**Batch: A2 Roll No.: 16010123032**

**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:**

+-----------------------------+

| MultiThreadedApplication |

+-----------------------------+

| - sharedData: SharedData |

| - generatorThread: Thread |

| - evenThread: Thread |

| - oddThread: Thread |

+-----------------------------+

| + main(args: String[]): void |

+-----------------------------+

|

|

V

+-------------------------+

| RandomNumberGenerator |

+-------------------------+

| - sharedData: SharedData |

+-------------------------+

| + run(): void |

+-------------------------+

|

|

V

+-------------------------+

| EvenSquareCalculator |

+-------------------------+

| - sharedData: SharedData |

+-------------------------+

| + run(): void |

+-------------------------+

|

|

V

+-------------------------+

| OddCubeCalculator |

+-------------------------+

| - sharedData: SharedData |

+-------------------------+

| + run(): void |

+-------------------------+

|

|

V

+------------------------+

| SharedData |

+------------------------+

| - number: int |

+------------------------+

| + setNumber(n: int): |

| synchronized void |

| + getNumber(): int |

+------------------------+

**Algorithm:**

 **Shared Data Setup**:

* Create a shared data object (SharedData) that stores a number.
* The object has two synchronized methods:
  + setNumber(int number) to update the number.
  + getNumber() to retrieve the current number.

 **Random Number Generation**:

* The RandomNumberGenerator thread continuously generates random numbers between 0 and 99.
* Each generated number is stored in the shared data object using setNumber().
* After generating and storing a number, the thread sleeps for 1 second.

 **Even Number Square Calculation**:

* The EvenSquareCalculator thread constantly retrieves the latest number from the shared data object using getNumber().
* If the number is even, it calculates the square of the number and prints the result.
* After each check, the thread sleeps for 100 milliseconds to allow time for the number to change.

 **Odd Number Cube Calculation**:

* The OddCubeCalculator thread constantly retrieves the latest number from the shared data object using getNumber().
* If the number is odd, it calculates the cube of the number and prints the result.
* After each check, the thread sleeps for 100 milliseconds to allow time for the number to change.

 **Thread Management**:

* In the MultiThreadedApplication class:
  + Create an instance of SharedData.
  + Create and start three threads:
    1. RandomNumberGenerator to generate random numbers.
    2. EvenSquareCalculator to compute squares of even numbers.
    3. OddCubeCalculator to compute cubes of odd numbers.

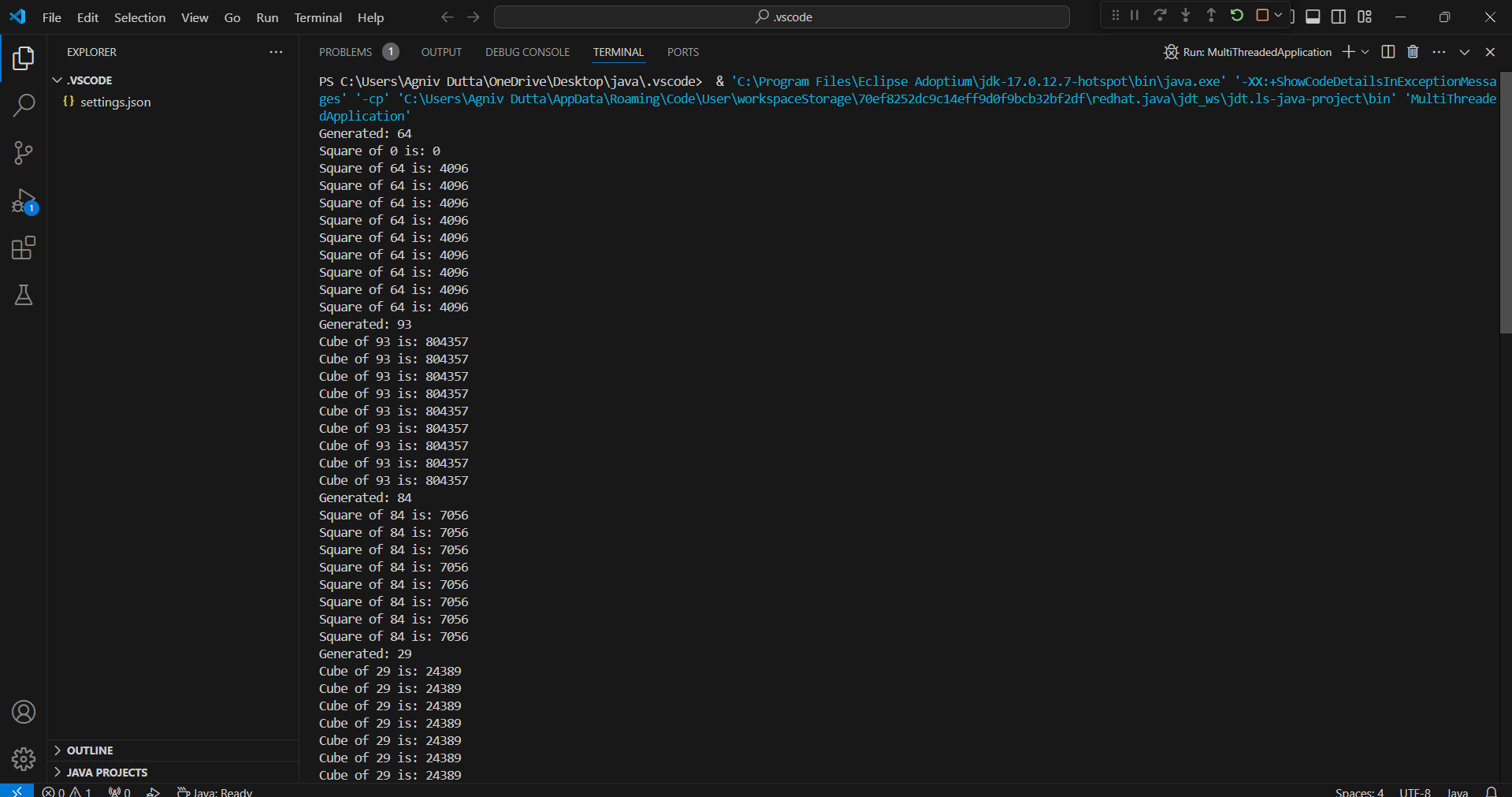
 **Synchronization**:

* The SharedData class uses synchronized methods to ensure that data access between threads is handled safely, preventing data inconsistency.

**Implementation details:**

import java.util.Random;  
  
class RandomNumberGenerator implements Runnable {  
    private final SharedData sharedData;  
  
    public RandomNumberGenerator(SharedData sharedData) {  
        this.sharedData = sharedData;  
    }  
  
    @Override  
    public void run() {  
        Random random = new Random();  
        while (true) {  
            int number = random.nextInt(100);   
            sharedData.setNumber(number);  
            System.out.println("Generated: " + number);  
            try {  
                Thread.sleep(1000);   
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
    }  
}  
  
class EvenSquareCalculator implements Runnable {  
    private final SharedData sharedData;  
  
    public EvenSquareCalculator(SharedData sharedData) {  
        this.sharedData = sharedData;  
    }  
  
    @Override  
    public void run() {  
        while (true) {  
            int number = sharedData.getNumber();  
            if (number % 2 == 0) {  
                int square = number \* number;  
                System.out.println("Square of " + number + " is: " + square);  
            }  
            try {  
                Thread.sleep(100);   
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
    }  
}  
  
class OddCubeCalculator implements Runnable {  
    private final SharedData sharedData;  
  
    public OddCubeCalculator(SharedData sharedData) {  
        this.sharedData = sharedData;  
    }  
  
    @Override  
    public void run() {  
        while (true) {  
            int number = sharedData.getNumber();  
            if (number % 2 != 0) {  
                int cube = number \* number \* number;  
                System.out.println("Cube of " + number + " is: " + cube);  
            }  
            try {  
                Thread.sleep(100);   
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
    }  
}  
  
class SharedData {  
    private int number;  
  
    public synchronized void setNumber(int number) {  
        this.number = number;  
    }  
  
    public synchronized int getNumber() {  
        return number;  
    }  
}  
  
public class MultiThreadedApplication {  
    public static void main(String[] args) {  
        SharedData sharedData = new SharedData();  
  
        Thread generatorThread = new Thread(new RandomNumberGenerator(sharedData));  
        Thread evenThread = new Thread(new EvenSquareCalculator(sharedData));  
        Thread oddThread = new Thread(new OddCubeCalculator(sharedData));  
  
        generatorThread.start();  
        evenThread.start();  
        oddThread.start();  
    }  
}

**Output:**



**Conclusion:**

This Java program demonstrates a multi-threaded application where one thread generates random integers every second, while two other threads process the generated numbers based on their parity. If the number is even, the second thread calculates and prints its square, while the third thread computes and prints its cube if the number is odd. Multi-threading enhances the program’s responsiveness and overall performance in real-time scenarios.

