USN										

Internal Assessment Test 2-JUNE. 2022

Sub:	**PROGRAMMING IN JAVA**			Sub Code:	18CS653	Branch:	ECE	
Date:	10-06-2022	Duration:	90 Minutes	Max Marks: 50	Sem / Sec:	**6/A, B,C,D**	**OBE**	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Discuss the types of Constructor in java with examples.	10	CO3	L2

- Java allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor
- A constructor initializes the instance variables of an object.
- It is called automatically immediately after the object is created but before the new operator completes.
  1) it is syntactically similar to a method:
  2) it has the same name as the name of its class
  3) it is written without return type, not even void; the default return type of a class

Types of Constructor

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor
2. Parameterized Constructor
3. Default Constructor

No-Arg Constructor: Similar to methods, a Java constructor may or may not have any parameters (arguments).
If a constructor does not accept any parameters, it is known as a no-argument constructor.

In Box example the dimensions of a box are automatically initialized when an object is constructed.

```
class Box {
      double width;
      double height;
      double depth;
      Box() {
               System.out.println("Constructing Box");
               width = 10; height = 10; depth = 10;
               }
               double volume() {
                        return width * height * depth;
                        }
               }
   public static void main(String args[]) {
               Box mybox1 = new Box();
               Box mybox2 = new Box();
               double vol;
               vol = mybox1.volume();
               System.out.println("Volume is " + vol);
```

```
                    vol = mybox2.volume();
                    System.out.println("Volume is " + vol);
                    }
          }
          it generates the following results:
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
                     constructor for the class is being called
```

Parameterized Constructor: The constructor which takes parameters while creating the object of a particular class. Here Box constructor is having width, height and depth parameters which will be initialized at the time of object creation which the supplied values.

```
 class Box {
          double width;
          double height;
          double depth;
          Box(double w, double h, double d) {
                    width = w; height = h; depth = d;
                              }
          double volume(){
                     return width * height * depth;
                              }
                    }
  class BoxDemo {
          public static void main(String args[]) {
                    Box mybox1 = new Box(10, 20, 15);
                    Box mybox2 = new Box(3, 6, 9);
                    double vol;
                    vol = mybox1.volume();
                    System.out.println("Volume is " + vol);
                    vol = mybox2.volume();
                    System.out.println("Volume is " + vol);
}
}
Volume is 3000.0
Volume is 162.0
```

Default Constructor:If we do not create any constructor, the Java compiler automatically creates a no-arg constructor during the execution of the program. This constructor is called default constructor.

2	Explain the following: a) Use of this keyword b) Garbage collector c) Finalize ()	10	CO3	L1

This Keyword:

- Sometimes a method will need to refer to the object that invoked it
- this is always a reference to the object on which the method was invoked
- Typically used to
  - Avoid variable name collisions
  - Pass the receiver as an argument

- Chain constructors
- Keyword this allows a method to refer to the object that invoked it.

- It can be used inside any method to refer to the current object:

```
        Box(double w, double h, double d) {
                this.width = w;
                this.height = h;
                this.depth = d;
                }
```

From the main function we can construct the Box object by:

```
public static void main(String args[]) {
        Box mybox1 = new Box(10,20,30);
        Box mybox2 = new Box(myBox1);  //here the myBox1 object has been
passed
```

This version of Box( ) operates exactly like the earlier version. The use of this is redundant, but perfectly correct. Inside Box( ), this will always refer to the invoking object. section. This version of Box( ) operates exactly like the earlier version. The use of this is redundant, but perfectly correct. Inside Box( ), this will always refer to the invoking object. when a local variable has the same name as an instance variable, the local variable hides the instance variable.

Garbage Collection:

- Garbage collection is a mechanism to remove objects from memory when they are no longer needed.
- Garbage collection is carried out by the garbage collector:

1) The garbage collector keeps track of how many references an object has.
2) when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
3) It removes an object from memory when it has no longer any references.
4) Thereafter, the memory occupied by the object can be allocated again.
5) The garbage collector invokes the finalize method.

