CMR
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

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1 (a)	Define Synchronization? Explain with example 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, with example.	[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 isAlive( ) and join( ) methods.	[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

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

CMR
INSTITUTE OF
TECHNOLOGY

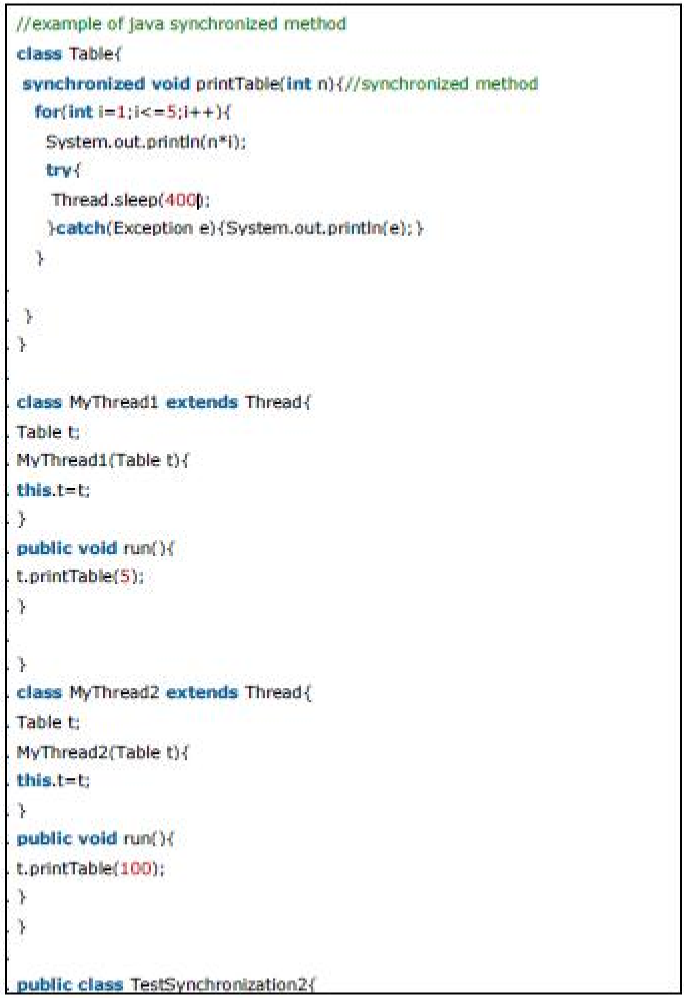
USN										

Internal Assessment Test – II

SCHEME OF EVALUATION AND SOLUTION

Sub:	OBJECT ORIENTED CONCEPTS							Code:	15CS45	Marks Distribution
Date:	10/ 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	CSE (All Sec)	
Answer Any FIVE FULL Questions										

		Marks	OBE		
			CO	RBT	
1	**Define Synchronization? Explain with example how Synchronization is achieved in JAVA?** When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called *synchronization.* Java provides unique, language-level support for it. Key to synchronization is the concept of the monitor (also called a *semaphore*). A *monitor* is an object that is used as a mutually exclusive lock, or *mutex.* Only one thread can *own* a monitor at a given time. When a thread acquires a lock, it is said to have *entered* the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread *exits* the monitor These other threads are said to be *waiting* for the monitor If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task **PROGRAM**.	[10]	CO1	L3	Explanation: 5m Program: 5m



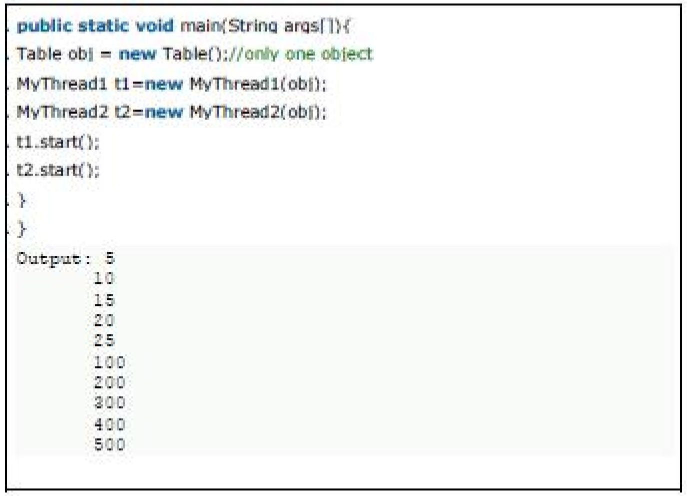
```java
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

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);
}
}

public class TestSynchronization2{
```



```
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();
}
}
Output: 5
       10
       15
       20
       25
       100
       200
       300
       400
       500
```

2	**What you mean by Thread? Explain how thread can be created in JAVA?** Unlike many other computer languages, Java provides built-in support for multithreaded programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.	[10]	CO1	L2	Explanation: 2m Thread Class : 4m Runnable : 4m

