CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 3–November 2018

Sub:	Java Programming						Code:	17MCA 32
Date:	19.11.2018	*Duration:*	*90 mins*	*Max Marks:*	*50*	*Sem:*	*3A*	*Branch:*	*MCA*

CMR
INSTITUTE OF
TECHNOLOGY

USN | 1 | C | | | | | | | | |

Internal Test-III – Nov. 2018

Sub:	Java Programming					Sub Code:	17MCA32	Branch:	MCA
Date:	19/11/18	Duration:	90 min's	Max Marks:	50	Sem / Sec:	3A		OBE

Answer any FIVE FULL Questions taking one from each part

		MARKS	CO	RBT
	Part I			
1(a)	Define method overloading and constructor overloading? Can final methods be overridden .Justify your answer.	[6]	CO1	L1
(b)	What is an array? Write the array declaration syntax for multi dimensional arrays.	[4]	CO1	L1,L2
	(OR)			
2 (a)	Explain wait(),notify(), notifyAll() in thread communication with example.	[6]	CO3	L2
(b)	What is thread priority? Explain it with an example.	[4]	CO3	L1,L2
	Part II			
3 (a)	What is an exception in java? Explain multi catch block statements with an example	[4]	CO3	L2
(b)	Explain the usage of keywords try, throw, catch, finally, throws with their syntax.	[6]	CO3	L2
	(OR)			
4 (a)	What is an enumeration? Explain the usage of values() and valueOf() methods.	[5]	CO4	L1
(b)	What is auto boxing and unboxing? Explain with an example	[5]	CO4	L1
	Part III	[6]	CO4	L1
5(a)	What is an applet? Explain the life cycle of an applet using an example.			
(b)	Write a java program to display the chess board pattern using applet.	[4]	CO4	L2
	(OR)			
6(a)	Explain any three object oriented features of java.	[6]	CO1	L2
(b)	Differentiate between abstract class and interface.	[4]	CO3	L2
	Part IV			
7(a)	What are collections? Write a program to demonstrate the linked list class	[05]	CO5	L3
(b)	What is a package? Explain the access protection for class members with respect to packages	[05]	CO3	L2
	(OR)			

8.	Explain the order of constructor execution in multi level hierarchy of classes	[10]	CO2	L2
	Part V			
9	Explain about the two types of TCP sockets used for networking in java.	[10]	CO5	L2
	(OR)			
10	Write a JAVA program which uses Datagram Socket for Client Server Communication	[10]	CO5	L2

1.a) What is method overloading and constructor overloading?can final methods be overridden? Justify your answer.

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Method overloading **increases the readability of the program**.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

```
class Calculation
{
        void sum(int a,int b)
        {
                System.out.println(a+b);
        }
        void sum(int a,int b,int c)
        {
                System.out.println(a+b+c);
        }
 }
Class MethodOverload
{
        public static void main(String args[])
        {
                 Calculation obj=new Calculation();
                obj.sum(10,10,10);
                obj.sum(20,20);

        }
}
```

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.

EX:

```
class Student{
   int id;
   String name;
   int age;
   Student(int i,String n)
{
   id = i;
```

```
    name = n;
    }
    Student(int i,String n,int a)
{
    id = i;
    name = n;
    age=a;
   }
    void display()
{
        System.out.println(id+" "+name+" "+age);
}

    public static void main(String args[]){
    Student  s1 = new Student(111,"Karan");
    Student  s2 = new Student(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

While method overriding is one of Java's most powerful features, there will be times when We want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden. The following fragment illustrates final:

