Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Experiment No 5

60009210105 Amitesh Sawarkar

D 12

Aim: - Implement Exception Handling and Multithreading in Java
Theory: -

1. **Exception in Java**

An exception is an “unwanted or unexpected event”, which occurs during the execution of the program i.e., at run-time, that disrupts the normal flow of the program’s instructions. When an exception occurs, the execution of the program gets terminated.

Why does an Exception occur?
An exception can occur due to several reasons like a Network connection problem, Bad input provided by a user, Opening a non-existing file in your program, etc.

Try, Catch, Throw, Throws and Finally
try: The try block contains a set of statements where an exception can occur.

```
try

{

    // statement(s) that might cause exception

}
```

catch: The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.

```
catch

{

   // statement(s) that handle an exception

   // examples, closing a connection, closing

   // file, exiting the process after writing

   // details to a log file.

}
```

throw: The throw keyword is used to transfer control from the try block to the catch block.

throws: The throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

finally:

The finally block performs the clean-up operations such closing the database connections, closing the files and resources which is opened.
The finally clause would always be executed irrespective of whether the exceptions happened or not and whether exception is handled or not.
This block will not get executed in a certain situation such as when the system got hung or

the program crashed due to some fatal error or exiting the JVM using System. Exit ()

The finally block also cannot exist separately, it has to be associated with a try block. Valid scenarios would be try – finally and try – catch – finally.

throw and throws in Java

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword.

The throws are used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly.

Catching Multiple Exceptions

There can be a situation where a particular code has the tendency to throw multiple exceptions. You want to catch each of the exceptions that may occur within a try block. There are two ways to handle this scenario

2. Multithreading in Java

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

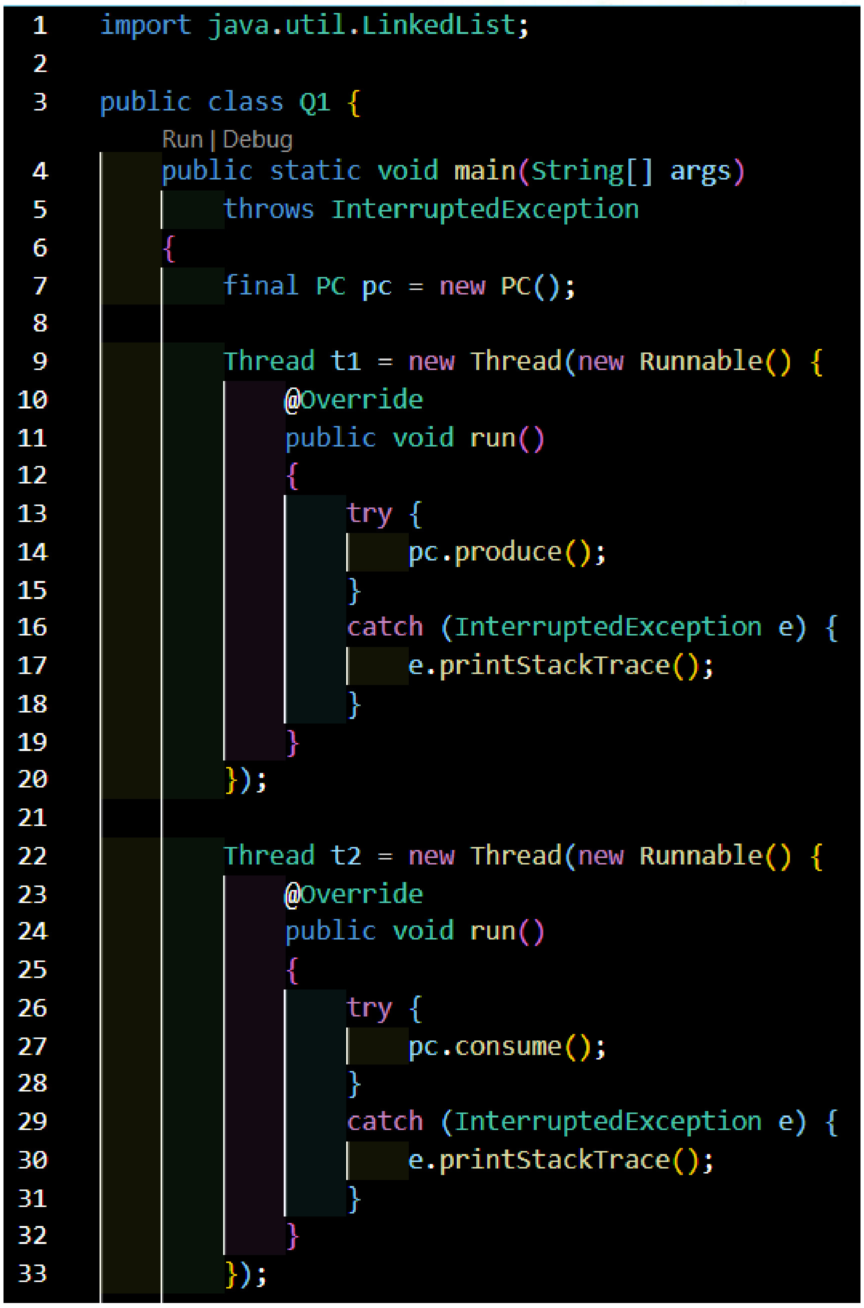
Thread creation by extending the Thread class
We create a class that extends the **java.lang.Thread** class. This class overrides the run () method available in the Thread class. A thread begins its life inside run () method. We create an object of our new class and call start () method to start the execution of a thread. Start () invokes the run () method on the Thread object.

