II INTERNAL QUESTION PAPER

CMR INSTITUTE OF TECHNOLOGY

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

Internal Assesment Test - II

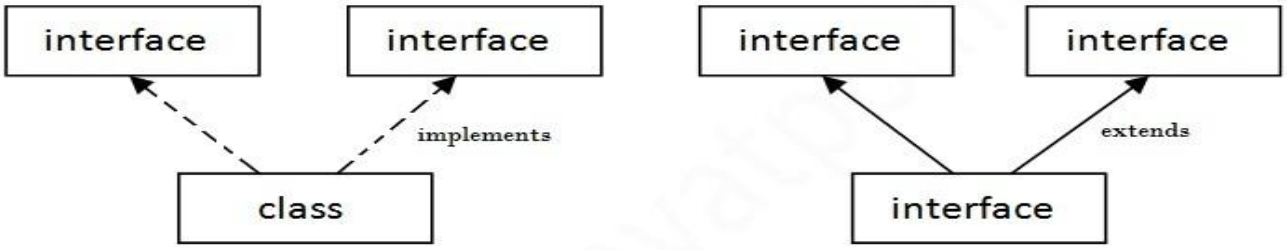
Sub:	OBJECT ORIENTED CONCEPTS							Code:	15CS45
Date:	10/05/2017 (8.30 TO 10 AM)	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	ISE

	Answer Any FIVE FULL Questions	Marks	OBE CO	OBE RBT
1(a)	Which is the alternative approach to implement multiple inheritances in Java? Explain with an example.	[10]	CO3	L5
2(a)	Explain the significance of super with example.	[07]	CO3	L4
2(b)	What is a default package and default class in JAVA?	[03]	CO3	L1
3(a)	Explain with syntax, the use of isAlive( ) and join( ) methods	[05]	CO4	L4
3(b)	What is meant by thread priority? How to assign and get the priority, Give example.	[05]		
4(a)	What is multithreading? What is the advantage of multithreaded programs? How to create a thread using the Runnable interface?	[10]	CO4	L3
5(a)	Define exception. Why is exception handling done? Demonstrate working of nested try block, with example.	[10]	CO3	L3
6(a)	Explain throw, throws and finally with respect to exceptions. Write a program which contains a method which will throw an IllegalAccessException and use proper exception handlers so that exception should be printed.	[10]	CO4	L5
7(a)	Create a Java class called Student with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone. Write a Java program to create nStudent objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.	[10]	CO3	L5

II INTERNAL SOLUTION

1a. Which is the alternative approach to implement multiple inheritances in Java? Explain with an example. [10]

Multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

The below program shows a class implementing more than one interface thus supporting multiple inheritance.

```
interface Printable
{
        void print();
}
interface Showable
{
        void show();
}
class A7 implements Printable,Showable
{
        public void print()
        {
            System.out.println("Hello");
        }
        public void show()
        {
            System.out.println("Welcome");
        }
       public static void main(String args[])
        {
            A7 obj = new A7();
            obj.print();
            obj.show();
         }
}
```

In the below program an interface Showable extends another interface Printable. When the class TestInterface implements Showable it has to implement all the functions present in both the interfaces. The class does multiple inheritance.

```
interface Printable
{
	void print();
}
interface Showable extends Printable
{
	void show();
}
class TestInterface4 implements Showable
{
	public void print()
	{
		System.out.println("Hello");
	}
	public void show()
	{
		System.out.println("Welcome");
	}
	public static void main(String args[])
	{
		TestInterface4 obj = new TestInterface4();
		obj.print();
		obj.show();
	 }
}
```

2a. Explain the significance of super with example. [07]

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 is mentioned below

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.

```
class Person
{
   Person()
   {
      System.out.println("Person class Constructor");
   }
   void message()
   {
      System.out.println("This is person class");
   }
}

/* Subclass Student */
```

```
class Student extends Person
{
    Student()
    {
        // invoke or call parent class constructor
        super();

        System.out.println("Student class Constructor");
    }
    void message()
    {
        System.out.println("This is student class");
    }

    // Note that display() is only in Student class
    void display()
    {
        // will invoke or call current class message() method
        message();

        // will invoke or call parent class message() method
        super.message();
    }
}

/* Driver program to test */
class Test
{
    public static void main(String args[])
    {
        Student s = new Student();

        // calling display() of Student
        s.display();
    }
}
```

2b. What is a default package and default class in JAVA? [03]

The default package is an unnamed package. The unnamed package contains java classes whose source files did not contain a package declaration. The purpose of default package is for convenience when developing small or temporary applications or when just beginning development. The compiled class files will be in the current working directory. Note that an unnamed package cannot have subpackages, since the syntax of a package declaration always includes a reference to a named top level package.

If a class has no modifier (the default, also known as package-private), it is visible only within its own package, this is default class.

