| | |
|---|---|
| **Batch: A1** | **Roll No.: 16010123012** |
| **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:**

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:

You can implement the Runnable interface.

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:**

1. Initialize Shared Data

Create an instance of SharedData to hold the number

2. Create and Start Threads

Create 3 threads: Thread1, Thread2, and Thread3, and pass the SharedData object to them

Start all threads

3. Thread 1 (Generates Random Number)

Continuously generate a random number between 0 and 99

Set the generated number in the shared data

Print the generated number

Wait for 1 second before generating the next number

4. Thread 2 (Calculates Square of Even Numbers)

Continuously check the current number in shared data

If the number is even, calculate and print its square

Wait for 100 milliseconds before checking again

5. Thread 3 (Calculates Cube of Odd Numbers)

Continuously check the current number in shared data

If the number is odd, calculate and print its cube

Wait for 100 milliseconds before checking again

6. Synchronization

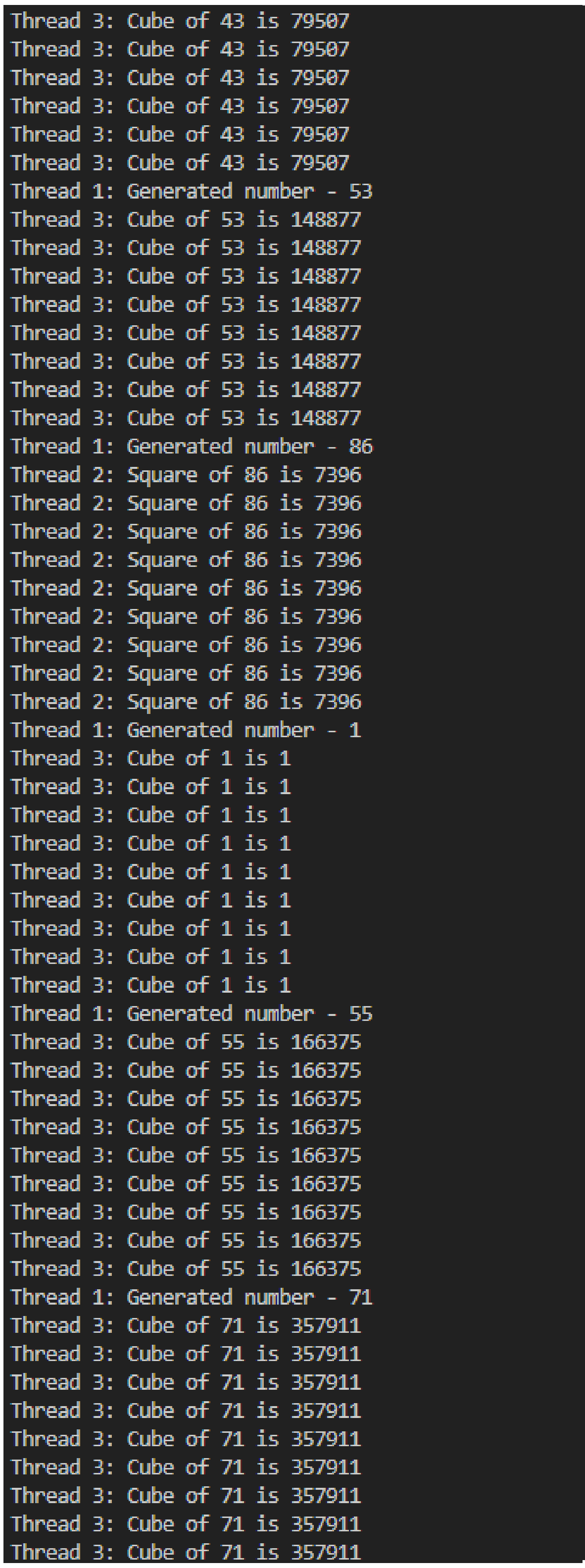
Use synchronization to ensure that threads safely access and modify the shared number

