14EI605

Hall Ticket Number:

III/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

April, 2018 **Electronics & Instrumentation Engineering**

Sixth Semester **Object Oriented Programming with JAVA**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

Scheme of instructions and solutions

1. Answer all questions (1X12=12 Marks)

   a) What is Encapsulation?
   Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

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

   c) How do Java different from C++?
   Java does not support multiple inheritance
   Java does not support pointers
   Java does not support operator over loading
   All other differences are listed below any two differences

   d) What is an exception?
   Exceptions are events that occur during the execution of programs that disrupt the normal flow of instructions (e.g. divide by zero, array access out of bound, etc.). In Java, an exception is an object that wraps an error event that occurred within a method and contains: Information about the error including its type.

   e) What is the difference between process and thread?
   A Process :A program in execution is often referred as process. A process consists of multiple threads.
   A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.

   f) Can we have try block without catch block?
   Yes, we can have try without catch block by using finally block. You can use try with finally. As you know finally block always executes even if you have exception or return statement in try block

   g) What are the benefits of multithreaded programming? .
   Enhanced performance by decreased development time.
   Simplified and streamlined program coding.
   Improvised GUI responsiveness.
   Simultaneous and parallelized occurrence of tasks.
   Better use of cache storage by utilization of resources.
   Decreased cost of maintenance.
   Better use of CPU resource.

   h) How will you initialize an applet?
   Make an HTML page with the appropriate tag to load the applet code.
   Supply a subclass of the JApplet class
   Eliminate the main method in the application

Move any initialization code from the frame window constructor to the init method of the applet.

i) What is source and listener?

Source − The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provides us with classes for the source object. Listener − It is also known as event handler. The listener is responsible for generating a response to an event.

j) What is the difference between a Window and a Frame?

A window is an undecorated Frame.

A Frame is derived from window but it has title bar, border, close button, minimize button, resizable and movable options.

k) What are different types of layout managers?

Border Layout.

Box Layout.

Card Layout.

Flow Layout.

Grid Bag Layout.

l) Why Swing components are called lightweight component?

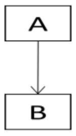
Swing is considered lightweight because it is fully implemented in Java, without calling the native operating system for drawing the graphical user interface components.

UNIT I

2. a) What are the forms of inheritance? Explain its briefly. 6M

1) Single Inheritance

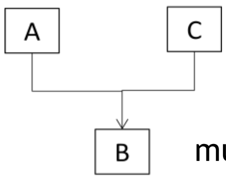
Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.



Single Inheritance example program in Java

2) Multiple Inheritance

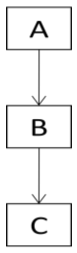
“**Multiple Inheritance**” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



multiple inheritance

3) Multilevel Inheritance

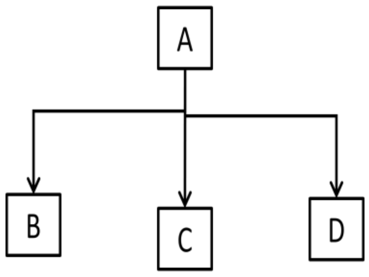
Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



Multilevel inheritance

4) Hierarchical Inheritance

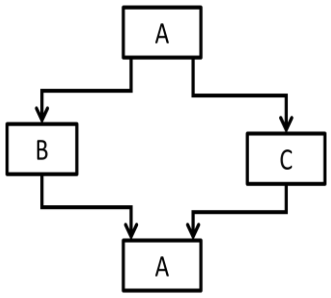
In an inheritance one class is inherited by many sub classes. In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D.



Hierarchical Inheritance

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritances. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritances can be done Using interfaces. By using interfaces you can have multiple as well as hybrid inheritance in Java.



hybrid inheritance

b) **What is polymorphism? Differentiate between method overloading and method overriding with an example.** 6M

Polymorphism in java is a concept by which we can perform a *single action by different ways*. There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Overloading vs Overriding in Java

1. Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.
2. Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do

with the same method of parent class.

