subclass. The version of the hidden method that gets invoked depends on whether it is invoked from the superclass or the subclass. Let's look at an example that contains two classes. The first is `Animal`, which contains one instance method and one class method:

```
public class Animal {
   public static void testClassMethod() {
      System.out.println("The class" + " method in Animal.");
   }
   public void testInstanceMethod() {
      System.out.println("The instance " + " method in Animal.");
   }
}
```

The second class, a subclass of `Animal`, is called `Cat`:

```
public class Cat extends Animal {
   public static void testClassMethod() {
      System.out.println("The class method" + " in Cat.");
   }
   public void testInstanceMethod() {
      System.out.println("The instance method" + " in Cat.");
   }

   public static void main(String[] args) {
      Cat myCat = new Cat();
      Animal myAnimal = myCat;
      Animal.testClassMethod();
      myAnimal.testInstanceMethod();
   }
}
```

The `Cat` class overrides the instance method in `Animal` and hides the class method in `Animal`. The `main` method in this class creates an instance of `Cat` and calls `testClassMethod()` on the class and `testInstanceMethod()` on the instance.

The output from this program is as follows:

```
The class method in Animal.
The instance method in Cat.
```

6. Algorithm

Step 1. Start

Step 2. Create base class Circle with method to accept radius from user.

Step 3. Derive class Area from the base class with methods calculate() to calculate area and display() to display area.

Step 4. Derive another class Volume from Area which will calculate volume of sphere and have method display () to display the volume.

Step 5. Stop.

Here, the method display () of the derived class 'Volume' overrides the method display () of the class 'Area'.

7. Conclusion:

From this experiment we have learnt how to inherit the property of base class. Also learnt how to perform hiding and overriding concept.

8. Viva Questions:

- What is multilevel inheritance?
- What is method overriding?

9. References:

● Jaime Nino, Frederick A. Hosch, *'An introduction to Programming and Object Oriented Design using Java'*, Wiley Student Edition.
● E Balgurusamy, *"Programming with JAVA"*, Tata McGraw Hill

OOPM JAVA LAB

Experiment No. : 10

Demonstration of Abstract Class

Experiment No.10

1. **Aim:** The purpose of an abstract class is to define some common behaviour that can be inherited by multiple subclasses, without implementing the entire class. Create an abstract class with common behaviour of different shapes.

2. **Objectives:**
- To understand the concept of inheritance.
- To understand how to use method overriding.

3. **Outcomes:** To understand and demonstrate the concept of inheritance and interfaces.

4. **Hardware / Software Required : JDK 1.5.0 onwards**

5. **Theory:**

Abstract class in Java:
A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction in Java:
Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

There are two ways to achieve abstraction in java
- Abstract class (0 to 100%)
- Interface (100%)

Example abstract class:

abstract class A{}
abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

abstract void printStatus();//no body and abstract
Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
Bike obj = new Honda4();
 obj.run();
}
}
Output: running safely..
```

6. **Algorithm :**

Step 1. Start

Step 2. Create abstract class Shape

Step 3. Declare protected variable float r, l, b, area.

Step 4. Declare a function calculate() prototype which is of type public abstract void

Step 5. Declare a method display() to display the area.

Step 6. Create a child class Circle from class Base by using extends.

Step 7. Create two methods read() to read the radius and calculate() to calculate area of circle inside Circle class.

Step 8. Create another child class Rectangle and Triangle from class Base by using extends with the two methods read() to read the length and breadth and calculate() to calculate area of rectangle.

Step 9. Create another child class Triangle from class Base by using extends with the two methods read() to read the height and base and calculate() to calculate area of triangle.

Step 10. Define class main, create an object of class Circle, Rectangle and triangle. Call the read() and calculate() method using these object.

Step 11. Stop

7. **Conclusion:**

With this experiment we have understood that what is abstraction and how to create an abstract class and method.

8. **Viva Questions:**

- What is abstract class?
- What is abstract method?

9. **References:**

- Jaime Nino, Frederick A. Hosch, *‘An introduction to Programming and Object Oriented Design using Java’,* Wiley Student Edition.
- E Balgurusamy, *“Programming with JAVA”*, Tata McGraw Hill

OOPM JAVA LAB

Experiment No. : 11

Demonstration of Interface

Experiment No.11

1. **Aim:** Consider a university where students who participate in the National Games or Olympics are given some grace marks. The grace marks provided are fixed and same for every student. Create an application that keeps student's academic marks and Sports grace marks separate and generate total of marks considering academics and sports both.
Also invoke methods of base class & interface using reference.

2. **Objectives:** From this experiment, the student will be able to
- Study feature of OOP like inheritance.
- Study implementation of multiple inheritance in Java

3. **Outcomes:** To understand and demonstrate the concept of inheritance and interfaces.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

The mechanism of inheriting the features of more than one base class into a single class is known as multiple inheritances. Java does not support multiple inheritance but the multiple inheritance can be achieved by using the **interface.**

In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interface in a class. An interface is a group of related methods with empty bodies. Interfaces define only abstract methods and final fields. Therefore it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

Syntax:

