
Model Solution for VTU Question Paper- July/Aug 2022
Object Oriented Concept (18CS45)

1a) Explain the concept of Object Oriented Programming Language . 6M

i) Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of its own class in which it is declared.

As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a combination of data-hiding and abstraction.

Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

It is more defined with the setter and getter method.

ii) Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Real-life Illustration: Polymorphism

A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, So it means many forms.

Types of polymorphism

In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

iii) Inheritance

Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.
Advantages of Inheritance :
Code Reusability: The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class's code.
Method Overriding: Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
Abstraction: The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

1b) What is an inline function? What are the advantages of inline function? Write a C++ program to find minimum of 2 numbers using inline function. 8M

Inline function is the optimization technique used by the compilers. One can simply prepend inline keyword to function prototype to make a function inline. Inline function instruct compiler to insert complete body of the function wherever that function got used in code.
Advantages :- 1) It does not require function calling overhead.
2) It also saves overhead of variables push/pop on the stack, while function calling.
3) It also saves overhead of return call from a function.
4) It increases locality of reference by utilizing instruction cache.
5) After in-lining compiler can also apply intraprocedural optimization if specified. This is the most important one, in this way compiler can now focus on dead code elimination, can give more stress on branch prediction, induction variable elimination etc.

Program:

```
#include <iostream>
using namespace std;
inline int min(int a,int b)
{
        if(a<b)
        return a;
        else
        return b;
}
int main() {
  cout<<"enter the value of A and B";
        int a,b;
```

```
        cin>>a>>b;
        cout<<"minimum is :"<<min(a,b);
}
```

1c) Define friend function. Illustrate with example.6M

A friend function in C++ is defined as a function that can access private, protected and public members of a class.

The friend function is declared using the friend keyword inside the body of the class.

Friend Function Syntax:

```
class className {
   ... .. ...
   friend returnType functionName(arguments);
   ... .. ...
}
```

By using the keyword, the 'friend' compiler understands that the given function is a friend function.

We declare friend function inside the body of a class, whose private and protective data needs to be accessed, starting with the keyword friend to access the data. We use them when we need to operate between two different classes at the same time.

Friend functions of the class are granted permission to access private and protected members of the class in C++. They are defined globally outside the class scope. Friend functions are not member functions of the class.

So, what exactly is the friend function?

A friend function in C++ is a function that is declared outside a class but is capable of accessing the private and protected members of the class. There could be situations in programming wherein we want two classes to share their members. These members may be data members, class functions or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

```
#include <iostream>
using namespace std;
class Box
{
  private:
     int length;
  public:
      Box (): length (0) {}
  friend int printLength (Box); //friend function
```

```
};
int printLength (Box b)
{
   b. length +=10;
   return b. length;
}
int main ()
{
  Box b;
  cout <<" Length of box:" <<printLength (b)<<endl;
   return 0;
}
```

2a) Why friend functions are required? Write a C++ program to illustrate the use of friend function. 6M

There are two reasons why we may use a friend function:

- Some functions need privileged access to more than one class. For example, we want to write a function that multiplies a matrix by a vector.
- All friend functions pass all their arguments through the argument list, and each argument value is subject to assignment-compatible conversions.

Here is an example of this situation.

Conversions would apply to an object passed explicitly and would be especially useful in cases of binary operator overloading, as we will see a bit later in this module.

Friends must be explicitly declared as part of Class Definition

The modifiers

private,

protected, and

public

give the programmer control over access to names, which implicitly gives the programmer control over access to their associated data values. However, the three possibilities of public, protected, and private cannot cover all situations. Frequently a programmer would like to make a data field or member function accessible to another function or another class that is not a derived class, and to do so without making the name accessible in a public fashion. The solution in this case is to declare a friend. A friend can be either another class or a function. The friend must be explicitly named within a class definition. By naming the friend, the class is granting access to all the private features of the class. Declaring a class as friend means that all member functions in the friend class are friends. Needless to say, this is a dangerous mechanism because it exposes the encapsulated state of a class to outside modification. Friendship is not something that should be given away freely. But because friends are named, and because friends are allowed direct access to internal data fields, friendship is a more precise and efficient mechanism than, for example, the creation of accessor functions.

```
class ClassName{
...
friend class ClassName;
friend return_type function_name(parameter list);
};
```

Example program:

```
#include <iostream>
using namespace std;
class Box
{
  private:
      int length;
  public:
       Box (): length (0) {}
  friend int printLength (Box); //friend function
};
int printLength (Box b)
{
   b. length +=10;
   return b. length;
}
int main ()
{
  Box b;
  cout <<" Length of box:" <<printLength (b)<<endl;
   return 0;
}
```

2b) what is function overloading? Write a C++ program to swap 2 integers by function overloading.8M

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.

The parameters should follow any one or more than one of the following conditions for Function overloading:

- Parameters should have a different type

```
add(int a, int b)
add(double a, double b)
```

Program:

```
#include<iostream.h>
class swap
{
int a,b;
long int m,n,temp;
public:
void input();
void display(int,int);
void display(long int,long int);
};
void swap::input()
{
cout<<"Enter any two integers: ";
cin>>a>>b;
cout<<"Before swapping integer values are: a= "<<a<<" b= "<<b<<endl;
display(a,b);
cout<<"Enter long integers: ";
cin>>m>>n;
cout<<"Before swapping float values are: m= "<<m<<" n= "<<n<<endl;
display(m,n);
}
void swap::display(int a,int b)
{
a=a+b;
b=a-b;
a=a-b;
cout<<"After swapping integer values are: a= "<<a<<" b= "<<b<<endl;
}
void swap::display(long int m, long int  n)
{
temp=m;
m=n;
n=temp;
cout<<"After swapping float values are:  m= "<<m<<" n= "<<n;
}
```

```
void main()
{
swap s1;
s1.input(); s1.display(); }
```

2c) Explain instance variable hiding. Explain with example how to overcome instance variable hiding. 6M

If the superclass and the subclass have instance variable of same name, if you access it using the subclass object, the instance variables of the subclass hides the instance variables of the superclass irrespective of the types. This mechanism is known as field hiding or, instance variable hiding.

Example

In the following example we have two classes Super and Sub one extending the other. They both have two fields with same names (name and age).