Thread Class vs Runnable Interface

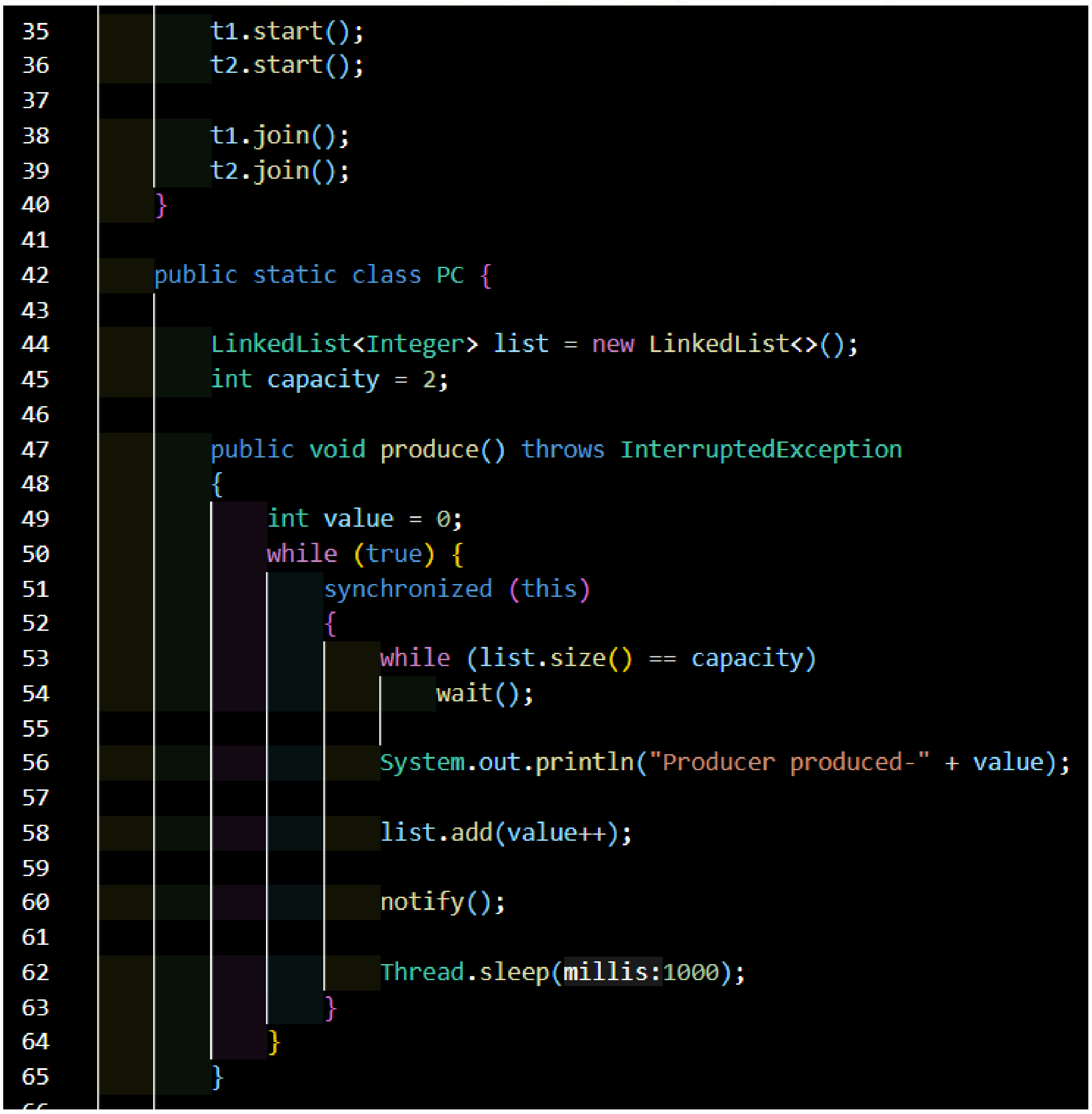
1. If we extend the Thread class, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield (), interrupt () etc. that are not available in Runnable interface.
3. Using runnable will give you an object that can be shared amongst multiple threads.

