Internal Assessment Test 2 October 2019									
Sub:	**Programming in Java**				Sub Code:	15CS561	Branch:	TCE/ECE	
Date :	15 -10 -19	Duration:90 m	Max Marks:	50	Sem /Sec:	VI B/C		OBE	

	Answer any FIVE FULL Questions	Marks	CO	RBT

1 (a)	**What is constructor? Name and explain different types of constructor with example program.**
	
	Constructor is a special type of member method which is invoked automatically when the object gets created. Constructors are used for object initialization. They have same name as that of the class. Since they are called automatically, there is no return type for them. Constructors may or may not take parameters.
	
	• Every class is provided with a ***default constructor*** which initializes all the data members to respective *default values*. (Default for numeric types is zero, for character and strings it is null and default value for Boolean type is false.)
	• In the statement *classname ob= new classname();* the term *classname()* is actually a constructor call.
	• If the programmer does not provide any constructor of his own, then the above statement will call default constructor.
	• If the programmer defines any constructor, then default constructor of Java cannot be used.
	• So, if the programmer defines any parameterized constructor and later would like to create an object without explicit initialization, he has to provide the default constructor by his own. For example, the above program, if we remove ordinary constructor, the statements like Box b1=new Box(); will generate error. To avoid the error, we should write a default constructor like – Box(){ } Now, all the data members will be set to their respective default values.
	
	class Box
	{
	double w, h, d;
	double volume()
	{
	return w*h*d;
	}
	Box() **//ordinary constructor**
		10	CO3	L1, L4

	{
	w=h=d=5;
	}
	Box(double wd, double ht, double dp) **//parameterized constructor**
	{
	w=wd;
	h=ht;
	d=dp;
	}
	}			

2. (a)	**Write short notes on: (i) this keyword (ii) static**
	
	(i)Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the ***this*** keyword. ***This*** can be used inside any method to refer to the ***current object***. That is, ***this*** is always a reference to the object which invokes the method call.
	
	(ii) When a member is declared **static**, it can be accessed before any objects of its class are created, and without reference to any object. Instance variables declared as **static** are 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.
	
	class StaticDemo
	{
	static int a = 42;
	static int b = 99;
	static void callme()
	{
	System.out.println("Inside static method, a = " + a);
	}
	}
	
	class StaticByName
	{
	public static void main(String args[])
	{
	StaticDemo.callme();
	System.out.println("Inside main, b = " + StaticDemo.b);
	}
	}
	
	Output:
	Inside static method, a = 42
	Inside main, b = 99
		6	CO3	L2

2.b	**Write a Java program to display the factorial of a number using recursion.**
	
	class Factorial
	{
	int fact(int n)
	{
	if (n==0)
	return 1;
	return n*fact(n-1);
	}
	}
	class FactDemo
	{
	public static void main(String args[])
	{
	Factorial f= new Factorial();
	System.out.println("Factorial 3 is "+ f.fact(3));
	System.out.println("Factorial 8 is "+ f.fact(8));
	}
	}
		4	CO3	L3

3.	**List and explain the uses of the keyword final with Java programs.**
	The keyword *final* can be used in three situations in Java:
	• To create the equivalent of a named constant.
	• To prevent method overriding
	• To prevent Inheritance
	
	To create the equivalent of a named constant: A variable can be declared as final. Doing so prevents
	its contents from being modified. This means that you must initialize a final variable when it is declared.
	For example:
	final int FILE_NEW = 1;
	
	To prevent method overriding: Sometimes, we do not want a super class method to be overridden in the subclass. Instead, the same super class method definition has to be used by every subclass. In such situation, we can prefix a method with the keyword *final* as shown below –
	
	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!");
		10	CO3	L1, L3

	}
	}
	
	To prevent Inheritance: As we have discussed earlier, the subclass is treated as a specialized class and super class is most generalized class. During multi-level inheritance, the bottom most class will be with all the features of real-time and hence it should not be inherited further. In such situations, we can prevent a particular class from inheriting further, using the keyword *final*. For example –
	
	final class A
	{
	// ...
	}
	class B extends A // ERROR! Can't subclass A
	{
	// ...
	}			

4.	**List and explain the uses of the keyword super with Java programs.**
	In Java, the keyword *super* can be used in following situations:
	• To invoke super class constructor within the subclass constructor
	• To access super class member (variable or method) when there is a duplicate member name in
	the subclass
	
