Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 01

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim:
A.Write a JAVA program to display default values of all primitive data types of JAVA.
B. Five Bikers Compete in a race such that they drive at a constant speed which may or may not be the same as the other. To qualify the race, the speed of a racer must be more than the average speed of all 5 racers. Take as input the speed of each racer and print back the speed of qualifying racers.

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Introduction to Java- Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

Java is:

Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machines, rather into platform independent bytecode. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.
Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
Architectural-neutral: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
Portable: Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset.
Robust: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multithreaded: With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
High Performance: With the use of Just-In-Time compilers, Java enables high performance.
Distributed: Java is designed for the distributed environment of the internet.
Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.
Tools you will need:
For developing a java program, you will need a Pentium 200-MHz computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).

You also will need the following software-

1) Ubuntu 14.04 (or any higher version) operating system.

2) Java JDK 8

3) Pico or any other open source text editor

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean.

Object - Objects have states and behaviours. Example: A dog has states - colour, name, breed as well as behaviours -wagging, barking, eating. An object is an instance of a class.
Class - A class can be defined as a template/ blueprint that describes the behaviours/states that object of its type support.
Methods - A method is basically a behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
Instance Variables - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.
Basic Syntax:
About Java programs, it is very important to keep in mind the following points.
Case Sensitivity - Java is case sensitive, which means identifier Hello and hello would have different meanings in Java.
Class Names - For all class names the first letter should be in Upper Case.
If several words are used to form a name of the class, each inner world's first letter should be in UpperCase.

Example- class MyFirstJavaClass

Method Names - All method names should start with a lower-case letter. If several words are used to form the name of the method, then each inner world's first letter should be in Upper Case.

Example- public void myMethodName()

Program File Name - Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile).
Example- Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'
public static void main(String args[]) - Java program processing starts from the main() method which is a mandatory part of every Java program.
Let us look at a simple code that would print the words Hello World.

```
public class MyFirstJavaProgram {

/* This is my first java program.
* This will print 'Hello World' as the output
*/
public static void main(String []args) {
        System.out.println("Hello World"); // prints Hello World
        }
}
```

Let's look at how to save the file, compile and run the program. Please follow the steps given below:

1) Open text editor and add the code as above.

2) Save the file as: MyFirstJavaProgram.java.

3) Open the terminal and go to the directory where you saved the class.
Assume it's student@gcoej-ThinkCentre-M70z:~$.

4) Type ' javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption: The path variable is set).

5) Now, type ' java MyFirstJavaProgram ' to run your program.

6) You will be able to see ' Hello World ' printed on the window.

```
student@gcoej-ThinkCentre-M70z:~$ javac MyFirstJavaProgram.java
student@gcoej-ThinkCentre-M70z:~$ java MyFirstJavaProgram

Hello World
```

Program:

A.

```
class demo
{
        static byte b;
        static short s;
        static int i;
        static long l;
        static float f;
        static double d;
        static char c;
        static boolean bl;
        public static void main(String[] args)
{
        System.out.println("The default values of primitive data types are:");
        System.out.println("Byte :"+b);
        System.out.println("Short :"+s);
        System.out.println("Int :"+i);
        System.out.println("Long :"+l);
        System.out.println("Float :"+f);
        System.out.println("Double :"+d);
        System.out.println("Char :"+c);
        System.out.println("Boolean :"+bl);
}
}
```

B.

```
import java.util.*;
class racedemo
        {
        public static void main(String[] args)
        {
          float s1,s2,s3,s4,s5,average;
        Scanner s = new Scanner(System.in);
        System.out.println("Enter speed of first racer:");
         s1 = s.nextFloat();
         System.out.println("Enter speed of second racer:");
         s2 = s.nextFloat();
         System.out.println("Enter speed of third racer:");
         s3 = s.nextFloat();
         System.out.println("Enter speed of fourth racer:");
         s4 = s.nextFloat();
        System.out.println("Enter speed of fifth racer:");
        s5 = s.nextFloat();
        average=(s1+s2+s3+s4+s5)/5;
        if(s1>average)
                System.out.println("First racer is qualify racer:");
                else if(s2>average)
                System.out.println("Second racer is qualify racer:");
                else if(s3>average)
                System.out.println("Third racer is qualify racer:");
                else if(s4>average)
                System.out.println("Fourth racer is qualify racer:");
                else if(s5>average)
                System.out.println("Fifth racer is qualify racer:");
}
}
```

Conclusion:

Name & Sign of Course Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering

Experiment No: 02

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim:

A.Write a java program to search for an element in a given list of elements using binary search mechanism
B. Write a java program to sort for an element in a given list of elements using bubble sort

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Binary Search:

- Search a sorted array by repeatedly dividing the search interval in half.
- Begin with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

program:

```
java.util.Scanner;
class binarysearch
        {
                public static void main(String args[])
        {
                int n, i, num,first, last, middle;
                int a[ ]=new int[20];
                Scanner s = new Scanner(System.in);
                System.out.println("Enter total number of elements:");
                n = s.nextInt();
                System.out.println("Enter elements in sorted order:");
                for (i = 0; i < n; i++)
                        a[i] = s.nextInt();
                        System.out.println("Enter the search value:");
                        num = s.nextInt();
                        first = 0;
                        last = n - 1;
                        middle = (first + last)/2;
                                while( first <= last )
                                        {
```

```
                                        if ( a[middle] < num )
                                        first = middle + 1;
                                        else if ( a[middle] == num )
                                        {
                                             System.out.println("number found");
                                             break;
                                             }
                                             else
                                             {
                                             last = middle - 1;
                                             }
                                             middle = (first + last)/2;
                                             }
                                             if ( first > last )
                                             System.out.println( " Number is not found");
        }
        }
```

Bubble sort:
Bubble sort is a simple sorting algorithm. This sorting algorithm is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n is the number of items.

Program:

```
import java.util.Scanner;
class bubbledemo
 {
  public static void main(String args[])
   {
    int n, i,j, temp;
    int a[ ]=new int[20];
    Scanner s = new Scanner(System.in);
    System.out.println("Enter total number of elements:");
    n = s.nextInt();
   System.out.println("Enter elements:");
    for (i = 0; i < n; i++)
    a[i] = s.nextInt();
    for(i=0;i<n;i++)
    {
      for(j=0;j<n-1;j++)
      {
         if(a[j]>a[j+1])
        {
    temp=a[j];
    a[j]=a[j+1];
      a[j+1]=temp;
     }
  }
}
  System.out.println("The sorted elements are:");
  for(i=0;i<n;i++)
  System.out.print("\t"+a[i]);
 }
}
```

Conclusion:

Name & Sign of Course Teacher

Experiment No: 03

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a program for the following
a. Write a java program to implement a class mechanism. – Create a class, methods and invoke them inside the main method.
b. Write a java program to implement constructor overloading
c. Write a java program implement method overloading

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Object: An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

or Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface
- is *an entity which has state and behavior*.
- The object is *an instance of a class*.

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.
Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in the Heap memory area.

Constructors:

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

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

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

Syntax of default constructor:

1. <class_name>(){}

Java Parameterized Constructor

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

Why use the parameterized constructor?

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

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	**Java Method**
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be the same as the class name.	The method name may or may not be the same as the class name.

Method Overloading:

When a class has two or more methods by the same name but different parameters, at the time of calling based on the parameters passed, the respective method is called (or the respective method body will be bonded with the calling line dynamically). This mechanism is known as method overloading.When a class has two or more methods by the same name but different parameters, at the time of calling based on the parameters passed respective method is called (or respective method body will be bonded with the calling line dynamically). This mechanism is known as method overloading.