3a. Explain with syntax, the use of isAlive( ) and join( ) methods. **[05]**

The java.lang.Thread.isAlive() method tests if this thread is alive. A thread is alive if it has been started and has not yet died. This method returns true if this thread is alive, false otherwise.
Following is the declaration of the method

public final boolean isAlive()

The java.lang.Thread.join() method waits for a thread to die. It causes the currently thread to stop executing until the thread it joins with completes its task. Following is the declaration of the method

public void join() throws InterruptedExceptionpublic
void join(long milliseconds)throws InterruptedException

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Status:" + isAlive());
    }
}
class alivetest
{
    public static void main(String args[])
    {
        A a = new A();
        a.start();
        try
        {
            a.join();
        }
        catch(InterruptedException e)
        {}
        System.out.println("Status:" + a.isAlive());
    }
}
```

At this point Thread A is alive so the value gets printed by **isAlive()** method is "***true***"

join() method is called from Thread A which stops executing of further statement until A is Dead

Now isAlive() method returns the value false as the Thread A is complete

Output

```
C:\NIEC Java>java alivetest
Status:true
Status:false
```

3b. What is meant by thread priority? How to assign and get the priority? **[05]**

Java assigns to each thread a priority that determines how that thread should be treated with respect to the others. Thread priorities are integers that specify the relative priority of one thread to another. As an absolute value, a priority is meaningless; a higher-priority thread doesn't run any faster than a lower-priority thread if it is the only thread running. Instead, a thread's priority is used to decide when to switch from one running thread to the next. This is called a context switch. The rules that determine when a context switch takes place are simple:

• A thread can voluntarily relinquish control. This is done by explicitly yielding, sleeping, or blocking on pending I/O. In this scenario, all other threads are examined, and the highest-priority thread that is ready to run is given the CPU.

• A thread can be preempted by a higher-priority thread. In this case, a lower-priority thread that does not yield the processor is simply preempted—no matter what it is doing— by a higher-priority thread. Basically, as soon as a higher-priority thread wants to run, it does. This is called preemptive multitasking.

In cases where two threads with the same priority are competing for CPU cycles, the situation is a bit complicated. For operating systems such as Windows, threads of equal priority are time-sliced automatically in round-robin fashion. For other types of operating systems, threads of equal priority must voluntarily yield control to their peers. If they don't, the other threads will not run.

To set a thread's priority, use the setPriority( ) method, which is a member of Thread. This is its general form:

final void setPriority(int level)

Here, level specifies the new priority setting for the calling thread. The value of level must be within the range MIN_PRIORITY and MAX_PRIORITY. Currently, these values are 1 and 10, respectively. To return a thread to default priority, specify NORM_PRIORITY, which is currently 5. These priorities are defined as static final variables within Thread. You can obtain the current priority setting by calling the getPriority( ) method of Thread,

shown here:

final int getPriority( )

```
// Demonstrate thread priorities.
class clicker implements Runnable
{       long click = 0;
        Thread t;
        private volatile boolean running = true;
        public clicker(int p)
        {
                t = new Thread(this);
                t.setPriority(p);
        }
        public void run()
        {       while (running)
                {       click++;
                }
        }
        public void stop()
        {       running = false;
        }
        public void start()
        {       t.start();
        }
}
class HiLoPri
{       public static void main(String args[])
        {       Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
                clicker hi = new clicker(Thread.NORM_PRIORITY + 2);
                clicker lo = new clicker(Thread.NORM_PRIORITY - 2);
                lo.start();
                hi.start();
                try
                {       Thread.sleep(10000);
                }
                catch (InterruptedException e)
```

```
                {       System.out.println("Main thread interrupted.");
                }
                lo.stop();
                hi.stop();
                // Wait for child threads to terminate.
                try
                {       hi.t.join();
                        lo.t.join();
                }
                 catch (InterruptedException e)
                {       System.out.println("InterruptedException caught");
                }
                System.out.println("Low-priority thread: " + lo.click);
                System.out.println("High-priority thread: " + hi.click);
        }
}
```

The output of this program, shown as follows when run under Windows, indicates that the threads did context switch, even though neither voluntarily yielded the CPU nor blocked for I/O. The higher-priority thread got the majority of the CPU time.

Low-priority thread: 4408112
High-priority thread: 589626904

4a. What is multithreading? What is the advantage of multithreaded programs? How to create a thread using the Runnable interface? [10]

Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel. 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.

Advantages of Multithreading

1. Enables programmers to do multiple things at one time
2. Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the program execution
3. Improved performance and concurrency
4. Simultaneous access to multiple applications

The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called **run( )**, which is declared like this:

public void run( )

You will define the code that constitutes the new thread inside **run()** method. It is important to understand that **run()** can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its **start( )** method, which is declared within Thread. The start( ) method is shown here:

void start( );

Here is an example that creates a new thread and starts it running:

```
// Create a second thread.
class NewThread implements Runnable {
Thread t;
NewThread() {
```

```
// Create a new, second thread
t = new Thread(this, "Demo Thread");
System.out.println("Child thread: " + t);
t.start(); // Start the thread
}
// This is the entry point for the second thread.
public void run() {
try {
for(int i = 5; i > 0; i--) {
System.out.println("Child Thread: " + i);
Thread.sleep(500);
}
} catch (InterruptedException e) {
System.out.println("Child interrupted.");
}
System.out.println("Exiting child thread.");
}
}
class ThreadDemo {
public static void main(String args[]) {
new NewThread(); // create a new thread
try {
for(int i = 5; i > 0; i--) {
System.out.println("Main Thread: " + i);
Thread.sleep(1000);
}
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
System.out.println("Main thread exiting.");
}
}
```

The output produced by this program is as follows

```
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

5a. Define exception. Why is exception handling done? Demonstrate working of nested try block, with example. [10]

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.
Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.
Java catch block is used to handle the Exception. It must be used after the try block only. You can use multiple catch block with a single try.
The try block within a try block is known as nested try block in java. Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

```
class Excep6
{       public static void main(String args[])
        {       try
                {
                        try
                        {
                                System.out.println("going to divide");
                                int b =39/0;
                        }
                        catch(ArithmeticException e)
                        {
                                System.out.println(e);
                        }
                        try
                        {
                                int a[]=new int[5];
                                a[5]=4;
                        }
                        catch(ArrayIndexOutOfBoundsException e)
                        {
                                System.out.println(e);
                        }
                        System.out.println("other statement);
                }
                catch(Exception e)
                {
                        System.out.println("handeled");
                }
                System.out.println("normal flow..");
        }
}
```

In the above example we have a outer try catch block that can handle all the exceptions that can be handled by the Exception class. Inside the outer try block there is two try block, one try block with set of statements that might invoke an arithmetic exception and a catch block to handle the same. The other try block has set of statements that might invoke an array out of bounds exception and a catch block to handle the same. If any other exception occurs inside the outer try it will be handled by the outer most catch block.

6a. Explain throw, throws and finally with respect to exceptions. Write a program which contains a method which will throw an IllegalAccessException and use proper exception handlers so that exception should be printed. [10]

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. The syntax of java throw keyword is given below.

```
throw exception;
```

The Java throws keyword is used to declare an exception. It gives information to the programmer that an exception may occur so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

```
return_type method_name() throws exception_class_name
{
    //method code
}
```

Java finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block follows try or catch block. Java finally block is always executed whether an exception occurred or not. If an exception occurred then whether it is handled or not finally block will be executed.

The program below demonstrates the use of all the three keywords (throw, throws and finally) through the IllegalAccessException. In the throwOne() method an IllegalAccessException might occur, but it doesn't want to handle it, thus it intimates the users of this method by using the throws keyword. Thus the IllegalAccessException has to be handled at the time of call to throwOne() method. Inside the throwOne() an IllegalAccessException needs to be invoked explicitly, thus we make use of the throw keyword to do the same. In the main() when the call to throwOne() is done it uses the try catch block to handle the IllegalAccessException. The finally block is also used to execute a set of code at the end of the try catch block.

```
class ThrowsDemo
{   static void throwOne() throws IllegalAccessException
    {
        System.out.println("Inside throwOne.");
        throw new IllegalAccessException("demo");
    }
    public static void main(String [] args)
    {
        try
        {
            throwOne();
        }
        catch(IllegalAccessException e)
        {
            System.out.println("Caught:" +e);
        }
        finally
        {
            System.out.println("Inside the finally block");
        }
    }
}
```

Output:

```
Inside throwOne.
Caught: java.lang. IllegalAccessException: demo
Inside the finally block
```

7a. Create a Java class called Student with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone. Write a Java program to create nStudent objects and print the USN, Name, Branch, and Phone of these objects with suitable headings. [10]

```
import java.util.Scanner;
class Student
{
	String USN, Name, Branch;
	long Phone;
	void getdata()
	{
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the USN, Name, Branch and Phone No of the Student");
		USN = sc.next();
		Name = sc.next();
		Branch = sc.next();
		Phone = sc.nextLong();
	}
	void display()
	{
		System.out.println("USN : " + USN + "\nName : " + Name + "\nBranch : " + Branch +
"\nPhone : " + Phone);
		System.out.println();
	}
}
public class prgm1a
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.println(" Enter the number of students ");
		int n = sc.nextInt();
		Student [] nstudent = new Student[n];
		for(int i=0;i<n;i++)
		{
			nstudent[i] = new Student();
			nstudent[i].getdata();
		}
		System.out.println("The Details of all the students are shown below ");
		for(int i=0; i<n;i++)
			nstudent[i].display();
	}
}
```