Batch: D2 Roll No.:16010122323
Experiment / assignment / tutorial No. 8
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 random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of 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 Programing 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.");
    }
}
```

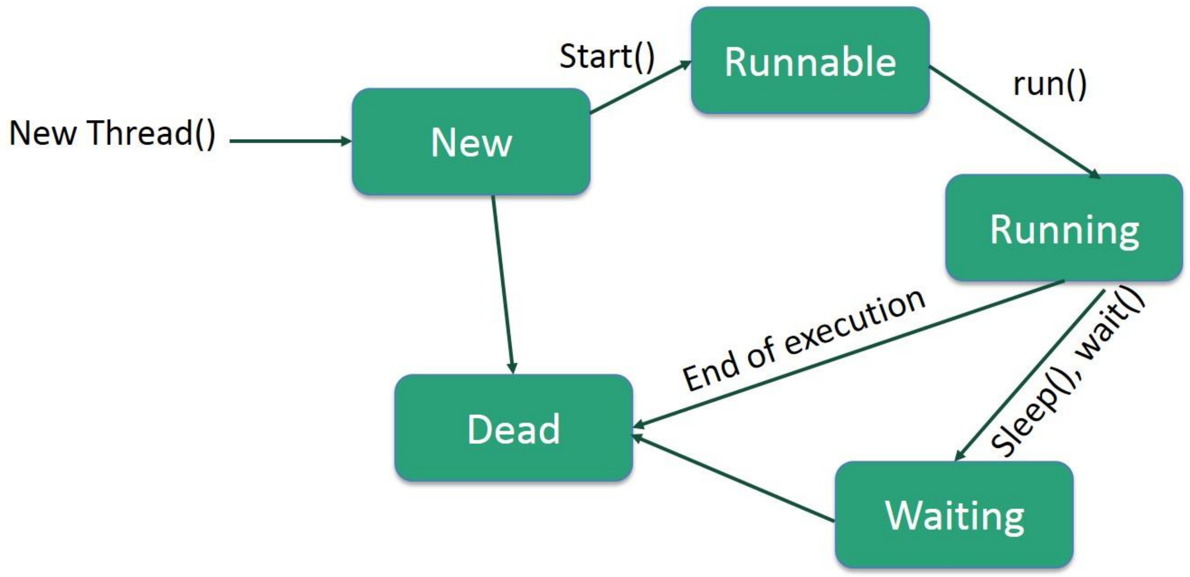
The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class.
The extending class must override the run( ) method, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

```
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.

Class Diagram:



Algorithm:

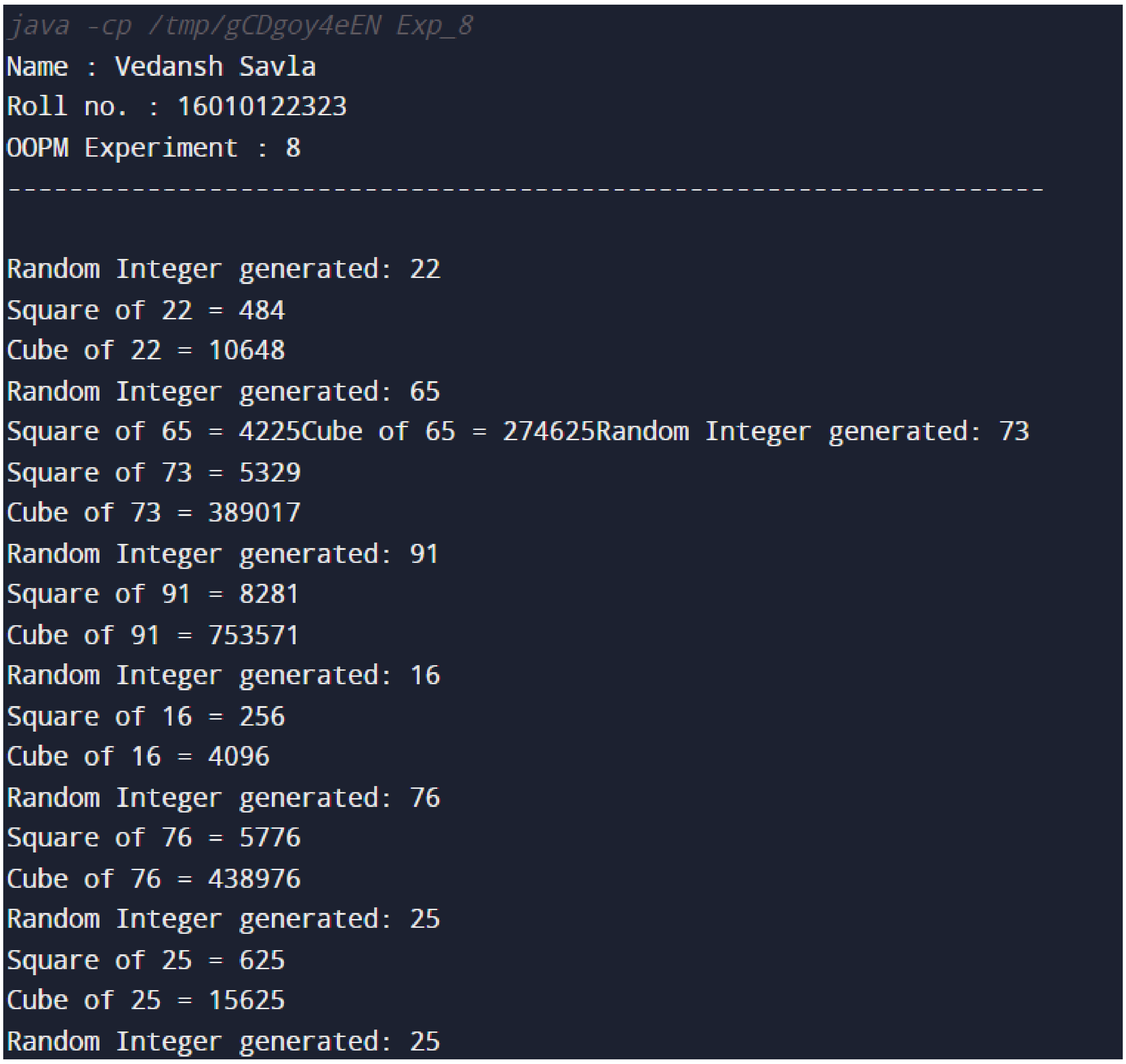
- **Code Line 2:** We are creating a class “thread_Example1” which is implementing the Runnable interface (it should be implemented by any class whose instances are intended to be executed by the thread.)
- **Code Line 4:** It overrides run method of the runnable interface as it is mandatory to override that method
- **Code Line 6:** Here we have defined the main method in which we will start the execution of the thread.
- **Code Line 7:** Here we are creating a new thread name as “guruthread1” by instantiating a new class of thread.
- **Code Line 8:** we will use “start” method of the thread using “guruthread1” instance. Here the thread will start executing.
- **Code Line 10:** Here we are using the “sleep” method of the thread using “guruthread1” instance. Hence, the thread will sleep for 1000 milliseconds.
- **Code 9-14:** Here we have put sleep method in try catch block as there is checked exception which occurs i.e. Interrupted exception.
- **Code Line 15:** Here we are setting the priority of the thread to 1 from whichever priority it was
- **Code Line 16:** Here we are getting the priority of the thread using getPriority()
- **Code Line 17:** Here we are printing the value fetched from getPriority
- **Code Line 18:** Here we are writing a text that thread is running.

