**Batch: C3 Roll No.: 16010123217**

**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.");

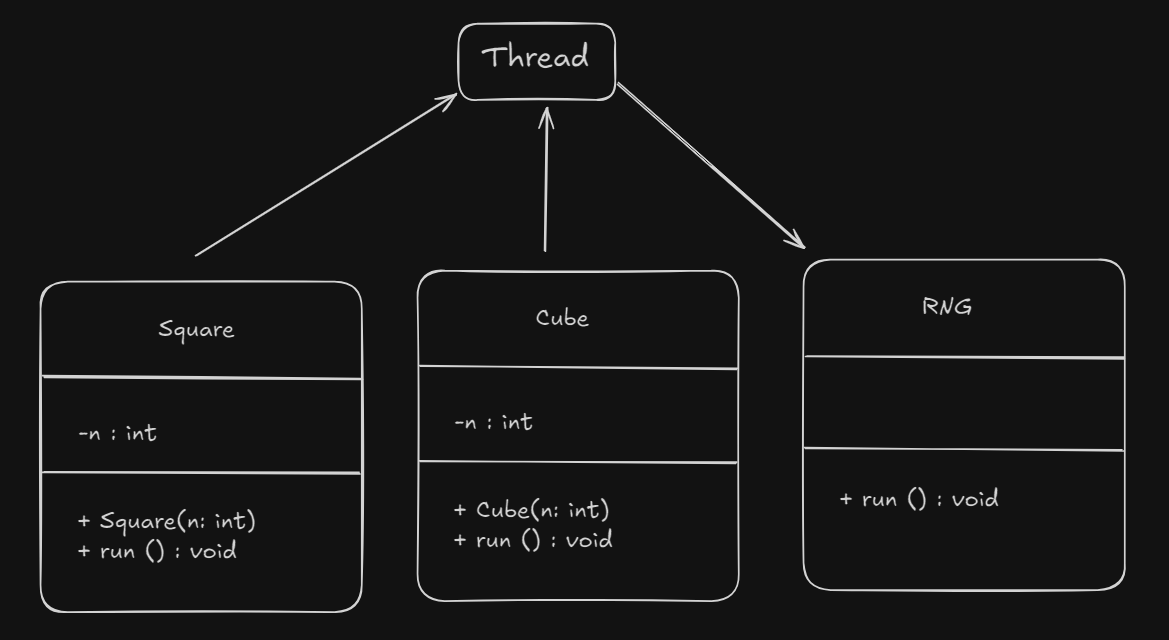
}

}

**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. |

**Class Diagram:**



**Algorithm:**

* + - 1. Start
      2. Generate the a Random Number in every 1 sec
      3. If the number is odd, run the run functions from class Cube
      4. If the number is even, run the run functions from class Square
      5. Display the result
      6. Sleep the thread for 1 sec
      7. Continue for 20 sec
      8. Exit

**Implementation details:**

import java.util.*\**;

class Square extends Thread {

    private int n;

    public Square(int n) {

*this*.n = n;

    }

    public void run() {

        System.out.println(n);

        System.out.println("Square: " + (n \* n));

    }

}

class Cube extends Thread {

    private int n;

    public Cube(int n) {

*this*.n = n;

    }

    public void run() {

        System.out.println(n);

        System.out.println("Cube: " + (n \* n \* n));

    }

}

class RNG extends Thread{

    public void run(){

        for(int i = 0; i<20; i++){

            Random r = new Random();

            int n = (r.nextInt())%10 + 10;

            Square s = new Square(n);

            Cube c = new Cube(n);

            if (n % 2 == 0){

                s.start();

                try {

                    Thread.sleep(1000);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

            else {

                c.start();

                try {

                    Thread.sleep(1000);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

        }

    }

}

public class Threading {

    public static void main(String[] args) {

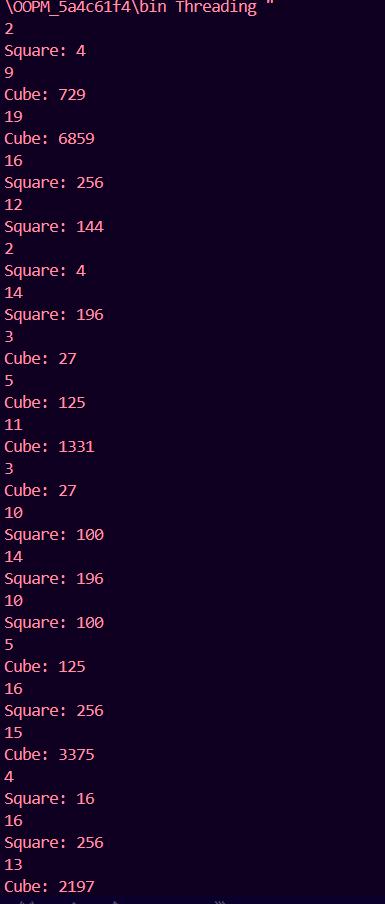
        RNG r = new RNG();

        r.start();

    }

}

**Output:**



**Conclusion:**

From this experiment, we learnt about Multi-threading which can be implemented in two ways Runnable interface and by extending the Thread class itself

**Date: 17/10/24 Signature of faculty in-charge**

**Post Lab Descriptive Questions**

1. What do you mean by Thread Synchronization ? Why is it needed? Explain with a program.

Ans Thread synchronization in Java is a way of programming several threads to carry out independent tasks easily. It is capable of controlling access to multiple threads to a particular shared resource.

