Batch: B2	**Roll No.: 16010124107**
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.

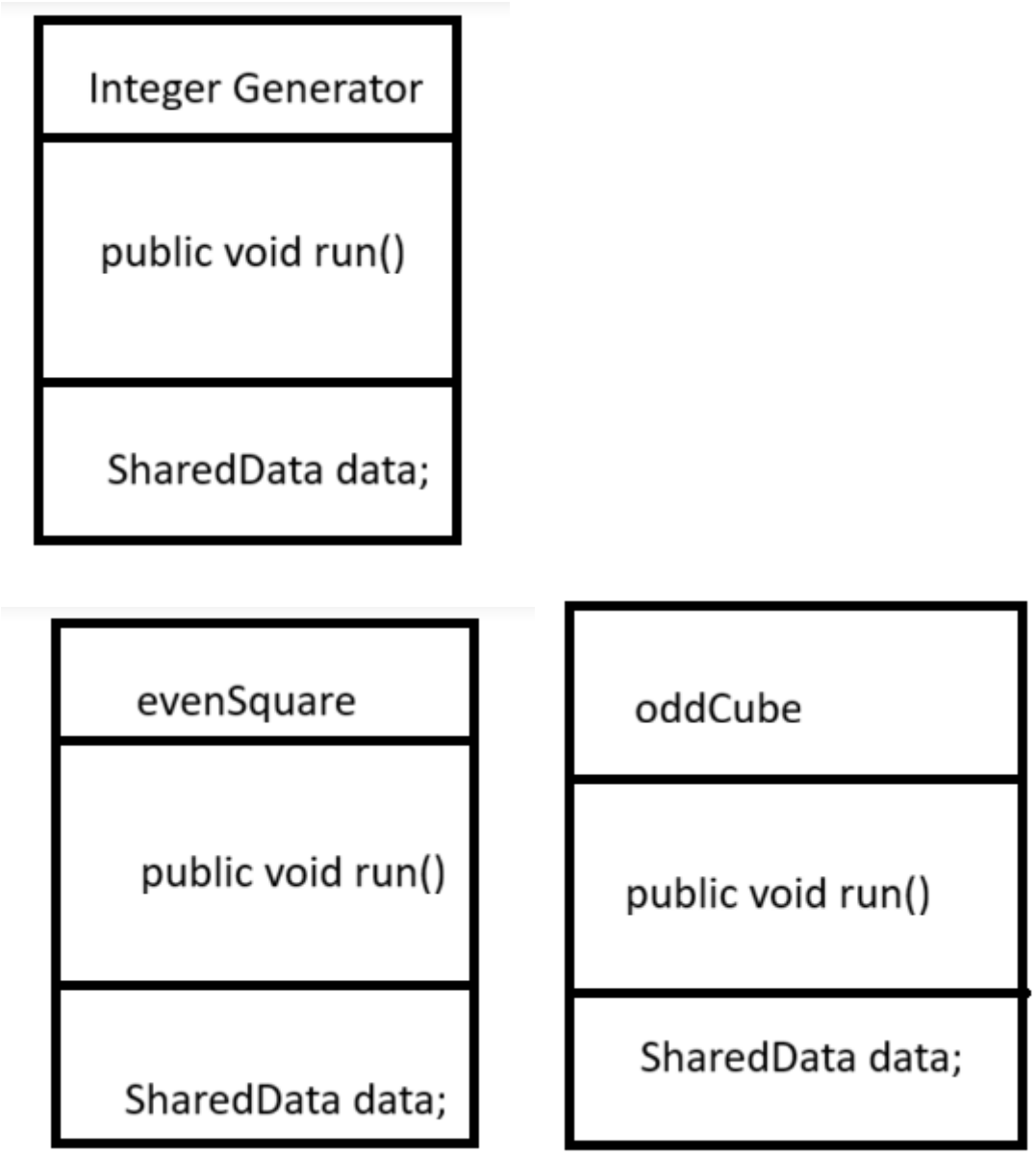
```
class NewThread extends Thread {
  NewThread() {
    super("Demo Thread");
    System.out.println("Child thread: " + this);
    start(); // Start the thread
  }
   public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println("Child Thread: " + i);
                // Let the thread sleep for a while.
        Thread.sleep(50);
      }
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
```