```
class A {
final void meth() {
}
System.out.println("This is a final method.");
}
class B extends A {
void meth() { // ERROR! Can't override.
}
System.out.println("Illegal!");
}
```

Because meth( ) is declared as final, it cannot be overridden in B. If you attempt to do so, a compile-time error will result.

Methods declared as final can sometimes provide a performance enhancement: The compiler is free to inline calls to them because it "knows" they will not be overridden by a subclass. When a small final method is called, often the Java compiler can copy the bytecode for the subroutine directly inline with the compiled code of the calling method, thus eliminating the costly overhead associated with a method call. Inlining is only an option with final methods. Normally, Java resolves calls to methods dynamically, at run time. This is called late binding. However, since final methods cannot be overridden, a call to one can be resolved at compile time. This is called early binding.

1(b) What is an array? Write the array declaration syntax for multi dimensional arrays

an array is a collection of similar type of elements that have a contiguous memory location. Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array. Array in java is index-based, the first element of the array is stored at the 0 index.

In Java, multidimensional arrays are actually arrays of arrays.
data is stored in row and column based index (also known as matrix form).
Syntax to Declare Multidimensional Array in Java
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
Example to instantiate Multidimensional Array in Java
int[][] arr=new int[3][3];//3 row and 3 column

Q2(a) Explain wait(),notify(), notifyAll() in thread communication with example.

Ans: Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:
wait()
notify()
notifyAll()

---

1) wait() method
Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

---

2) notify() method
Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:
public final void notify()

---

3) notifyAll() method
Wakes up all threads that are waiting on this object's monitor. Syntax:
public final void notifyAll()

Q2(b) What is thread priority? Explain it with an example.

Ans: Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defiend in Thread class:

public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY).
The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
 public void run(){
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
  }
 public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();
 }
}
```

3(a) What is an exception? Explain multiple catch block statements with an example program

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on.

In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in orderand the first one whose type matches that of the exception is executed. After one catch statement executes, the others are bypassed, and execution continues after the try/catch block. The following example traps two different exception types:

```
// Demonstrate multiple catch statements.
class MultiCatch {
public static void main(String args[]) {
try {
}
}
```

```
int a = args.length;
System.out.println("a = " + a);
int b = 42 / a;
int c[] = { 1 };
c[42] = 99;
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index oob: " + e);
}
System.out.println("After try/catch blocks.");
}
}
```

Q3(b) Explain the usage of keywords try, throw, catch, finally, throws with their syntax.

Ans: Try:

Try is used to guard a block of code in which exception may occur. This block of code is called guarded region

Syn:

```
try
{
Stmt-1;
}
```

Catch:

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occured is listed in the catch block then the exception is handed over to the catch block which then handles it.

Syn:

```
Catch(ExceptionType obj)
{
Stmt
}
```

Throw:

Throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.

Syn:

```
Throw new throwableinstance;
```

Throws:

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

Syn:

```
type method_name(parameter_list) throws exception_list
{
 //definition of method
}
```

Finally:

A finally keyword is used to create a block of code that follows a try block. A finally block of code always executes whether or not exception has occurred. Using a finally block, lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.
Syn:
Finally
{
Block of statements
}

Q4(a) What is an enumeration? Explain the usage of values() and valueOf().

Ans: **Enum in java** is a data type that contains fixed set of constants.
Java Enums can be thought of as classes that have fixed set of constants.
enum improves type safety
enum can be easily used in switch
enum can be traversed
enum can have fields, constructors and methods
enum may implement many interfaces but cannot extend any class because it internally extends Enum class

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }

public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);

}}
```

Q4(b) Explain Auto boxing and Unboxing with an example.

Ans: Autoboxing and Unboxing:
The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.
Advantage of Autoboxing and Unboxing:
No need of conversion between primitives and Wrappers manually so less coding is required.

Simple Example of Autoboxing in java:

```
class BoxingExample1{
  public static void main(String args[]){
    int a=50;
        Integer a2=new Integer(a);//Boxing

        Integer a3=5;//Boxing

        System.out.println(a2+" "+a3);
 }
```

```
}

  class UnboxingExample1{
  public static void main(String args[]){
    Integer i=new Integer(50);
        int a=i;