When we print values of these fields using the object of Sub. The values of the subclass are printed.

```
class Super {
  String name = "Krishna";
  int age = 25;
}
class Sub extends Super {
  String name = "Vishnu";
  int age = 22;
  public void display(){
    Sub obj = new Sub();
    System.out.println("Name: "+obj.name);
    System.out.println("age: "+obj.age);
  }
}
public class FieldHiding{
  public static void main(String args[]){
    new Sub().display();
  }
}
```

Output

```
Name: Vishnu
age: 22
```

In Java, if there is a local variable in a method with the same name as the instance variable, then the local variable hides the instance variable. If we want to reflect the change made over to the instance variable, this can be achieved with the help of this reference.

```
class Test {
```

```
    // Instance variable or member variable
    private int value = 10;
    // Method
    void method() {
        // This local variable hides instance variable
        int value = 40;

        // Note: this keyword refers to the current instance
        // Printing the value of instance variable
        System.out.println("Value of Instance variable : "
                           + this.value);
        // Printing the value of local variable
        System.out.println("Value of Local variable : "
                           + value);
    }
}
// Class 2
// Main class
class GFG {

    // Main driver method
    public static void main(String args[]) {

        // Creating object of current instance
        // inside main() method
        Test obj1 = new Test();
        // Calling method of above class
        obj1.method();
    }
}
```

3a) what are constructors and destructors? Explain Default constructor with Example.8M

Constructors are special class functions which perform initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

Whereas Destructor on the other hand is used to destroy the class object.

Let's start with Constructors first, following is the syntax of defining a constructor

function in a class:

```
class A
{
 public:
 int x;
 // constructor
 A()
 {
 // object initialization
 }
};
```

While defining a constructor you must remember that the name of constructor will be same as the name of the class, and constructors will never have a return type.

Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution :: operator.

```
class A
{
 public:
 int i;
 A(); // constructor declared
};
// constructor definition
A::A()
{
 i = 1;
```

Default Constructors

Default constructor is the constructor which doesn't take any argument. It has no parameter.

Syntax:

```
class_name(parameter1, parameter2, ...)
{
 // constructor Definition
}
```

For example:

```
class Cube
{
 public:
```

```
 int side;
 Cube()
 {
 side = 10;
 }
};
int main()
{
 Cube c;
 cout << c.side;
}
```

Output:
10

In this case, as soon as the object is created the constructor is called which initializes its data members.

A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

```
class Cube
{
 public:
 int side;
};
int main()
{
 Cube c;
 cout << c.side;
}
```

Output:

0 or any random value

In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, that will be 0 or any random integer value in this case.

3b) Illustrate with an example the order of calling a constructor and destructor. 8M

For multiple inheritance order of constructor call is, the base class's constructors are called in the order of inheritance and then the derived class's constructor.

```
// C++ program to show the order of constructor calls in Multiple Inheritance
```

```
#include <iostream>
using namespace std;
// first base class
class Parent1
{
   public:

   // first base class's Constructor
   Parent1()
   {
      cout << "Inside first base class" << endl;
   }
};
// second base class
class Parent2
{
   public:

   // second base class's Constructor
   Parent2()
   {
      cout << "Inside second base class" << endl;
   }
};
// child class inherits Parent1 and Parent2
class Child : public Parent1, public Parent2
{
   public:

   // child class's Constructor
   Child()
   {
      cout << "Inside child class" << endl;
   }
};
// main function
int main() {
```

```
    // creating object of class Child
    Child obj1;
    return 0;
}
```

Output:

Inside first base class
Inside second base class
Inside child class

3c) Explain namespace with example.4M

Consider a situation when we have two people with the same name, Zara, in the same class. Whenever we need to differentiate them definitely, we would have to use some additional information along with their name, like either the area, if they live in different area or their mother's or father's name, etc.

Same situation can arise in your C++ applications. For example, you might be writing some code that has a function called xyz() and there is another library available which also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code.

A namespace is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope.

Defining a Namespace

A namespace definition begins with the keyword namespace followed by the namespace name as follows −

```
namespace namespace_name {
   // code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend (::) the namespace name as follows −

```
name::code;  // code could be variable or function.
```

Let us see how namespace scope the entities including variable and functions −

Live Demo

```
#include <iostream>
using namespace std;
// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
```

```
  }
}
// second name space
namespace second_space {
  void func() {
    cout << "Inside second_space" << endl;
  }
}
int main () {
  // Calls function from first name space.
  first_space::func();
  // Calls function from second name space.
  second_space::func();
  return 0;
}
```

If we compile and run above code, this would produce the following result −

```
Inside first_space
Inside second_space
```

4a) Explain the following Java buzzword.8M

Object-oriented

Java is said to be a pure object-oriented programming language. In java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types java also implemented as objects using wrapper classes, but still, it allows primitive data types to archive high-performance.

Robust

Java is more robust because the java code can be executed on a variety of environments, java has a strong memory management mechanism (garbage collector), java is a strictly typed language, it has a strong set of exception handling mechanism, and many more.

Architecture-neutral

Java has invented to archive "write once; run anywhere, any time, forever". The java provides JVM (Java Virtual Machine) to to archive architectural-neutral or platform-independent. The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded

Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

4b) Write a java program to find the sum of even numbers using foreach version of for loop and print the result.6M

```
import java.util.Scanner;
public class Main
{
        public static void main(String[] args)
{
int n;
Scanner sc=new Scanner(System.in);
System.out.print("Enter the number of elements you want to store: ");
n=sc.nextInt();
int[] array = new int[10];
System.out.println("Enter the elements of the array: ");
for(int i=0; i<n; i++)
{
array[i]=sc.nextInt();
}
System.out.println("Array elements are: ");
for (int i=0; i<n; i++)
{
System.out.println(array[i]);
}
 int sum = 0;
    for (int i:array)
 {
if(i % 2 == 0)
      {
         sum=sum+i;
      }
      }
      System.out.println("The sum of even numbers is : "+sum);
  }}
```

4c) Explain labelled break and continue with example. 6M

Java Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

This example stops the loop when i is equal to 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

Java Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 4:

Example

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

5a) Explain general form of a class with Example. 6M

A class in Java is **a logical template to create objects that share common properties and methods**. Hence, all objects in a given class will have the same methods or properties.

A class is declared by use of the class keyword.

class *classname* {
type instance-variable1; *type instance-variable2*; // ...
type instance-variableN;
type methodname1(*parameter-list*) {
// body of method }
type methodname2(*parameter-list*) {
// body of method // ...
type methodnameN(*parameter-list*) { // body of method } }

The data, or variables, defined within a class are called *instance variables.* The code is contained within *methods.* Collectively, the methods and variables defined within a class are called *members* of the class. In

most classes, the instance variables are acted upon and accessed by the methods defined for that class. Thus, as a general rule, it is the methods that determine how a class' data can be used.

Example of a class