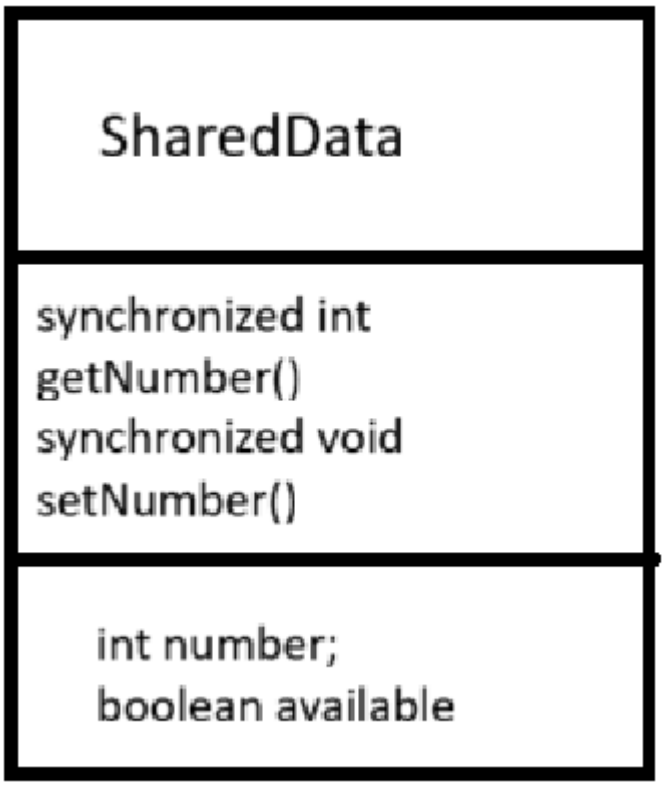
```
  }
}

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.");
  }
}
```

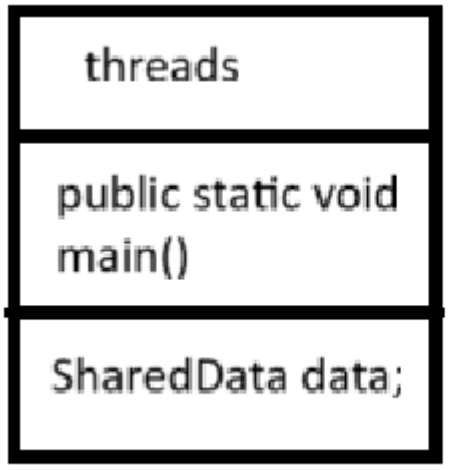
Some of the Thread methods

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:


Integer Generator
public void run()
SharedData data;
evenSquare
public void run()
SharedData data;
oddCube
public void run()
SharedData data;





Algorithm:

1. Start
2. Create an IntegerGenerator class that extends Thread
3. Initialize data members and constructor

4. Create a function run that generates random numbers from 0 to 100. It also handles exception with a try-catch block
5. Create another class evenSquare that also extends thread.
6. Repeat step 3.
7. Create a function void run that returns the square of the number if the number is even.
8. Create another class oddCube that also extends thread
9. Repeat step 3
10. Create function void run that returns the cube of the number if the number is odd
11. Add try-catch blocks in each class for exception handling
12. create class SharedData and initialise its data members. Also set flag as false and initialize the constructors.
13. create public synchronized void setNumber and getNumber that actually sends value from generator to oddcube and evensquare.
14. finally create a class threads that ties everything together under main() function.
15. stop

Implementation details:

```
import java.util.*;


class IntegerGenerator extends Thread{

    SharedData data;


    IntegerGenerator(SharedData data){

        this.data = data;

    }


    public void run(){

        Random rand = new Random();

        while(true){
```

```
            int num = rand.nextInt(100);
            data.setNumber(num);
            try{
                Thread.sleep(100);
            }catch(InterruptedException e){
                System.out.println(e);
            }
        }
    }
}

class evenSquare extends Thread{
    SharedData data;

    evenSquare(SharedData data){
        this.data = data;
    }

    public void run(){
        while(true){
            int num = data.getNumber();
            if(num % 2 == 0){
                  System.out.println("Square of " + num + " is " + (num *
num));
```

```
            }
            try{
                Thread.sleep(100);
            }catch(InterruptedException e){
                System.out.println(e);
            }
        }
    }
}

class oddCube extends Thread{
    SharedData data;

    oddCube(SharedData data){
        this.data = data;
    }

    public void run(){
        while(true){
            int num = data.getNumber();
            if(num % 2 != 0){
                System.out.println("Cube of " + num + " is " + (num * num
* num));
            }
```

```
            try{
                Thread.sleep(100);
            }catch(InterruptedException e){
                System.out.println(e);
            }
        }
    }
}

class SharedData{
    int number;
    boolean available = false;


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

    synchronized int getNumber() {
        while (!available) {
            try {
                wait();
```

```
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
        available = false;
        return number;
    }
}

class threads{
    public static void main(String[] args) {
        SharedData data = new SharedData();
        new IntegerGenerator(data).start();
        new evenSquare(data).start();
        new oddCube(data).start();
    }
}
```

