Batch: A4 Roll No.: 16010122083

Experiment / assignment / tutorial No.08

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Multithreading Programming

AIM: Write a java program that implements a multi-thread application that has three threads. First thread generates a random integer every 1 second and if the value is even, the second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of the cube of the number.

Expected OUTCOME of Experiment:

CO1: Understand the features of object oriented programming compared with procedural approach with C++ and Java

CO4: Explore the interface, exceptions, multithreading, packages.

. ---

Books/ Journals/ Websites referred:

1. Ralph Bravaco , Shai Simoson , “Java Programming From the Group Up” Tata McGraw-Hill.

2.Grady Booch, Object Oriented Analysis and Design .

Pre Lab/ Prior Concepts:

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. A multithreading is a specialized form of multitasking. Multithreading requires less overhead than multitasking processing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

Creating a Thread:
Java defines two ways in which this can be accomplished:

1. You can implement the Runnable interface.
2. You can extend the Thread class itself.

Create Thread by Implementing Runnable:
The easiest way to create a thread is to create a class that implements the Runnable interface.
To implement Runnable, a class needs to 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:

```
class NewThread implements Runnable {
  Thread t;
  NewThread() {
    t = new Thread(this, "Demo Thread");
    System.out.println("Child thread: " + t);
    t.start(); // Start the thread
  }
    public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
        // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
```

```
    }
    System.out.println("Exiting child thread.");
  }
}

public class ThreadDemo {
  public static void main(String args[]) {
    new NewThread();
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Main Thread: " + i);
        Thread.sleep(100);
      }
    } catch (InterruptedException e) {
      System.out.println("Main thread interrupted.");
    }
    System.out.println("Main thread exiting.");
  }
}
```

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
The extending class must override the run( ) method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

```
class NewThread extends Thread {
  NewThread() {
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start(); // Start the thread
  }
   public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
                // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}

public class ExtendThread {
```

```
  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(100);
      }
    } catch (InterruptedException e) {
      System.out.println("Main thread interrupted.");
    }
    System.out.println("Main thread exiting.");
  }
}
```

Some of the Thread methods

Methods	Description
void setName(String name)	Changes the name of the Thread object. There is also a getName() method for retrieving the name
Void setPriority(int priority)	Sets the priority of this Thread object. The possible values are between 1 and 10. 5
boolean isAlive()	Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.
void yield()	Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
void sleep(long millisec)	Causes the currently running thread to block for at least the specified number of milliseconds.
Thread currentThread()	Returns a reference to the currently running thread, which is the thread that invokes this method.

Algorithm:

1. Start
2. In the Main class all the threads are implemented which works in an infinite loop.
3. First thread is used to generate Random number which uses Exception handling and has sleep time of 1000ms and if the number is even it goes to the Square Thread and for odd its Cube of the number
4. In second thread when called it gives the Square of the number generated and prints it .
5. In the third thread when called it gives the Cube of the number generated and prints it.
6. Stop

Implementation details:

```
import java.util.Random;



class NumberGenerator implements Runnable
{
   private Random random = new Random();


   @Override
   public void run()
   {
       try {
           while (true) {
               int randomNumber = random.nextInt(100);
               System.out.println("Generated Number: " + randomNumber);


               if (randomNumber % 2 == 0)
               {


                   synchronized (SquareCalculator.lock)
                   {
                       SquareCalculator.number = randomNumber;
                       SquareCalculator.lock.notify();
                   }
               }
               else
               {
                   synchronized (CubePrinter.lock)
                   {
```

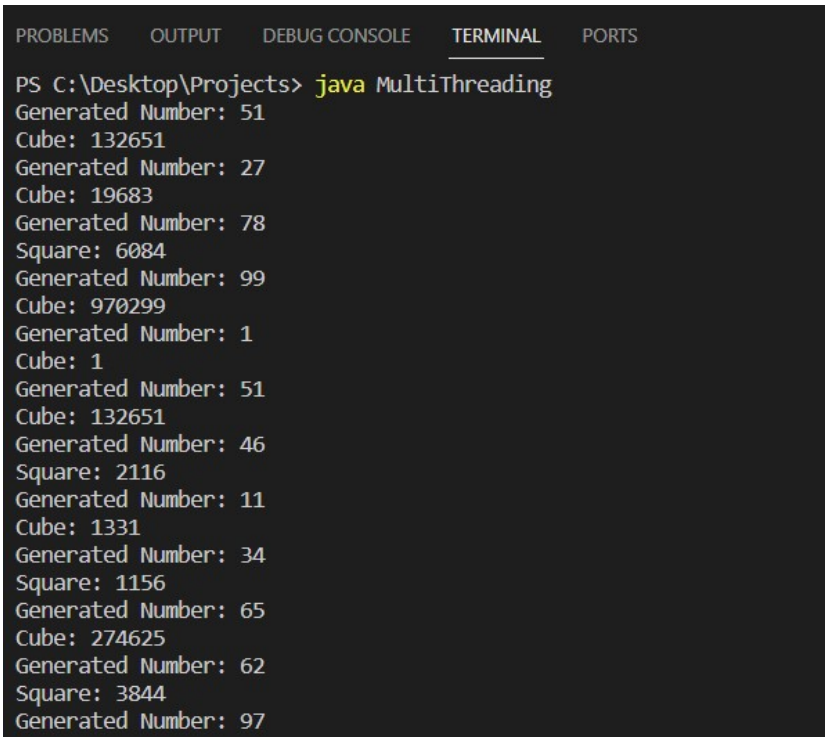
```
                    CubePrinter.number = randomNumber;
                    CubePrinter.lock.notify();
                }
            }

            Thread.sleep(1000);
        }
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
  }
}



class SquareCalculator implements Runnable
{
  static final Object lock = new Object();
  static int number;



  @Override
  public void run()
  {
    try
    {
        while (true)
        {
```

```
                synchronized (lock)
                {
                    lock.wait();

                    int square = number * number;
                    System.out.println("Square: " + square);
                }
            }
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}

class CubePrinter implements Runnable
{
    static final Object lock = new Object();
    static int number;

    @Override
    public void run()
    {
        try {
            while (true) {
                synchronized (lock)
                {
                    lock.wait();
```

```
                int cube = number * number * number;
                System.out.println("Cube: " + cube);
            }
        }
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
  }
}
```

```
public class MultiThreading
{
   public static void main(String[] args)
   {
       Thread generatorThread = new Thread(new NumberGenerator());
       Thread squareThread = new Thread(new SquareCalculator());
       Thread cubeThread = new Thread(new CubePrinter());
       generatorThread.start();
       squareThread.start();
       cubeThread.start();
   }
}
```