```
/Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
public static void main(String args[]){
//Creating an object or instance
 Student s1=new Student();//creating an object of Student
 //Printing values of the object
 System.out.println(s1.id);//accessing member through reference variable
System.out.println(s1.name);
} }
```

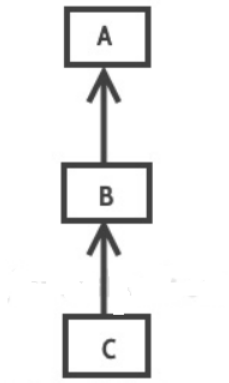
5b) Write a JAVA program to implement stack of integers. Provide constructors and methods to PUSH an element, POP an element and display the content of the stack. 14M

```
import java.io.*;
import java.util.Scanner;
public class stack {
 int element,maxsize,top; int[] st;
public stack() {
Scanner integer=new Scanner(System.in); System.out.println("Enter stack size");
maxsize=integer.nextInt(); st=new int[maxsize]; top=-1; }
public void push(int element) {
if (top>=maxsize) {
System.out.println("Overflow!!");
return(0); }
try { st[++top]=element; }
 catch(ArrayIndexOutOfBoundsException e) { System.out.println(e);
 } }
 public int pop() {
if(top==-1) {
System.out.println("UnderFlow"); return(-1); }
return(st[top--]); }
```

```
public void display(int[] st, int max_size) {
int i; System.out.println("Stack Elements:");
for(i=0;i<=max_size;i++) System.out.println(st[i]); } }
/*myStack.java*/
 import java.io.*;
 import java.util.Scanner;
public class myStack {
 static int option;
public static void main(String[] args) {
stack obj=new stack();
while(true) {
System.out.println("\nEnter yours choice\n1.PUSH\n2.POP\n3.Display\n4..EXIT");
Scanner integer=new Scanner(System.in);
option=integer.nextInt();
switch(option) {
case 1: System.out.println("Enter Element");
obj.element=integer.nextInt(); obj.push(obj.element); break;
case 2:System.out.printf("Poped element is %d",obj.pop());
break;
case 3:obj.display(obj.st,obj.top); break;
case 4:System.exit(0);
default:System.out.println("Wrong option"); } }
```

6a) Explain a multilevel inheritance with an example. 6M

When a class extends a class, which extends another class then this is called multilevel inheritance. For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.Lets see this in a diagram:



Multilevel Inheritance

It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of this diagram, then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and methods of class A along with class B that's what is called multilevel inheritance.

Example:

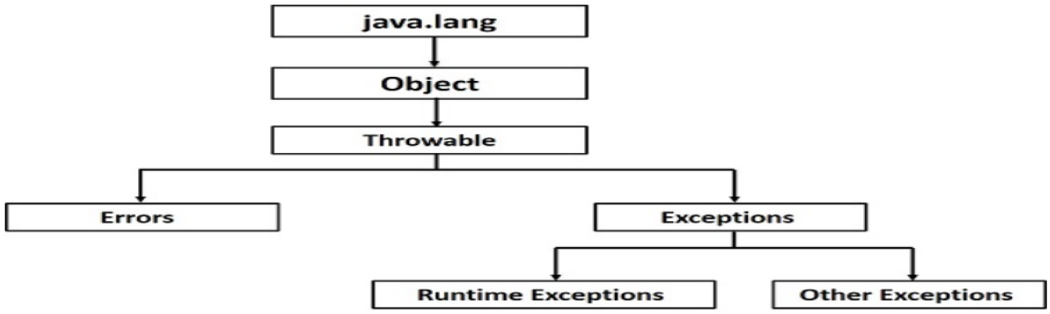
In this example we have three classes – Car, Maruti and Maruti800. We have done a setup – class Maruti extends Car and class Maruti800 extends Maruti. With the help of this Multilevel hierarchy setup our Maruti800 class is able to use the methods of both the classes (Car and Maruti).

```
class Car{
  public Car()  {
        System.out.println("Class Car");  }
  public void vehicleType() {
        System.out.println("Vehicle Type: Car");  }}
class Maruti extends Car{
   public Maruti()  {
        System.out.println("Class Maruti");   }
 public void brand()  {
        System.out.println("Brand: Maruti");  }
   public void speed()  {
        System.out.println("Max: 90Kmph"); }}
public class Maruti800 extends Maruti{

   public Maruti800()  {
```

```
        System.out.println("Maruti Model: 800"); }
  public void speed() {
        System.out.println("Max: 80Kmph"); }
  public static void main(String args[]){
        Maruti800 obj=new Maruti800();  obj.vehicleType();  obj.brand();  obj.speed(); } }
```

6b) Explain the exception handling mechanism provided in java. Give syntax. Write a java program to demonstrate exception handling construct 8M

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class. Following is the exception hierarchy.



Syntax

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
     //code that may raise exception
      }
catch(Name of the Exception e){
   //statement to execute if exception occures
}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

java program to demonstrate exception handling construct

```
public class JavaExceptionExample{
```

```
 public static void main(String args[]){
  try{
        int data=100/0;     }
catch(ArithmeticException e){
System.out.println(e);
}
  System.out.println("rest of the code...");
 } }
```

6c) Write a java program to create a user defined exception and demonstrate its use. 6M

```
// A Class that represents use-defined exception
 class MyException extends Exception {
   public MyException(String s)  {
            super(s);   }   }
 // A Class that uses above MyException
public class Main {
 public static void main(String args[]) {
      try {
         // Throw an object of user defined exception
         throw new MyException("GeeksGeeks");
      }
      catch (MyException ex) {
         System.out.println("Caught");
  // Print the message from MyException object
         System.out.println(ex.getMessage());
      }
   } }
```

Output :

Caught

GeeksGeeks

In the above code, the constructor of MyException requires a string as its argument. The string is passed to the parent class Exception's constructor using super().

7a) Explain the steps to create a package in the Java with an example 8M

Steps to cerate package in java

1. First create a directory within name of package.
2. Create a java file in newly created directory.
3. In this java file you must specify the package name with the help of package keyword.
4. Save this file with same name of public class
   Note: only one class in a program can declare as public.
5. Now you can use this package in your program.

Package Pack1

```
public class Demo{
  public void show() {
    System.out.print("Package called"); } }
```

After that use this package in the program,

```
import Pack1.*;
class A{
  public static void main(String ..args[]) {
    demo ob1 = new demo();  ob1.show();
}}
```

7b) Explain interfaces in java with example 6M

An interface is a completely "abstract class" that is used to group related methods with empty bodies. Another way to achieve abstraction in Java, is with interfaces.

Example

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

Example:

```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}
// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
```

```
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  } }
```

7c) Can an interface will be inherited? Justify with an example 6M

Interface can be Extended

- one interface can inherit another by use of the keyword extends.
- When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

Eg:

```
// One interface can extend another.