Implmentation details:

```
import java.util.Random;

class Square extends Thread {
    int x;

    Square(int n) {
        x = n;
    }

    public void run() {
        int sqr = x * x;
        System.out.println("Square of " + x + " = " + sqr);
    }
```

```java
}

class Cube extends Thread {
    int x;

    Cube(int n) {
        x = n;
    }

    public void run() {
        int cub = x * x * x;
        System.out.println("Cube of " + x + " = " + cub);
    }
}

class Number extends Thread {
    public void run() {
        System.out.println("Name : Vedansh Savla \nRoll no. : 16010122323 
\nOOPM Experiment : 8");
        System.out.println("--------------------------------------------------
-----------------\n");
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            int randomInteger = random.nextInt(100);
            System.out.println("Random Integer generated: " + randomInteger);
            Square s = new Square(randomInteger);
            s.start();
            Cube c = new Cube(randomInteger);
            c.start();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.out.println(ex);
            }
        }
    }
}

class Exp_8 {
    public static void main(String args[]) {
        Number n = new Number();
        n.start();
    }
}
```

Output:



```
java -cp /tmp/gCDgoy4eEN Exp_8
Name : Vedansh Savla
Roll no. : 16010122323
OOPM Experiment : 8
-------------------------------------------------------------------

Random Integer generated: 22
Square of 22 = 484
Cube of 22 = 10648
Random Integer generated: 65
Square of 65 = 4225Cube of 65 = 274625Random Integer generated: 73
Square of 73 = 5329
Cube of 73 = 389017
Random Integer generated: 91
Square of 91 = 8281
Cube of 91 = 753571
Random Integer generated: 16
Square of 16 = 256
Cube of 16 = 4096
Random Integer generated: 76
Square of 76 = 5776
Cube of 76 = 438976
Random Integer generated: 25
Square of 25 = 625
Cube of 25 = 15625
Random Integer generated: 25
```

Conclusion:

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

Post Lab Descriptive Questions

1. What do you mean by multithreading?

Multithreading in java

is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading :

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

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

The Java Thread class provides the two variant of the sleep() method. First one accepts only an arguments, whereas the other variant accepts two arguments. The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

EXAMPLE :

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
  // the thread will sleep for the 500 milli seconds
    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

Output:

```
1
1
2
2
3
```

```
3
4
4
```

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

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.

EXAMPLE :

```
public class RunExp1 implements Runnable

{

  public void run()

  {

    System.out.println("Thread is running...");

  }

  public static void main(String args[])

  {

    RunExp1 r1=new RunExp1();

    Thread t1 =new Thread(r1);

    // this will call run() method

    t1.start();

  }

}
```

Output:

```
Thread is running...
```