Output:



Conclusion: We learned the concept of the Multithreading and also implement with an example.

Date:_____ **Signature of faculty in-charge**

Post Lab Descriptive Questions

1. What do you mean by multithreading?

   Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.

2. Explain the use of sleep and run function with an example?

   sleep: It is used to temporarily halt the execution of a thread or program for a specified period of time. This can be useful for introducing delays or controlling the timing of actions in your code, like waiting for a specific event or implementing time-related functionality.

```java
import java.io.*;
import java.lang.Thread;
class SleepExample
{
    public static void main(String[] args)
    {

        try
        {
            for (int i = 0; i < 5; i++)
            {


                Thread.sleep(1000);

                System.out.println(i);
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

   run: This method is defined in a class that extends the Thread class or implements the Runnable interface in languages like Java. The run method contains the actual code or instructions that a thread will execute when it's started. To run this code, you

create an instance of the thread class and call the start method, which initiates the thread's execution by invoking the run method.

```
// Java Program for sleeping the custom thread.

import java.io.*;
import java.lang.Thread;
class RunExample extends Thread
{
    public void run()
    {

        try
        {
            for (int i = 0; i < 5; i++)
            {

                Thread.sleep(1000);

                System.out.println(i);
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
    public static void main(String[] args)
    {
        RunExample obj = new RunExample();
        obj.start();
    }
}
```

3. Explain any five methods of Thread class with Example ?

1. **public static Thread currentThread()**

The currentThread() method of thread class is used to return a reference to the currently executing thread object.

```
public class MultiThreading extends Thread
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String args[])
    {

        MultiThreading t1=new MultiThreading();
        MultiThreading t2=new MultiThreading();
        t1.start();
        t2.start();
    }
}
```

2. **public void setName(String name):**

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using the setName() method

```
class TestMultiNaming1 extends Thread{
  public void run()
  {
   System.out.println("running...");
  }
 public static void main(String args[])
 {
  TestMultiNaming1 t1=new TestMultiNaming1();
  TestMultiNaming1 t2=new TestMultiNaming1();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());

  t1.start();
  t2.start();

  t1.setName("Minav Karia");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

3. Void setPriority(int priority)

The setPriority() method of thread class is used to change the thread's priority. Every thread has a priority which is represented by the integer number between 1 to 10.

```
public class JavaSetPriority extends Thread
{
    public void run()
    {
        System.out.println("Priority of thread is: "+Thread.currentThread().getPriority());
    }
    public static void main(String args[])
    {

        JavaSetPriority t1=new JavaSetPriority();
        t1.setPriority(Thread.MAX_PRIORITY);
        t1.start();
    }
}
```

4. boolean isAlive()

The isAlive() method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.

```
public class JavaIsAliveExp extends Thread
{
    public void run()
    {
        try
        {
            Thread.sleep(300);
            System.out.println("is run() method isAlive "+Thread.currentThread().isAlive());
        }
        catch (InterruptedException ie) {
        }
    }
    public static void main(String[] args)
    {
        JavaIsAliveExp t1 = new JavaIsAliveExp();
        System.out.println("before starting thread isAlive: "+t1.isAlive());
        t1.start();
        System.out.println("after starting thread isAlive: "+t1.isAlive());
    }
}
```

5. join ()

The join method is used to wait for a thread to finish its execution before the current thread continues.

```
Thread thread1 = new Thread(() ->
{
    for (int i = 1; i <= 5; i++) {
        System.out.println("Thread 1 - Count: " + i);
    }
});

Thread thread2 = new Thread(() ->
{
    for (int i = 1; i <= 5; i++) {
        System.out.println("Thread 2 - Count: " + i);
    }
});
thread1.start();
thread2.start();
try
{
    thread1.join();
    thread2.join();
} catch (InterruptedException e)
{
    e.printStackTrace();
}
```