CMR
INSTITUTE OF
TECHNOLOGY

<u>Internal Assessment Test 1–September 2018</u>

Sub:	Java Programming							Code:	17MCA 32
Date:	7.09.2018	*Duration:*	*90 mins*	*Max Marks:*	*50*	*Sem:*	*3*	*Branch:*	*MCA*

1. a) What is 'this' keyword? Demonstrate 'this' with a suitable program

The 'this' keyword is used for two purposes

1. It is used to point to the current active object.
2. Whenever the formal parameters and data members of a class are similar, to differentiate the data members of a class from formal arguments the data members of a class are preceeded with 'this'.

This(): It is used for calling current class default constructor from current class parameterized constructor.

This(…): It is used for calling current class parameterized constructor from other category constructors of the same class.

```
Ex:  class  Test
        {
                int a,b;
                Test()
                {
                        This(10);
                        System.out.println("I am from default constructor");
                        a=1;
                        b=2;
                        System.out.println("Value of  a="+a);
                        System.out.println("Value of b="+b);
                }
                Test(int x)
                {
                        This(100,200);
                        System.out.println("I am from parameterized constructor");
                        a=b=x;
                        System.out.println("Value of  a="+a);
                        System.out.println("Value of b="+b);
                }
                Test(int a,int b)
                {
                        System.out.println("I am from double parameterized constructor")
                        this.a=a+5;
                        This.b=b+5;
                        System.out.println("Value of  instance variable a="+this.a);
                        System.out.println("Value of instance variable b="+this.b);
                        System.out.println("Value of  a="+a);
                        System.out.println("Value of b="+b);
```

```
                    }
}
Class TestDemo3
{
          Public static void main(String k[])
          {
                    Test t1=new Test();
          }
}
```

b) What are instance and static variables? Explain with a suitable program.
Static Variables:

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.

- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.

- Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.

- Static variables are created when the program starts and destroyed when the program stops.

- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initializer blocks.

- Static variables can be accessed by calling with the class name as *ClassName.VariableName*.

- When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

Instance Variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class (when instance variables are given accessibility) should be called using the fully qualified name *.ObjectReference.VariableName*

Ex:

```
        Class Sample
        {
                int i,n;
                static int count=0;
                Sample(int a)
                {
                        n=a;
                }
                void even()
                 {
                        for(i=2;i<=n;i++)
                        {
                                If(i%2==0)
```

```
                                        Count++;
                        }
                }
}
}
Class Demo
{
        Public static void main(String k[])
        {
                Sample s=new Sample(50);
                s.even();
                System.out.println(Sample.count+even numbers are present upto"+s.n);
        }
}
```

2.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.

2(b) What is method overriding? Write a program in java to illustrate it.

If a sub class has the same method as declared in the parent class then it is called as method overriding.

Usage:

It is used to provide specific implementation of a method that is already provided by its super class used for run time polymorphism.

Rules:

1. Method must have same name and parameters as in parent class.
2. Method must be having is-a relation ship.

Ex:

```
class A
{
        int i,j;
        A(int a, int b)
        {
                i=a;
                j=b;
        }
        void show()
        {
                System.out.println(“i and j value”+i+”    “+j);
        }
}
Class B  extends A
{
        int k;
        B(int a, int b,int c)
        {
                Super(a,b);
                k=c;
        }
        void show()
        {
                System.out.println(“k  value”+k);
        }
}
Class Override
{
        Public static void main(String[] k)
        {
                B subob=new B(1,2,3);
```

```
                    Subob.show();
          }
}
```

3(a) Breifly explain varargs with example program.

The varrags allows the method to accept zero or muliple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

```
return_type method_name(data_type... variableName){}
Ex:
     class VarargsExample1{

      static void display(String... values){
       System.out.println("display method invoked ");
 }

 public static void main(String args[]){

 display();//zero argument
display("my","name","is","varargs");//four arguments
 }
}
```

3(b) what is a constructor? Explain parameterized constructors in detail.
It is a special type of method which is used to initialize the object. It is called when an instance of the object is created, and memory is allocated for the object.
These are the rules to be followed for defining the constructor.
Constructor name must be the same as its class name
A Constructor must have no explicit return type
A Java constructor cannot be abstract, static, final, and synchronized
There are two types of constructors in Java:
Default constructor (no-arg constructor)
Parameterized constructor
Parameterized constructor
A constructor that have parameters is known as parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.
Syn:

```
Classname(datatype  parameter1,datatype parameter2)
{
//initializes the instance variables with the values of parameters
}
```

Ex:

```
class Rectangle
{
 int length;
 int breadth;
 Rectangle(int len,int bre)
{
length  = len;
breadth = bre;
}
}
class RectangleDemo
 {
public static void main(String args[])
{
 Rectangle r1 = new Rectangle(20,10);
 System.out.println("Length of Rectangle : " + r1.length);
 System.out.println("Breadth of Rectangle : " + r1.breadth);
}
}
```

4(a) what is a polymorphism? Explain how to achieve runtime polymorphism in java

Polymorphism is derived from 2 greek words poly and morphis. Poly means many and morphis means forms. Polymorphism means one thing existing in many forms.

Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile time.

In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

```
// Dynamic Method Dispatch
class A {
}
}
void callme() {
System.out.println("Inside A's callme method");
class B extends A {
// override callme()
void callme() {
}
System.out.println("Inside B's callme method");
}
class C extends A {
// override callme()
void callme() {
}
System.out.println("Inside C's callme method");
}
class Dispatch {
public static void main(String args[]) {
A a = new A(); // object of type A
B b = new B(); // object of type B
C c = new C(); // object of type C
A r; // obtain a reference of type A
r = a; // r refers to an A object
r.callme(); // calls A's version of callme
r = b; // r refers to a B object
r.callme(); // calls B's version of callme
r = c; // r refers to a C object
r.callme(); // calls C's version of callme
}
}
```

(b)Write a program to find the maximum and minimum number of an array.

```
public class LargestSmallest
{
	public static void main(String[] args)
	{
		int a[] = new int[] { 23, 34, 13, 64, 72, 90, 10, 15, 9, 27 };

		int min = a[0]; //  assume first elements as smallest number
		int max = a[0]; //  assume first elements as largest number

		for (int i = 1; i < a.length; i++)  // iterate for loop from arrays 1st index
(second element)
		{
			if (a[i] > max)
			{
				max = a[i];
			}
			if (a[i] < min)
			{
				min = a[i];
			}
		}

		System.out.println("Largest Number in a given array is : " + max);
		System.out.println("Smallest Number in a given array is : " + min);
	}

}
```

5.a)What are static methods and static block? Write a program to illustrate static method and static block?

Static Method:

Any method which is declared with static keyword, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.

Example:

```
class Student9{

    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
    college = "BBDIT";
    }

    Student9(int r, String n){
    rollno = r;
    name = n;
    }

    void display (){System.out.println(rollno+" "+name+" "+college);
}
}
Class Demo
{
    static
   {
          System.out.println("Static block invoked");

        public static void main(String args[])
       {
                     Student9.change();
        Student9 s1 = new Student9 (111,"Karan");
     Student9 s2 = new Student9 (222,"Aryan");
   Student9 s3 = new Student9 (333,"Sonoo");

        s1.display();
        s2.display();
        s3.display();
        }
    }
```

b) In how many ways can we create a string in java? Why strings are immutable? Show the immutability with an example.

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

There are two ways to create String object:

1. By string literal
2. By new keyword

string objects are immutable. Immutable simply means unmodifiable or unchangeable.Once string object is created its data or state can't be changed but a new string object is created.

```
Class Test{

Public static void main(Strin[] k)

{

String s=”CMR”;

s.concat(“college”);

System.out.println(s);

}

}
```

6)a)Explain garbage collection and finalizers in java?

The technique of deallocating the memory automatically is called garbage collection. When an object

Is no longer used then JVM automatically deletes it to free the memory

Before an object is destroyed ,the resources linked to it should be freed. By using finalization, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

The java run time calls that method whenever it is about to recycle an object of that class. Inside the **finalize()** method we will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects

The **finalize()** method has this general form:

```
protected void finalize()
{
// finalization code here
}
```

6)b) What is for-each loop? Write its syntax.

For-each loop is used to access the elements of an array or list. We can access the elements directly instead of index. The type of the loop variable and array type should be the same.

```
Syntax: for(type var:array)
        {
                Body of the loop;
        }
```

7(a) Explain the difference between String, String Buffer and String Buider class.

