Course Name:	**Object Oriented Programming Methodology**	**Semester:**	**III**
Date of Performance:	**13 / 10 / 2023**	**Batch No:**	**B2**
Faculty Name:	**Prof. Kiran Thale**	**Roll No:**	**16010122151**
Faculty Sign & Date:		**Grade/Marks :**	**___/25**

Experiment No: 8

Title: Multithreading Programming

Aim and Objective of the Experiment:

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.

COs to be achieved:

CO1: Understand the features of object oriented programming compared with procedural approach with C++ and Java.
CO2: Explore arrays, vectors, classes and objects in C++ and Java.
CO3: Implement scenarios using object oriented concepts (Drawing class diagram, relationship between classes, sequence diagram)
CO4: Explore the interface, exceptions, multithreading, packages

Tools used:

JDK, VScode / Eclipse

Theory:

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:-

```
1  class NewThread implements Runnable {
2      private Thread thread;
3
4      NewThread() {
5          thread = new Thread(this, "Demo Thread");
6          System.out.println("Child thread: " + thread);
7          thread.start(); // Start the thread
8      }
9
10     public void run() {
11         try {
12             for (int i = 5; i > 0; i--) {
13                 System.out.println("Child Thread: " + i);
14                 // Let the thread sleep for a while.
15                 Thread.sleep(50);
16             }
17         } catch (InterruptedException e) {
18             System.out.println("Child interrupted.");
19         }
20         System.out.println("Exiting child thread.");
```

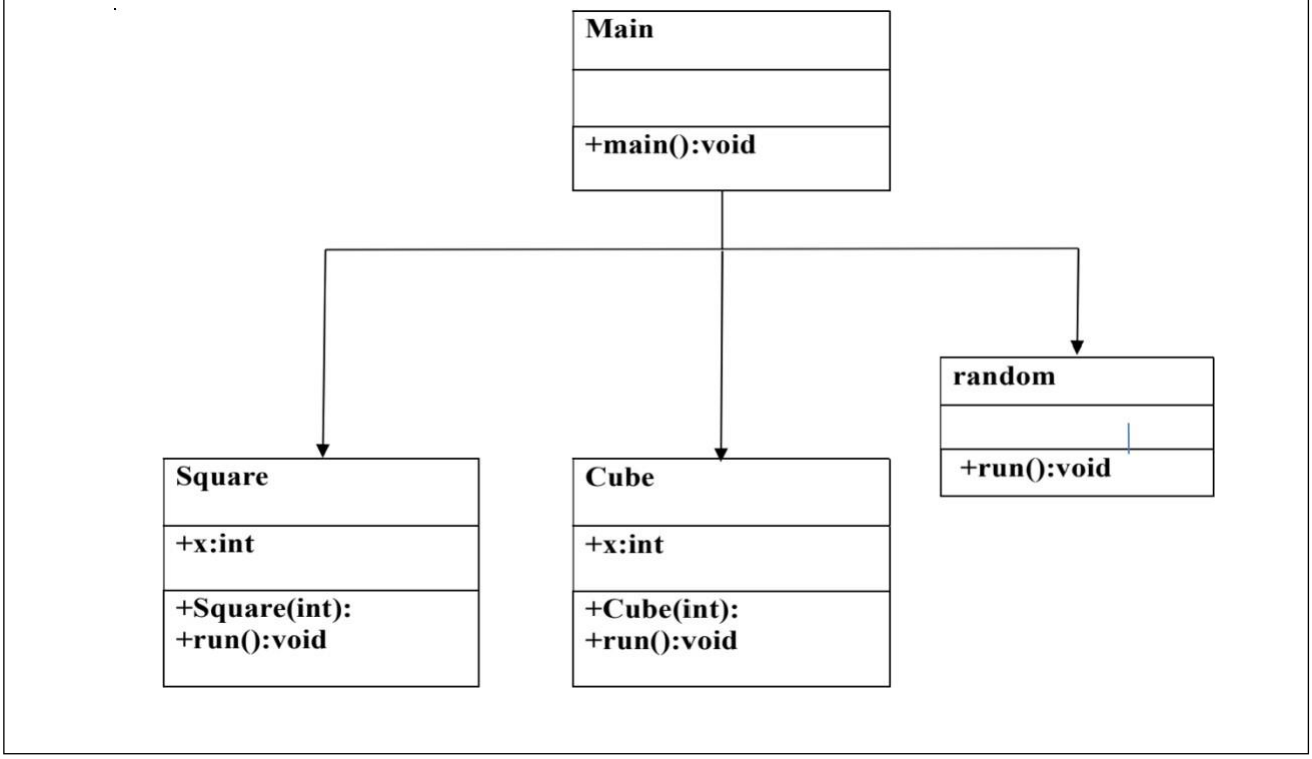
```
    }
}

public class ThreadDemo {
    public static void main(String args[]) {
        NewThread newThread = 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.");
    }
}
```

```java

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.");
    }
}

