**Batch: A3 Roll No.: 16010122074**

**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 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:**

This Java code demonstrates a simple example of multi-threading using three threads: NumberGenerator, SquareCalculator, and CubePrinter. The program generates random numbers in an infinite loop, and based on whether the generated number is even or odd, it notifies either the square or cube calculation thread to perform its task.

Here's a step-by-step explanation of the algorithm:

* NumberGenerator Class:

Implements the Runnable interface, making it suitable for execution by a thread.

Generates a random number between 0 and 99 (inclusive) in an infinite loop.

Prints the generated number to the console.

If the number is even, it notifies the SquareCalculator thread by acquiring the lock on SquareCalculator.lock.

If the number is odd, it notifies the CubePrinter thread by acquiring the lock on CubePrinter.lock.

Sleeps for 1 second before generating the next number.

* SquareCalculator Class:

Implements the Runnable interface.

Runs in an infinite loop.

Waits for a notification from the NumberGenerator thread by acquiring the lock on SquareCalculator.lock.

Upon notification, calculates the square of the number received from NumberGenerator.

Prints the square to the console.

* CubePrinter Class:

Implements the Runnable interface.

Runs in an infinite loop.

Waits for a notification from the NumberGenerator thread by acquiring the lock on CubePrinter.lock.

Upon notification, calculates the cube of the number received from NumberGenerator.

Prints the cube to the console.

* Main Class (MultiThreadExample):

Creates instances of the NumberGenerator, SquareCalculator, and CubePrinter classes.

Creates separate threads for each of the instances.

Starts all three threads concurrently.

In summary, the NumberGenerator generates random numbers and notifies either the SquareCalculator or CubePrinter thread based on whether the number is even or odd. The other two threads wait for notifications, perform their respective calculations, and print the results. The program runs indefinitely until manually terminated.

**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); // Assuming a range, you can adjust it

                System.out.println("Generated Number: " + randomNumber);

                if (randomNumber % 2 == 0) {

                    // If the number is even, notify the SquareCalculator thread

                    synchronized (SquareCalculator.lock) {

                        SquareCalculator.number = randomNumber;

                        SquareCalculator.lock.notify();

                    }

                } else {

                    // If the number is odd, notify the CubePrinter thread

                    synchronized (CubePrinter.lock) {

                        CubePrinter.number = randomNumber;

                        CubePrinter.lock.notify();

                    }

                }

                Thread.sleep(1000); // Wait for 1 second

            }

        } 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();

        }

    }

}

 class MultiThreadExample {

    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();

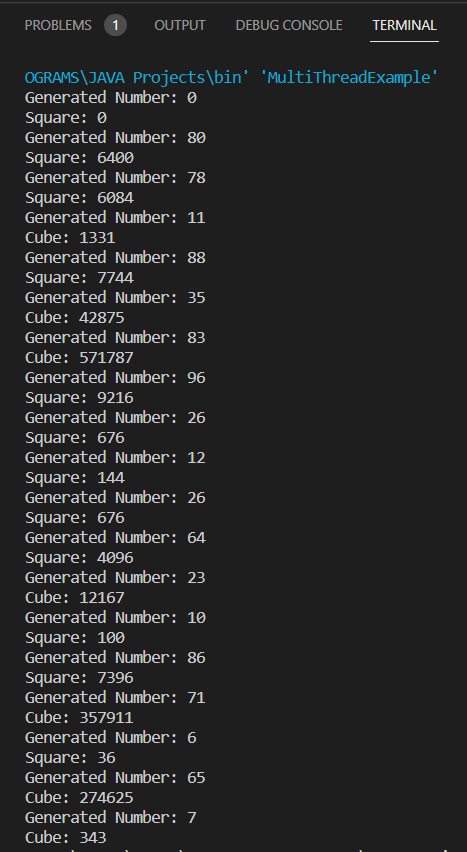
        squareThread.start();

        cubeThread.start();

    }

}

**Output:**



**Conclusion:** I learned how to write a java program that implements a multi-thread application that has three threads.

**Date:\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. What do you mean by multithreading?

Multithreading refers to concurrent execution of multiple threads within the same program. A thread is the smallest unit of a process that can run independently, allowing multiple tasks to be performed simultaneously. Multithreading enables efficient utilization of system resources and can improve the responsiveness and performance of a program.

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

sleep() Function:

Used to pause the execution of a thread for a specified duration.

Helps control the timing or pacing of thread execution.

run() Function:

Contains the code to be executed by a thread when it is started.

Called automatically when a thread starts.

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

* **start() Method:**

Used to start the execution of a thread by invoking its run method in a separate thread of control.

**Example**

class MyThread extends Thread {

@Override

public void run() {

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

}

}

public class StartMethodExample {

public static void main(String[] args) {

MyThread myThread = new MyThread();

myThread.start(); // Starts a new thread, invoking the run method

}

}

* **join() Method:**

Waits for the thread on which it is called

**Example**

class MyThread implements Runnable {

@Override

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("Count: " + i);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class JoinMethodExample {

public static void main(String[] args) {

Thread thread = new Thread(new MyThread());

thread.start();

try {

thread.join(); // Waits for the thread to complete

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread has finished.");

}

}

* **sleep() Method:**

Causes the currently executing thread to sleep for a specified number of milliseconds.

**Example**public class SleepMethodExample {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

System.out.println("Count: " + i);

try {

Thread.sleep(1000); // Sleep for 1 second

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

* **isAlive() Method:**

Returns true if the thread upon which it is called is still running

**Example**

public class IsAliveMethodExample {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

for (int i = 1; i <= 5; i++) {

System.out.println("Count: " + i);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

thread.start();

while (thread.isAlive()) {

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

}

System.out.println("Thread has finished.");

}

}

* **setName() and getName() Methods:**

setName() is used to set the name of a thread.

getName() is used to obtain the name of a thread.

**Example**

public class NameMethodExample {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

System.out.println("Thread Name: " + Thread.currentThread().getName());

});

thread.setName("MyThread");

thread.start();

}

}