Interface A {
   void meth1();
   void meth2();
}
// B now includes meth1() and meth2()  --- it adds meth3()
interface B extends A {
   void meth3();
}
// This class must implement all of  A and B
class MyClass implements B {
   public void meth1() {
      System.out.println(“Method1”);
   }
   public void meth2() {
       System.out.println(“Method2”);
```

```
  }
   public void meth3 () {
      System.out.println("Method3");
  }
}

Class IFExtend {
   public static void main(String args[]) {
   MyClass ob = new MyClass();

   ob.meth1();
   ob.meth2();
   ob.meth3();
}
```

8 a) Explain the following method of Thread class, getName(), getPriority(), isAlive(), join(). 8M

(i) getName(): We can obtain the name of the thread by calling getName().
The general form is
final String getName()

(ii) getPriority(): We can obtain the current priority setting by calling the **getPriority()** method of Thread.
Its general form is
final int getPriority()

(iii) isAlive() method:
We can call **isAlive()** on the thread. Its general form is

final boolean isAlive()

The isAlive method returns true if the thread upon which it is called is still running. It returns false otherwise.

(iv) join() method:
The second method that will be more commonly used to wait for a thread to finish is called join(). Its general form is

final void join() throws InterruptedException

This method waits until the tread on which it is called terminates. **Its name comes from the concept of the called thread waiting until the specified thread joins.** Additional forms of join() allows you to specify a maximum amount of time that you want to wait for the specified thread to terminate.

// Example program to demonstrate isAlive() and join() to ensure that
// the main thread is the last to stop. main thread calls join() on

// all the three child threads it creates.

```
class NewThread implements Runnable {
   // This is the entry point for thread.
   public void run() {
          Thread t = Thread.currentThread();
          String name = t.getName();
          try {
                for(int i = 5; i > 0; i--) {
                      System.out.println(name + ": " + i);
                      Thread.sleep(1000);
                }
          }
          catch (InterruptedException e) {
                System.out.println(name + " interrupted.");
          }
          System.out.println(name + " exiting.");
   }
}
class Demo_Join_isAlive {
  public static void main(String args[]) {
          NewThread ob1 = new NewThread();
          NewThread ob2 = new NewThread();
          NewThread ob3 = new NewThread();

          Thread t1 = new Thread(ob1, "One");
          Thread t2 = new Thread(ob2, "Two");
          Thread t3 = new Thread(ob3, "Three");

          t1.start();
          t2.start();
          t3.start();

          System.out.println("Thread One is alive: " + t1.isAlive());
          System.out.println("Thread Two is alive: " + t2.isAlive());
          System.out.println("Thread Three is alive: " + t3.isAlive());

          // wait for threads to finish
          try {
```

```
                System.out.println("Waiting for threads to finish.");
                t1.join();
                t2.join();
                t3.join();
            }
            catch (InterruptedException e) {
                System.out.println("Main thread Interrupted");
            }
            System.out.println("Thread One is alive: " + t1.isAlive());
            System.out.println("Thread Two is alive: " + t2.isAlive());
            System.out.println("Thread Three is alive: " + t3.isAlive());
            System.out.println("Main thread exiting.");
    }
}
```

Output:

```
$ javac Demo_Join_isAlive.java
$ java Demo_Join_isAlive
Two: 5
One: 5
Three: 5
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for threads to finish.
Two: 4
Three: 4
One: 4
Two: 3
One: 3
Three: 3
Two: 2
Three: 2
One: 2
Two: 1
Three: 1
One: 1
Two exiting.
Three exiting.
One exiting.
```

Thread One is alive: false
Thread Two is alive: false
Thread Three is alive: false
Main thread exiting.

b) Write a Java program to illustrate thread creation using Runnable interface. 6M

Creating a thread by Implementing Runnable interface:

To create a thread by implementing a Runnable interface

- **create a class that implements the Runnable interface.** The Runnable interface has only one method called run(). Hence the class has to implement the method run(). The syntax of the run() method is

  public void run()

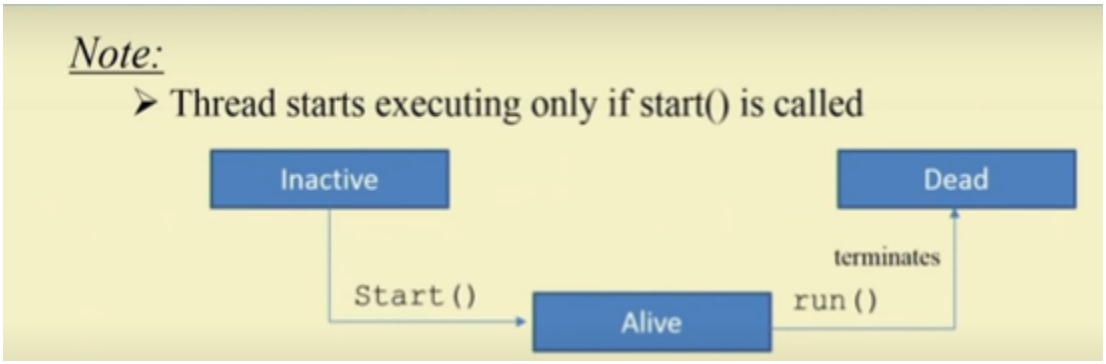
  Inside run(), you will define the code that constitutes the new thread. Run() can call other methods, use other classes and declare variables just like the main thread can. The only difference is that run() establishes the entry point for another concurrent thread of execution within your program. This thread will end when run() returns.

- After you create a class that implements Runnable, create an object of that class.
- Also create an object of Thread and pass Runnable object to its constructor. Thread defines several constructors.

  Thread ()
  Thread (String threadName)
  Thread (Runnable threadOb)
  Thread (Runnable threadOb,String threadName)

  In this constructor, threadOb is an instance of a class that implements the Runnable interface. This defines where execution of the thread will begin. The name of the new thread is specified by threadName.
- After the new thread is created, it will not start running until you call its start() method, which is declared within Thread. In essence, start executes a call to run(). The start() method is shown here

  void start()



Example program to create thread by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Output:

thread is running…