```
interface InterfaceName

{

return_type method_name1(parameter-list);

data_type variable_name =value;

}
```

6. **Algorithm :**

Step 1. Start

Step 2. Create class Student

2.1 Declare roll_no as int type

2.2 Write getnumber() method to get roll number of a student

2.3 Write putnumber() method to print roll number of a student

Step 3. Define class test which extends student

3.1 Declare sem1,sem2 as float type

3.2 Write getmarks() method to get marks of a student

3.3 Write putmarks() method to print marks of a student

Step 4. Define interface sports

4.1 Declare score of float type

4.2 declare putscore();

Step 5. Define class Result which extends test & implements sports

5.1 Declare total of type float

5.2 Write display() method; Calculate total of sem1,sem2 & score

5.2.2 Invoke putnumber() ,putmarks(),putscore() methods

5.2.3 Print total

5.3 Write putscore() method to display score

Step 6. Define class Hybrid

Step 7. Define Main method, Create object of class Result and Invoke getnumber(), getmarks(), display().

Step 8. Stop

7. Conclusion:

With this experiment we have understood that Interfaces are mainly used to provide polymorphic behaviour and its function to break up the complex designs and clear the dependencies between objects.

8. Viva Questions:

- What is interface?
- How multiple inheritance can be implemented in java?
- What are advantages and disadvantages of interface?

9. References:

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Grady Booch, Object Oriented Analysis and Design
- Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
- Java 2, *"The Complete Reference of Java*", Schildt publication

OOPM JAVA LAB

Experiment No. : 12

Create User-Defined Exception

Experiment No.12

1. **Aim:** Through Custom exception user can raise application-specific error code. You are required to calculate a square of even number provided as input by user. However, if a user provides an odd number as input, then an exception must be thrown explicitly with message indicating the input number must be even number.

2. **Objectives:** From this experiment, the student will be able to

- Study feature of OOP like exception handing.
- Study how to create own exception in Java.

3. **Outcomes:** To interpret and apply the notion of exception handling and multithreading.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

Exceptions are usually used to denote something unusual that does not conform to the standard rules. In programming, exceptions are events that arise due to the occurrence of unexpected behaviour in certain statements, disrupting the normal execution of a program. Exception is a run time error. Exceptions can arise due to a number of situations. For example,

- Trying to access the 11th element of an array when the array contains of only 10 element (*ArrayIndexOutOfBoundsException*)
- Division by zero (*ArithmeticException*)
- Accessing a file which is not present (*FileNotFoundException*)
- Failure of I/O operations (*IOException*)
- Illegal usage of null. (*NullPointerException*)

Top class in exception hierarchy is *Throwable.* This class has two siblings: Error and Exception. All the classes representing exceptional conditions are subclasses of the Exception class.

The exception handling mechanism consists of following elements:

- **try/catch**: All the statements to be tried for exceptions are put in a try block. The catch block is used to catch any exception raised from the try block. If exception occurs in any statement in the try block control immediately passes to the corresponding catch block.
- **finally**: The finally block will execute whether or not an exception is thrown.
- **throw**: It is used to explicitly throw an exception. It is useful when we want to throw a user-defined exception.
- **throws**: If a method is capable of causing an exception that it does not handle, it must specify this behavior using throws keyword so that callers of the method can guard themselves against that exception

6. Algorithm :

```
Step 1. Start
Step 2. Create class OddException which extends Exception class
Step 3. Declare num as int type
Step 4. Write OddException(int x) constructor to get numbers as input.
Step 5. Write toString() method to display a message when exception is thrown.
Step 6. Create a class MyExceptionDemo with main function and Write static function OddNoException() throws OddException when number is odd.
Step 7. Defime main method
Step 8. In try block
Step 9. Invoke OddException(3)
Step 10.    In catch block
Step 11.    Print catch exception
Step 12.    Stop
```

7. Conclusion:

With this experiment we implemented exception handling mechanism and understood how to create and handle user defined exception.

8. Viva Questions:

- Define exception.
- Differentiate between error and exception.
- What is meant by the runtime exception.
- What are different types of exceptions.
- Explain role of try catch.
- What is use of finally block and when it gets executed.
- How to create user defined exception.

9. References:

- Java 2, *"The Complete Reference of Java*", Schildt publication.
- Ivor Horton, 'Beginning JAVA', Wiley India.
- DietalandDietal, 'Java: How to Program', 8/e,PHI
- 'JAVA Programming', Black Book, Dreamtech Press.

OOPM JAVA LAB

Experiment No. : 13

Demonstration of Multithreading

Experiment No.13

1. Aim: Divide your program into two parts: One to read a number and the other to calculate its square. Provide a simultaneous execution of both parts of a program to maximum utilize the CPU time.

2. Objectives: From this experiment, the student will be able to

- Study basic constructs of java
- Understand importance of multithreading

3. Outcomes: To interpret and apply the notion of exception handling and multithreading.

4. Hardware / Software Required : Ubuntu, Java

5. Theory:

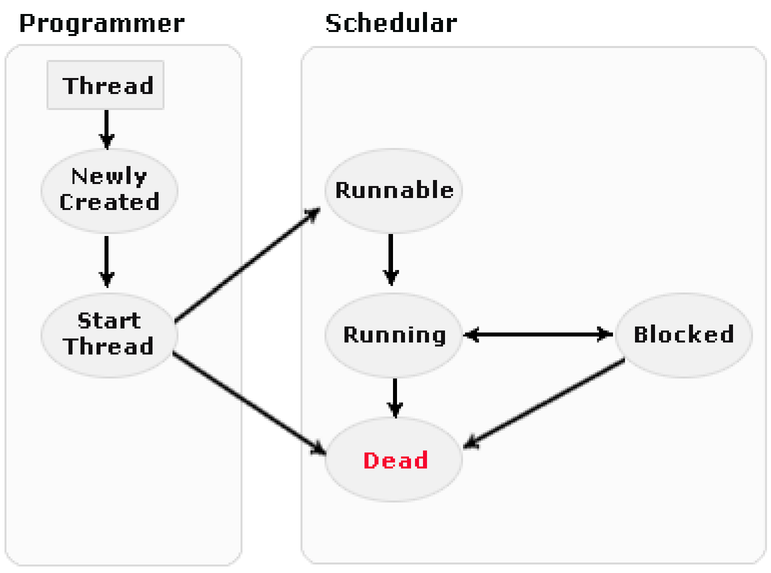
Processes and Threads

In concurrent programming, there are two basic units of execution: processes and threads. In the Java programming language, concurrent programming is mostly concerned with threads.

Processes : A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.

Threads : Threads are sometimes called lightweight processes. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process. Threads exist within a process — every process has at least one thread. Threads share the process's resources, including memory and open files. Multithreaded execution is an essential feature of the Java platform. Every application has at least one thread — or several, if you count "system" threads that do things like memory management and signal handling. But from the application programmer's point of view, you start with just one thread, called the main thread.

Different states of a thread are :



New state – After the creations of Thread instance the thread is in this state but before the start() method invocation. At this point, the thread is considered not alive.

Runnable (Ready-to-run) state – A thread start its life from Runnable state. A thread first enters runnable state after the invoking of start() method but a thread can return to this state after either running, waiting, sleeping or coming back from blocked state also. On this state a thread is waiting for a turn on the processor.

Running state – A thread is in running state that means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in Running state: the scheduler select a thread from runnable pool.

Dead state – A thread can be considered dead when its run() method completes. If any thread comes on this state that means it cannot ever run again.

Blocked - A thread can enter in this state because of waiting the resources that are hold by another thread.

As we have seen different states that may be occur with the single thread. A running thread can enter to any non-runnable state, depending on the circumstances. A thread cannot enter directly to the running state from non-runnable state, firstly it goes to runnable state. Now lets understand the some non-runnable states which may be occur handling the multithreads.

Sleeping – On this state, the thread is still alive but it is not runnable, it might be return to runnable state later, if a particular event occurs. On this state a thread sleeps for a specified amount of time. You can use the method sleep( ) to stop the running state of a thread.

static void sleep(long millisecond) throws InterruptedException

Method	**Return Type**	**Description**
currentThread( )	Thread	Returns an object reference to the thread in which it is invoked.
getName( )	String	Retrieve the name of the thread object or instance.
start( )	Void	Start the thread by calling its run method.
run( )	Void	This method is the entry point to execute thread, like the main method for applications.
sleep( )	Void	Suspends a thread for a specified amount of time (in milliseconds).
isAlive( )	Boolean	This method is used to determine the thread is running or not.
activeCount( )	Int	This method returns the number of active threads in a particular thread group and all its subgroups.
interrupt( )	Void	The method interrupt the threads on which it is invoked.
yield( )	Void	By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
join( )	Void	This method and join(long millisec) Throws InterruptedException. These two methods are invoked on a thread. These are not returned until either the thread has completed or it is timed out respectively.

Waiting for Notification – A thread waits for notification from another thread. The thread sends back to runnable state after sending notification from another thread.

final void wait(long timeout) throws InterruptedException

final void wait(long timeout, int nanos) throws InterruptedException

final void wait() throws InterruptedException

Some Important Methods defined in java.lang.Thread are shown in the table:

Threads can be created in Java in the following ways :

- Extending the java.lang.Thread class
- Implementing the java.lang.Runnable Interface.

1. **Extending the java.lang.Thread class**

This is a simplest form of creating threads in a program. Consider the following code snippet.