Output:

```
PS C:\Users\syeda\OneDr
Cube of 27 is 19683
Square of 40 is 1600
Cube of 61 is 226981
Cube of 49 is 117649
Square of 76 is 5776
Square of 38 is 1444
Square of 58 is 3364
Square of 72 is 5184
Square of 38 is 1444
Cube of 71 is 357911
Square of 30 is 900
Cube of 59 is 205379
Square of 88 is 7744
Cube of 73 is 389017
Square of 42 is 1764
Cube of 25 is 15625
Square of 26 is 676
Cube of 23 is 12167
Cube of 5 is 125
Cube of 57 is 185193
Square of 46 is 2116
```

Conclusion:

This program demonstrates multithreading in Java using OOP concepts. Three threads run concurrently — one generates random integers, another computes the square of even numbers, and the third computes the cube of odd numbers. Using synchronization (wait() and notifyAll()), the threads coordinate properly, ensuring each number is processed exactly once. It effectively shows thread communication, synchronization, and parallel execution in Java.

Date: 13/10/2025 **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 in Java means controlling the access of multiple threads to a shared resource or data. It ensures that only one thread can access the critical section of code at a time.

- When multiple threads access shared data simultaneously, they may cause data inconsistency or race conditions.
- Synchronization prevents this by making threads execute one after another in the critical section.

Example code:

```
class Table {
   synchronized void printTable(int n) {
      for (int i = 1; i <= 5; i++) {
         System.out.println(n * i);
         try {
            Thread.sleep(500);
         } catch (InterruptedException e) {
            System.out.println(e);
         }
      }
   }
}

class Thread1 extends Thread {
   Table t;
   Thread1(Table t) {
      this.t = t;
   }
   public void run() {
      t.printTable(5);
   }
}

class Thread2 extends Thread {
   Table t;
   Thread2(Table t) {
      this.t = t;
   }
   public void run() {
```

```
        t.printTable(10);
    }
}

public class SyncDemo {
    public static void main(String[] args) {
        Table obj = new Table();
        Thread1 t1 = new Thread1(obj);
        Thread2 t2 = new Thread2(obj);
        t1.start();
        t2.start();
    }
}
```

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.

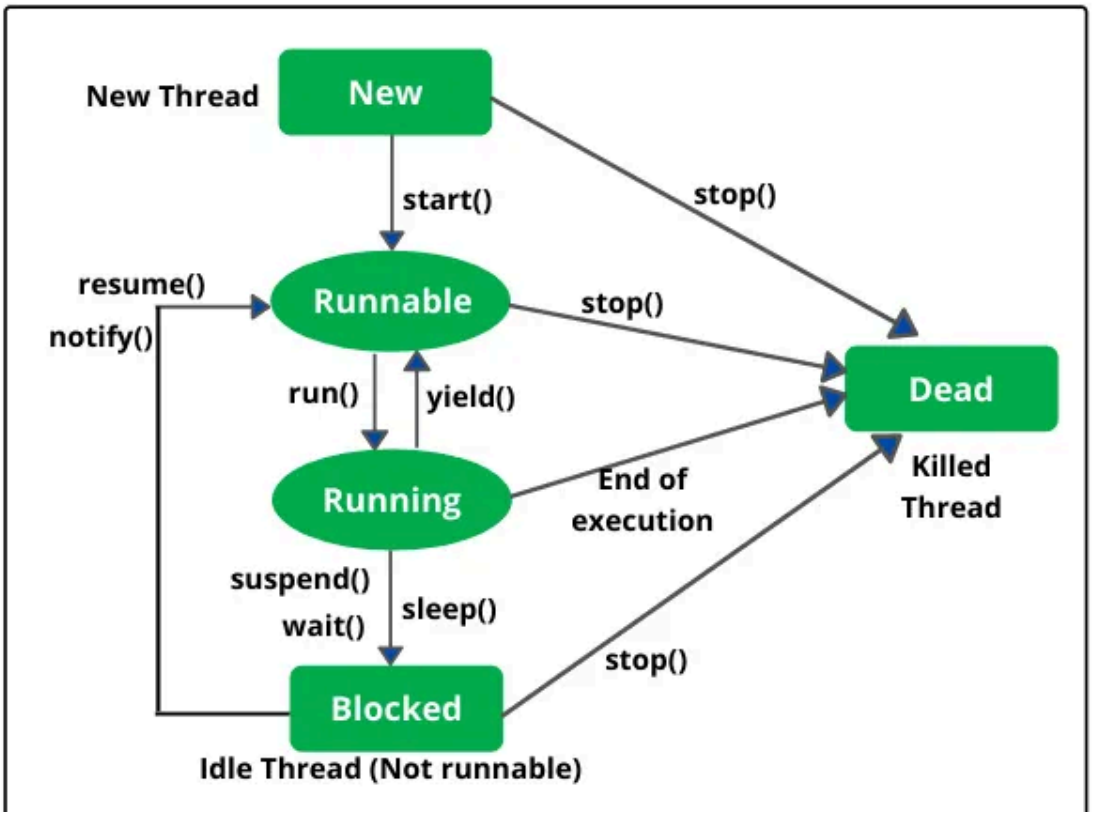
```
import java.util.*;

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("Deposited: " + amount + ", New Balance: "
+ balance);
        }
    }

    public synchronized void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
```

```
            balance -= amount;
            System.out.println("Withdrew: " + amount + ", New Balance: "
+ balance);
        } else {
            System.out.println("Withdrawal of " + amount + " failed.
Insufficient funds.");
        }
    }

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

class Transaction implements Runnable {
    private BankAccount account;
    private String type;
    private double amount;

    public Transaction(BankAccount account, String type, double amount) {
        this.account = account;
        this.type = type;
        this.amount = amount;
    }

    @Override
    public void run() {
        if (type.equals("deposit")) {
            account.deposit(amount);
        } else if (type.equals("withdraw")) {
            account.withdraw(amount);
        }
    }
}

public class threads {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);
```

```
        List<Thread> threads = new ArrayList<>();
        threads.add(new Thread(new Transaction(account, "deposit",
200)));
        threads.add(new Thread(new Transaction(account, "withdraw",
150)));
        threads.add(new Thread(new Transaction(account, "withdraw",
300)));
        threads.add(new Thread(new Transaction(account, "deposit",
400)));
        threads.add(new Thread(new Transaction(account, "withdraw",
500)));

        for (Thread thread : threads) {
            thread.start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

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

```
PS C:\Users\syeda\OneDrive\Desktop\personal> java threads
Deposited: 200.0, New Balance: 1200.0
Withdrew: 500.0, New Balance: 700.0
Deposited: 400.0, New Balance: 1100.0
Withdrew: 300.0, New Balance: 800.0
Withdrew: 150.0, New Balance: 650.0
Final Balance: 650.0
PS C:\Users\syeda\OneDrive\Desktop\personal>
```

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



- start() – Starts the thread and calls run()

```
class A extends Thread{public void run(){
System.out.println("Inside run");}
}
```

- run() – Contains the code executed by the thread.

```
class A extends Thread{public void run(){
try{
Thread.sleep(500);}
catch(Exception e){}}
}
```

- sleep(ms) – Pauses thread for given milliseconds.

```
class A extends Thread{public void run(){
try{
Thread.sleep(500);}
catch(Exception e){}
}}
```

- setName() / getName() – Sets or gets thread name.

```
class A extends Thread{public void run(){
try{
Thread.sleep(500);}
catch(Exception e){}
}}
```

- join() – Waits for a thread to finish.

```
class A extends Thread{public void run(){System.out.println("Thread done");}}

class Main{public static void main(String[]a)throws Exception{A t=new
A();t.start();t.join();System.out.println("Main ends");}}
```