```
// Second example program to illustrate creation of thread using Runnable
class MultiThread implements Runnable
 {
        public void run()
        {
```

```
                System.out.println("thread is running...");
                Thread t = Thread.currentThread();
                System.out.println("Child Thread : " + t);
        }
}

class MultiThreadDemo1{

        public static void main(String args[])
        {
                Thread t = Thread.currentThread();  //main thread
                System.out.println("Current thread : " + t);
                MultiThread m1=new MultiThread();  // Creating a new thread
                Thread t1 =new Thread(m1,"Demo Thread");
                t1.start();  // starting the new thread
        }
}
output:
$ javac MultiThreadDemo1.java
$ java MultiThreadDemo1
Current thread : Thread[main,5,main]
thread is running...
Child Thread : Thread[Demo Thread,5,main]
```

c) Write a Java program to illustrate synchronization using synchronized methods 6M

Using Synchronized Methods:

All objects have their own **implicit monitor** associated with them. To enter an object's monitor, just call a method that has been modified with the **synchronized** keyword. While a thread is inside a synchronized method, all other threads that try to call it on the **same instance** have to wait. To exit the monitor and relinquish control of the object to the next waiting thread, the owner of the monitor simply returns from the synchronized method.

To understand the need of synchronization, first let's see an example that does not use synchronization even though it is required. The example program below has three classes.

The first one, **Callme** class has a single method named call(). The call() method takes a String parameter called msg. This method tries to print the msg string inside of square brackets. But, after call() prints the opening bracket and the msg string, it calls Thread.sleep(1000), which pauses the current thread for one second.

The second class is Caller. The constructor of this class takes a reference to an instance of the Callme class and a String, which are stored in target and msg, respectively. The constructor also creates a new thread that will call this object's run() method. The thread is started immediately. The run() method of caller calls the call() method on the target instance of Callme, passing in the msg string.

Finally, the third class, Synch class starts by creating a single instance of Callme, and three instances of Caller, each with a unique message string. **The same instance of Callme is passed to each Caller.**

We must serialize access to call(). That is, we must restrict its access to only one thread at a time.
To do this, we have to precede call()'s definition with the keyword **synchronized** as shown here

```
class Callme {
synchronized void call (String msg) {
....
```

This prevents other threads from entering call() while another thread is using it.

// Example program to illustrate synchronization using synchronized method

```
class Callme {
                synchronized void call(String msg) {
                System.out.print("[" + msg);
                try {
                Thread.sleep(1000);
                }
                catch (InterruptedException e) {
                System.out.println("Interrupted");
                }
                System.out.println("]");
                }
}

class Caller implements Runnable {
                String msg;
                Callme target;
                Thread t;
                public Caller(Callme targ, String s) {
                target = targ;
                msg = s;
                t = new Thread (this);
                t.start();
                }
```

```
                    public void run()  {
                    target.call(msg);
                    }
}

class Synch  {
                    public static void main(String args[])  {
                    Callme target = new Callme();
                    Caller ob1 = new Caller (target, "Hello");
                    Caller ob2 = new Caller(target, "Synchronized");
                    Caller ob3 = new Caller (target, "World");