3. The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
4. Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.
5. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
6. private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
7. Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the overriding method can have more specific return type (refer this).
8. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

Over loading

```
class  Sum
{
int  add ( int n1,int n2)
   {
    return  n1+n2;
   }
int  add ( int n1,int n2, int n3)
   {
    return  n1+n2+n3;
   }
int  add ( int n1,int n2, int n3, int n4)
   {
    return  n1+n2+n3+n4;
   }
int  add ( int n1,int n2, int n3, int n4, int n5)
   {
    return  n1+n2+n3+n4+n5;
  }
Public static void main (String args[])
{
Sum obj = new Sum();
System.out.println("The sum of two numbers:
"+obj.add(2,3));
System.out.println("The sum of three
numbers: "+obj.add(2,3,4));
System.out.println("The sum of four
numbers: "+obj.add(2,3,4,5));
System.out.println("The sum of five numbers:
"+obj.add(2,3,4,5,6));
}
}
```

Overriding

```
class CarClass
        {
                Public int  speedlimit()
                        {
                        return(100)
                        }
        }
Public ford extends carclass
       {
                Public int  speedlimit()
                        {
                                return(150);
                        }

        Public static void main(String arg[])
                {
                CarClass obj = new ford();
                Int n = obj.speedlimit();
                System.out.println("Speed
                Limit:"+n);
                }
        }
```

Any examples of their choice

Polymorphism in java is a concept by which we can perform a *single action by different ways*. There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

(OR)

3. a) Define an interface in java. How interfaces are implemented? 6M

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.**

Interface

```
interface Drawable{

void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
    Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable(
    )
    d.draw();
    }}
```

Multiple inheritance:

```
interface Printable{
void print();  }
interface Showable{
void show();  }
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();  obj.show();
 }
}
```

b) What are the methods in String buffer class? Explain. 6M

A StringBuffer object holds characters that can be changed by appending and by inserting characters. Also, a StringBuffer object includes many useful character manipulation methods.

If you insert characters into a StringBuffer, it **automatically** grows to the length you need. This relieves much of the tedium that directly using an array would involve.

Constructors and methods of StringBuffer are:

StringBuffer constructores

public StringBuffer()	create an empty StringBuffer
public StringBuffer(int capacity)	create a StringBuffer with initial room for capacity number of characters
public StringBuffer(String str)	create a StringBuffer containing the characters from str

StringBuffer Methods	
StringBuffer append(char c)	append c to the end of the StringBuffer
StringBuffer append(int i)	convert i to characters, then append them to the end of the StringBuffer
StringBuffer append(long L)	convert L to characters, then append them to the end of the StringBuffer
StringBuffer append(float f)	convert f to characters, then append them to the end of the StringBuffer
StringBuffer append(double d)	convert d to characters, then append them to the end of the StringBuffer
StringBuffer append(String s)	append the characters in s to the end of the StringBuffer
int capacity()	return the current capacity (capacity will grow as needed).
char charAt(int index)	get the character at index.
StringBuffer delete(int start, int end)	delete characters from start to end-1
StringBuffer deleteCharAt(int index)	delete the character at index
StringBuffer insert(int index, char c)	insert character c at index (old characters move over to make room).
StringBuffer insert(int index, String st)	insert characters from st starting at position i.
StringBuffer insert(int index, int i)	convert i to characters, then insert them starting at index.
StringBuffer insert(int index, long L)	convert L to characters, then insert them starting at index
StringBuffer insert(int index, float f)	convert f to characters, then insert them starting at index.
StringBuffer insert(int index, double d)	convert d to characters, then insert them starting at index.
int length()	return the number of characters presently in the buffer.
StringBuffer reverse()	Reverse the order of the characters.
void setCharAt(int index, char c)	set the character at index to c.
String toString()	return a String object containing the characters in the StringBuffer.

UNIT II

4. a) Write the role of throw and throws in exception handling. 6M

S.No.	**throw**	**throws**
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	Cannot throw multiple exceptions.	Declare multiple exceptions

		e.g. public void method()throws IOException, SQLException.

