Course Code : MCS-024
Course Title: Object Oriented Technologies and Java Programming
Assignment Number : MCA (2)/024/Assign/
Assignment Marks : 100
Maximum Marks : 25%

There are eight questions in this assignment which carries 80 marks. Rest of 20 marks are for viva-voce. Answer all the questions. Write and execute the program given in this assignment. Also in

your programs give appropriate comments to increase understandability. Please go through the

guidelines regarding assignments given in the Programme Guide for the format of presentation.

Question 1 : a) What is Object Oriented Paradigm? Explain features of Object Oriented Paradigm.

Why Object Oriented Programming are preferred over structured programming ? Solution :

Oops is a process of solving problems in computers compared to the procedural language programming such as in C.

Object-oriented programming is a programming paradigm

that uses "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs. Programming techniques may include

features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance.

Following are the fundamental features we get:

Encapsulation
Inheritance
Re-usability
Information Hiding

Encapsulation:

The way we make a logical boundary around behaviors (methods) and data (properties) they work

on is called Encapsulation.

Inheritance:

Objects in OOP relate to each other in one way or another, the relationship in most of the case is

parent/child relationship. The child objects inherent all the functionalities (methods) and data

(properties) of their parents.

Re-usability

Along with inheritance, some other phenomena like method overloading and overriding, provide code-reuse, which is known to be a very basic feature of object oriented programming.

Information Hiding

When we have ways to reuse our code through one way or other, we are also in need of some security

regarding our source code. To protect it from unauthorized access/ alteration. In object oriented programming, this is called Information Hiding

Why Object Oriented Programming are preferred over structured programming ?

One of the principal advantages of object-oriented programming techniques over procedural

programming techniques is that they enable programmers to create modules that do not need

to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

b) Explain basic features of Java programming language Solution :

Various features of java programming language are as given below :

Simple :

Java is Easy to write and more readable and eye catching.

Java has a concise, cohesive set of features that makes it easy to learn and use.
Most of the concepts are drew from C++ thus making Java learning simpler.

Secure :

Java program cannot harm other system thus making it secure.
Java provides a secure means of creating Internet applications.
Java provides secure way to access web applications.

Portable :

Java programs can execute in any environment for which there is a Java run-time system.(JVM)

Java programs can be run on any platform (Linux,Window,Mac) Java programs can be transferred over world wide web (e.g applets)

Object-oriented :

Java programming is object-oriented programming language.
Like C++ java provides most of the object oriented features.
Java is pure OOP. Language. (while C++ is semi object oriented)

Robust :

Java encourages error-free programming by being strictly typed and performing run-time checks.

Multithreaded :

Java provides integrated support for multithreaded programming.

Architecture-neutral :

Java is not tied to a specific machine or operating system architecture.
Machine Independent i.e Java is independent of hardware .

Interpreted :

Java supports cross-platform code through the use of Java bytecode.
Bytecode can be interpreted on any platform by JVM.

High performance :

Bytecodes are highly optimized.
JVM can executed them much faster .

Distributed :

Java was designed with the distributed environment.
Java can be transmit,run over internet.

Dynamic :

Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time.

Question 2 : a) What is static method? Explain why main method in Java is always static .

Solution : A Static method in Java is one that belongs to a class rather than an object of a class.

Normal methods of a class can be invoked only by using an object of the class but a Static method
can be invoked Directly.

Example:

```
public class A {
.....
public static int getAge(){
....
}
}

public class B {
```

.....
int age = A.getAge();
}

In class B when we wanted the age value we directly called the method using the instance of the
class instead of instantiating an object of the class.
A static method can access only static variables. The reason is obvious. Something that is common to
a class cannot refer to things that are specific to an object...