Program:A.

```
class A
 {
    int l=10,b=20;
    void display()
    {
      System.out.println(l);
      System.out.println(b);
    }
}
class methoddemo
  {
     public static void main(String args[])
     {
          A a1=new A();
           a1.display();
       }

}
```

B.

```
class Student5
{
    int id;

    String name;

    int age;

    //creating two arg constructor
```

```
   Student5(int i,String n){
   id = i;
   name = n;
 }
   //creating three arg constructor
   Student5(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
   void display(){System.out.println(id+" "+name+" "+age);}
    public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

C.

```
class A
{
int l=10,b=20;
int area()
{
return l*b;
}
int area(int l,int b)
{
```

```
return l*b;
}
}
class overmethoddemo
{
public static void main(String args[])
{
A a1=new A();
int r1=a1.area();
System.out.println("The area is: "+r1);
int r2=a1.area(5,20);
System.out.println("The area is: "+r2);
}
}
```

Conclusion:

Name & Sign of Course Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 04

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

---

obj.run();

Aim: Write a program for the following
a. Write a java program for abstract class to find areas of different shape
b. Write a java program to give examples for “super” keywords.
c. Write a java program to implement Interface.
d. Write a program for the implementation of Multiple inheritance using interfaces to calculate the area of a rectangle and triangle

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131
Theory:

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).Abstraction is a process of hiding the implementation details and showing only functionality to the user.another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you 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.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.An abstract class must be declared with an abstract keyword.

- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method

Example of abstract class

```
1. abstract class A{}
```

A method which is declared as abstract and does not have implementation is known as an abstract method.Example of abstract method

```
1. abstract void printStatus();//no method body and abstract
```

Super keyword:

The super keyword in Java is a reference variable which is used to refer to an immediate parent class object.Whenever you create the instance of a subclass, an instance of the parent class is created implicitly which is referred to by a super reference variable.

Usage of Java super Keyword

1. super can be used to refer to the immediate parent class instance variable.
2. super can be used to invoke immediate parent class methods.
3. super() can be used to invoke immediate parent class constructor.

We can use super keywords to access the data member or field of the parent class. It is used if parent class and child class have the same fields.The super keyword can also be used to invoke the parent class method. It should be used if the subclass contains the same method as the parent class. In other words, it is used if the method is overridden.The super keyword can also be used to invoke the parent class constructor.

Interface in JAVA:

An interface in Java is a blueprint of a class. It has static constants and abstract methods.The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.Multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

- Java Interface also represents the IS-A relationship.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have default and static methods in an interface.
- Since Java 9, we can have private methods in an interface.

Why use Java interface?

There are mainly three reasons to use interfaces. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

A:

```
abstract class shape
{
abstract double area();
}
class rectangle extends shape
{
double l=12.5,b=2.5;
double area()
{
return l*b;
}
}
class triangle extends shape
{
double b=4.2,h=6.5;
double area()
{
return 0.5*b*h;
}
}
class square extends shape
{
double s=6.5;
double area()
{
return 4*s;
}
}
class shapedemo
{
public static void main(String[] args)
{
rectangle r1=new rectangle();
triangle t1=new triangle();
square s1=new square();
System.out.println("The area of rectangle is: "+r1.area());
System.out.println("The area of triangle is: "+t1.area());
System.out.println("The area of square is: "+s1.area());
}
}
```

B:

```
class A
  {
    int l,b;
    A()
```

```
   {
     l=10;
     b=20;
   }
}
class B extends A
{
  int h;
  B()
  {
   super();
   h=30;
  }
int volume()
  {
    return l*b*h;
  }
}
class superdemo
{
   public static void main(String args[])
   {
     B b1=new B();
     int r=b1.volume();
     System.out.println("The vol. is: "+r);
   }
}
```

C:

```
 Interface A
{
     void display();
}
class B implements A
  {
    public void display()
    {
       System.out.println("B's method");
     }
}
class C extends B
 {
     public void callme()
       {
          System.out.println("C's method");
        }
}
class interfacedemo
 {
     public static void main(String args[])
     {
       C c1=new C();
       c1.display();

     }
}
```

D:

```
import java.io.*;
interface area
  {
   float compute(float x, float y);
  }

class rectangle
  {
   public float compute(float x, float y)
     {
      return (x*y);
     }
  }

class triangle
  {
   public float compute(float x, float y)
     {
      return (x*y/2);
     }
  }

class result extends rectangle implements area
  {
   public float compute(float x, float y)
     {
      return (x*y);
     }
  }

class result1 extends triangle implements area
  {
   public float compute(float x, float y)
     {
      return (x*y/2);
     }
  }

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

```
    {
     result rect = new result();
     result1 tri = new result1();
     area a;
     a = rect;
     System.out.println("\nArea of rectangle = " + a.compute(10,20));
     a = tri;
     System.out.println("\nArea of triangle  = " +a.compute(10,2));
    }
 }
```

Conclusion:

Name & Sign of Course Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 05

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a program for the following
a. Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line
b. Write a Java program that checks whether a given string is a palindrome or not.

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Java string

In Java, string is basically an object that represents a sequence of char values. An array of characters works the same as Java string.. For example:

```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

CharSequence Interface.

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance
```

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.String objects are stored in a special memory area known as the "string constant pool".

Why does Java use the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

Java String class methods

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.

6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	String concat(String str)	concatenates the specified string.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.

14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns a split string matching regex.
17	String[] split(String regex, int limit)	returns a split string matching regex and limit.
18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	String toLowerCase()	returns a string in lowercase.

24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	String toUpperCase()	returns a string in uppercase.
26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is an overloaded method.

Program:

A.

```
import java.io.*       //Importing io package
class Occurance        //defining class
{
        public static void main (String args[])           //defining main method
        {
                try                                        //using try
                {
                        FileInputStream fis=new FileInputStream("Input.txt");
                                                           //reading from file
                        BufferedInputStream bis=new BufferedInputStream(fis);
                                        //sorting temporary in file
                        int i;          //declaring variables
                        int occur=0;
                        while((i=bis.read())!=-1)       //using while loop
                        {
                                char a=(char)i;         //using type casting
                                if(i==args[0].charAt(0))        //using if condition
                                {
                                        occur++;                //incrementing variable
                                }
                        }                               //closing whlie loop
                        System.out.println("The letter "+ args[0].charAt(0)+" occurs "+occur+" times
in the file ");
                }       catch(Exception ex)             //using catch
                        {
                                System.out.println(ex.getMessage());
                        }
        }                                               //closing main
}                                                       //closing class
```

B.

```
import java.util.Scanner;
class ChkPalindrome
{
  public static void main(String args[])
  {
    String str, rev = "";
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a string:");
    str = sc.nextLine();
    int length = str.length();
    for ( int i = length - 1; i >= 0; i-- )
        rev = rev + str.charAt(i);
     if(str.equals(rev))
        System.out.println(str+" is a palindrome");
     else
        System.out.println(str+" is not a palindrome");
 }
}
```

Conclusion:

Name & sign of Course Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 06

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21

Date of Performance: **Date of Completion:**

Aim: Write a java program that creates threads by extending the Thread class .First thread display “Good Morning “every 1 sec, the second thread displays “Hello “every 2 seconds and the third display “Welcome” every 3 seconds ,(Repeat the same by implementing Runnable)

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Threading in java:

Multithreading in Java is a process of executing multiple threads simultaneously.A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.However, we use multithreading rather than multiprocessing because threads use a shared memory area. They don't allocate separate memory areas so saves memory, and context-switching between the threads takes less time than process.Java Multithreading is mostly used in games, animation, etc.

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.Threads are independent. If there occurs an exception in one thread, it doesn't affect other threads. It uses a shared memory area.Java provides Thread class to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Java Thread Methods

S. N.	Modifier and Type	Method	Description
1)	void	start()	It is used to start the execution of the thread.
2)	void	run()	It is used to do an action for a thread.
3)	static void	sleep()	It sleeps a thread for the specified amount of time.