                    // wait for the threads to end
                    try   {
                    ob1.t.join();
                    ob2.t.join();
                    ob3.t.join();
                    }
                    catch (InterruptedException e)  {
                    System.out.println("Interrupted");
                    }
                    }
}
```

After synchronized has been added to call(), the output of the program is as follows

Output:

$ javac Synch.java
$ java Synch
[Hello]
[World]
[Synchronized]

If **synchronized** is not added to call(), then we are not serializing access to call(). That is, we are not restricting its access to only one thread at a time. So the output obtained is as shown below

Output:

$ javac Synch.java
$ java Synch

[World[Hello[Synchronized]
]
]

In the above output, by calling sleep(), the call() method allows execution to switch to another thread. This results in the mixed up output of the three message strings.

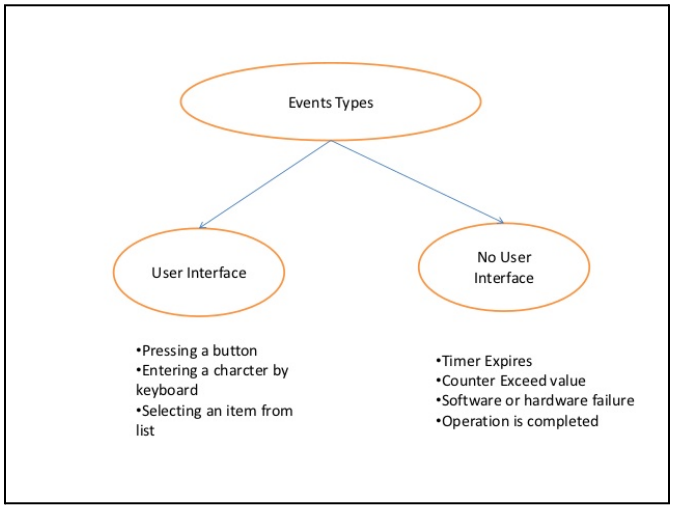
Nothing exists to stop all three threads from calling the same method, on the same object, at the same time. This is known as race condition, because the three threads are racing each other to complete the method.

Module 5

9 a) What are events, event listener and event source? Explain delegation event model used to handle events in Java. 7M

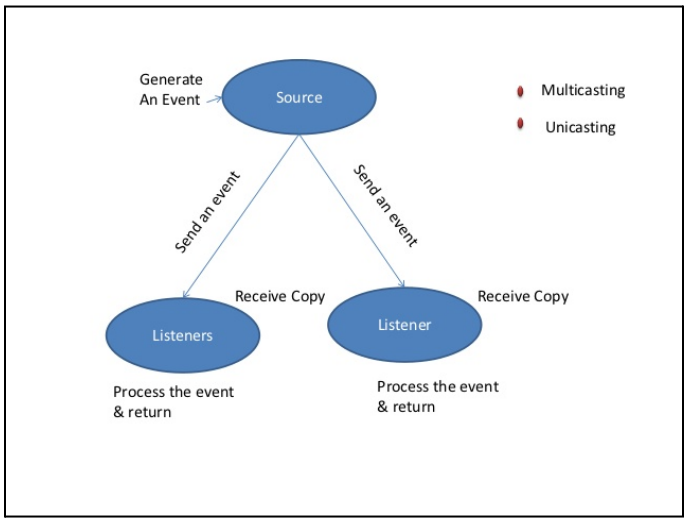
EVENTS:

- **An event is an object that describes a change of state in a source.**
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Some of the events that cause events to be generated are **pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.**
- Events may also occur that are not directly caused by interactions with a user interface. For example, **an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is complete.**



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. The general form is :

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

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 to listeners that register to receive them.

Some sources may allow only one listener to register. The general form of such a method is

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

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 -

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, 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:

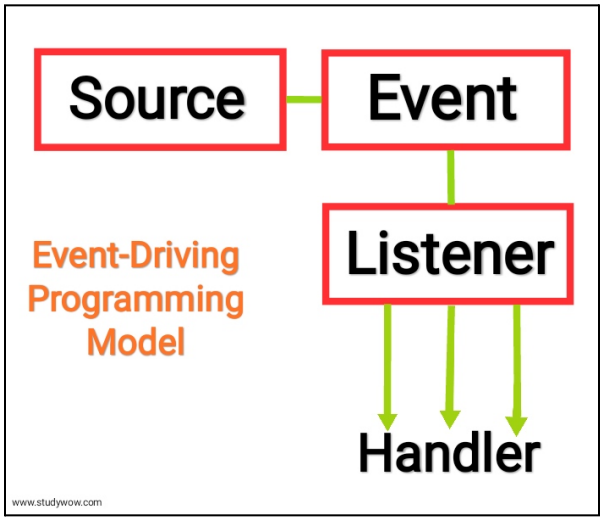
A listener is an object that is notified when an event occurs. It has two major requirements.

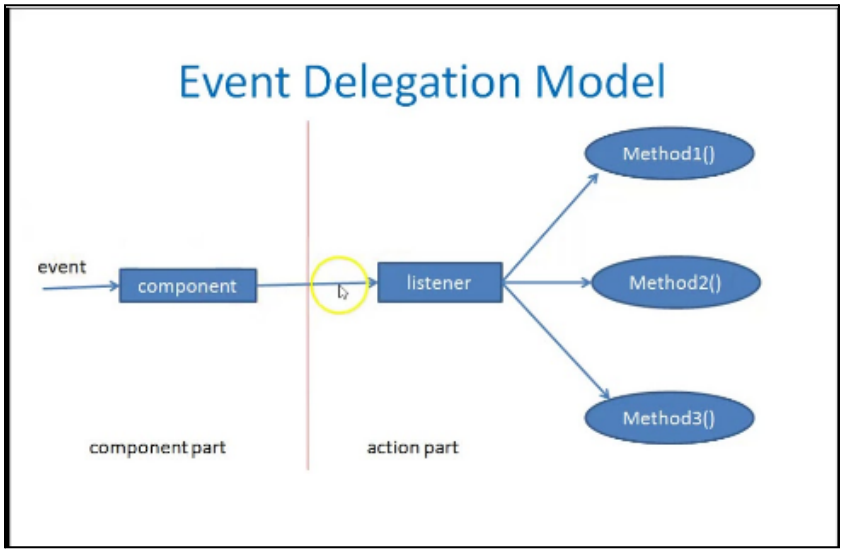
- First it must have registered with one or more sources to receive notifications about specific types of events.
- Second, it must implement methods to receive and process these notifications.

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 these events if it provides an implementation of this interface.

THE DELEGATION EVENT MODEL:

Event Handling in JAVA





The modern approach to handling events is based on the **delegation event model**.
Its concept is quite simple: a **source** generates an event and sends it to one or more **listeners**. The listener simply waits until it receives an event. Once the event is received, the listener processes the event and then returns.

The advantage of this design is that the **application logic that processes events is separated from the user interface logic that generates those events.** A user interface element is able to "**delegate**" the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification.

This provides an important benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events than the design used by the old Java 1.0 approach. Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

b) Write a Java program to handle mouse dragged and mouse moved events. 7M

HANDLING MOUSE EVENTS

To handle mouse events, we must implement the **MouseListener** and **MouseMotionListener** interfaces. The applet displays the current coordinates of the mouse in the applet's status window. Each time a button is pressed, the word **"Down" is displayed at the location of the mouse pointer.** Each time the button is released, the word "**Up**" is shown. If a button is clicked, the message, "**Mouse clicked" is displayed in the upper-left corner of the applet display area**.

As the mouse enters or exits the applet window, a message is displayed in the upper-left corner of the applet display area. When dragging the mouse, a '*' is shown, which tracks with the mouse pointer as it is dragged. The two variables **mouseX** and **mouseY**, store the location of the mouse when a mouse pressed, released or dragged event occurs. These coordinates are then used by paint() to display output at the point of these occurrences.

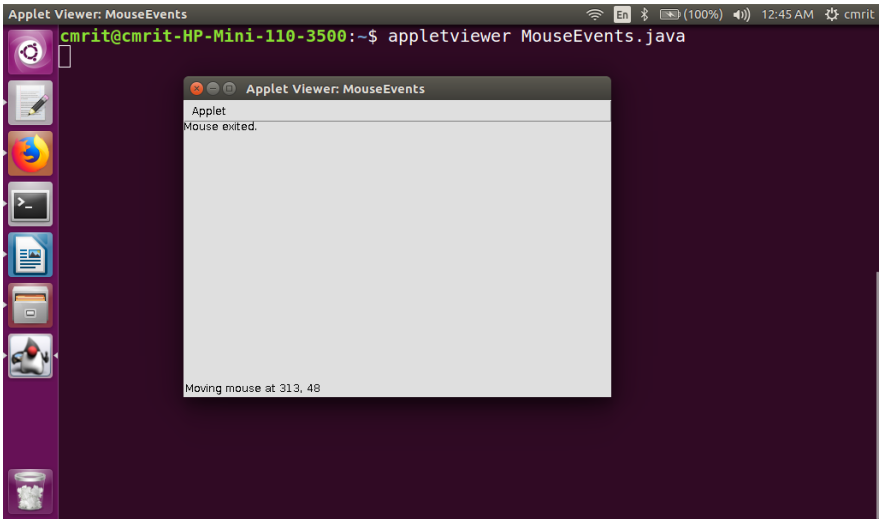
```
// demonstrate the mouse event handlers
import java.awt.*;    // Contains all AWT based event classes used by Delegation Event Model
import java.awt.event.*;      // Contains all Event Listener Interfaces
import java.applet.*;         // For Applets

/*
<applet code = "MouseEvents" width=500 height=300>
</applet>
*/

public class MouseEvents extends Applet
        implements MouseListener, MouseMotionListener  {
                                // MouseListener defines 5 methods
                                // MouseMotionListener defines 2 methods
```