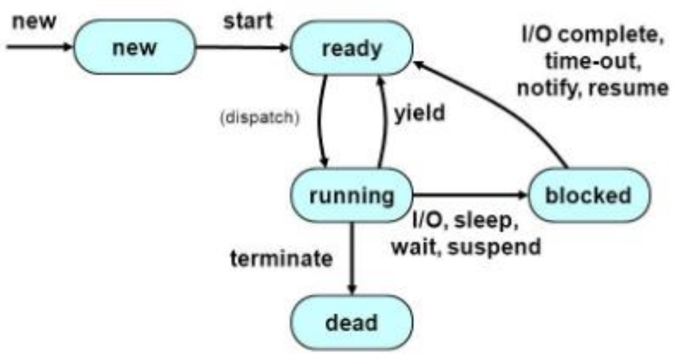
```
public class Example1{
   void checkAge(int age){
        if(age<18)
           throw new ArithmeticException("Not Eligible for voting");
        else
           System.out.println("Eligible for voting");
   }
   public static void main(String args[]){
        Example1 obj = new Example1();
        obj.checkAge(13);
        System.out.println("End Of Program");
   }
}
```

```
public class Example1{
   int division(int a, int b) throws ArithmeticException{
        int t = a/b;
        return t;
   }
   public static void main(String args[]){
        Example1 obj = new Example1();
        try{
           System.out.println(obj.division(15,0));
        }
        catch(ArithmeticException e){
           System.out.println("You shouldn't divide number by zero");
        }
   }
}
```

b) **Explain thread life cycle in detail.** 6M

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.
The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated


new
new
start
ready
I/O complete, time-out, notify, resume
(dispatch)
yield
running
blocked
I/O, sleep, wait, suspend
terminate
dead

1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

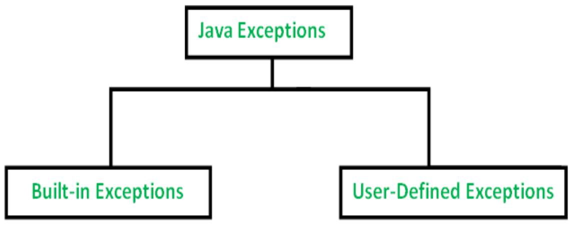
5) Terminated

A thread is in terminated or dead state when its run() method exits.

(OR)

5. a) What are different types of exceptions and explain with a program. 8M

EXCEPTION: The exception is object created at the time of exceptional/error condition which will be thrown from the program and halt normal execution of the program. Java exceptions object hierarchy is as below



Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **Arithmetic Exception**
   It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **ArrayIndexOutOfBoundException**
   It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException**
   This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException**
   This Exception is raised when a file is not accessible or does not open.
5. **IOException**
   It is thrown when an input-output operation failed or interrupted
6. **InterruptedException**
   It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
7. **NoSuchFieldException**
   It is thrown when a class does not contain the field (or variable) specified

8. **NoSuchMethodException**
   It is thrown when accessing a method which is not found.
9. **NullPointerException**
   This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException**
    This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException**
    This represents any exception which occurs during runtime.
12. **StringIndexOutOfBoundsException**
    It is thrown by String class methods to indicate that an index is either negative than the size of the string

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
        }
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a number by 0");
        }
    }
     }
```

User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called 'user-defined Exceptions'.
Following steps are followed for the creation of user-defined Exception.

- The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:

```
class MyException extends Exception
```

- We can write a default constructor in his own exception class.

```
MyException(){}
```

- We can also create a parameterized constructor with a string as a parameter. We can use this to store exception details. We can call super class(Exception) constructor from this and send the string there.

```
// Java program to demonstrate user defined exception

// This program throws an exception whenever balance
// amount is below Rs 1000
class MyException extends Exception
{
    //store account information
    private static int accno[] = {1001, 1002, 1003, 1004};

    private static String name[] =
                 {"Nish", "Shubh", "Sush", "Abhi", "Akash"};

    private static double bal[] =
         {10000.00, 12000.00, 5600.0, 999.00, 1100.55};

    // default constructor
    MyException() {    }

    // parametrized constructor
    MyException(String str) { super(str); }

    // write main()
    public static void main(String[] args)
    {
        try  {
            // display the heading for the table
            System.out.println("ACCNO" + "\t" + "CUSTOMER" +
                                              "\t" + "BALANCE");