3. Lab Assignments to complete in this session

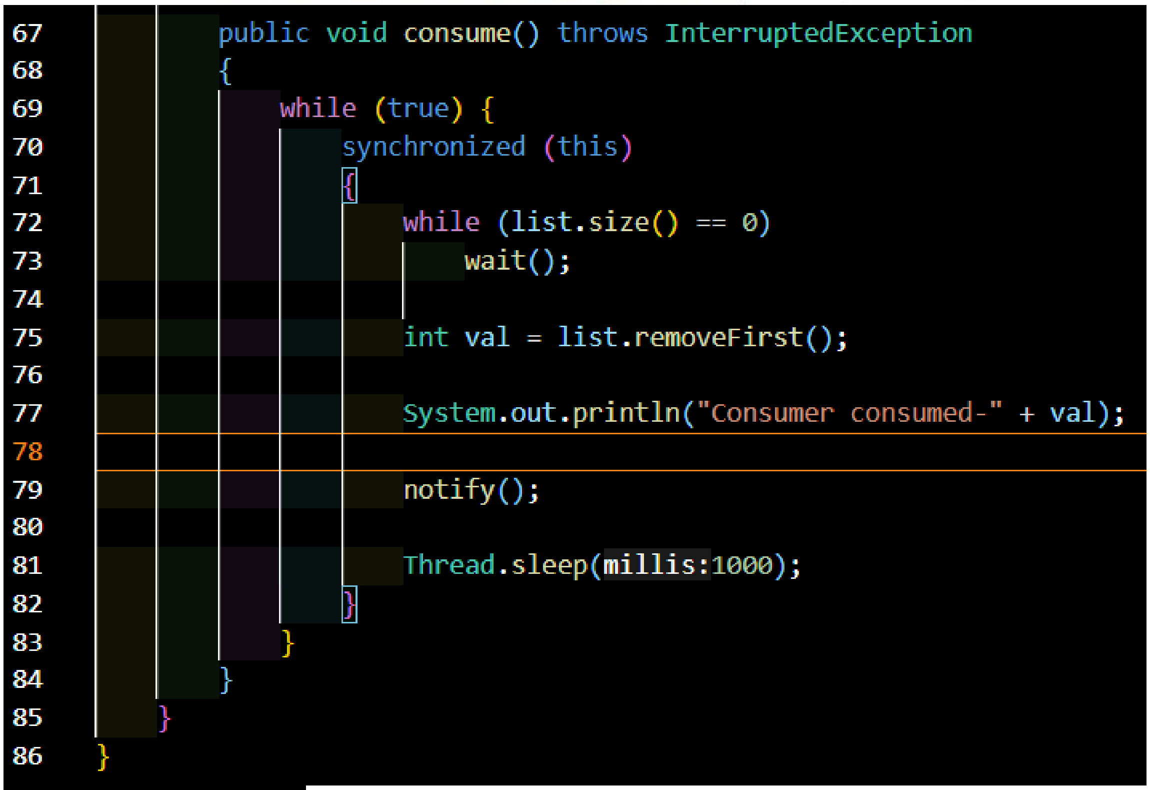
i. Write A Program for producer consumer problem in java