	To invoke super class constructor within the subclass constructor: Sometimes, we may need to initialize the members of super class while creating subclass object. Writing such a code in subclass constructor may lead to redundancy in code. For example,
	
	class Box
	{
	double w, h, b;
	Box(double wd, double ht, double br)
	{
	w=wd; h=ht; b=br;
	}
	}
	class ColourBox extends Box
	{
	int colour;
	ColourBox(double wd, double ht, double br, int c)
	{
	w=wd; h=ht; b=br; **//code redundancy**
	colour=c;
	}
	}
	
	To access super class member variable when there is a duplicate variable name
		[10]	CO3	L1, L3

	in the subclass: This form of super is most applicable to situations in which member names of a subclass hide members by the same name in the super class.
	
	class A
	{
	int a;
	}
	class B extends A
	{
	int a; //duplicate variable *a*
	B(int x, int y)
	{
	super.a=x; //accessing superclass *a*
	a=y; //accessing own member *a*
	}
	void disp()
	{
	System.out.println("super class a: "+ super.a);
	System.out.println("sub class a: "+ a);
	}
	}
	class SuperDemo
	{
	public static void main(String args[])
	{
	B ob=new B(2,3);
	ob.disp();
	}
	}			

5.	**Distinguish between method overloading and method overriding. Write Java programs to demonstrate the use of method overloading and method overriding.**
	
	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);
		[10]	CO3	L2

```
}
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();
	}
	}			

6.	**Create a Java class called Student with the following instance variables (USN, Name, Branch, Phone Number). Write a Java program to create 2 Student objects and print USN, Name, Branch and phone number with suitable message.**
	
	import java.io.*;
	class Student
	{
	String usn, name, branch;
	long ph;
	
	Student()
	{
	usn = name = branch = "no value";
	ph = 0;
	}
	
	void read_data(String u, String n, String b, long p)
	{
	usn = u;
	name = n;
	branch = b;
	ph =p;
	}
	void display()
	{
	System.out.println(usn + "\t" + name + "\t" + branch + "\t\t" + ph);
	}
	}
	class Lab1A
	{
	public static void main(String args[]) throws Exception
	{
	String u, n, b;
	long p;
	int no;
	BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
	System.out.println("Enter number of records");
	no = Integer.parseInt(br.readLine());
	
	Student[] s = new Student[no];
	
	for(int i=0; i<s.length;i++)
	{
		[10]	CO3	L4

```
System.out.println("Enter " + (i + 1) + " Student record");
s[i] = new Student();
System.out.println("Enter student USN");
u = br.readLine();
System.out.println("Enter student Name");
n = br.readLine();
System.out.println("Enter student Branch");
b = br.readLine();
System.out.println("Enter student Phone number");
p = Long.parseLong(br.readLine());
s[i].read_data(u, n, b, p);
}

System.out.println("USN \t\t NAME \t BRANCH \t PHONE NO");
for(int i=0; i<s.length;i++)
{
s[i].display();
}
}
}
```

7. (a) **What is inheritance? Explain inheritance with the help of a Java program.** [06] 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. In the terminology of Java, a class that is inherited is called a ***superclass***. The class that does the inheriting is called a ***subclass***.

```
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();
	}
	}			

(b)	**What is 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.
	
	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 b = new B(); //subclass object
	b.callme(); //calling abstract method
	b.callmetoo(); //calling concrete method
	}
	}
		[04]	CO3	L4

8.	**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 1 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.
	
	class Stack
	{
	int st[] = new int[5];
	int top;
	Stack()
	{
	top = -1;
	}
	void push(int item)
	{
	if(top==4)
	System.out.println("Stack is full.");
	else
	st[++top] = item;
	}
	int pop()
	{
	if(top==-1)
	{
	System.out.println("Stack underflow.");
	return 0;
	}
	else
	return st[top--];
	}
	}
	
	class StackDemo
	{
	public static void main(String args[])
	{
	Stack mystack1 = new Stack();
	Stack mystack2 = new Stack();
	for(int i=0; i<5; i++)
	mystack1.push(i);
	for(int i=5; i<10; i++)
	mystack2.push(i);
	System.out.println("Contents of mystack1:");
	for(int i=0; i<5; i++)
	System.out.println(mystack1.pop());
	System.out.println("Contents of mystack2:");
	for(int i=0; i<5; i++)
		[10]	CO3	L3

	System.out.println(mystack2.pop()); } }			