            // display the actual account information
            for (int i = 0; i < 5 ; i++)
            {
                System.out.println(accno[i] + "\t" + name[i] +
                                                  "\t" + bal[i]);

                // display own exception if balance < 1000
                if (bal[i] < 1000)
                {
                    MyException me =
                       new MyException("Balance is less than 1000");
                    throw me;
                }
            }
        } //end of try

        catch (MyException e) {
            e.printStackTrace();
        }
    }
}
```

b) What are the differences between multitasking and multithreading? 4M

Differences Between Multitasking and Multithreading

1. The basic difference between multitasking and multithreading is that in **multitasking**, the system allows executing multiple programs and tasks at the same time, whereas, in **multithreading**, the system executes multiple threads of the same or different processes at the same time.

2. In multitasking **CPU** has to **switch** between **multiple programs** so that it appears that multiple programs are running simultaneously. On other hands, in multithreading **CPU** has to **switch** between **multiple threads**to make it appear that all threads are running simultaneously.

3. Multitasking allocates **separate memory and resources** for each process/program whereas, in multithreading threads belonging to the same process **shares the same memory and resources** as that of the process.

UNIT III

6. a) Write a java program to pass parameters to applet. 6M

Parameters are passed to **applets** in NAME=**VALUE** pairs in <PARAM> tags between the opening and closing **APPLET** tags. Inside the **applet**, you read the**values** passed through the PARAM tags with the getParameter() method of the**java.applet.Applet** class.

PARAM TAG

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes: *name* and *value* which are used to specify the name of the parameter and the value of the parameter respectively. For example, the param tags for passing name and age parameters looks as shown below:

<param name=”name” value=”Ramesh” />
<param name=”age” value=”25″ />

***getParameter()* method**

The *getParameter()* method of the *Applet* class can be used to retrieve the parameters passed from the HTML page. The syntax of *getParameter()*method is as follows:

String getParameter(String param-name)
Let's look at a sample program which demonstrates the <param> HTML tag and the *getParameter()* method

```
import java.applet.*;
public class MyApplet extends Applet
{
        String n;
        String a;
        public void init()
        {
                n = getParameter("name");
                a = getParameter("age");
        }
        public void paint(Graphics g)
        {
                g.drawString("Name is: " + n, 20, 20);
                g.drawString("Age is: " + a, 20, 40);
        }
}
/*
        <applet code="MyApplet" height="300" width="500">
                <param name="name" value="Ramesh" />
                <param name="age" value="25" />
        </applet>
*/
```

b) Explain event handling along with different event types in java. 6M

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

- **Foreground Events** - Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.

- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

Event handling

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.Let's have a brief introduction to this model.

Java core consists of 12 event types defined in java.awt.events :

1. ActionEvent.
2. AdjustmentEvent.
3. ComponentEvent.
4. ContainerEvent.
5. FocusEvent.
6. InputEvent.
7. ItemEvent.
8. KeyEvent.

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){
 //create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
 //register listener
    b.addActionListener(this);//passing current instance    //add components and set s
    ize, layout and visibility
    add(b);add(tf);
    setSize(300,300);
    setLayout(null);
    setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
    tf.setText("Welcome");
    }
    public static void main(String args[]){
    new AEvent();
    }
    }
```

(OR)

7. a) **What are the applications and uses of an applet?** 6M

An applet is a Java™ program designed to be included in an HTML Web document. You can write your Java applet and include it in an HTML page, much in the same way an image is included. When you use a Java-enabled browser to view an HTML page that contains an applet, the applet's code is transferred to your system and is run by the browser's Java virtual machine.

Applications are stand-alone programs that do not require the use of a browser. Java applications run by starting the Java interpreter from the command line and by specifying the file that contains the compiled application. Applications usually reside on the system on which they are deployed. Applications access resources on the system, and are restricted by the Java security model.