**Date:\_\_\_\_\_\_\_\_ 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.

**Thread Synchronization** is the process of controlling the access of multiple threads to shared resources to prevent inconsistent behavior or data corruption. When multiple threads operate on shared data, there's a risk of *race conditions*, where the output or the behavior of a program becomes dependent on the sequence of thread execution. To ensure that only one thread accesses the shared resource at a time, we use synchronization.

Without synchronization, when two or more threads try to modify shared data concurrently, it can lead to unexpected results because the threads may interrupt each other’s operations.

Thread synchronization is needed to:

1. **Prevent race conditions**: Ensure that multiple threads do not read/write shared data at the same time.
2. **Ensure thread safety**: Maintain consistency of shared data when accessed by multiple threads.
3. **Avoid data corruption**: Synchronization prevents the threads from interfering with each other during data manipulation.

class Counter {

private int count = 0;

// Synchronized increment method

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

}

class MyThread extends Thread {

private final Counter counter;

public MyThread(Counter counter) {

this.counter = counter;

}

@Override

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

}

public class WithSynchronizationExample {

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

Counter counter = new Counter();

// Creating two threads that share the same Counter object

Thread t1 = new MyThread(counter);

Thread t2 = new MyThread(counter);

t1.start();

t2.start();

t1.join();

t2.join();

// The expected output is now guaranteed to be 2000

System.out.println("Final Count: " + counter.getCount());

}

}

Explanation:

* The increment() method is marked as synchronized, which ensures that only one thread can execute this method at a time.
* As a result, the shared variable count is safely updated by one thread at a time, preventing race conditions.

### 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 double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public synchronized void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println(Thread.currentThread().getName() + " deposited: $" + amount + " | New Balance: $" + balance);

}

}

public synchronized void withdraw(double amount) {

if (amount > 0 && balance >= amount) {

balance -= amount;

System.out.println(Thread.currentThread().getName() + " withdrew: $" + amount + " | New Balance: $" + balance);

} else {

System.out.println(Thread.currentThread().getName() + " attempted to withdraw: $" + amount + " | Insufficient Funds!");

}

}

public synchronized double getBalance() {

return balance;

}

}

class Transaction implements Runnable {

private final BankAccount account;

private final boolean isDeposit;

private final double amount;

public Transaction(BankAccount account, boolean isDeposit, double amount) {

this.account = account;

this.isDeposit = isDeposit;

this.amount = amount;

}

@Override

public void run() {

if (isDeposit) {

account.deposit(amount);

} else {

account.withdraw(amount);

}

}

}

public class MultiThreadedBankAccount {

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

BankAccount account = new BankAccount(1000.00);

Thread t1 = new Thread(new Transaction(account, true, 500), "Thread 1");

Thread t2 = new Thread(new Transaction(account, false, 700), "Thread 2");

Thread t3 = new Thread(new Transaction(account, true, 300), "Thread 3");

Thread t4 = new Thread(new Transaction(account, false, 900), "Thread 4");

t1.start();

t2.start();

t3.start();

t4.start();

t1.join();

t2.join();

t3.join();

t4.join();

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

}

}

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

Here are five commonly used methods of the Thread class, along with examples:

1. **start()**
   1. Begins the execution of the thread. The Java Virtual Machine (JVM) calls the run() method of this thread.

Thread t1 = new Thread(() -> {

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

});

t1.start();

**run()**

This method contains the code that constitutes the new thread. It can be overridden to define the thread's task.

class MyThread extends Thread {

@Override

public void run() {

System.out.println("Thread is running in run() method.");

}

}

MyThread thread = new MyThread();

thread.start();

1. **sleep(long millis)**

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

try {

System.out.println("Thread is going to sleep.");

Thread.sleep(2000); // Sleeps for 2 seconds

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread woke up.");

1. **join()**

Waits for the thread to die. If another thread calls join() on this thread, it will block until this thread completes execution.

Thread t2 = new Thread(() -> {

for (int i = 0; i < 5; i++) {

System.out.println("Running in t2: " + i);

}

});

t2.start();

try {

t2.join(); // Main thread waits for t2 to finish

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("t2 has finished execution.");

1. **yield()**

A static method that suggests to the thread scheduler that the current thread is willing to yield its current use of the CPU. It does not guarantee that the thread will be suspended; it just indicates that the thread is willing to let others run.

Thread t3 = new Thread(() -> {

for (int i = 0; i < 5; i++) {

System.out.println("Running in t3: " + i);

Thread.yield(); // Yield the CPU to other threads

}

});

t3.start();