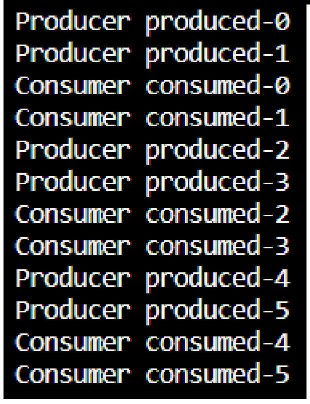
```
import java.util.LinkedList;

public class Q1 {
    Run | Debug
    public static void main(String[] args)
        throws InterruptedException
    {
        final PC pc = new PC();

        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.produce();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run()
            {
                try {
                    pc.consume();
                }
                catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
```



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

        t1.join();
        t2.join();
    }

    public static class PC {

        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;

        public void produce() throws InterruptedException
        {
            int value = 0;
            while (true) {
                synchronized (this)
                {
                    while (list.size() == capacity)
                        wait();

                    System.out.println("Producer produced-" + value);

                    list.add(value++);

                    notify();

                    Thread.sleep(millis:1000);
                }
            }
        }
```

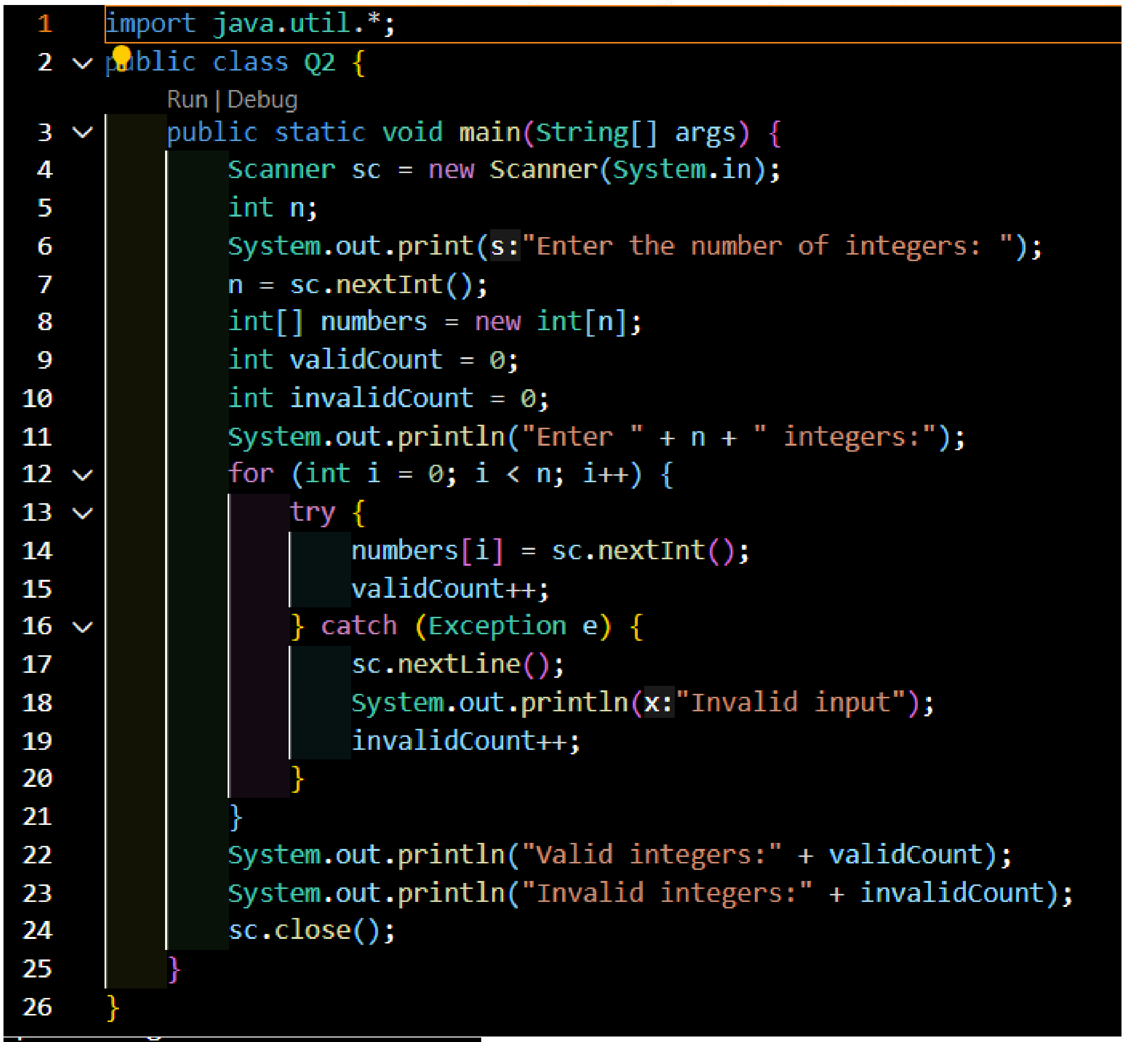


```
        public void consume() throws InterruptedException
        {
            while (true) {
                synchronized (this)
                {
                    while (list.size() == 0)
                        wait();

                    int val = list.removeFirst();

                    System.out.println("Consumer consumed-" + val);

                    notify();

                    Thread.sleep(millis:1000);
                }
            }
        }
    }
}
```

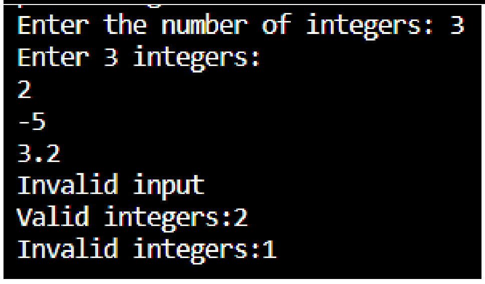


```
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
```

ii. Write a Java Program to input the data through command Line and Find out total valid and in-valid integers. (Hint: use exception handling)

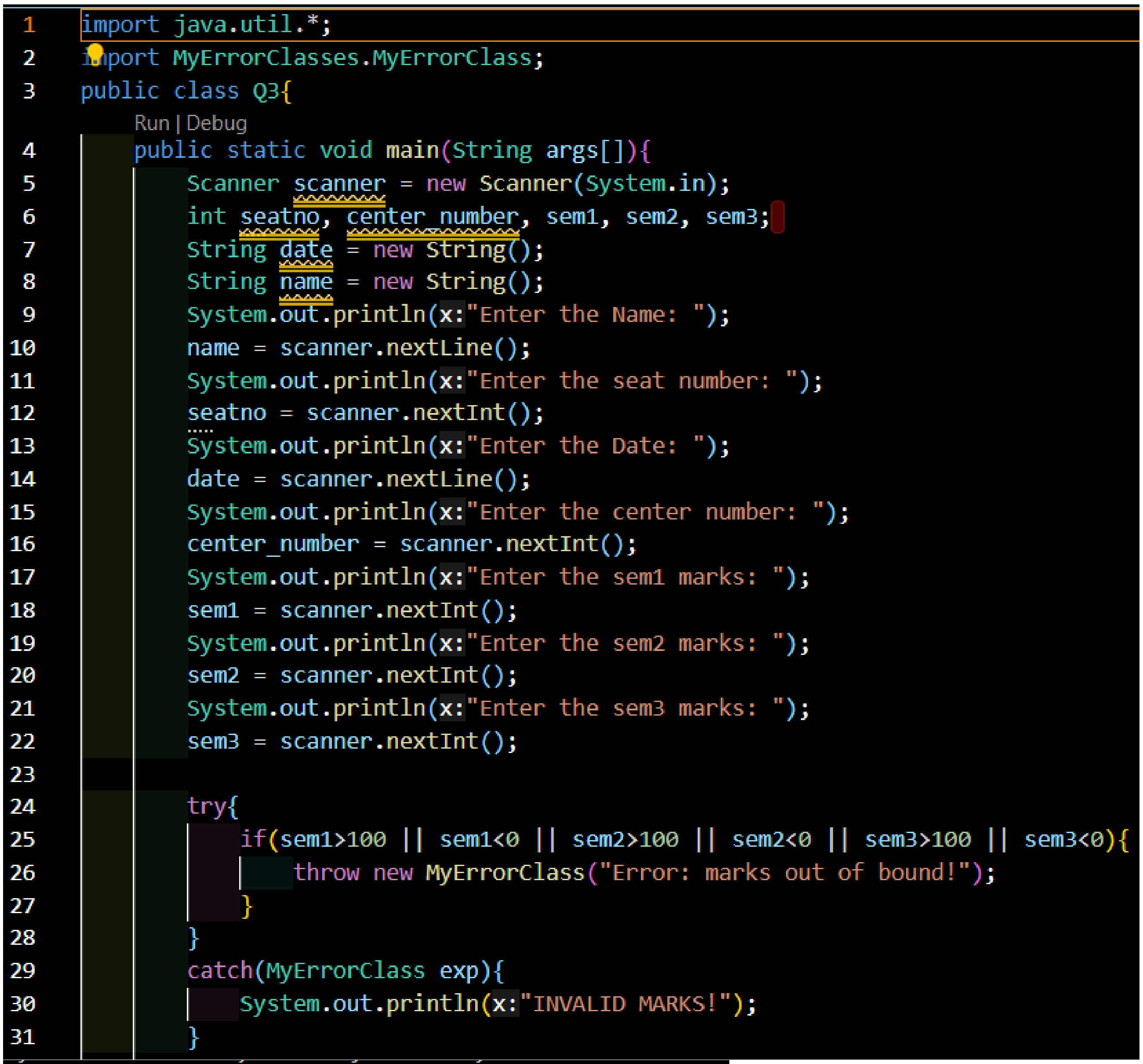


```java
import java.util.*;
public class Q2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n;
        System.out.print(s:"Enter the number of integers: ");
        n = sc.nextInt();
        int[] numbers = new int[n];
        int validCount = 0;
        int invalidCount = 0;
        System.out.println("Enter " + n + " integers:");