Uses : Java Applets are usually used to add small, interactive components or enhancements to a webpage. These may consist of buttons, scrolling text, or stock tickers, but they can also be used to display larger programs like word processors or games. Java's incredible portability (the ability to run on many different operating systems and browsers) is what makes it ideal for use on the World Wide Web.

b) What are the event classes and listeners in java.awt.event package. 6M

Event classes :

The Event classes represent the event. Java provides us various Event classes

EventObject class

It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the **source**, that is logically deemed to be the object upon which the Event in question initially occurred upon.This class is defined in java.util package.

Class declaration

Following is the declaration for **java.util.EventObject** class:

Public class EventObject

extends object

implements Serilizable

Event listeners :

The Event listener represent the interfaces responsible to handle events. Java provides us various Event listener classes but we will discuss those which are more frequently used. Every method of an event listener method has a single argument as an object which is subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

EventListner interface

It is a marker interface which every listener interface has to extend.This class is defined in java.util package.

Class declaration

Following is the declaration for **java.util.EventListener** interface:

public interface EventListener.

List of commonly used event classes :

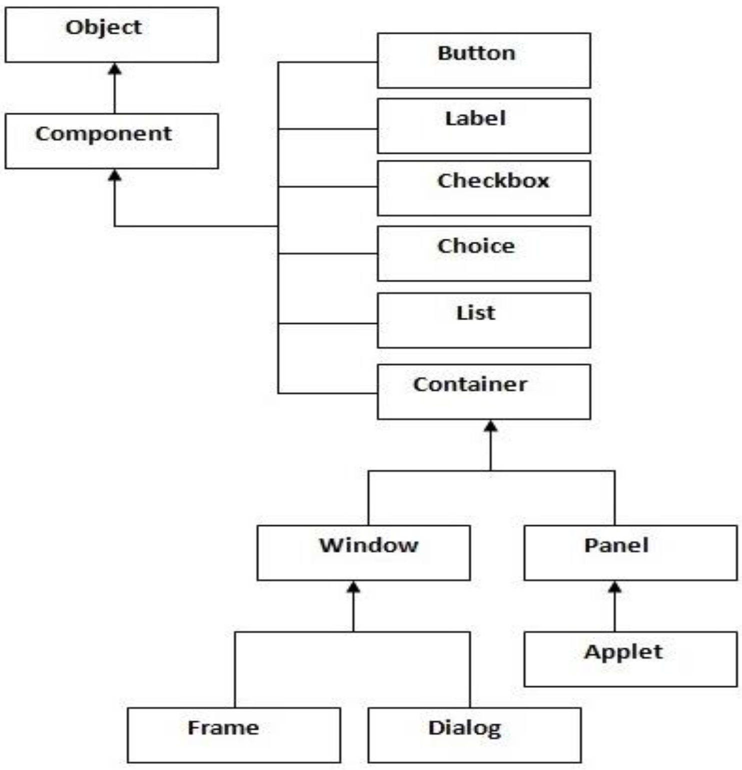
S. No.	Control & Description
1	**AWTEvent** :It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.
2	**ActionEvent** :The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	**InputEvent** :The InputEvent class is root event class for all component-level input events.
4	**KeyEvent**: On entering the character the Key event is generated.

5	**MouseEvent** :This event indicates a mouse action occurred in a component.
6	**TextEvent**: The object of this class represents the text events.
7	**WindowEvent** :The object of this class represents the change in state of a window.
8	**AdjustmentEvent** :The object of this class represents the adjustment event emitted by Adjustable objects.
9	**ComponentEvent** :The object of this class represents the change in state of a window.
10	**ContainerEvent**: The object of this class represents the change in state of a window.
11	**MouseMotionEvent** :The object of this class represents the change in state of a window.
12	**PaintEvent** :The object of this class represents the change in state of a window.