The main method needs to be static so your class does not need to be instantiated in order to run. When you run your application, the JVM can not create an instance of the class. That can
only be done once the main method is called. Methods of classes that can be called without the
class being instantiated are static.

b) What are different bitwise operators available in Java? Write a Java program to explain the use
of bitwise operators.
Solution :

Bitwise Operators
The Bitwise Operators the integer types, long, int, short, char, and byte.
The Bitwise Logical Operators Bitwise AND (&)
Bitwise OR (|)

Bitwise XOR (^)
Left shift (<<)
Bitwise complement (~): inverts ones and zeros in a number Demonstrate the bitwise logical operators
All bitwise operators in action
Bitwise Operator Assignments
The Left ShiftThe Right Shift
Use bitwise operator to create hash code

Example :--

```
class Main {
public static void main(String[] a) {

System.out.println(9 << 7);
```

```
}
}
```
output: 1152

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

int a = 32;
a = a >> 2;

System.out.println(a);

}
}
```
output: 8

c) What is constructor? Explain constructor overloading. Write a program in java to create Book class and define its constructor(s). Solution :

Constructor - A constructor is a special method that is used to initialize a newly created objectand is called just after the memory is allocated for the objectIt can be used to initialize the objects ,to required ,or default valuesat the time of object creationIt is not mandatory for the coder to write a constructor for the class

If no user defined constructor is provided for a class, compiler initializes member variables to its default values.

numeric data types are set to 0
char data types are set to null character(‘’)
reference variables are set to null

In order to create a Constructor observe the following rules

1 - It has the same name as the class
2 - It should not return a value not even void

Constructor overloading
Constructor overloading is a technique in Java in which a class can have any number of

constructors

that differ in parameter lists.The compiler differentiates these constructors by taking into account

the number of parameters in the list and their type.
Examples of valid constructors for class Account are

Account(int a);
Account (int a,int b);
Account (String a,int b);

Program in java to create Book class and define its constructor(s).

```
class BOOK
{
int name, author;
double year;

public Time() {
this.name = 0;
this.auther = 0;
this.year = 0.0;
}
```


Question 3 : a) What is overloading of methods? Explain with an example how overloading of methods is different from overriding of methods.

Solution :

Method Overloading

Method **overloading** means having two or more methods with the same name but different signatures in the same scope. These two methods may exist in the same class or anoter one in

base class and another in derived class.

```
class Person
{
private String firstName;
private String lastName;
Person()
{
this.firstName = "";
this.lastName = "";
}
Person(String FirstName)
{
this.firstName = FirstName;
```

this.lastName = "";
}
Person(String FirstName, String LastName)
{
this.firstName = FirstName;
this.lastName = LastName;

}
}

Calling Overloaded Methods.

Person(); // as a constructor and call method without parameter

Person(userFirstName); // as a constructor and call method with one parameter(like User's first Name)

Person(userFirstName,userLastName); // as a constructor and call method with one parameter(like User's first Name)

When to use Method Overloading?

Generally, you should consider **overloading** a method when you have required same reason that take different signatures, but conceptually do the same thing.

Method Overriding

Method overriding means having a **different implementation of the same method** in the **inherited class**. These two methods would have the **same signature, but different implementation**. One of these would exist in the **base class** and another in the **derived class**.

These cannot exist in the same class.

Overriding methods
Overriding method definitions

In a derived class, if you include a method definition that has the same name and exactly the same number and types of parameters as a method already defined in the base class, this new definition replaces the old definition of the method. Explanation

A subclass inherits methods from a superclass. Sometimes, it is necessary for the subclass to modify the methods defined in the superclass. This is referred to as method overriding. The fol owing example demonstrates method overriding.

Step 1

In this example we wil define a base class called Circle

class Circle

```
{
//declaring the instance variable
protected double radius;
public Circle(double radius)
{
this.radius = radius;
}
// other method definitions
here public double getArea()
{
return Math.PI*radius*radius;
}//this method returns the area of the
circle }// end of class circle
```

When the getArea method is invoked from an instance of the Circle class, the method returns the area of the circle.

Step 2

The next step is to define a subclass to override the getArea() method in the Circle class. The derived class wil be the Cylinder class. The getArea() method in the Circle class computes the area of a circle, while the getArea method in the Cylinder class computes the surface area of a cylinder.

The Cylinder class is defined below.