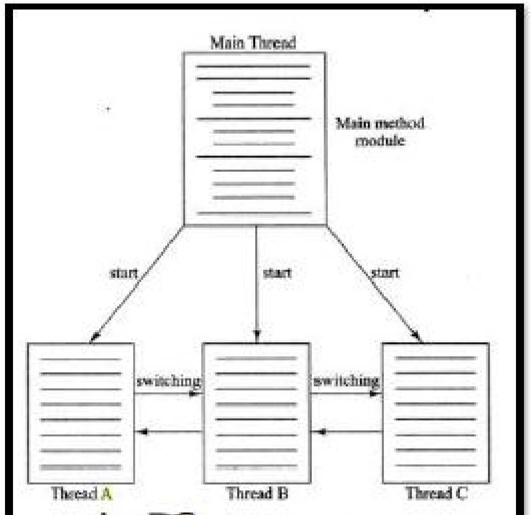


Fig. A Multithreaded program

Java defines two ways in which this can be accomplished:

The two ways to create a thread:
1. By extending **Thread** class
2. By implementing **Runnable** interface.

```
class A extends Thread {
public void run() {
for (int j = 1; j <= 5; j++) {
System.out.println("\t From ThreadA : j = " + j);
}
System.out.println("Exit from A ");
}
}

public class Threads {
public static void main(String[] args) {
new A().start();
}
}
```

```
Output:
run:
Exit from A
From ThreadA : j = 1
From ThreadA : j = 2
From ThreadA : j = 3
From ThreadA : j = 4
From ThreadA : j = 5
Exit from A
```

Runnable interface:

The Runnable interface should be implemented by any class whose instances are int

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 followi

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	**Write a Java program to create two threads one displays "computer science" and other thread displays "Information science" for five times.**	[10]	CO2	L2	Program : 10m

```
//Java Program to illustrate Creating threads using thread class

class A extends Thread {
public void run() {
for (int i = 1; i <= 5; i++) {
System.out.println("\t  Computer Science");
}
System.out.println("Exit from A");
}
}
```

	class B extends Thread {
	public void run() {
	for (int j = 1; j <= 5; j++) {
	System.out.println("\t Information Science");
	}
	System.out.println("Exit from B ");
	}
	}
	
	public class Threads {
	public static void main(String[] args) {
	new A().start();
	new B().start();
	}
	}				

4	**What is an Exception Handling? Explain Exception Handling Mechanism in JAVA, with example.** 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.	[10]	CO1	L3	Explanation:5 Program : 5

- 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.
}
```

5 **What is super? Explain the Significance of super (both constructor and method) with example?**

Explanation: 2m
Constuctor : 4m
variable : 4m

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.

```java
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();
}}
```

6(a)	**Differentiate between Method Overriding and Method Overloading in JAVA with an Example?**	[05]	CO2	L2	Explanation:5 Program : 5

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.				
(b)	**Explain with syntax, the use of isAlive( ) and join( ) methods**	[05]	CO2	L2	Explanation:5 Program : 5

Sometimes one thread needs to know when another thread is ending. In java, **isAlive()** and **join()** are two different methods to check whether a thread has finished its execution.

The **isAlive()** method returns **true** if the thread upon which it is called is still running otherwise it returns **false**.

```
final boolean isAlive()
```

But, **join()** method is used more commonly than **isAlive()**. This method waits until the thread on which it is called terminates.

```
final void join() throws InterruptedException
```

Using **join()** method, thread will wait until the specified thread completes its execution. There are overloaded versions of **join()** method, which allows us to specify time for which you want to wait for the specified thread to terminate.

```
final void join(long milliseconds) throws InterruptedException
```

Example of `isAlive` method

```
public class MyThread extends Thread
{
	public void run()
	{
		System.out.println("r1 ");
		try {
			Thread.sleep(500);
		}
		catch(InterruptedException ie) { }
		System.out.println("r2 ");
```

```
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();
                t2.start();
                System.out.println(t1.isAlive());
                System.out.println(t2.isAlive());
        }
}
```

Output :

```
r1
true
true
r1
r2
r2
```

Example of thread without `join()` method

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                        Thread.sleep(500);
                }
                catch(InterruptedException ie){ }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();
                t2.start();
        }
}
```

Output :

```
r1
r1
r2
r2
```

In this above program two thread t1 and t2 are created. t1 starts first and after printing "r1" on console thread t1 goes to sleep for 500 ms. At the same time Thread t2 will start its process and print "r1" on console and then go into sleep for 500 ms. Thread t1 will wake up from sleep and print "r2" on console similarly thread t2 will wake up from sleep and print "r2" on console. So you will get output like `r1 r1 r2 r2`

Example of thread with `join()` method

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                        Thread.sleep(500);
                }catch(InterruptedException ie){ }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();

                try{
                        t1.join();      //Waiting for t1 to finish
                }catch(InterruptedException ie){}

                t2.start();
        }
}
```

	Output :
	
	`r1`
	`r2`
	`r1`
	`r2`
	
	In this above program join() method on thread t1 ensures that t1 finishes it process before thread t2 starts.				

	Specifying time with join() If in the above program, we specify time while using **join()** with **t1**, then **t1** will execute for that time, and then **t2** will join it. **`t1.join(1500);`** Doing so, initially t1 will execute for 1.5 seconds, after which t2 will join it.				

7	**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");`
		[10]	CO2	L2	Definition :2m
					Program and Explantion: 8m

```
     }
     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();
      }
}
```