```

Class Diagram:



Algorithm:

Step 1: Create class Random and extend Thread to it

Step 2: Override the function run in the Random class
i. Generate a random number using the Random class
ii. Create objects of thread Square and thread Cube
iii. Check if the number generated is even:
- If even, start thread Square
- If odd, start thread Cube

Step 3: Create a class Square
- Define a class named Square
- Create a parametrized constructor to initialize the class variables

Step 4: Override the function run in the Square class
i. Calculate and print the square of the number received from the Random class

Step 5: Create a class Cube
- Define a class named Cube
- Create a parametrized constructor to initialize the class variables

Step 6: Override the function run in the Cube class
i. Calculate and print the cube of the number received from the Random class

Step 7: Create a class Main with a main function

Step 8: In the main function:
- Create an object of the Random class
- - Start the Random thread, which generates random numbers and decides whether to start the Square or Cube thread based on the randomness and even/odd criteria

Code:

```java
import java.util.Random;

class RandomNumberGenerator extends Thread {
    public void run() {
        Random random = new Random();
        for (int i = 0; i < 5; i++) {
            int number = random.nextInt(100);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                SquareThread squareThread = new SquareThread(number);
                CubeThread cubeThread = new CubeThread(number);

                if (number % 2 == 0) {
                    squareThread.start();
                } else {
                    cubeThread.start();
                }
            }
        }
    }
}

class SquareThread extends Thread {
    private int number;

    public SquareThread(int number) {
        this.number = number;
    }

    public void run() {
        System.out.println("Square of " + number + " is " + number * number);
    }
}

class CubeThread extends Thread {
    private int number;

    public CubeThread(int number) {
        this.number = number;
    }
```

```
43      }
44
45      public void run() {
46          System.out.println("Cube of " + number + " is " + number * number * number);
47      }
48  }
49
50  public class NumberOperations {
51      public static void main(String[] args) {
52          RandomNumberGenerator randomNumberGenerator = new RandomNumberGenerator();
53          randomNumberGenerator.start();
54      }
55  }
56
```

OUTPUT

```
Square of 64 is 4096
Square of 6 is 36
Cube of 15 is 3375
Square of 20 is 400
Cube of 43 is 79507
```

```
Cube of 71 is 357911
Square of 56 is 3136
Cube of 23 is 12167
Square of 56 is 3136
Cube of 59 is 205379
```

Post Lab Subjective/Objective type Questions:

1. What do you mean by multithreading?

Multithreading is a concurrent execution technique in which multiple threads run independently within a single process. Each thread represents a separate unit of execution and has its own program counter, stack, and set of registers, allowing it to execute code independently of other threads in the same process. Multithreading enables a program to perform multiple tasks or operations in parallel, taking advantage of the underlying hardware's ability to execute multiple threads simultaneously, or at least in an interleaved manner.

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

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.

Example :

```
public class ThreadSleep {
 public static void main(String[] args) throws
InterruptedException {
 long start = System.currentTimeMillis();
Thread.sleep(2000);
System.out.println("Sleep time in ms = " +
(System.currentTimeMillis() - start));
 }
```

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

---

```
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();
}
```

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

activeCount() : Returns an estimate of the number of active threads in the current thread's thread group and its subgroups.

checkAccess() : Determines if the currently running thread has permission to modify this thread.

Clone(): Throws CloneNotSupportedException as a Thread can not be meaningfully Cloned

currentThread() : Returns a reference to the currently executing thread object

dumpStack(): Prints a stack trace of the current thread to the standard error stream

---

```
class MyThread extends Thread {
 public void run() {
 System.out.println("Thread is running created by
extending to parent Thread class");
 }
 public static void main(String[] args) {
 MyThread myThread = new MyThread();
 myThread.start();
 }
```

Conclusion:

Thus, multithreading was understood and implemented

Signature of faculty in-charge with Date: