Lab 8_ 9-10-2025

1. Write a Java program that creates two threads — one to print even numbers and another to print odd numbers up to 20. Use proper synchronization to ensure numbers are printed in sequence.

Code:-

```
class NumberPrinter {
    int number = 1;
    final int MAX = 20;
}

class EvenThread extends Thread {
    NumberPrinter np;

    public EvenThread(NumberPrinter np) {
        this.np = np;
    }

    public void run() {
        while (np.number <= np.MAX) {
            synchronized (np) {
                if (np.number % 2 == 0) {
                    System.out.print(np.number + " ");
                    np.number++;
                }
            }
        }
    }
}

class OddThread extends Thread {
    NumberPrinter np;

    public OddThread(NumberPrinter np) {
        this.np = np;
    }

    public void run() {
        while (np.number <= np.MAX) {
            synchronized (np) {
                if (np.number % 2 != 0) {
                    System.out.print(np.number + " ");
                    np.number++;
```

```
            }
        }
    }
  }
}

public class ThreadsEven {
    public static void main(String[] args) {
        NumberPrinter np = new NumberPrinter();

        Thread t1 = new EvenThread(np);
        Thread t2 = new OddThread(np);

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

```
C:\Users\Lenovo\Desktop\Java lab\Lab 8>javac ThreadsEven.java

C:\Users\Lenovo\Desktop\Java lab\Lab 8>java ThreadsEven
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
C:\Users\Lenovo\Desktop\Java lab\Lab 8>
```

2. Write a Java program that simulates downloading multiple files simultaneously using threads. Each thread should print progress from 0% to 100%.

Code:-

```
import java.io.*;

class FileDownload extends Thread {
    private String fileName;

    public FileDownload(String fileName) {
        this.fileName = fileName;
    }

    public void run() {
        for (int i = 0; i <= 100; i += 10) {
            System.out.println(fileName + " downloaded: " + i + "%");
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
        }
        System.out.println(fileName + " download complete!");
    }
}

public class ThreadsDownload {
    public static void main(String[] args) {
        Thread t1 = new FileDownload("File1");
        Thread t2 = new FileDownload("File2");
        Thread t3 = new FileDownload("File3");

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

```
C:\Users\Lenovo\Desktop\Java lab\Lab 8>java ThreadsDownload
File2 downloaded: 0%
File3 downloaded: 0%
File1 downloaded: 0%
File1 downloaded: 10%
File2 downloaded: 10%
File3 downloaded: 10%
File3 downloaded: 20%
File1 downloaded: 20%
File2 downloaded: 20%
File2 downloaded: 30%
File3 downloaded: 30%
File1 downloaded: 30%
File1 downloaded: 40%
File2 downloaded: 40%
File3 downloaded: 40%
File3 downloaded: 50%
File1 downloaded: 50%
File2 downloaded: 50%
File2 downloaded: 60%
File3 downloaded: 60%
File1 downloaded: 60%
File2 downloaded: 70%
File1 downloaded: 70%
File3 downloaded: 70%
File1 downloaded: 80%
File2 downloaded: 80%
File3 downloaded: 80%
File1 downloaded: 90%
File3 downloaded: 90%
File2 downloaded: 90%
File3 downloaded: 100%
File1 downloaded: 100%
File2 downloaded: 100%
File2 download complete!
File3 download complete!
File1 download complete!
```

3. Create three threads with different priorities (MIN_PRIORITY, NORM_PRIORITY, MAX_PRIORITY). Observe and explain the effect of thread priorities on execution order.

Code:-

```
class PriorityDemo extends Thread {
    private String name;

    public PriorityDemo(String name) {
        this.name = name;
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(name + " running, iteration: " + i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ThreadsPriority {
    public static void main(String[] args) {
        Thread t1 = new PriorityDemo("Thread-MIN");
        Thread t2 = new PriorityDemo("Thread-NORM");
        Thread t3 = new PriorityDemo("Thread-MAX");

        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.NORM_PRIORITY);
        t3.setPriority(Thread.MAX_PRIORITY);

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

}

```
C:\Users\Lenovo\Desktop\Java lab\Lab 8>javac ThreadsPriority.java

C:\Users\Lenovo\Desktop\Java lab\Lab 8>java ThreadsPriority
Thread-NORM running, iteration: 0
Thread-MAX running, iteration: 0
Thread-MIN running, iteration: 0
Thread-MAX running, iteration: 1
Thread-MIN running, iteration: 1
Thread-NORM running, iteration: 1
Thread-MAX running, iteration: 2
Thread-MIN running, iteration: 2
Thread-NORM running, iteration: 2
Thread-MAX running, iteration: 3
Thread-MIN running, iteration: 3
Thread-NORM running, iteration: 3
Thread-MAX running, iteration: 4
Thread-NORM running, iteration: 4
Thread-MIN running, iteration: 4
```

4. Develop a multithreaded Java program where multiple threads represent customers depositing and withdrawing money from a shared bank account. Use synchronization to prevent race conditions.

Code:-

```
class BankAccount {
    private int balance = 1000;

    public synchronized void deposit(int amount) {
        balance += amount;
        System.out.println(Thread.currentThread().getName() + " deposited " + amount + ",
balance: " + balance);
    }

    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ",
balance: " + balance);
        } else {
            System.out.println(Thread.currentThread().getName() + " failed to withdraw " + amount +
", balance: " + balance);
        }
    }

    public synchronized int getBalance() {
```

```
        return balance;
    }
}

class Customer extends Thread {
    private BankAccount account;

    public Customer(BankAccount account, String name) {
        super(name);
        this.account = account;
    }

    public void run() {
        for (int i = 0; i < 3; i++) {
            account.deposit(600);
            account.withdraw(800);
        }
    }
}

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

        Thread c1 = new Customer(account, "Customer-1");
        Thread c2 = new Customer(account, "Customer-2");

        c1.start();
        c2.start();
    }
}
```

```
C:\Users\Lenovo\Desktop\Java lab\Lab 8>javac ThreadsBank.java

C:\Users\Lenovo\Desktop\Java lab\Lab 8>java ThreadsBank
Customer-1 deposited 600, balance: 1600
Customer-1 withdrew 800, balance: 800
Customer-1 deposited 600, balance: 1400
Customer-1 withdrew 800, balance: 600
Customer-1 deposited 600, balance: 1200
Customer-1 withdrew 800, balance: 400
Customer-2 deposited 600, balance: 1000
Customer-2 withdrew 800, balance: 200
Customer-2 deposited 600, balance: 800
Customer-2 withdrew 800, balance: 0
Customer-2 deposited 600, balance: 600
Customer-2 failed to withdraw 800, balance: 600
```