```
public class ThreadExample extends Thread
        {
         public void run()
         {
         System.out.println("This is an example on extending thread class");
         }
         -----
         -----
}
```

In this code snippet, the ThreadExample class is created by extending the Thread class. The run( ) method of the Thread class is overridden by the ThreadExample class to provide the code to be executed by a thread.

In Java we cannot extend a class from more than one class. Therefore while creating a thread by extending the Thread class we cannot simultaneously extend any other class.

2. **Extending the java.lang.Runnable Interface**

Threads can also be created by implementing the Runnable interface of the java.lang package. This package allows to define a class that can implement the Runnable interface and extend any other class. Consider the following code snippet.

```
public class ThreadExample implements Runnable
                {
```

```
public void run()
        {
          System.out.println("This is an example on runnable interface");
        }
public static void main(String[] args)
        {
          ThreadExample thrd = new ThreadExample( );
          Thread T = new Theard(thrd);
        }
        }
```

In above code snippet, the ThreadExample class is declared by implementing Runnable interface and overriding the run( ) method in which it defines the code to be executed by a thread.

6. Algorithm :

```
1. Start
2. Define class which implements Runnable interface.
   2.1 Create Thread t1.
   2.2 Create run( ) method to print 1 to 10 numbers
       For( int i=1;i<=10;i++)
        Print i
3. Define class which implements Runnable interface.
   3.1 Create Thread t2.
   3.2 Create run() method to print it's square
       For(int i=1;i<=10;i++)
        Print i*i
4. Define class multithread
5. Define main method
6. Invoke t1.start( ) then t2.start( )
7. Stop
```

7. Conclusion:

In this experiment we have implemented multithreading which is helpful to improve performance of system.

8. Viva Questions:

- What is multithreading?
- What are advantages of multithreading?
- What is synchronization in multithreading?

9. References:

- Ralph Bravaco , Shai Simoson , "*Java Programing From the Group Up*" ,Tata McGraw-Hill
- Grady Booch, Object Oriented Analysis and Design
- Jaime Nino, Frederick A. Hosch, '*An introduction to Programming and Object Oriented Design using Java*', Wiley Student Edition. Wayne Tomasi, "*Electronics Communication Systems*", Pearson education, Fifth edition.
- Java 2, *"The Complete Reference of Java*", Schildt publication.

OOPM JAVA LAB

Experiment No. : 14

Demonstration of Applet- Parameter Passing

Experiment No.14

1. **Aim:** Create a GUI application that runs in a browser and displays the name and roll no of student provided as specific attribute for displaying on browser.

2. **Objectives:** From this experiment, the student will be able to
- Study the concept of Applet
- Understand various methods and states in Applet life cycle
- Understand how to create and run Applets.

3. **Outcomes:** To develop GUI based application to understand event-based GUI handling principles.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**
Applet is small java program that can be easily transported over the network from one computer to other. It is used in internet applications. Generally embedded in an html page, can be downloaded from the server and run on the client, so as to do a specific kind of job. An applet may move from one state to another depending upon a set of default behaviours inherited in the form of methods from 'Applet' class.

The Applet life cycle is demonstrated in following figure.

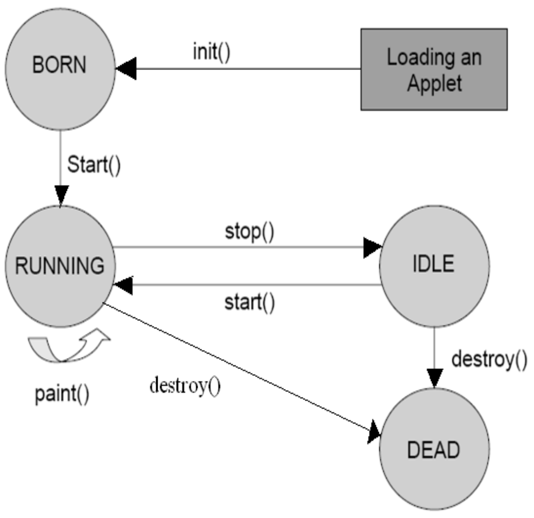


Figure: Applet Life Cycle

- **Born state**:- when applet is loaded ,Applet is in born state.

- **Running state**:- when start() method is invoked, applet state changes from born to running state. In this state, output is displayed, by invoking paint() method.
- **Idle state**:- when stop() method is invoked ie. If we minimize the applet window then applet state changes from running to idle state.
- **Dead state**:- when destroy() method is invoked ie. If we close the applet window then state changes from idle or running to dead.

The following are the methods in Applet life cycle:

Method	Meaning
init()	creates the objects needed by the applet; sets up initial values, load font and images or set up colors. called only once during the lifetime of on Applet.
start()	Applet moves to this phase automatically after the initialization state. If the applet is stopped or it goes to idle state, start() method must be called in order to force the applet again to the running state.
paint()	This method is called each time to draw and redraw the output of an applet.
stop()	Applet move to idle state, once it is stopped from running
destroy()	An applet goes to dead state when it is destroyed by invoking the destroy() method of Applet class. It results in complete removal of applet from the memory

The structure of Applet Tag is as follows:

< APPLET
[CODEBASE = *codebaseURL]*
CODE = *appletFile*