Ans:

	String	S***tringBuffer***	***StringBuilder***
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Performance	Fast	Very slow	Fast

7(b) Explain all the methods to modify a String.

substring()

You can extract a substring using **substring( )**. It has two forms. The first is
String substring(int *startIndex*)
Here, *startIndex* specifies the index at which the substring will begin. This form returns a copy of the substring that begins at *startIndex* and runs to the end of the invoking string.
The second form of **substring( )** allows you to specify both the beginning and ending index of the substring:
String substring(int *startIndex*, int *endIndex*)
Here, *startIndex* specifies the beginning index, and *endIndex* specifies the stopping point. The string returned contains all the characters from the beginning index, up to, but not including, the ending index.
The following program uses **substring( )** to replace all instances of one substring with another within a string:

```
// Substring replacement.
class StringReplace {
public static void main(String args[]) {
String org = "This is a test. This is, too.";
String search = "is";
String sub = "was";
String result = "";
int i;
do { // replace all matching substrings
System.out.println(org);
i = org.indexOf(search);
if(i != -1) {
result = org.substring(0, i);
}
}
result = result + sub;
result = result + org.substring(i + search.length());
org = result;
}
} while(i != -1);
```

replace()

The **replace( )** method has two forms. The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

String replace(char *original*, char *replacement*)

Here, *original* specifies the character to be replaced by the character specified by *replacement*. The resulting string is returned. For example,

```
String s = "Hello".replace('l', 'w');
```

puts the string "Hewwo" into **s**.

The second form of **replace( )** replaces one character sequence with another. It has this general form:

String replace(CharSequence *original*, CharSequence *replacement*)

trim()

The **trim( )** method returns a copy of the invoking string from which any leading and trailing whitespace has been removed. It has this general form:

String trim( )

Here is an example:

```
String s = "   Hello World    ".trim();
```

This puts the string "Hello World" into **s**.

The **trim( )** method is quite useful when you process user commands. For example, the following program prompts the user for the name of a state and then displays that state's capital. It uses **trim( )** to remove any leading or trailing whitespace that may have inadvertently been entered by the user.

```
// Using trim() to process commands.
import java.io.*;
class UseTrim {
public static void main(String args[])
}
throws IOException
{
}
// create a BufferedReader using System.in
BufferedReader br = new
BufferedReader(new InputStreamReader(System.in));
```

```
String str;
System.out.println("Enter 'stop' to quit.");
System.out.println("Enter State: ");
do {
str = br.readLine();
str = str.trim(); // remove whitespace
if(str.equals("Illinois"))
System.out.println("Capital is Springfield.");
else if(str.equals("Missouri"))
System.out.println("Capital is Jefferson City.");
else if(str.equals("California"))
System.out.println("Capital is Sacramento.");
else if(str.equals("Washington"))
System.out.println("Capital is Olympia.");
// ...
} while(!str.equals("stop"));
```

8(a) Explain abstract classes.

A class which contains the **abstract** keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain *abstract methods* ie., methods with out body (public void get();)
- But, if a class have at least one abstract method, then the class **must**be declared abstract.
- If a class is declared abstract it cannot be instantiated.
- To use an abstract class you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class you have to provide implementations to all the abstract methods in it.

Ex:

```
 abstract class Bank{
     abstract int getRateOfInterest();
     }

     class SBI extends Bank{
     int getRateOfInterest(){return 7;}
     }
     class PNB extends Bank{
     int getRateOfInterest(){return 7;}
     }

     class TestBank{
     public static void main(String args[]){
     Bank b=new SBI();
     int interest=b.getRateOfInterest();
```

```
System.out.println("Rate of Interest is: "+interest+" %");
}}
```

8(b) what is inner class? Write a program to demonstrate inner class.

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.