4)	static Thread	currentThread()	It returns a reference to the currently executing thread object.
5)	void	join()	It waits for a thread to die.
6)	int	getPriority()	It returns the priority of the thread.
7)	void	setPriority()	It changes the priority of the thread.
8)	String	getName()	It returns the name of the thread.
9)	void	setName()	It changes the name of the thread.
10)	long	getId()	It returns the id of the thread.
11)	boolean	isAlive()	It tests if the thread is alive.
12)	static void	yield()	It causes the currently executing thread object to pause and allow other threads to execute temporarily.
13)	void	suspend()	It is used to suspend the thread.
14)	void	resume()	It is used to resume the suspended thread.

15)	void	stop()	It is used to stop the thread.
16)	void	destroy()	It is used to destroy the thread group and all of its subgroups.
17)	boolean	isDaemon()	It tests if the thread is a daemon thread.
18)	void	setDaemon()	It marks the thread as daemon or user thread.
19)	void	interrupt()	It interrupts the thread.
20)	boolean	isinterrupted()	It tests whether the thread has been interrupted.
21)	static boolean	interrupted()	It tests whether the current thread has been interrupted.
22)	static int	activeCount()	It returns the number of active threads in the current thread's thread group.
23)	void	checkAccess()	It determines if the currently running thread has permission to modify the thread.
24)	static boolean	holdLock()	It returns true if and only if the current thread holds the monitor lock on the specified object.

25)	static void	dumpStack()	It is used to print a stack trace of the current thread to the standard error stream.
26)	StackTraceElement[]	getStackTrace()	It returns an array of stack trace elements representing the stack dump of the thread.
27)	static int	enumerate()	It is used to copy every active thread's thread group and its subgroup into the specified array.
28)	Thread.State	getState()	It is used to return the state of the thread.
29)	ThreadGroup	getThreadGroup()	It is used to return the thread group to which this thread belongs
30)	String	toString()	It is used to return a string representation of this thread, including the thread's name, priority, and thread group.
31)	void	notify()	It is used to give the notification for only one thread which is waiting for a particular object.
32)	void	notifyAll()	It is used to give the notification to all waiting threads of a particular object.

33)	void	setContextClassLoader()	It sets the context ClassLoader for the Thread.
34)	ClassLoader	getContextClassLoader()	It returns the context ClassLoader for the thread.
35)	static Thread.UncaughtExceptionHandler	getDefaultUncaughtExceptionHandler()	It returns the default handler invoked when a thread abruptly terminates due to an uncaught exception.
36)	static void	setDefaultUncaughtExceptionHandler()	It sets the default handler invoked when a thread abruptly terminates due to an uncaught exception.

Program:

1.

```
class A extends Thread
{
public void run()
{
try
{
for(int i=1;i<=10;i++)
{
sleep(1000);
System.out.println("good morning");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class B extends Thread
{
public void run()
{
try
{
for(int j=1;j<=10;j++)
{
sleep(2000);
System.out.println("hello");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class C extends Thread
{
public void run()
```

```
{
try
{
for(int k=1;k<=10;k++)
{
sleep(3000);
System.out.println("welcome");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class threaddemo
{
public static void main(String args[])
{
A a1=new A();
B b1=new B();
C c1=new C();
a1.start();
b1.start();
c1.start();
}
}
```

B.

```
class A implements Runnable
```

```
{
public void run()
{
try
{
for(int i=1;i<=10;i++)
{
Thread.sleep(1000);
System.out.println("good morning");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class B implements Runnable
{
public void run()
{
try
{
for(int j=1;j<=10;j++)
{
Thread.sleep(2000);
System.out.println("hello");
}
}
```

```
catch(Exception e)
{
System.out.println(e);
}
}
}
class C implements Runnable
{
public void run()
{
try
{
for(int k=1;k<=10;k++)
{
Thread.sleep(3000);
System.out.println("welcome");
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
class runnabledemo
{
public static void main(String args[])
{
A a1=new A();
```

```
B b1=new B();
C c1=new C();
Thread t1=new Thread(a1);
Thread t2=new Thread(b1);
Thread t3=new Thread(c1);
t1.start();
t2.start();
t3.start();
}
}
```

Conclusion:

Name & sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 07

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a JAVA program for creation of User Defined Exception
Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:
Exception Handling in JAVA:
The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling.Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception

3. Error

1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.There are 5 keywords which are used in handling exceptions in Java.

Keyword	**Description**
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means we can't use try blocks alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Program:

```
A extends Exception
 {
   A(String s1)
   {
    super(s1);
  }
}
class owndemo
{
  public static void main(String args[])
 {
   try
  {
   throw new A("demo ");
 }
catch(Exception e)
   {
     System.out.println(e);
  }
}
}
```

Conclusion:

Name & sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 08

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a java program that import and use the defined your package in the previous Problem
Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131
Theory:

A java package is a group of similar types of classes, interfaces and sub-packages.Package in java can be categorized in two form, built-in package and user-defined package.There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

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

- //save as Simple.java
- package mypack; -
- public class Simple
- {
- public static void main(String args[])

- {
- System.out.println("Welcome to package");
- }
- }

How to compile java package

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

1. javac -d directory javafilename

For example

1. javac -d . Simple.java

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

How to run java package program

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

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:Welcome to package

```
package mypack;

public class box

{

public int l=100,b=200;
public void display()
{
System.out.println(l);
System.out.println(b);
}
}
```

3. Create sub directory with a name same that of package name under the current working directory by as follows. d:\>md mypack

4. Under this subdirectory store the above program with a file name “box.java”.

(ii) importing a package:

Steps:

1. packages can be accessed by using the import statement

General form: import pack1[.pack2].(classname/*);

Example: import java.io.*;

Here pack1 is name of top level package and pack2 is name of sub package

2. Type the following program under the current working directory and save the program with a file name “example.java”.

```
import mypack.box;
class packagedemo
{
public static void main(String args[])
{
```

```
box b1=new box();

b1.display();

}

}
```

3. Now compile the above program in the current working directory d:\

```
javac packagedemo.java
```

4. Execute the above program in current working directory

```
java packagedemo
```

Conclusion:

Name & sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 09

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a java program to paint like paint brush in applet
Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at the client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Diagram:

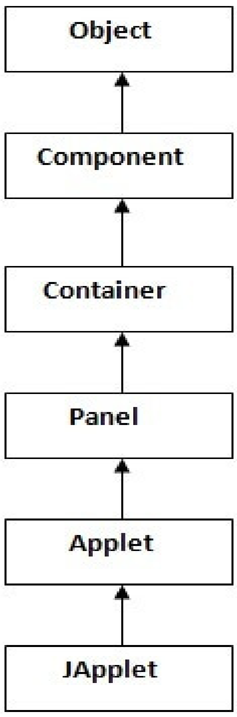
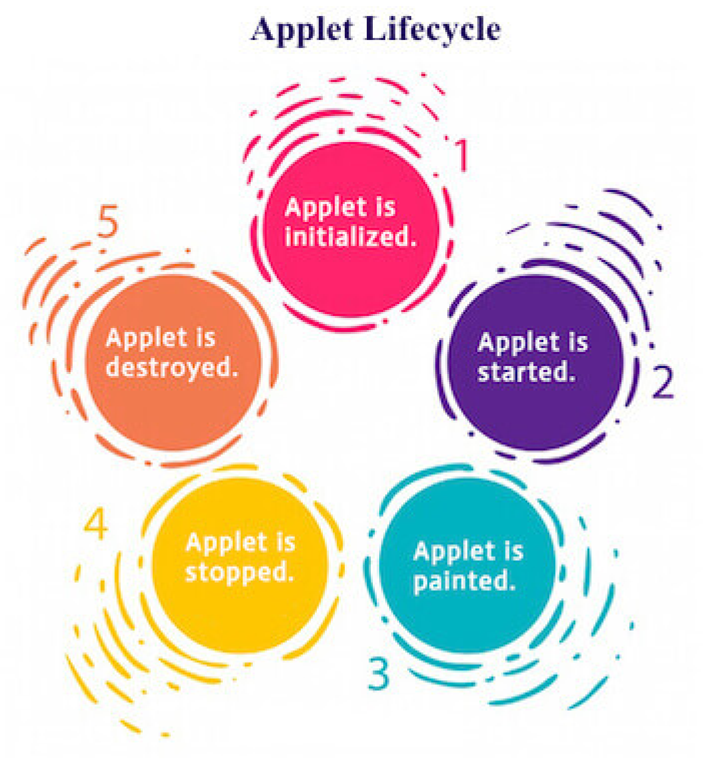