        for (int i = 0; i < n; i++) {
            try {
                numbers[i] = sc.nextInt();
                validCount++;
            } catch (Exception e) {
                sc.nextLine();
                System.out.println(x:"Invalid input");
                invalidCount++;
            }
        }
        System.out.println("Valid integers:" + validCount);
        System.out.println("Invalid integers:" + invalidCount);
        sc.close();
    }
}
```

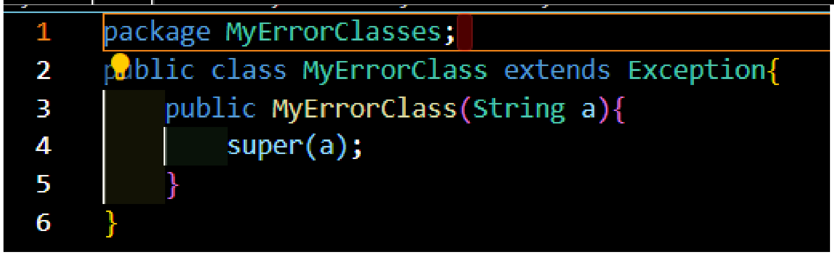


```
Enter the number of integers: 3
Enter 3 integers:
2
-5
3.2
Invalid input
Valid integers:2
Invalid integers:1
```

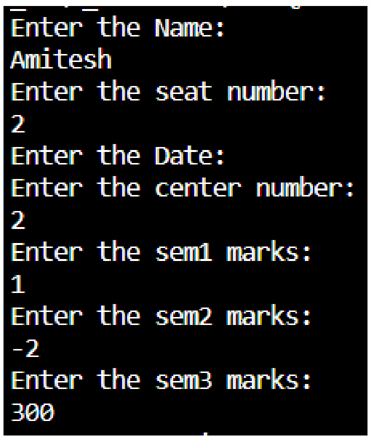
iii. Write a Java Program to calculate the Result. Result should consist of name, seatno, date, center number and marks of semester three exam. Create a User Defined Exception class MarksOutOfBoundsException, If Entered marks of any subject is greater than 100 or less than 0, and then program should create a user defined Exception of type MarksOutOfBoundsException and must have a provision to handle it



```java
import java.util.*;
import MyErrorClasses.MyErrorClass;
public class Q3{
    public static void main(String args[]){
        Scanner scanner = new Scanner(System.in);
        int seatno, center_number, sem1, sem2, sem3;
        String date = new String();
        String name = new String();
        System.out.println(x:"Enter the Name: ");
        name = scanner.nextLine();
        System.out.println(x:"Enter the seat number: ");
        seatno = scanner.nextInt();
        System.out.println(x:"Enter the Date: ");
        date = scanner.nextLine();
        System.out.println(x:"Enter the center number: ");
        center_number = scanner.nextInt();
        System.out.println(x:"Enter the sem1 marks: ");
        sem1 = scanner.nextInt();
        System.out.println(x:"Enter the sem2 marks: ");
        sem2 = scanner.nextInt();
        System.out.println(x:"Enter the sem3 marks: ");
        sem3 = scanner.nextInt();

        try{
            if(sem1>100 || sem1<0 || sem2>100 || sem2<0 || sem3>100 || sem3<0){
                throw new MyErrorClass("Error: marks out of bound!");
            }
        }
        catch(MyErrorClass exp){
            System.out.println(x:"INVALID MARKS!");
        }
```

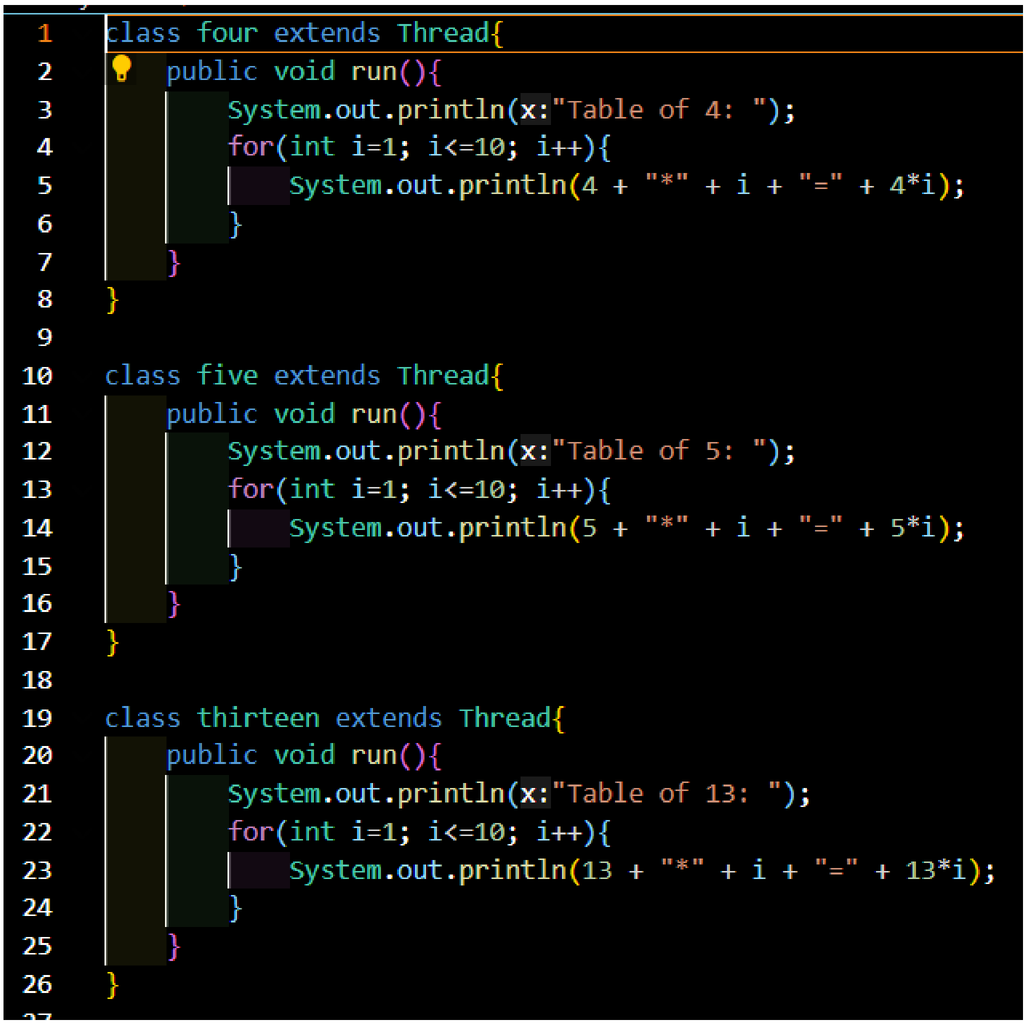