```
class Cylinder extends Circle
{
//declaring the instance variable
protected double length;
public Cylinder(double radius, double length)
{
super(radius);
this.length = length;
}
// other method definitions here
```



```
public double getArea()
{
// method overriden here
return 2*super.getArea()+2*Math.PI*radius*length;
}//this method returns the cylinder surface area
}// end of class Cylinder
```

When the overriden method (getArea) is invoked for an object of the Cylinder class, the new

definition of the method is called and not the old definition from the superclass(Circle).

Example code

This is the code to instantiate the above two classes
Circle myCircle;
myCircle = new Circle(1.20);
Cylinder myCylinder;
myCylinder = new Cylinder(1.20,2.50);

b) What is abstract class? Explain advantages of abstract class with the help of an example.

Solution :

Java Abstract classes are used to declare common characteristics of subclasses. An abstract class cannot be instantiated. It can only be used as a superclass for other classes that extend the abstract class. Abstract classes are declared with the abstract keyword. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

Like any other class, an abstract class can contain fields that describe the characteristics and methods thatdescribe the actions that a class can perform. An abstract class can include methods that contain no implementation. These are called abstract methods. The abstract method declaration must then end with a semicolon rather than a block. If a class has any abstract methods, whether declared or inherited, the entire class must be declared abstract. Abstract methods are used to provide a template for the classes that inherit the abstract methods.

Abstract classes cannot be instantiated; they must be subclassed, and actual implementations must be provided for the abstract methods. Any implementation specified can, of course, be overridden by additional subclasses. An object must have an implementation for all of its methods. You need to create a subclass that provides an implementation for the abstract method.

A class abstract Vehicle might be specified as abstract to represent the general abstraction of a vehicle, as creating instances of the class would not be meaningful. abstract class Vehicle {

int numofGears;

String color;

abstract boolean hasDiskBrake();

abstract int getNoofGears();
}
Example of a shape class as an abstract
class abstract class Shape {

```
public String color;

public Shape() {

}

public void setColor(String c) {


color = c;

}

public String getColor() {


return color;

}

abstract public double area();
}
```

We can also implement the generic shapes class as an abstract class so that we can draw lines, circles, trianglesetc. All shapes have some common fields and methods, but each can, of course, add more fields and methods. The abstract class guarantees that each shape will have the same set of basic properties. We declare this class abstract because there is no such thing as a generic shape. There can only be concrete shapes such as squares, circles, triangles etc.

```
public class Point extends Shape {


static int x, y;

public Point() {


x = 0;


y = 0;

}
```

```
public double area() {


return 0;
```



```
}

public double perimeter() {


return 0;

}

public static void print() {


System.out.println("point: " + x + "," + y);

}

public static void main(String args[]) {


Point p = new Point();


p.print();

}
}
```

Output
point: 0, 0

Notice that, in order to create a Point object, its class cannot be abstract. This means that all of the abstractmethods of the Shape class must be implemented by the Point class.

The subclass must define an implementation for every abstract method of the abstract superclass, or the subclass itself will also be abstract. Similarly other shape objects can be created using the generic Shape Abstract class.

A big Disadvantage of using abstract classes is not able to use multiple inheritance. In the sense, when a class extends an abstract class, it can't extend any other class.

Question 4 : a) What is polymorphism? Write a java program to show the

advantage of polymorphism Solution :

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP

occurs when a parent class reference is used to refer to a child class object.

Any java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Example:

Let us look at an example.

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

Now the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above example:

A Deer IS-A Animal A
Deer IS-A Vegetarian
A Deer IS-A Deer
A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal: Deer d = new Deer();

```
Animal a = d;
Vegetarian v = d;
Object o = d;
```

All the reference variables d,a,v,o refer to the same Deer object in the heap.

Virtual Methods:

In this section, I will show you how the behavior of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

```
/* File name : Employee.java
*/ public class Employee
{
private String name;
```

```
private String address;
private int number;
public Employee(String name, String address, int number)
{
System.out.println("Constructing an Employee");
this.name = name;
this.address = address;
this.number = number;
}
public void mailCheck()
{
System.out.println("Mailing a check to " +
this.name + " " + this.address);
}
public String toString()
{
return name + " " + address + " " + number;
}
public String getName()
{
```



