Date : __/__/__
Page : ______

Countineous Assessment 2

Course Code : 615

Course Title : Programming in Java

Name : Jayshri Lal Pandit

Roll NO. : RD2112A03

Reg No. : 12111670

Section : D2112

Date : / /
Page :

(1.) "At a specific point of time, it becomes ~~ness~~ necessary to terminate a thread before the task has been completed". Justify your answere with the help of an example code.

Ans:- A Thread is automatically destroyed when the run() method has completed. But it might be required to kill/stop a thread before it has completed its life cycle. previously, methods suspend(), resume() and stop() were used to manage the execution of threads. But these methods were deprecated by Java 2 because they could result in system failures. Modern ways to suspend/stop a thread are by using a boolean flag and Thread.interrupt() method.

• **Using a boolean flag :-**
we can define a boolean variable which is used for stopping/killing threads say 'exit'. whenever we want to stop a thread, the 'exit' variable will be seht to true.

// Java program to illustrate

// Stopping a thread using boolean flag.

```
Class MyThread implements Runnable
{   // to stop the thread
    private boolean exit;

    private String name;
     Thread t;

    MyThread (String threadname)
    {

        name = threadname;
        t    = new Thread(This, name);
     System.out.println("New thread:"+t);
        exit = false;
        t.start(); // starting the thread.
    }.

   public void run()
   {   int i = 0
       while (!exit)
       {

           System.out.println(name + ":"+i);
           i++;
    try
    {
        Thread.sleep(100);
    }
    catch (InterruptedException e)
    {
        System.out.println("Caught :" +e);
    }
    System.out.println(name + "Stopped.");
    }
    // for stopping thread.
    public void stop ()
    {
        exit = true;
    }
}
// Main class
public class Main
{
    public static void main(String arg[])
    {
     MyThread t1 = new MyThread("First thread");
     MyThread t2 = new MyThread("Second thread");
```

```
try
{
   Thread.sleep(500);
     t1.stop();
     t2.Stop();
     Thread.sleep(500);
  }
catch (InterruptedException e)
{
  System.out.println("catch : "+e);
 }
  System.out.println("Exiting the main
                        Thread");
}
}
```

Output

```
New thread: Thread[First thread, 5, main]
New thread : Thread [Second thread, 5, main]
First thread: 0
Second thread : 0
First thread : 1
Second thread : 1
First thread : 2
Second thread : 2
```

```
|
|
|
First thread : 5
Second thread Stopped.
First thread stopped.
Exiting the main Thread.
```

Note! output may vary every time.

• Using Thread.interrupt () method :-
Whever an interrupt has been se
to a thread, it Should Stop what
task it is performing. It is very
likely that whenever the thread
receives an interrupt, it is to be
terminated. This action can be do
by using the intrrupt () method.

```
// Java program to illustrate
// stopping a thread
// using the intrupt () method

class MyThread implements Runnable
{
    Thread t;

    My.Thread ()
    {
        t = new Thread(this);
```

```
System.out.println("New thread:"+t);
t.start();

}

public void ~~run~~ ()
{
    while (! Thread.interrupted())
    {
        System.out.println("Thread is running");
    }
   System.out.println("Thread has stopped.");
}
}

// Main class
public class Main
{
    public static void main(String args[])
    {
        MyThread t1 = new MyThread();
        try
        {
            Thread.sleep(1);
~~t1~~.t.interrupt ();
 Thread.sleep(5);
}
catch (InterruptedException e)
{
   System.out.println ("Caught :" +e);
 }
~~System.out.println("Exiting the main Thread");~~
}

}
```

Output

```
New thread : Thread [Thread-0, 5, main]
Thread is running
Thread is running
     |
     |
     |
     |
     |
     |
     |
     |
Thread has stopped.
Exiting the main Thread
```

(2.) Which are the different ways to implement multithreading and also explain the role of the start, run and stop methods.

