Internal Assessment Test 2 January 2024								
Sub:	**Object Oriented Programming with Java**			Sub Code:	BCS306A	Branch:	AIML	
Date :	15 -10 -19	Duration:90 m	Max Marks:	50	Sem /Sec:	III A/B/C		OBE

Answer any FIVE FULL Questions		Marks	CO	RBT
1 (a)	**What is a constructor? Name and explain the different types of constructor with example program.**	10	CO2	L2

- A constructor initializes an object immediately upon creation.
- It has the same name as the class in which it resides and is syntactically similar to a method.
- Once defined, the constructor is automatically called when the object is created, before the new operator completes.
- Constructors no return type, not even void. This is because the implicit return type of a class' constructor is the class type itself.
- It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

Types of Constructors
1.*Default Constructor*
2.*Parameterised constructor*

Default Constructor : When the constructor is not define explicitly for a class, then Java creates a default constructor for the class.

- The default constructor automatically initializes all instance variables to their default values, which are zero, null, and false, for numeric types, reference types, and boolean, respectively.

.*Parameterised constructor:* Constructor with arguments(or you can say parameters) is known as Parameterized constructor

Example
```
class Box {
double width;
double height;
double depth;

Box() { // This is Default constructor for Box.

System.out.println("Constructing Box");
width = 10;
height = 10;
depth = 10;
}
Box(double w, double h, double d) { // This is parameter constructor for Box.
width = w;
height = h;
depth = d;
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
class BoxDemo7 {
public static void main(String args[]) {
// declare, allocate, and initialize Box objects
Box mybox1 = new Box()
Box mybox2 = new Box(10, 20, 15);

double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Output

```
Constructing Box
Volume is 1000.0
Volume is 3000.0
```

2.	**List and explain the uses of the keyword 'super' with Java programs.**	10	CO3	L3

- Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
- super has two general forms.
  - ➢ 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.

Using super to Call Superclass Constructors

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

```
super(arg-list);
```

- Here, 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

```
class Box {
private double width;
private double height;
private double depth;
// construct clone of an object
Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.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
}
// compute and return volume
double volume() {
return width * height * depth;
}
}
// BoxWeight now fully implements all constructors.
class BoxWeight extends Box {
double weight; // weight of box
// 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 mybox3 = new BoxWeight(); // default
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println();
vol = mybox3.volume();
System.out.println("Volume of mybox3 is " + vol);
System.out.println("Weight of mybox3 is " + mybox3.weight);
System.out.println();
}
}
```

Second use of super to access a member of the superclass:

- The second form of super always refers to the superclass of the subclass in which it is used.

The general form is
super.member

- Here, member can be either a method or an instance variable. The second form of super is most

applicable to situations in which member names of a subclass hide members by the same name in
the superclass.
Example:

```
/ Using super to overcome name hiding.
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
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();
}
}
```

This program displays the following:

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

3	**Distinguish between method overloading and method overriding. Write Java programs to demonstrate the use of method overloading and method overriding.**	10	CO2	L2

Having more than one method with a same name is called as **method overloading**. To implement this concept, the constraints are:

- The number of arguments should be different, and/or
- Type of the arguments must be different.

```
class Overload
{
void test() //method without any arguments
{
System.out.println("No parameters");
}
void test(int a) //method with one integer argument
{
System.out.println("Integer a: " + a);
}
void test(int a, int b) //two arguments
{
System.out.println("With two arguments : " + a + " " + b);
}
void test(double a) //one argument of double type
{
System.out.println("double a: " + a);
}
}
class OverloadDemo
{
public static void main(String args[])
{
Overload ob = new Overload();
ob.test();
ob.test(10);
ob.test(10, 20);
ob.test(123.25);
}
}
```

- In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to ***override*** the method in the super class.
- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the super class will be hidden.

```
class A
{
int i, j;
A(int a, int b)
{
i = a;
j = b;
}
void show() //suppressed
{
System.out.println("i and j: " + i + " " + j);
}
}
class B extends A
{
int k;
B(int a, int b, int c)
{
super(a, b);
k = c;
}
void show() //Overridden method
{
System.out.println("k: " + k);
}
}
class Override
{
public static void main(String args[])
{
B subOb = new B(1, 2, 3);
subOb.show();
}
}
```

4a.	**What is inheritance? Explain inheritance with the help of a Java program.**	6	CO3	L2

- Inheritance is one of the building blocks of object oriented programming languages. It allows creation of classes with hierarchical relationship among them.
- Using inheritance, one can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it.
- A class that is inherited is called a ***superclass***. The class that does the inheriting is called a ***subclass***.
- The general form of a **class** declaration that inherits a superclass:
  class *subclass-name* **extends** *superclass-name* {
  // body of class
  }

Example