Figure.Hierarchy of Applet

Life Cycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applets are painted.
4. Applet is stopped.
5. Applet is destroyed



The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. public void init(): is used to initialize the Applet. It is invoked only once.
2. public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. public void stop(): is used to stop the Applet. It is invoked when the Applet is stopped or browser is minimized.
4. public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java   (if program is having First as class name)
c:\>appletviewer First.java
```

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill a rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw a line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used to draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.

11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Program:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
//<applet code="paintdemo" width="800" height="500"></applet>
public class paintdemo extends Applet implements MouseMotionListener
{
int w, h;
Image i;
Graphics g1;
public void init()
{
w = getSize().width; h = getSize().height;
i = createImage( w, h );
g1 = i.getGraphics();
g1.setColor( Color.white ); g1.fillRect( 0, 0, w, h ); g1.setColor( Color.red );
i = createImage( w, h );
g1 = i.getGraphics();
g1.setColor( Color.white ); g1.fillRect( 0, 0, w, h ); g1.setColor( Color.blue );
addMouseMotionListener( this );
}
public void mouseMoved( MouseEvent e ) { }
public void mouseDragged( MouseEvent me )
{
int x = me.getX(); int y = me.getY();
g1.fillOval(x-10,y-10,20,20);
repaint();
me.consume();
}
public void update( Graphics g )
{
g.drawImage( i, 0, 0, this );
}
public void paint( Graphics g )
{
update(g);
}
}
```

Conclusion:

Name and Sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering

Experiment No: 10

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a program to draw a form using GUI components to accept details from a customer for a bank.

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

The AWT contains numerous classes and methods that allow you to create and manage windows. It is also the foundation upon which Swing is built. In this experiment, you will learn how to create and manage windows, manage fonts, output text, and utilize graphics, pushbuttons, supported by the AWT. It also explains further aspects of Java's event handling mechanism..Although a common use of the AWT is in applets, it is also used to create stand-alone windows that run in a GUI environment, such as Windows.

AWT Classes

The AWT classes are contained in the java.awt package. It is one of Java's largest packages.Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use than you might at first believe. Figure shows JAVA AWt hierarchy.

Container

The Container is a component in AWT that can contain other components like buttons, text fields, labels etc. The classes that extend Container class are known as containers such as Frame, Dialog and Panel.

Window

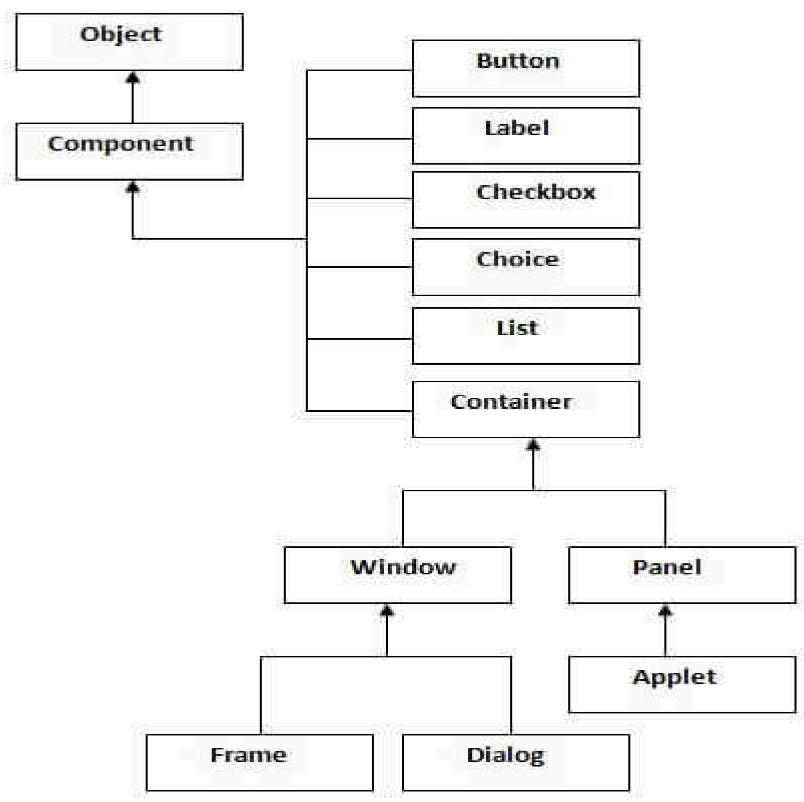
The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain a title bar and menu bars. It can have other components like buttons, textfield 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



Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java Event classes and Listener interfaces

Event Classes	**Listener Interfaces**
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Window Fundamentals

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard application window. Much of the functionality of these windows is derived from their parent classes. Thus, a description of the class hierarchies relating to these two classes is fundamental to their understanding. Below figure shows the class hierarchy for Panel and

Frame. Let's look at each of these classes now.

Component

At the top of the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component. It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container (since they are themselves instances of Component). This makes for a multileveled containment system. A container is responsible for laying out (that is, positioning) any components that it contains..

Panel

The Panel class is a concrete subclass of Container. It doesn't add any new methods; it simply implements Container. A Panel may be thought of as a recursively nestable, concrete screen component. Panel is the superclass for Applet. When screen output is directed to an applet, it is drawn on the surface of a Panel object. In essence, a Panel is a window that does notcontain a title bar, menu bar, or border. This is why you don't see these items when an appletis run inside a browser. When you run an applet using an applet viewer, the applet viewerprovides the title and border.Other components can be added to a Panel object by its add( ) method (inherited fromContainer). Once these components have been added, you can position and resize themmanually using the setLocation( ), setSize( ), setPreferredSize( ), or setBounds( ) methods defined by Component.

Window

TheWindow class creates a top-level window. Atop-level window is not contained within any other object; it sits directly on the desktop. Generally, you won't create Window objects directly. Instead, you will use a subclass of Window called Frame, described next.

Frame

Frame encapsulates what is commonly thought of as a "window." It is a subclass ofWindowand has a title bar, menu bar, borders, and resizing corners. If you create a Frame object from within an applet, it will contain a warning message, such as “Java Applet Window,” to the user that an applet window has been created. This message warns users that the window they see was started by an applet and not by software running on their computer . When a Frame window is created by a stand-alone application rather than an applet, a normal window is created.

Canvas

Although it is not part of the hierarchy for applet or frame windows, there is one other type of window that you will find valuable: Canvas. Canvas encapsulates a blank window upon which you can draw.

Working with Frame Windows

After the applet, the type of window you will most often create is derived from Frame. You will use it to create child windows within applets, and top-level or child windows for stand-alone applications. As mentioned, it creates a standard-style window.