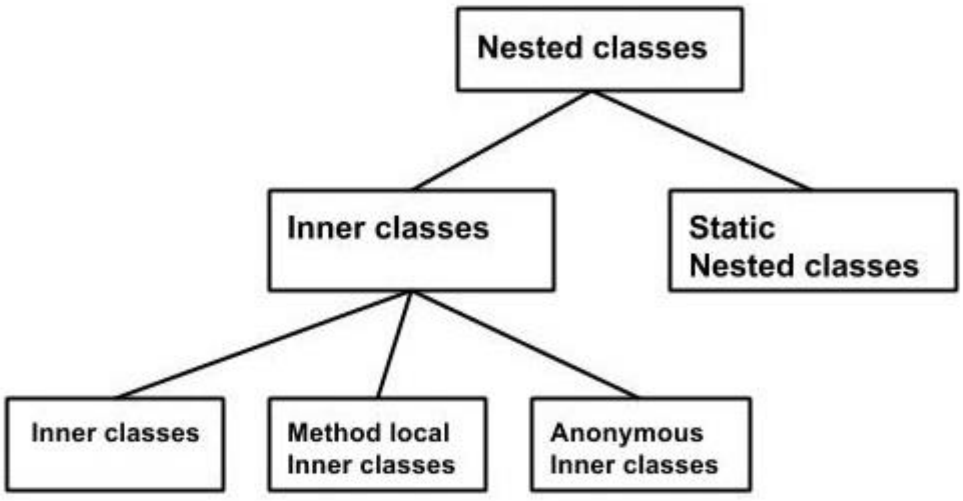
Syntax

Following is the syntax to write a nested class. Here, the class **Outer_Demo**is the outer class and the class **Inner_Demo** is the nested class.

```
class Outer_Demo {
  class Nested_Demo {
  }
}
```

Nested classes are divided into two types −

- **Non-static nested classes** − These are the non-static members of a class.
- **Static nested classes** − These are the static members of a class.



Inner Classes (Non-static Nested Classes)

Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier **private**, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.

Inner classes are of three types depending on how and where you define them. They are −

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

Inner Class

Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

Following is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.

Example

```
class Outer_Demo {
   int num;

   // inner class
   private class Inner_Demo {
      public void print() {
         System.out.println("This is an inner class");
      }
   }

   // Accessing he inner class from the method within
   void display_Inner() {
      Inner_Demo inner = new Inner_Demo();
      inner.print();
   }
}

public class My_class {

   public static void main(String args[]) {
      // Instantiating the outer class
      Outer_Demo outer = new Outer_Demo();

      // Accessing the display_Inner() method.
      outer.display_Inner();
   }
}
```

9(a) Explain how super class constructor is called using Super.

Ans: The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```java
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

Output:

```
animal is created
dog is created
```

9(b) 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.

10(a) Write a program in Java for String handling which performs the following:

i. **Checks the capacity of StringBuffer objects.**
ii. **Reverses the contents of a string given on console and converts the resultant string in upper case.**

iii Reads a string from console and appends it to the resultant string of ii.

SOURCE CODE:-

```
import java.util.Scanner;

public class stringclass {
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("MCA");
        System.out.println("length="+sb.length());
        System.out.println("total capacity="+sb.capacity());
        String s = new String("Aslam");
        s=s.toUpperCase();
        System.out.println(s);
        String reverse =new StringBuffer(s).reverse().toString();
        System.out.println("reversed string is"+reverse);
        Scanner user = new Scanner(System.in);
        System.out.println("enter any string");
        String s3=user.next();
        reverse=reverse.concat(s3);
        System.out.println(reverse);

    }

}
```

10.b Write a java program to perform sum of 2 objects and return an object as the result and display it.

```
Class Test
{
    int a,b;
    Test()
     {
         a=b=0;
     }
     Test(int a, int b)
     {
          this.a=a;
          this.b=b;
      }
      Test sum(Test T)
        {
                Test T11=new Test();
               T11.a=this.a+T.a;
                T11.b=this.b+T.b;
                 return T11;
         }
         Void display()
          {
               System.out.println("Value of a=" +a);
               System.out.println("Value of b="+b);
          }
}
Class SumDemo1
```

```
{
Public static void main(String k[])
{
          int  n1=Integer.parseInt(k[0]);
          int  n2=Integer.parseInt(k[1]);
          int  n3=Integer.parseInt(k[2]);
          int  n4=Integer.parseInt(k[3]);
          Test  t1=new Test(n1,n2);
          Test  t2=new Test(n3,n4);
           Test t3=new Test();
             t3=t1.sum(t2);
             t1.display();
              t2.display();
              t3.display();
}
}
```