K. J. Somaiya College of Engineering, Mumbai-77

Batch: B1 **Roll No.: 1711072**

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 multithread application that has three threads. First thread generates random integer every second and if value is even, second thread computes the square of the number then prints it. If the value is odd, the third thread will print the cube of that number.

Expected OUTCOME of Experiment:

CO4: Demonstrate programs on interface, exceptions, multithreading and applets.

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.

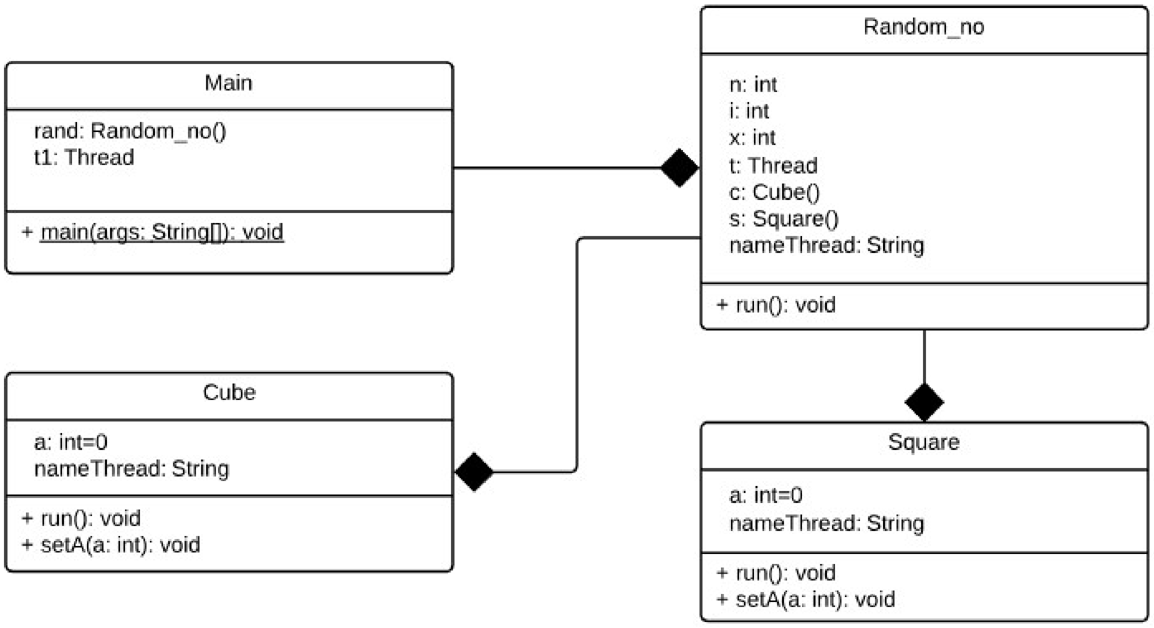
```
class NewThread extends Thread {
  NewThread() {
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start(); // Start the thread
  }
   public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
                // 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:



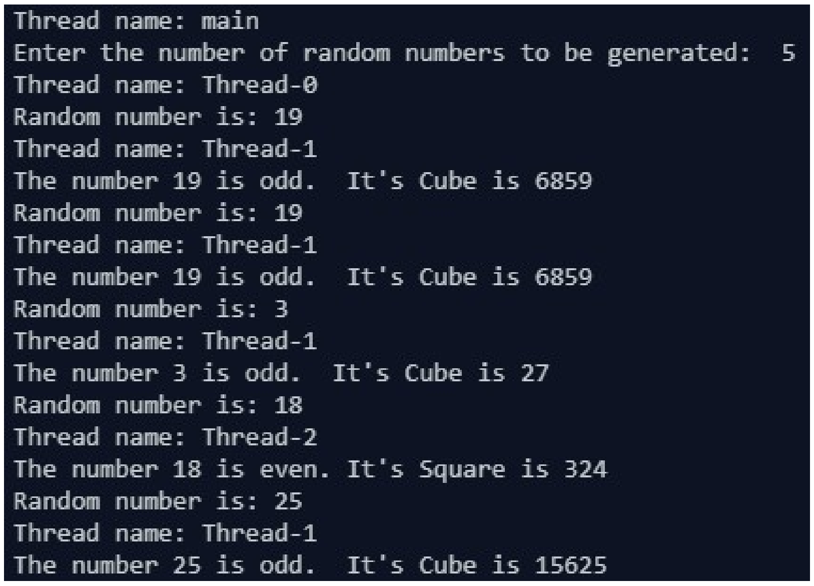
Implementation details:

NOTE: The program is using only three threads and not multiple threads. This has been done using resume() and suspend().