        System.out.println(a);
 }
}
```

5(a) What is an applet? Explain the life cycle of an applet using an example.

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

Advantages of Applet

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

It works at client side so less response time.

Secured

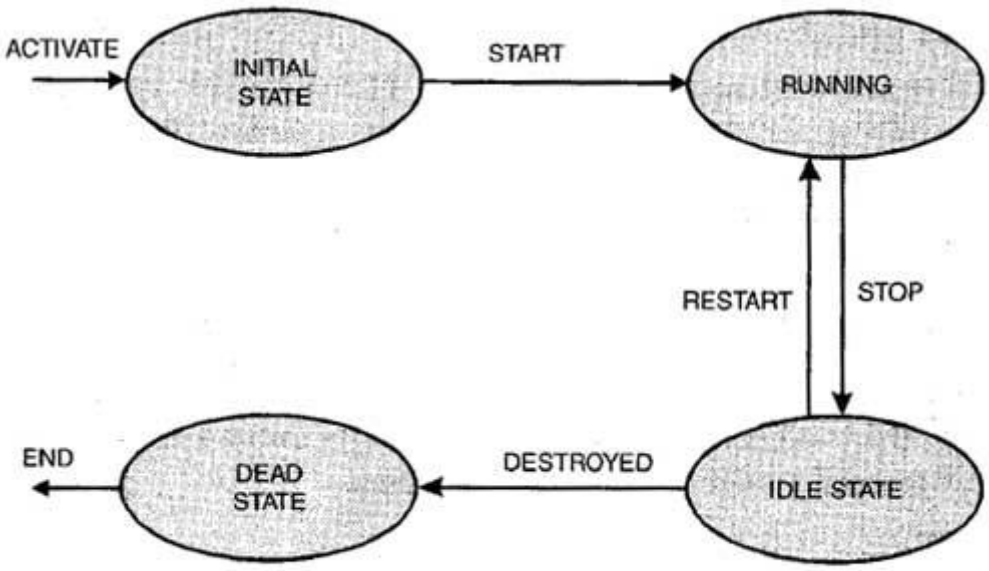
It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

Plugin is required at client browser to execute applet.

Lifecycle of an Applet:

Java applet inherits features from the class Applet. Thus, whenever an applet is created, it undergoes a series of changes from initialization to destruction. Various stages of an applet life cycle are depicted in the figure below:



Life Cycle of an Applet

Initial State

When a new applet is born or created, it is activated by calling init() method. At this stage, new objects to the applet are created, initial values are set, images are loaded and the colors of the images are set. An applet is initialized only once in its lifetime. It's general form is:

```
 public void init( )
//Action to be performed
}
```

Running State

An applet achieves the running state when the system calls the start() method. This occurs as soon as the applet is initialized. An applet may also start when it is in idle state. At that time, the start() method is overridden. It's general form is:

```
 public void start( )
{
//Action to be performed
}
```

Idle State

An applet comes in idle state when its execution has been stopped either implicitly or explicitly. An applet is implicitly stopped when we leave the page containing the currently running applet. An applet is explicitly stopped when we call stop() method to stop its execution. It's general form is:

```
 public void stope
{
//Action to be performed
}
```

Dead State

An applet is in dead state when it has been removed from the memory. This can be done by using destroy() method. It's general form is:

```
 public void destroy( )
{
//Action to be performed
}
```

Apart from the above stages, Java applet also possess paint( ) method. This method helps in drawing, writing and creating colored backgrounds of the applet. It takes an argument of the graphics class. To use The graphics, it imports the package java.awt.Graphics

Ex:

```
Public class LifeApp extends Applet
{
String s=" ";
Public void init()
{
        s=s+"init";
}
Public void start()
{
        s=s+"started";
}
Public void stop()
```

```
{
      s=s+”stop”;
}
Public void destroy()
{
      s=s+”destroyed”;
}
Public void paint(Graphics g)
{
Font f=new Font(“Arial”,Font.BOLd,20);
g.setFont(f);
g.drawString(s,20,40);
}
}
```

5(b) Write a java program to display the chess board pattern using applet.

```
/* Applet code="P10.class" height=200 width=300>
</applet>*/
import java.applet.Applet;
import java.awt.*;
public class P10 extends Applet {
   public void paint(Graphics g)
   {
      int row,x,y,m;

      for(row=0;row<8;row++)
      {
         for(m=0;m<8;m++)
         {
        x=m*40;
         y=row*40;
         if((row%2)==(m%2))
            g.setColor(Color.WHITE);
         else
            g.setColor(Color.BLACK);
         g.fillRect(x,y,40,40);
      }
      }

   }

}
```

6(a) Explain any three object oriented features of Java.

Ans: Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Q6(b) Differentiate between abstract class and interface.

oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

7(a) What are collections? Write a program to demonstrate the linked list class

Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java *Collections*.

Java *Collection* simply means a single unit of objects.

LinkedList class

1. LinkedList class extends **AbstractSequentialList** and implements **List**,**Deque** and **Queue**inteface.
2. LinkedList has two constructors.
3. LinkedList() *//It creates an empty LinkedList*
4.
5. LinkedList(Collection C) *//It creates a LinkedList that is initialized with elements of the Collection c*
6. It can be used as List, stack or Queue as it implements all the related interfaces.
7. They are dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.
8. It can contain duplicate elements and it is not synchronized.
9. Reverse Traversing is difficult in linked list.
10. In LinkedList, manipulation is fast because no shifting needs to be occurred.

Example of LinkedList class

