INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test – II

Sub:	OBJECT ORIENTED CONCEPTS							Code:	15CS45
Date:	10/ 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	CSE (C,D)

Answer Any FIVE FULL Questions

		Marks	OBE CO	OBE RBT
1 (a)	Define Synchronization? Explain how Synchronization is achieved in JAVA?	[10]	CO1	L3
2 (a)	What you mean by Thread? Explain how thread can be created in JAVA?	[10]	CO1	L2
			CO2	L2
3 (a)	Write a Java program to create two threads one displays “computer science” and other thread displays “Information science” for five times.	[10]	CO2	L3
4 (a)	What is an Exception Handling? Explain Exception Handling Mechanism in JAVA?	[10]	CO1	L3
5(a)	What is super? Explain the Significance of super (both constructor and method) with example?	[10]	CO2	L2
6(a)	Differentiate between Method Overriding and Method Overloading in JAVA with an Example?	[05]	CO2	L2
(b)	Explain with syntax the use of is Alive() and join()	[05]	CO2	L2
7 (a)	Define Inheritance? Write a JAVA program to illustrate Multi-level Inheritance?	[10]	CO2	L3

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Explain the object oriented concepts	2	1	2	0	1	0	0	0	0	0	1	0
CO2:	Explain Classes, objects ,constructor	2	1	2	0	1	0	0	0	0	0	1	1
CO3:	Implement inheritance, interfaces and exception in java	2	1	2	0	1	0	0	0	0	0	0	0
CO4:	Implement Multi threaded programming and event handling in java	2	1	2	0	1	0	0	0	0	0	0	0
CO5:	Develop computer programs to solve real world problems in java	2	1	2	0	1	0	0	0	0	0	0	0
CO6:	Develop GUI interfaces using Applet and swings	2	1	2	0	1	0	0	0	0		0	0

1. (a) Define Inheritance? Write a JAVA program to illustrate Multi-level Inheritance?

- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
- **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

```
class Car{
     public Car()
     {
              System.out.println("Class Car");
     }
     public void vehicleType()
     {
              System.out.println("Vehicle Type: Car");
     }
}
class Maruti extends Car{
     public Maruti()
     {
              System.out.println("Class Maruti");
     }
     public void brand()
     {
              System.out.println("Brand: Maruti");
     }
     public void speed()
     {
              System.out.println("Max: 90Kmph");
     }
}
public class Maruti800 extends Maruti{

      public Maruti800()
      {
               System.out.println("Maruti Model: 800");
      }
      public void speed()
              {
                      System.out.println("Max: 80Kmph");
              }
      public static void main(String args[])
      {
               Maruti800 obj=new Maruti800();
               obj.vehicleType();
               obj.brand();
               obj.speed();
      }
}
```

2. Define thread? Explain how threads are created in java?

There are two ways to create a thread:

By extending Thread class

By implementing Runnable interface

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

A new thread starts(with new callstack).

The thread moves from New state to the Runnable state.

When the thread gets a chance to execute, its target run() method will run.

Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
```

```
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:thread is running...

Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

3. Define Synchronization? Explain how synchronization is handled in java?

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

Mutual Exclusive

Synchronized method.

Synchronized block.

static synchronization.

Cooperation (Inter-thread communication in java)

```
Class Table{

void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
```

```
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

4. Define Exception? Explain how exceptions are handling in java

An exception (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.

- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Syntax:

```
try {
   // Protected code
}catch(ExceptionType1 e1) {
   // Catch block
}catch(ExceptionType2 e2) {
   // Catch block
}catch(ExceptionType3 e3) {
   // Catch block
}
```

The Throws/Throw Keywords

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.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

Try to understand the difference between throws and throw keywords, *throws* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

The following method declares that it throws a RemoteException −

Example

```
import java.io.*;
public class className {


  public void deposit(double amount) throws RemoteException {
    // Method implementation
    throw new RemoteException();
  }
  // Remainder of class definition
}
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException −

Example

```
import java.io.*;
public class className {


  public void withdraw(double amount) throws RemoteException,
    InsufficientFundsException {
    // Method implementation
  }
  // Remainder of class definition
}
```

The Finally Block

The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.

Using a finally block allows you to 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 the catch blocks and has the following syntax −

Syntax

```
try {
  // Protected code
}catch(ExceptionType1 e1) {
  // Catch block
}catch(ExceptionType2 e2) {
  // Catch block
}catch(ExceptionType3 e3) {
  // Catch block
}finally {
  // The finally block always executes.
}
```

6. a) Difference between method Overloading and Overriding in java

1. Overloading happens at **compile-time** while Overriding happens at **runtime**: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.

2. Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
3. The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
4. **Static binding** is being used for overloaded methods and **dynamic binding** is being used for overridden/overriding methods.
5. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
6. private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
7. Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the overriding method can have more specific return type.
8. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

5. Define super()?explain the significance of super with a example?

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.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) **super is used to refer immediate parent class instance variable.**

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
```

```
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```