```
return name;
}
public String getAddress()
{
return address;
}
public void setAddress(String newAddress)
{
address = newAddress;
}
public int getNumber()
{
return number;
}
}
```

Now suppose we extend Employee class as follows: /* File name : Salary.java */

```
public class Salary extends Employee
{
private double salary; //Annual salary
public Salary(String name, String address, int number,
double salary)
{
super(name, address, number);
```

```
setSalary(salary);
}
public void mailCheck()
{
System.out.println("Within mailCheck of Salary class
"); System.out.println("Mailing check to " + getName()
+ " with salary " + salary);
}
public double getSalary()
{
return salary;
}
public void setSalary(double newSalary)
{
if(newSalary >= 0.0)
{
salary = newSalary;
}
}
public double computePay()
{
System.out.println("Computing salary pay for " + getName());
return salary/52;
}
}
```

Now you study the following program carefully and try to determine its output: /* File name : VirtualDemo.java */

```
public class VirtualDemo
{
public static void main(String [] args)
{
Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3,
3600.00); Employee e = new Salary("John Adams", "Boston, MA", 2,
2400.00); System.out.println("Call mailCheck using Salary reference --
"); s.mailCheck();
System.out.println("Call mailCheck using Employee reference--
"); e.mailCheck();
```



```
}
}
```

This would produce following result:

```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference -
- Within mailCheck of Salary class
```

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference-
- Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0

Here we instantiate two Salary objects . one using a Salary reference s, and the other using an Employee reference e.

While invoking *s.mailCheck()* the compiler sees mailCheck() in the Salary class at compile time, and the JVM
invokes mailCheck() in the Salary class at run time.

Invoking mailCheck() on e is quite different because e is an Employee reference. When the compiler sees *e.mailCheck()*, the compiler sees the mailCheck() method in the Employee class.

Here, at compile time, the compiler used mailCheck() in Employee to validate this statement. At run time, however, the JVM invokes mailCheck() in the Salary class.

This behavior is referred to as virtual method invocation, and the methods are referred to as virtual methods. All methods in Java behave in this manner, whereby an overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

b) What is package in Java ? Explain how to decide the need of package(s) in a system which is to
be developed using Java.

Solution :

Packages are used in Java in-order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier etc.

A Package can be defined as a grouping of related types(classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are::
java.lang - bundles the fundamental classes
java.io - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces etc. It is a good practice to group related classes implemented by you so that a programmers can easily determine that the classes, interfaces, enumerations, annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages.

Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Creating a package:

When creating a package, you should choose a name for the package and put a **package** statement with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The **package** statement should be the first line in the source file. There can be only one

package statement in each source file, and it applies to all types in the file.

If a package statement is not used then the class, interfaces, enumerations, and annotation types will be put into an unnamed package.

Example:

Let us look at an example that creates a package called **animals**. It is common practice to use lowercased names of packages to avoid any conflicts with the names of classes, interfaces.

Put an interface in the package *animals*:

```
/* File name : Animal.java */
package animals;

interface Animal {
public void eat();
public void travel();
}
```

Now put an implementation in the same package *animals*:

```
package animals;

/* File name : MammalInt.java */
public class MammalInt implements Animal{

public void eat(){
System.out.println("Mammal eats");
}

public void travel(){
System.out.println("Mammal travels");
}

public int
noOfLegs(){ return 0;
}

public static void main(String args[]){
MammalInt m = new MammalInt();
m.eat();
m.travel();
}
}
```

Now you compile these two files and put them in a sub-directory called **animals** and try to run as follows: $ mkdir animals

```
$ cp Animal.class MammalInt.class animals
$ java animals/MammalInt
Mammal eats
Mammal travels
```

Question 5: a) What is an exception? Explain haw an exception is handled in Java. Also explain

hierarchy of different exception classes in
Java. Solution :

An *exception* is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions

When an error occurs within a method, the method creates an object and hands it off to the runtime

system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to

the runtime system is called *throwing an exception*.

Java exception handling covers the basics of Java's exception throwing and catching mechanism, and exception hierarchies. These texts are:

Basic Try Catch Finally
Exception Hierarchies