Here are two of Frame’s constructors:

-Frame( )

-Frame(String title)

The first form creates a standard window that does not contain a title. The second form creates a window with the title specified by title. Notice that you cannot specify the dimensions of the window. Instead, you must set the size of the window after it has been created. There are several key methods you will use when working with Frame windows. They are examined here

Setting the Window’s Dimensions

The setSize( ) method is used to set the dimensions of the window. Its signature is shown here:

void setSize(int newWidth, int newHeight)

void setSize(Dimension newSize)

The new size of the window is specified by newWidth and newHeight, or by the width and height fields of the Dimension object passed in newSize. The dimensions are specified in terms of pixels. The getSize( ) method is used to obtain the current size of a window. Its signature is shown here: **Dimension getSize( ) -**This method returns the current size of the window contained within the width and height fields of a Dimension object**.**

Hiding a Window

After a frame window has been created, it will not be visible until you call setVisible( ). Its signature is shown here: void setVisible(boolean visibleFlag).The component is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a Window’s Title

You can change the title in a frame window using setTitle( ), which has this general form: void setTitle(String newTitle) Here, newTitle is the new title for the window.

Closing a Frame Window

When using a frame window, your program must remove that window from the screen when it is closed, by calling setVisible(false). To intercept a window-close event, you must implement the windowClosing( ) method of the WindowListener interface. Inside windowClosing( ), you must remove the window from the screen. The example in the next section illustrates this technique.

Control Fundamentals
The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

These controls are subclasses of Component

Adding and Removing Controls
To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling add( ), which is defined by Container. The add( ) method has several forms. The following form is the one that is used for the first part of this chapter:
Component add(Component compObj)
Here, compObj is an instance of the control that you want to add. A reference to compObj is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed. Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call remove( ). This method is also defined by Container. It has this general form:
void remove(Component obj)
Here, obj is a reference to the control you want to remove. You can remove all controls by calling removeAll( ).

Responding to Controls
Except for labels, which are passive, all controls generate events when they are accessed by the user. For example, when the user clicks on a push button, an event is sent that identifies the push button. In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor. Once a listener has been installed, events are automatically sent to it

The HeadlessException
Most of the AWT controls described in this practical now have constructors that can throw a HeadlessException when an attempt is made to instantiate a GUI component in a non-interactive environment (such as one in which no display, mouse, or keyboard is present). The HeadlessException was added by Java 1.4. You can use this exception to write code that can adapt to non-interactive environments. (Of course, this is not always possible.)

Using Buttons
Perhaps the most widely used control is the push button. A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button. Button defines these two constructors:
Button( ) throws HeadlessException
Button(String str) throws HeadlessException

The first version creates an empty button. The second creates a button that contains str as a label. After a button has been created, you can set its label by calling setLabel( ). You can retrieve its label by calling getLabel( ). These methods are as follows:
void setLabel(String str)
String getLabel( ) Here, str becomes the new label for the button

Handling Buttons

Each time a button is pressed, an action event is generated. This is sent to any listeners that previously registered an interest in receiving action event notifications from that component. Each listener implements the ActionListener interface. That interface defines the actionPerformed( ) method, which is called when an event occurs. An ActionEvent object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the action command string associated with the button.
By default, the action command string is the label of the button. Usually, either the button reference or the action command string can be used to identify the button. Here is an example that creates three buttons labeled “Yes”, “No”, and “Undecided”

Each time one is pressed, a message is displayed that reports which button has been pressed. In this version, the action command of the button (which, by default, is its label) is used to determine which button has been pressed. The label is obtained by calling the getActionCommand( ) method on the ActionEvent object passed to actionPerformed( ).

```
/ Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet> */
public class ButtonDemo extends Applet implements ActionListener {
String msg = "";
Button yes, no, maybe;
public void init() {
yes = new Button("Yes");
no = new Button("No");
maybe = new Button("Undecided");
add(yes);
add(no);
add(maybe);
yes.addActionListener(this);
no.addActionListener(this);
maybe.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("Yes"))
 {
```

```
msg = "You pressed Yes.";
}
else if(str.equals("No"))
 {
msg = "You pressed No.";
}
else {
msg = "You pressed Undecided.";
}
repaint();
}
public void paint(Graphics g) {
g.drawString(msg, 6, 100);
}
}
```

Sample output from the ButtonDemo program is shown below

As mentioned, in addition to comparing button action command strings, you can also determine which button has been pressed, by comparing the object obtained from the getSource( ) method to the button objects that you added to the window. To do this, you must keep a list of the objects when they are added.

Program:

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

public class BankForm extends JFrame implements ActionListener
{
        Label title = new Label("BANK OF JALGAON");
        Label name= new Label("Name:");
        TextField name1= new TextField();
        Label aadhar= new Label("Aadhar No:");
        TextField aadhar1= new TextField();
        Label address= new Label("Address:");
        TextField address1= new TextField();
        Label city= new Label("City:");
        TextField city1= new TextField();
        Label state= new Label("State:");
        TextField state1= new TextField();
        Label country= new Label("Country:");
        TextField country1= new TextField();
        Label mob= new Label("Mobile No:");
        TextField mob1= new TextField();
        Label email= new Label("Email:");
        TextField email1= new TextField();
        Label password= new Label("Create Password:");
        TextField password1= new TextField();
```

```
	Label cpassword= new Label("Confirm Password:");
	TextField cpassword1= new TextField();

	Label info= new Label("The bank name is imaginary, resemblance to any bank functional or
non functional is purely coincidental.");

	Button save= new Button("SAVE");
	Button clear= new Button("CLEAR");
	Button cancel= new Button("CANCEL");

	BankForm()
	{
		setSize(1000,700);
		setTitle("Bank Form");
		setContentPane(new JLabel(new ImageIcon("/home/harish/Desktop/abc.jpg")));
		setLayout(null);

		title.setBounds(310,50,380,40);
		title.setAlignment(Label.CENTER);
		title.setFont(new Font("Serif", Font.BOLD, 30));
		add(title);
		name.setBounds(290,120,200,30);
		add(name);
		name1.setBounds(510,120,200,30);
		add(name1);
		aadhar.setBounds(290,160,200,30);
		add(aadhar);
		aadhar1.setBounds(510,160,200,30);
		add(aadhar1);
		address.setBounds(290,200,200,30);
		add(address);
		address1.setBounds(510,200,200,30);
		add(address1);
		city.setBounds(290,240,200,30);
		add(city);
		city1.setBounds(510,240,200,30);
		add(city1);
		state.setBounds(290,280,200,30);
		add(state);
		state1.setBounds(510,280,200,30);
		add(state1);
		country.setBounds(290,320,200,30);
		add(country);
		country1.setBounds(510,320,200,30);
		add(country1);
		email.setBounds(290,360,200,30);
		add(email);
		email1.setBounds(510,360,200,30);
		add(email1);
		password.setBounds(290,400,200,30);
		add(password);
		password1.setEchoChar('*');
		cpassword1.setEchoChar('*');
		password1.setBounds(510,400,200,30);
		add(password1);
		cpassword.setBounds(290,440,200,30);
		cpassword1.setBounds(510,440,200,30);
		add(cpassword1);
		add(cpassword);

		info.setBounds(150,650,700,30);
		info.setAlignment(Label.CENTER);
		info.setFont(new Font("Arial", Font.BOLD, 11));
		add(info);

		save.setBounds(310,520,100,30);
		add(save);
		save.addActionListener(this);
		clear.setBounds(450,520,100,30);
		add(clear);
		clear.addActionListener(this);
		cancel.setBounds(590,520,100,30);
		add(cancel);
		cancel.addActionListener(this);