UNIT IV

8. a) What is AWT class? Explain in detail. 6M

The Abstract Window Toolkit (**AWT**) is **Java**'s original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing. The **AWT** is part of the **Java**Foundation **Classes** (JFC) — the standard API for providing a graphical user interface (GUI) for a **Java** program.



b) Write a java program with swing components button, text field and check box. 6M

Button:

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

```
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```

Text field:
The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
  {
  JFrame f= new JFrame("TextField Example");
  JTextField t1,t2;
  t1=new JTextField("Welcome to Javatpoint.");
  t1.setBounds(50,100, 200,30);
  t2=new JTextField("AWT Tutorial");
     t2.setBounds(50,150, 200,30);
     f.add(t1); f.add(t2);
     f.setSize(400,400);
     f.setLayout(null);
     f.setVisible(true);
  }
  }
```

Check box:
The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

```
import javax.swing.*;
public class CheckBoxExample
{
   CheckBoxExample(){
     JFrame f= new JFrame("CheckBox Example");
     JCheckBox checkBox1 = new JCheckBox("C++");
```

```
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
      f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
    {
   new CheckBoxExample();
   }}
```

(OR)

9. a) **Explain clearly the following components of AWT with suitable examples each** 8M

i. Label ii. Scrollbar

i. Class **java.awt.Label**. A **Label** object is a **component** for placing text in a container. A **label** displays a single line of read-only text. The text can be changed by the application, but a user cannot edit it directly.

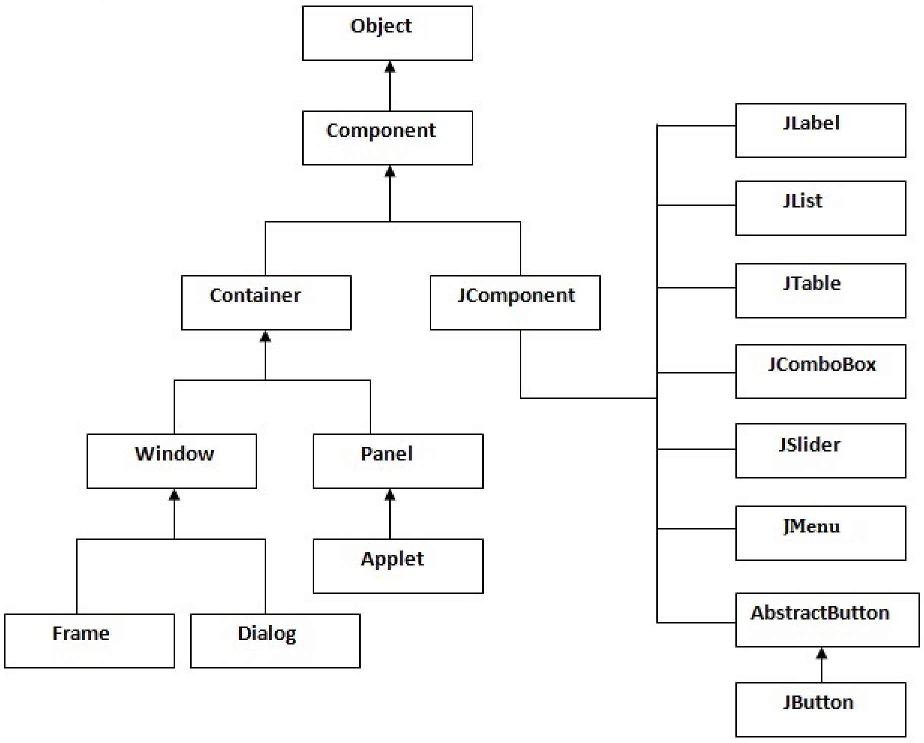
```
public class Label extends Component implements Accessible

import java.awt.*;
class LabelExample{
public static void main(String args[]){
    Frame f= new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

ii. **Scrollbar** The object of **Scrollbar** class is used to add horizontal and vertical **scrollbar**. **Scrollbar** is a GUI **component** allows us to see invisible number of rows and columns.

```
public class Scrollbar extends Component implements Adjustable, Accessible
import java.awt.*;
class ScrollbarExample{
ScrollbarExample(){
```

```
            Frame f= new Frame("Scrollbar Example");
            Scrollbar s=new Scrollbar();
            s.setBounds(100,100, 50,100);
            f.add(s);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
    }
    public static void main(String args[]){
             new ScrollbarExample();
    }
    }
```

b) What are the swing component classes? Explain in detail. 4M



Explain in brief about the classes