Ans:→ Multithreading is a Java feature that allows concurrent exception of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Therads can be created by using two mechanisms :-

(1.) Extending the Thread class.
(2.) Implementing the Runnable Interface

(1.) Thread creation by extending the Thread class

We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call Start () method to start the execution of a thread. Start () invokes the run () method on the Thread object.

```
// Java code for thread creation by
// extending Thread class.

class MultithreadingDemo extends Thread
{
    public void run()
    {
        try
        {
            System.out.println("Thread" +
                      Thread.currentThread().getId()
                      + " is running");
        }
        catch (Exception e)
        {
            System.out.println("Exception is caught");
        }
    }
}

// Main class
```

Date : / /
Page :

```
public class Multithread
{
    public static void main(String[] args)
    {
        int n=8;
        for( int i=0; i<n; i++)
        {
            MultithreadingDemo object =
                new MultithreadingDemo();
            object.start();
        }
    }
}
```

Output

```
Thread 15 is running
Thread 14 is running
  "    16  "    "
  "    12  "    "
  "    11  "    "
  "    13  "    "
  "    18  "    "
  "    17  "    "
```

(2) Thread creation by implementing the Runnable Interface

———— x ———— x ———— x ————

We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

```
// Java code for thread creation by
// implementing the Runnable Interface

class MultithreadingDemo implements Runnable
{
    public void run()
    {
        try
        {
            System.out.println("Thread" +
            Thread.currentThread().getId() + " is
                                        running");
        }
        catch (Exception e)
        {
            System.out.println("Exception is
                                        caught");
        }
    }
}
// Main class
class Multithread
{
```

```
public static vaid main (String []args)
{
    int n = 8; // number of threads
    for( int i=0 ; i<n ; i++)
    {
            Thread object = new Thread
                (new multithreadingDemo());

            object.start();
    }
}

}
```

Output :-

```
Thread 13 is running
Thread 11 is running
  "    12  "    "
  "    15  "    "
  "    14  "    "
  "    18  "    "
  "    17  "    "
  "    16  "    "
```

*** Start () method :-**

The start () method of thread class is used to begin the execution of thread. The result of this method is two threads that are running concurrently : the current thread (which returns from the call to the Start method) and the other thread (which executes its run. method).

The start () method internally calls the run() method of Runnable interface to execute the code Specified in the run () method in a separate thread.

The start thread performs the following tasks :

- it ~~staurt~~ stats a new thread.
- The thread moves from New State to Runnable state.
- When the thread gets a ~~chang~~ chance to execute, its target run () method will run.

<u>Syntax</u> :. public void start ();

Return value : It does not return any value.

* <u>run () method</u> :-

The run () method of thread class is called if the thread was constructed using a separate Runnable object otherwise this method does nothing and returns.

When the run () method calls, the code specified in the run () method is executed. you can call the run () method multiple times.
The run() method can be called using the start () method or by calling the run() method itself. But when you use run () method for calling itself, it creates problems.

Return : It does not return any value.

* **Stop () method :-**
~~1st~~ Whenever we want to stop a thread from running state by calling stop () method of thread class in Java. This method stops the execution of a running thread and removes it from the waiting threads pool and garbage collected. A thread will also move to the dead state automatically when it reaches the end of its method. The stop () method is deprected in Java due to thread-safety issues.
Syntax :- public final void stop ()

Date : / /
Page :

(3.) What way you would create a program to generate the threads :-
- To display Armstrong number upto n numbers.
- To display the table of a given number.

Ans:- // Demo program illustrate a program
// to generate the threads
// display Armstrong number
// and display table of a given number.

```
import java.lang.*;
import java.util.*;

class Armstrong extends Thread
{
    public void run()
    {
        int n, ~~a~~ t, b, sum=0;
        Scanner sc = new Scanner(System.in);
       System.out.println("Enter a number
            for limit of Armstrong:");
       int number = sc.nextInt()'
      System.out.println("~~Enter~~ Armstrong
      numbers from 1 to" + number + ":");
      for (int i=1; i<=number; i++)
      {
          n=i;
          while(n>0)
          {
```

Date : / /
Page :

```
        b = n % 10 ;
        Sum = sum + (b*b*b);
        n = n / 10 ;
      }
     if (sum == i)
     {
        System.out.println(i+" ");
     }
     sum = 0;
    }
   }
  }

  Class Table extends Thread
  {
      public void run()
      {
        Scanner sc = new scanner(system.in);
        int n, i;
        System.out.println("Enter a
        number for printing table: ");
        n = sc.nextInt ();
       for( i=1; i<=10; i++)
       {  system.out.println((n*i));
       }
    }
}
public class MyThread
{   public static void main(String args[])
    {
        Armstrong t1 = new Armstrong();
        Table t2 = new Table();
         t1.start();
         t2.start();
    }
}
```

Output

```
Enter a number for printing table :
Enter a number for limit of Armstrong :
1000
 2
2
4
6
8
10
12
14
16
18
20
Armstrong numbers from 1 to 1000 : 1 153 370 371 40
```