## Implementation details:

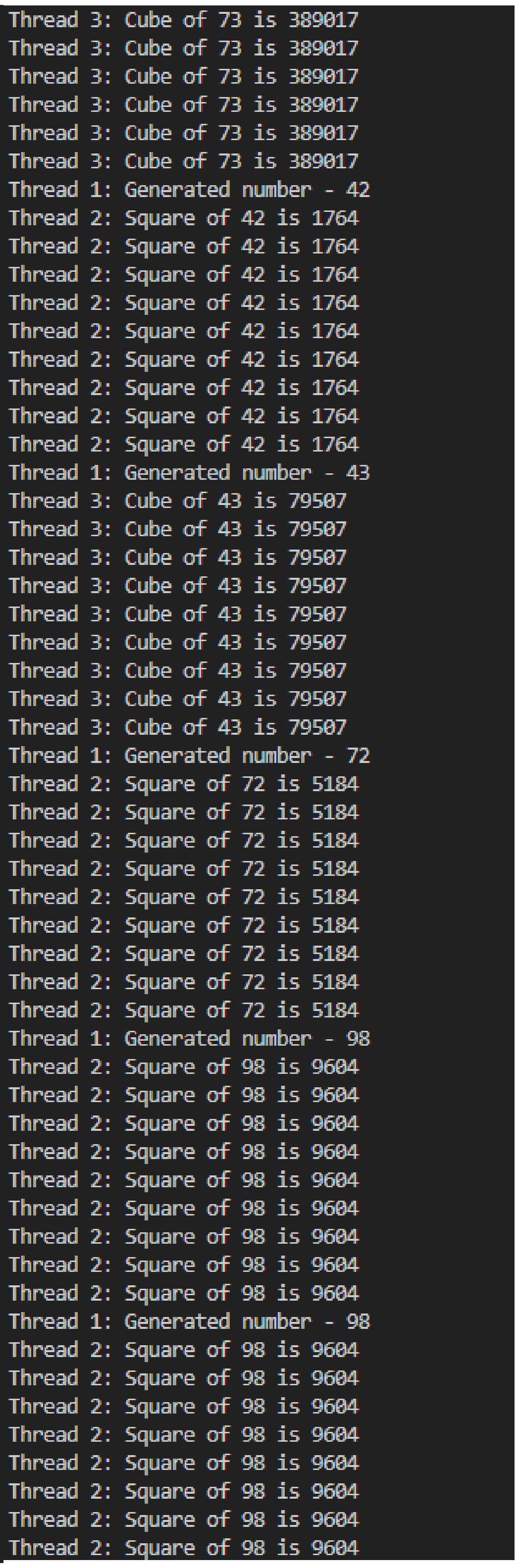
```
import java.util.Random;

class Thread1 extends Thread {
    private SharedData sharedData;

    public Thread1(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("Thread 1: Generated number - " + number);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class Thread2 extends Thread {
    private SharedData sharedData;

    public Thread2(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        while (true) {
            if (sharedData.getNumber() % 2 == 0) {
                int square = sharedData.getNumber() *
sharedData.getNumber();
                System.out.println("Thread 2: Square of " +
sharedData.getNumber() + " is " + square);
            }
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class Thread3 extends Thread {
    private SharedData sharedData;

    public Thread3(SharedData sharedData) {
        this.sharedData = sharedData;
    }
```

```java
    @Override
    public void run() {
        while (true) {
            if (sharedData.getNumber() % 2 != 0) {
                int cube = sharedData.getNumber() *
sharedData.getNumber() * sharedData.getNumber();
                System.out.println("Thread 3: Cube of " +
sharedData.getNumber() + " is " + cube);
            }
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class SharedData {
    private int number;

    public synchronized int getNumber() {
        return number;
    }

    public synchronized void setNumber(int number) {
        this.number = number;
        notifyAll();
    }
}

public class Exp8 {
    public static void main(String[] args) {
        SharedData sharedData = new SharedData();
        Thread1 thread1 = new Thread1(sharedData);
        Thread2 thread2 = new Thread2(sharedData);
        Thread3 thread3 = new Thread3(sharedData);

        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

**Output:**



```
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 1: Generated number - 53
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 3: Cube of 53 is 148877
Thread 1: Generated number - 86
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 2: Square of 86 is 7396
Thread 1: Generated number - 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 3: Cube of 1 is 1
Thread 1: Generated number - 55
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 3: Cube of 55 is 166375
Thread 1: Generated number - 71
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
Thread 3: Cube of 71 is 357911
```