[ALT = *alternateText]*
[NAME = *appletInstanceName]*
WIDTH = *pixels*
HEIGHT = pixels
[ALIGN = *alignment]*
[VSPACE = *pixels]*
[HSPACE = pixels]
>
[< PARAM NAME = *AttributeName VALUE = AttributeValue>]*
[< PARAM NAME = *AttributeName2 VALUE = AttributeValue>]*
. . .
</APPLET>

Field	Meaning
CODEBASE	specifies the URL of the directory where the executable class file of the applet will be searched for.
CODE	It gives the name of the file containing the applet's compiled class file.

ALT	specifies the alternate short text message that should be displayed in case the browser recognizes the HTML tag but cannot actually run the applet because of some reason.
NAME	give a name to an applet’s instance
WISTH	gives the width of the applet display area in terms of pixels.
HEIGHT	gives the height of the applet display area in terms of pixels.
ALIGN	set the alignment of an applet
HSPACE	These are used to specify the space, in pixels, on each side of the applet
PARAM sub tag	provides information about parameters, or arguments, to be used by the Java applet. This tag has two attributes NAME: attribute name VALUE: value of the attribute named by corresponding PARAM NAME

6. Algorithm :

Step 1. Start
Step 2. Add applet tag with two param tags to pass name and roll number as an argument to applet
Step 3. Create class ParameterDemo which extends Applet class
a. Write paint() method to display applet
b. Invoke getParameter() method of Applet class to get the value of name and roll number from *value* attribute param tag.
c. Create object of Font class with necessary parameters to set the font.
d. Invoke setFont() method to set the font.
e. Create object of Color class with necessary parameters to set the colour of text.
f. Invoke setColor() method to set the colour.
g. Invoke drawstring() method to display output in applet window.
Step 4. Stop

7. Conclusion:

In this experiment we have implemented Applet with param tag. We have understood how to create an Applet and how to pass parameters to applet.

8. Viva Questions:

- What is Applet?
- How it is different from application program?
- What are the fields of Applet tag?
- Which are the states in Applet life cycle?
- Which are the methods of Applet class?

9. References:

- Java 2, “*The Complete Reference of Java*”, Schildt publication.
- Ivor Horton, ‘Beginning JAVA’, Wiley India.

- DietalandDietal, ‘Java: How to Program’, 8/e,PHI
- ‘JAVA Programming’, Black Book, Dreamtech Press.

OOPM JAVA LAB

Experiment No. : 15

Usage of AWT package for creating interactive GUI

Experiment No.15

1. **Aim:** You need to create an interactive GUI based form for reading student information such as name, date of birth, gender, department, contact no., address etc. using AWT components and handle events such as mouse movement or button click to store information.

2. **Objectives:** From this experiment, the student will be able to

- Create a frame and add buttons to it.
- Study the various methods of a button in order to handle events.

3. **Outcomes:** To develop GUI based application to understand event-based GUI handling principles.

4. **Hardware / Software Required :** Ubuntu, Java

5. **Theory:**

Java AWT: Abstract Window Toolkit is *an API to develop GUI or window-based applications* in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Frame: The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Java AWT Button: The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

Constructors for Button class	**Description**
Button()	Constructs a button without a label.
Button(String str)	Construct a button with the specific label.

Methods of the Button Class	**Description**
String getLabel()	Returns the label of a button.
void setLabel(String str)	Sets the specified string as a button label.
void addActionListener(ActionListener al)	Adds a specified listener to a component to receive the events from a button.
void removeActionListener(ActionListen al)	Removes an action listener from a component so that it no longer receives the events from a button.
String getActionCommand()	Returns the command of the event caused by a button.
String setActionCommand(string command)	Sets the command name for the action of an event caused by a button.

Event Handlers: The change in the state of an object is known as an event. For example, click on button, dragging mouse etc. The *java.awt.event* package provides many event classes and Listener interfaces for event handling.

Action Listener: Java AWT Listeners are a group of interfaces from java.awt.event package. Listeners are capable to handle the events generated by the components like button, choice, frame etc. These listeners are implemented to the class which requires handling of events.

ActionListener has only one method.

Method	**Description**
actionPerformed(actionEvent)	Called just after the user performs an action.

ActionEvent class:

Method	**Description**
String getActionCommand()	Returns the string associated with this action. Most objects that can fire action events support a method called setActionCommand that lets you set this string.
Object getSource()	Returns the object that fired the event.

Sample Program:

```
import java.awt.*;
import java.awt.event.*;
public class ButtonDemo extends Frame implements ActionListener
{
  Button rb, gb, bb;
  public ButtonDemo()
  {
    FlowLayout fl = new FlowLayout();
    setLayout(fl);
    rb = new Button("Red");
    gb = new Button("Green");
    bb = new Button("Blue");
    rb.addActionListener(this);
    gb.addActionListener(this);
    bb.addActionListener(this);
    add(rb);
    add(gb);
    add(bb);
    setTitle("Button in Action");
    setSize(300, 350);
    setVisible(true);
  }
  public void actionPerformed(ActionEvent e)
```

```
 {
  String str = e.getActionCommand();
      if(str.equals("Red"))
  {
    setBackground(Color.red);
  }
  else if(str.equals("Green"))
  {
    setBackground(Color.green);
  }
  else if(str.equals("Blue"))
  {
    setBackground(Color.blue);
  }
 }
 public static void main(String args[])
 {
  new ButtonDemo();
 }
}
```

6. Algorithm :

Step 1. Start.

Step 2. Create a frame in AWT by extending the Frame class and set its properties.

Step 3. Create button object and register this object as an action listener on the button, using the addActionListener method.

Step 4. Get the user input by *getActionCommand()* which invokes actionPerformed().

Step 5. Set the background color using Color class methods.

Step 6. Stop.

7. Conclusion:

In this experiment we have implemented event handling of button by changing the background color of a frame window in AWT.

8. Viva Questions:

- Why are event handlers used?
- How is an object registered as an action listener?
- How is a button with label created?
- Define the methods of actionEvent class.

9. References:

- "Object Oriented Programming Methodology", Radha Shankarmani, DreamTech Press.
- Java 2, *"The Complete Reference of Java*", Schildt publication.

- https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html
- https://www.javatpoint.com/event-handling-in-java