```
import java.util.* ;
class Test
{
 public static void main(String[] args)
 {
  LinkedList< String> ll = new LinkedList< String>();
  ll.add("a");
  ll.add("b");
  ll.add("c");
  ll.addLast("z");
  ll.addFirst("A");
  System.out.println(ll);
 }
}
```

Output:

```
[A, a, b,c, z]
```

7(b) What is a package? Explain the access protection for class members with respect to packages

A java package is a group of similar types of classes, interfaces and sub-packages.
Advantage of Java Package
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.
The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:
private
default
protected
public

Access Modifier	within class	within package	outside package by subclass only	outside
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

EX:
class A{
private A(){}//private constructor
void msg()

```
{
System.out.println("Hello java");
}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

8) Explain the order of constructor execution in multi level hierarchy of classes

When a class hierarchy is created, constructors complete their execution in order of derivation, from superclass to subclass. Further, since super( ) must be the first statement executed in a subclass' constructor, this order is the same whether or not super( ) is used. If super( ) is not used, then the default or parameterless constructor of each superclass will be executed. The following program illustrates when constructors are executed: // Demonstrate when constructors are executed.

```
 // Create a super class.
class A {
 A() {
System.out.println("Inside A's constructor.");
 } }
// Create a subclass by extending class A.
class B extends A {
 B() {
 System.out.println("Inside B's constructor.");
 }
}
// Create another subclass by extending B.
class C extends B {
 C() {
 System.out.println("Inside C's constructor.");
 }
}
class CallingCons {
 public static void main(String args[]) {
 C c = new C();
 }
}
```

The output from this program is shown here:

```
 Inside A's constructor
 Inside B's constructor
 Inside C's constructor
```

9) Explain about the two types of TCP sockets used for networking in java.

Sockets provide the communication mechanism between two computers using TCP.

Java support two types of TCP sockets.

1. **ServerSocket -** ServerSocket class is a listener which waits for client to connect. ServerSocket is used for TCP/IP servers.
2. **Socket -** Socket class is for clients and they are used to connect to server sockets.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets −

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

10) Write a JAVA program which uses Datagram Socket for Client Server Communication

```
import java.net.*;
import java.io.*;

public class DGClient {

  public static int clientport = 50000;
  public static int buffer_size=1024;
  public static DatagramSocket ds;
```

```
  public static void dgClient() throws IOException
  {
      String str;
      byte[] buffer = new byte[buffer_size];
      System.out.println(" Receiving Data ");

      for(;;)
      {

          DatagramPacket p=new DatagramPacket(buffer,buffer_size);

          ds.receive(p);
          System.out.println("Receiving Data");

          str=new String(p.getData(),0,p.getLength());// used to receive data from datagram packet

          System.out.println(str);

          if(str.equals("stop"))
          {
             System.out.println("Client Stopping");
             break;
          }
      }
  }

  public static void main(String[] arg)
  {
     ds=null;

     try
     {
       ds= new DatagramSocket(clientport);
        dgClient();

      }
     catch(IOException e)
     {
        System.out.println("Communication error "+e);

     }
     finally
     {
        if(ds!=null)
           ds.close();

     }

        }

}

import java.net.*;
import java.io.*;
public class DGserver {
```

```
    public static int clientport= 50000;
    public static int serverport=50001;

    public static DatagramSocket ds;

    public static void dgserver() throws IOException
    {
       byte[] buffer;
       String str;

       BufferedReader conin = new BufferedReader(new InputStreamReader(System.in));

       System.out.println("Enter characters. Enter stop to quit");

       for(;;)
       {
           str=conin.readLine();
           buffer=str.getBytes();


           ds.send(new DatagramPacket(buffer,buffer.length,InetAddress.getLocalHost(),clientport));

           if(str.equals("stop"))
           {
              System.out.println("Server Quits");
              return;
           }
       }
    }

    public static void main(String arg[])
    {
       ds=null;
       try
       {
           ds=new DatagramSocket(serverport);
           dgserver();
           }
       catch(IOException ex)
       {


       }
       finally
       {
          if(ds!=null)
             ds.close();
       }
    }
}
```