Finalize() :

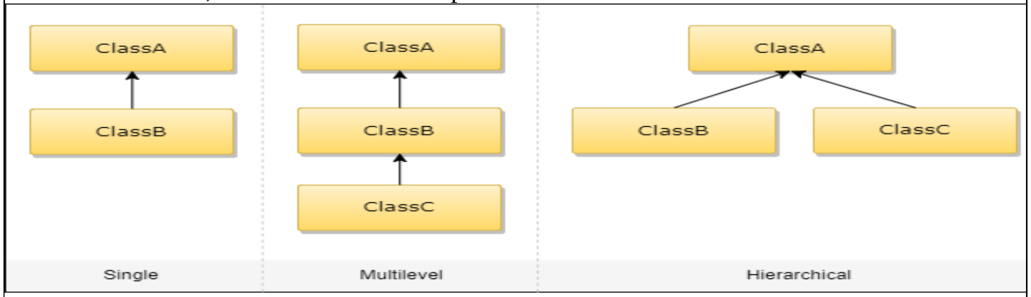
- Sometimes an object will need to perform some action when it is destroyed. To handle such situations, Java provides a mechanism called finalization
- the finalize method is invoked just before the object is destroyed:
- By using finalization, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.
- To add a finalizer to a class, we simply define the finalize( ) method. The
- Java run time calls that method whenever it is about to recycle an object of that class.
- Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed.
- Implemented inside a class as:
- `protected void finalize() { … }`
- Implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out.
- Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class
- It is important to understand that finalize( ) is only called just prior to garbage collection.

3	Write a java program which creates class Student (Rollno, Name, Number of subjects, Marks of each subject)(Number of subjects varies for each student) Write a parameterized constructor which initializes roll no, name & Number of subjects and create the array of marks dynamically. Display the details of all students with percentage.	10	CO2,CO3	L3

```
import java.io.*;
class Student
   {
      int rollno;
      String name;
      int number_of_subjects;
      int mark[];
     Student(int roll,String stud_name,int noofsub) throws IOException
        {
          rollno=roll;
          name=stud_name;
          number_of_subjects= noofsub;
          getMarks(noofsub);
        }
    public void getMarks(int nosub ) throws IOException
       {
          mark=new int[nosub];
          BufferedReader br= new BufferedReader (new InputStreamReader(System.in));
          for (int i=0; i<nosub;i++)
              {
               System.out.println(“Enter “+i+”Subject Marks.:=> “);
                 mark[i]=Integer.parseInt(br.readLine());
               System.out.println(“”);
               }
        }
  public void calculateMarks()
   {
      double percentage=0;
      String grade;
      int tmarks=0;
      for (int i=0;i<mark.length;i++)  {
          tmarks+=mark[i];
      }
    percentage=tmarks/number_of_subjects;
    System.out.println(“Roll Number :=> “+rollno);
    System.out.println(“Name Of Student is :=> “+name);
    System.out.println(“Number Of Subject :=> “+number_of_subjects);
    System.out.println(“Percentage Is :=> “+percentage);
    if (percentage>=70)
      System.out.println(“Grade Is First Class With Distinction “);
    else if(percentage>=60 && percentage<70)
      System.out.println(“Grade Is First Class”);
    else if(percentage>=50 && percentage<60)
     System.out.println(“Grade Is Second Class”);
     else if(percentage>=40 && percentage<50)
       System.out.println(“Grade Is Pass Class”);
    else
     System.out.println(“You Are Fail”);
       }
   }

class StudentDemo  {
      public static void main(String args[])throws IOException {
           int rno,no,nostud;
           String name;
           BufferedReader br= new BufferedReader (new InputStreamReader(System.in));
          System.out.println(“Enter How many Students:=> “);
           nostud=Integer.parseInt(br.readLine());
          Student s[]=new Student[nostud];
          for(int i=0;i<nostud;i++)    {
            System.out.println(“Enter Roll Number:=> “);
             rno=Integer.parseInt(br.readLine());
```

	System.out.println("Enter Name:=> ");
	name=br.readLine();
	System.out.println("Enter No of Subject:=> ");
	no=Integer.parseInt(br.readLine());
	s[i]=new Student(rno,name,no);
	}
	for(int i=0;i<nostud;i++) {
	s[i].calculateMarks();
	}
	}
	}			

4	What is Inheritance? What are the different types of Inheritance? Explain the use of super with a suitable programming example.
	
	Inheritance:
	
	• Allows the creation of hierarchical classifications.
	• For creating inheritance:
	nheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes.
	
	A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.
	• A class that is inherited is called a superclass. The class that does the inheriting is called a subclass.
	• Subclass is a specialized version of a superclass.
	• To inherit a class, you simply incorporate the definition of one class into another by
	using the extends keyword
	• The general form of a class declaration that inherits a superclass is shown here:
	Class subclass-name extends superclass-name {
	// body of class
	}
	A is a superclass for B (In next slide eg.), it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself.
	• Further, a subclass can be a superclass for another subclass.
	 ClassA
	ClassB
	ClassA
	ClassB
	ClassC
	ClassA
	ClassB
	ClassC
	Single
	Multilevel
	Hierarchical
	Java Inheritance
	
	Super:
	● super is a keyword.
	● It is used inside a sub-class method definition to call a method defined in the super class. Private methods of the super-class cannot be called. Only public and protected methods can be called by the super keyword.
	● It is also used by class constructors to invoke constructors of its parent class.
	● Super keyword are not used in static Method.
	
	Two general form of super:
	● The first calls the superclass' constructor.
	● The second is used to access a member of the superclass that has been hidden by a member of a subclass
		10	CO1,CO3	L2

- A subclass can call a constructor defined by its superclass by use of the following form of super:

                      super(arg-list);   //arg-list specifies any arguments needed by the
constructor in the superclass.

- super( ) must always be the first statement executed inside a subclass' constructor
- Example:

a) First use of super : to call super class constructor:

```
class Box{                                           class BoxWeight entends Box{
  double height;                                           double mass;
  double widht;                                            BoxWeight(double w, double h,
double d, double m) {
  double depth;                                              super(w,h,d);
 Box(double w, double h, double d) {                       mass=m ;
                    width = w;                           }
                    height = h;                        }
                    depth = d;

  double vol()                                   Here the value of height,width, depth in
Boxweight contructor
       {                                       has been initialized by super class constructor
with "super".
       // body of the function
     }
}
```

b) Second use of super : to call super class constructor:
- This second form of super is most applicable to situations when member names of a subclass hide members by the same name in the superclass
- This usage has the following general form:

               super.member   //Here, member can be either a method or an
instance variable.

```
class A {
     int i;
    }
// Create a subclass by extending class A.
 class B extends A {
         int i; // this i hides the i in A
         B(int a, int b) {
             super.i = a; // i in A
             i = b; // i in B
             }                                                                              class
UseSuper {
 void show() {
             System.out.println("i in superclass: " + super.i);
             public static void main(String args[]) {

             System.out.println("i in subclass: " + i);
subOb = new B(1, 2);
                  }
        }
subOb.show();}}
```

Output:

```
i in superclass: 1
i in subclass: 2
```

5	Write a short note on following terms · Static keyword · Dynamic method dispatching · Overloaded constructors	10	CO2	L2

Static:

There will be times when you will want to define a class member that will be used independently of any object of that class. Normally, a class member must be accessed only in conjunction with an object of its class. However, it is possible to create a member that can be used by itself, without reference to a specific instance. To create such a member, precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. You can declare both methods and variables to be static. The most common example of a static member is main( ). main( ) is declared as static because it must be called before any objects exist. Instance variables declared as static are, essentially, global variables. When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable.

Methods declared as static have several restrictions:

- They can only call other static methods.
- They must only access static data.
- They cannot refer to this or super in any way. (The keyword super relates to

  inheritance and is described in the next chapter.)

If you need to do computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.

Dynamic method dispatching:

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism.Let’s begin by restating an important principle: a superclass reference variable can refer to a subclass object. Java uses this fact to resolve calls to overridden methods at run time. Here is how. When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed. Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

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 Dispatch {

public static void main(String args[]) {

A a = new A(); // object of type A

B b = new B(); // object of type B

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

}

}
```

The output from the program is shown here:

```
Inside A’s callme method

Inside B’s callme method
```

Overloaded constructors:

In addition to overloading normal methods, you can also overload constructor methods. In fact, for most real-world classes that you create, overloaded constructors will be the norm not the exception. As construction overloading enables the creation of the object of a specific class in several ways, it is most commonly used in Java programs based on the requirement of the programmer. With the use of constructor overloading, objects can be initialized with different data types. Consider that an object with three class instance variables is taken as an example where a particular value is to be assigned to the second instance variable and the other variables are to be assigned default values. This can be accomplished by the declaration of multiple constructors according to the different signatures in the constituent class.

```
class Box {
```

```
double width;
double height;
double depth;
// constructor used when all dimensions specified
Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box() {
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when cube is created
Box(double len) {
width = height = depth = len;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
class OverloadCons {
public static void main(String args[]) {
// create boxes using the various constructors
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box();
Box mycube = new Box(7);
double vol;
```

	// get volume of first box
	vol = mybox1.volume();
	System.out.println("Volume of mybox1 is " + vol);
	// get volume of second box
	vol = mybox2.volume();
	System.out.println("Volume of mybox2 is " + vol);
	// get volume of cube
	vol = mycube.volume();
	System.out.println("Volume of mycube is " + vol);
	}
	}
	The output produced by this program is shown here:
	Volume of mybox1 is 3000.0
	Volume of mybox2 is -1.0
	Volume of mycube is 343.0
	As you can see, the proper overloaded constructor is called based upon the parameters
	specified when new is executed.			

6	Write a program to implement stack using class with the methods to push and pop the elements.
	
	// This class defines an integer stack that can hold 10 values.
	    class Stack {
	        int stck[] = new int[10];
	        int tos;
	// Initialize top-of-stack
	      Stack() {
	        tos = -1;
	      }
	// Push an item onto the stack
	    void push(int item) {
	        if(tos==9)
	          System.out.println("Stack is full.");
	       else
	         stck[++tos] = item;
	       }
	// Pop an item from the stack
	    int pop() {
	       if(tos < 0) {
	         System.out.println("Stack underflow.");
	         return 0;
	            }
	       else
	          return stck[tos--];
	         }
		10	CO2,CO3	L3

	}
	
	class TestStack {
	public static void main(String args[]) {
	Stack mystack1 = new Stack();
	Stack mystack2 = new Stack();
	// push some numbers onto the stack
	for(int i=0; i<10; i++)
	mystack1.push(i);
	for(int i=10; i<20; i++)
	mystack2.push(i);
	// pop those numbers off the stack
	System.out.println("Stack in mystack1:");
	for(int i=0; i<10; i++)
	System.out.println(mystack1.pop());
	System.out.println("Stack in mystack2:");
	for(int i=0; i<10; i++)
	System.out.println(mystack2.pop());
	}
	}			

7	Create a class named ‘Box' with data members to store height, width, length and two methods to print the volume and initialize the dimensions of Box using a constructor. Add the color of the box to the existing features. Create the objects of both classes and initialize the instance variable using parameterized constructor from main(). Again add one more feature of weight to Box and initialize the instance variable using super constructor. Finally display the volume, color, weight of the box using all the classes created.
	
	class Box {
	double width;
	double height;
	double length;
	Box(double w, double h, double l) {
	width = w;
	height = h;
	length = l;
	}
	double volume() {
	return width * height * depth;
	}
	}
	class Boxcolor extends Box {
	String color; // color of box
	// construct clone of an object
	Boxcolor(Boxcolor ob) { // pass object to constructor
	super(ob);
	color = ob.color;
	}
	// constructor when all parameters are specified
	Boxcolor(double w, double h, double l, String m) {
	super(w, h, l); // call superclass constructor
	color = m;
	}
	// default constructor
		10	CO1,CO3	L3

```
BoxWeight() {
super();
color = “red”;
}
}
class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
BoxWeight(BoxWeight ob) { // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor when all parameters are specified
BoxWeight(double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
BoxWeight() {
super();
weight = -1;
}
}

class DemoSuper {
public static void main(String args[]) {
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox2 = new Boxcolor(2, 3, 4, “green”);
double vol;
vol=mybox1.vol();
System.out.println(“volume is :”+vol);
System.out.println(“colour  is :”+mybox2.color);
System.out.println(“weight is :”+mybox1.weight);
}
}
```