	}

	public void actionPerformed(ActionEvent e)
	{
		String s1= name1.getText();
		String s2= address1.getText();
		String s3= city1.getText();
		String s4= state1.getText();
		String s5= country1.getText();
		String s6= email1.getText();
		String s7= password1.getText();
		String s8= cpassword1.getText();
		boolean valid= false;

		if(e.getActionCommand()=="SAVE")
		{
			if(checked())
			{
				if(s7!=s8)
				{
					JOptionPane.showMessageDialog(save,"Password Does Not
Match");
					cpassword1.setText("");
				}
				}
				else
				{
					save.setVisible(true);
					JOptionPane.showMessageDialog(save,"Data Saved");
					name1.setText("");
					//aadhar1.setText("");
					address1.setText("");
					city1.setText("");
					state1.setText("");
					country1.setText("");
					email1.setText("");
					password1.setText("");
					cpassword1.setText("");
				}
			}
			else
				JOptionPane.showMessageDialog(save,"Data Not Saved\n Please Enter 
Correct Data");

			if(e.getActionCommand()=="CLEAR")
			{
				name1.setText("");
				//aadhar1.setText("");
				address1.setText("");
				city1.setText("");
				state1.setText("");
				country1.setText("");
				email1.setText("");
				password1.setText("");
				cpassword1.setText("");

			}

			if(e.getActionCommand()=="CANCEL")
				System.exit(0);
		}

		public int checkInput(String s)
		{
			int i,flag=1;
			for(i=0;i<s.length();i++)
			{
				if(!Character.isLetter(s.charAt(i)) && !Character.isWhitespace(s.charAt(i)))
				{
					flag=1;
					return(1);
				}
			}
			return(0);
```

```
}

public boolean checked()
{
        String s1= name1.getText();
        String s2= city1.getText();
        String s3= state1.getText();
        String s4= country1.getText();
        //int s9= aadhar1.getText();

        int n1= checkInput(s1);
        int n2= checkInput(s2);
        int n3= checkInput(s3);
        int n4= checkInput(s4);
        //int n9= checkInput(s9);

        if(n1==1)
        {
                JOptionPane.showMessageDialog(save,"Please Enter Valid Name");
                name1.setText("");
                return(false);
        }

        if(n2==1)
        {
                JOptionPane.showMessageDialog(save,"Please Enter Valid city");
                city1.setText("");
                return(false);
        }

        if(n3==1)
        {
                JOptionPane.showMessageDialog(save,"Please Enter Valid State");
                state1.setText("");
                return(false);
        }

        if(n4==1)
        {
                JOptionPane.showMessageDialog(save,"Please Enter Valid country");
                country1.setText("");
                return(false);
        }

                return(false);
}

public static void main(String args[])
{
        BankForm f= new BankForm();
```

```
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                f.setVisible(true);
        }
}
```

Conclusion:

Name and Sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 12

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a program Event handling by anonymous class
Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

Event- In computing, an **event** is an action or occurrence recognized by software that may be handled by the software. Computer events can be generated or triggered by the system, by the user or in other ways. A source of events includes the user, who may interact with the software by way of, for example, keystrokes on the keyboard. Another source is a hardware device such as a timer. Event driven systems are typically used when there is some asynchronous external activity that needs to be handled by a program; for example, a user who presses a button on his mouse. An event driven system typically runs an event loop, that keeps waiting for such activities, e.g. input from devices or internal alarms. When one of these occurs, it collects data about the event and dispatches the event to the *event handler* software that will deal with it.

Delegate Event Model: A common variant in object-oriented programming is the delegate event model, which is provided by some graphic user interfaces. This model is based on three entities:

- a control, which is the event source
- listeners, also called event handlers, that receive the event notification from the source
- interfaces (in the broader meaning of the term) that describe the protocol by which the event is to be communicated.

Furthermore, the model requires that:

- every listener must implement the interface for the event it wants to listen to
- every listener must register with the source to declare its desire to listen to the event

- every time the source generates the event, it communicates it to the registered listeners, following the protocol of the interface.

Event Sources- A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method. Here is the general form:

public void add*Type*Listener(*Type*Listener *el*)

Here, *Type* is the name of the event, and *el* is a reference to the event listener. For example, the method that registers a keyboard event listener is called **addKeyListener( )**. The method that registers a mouse motion listener is called **addMouseMotionListener( )**. When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. In all cases, notifications are sent only to listeners that register to receive them Some sources may allow only one listener to register. The general form of such a method is this:

public void add*Type*Listener(*Type*Listener *el*) throws java.util.TooManyListenersException

Here, *Type* is the name of the event, and *el* is a reference to the event listener. When such an event occurs, the registered listener is notified. This is known as *unicasting* the event. A source must also provide a method that allows a listener to unregister an interest in a specific type of event. The general form of such a method is this:

public void remove*Type*Listener(*Type*Listener *el*)

Here, *Type* is the name of the event, and *el* is a reference to the event listener. For example, to remove a keyboard listener, you would call **removeKeyListener( )**.
The methods that add or remove listeners are provided by the source that generates
events. For example, the **Component** class provides methods to add and remove keyboard and mouse event listeners.

Event Listeners- A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**. For example, the **MouseMotionListener** interface defines two methods to receive notifications when the mouse is dragged or moved. Any object may receive and process one or both of these events if it provides an implementation of this interface. Many other listener interfaces are discussed later in this and other chapters.

Event Classes- The classes that represent events are at the core of Java's event handling mechanism. Thus, a discussion of event handling must begin with the event classes. It is important to understand, however, that Java defines several types of events. The most widely used events are those defined by the AWT and those defined by Swing. At the root of the Java event class hierarchy is **EventObject**, which is in **java.util**. It is the superclass for all events. Its one constructor is shown here:

EventObject(Object *src*)

Here, *src* is the object that generates this event.

EventObject contains two methods:
getSource( ) and toString( ).

The **getSource( )** method returns the source of the event. Its general form is shown here:
Object getSource( )

As expected, **toString( )** returns the string equivalent of the event The class **AWTEvent**, defined within the **java.awt** package, is a subclass of **EventObject**.
It is the superclass (either directly or indirectly) of all AWT-based events used by the delegation event model. Its **getID( )** method can be used to determine the type of the event. The signature of this method is shown here:
int getID( )

At this point, it is important to know only that all of the other classes discussed in this section are subclasses of **AWTEvent**.

To summarize:
• **EventObject** is a superclass of all events.
• **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.
The package **java.awt.event** defines many types of events that are generated by various
user interface elements. Commonly used constructors and methods in each class are described in the following sections.

User generated events:

Mouse events- A pointing device can generate a number of software recognizable pointing device gestures. A mouse can generate a number of mouse events, such as mouse move (including direction of move and distance), mouse left/right button up/down and mouse wheel motion, or a combination of these gestures.

Keyboard events- Pressing a key on a keyboard or a combination of keys generates a keyboard event, enabling the program currently running to respond to the introduced data such as which key/s the user pressed.

Touchscreen events- The events generated using a touchscreen are commonly referred to as touch events or gestures.

Device events- Device events include action by or to a device, such as a shake, tilt, rotation, move etc.

Inner class- Inner class is a class that is defined inside another class. Inner class are scoped to the class used to declare them thus they are effectively invisible to the other class in the same package. This lack of visibility to other classes in the package gives the programmer the opportunity to create a set of classes.

Features of inner class-

*. An object of inner class can access the member of outer class.

*. Anonymous inner classes are handy when you want to define callbacks.

*. Inner class can be hidden from other classes in the same package.
Inner classes are the classes that are defined as members of another class. If the inner class is defined at the same level as the enclosing classes' instances/field variables, Inner class can access those instance variables regardless of their access control. This is the same as any method in class that can access the instance variable.

Syntax for inner class