The main reasons for using thread synchronization in Java are as follows:

* To prevent interference between threads.
* To prevent the problem of consistency.

Without Synchronization

class Counter {

    private int count = 0;

    public void increment() {

        count++;

    }

    public int getCount() {

        return count;

    }

}

class Worker extends Thread {

    private Counter counter;

    public Worker(Counter counter) {

*this*.counter = counter;

    }

    public void run() {

        for (int i = 0; i < 10000000; i++) {

            counter.increment();

        }

    }

}

public class WithoutSync {

    public static void main(String[] args) throws InterruptedException {

        Counter counter = new Counter();

        Worker t1 = new Worker(counter);

        Worker t2 = new Worker(counter);

        t1.start();

        t2.start();

        t1.join();

        t2.join();

        System.out.println("Final count (without synchronization): " + counter.getCount());

    }

}



As we can see the output is not 20000000 which means that some incrememts are done at the same time.

With Synchronisation

class Counter {

    private int count = 0;

    public synchronized void increment() {

        count++;

    }

    public int getCount() {

        return count;

    }

}

class Worker extends Thread {

    private Counter counter;

    public Worker(Counter counter) {

*this*.counter = counter;

    }

    public void run() {

        for (int i = 0; i < 10000000; i++) {

            counter.increment();

        }

    }

}

public class WithSync {

    public static void main(String[] args) throws InterruptedException {

        Counter counter = new Counter();

        Worker t1 = new Worker(counter);

        Worker t2 = new Worker(counter);

        t1.start();

        t2.start();

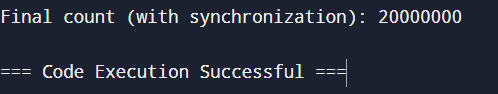
        t1.join();

        t2.join();

        System.out.println("Final count (with synchronization): " + counter.getCount());

    }

}



As we can see the output shows that the threads are correctly synchronized.

### 2.Write a program for multithreaded Bank Account System

Implement a multithreaded bank account system in Java such that the system should simulate transactions on a bank account that can be accessed and modified by multiple threads concurrently. Your goal is to ensure that all transactions are handled correctly and that the account balance remains consistent.

class BankAccount {

    private int balance;

    public BankAccount(int balance) {

*this*.balance = balance;

    }

    public synchronized void deposit(int amount) {

        balance += amount;

        System.out.println(Thread.currentThread().getName() + " deposited " + amount + ". Current balance: " + balance);

    }

    public synchronized void withdraw(int amount) {

        if (balance >= amount) {

            balance -= amount;

            System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ". Current balance: " + balance);

        } else {

            System.out.println(Thread.currentThread().getName() + " tried to withdraw " + amount + " but insufficient balance.");

        }

    }

    public int getBalance() {

        return balance;

    }

}

class TransactionThread extends Thread {

    private BankAccount account;

    private boolean deposit;

    private int amount;

    public TransactionThread(BankAccount account, boolean deposit, int amount) {

*this*.account = account;

*this*.deposit = deposit;

*this*.amount = amount;

    }

    public void run() {

        if (deposit) {

            account.deposit(amount);

        } else {

            account.withdraw(amount);

        }

    }

}

public class BankAccountSystem {

    public static void main(String[] args) throws InterruptedException {

        BankAccount account = new BankAccount(1000);

        TransactionThread t1 = new TransactionThread(account, true, 500);

        TransactionThread t2 = new TransactionThread(account, false, 200);

        TransactionThread t3 = new TransactionThread(account, false, 300);

        TransactionThread t4 = new TransactionThread(account, true, 700);

        TransactionThread t5 = new TransactionThread(account, false, 1500);

        t1.start();

        t2.start();

        t3.start();

        t4.start();

        t5.start();

        t1.join();

        t2.join();

        t3.join();

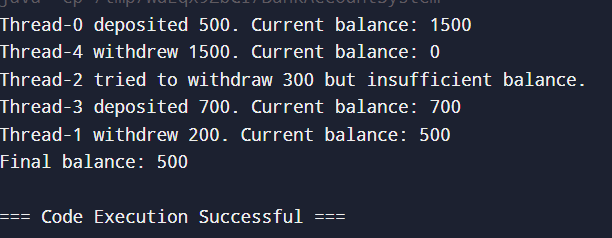
        t4.join();

        t5.join();

        System.out.println("Final balance: " + account.getBalance());

    }

}

Output:  


3. Draw thread lifecycle diagram. Explain any five methods of Thread class with Example ?