```
class A
{
int i, j;
void showij()
{
System.out.println("i and j: " + i + " " + j);
}
}
class B extends A
{
int k;
void showk()
{
System.out.println("k: " + k);
}
void sum()
{
System.out.println("i+j+k: " + (i+j+k));
}
}

class SimpleInheritance
{
public static void main(String args[])
{
A superOb = new A();
B subOb = new B();
superOb.i = 10;
superOb.j = 20;
System.out.println("Contents of superOb: ");
superOb.showij();
subOb.i = 7;
subOb.j = 8;
subOb.k = 9;
System.out.println("Contents of subOb: ");
subOb.showij();
```

	subOb.showk();
	System.out.println("Sum of i, j and k in subOb:");
	subOb.sum();
	}
	}			

4. b	**What is an abstract class? Explain abstract class with the help of a Java program**
	
	A class containing at least one abstract method is called as ***abstract class.*** Abstract classes cannot be instantiated, that is one cannot create an object of abstract class. Whereas, a reference can be created for an abstract class.
	• To declare an abstract method, use this general form:
	abstract *type name(parameter-list);*
	No method body is present.
	
	abstract class A
	{
	abstract void callme();
	void callmetoo()
	{
	System.out.println("This is a concrete method.");
	}
	}
	
	class B extends A
	{
	void callme() //**overriding abstract method**
	{
	System.out.println("B's implementation of callme.");
	}
	}
	class AbstractDemo
	{
	public static void main(String args[])
	{
	B = new B(); //subclass object
	b.callme(); //calling abstract method
	b.callmetoo(); //calling concrete method
	}
	}
		4	CO3	L4

5. a	**With an example explain finalize() method in java**
	• Java provides a mechanism called finalization to handle situations where specific actions are required before an object is reclaimed by the garbage collector.
	
	• Objects may hold non-Java resources like file handles or character fonts.
	• It is essential to free these resources before an object is destroyed.
	• To add a finalizer to a class, define the `finalize()` method.
	• The Java runtime automatically calls this method when it is about to recycle an object of that class.
	• • Inside the `finalize()` method, specify actions that must be performed before an object is destroyed.
	• This method is called by the garbage collector just before reclaiming the object
	
	• The garbage collector runs periodically to identify and reclaim objects that are no longer referenced.
	• Objects without any live references are candidates for garbage collection.
	• Just before an object is freed, the Java runtime invokes the `finalize()` method on that object.
	• This provides an opportunity to release resources or perform other necessary cleanup tasks.
	• The garbage collector identifies and marks objects that are eligible for finalization.
	• Finalization ensures proper resource management and cleanup before the object is deallocated.
	
	The **finalize( ) method has this general form:**
	
	protected void finalize( ) {
	
	// finalization code here
	
	}
		6	CO2	L2

5.b	**Write a note on use of 'this' keyword** • Sometimes a method will need to refer to the object that invoked it. • **this** can be used inside any method to refer to the *current object. That is,* **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class ' type is permitted. **Example** **// A redundant use of this** Box(double w, double h, double d) {	4	CO2	L2

```
        this.width = w;

        this.height = h;

        this.depth = d;

        }
```

6 **Design a Java class called Stack with the following instance variables**
(i) private int stck[] (ii) private int tos
and methods
(i) void push(int)
(ii) int pop()
Write a Java program to create Stack object with stack size 5. Call the method push() to push 5 elements on to stack and display the output of the pop() operation.

```
// This class defines an integer stack that can hold 5values

class Stack
{
   private int stck[];
   private int tos;
   // allocate and initialize stack
   Stack(int size)
   {
        stck = new int[size];
        tos = -1;
   }

   // Push an item onto the stack
   void push(int item)
   {
        if(tos==stck.length-1) // use length member
          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--];
   }
}
```

CO3 L3

```
class TestStack2
{
   public static void main(String args[])
   {
        Stack mystack = new Stack(5);

        // push some numbers onto the stack
        for(int i=0; i<5; i++)
          mystack.push(i);

        // pop those numbers off the stack
        System.out.println("Stack in mystack:");
        for(int i=0; i<5; i++)
          System.out.println(mystack.pop());

   }
}
```

Output:

```
Stack in mystack:
4
3
2
1
0
```