```
public void openFile(){
FileReader reader =
null; try {
reader = new
FileReader("someFile"); int i=0;
while(i != -1){
i = reader.read();
System.out.println((char) i );
}
} catch (IOException e) {
//do something clever with the exception
} finally {
if(reader !=
null){ try {
reader.close();
} catch (IOException e) {
//do something clever with the exception
}
}
System.out.println("--- File End ---");
}
}
```

b) What is String class in java? Explain how it is different from String Buffer class. Also write a java program to find whether a given string is a palindrome or not.

Solution :

Java String Class is immutable, i.e. Strings in java, once created and initialized, cannot be changed on the same reference. A java.lang.String class is final which implies no class and extend it. The

java.lang.String class differs from other classes, one difference being that the String objects can be used with the += and + operators for concatenation.

Two useful methods for String objects are equals( ) and substring( ). The equals( ) method is used for testing whether two Strings contain the same value. The substring( ) method is used to obtain a

selected portion of a String.

Java provides the StringBuffer and String classes, and the String class is used to manipulate character strings that cannot be changed. Simply stated, objects of type String are read only and immutable.

The StringBuffer class is used to represent characters that can be modified.

The significant performance difference between these two classes is that StringBuffer is faster than String when performing simple concatenations. In String manipulation code, character strings are routinely concatenated. Using the String class, concatenations are typically performed as follows:

```
String str = new String ("Stanford ");
str += "Lost!!";
```

If you were to use StringBuffer to perform the same concatenation, you would need code that looks like this:

```
StringBuffer str = new StringBuffer ("Stanford
"); str.append("Lost!!");
```

program to find whether a given string is a palindrome or not.

```
import java.lang.*;
import java.io.*;
import java.util.*;
class Palindrome
{
```

```
public static void main(String arg[ ]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));


String word=br.readLine( );

int flag=1;
int left=0;
int right=word.length( )-1;

while(left<right)
{
if(word.charAt(left)!=word.charAt(right))
{


flag=0;

}
left++;
right--;
}

if(flag==1)
System.out.println("palindrome");
else
System.out.println("not palindrome");
}
}
```


Question 6 : a) What is multithreading ? Explain advantages of multithreaded programming with the help of a Java program. Solution :

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes.

Life Cycle of a Thread:

New: A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

Runnable: After a newly born thread is started, the thread becomes runnable. A thread in this

state is considered to be executing its task.

Waiting: Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread

signals the waiting thread to continue executing.

Timed waiting: A runnable thread can enter the timed waiting state for a specified interval of

time. A thread in this state transitions back to the runnable state when that time interval expires

or when the event it is waiting for occurs.

Terminated: A runnable thread enters the terminated state when it completes its task or otherwise terminates

Example

```
// Create a new thread.
class NewThread implements Runnable {
Thread t;
NewThread() {
// Create a new, second thread
t = new Thread(this, "Demo Thread");
System.out.println("Child thread: " + t);
t.start(); // Start the thread
}

// This is the entry point for the second thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println("Child Thread: " + i);
// Let the thread sleep for a
while. Thread.sleep(500);
}
} catch (InterruptedException e) {
```

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

public class writer {
public void writing() {
```



```
System.out.println("Child interrupted.");
```

```
}
System.out.println("Exiting child thread.");
}
}

class ThreadDemo {
public static void main(String args[]) {
new NewThread(); // create a new
thread try {
for(int i = 5; i > 0; i--) {
System.out.println("Main Thread: " +
i); Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

b) What is I/O stream in Java? Write a program in Java to create a file to store a Java program in it.

Solution :

The Java I/O means Java Input/Output. It is provided by the java.io package. This package has an

InputStream and OutputStream. Java InputStream is defined for reading the stream, byte stream

and array of byte stream.

```
    try {
//What ever the file path is.
        File statText = new File("E:/Java/Reference/bin/images/statsTest.txt");
        FileOutputStream is = new FileOutputStream(statText);
        OutputStreamWriter osw = new OutputStreamWriter(is);
        Writer w = new BufferedWriter(osw);
        w.write("POTATO!!!");
        w.close();
    } catch (IOException e) {
        System.err.println("Problem writing to the file statsTest.txt");
    }
}

public static void main(String[]args) {
    writer write = new writer();
    write.writing();
}
}
```

www.IGNOUHelp.in

Question 7 : a) How a Java Applet is different from Java Application program? Create an Applet program to display current date and time. Solution :

All Java programs can be classified as **Applications** and **Applets**. The striking differences are that applications contain main() method where as applets do not. One more is, applications can be executed at DOS prompt and applets in a browser. We can say, an applet is an **Internet application**.

Applet is a Java program executed by a browser. The position of applets in software world is they occupy the client-side position in Web communication. On the server-side, you guess, another Java program comes, **Servlets**. Applets on client-side and servlets on server-side makes Java a truly " **Internet-based language**" . To execute applets, the browsers come with JRE (Java Runtime Environment). The browsers with Java Runtime Environment (or to say, JVM) loaded are known as **Java enabled browsers**.

Applications Vs. Applets

Feature	Application	Applet
main() method	Present	Not present
Execution	Requires JRE	Requires a browser like Chrome
Nature	Called as stand-alone application as application can be executed from command prompt	Requires some third party tool help like a browser to execute
Restrictions	Can access any data or software available on the system	cannot access any thing on the system except browser's services
Security	Does not require any security	Requires highest security for the system as they are untrusted

Applet program to display current date and time.

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;

public class ClockApplet extends Applet implements Runnable{
Thread t,t1;
public void start(){
t = new Thread(this);
t.start();
}

public void run(){
t1 = Thread.currentThread();
while(t1 == t){
repaint();
```

```
try{
t1.sleep(1000);
}catch(InterruptedException e){}
}
}

public void paint(Graphics g){
Calendar cal = new GregorianCalendar();
Date d=cal.getTime();
SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy
hh:mm:ss"); g.drawString(sdf.format(d), 20, 30);
}
}
```


To call this applet, you need to create an html file <HTML>

```
<BODY>
<div align = "center">
<APPLET CODE = "ClockApplet" WIDTH = "500" HEIGHT = "400"></APPLET>
</div>
</BODY>
</HTML>
```

b) What is layout manager? Which is the default layout manager? Differentiate among the way of managing components by 5 different layouts.

Solution :

A layout manager is an object that is used to organize components in a container.And it is an

interface in the java class libraries that describes how a container and a layout manager communicate.

Border Layout is the default layout manager .

There are 5 main Layouts in java :------

1. Border Layout:

- It has five areas for holding components: north, east, west, south and center
- When there are only 5 elements to be placed, border layout is the right choice

2. Flow Layout:

- Default layout manager.
- It lays out the components from left to right

3. Card Layout:

- Different components at different times are laid out

- Controlled by a combo box, determining which panel the Card layout displays

4. Grid Layout:

- A group of components are laid out I equal size and displays them in certain rows and columns

5. Grid Bag Layout:

- Flexible layout for placing components within a grid of cells
- Allows certain components to span more than one cell
- The rows of the grid are not of the same height and height

Question 8 : a) What is a socket ? Explain how a network socket is created using Java.

Solution :

A socket is one end-point of a two-way communication link between two programs running on the

network. Socket classes are used to represent the connection between a client program and a server

program. The java.net package provides two classes--Socket and ServerSocket--that implement the

client side of the connection and the server side of the connection, respectively.

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and

the port number on which the server is listening. To make a connection request, the client tries to

rendezvous with the server on the server's machine and port. The client also needs to identify itself

to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be

uniquely identified by its two endpoints. That way you can have multiple connections between your

host and the server.

b) What is DNS? Explain components of DNS.

Solution :

The **DNS** translates Internet domain and host names to IP addresses. DNS automatically converts the names we type in our Web browser address bar to the IP addresses of Web servers hosting those sites.

DNS implements a distributed database to store this name and address information for all public hosts on the Internet. DNS assumes IP addresses do not change (are statically assigned rather than dynamically assigned).

The DNS database resides on a hierarchy of special database servers. When clients like Web browsers issue requests involving Internet host names, a piece of software called the *DNS resolver*(usually built into the network operating system) first contacts a DNS server to determine the server's IP address. If the DNS server does not contain the needed mapping, it will in turn forward the request to a different DNS server at the next higher level in the hierarchy. After potentially several forwarding and delegation messages are sent within the DNS hierarchy, the IP

address for the given host eventually arrives at the resolver, that in turn completes the request over Internet Protocol.

DNS additionally includes support for *caching* requests and for *redundancy*. Most network operating systems support configuration of primary, secondary, and tertiary DNS servers, each of which can service initial requests from clients. Internet Service Providers (ISPs) maintain their own DNS servers and use DHCP to automatically configure clients, relieving most home users of the burden of DNS configuration.

DNS COMPONENTS:

The DNS consists of three components. The first is a "Name Space" that establishes the syntactical rules for creating and structuring legal DNS names. The second is a "Globally Distributed Database" implemented on a network of "Name Servers". The third is "Resolver" software, which understands how to formulate a DNS query and is built into practically every Internet-capable application.

(A) Name Space:

The DNS "Name Space" is the familiar inverted tree hierarchy with a null node named "" at the top. The child nodes of the root node are the Top Level Domains (TLDs)-.com, .net, .org, .gov,

.mil-and the country code TLDs, including .jp, .uk, .us, .ca, and so forth. Node names, known as labels, can be as many as 63 characters long, with upper- and lower-case alphabetical letters, numerals, and the hyphen symbol constituting the complete list of legal characters. Labels cannot begin with a hyphen. Upper- and lower-case letters are treated equivalently. A label can appear in multiple places within the name space, but no two nodes with the same label can have the same parent node: A node name must be unique among its siblings.

(B) Name Servers:

The second key component of the DNS is a globally connected network of "name servers". Each zone has a primary or master name server, which is the authoritative source for the zone's resource records. The primary name server is the only server that can be updated by means of local administrative activity. Secondary or slave name servers hold replicated copies of the primary server's data in order to provide redundancy and reduce the primary server's workload.

Furthermore, name servers generally cache data they have looked up, which can greatly speed up subsequent queries for the same data. Name servers also have a built-in agent mechanism that knows where to ask for data it lacks. If a name server can't find a domain within its zone, it sends the query a step closer to the root, which will resend it yet a step closer if it can't find the domain itself. The process repeats until it reaches a TLD, which ensures that the entire depth of the name space will be queried if necessary.

The combination of all the DNS name servers and the architecture of the system creates a remarkable database. There are more than 32 million domain names in the popular TLDs for which the whois utility works. Nominum, whose chief scientist, Paul Mockapetris, invented DNS, claims that there are more than 100 million domain names stored and that the system can easily handle 24,000 queries per second. The database is distributed-no single computer contains all the data. Nevertheless, data is maintained locally even though it's distributed globally, and any device connected to the IP network can perform lookups. The update serial number mechanism in each zone ensures a form of loose coherency on the network-if a record is out of date, the

querier knows to check a more authoritative name server.

(C) Resolver:

The third component of the DNS is the "resolver". The resolver is a piece of software that's implemented in the IP stack of every destination point, or "host" in IETF-speak. When a host is configured, manually or through DHCP, it's assigned at least one default name server along with its IP address and subnet mask. This name server is the first place that the host looks in order to resolve a domain name into an IP address. If the domain name is in the local zone, the default name server can handle the request. Otherwise, the default name server queries one of the root servers. The root server responds with a list of name servers that contain data for the TLD of the query. This response is known as a referral. The name server now queries the TLD name server and receives a list of name servers for the second-level domain name. The process repeats until the local name server receives the address for the domain name. The local server then caches the record and returns the address or other DNS data to the original querier.newdiv