```
Thread 3: Cube of 73 is 389017
Thread 3: Cube of 73 is 389017
Thread 3: Cube of 73 is 389017
Thread 3: Cube of 73 is 389017
Thread 3: Cube of 73 is 389017
Thread 3: Cube of 73 is 389017
Thread 1: Generated number - 42
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 2: Square of 42 is 1764
Thread 1: Generated number - 43
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 3: Cube of 43 is 79507
Thread 1: Generated number - 72
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 2: Square of 72 is 5184
Thread 1: Generated number - 98
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 1: Generated number - 98
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
Thread 2: Square of 98 is 9604
```

**Conclusion:**

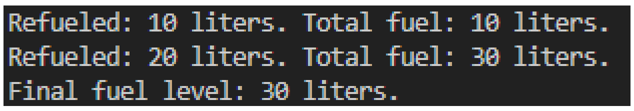
We learned about multithreading in java and how can it be implemented in our code.

**Date: 05 / 11 / 2024** **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 a way to make sure that only one thread at a time can access shared resources, like a variable or a piece of code, to avoid mistakes or inconsistent results. When multiple threads try to use the same data at the same time, it can cause errors or wrong results. Synchronization makes sure only one thread can use the data at a time.

```java
class Car {
    private int fuel = 0;

    public synchronized void refuel(int amount) {
        fuel += amount;
        System.out.println("Refueled: " + amount + " liters. Total fuel: 
" + fuel + " liters.");
    }

    public int getFuel() {
        return fuel;
    }
}

public class PostLab1 {
    public static void main(String[] args) throws InterruptedException {
        Car myCar = new Car();
        Thread t1 = new Thread(() -> myCar.refuel(10 ));
        Thread t2 = new Thread(() -> myCar.refuel(20));
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Final fuel level: " + myCar.getFuel() + " 
liters.");
    }
}
```



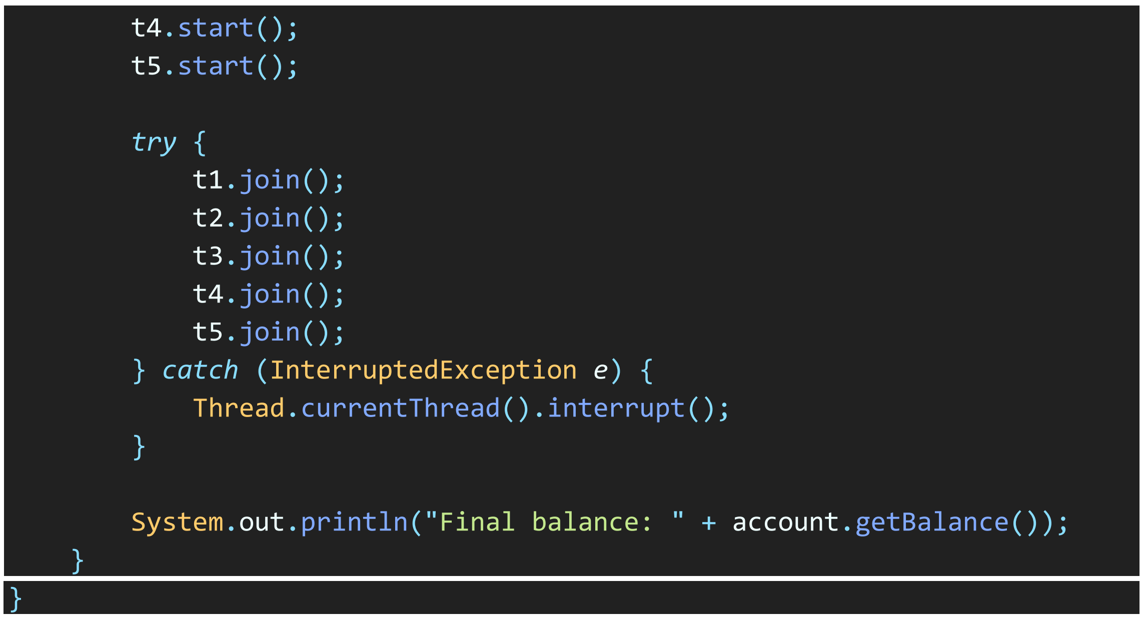
```
Refueled: 10 liters. Total fuel: 10 liters.
Refueled: 20 liters. Total fuel: 30 liters.
Final fuel level: 30 liters.
```

2. Write a program for multithreaded Bank Account System

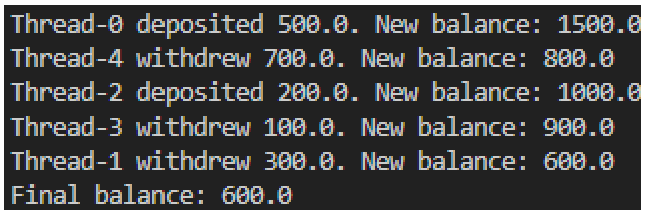
Implement a multithreaded bank account system in Java such that the system should simulate transactions on a bank account that can be accessed and modified by multiple threads concurrently. Your goal is to ensure that all transactions are handled correctly and that the account balance remains consistent.

```java
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);
        } else {
            System.out.println("Deposit amount must be greater than
zero.");
        }
    }

    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("Insufficient funds or invalid withdrawal
amount.");
        }
    }

    public double getBalance() {
        return balance;
    }
```

```java
}

class DepositThread extends Thread {
    private BankAccount account;
    private double amount;

    public DepositThread(BankAccount account, double amount) {
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run() {
        account.deposit(amount);
    }
}

class WithdrawThread extends Thread {
    private BankAccount account;
    private double amount;

    public WithdrawThread(BankAccount account, double amount) {
        this.account = account;
        this.amount = amount;
    }

    @Override
    public void run() {
        account.withdraw(amount);
    }
}

public class BankTransactionSystem {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
        Thread t1 = new DepositThread(account, 500);
        Thread t2 = new WithdrawThread(account, 300);
        Thread t3 = new DepositThread(account, 200);
        Thread t4 = new WithdrawThread(account, 100);
        Thread t5 = new WithdrawThread(account, 700);

        t1.start();
        t2.start();
        t3.start();
```



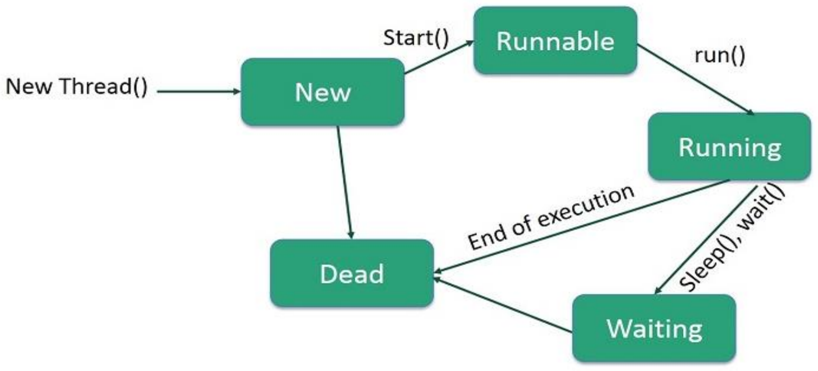
```
        t4.start();
        t5.start();

        try {
            t1.join();
            t2.join();
            t3.join();
            t4.join();
            t5.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

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



```
Thread-0 deposited 500.0. New balance: 1500.0
Thread-4 withdrew 700.0. New balance: 800.0
Thread-2 deposited 200.0. New balance: 1000.0
Thread-3 withdrew 100.0. New balance: 900.0
Thread-1 withdrew 300.0. New balance: 600.0
Final balance: 600.0
```

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

- start(): Starts a new thread and moves it to the "Runnable" state.
- sleep(long millis): Pauses the thread for a specified time, putting it in the Timed Waiting state.
- join(): Makes the current thread wait for the specified thread to finish.
- interrupt(): Interrupts the thread, potentially causing it to stop if it's in a blocked state.
- isAlive(): Checks if the thread is still running (alive).


New Thread()
New
Start()
Runnable
run()
Running
End of execution
Sleep(), wait()
Dead
Waiting