```
String msg = "";
int mouseX = 0, mouseY = 0;  // Coordinates of mouse

// Listener must register with the source. General form -
// public void addTypeListener (TypeListener el)   Type is the name of the event

public void init()  {
        addMouseListener(this); //applet registers itself as listener for mouse events
        addMouseMotionListener(this);
}

// Handle mouse clicked
public void mouseClicked(MouseEvent me)  {
        // Save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse clicked";
        repaint();
}

// Handle mouse entered
public void mouseEntered (MouseEvent me)  {
        // Save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse Entered.";
        repaint();
}

// Handle mouse exited.
public void mouseExited(MouseEvent me)  {
        // Save coordinates
        mouseX = 0;
        mouseY = 10;
        msg = "Mouse exited.";
        repaint();
}

// Handle button pressed
```

```
    public void mousePressed(MouseEvent me)  {
        // Save coordinates
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "Down";
        repaint();
    }

    // Handle button released
    public void mouseReleased(MouseEvent me)  {
        // Save coordinates
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "Up";
        repaint();
    }

    // Handle mouse dragged
    public void mouseDragged(MouseEvent me)  {
        // Save coordinates
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "*";
        showStatus("Dragging mouse at  " + mouseX + ", " + mouseY);
        repaint();
    }

    // Handle mouse moved
    public void mouseMoved(MouseEvent me)  {
        // Show status
        showStatus( "Moving mouse at " + me.getX() + ", " + me.getY());
    }

    // Display message in applet window at current X,Y location
    public void paint(Graphics g)  {
        g.drawString(msg, mouseX, mouseY);
    }
}
```

$ javac MouseEvents.java
$ appletviewer MouseEvents.java

Output:



Program Explanation:

- The **MouseEvent** class extends **Applet** and implements both the **MouseListener** and **MouseMotionListener** interfaces. You may also want to implement MouseWheelListener, but it is not shown in this example program.
- These two interfaces contain methods that receive and process the various types of mouse events.
- The applet is both the source and listener for these events, because **Component** which supplies the **addMouseListener**() and **addMouseMotionListener**() methods, is a superclass of Applet. (Being both the source and the listener for events is a common situation for applets)
- Inside **init**(), the applet registers itself as a listener for mouse events. This is done by using **addMouseListener**() and **addMouseMotionListener**(), which are members of **Component**.
- The applet then implements all of the methods defined by the MouseListener and MouseMotionListener interfaces. These are the event handlers for the various mouse events. Each method handles its event and then returns.

c) Explain Adapter class with example 6M

ADAPTER CLASSES

- Java provides a special feature called an adapter class, which simplifies the creation of event handlers in certain situations.
- **An adapter class provides an empty implementation of all methods in an event listener interface.**
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- We have to define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which we are interested.
- For example, the **MouseMotionAdapter** class has two methods, **mouseDragged**() and **mouseMoved**(), which are the methods defined by the **MouseMotionListener** interface. If you are interested only in mouse dragged events, then you could simply extend **MouseMotionAdapter** and override **mouseDragged**(). The empty implementation of **mouseMoved**() would handle the mouse motion events for you.

Table list the commonly used adapter classes in **java.awt.event**

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener and (as of JDK 6) MouseMotionListener and MouseWheelListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener, WindowFocusListener, and WindowStateListener

Example program to demonstrate an adapter:

- It displays a message in the status bar of an applet or browser when the mouse is **clicked or dragged. No action is taken when the mouse is moved.**
- All other mouse events are silently ignored.
- The program has three classes
  1. **AdapterDemo extends Applet.**
     - Its init() method creates an instance of MyMouseAdapter and registers that object to receive notifications of mouse events.
     - It also creates an instance of MyMouseMotionAdapter and registers that object to receive notifications of mouse events

- Both of the constructors take reference to the applet as an argument.

2. **MyMouseAdapter extends MouseAdapter and overrides the mouseClicked() method.**
   - The other mouse events are silently ignored by code inherited from the MouseAdapter class.
3. **MyMouseMotionAdapter extends MouseMotionAdapter and overrides the mouseDragged() method.**
   - The other mouse motion event is silently ignored by code inherited from the MouseMotionAdapter class.

```
// demonstrate an Adapter
import java.awt.*;    // Contains all AWT based event classes used by Delegation Event Model
import java.awt.event.*;              // Contains all Event Listener Interfaces
import java.applet.*;                 // For Applets

/*
<applet code = "AdapterDemo" width=500 height=300>
</applet>
*/

public class AdapterDemo extends Applet {
        public void init()  {
                addMouseListener(new MyMouseAdapter (this));
                addMouseMotionListener(new MyMouseMotionAdapter(this));
        }
}

class MyMouseAdapter extends MouseAdapter {
        AdapterDemo adapterDemo;
        public MyMouseAdapter (AdapterDemo adapterDemo)  {
                this.adapterDemo = adapterDemo;
        }

        // Handle mouse clicked
        public void mouseClicked(MouseEvent me)  {
                adapterDemo.showStatus("Mouse Clicked");
        }
}

class MyMouseMotionAdapter extends MouseMotionAdapter {
        AdapterDemo adapterDemo;
```

```
        public  MyMouseMotionAdapter (AdapterDemo adapterDemo)  {
                this.adapterDemo = adapterDemo;
        }

        // Handle mouse dragged
        public void mouseDragged(MouseEvent me)  {
                adapterDemo.showStatus("Mouse Dragged");
        }
}
```

10 a) Explain the following with examples 4M
(i) JLabel
(ii) JTextField

JLabel

JLabel is Swing's easiest-to-use component. It creates a label. JLabel can be used to display text and/or an icon. It is a passive component in that it does not respond to user input. JLabel defines several constructors. Here are three of them:

JLabel(Icon icon)
JLabel(String str)
JLabel(String str, Icon icon, int align)

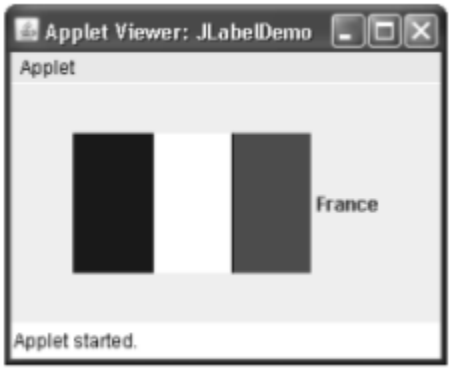
Here, str and icon are the text and icon used for the label. The align argument specifies the horizontal alignment of the text and/or icon within the dimensions of the label. It must be one of the following values: LEFT, RIGHT, CENTER, LEADING, or TRAILING. These constants are defined in the SwingConstants interface, along with several others used by the Swing classes.

Notice that icons are specified by objects of type Icon, which is an interface defined by Swing. The easiest way to obtain an icon is to use the ImageIcon class. ImageIcon implements Icon and encapsulates an image. Thus, an object of type ImageIcon can be passed as an argument to the Icon parameter of JLabel's constructor. There are several ways to provide the image, including reading it from a file or downloading it from a URL. Here is the ImageIcon constructor used by the example in this section:

The following applet illustrates how to create and display a label containing both an icon and a string. It begins by creating an ImageIcon object for the file france.gif, which depicts the flag for France. This is used as the second argument to the JLabel constructor. The first and last arguments for the JLabel constructor are the label text and the alignment. Finally, the label is added to the content pane.

```
// Demonstrate JLabel and ImageIcon.
import java.awt.*;
import javax.swing.*;
/*
<applet code="JLabelDemo" width=250 height=150>
</applet>
*/
public class JLabelDemo extends JApplet {
        public void init() {
          try {
                  SwingUtilities.invokeAndWait(new Runnable() {
                          public void run() {
                                  makeGUI();
                          }
                  }
          }
          catch (Exception exc) {
                  System.out.println("Can't create because of " + exc);
          }
        }
        private void makeGUI() {
                  // Create an icon.
                  ImageIcon ii = new ImageIcon("france.gif");
                  // Create a label.
                  JLabel jl = new JLabel("France", ii, JLabel.CENTER);
                  // Add the label to the content pane.
                  add(jl);
        }
}
```

Output from the label example is shown here:



JTextField

JTextField is the simplest Swing text component. It is also probably its most widely used ext component. JTextField allows you to edit one line of text. It is derived from JTextComponent, which provides the basic functionality common to Swing text components. JTextField uses the Document interface for its model.

Three of JTextField's constructors are shown here:

JTextField(int cols)
JTextField(String str, int cols)
JTextField(String str)

Here, str is the string to be initially presented, and cols is the number of columns in the text field. If no string is specified, the text field is initially empty. If the number of columns is not specified, the text field is sized to fit the specified string.

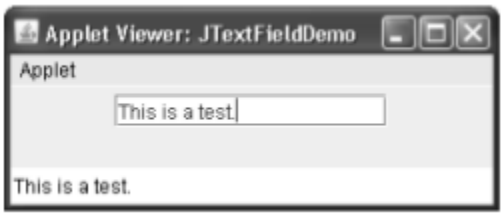
JTextField generates events in response to user interaction. For example, an ActionEvent is fired when the user presses ENTER . A CaretEvent is fired each time the caret (i.e., the cursor) changes position. (CaretEvent is packaged in javax.swing.event.) Other events are also possible. In many cases, your program will not need to handle these events. Instead, you will simply obtain the string currently in the text field when it is needed. To obtain the text currently in the text field, call getText( ).

The following example illustrates JTextField. It creates a JTextField and adds it to the content pane. When the user presses ENTER , an action event is generated. This is handled by displaying the text in the status window.

```
// Demonstrate JTextField.
import java.awt.*;
```

```
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="JTextFieldDemo" width=300 height=50>
</applet>
*/
public class JTextFieldDemo extends JApplet {
        JTextField jtf;
        public void init() {
                try {
                        SwingUtilities.invokeAndWait(new Runnable() {
                                public void run() {
                                        makeGUI();
                                }
                        });
                }
                catch (Exception exc) {
                        System.out.println("Can't create because of " + exc);
                }
                private void makeGUI() {
                        // Change to flow layout.
                        setLayout(new FlowLayout());
                        // Add text field to content pane.
                        jtf = new JTextField(15);
                        add(jtf);
                        jtf.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent ae) {
                                        // Show text when user presses ENTER.
                                        showStatus(jtf.getText());
                                }
                        });
                }
}
```

Output from the text field example is shown here:



b) Write a Java program to create a button, on clicking which displays "Welcome to VTU" 6M

```
// Handle an event in a Swing program.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class EventDemo
{
	JLabel jlab;
	EventDemo()
	{
		// Create a new JFrame container.
		JFrame jfrm = new JFrame("An Event Example");

		// Specify FlowLayout for the layout manager.
		jfrm.setLayout(new FlowLayout());

		// Give the frame an initial size.
		jfrm.setSize(220, 90);

		// Terminate the program when the user closes the application.
		jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		// Make two buttons.
		JButton jbtnAlpha = new JButton("Alpha");

		// Add action listener  for Alpha.
		jbtnAlpha.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent ae)
			{
				jlab.setText("Welcome to VTU");
			}
```

```
                });


                // Add the buttons to the content pane.
                jfrm.add(jbtnAlpha);

                // Create a text-based label.
                jlab = new JLabel("Press a button.");

                // Add the label to the content pane.
                jfrm.add(jlab);

                // Display the frame.
                jfrm.setVisible(true);
        }
        public static void main(String args[])
        {
                // Create the frame on the event dispatching thread.
                SwingUtilities.invokeLater(new Runnable()
                {
                        public void run()
                        {
                                new EventDemo();
                        }
                });
        }
}
```

Output:
javac EventDemo.java
$ java EventDemo

c) Write a Java program to create a table with column heading as FirstName, LastName, and Age. Insert atleast three records in the table and display. 10M

```
// Demonstrate JTable.
import java.awt.*;
import javax.swing.*;
/*
<applet code="JTableDemo" width=400 height=200>
```

```
</applet>
*/
public class JTableDemo extends JApplet {
public void init() {
try {
SwingUtilities.invokeAndWait(
new Runnable() {
public void run() {
makeGUI();
}
}
);
} catch (Exception exc) {
System.out.println("Can't create because of " + exc);
}
}
private void makeGUI() {
// Initialize column headings.
String[] colHeads = { "FirstName", "LastName", "Age" };
// Initialize data.
Object[][] data = {
{ "Sanchari", "Saha", "18" },
{ "Shyamasree", "Gosh", "18" },
{ "Savitha", "Nadig", "18" }
};

// Create the table.
JTable table = new JTable(data, colHeads);
// Add the table to a scroll pane.
JScrollPane jsp = new JScrollPane(table);
// Add the scroll pane to the content pane.
add(jsp);
}
}
```