```
Class <outclass>
{ _ _
        Class <inner class>
        {        _
                 _
        }
}
```

Anonymous Inner class- When using a local inner class & we want to make only a single object of this class, you don’t even need to give a class name. such class is called an anonymous inner class. Anonymous inner class can’t have a constructor because the name of the constructor must be the same as the name of the class, and the class has no name. Anonymous inner classes of Java are called *anonymous* because they have no name. They are anonymous and inline. Anonymous classes are essentially inner classes and defined within some other classes.

However, the way anonymous classes are written in Java code may look weird but anonymous inner classes facilitate programmers to declare and instantiate the class at the same time. Another important point to note about anonymous inner classes is that they can be used only once on the place they are coded. In other words, if you want to create only one sub-classed object of a class, then you need not to give the class a name and you can use anonymous inner class in such a case. Anonymous inner classes can be defined not just within a method, but even within an argument to a method. Anonymous inner classes cannot have explicit constructors declared because they have no name to give the constructor.

How to Declare Java Anonymous Inner Classes?

Anonymous inner classes are defined at the same time they are instantiated with new. They are not declared as local classes; rather anonymous inner classes are defined in the new expression itself, as part of a statement. An anonymous inner class declaration expression looks like a constructor invocation, except that there is a class definition contained in a block of code. Before going into further details of anonymous inner classes we must understand that an anonymous inner class is not an independent inner class rather it is a *sub-class* of either a *class* type or an anonymous *implementer* of the specified *interface* type. So, when anonymous inner classes are in picture *polymorphism* must be there. And when polymorphism is there you can only call methods from parent class reference that are defined in the reference variable type. Java's anonymous inner classes being sub-classes or implementers do strictly adhere to the polymorphism rules.

Java Anonymous Inner Class of a Class Type

Let's take an example of seemingly strange looking syntax that defines an anonymous inner class. In the following example code, the Dog reference variable dog refers *not* to an instance of Dog but to an instance of an anonymous inner subclass of Dog.

```
/* AnonymousClassDemo.java */
public class AnonymousClassDemo
{
        public static void main(String[] args)
        {
                Dog dog = new Dog() {
                        public void someDog ()
                        {
                                System.out.println("Anonymous Dog");
                        }
                }; // anonymous class body closes here
                        //dog contains an object of anonymous subclass of Dog.
        dog.someDog();
        }
}
class Dog
{
        public void someDog()
        {
                System.out.println("Classic Dog");
        }
}
```

In the above piece of code; see the code line Dog dog = new Dog() {, there is a brace at the end of line, not a semicolon. This curly brace opens the class definition and declares a new class that has no name (anonymous class). Now let's enter into the body of newly defined subclass of class Dog and you will see that someDog() is being overridden. This is the crux of defining an anonymous inner class because we want to override one or more methods of the super class on the fly.

Remember, anonymous inner classes are inherited ones, and we always use a superclass reference variable to refer to an anonymous subclass object. And, we can only call methods on an anonymous inner class object that are defined in the superclass. Though, we can introduce new methods in anonymous inner class, but we cannot access them through the reference variable of superclass because superclass does not know anything about new methods or data members introduced in subclass.
It would be interesting to know that you can also create an anonymous inner class for an interface type. Magically you can also pass anonymous inner class as an argument to a method. We will talk of them in subsequent sections.

Program:

```
/*
Aim: Write a program for Event handling by anonymous class.


import java.awt.*;                    //anonymous inner class
import java.awt.event.*;
//import javax.swing.*;
//class AEvent3 extends JFrame
class AEvent3 extends Frame
{
TextField tf;
AEvent3()
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener() //anonymous inner class starts here
{
public void actionPerformed(ActionEvent ae){
tf.setText("hello");
}
}); //anonymous inner class ending here
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[])
{
 AEvent3 f= new AEvent3();
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 //new AEvent3();
}
}
```

Conclusion:

Name and Sign of Teacher

Government College of Engineering, Jalgaon

Department of Computer Engineering
Experiment No: 11

Subject:CO310U (Application programming Lab) **Sem:**V(Odd)
Class:T.Y. B.Tech **Academic Year:**2020-21
Date of Performance: **Date of Completion:**

Aim: Write a program using a graphics method to draw an object ; provide direction buttons and move the object in the direction specified by the user through the button.

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131
Theory:

Graphics

The AWT supports a rich assortment of graphics methods. All graphics are drawn relative to a window. This can be the main window of an applet, a child window of an applet, or a stand-alone application window. The origin of each window is at the top-left corner and is 0,0.Coordinates are specified in pixels. All output to a window takes place through a graphics context. A*graphics context* is encapsulated by the **Graphics** class and is obtained in two ways:

- It is passed to an applet when one of its various methods, such as **paint( )** or **update( )**, is called.
- It is returned by the getGraphics( ) method of Component.

The Graphics class defines a number of drawing functions. Each shape can be drawn edge only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When a graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped.
Let's look at several of the drawing methods.

Drawing Lines

Lines are drawn by means of the drawLine( ) method, shown here:
void drawLine(int startX, int startY, int endX, int endY)
drawLine( ) displays a line in the current drawing color that begins at startX,startY and ends at endX,endY. The following applet draws several lines:

```
// Draw lines

        import java.awt.*;
        import java.applet.*;
        /*
        <applet code="Lines" width=300 height=200>
        </applet>
        */
        public class Lines extends Applet {
        public void paint(Graphics g) {
        g.drawLine(0, 0, 100, 100);
        g.drawLine(0, 100, 100, 0);
        g.drawLine(40, 25, 250, 180);
        g.drawLine(75, 90, 400, 400);
        g.drawLine(20, 150, 400, 40);
        g.drawLine(5, 290, 80, 19);
         }
         }
```

Drawing Rectangles

The drawRect( ) and fillRect( ) methods display an outlined and filled rectangle, respectively.

They are shown here:

void drawRect(int top, int left, int width, int height)

void fillRect(int top, int left, int width, int height)

The upper-left corner of the rectangle is at top,left. The dimensions of the rectangle are specified by width and height.

To draw a rounded rectangle, use drawRoundRect( ) or fillRoundRect( ), both shown here: void drawRoundRect(int top, int left, int width, int height, int xDiam, int yDiam) void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam). A Rounded rectangle has rounded corners. The upper-left corner of the rectangle is at top,left. The dimensions of the rectangle are specified by width and height. The diameter of the rounding arc along the X axis is specified by xDiam. The diameter of the rounding arc along the Y axis is specified by *yDiam.*

The following applet draws several rectangles:

```
 // Draw rectangles

import java.awt.*;

import java.applet.*;
```

```
/*
<applet code="Rectangles" width=300
height=200> </applet>
*/
public class Rectangles extends Applet
{
  public void paint(Graphics g)
{
  g.drawRect(10, 10, 60, 50);
  g.fillRect(100, 10, 60, 50);
 g.drawRoundRect(190, 10, 60, 50, 15,15);
 g.fillRoundRect(70, 90, 140, 100, 30, 40); }
}
```

Sample output from this program is shown here:

Drawing Ellipses and Circles

To draw an ellipse, use drawOval( ). To fill an ellipse, use fillOval( ). These methods are shown here:

```
void drawOval(int top, int left, int width, int height)
void fillOval(int top, int left, int width, int height)
```

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by top,left and whose width and height are specified by width and height. To draw a circle, specify a square as the bounding rectangle.

The following program draws several ellipses:

```
// Draw Ellipses
import java.awt.*;
import java.applet.*;
/*
```

```
<applet code="Ellipses" width=300 height=200>

        </applet>

*/

public class Ellipses extends Applet {

public void paint(Graphics g) {

g.drawOval(10, 10, 50, 50);

g.fillOval(100, 10, 75, 50);

g.drawOval(190, 10, 90, 30);

g.fillOval(70, 90, 140, 100);

}

}
```

Drawing Arcs

Arcs can be drawn with drawArc( ) and fillArc( ), shown here:

void drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle) void fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle)

The arc is bounded by the rectangle whose upper-left corner is specified by top,left and whose width and height are specified by width and height. The arc is drawn from startAngle through the angular distance specified by sweepAngle. Angles are specified in degrees. Zero degrees are on the horizontal, at the three o'clock position. The arc is drawn counterclockwise if sweepAngle is positive, and clockwise if sweepAngle is negative. Therefore, to draw an arc from twelve o'clock to six o'clock, the start angle would be 90 and the sweep angle 180. The following applet **draws several arcs:**

```
// Draw Arcs
import java.awt.*;
import java.applet.*;
/*
<applet code="Arcs" width=300 height=200>
</applet>
*/
public class Arcs extends Applet {
public void paint(Graphics g) {
g.drawArc(10, 40, 70, 70, 0, 75);
g.fillArc(100, 40, 70, 70, 0, 75);
g.drawArc(10, 100, 70, 80, 0, 175);
g.fillArc(100, 100, 70, 90, 0, 270);
g.drawArc(200, 80, 80, 80, 0, 180);
}
}
```

Working with Color

Java supports color in a portable, device-independent fashion. The AWT color system allows you to specify any color you want. It then finds the best match for that color, given the limits of the display hardware currently executing your program or applet. Thus, your code does not need to be concerned with the differences in the way color is supported by various hardware devices. Color is encapsulated by the **Color** class.

You can also create your own colors, using one of the color constructors. Three commonly used forms are shown here:

Color(int *red*, int *green*, int *blue*)

Color(int rgbValue)

Color(float red, float green, float blue)

The first constructor takes three integers that specify the color as a mix of red, green, and blue. These values must be between 0 and 255, as in this example:

```
new Color(255, 100, 100); // light red
```

The second color constructor takes a single integer that contains the mix of red, green, and blue packed into an integer. The integer is organized with red in bits 16 to 23, green in bits 8 to 15, and blue in bits 0 to 7. Here is an example of this constructor:

```
int newRed = (0xff000000 | (0xc0 << 16) | (0x00 << 8) | 0x00);
```

```
Color darkRed = new Color(newRed);
```

The final constructor, Color(float, float, float), takes three float values (between 0.0 and 1.0) that specify the relative mix of red, green, and blue.

Once you have created a color, you can use it to set the foreground and/or background color by using the setForeground( ) and setBackground( ) methods described in Chapter 21. You can also select it as the current drawing color.

Color Methods

The Color class defines several methods that help manipulate colors. They are examined here. Using Hue, Saturation, and Brightness .The hue-saturation-brightness (HSB) color model is an alternative to red-green-blue (RGB) for specifying particular colors. Figuratively, *hue* is a wheel of color. The hue is specified with a number between 0.0 and 1.0 (the colors are approximately red, orange, yellow, green, blue, indigo, and violet). *Saturation* is another scale ranging from 0.0 to 1.0, representing light pastels to intense hues. *Brightness* values also range from 0.0 to 1.0, where 1 is bright white and 0 is black. **Color** supplies two methods that let you convert between RGB and HSB. They are shown here:

static int HSBtoRGB(float hue, float saturation, float brightness)

static float[ ] RGBtoHSB(int red, int green, int blue, float values[ ])

HSBtoRGB( ) returns a packed RGB value compatible with the Color(int) constructor. RGBtoHSB( ) returns a float array of HSB values corresponding to RGB integers. If values is not null, then this array is given the HSB values and returned. Otherwise, a new array is created and the HSB values are returned in it. In either case, the array contains the hue at index 0, saturation at index 1, and brightness at index 2.

getRed( ), getGreen( ), getBlue( )

You can obtain the red, green, and blue components of a color independently using getRed( ), getGreen( ), and getBlue( ), shown here:

int getRed( )

int getGreen( )

int getBlue( )

Each of these methods returns the RGB color component found in the invoking Color object in the lower 8 bits of an integer.

getRGB( )

To obtain a packed, RGB representation of a color, use getRGB( ), shown here: int getRGB( )

The return value is organized as described earlier.

Setting the Current Graphics Color

By default, graphics objects are drawn in the current foreground color. You can change this color by calling the Graphics method setColor( ):

void setColor(Color newColor)

Here, newColor specifies the new drawing color.

You can obtain the current color by calling getColor( ), shown here:

Color getColor( )

Setting the Paint Mode

The paint mode determines how objects are drawn in a window. By default, new output to

a window overwrites any preexisting contents. However, it is possible to have new objects XORed onto the window by using setXORMode( ), as follows:

void setXORMode(Color xorColor)

Here, xorColor specifies the color that will be XORed to the window when an object is drawn.

The advantage of XOR mode is that the new object is always guaranteed to be visible no matter

what color the object is drawn over. To return to overwrite mode, call **setPaintMode( )**, shown here:

void setPaintMode( )

In general, you will want to use overwrite mode for normal output, and XOR mode for special purposes. For example, the following program displays cross hairs that track the mouse pointer. The cross hairs are XORed onto the window and are always visible, no matter what the underlying color is.

// Demonstrate XOR mode.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="XOR" width=400 height=200>
</applet>
*/
public class XOR extends Applet {
int chsX=100, chsY=100;
public XOR() {
addMouseMotionListener(new MouseMotionAdapter() {
public void mouseMoved(MouseEvent me) {
int x = me.getX();
int y = me.getY();
chsX = x-10;
chsY = y-10;
repaint();
}
});
}
public void paint(Graphics g) {
g.drawLine(0, 0, 100, 100);
g.drawLine(0, 100, 100, 0);
g.setColor(Color.blue);
g.drawLine(40, 25, 250, 180);
g.drawLine(75, 90, 400, 400);
g.setColor(Color.green);
g.drawRect(10, 10, 60, 50);
g.fillRect(100, 10, 60, 50);
g.setColor(Color.red);
g.drawRoundRect(190, 10, 60, 50, 15, 15);
g.fillRoundRect(70, 90, 140, 100, 30, 40);
 g.setColor(Color.cyan);
g.drawLine(20, 150, 400, 40);
g.drawLine(5, 290, 80, 19);
// xor cross hairs
g.setXORMode(Color.black);
g.drawLine(chsX-10, chsY, chsX+10, chsY);
g.drawLine(chsX, chsY-10, chsX, chsY+10);
g.setPaintMode();
}
```

```
        }
```

Program:

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

public class Move extends Frame implements ActionListener
{
	Button up = new Button("Up");
	Button down = new Button ("Down");
	Button right = new Button ("Right");
	Button left = new Button ("Left");
	Button exit = new Button ("Exit");

	int x=200;
	int y=200;

	Object s;

	Move()
	{
		setSize(500,500);
		setTitle("Move Object");
		setLayout(new FlowLayout());
		setBackground(Color.red);

		add(up);
		add(down);
		add(right);
		add(left);
		add(exit);

		up.addActionListener(this);
		down.addActionListener(this);
		right.addActionListener(this);
		left.addActionListener(this);
		exit.addActionListener(this);
	}
		public void actionPerformed(ActionEvent ae)
		{
			s=ae.getSource();
			repaint();
		}

		public void paint(Graphics g)
		{
			if(s==up)
				y=y-20;

			else
```

```
                        if(s==down)
                        y=y+20;

                else
                        if(s==right)
                        x=x+20;

                else
                        if(s==left)
                        x=x-20;

                else
                        if(s==exit)
                        System.exit(0);
                        g.drawRect(x,y,50,50);
        }


    public static void main(String args[])
    {
        Move m = new Move();
        m.setVisible(true);
    }
}
```

Conclusion:

Name and Sign of Teacher