```
import java.util.*;
class Main{
  public static void main(String [] args){
    Random_no rand = new Random_no();
    Thread t1 = new Thread(rand);
    t1.start();
    Thread t=Thread.currentThread();
    String nameThread=t.getName();
    System.out.println("Thread name: "+nameThread);
  }
}
class Random_no implements Runnable{
   public void run(){
     System.out.print("Enter the number of random numbers to be
generated: ");
     Scanner sc=new Scanner(System.in);
     int n=sc.nextInt();
     int x,i;
     Thread t=Thread.currentThread();
     String nameThread=t.getName();
     Cube c=new Cube();
     Square s=new Square();
     c.start();
     s.start();
     System.out.println("Thread name: "+nameThread);
     for(i=0;i<n;i++){
         Random r=new Random();
         x=r.nextInt(50);
         System.out.println("Random number is: "+x);
         if(x%2==0){
           s.setA(x);
           s.resume();
         }
         else{
           c.setA(x);
           c.resume();
         }
      try{
        Thread.sleep(1000);
      }
      catch(InterruptedException e){
      System.out.println(e);
      }
    }
    s.stop();
    c.stop();
  }
}
class Square extends Thread
{
  int a=0;
  public void setA(int a){
    this.a = a;
  }
  public void run(){
    String nameThread=this.getName();
    while(true){
      this.suspend();
      System.out.println("Thread name: "+nameThread);
      System.out.println("The number " + a + " is even."+ " It's
Square is "+a*a);
    }
  }
}
class Cube extends Thread
{
  int a = 0;
  public void setA(int a){
    this.a = a;
  }
  public void run(){
    String nameThread=this.getName();
    while(true){
      this.suspend();
      System.out.println("Thread name: "+nameThread);
      System.out.println("The number " + a + " is odd."+ "  It's
Cube is "+a*a*a);
    }
  }
}
```

For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/MultiThreading

Output Screen:



```
Thread name: main
Enter the number of random numbers to be generated:  5
Thread name: Thread-0
Random number is: 19
Thread name: Thread-1
The number 19 is odd.  It's Cube is 6859
Random number is: 19
Thread name: Thread-1
The number 19 is odd.  It's Cube is 6859
Random number is: 3
Thread name: Thread-1
The number 3 is odd.  It's Cube is 27
Random number is: 18
Thread name: Thread-2
The number 18 is even. It's Square is 324
Random number is: 25
Thread name: Thread-1
The number 25 is odd.  It's Cube is 15625
```

Conclusion: The program ran successfully as we were able to execute the program using three threads only and got correct output every time for all test cases.

Post Lab Descriptive Questions (Add questions from examination point view)

1. **What do you mean by multithreading?**

Ans. 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?**

Ans.

1. **Thread.sleep()** method can be used to pause the execution of current thread for specified time in milliseconds. The argument value for milliseconds can't be negative, else it throws **IllegalArgumentException**. There is another overloaded method **sleep(long millis, int nanos)** that can be used to pause the execution of current thread for specified milliseconds and nanoseconds. The allowed nano second value is between 0 and 999999. sleep() interacts with the thread scheduler to put the current thread in wait state for specified period of time. Once the wait time is over, thread state is changed to runnable state and wait for the CPU for further execution. So the actual time that current thread sleep depends on the thread scheduler that is part of OS.

2. The **java.lang.Thread.run()** method is called if this thread was constructed using a separate Runnable run object, else this method does nothing and returns.

When creating and starting a thread a common mistake is to call the run() method of the Thread instead of start(), like this:

```
Thread newThread = new Thread(MyRunnable());
 newThread.run();  //should be start();
```

At first you may not notice anything because the Runnable's run() method is executed like you expected. However, it is NOT executed by the new thread you just created. Instead the run() method is executed by the thread that created the thread. In other words, the thread that executed the above two lines of code. To have the run() method of the MyRunnable instance called by the new created thread, newThread, you MUST call the newThread.start() method.

An example:

```
class TestCallRun2 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{Thread.sleep(500);}catch(InterruptedException e){System.
out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestCallRun2 t1=new TestCallRun2();
  TestCallRun2 t2=new TestCallRun2();
     t1.run();
     t2.run();
    }
   }
```

Date: 19/10/2018 **Signature of faculty in-charge**