```java
package MyErrorClasses;
public class MyErrorClass extends Exception{
    public MyErrorClass(String a){
        super(a);
    }
}
```

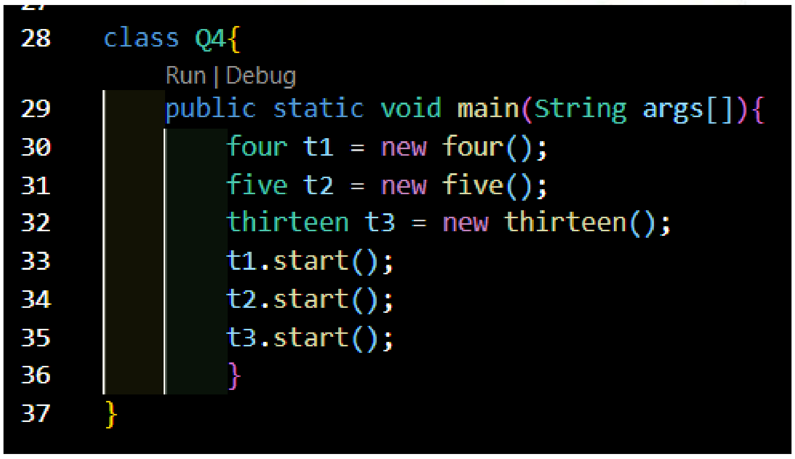


```
Enter the Name:
Amitesh
Enter the seat number:
2
Enter the Date:
Enter the center number:
2
Enter the sem1 marks:
1
Enter the sem2 marks:
-2
Enter the sem3 marks:
300
```

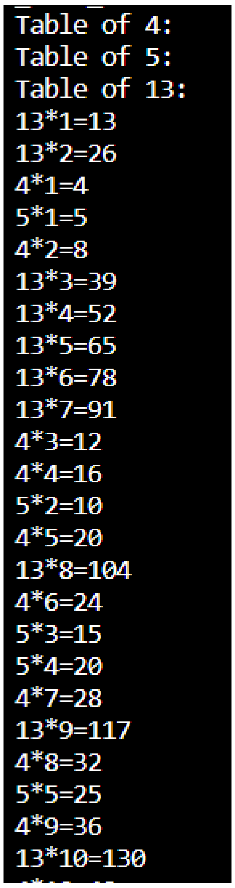
iv. Write java program to print Table of Five, Seven and Thirteen using Multithreading (Use Thread class for the implementation). Also print the total time taken by each thread for the execution.



```java
class four extends Thread{
    public void run(){
        System.out.println(x:"Table of 4: ");
        for(int i=1; i<=10; i++){
            System.out.println(4 + "*" + i + "=" + 4*i);
        }
    }
}

class five extends Thread{
    public void run(){
        System.out.println(x:"Table of 5: ");
        for(int i=1; i<=10; i++){
            System.out.println(5 + "*" + i + "=" + 5*i);
        }
    }
}

class thirteen extends Thread{
    public void run(){
        System.out.println(x:"Table of 13: ");
        for(int i=1; i<=10; i++){
            System.out.println(13 + "*" + i + "=" + 13*i);
        }
    }
}
```

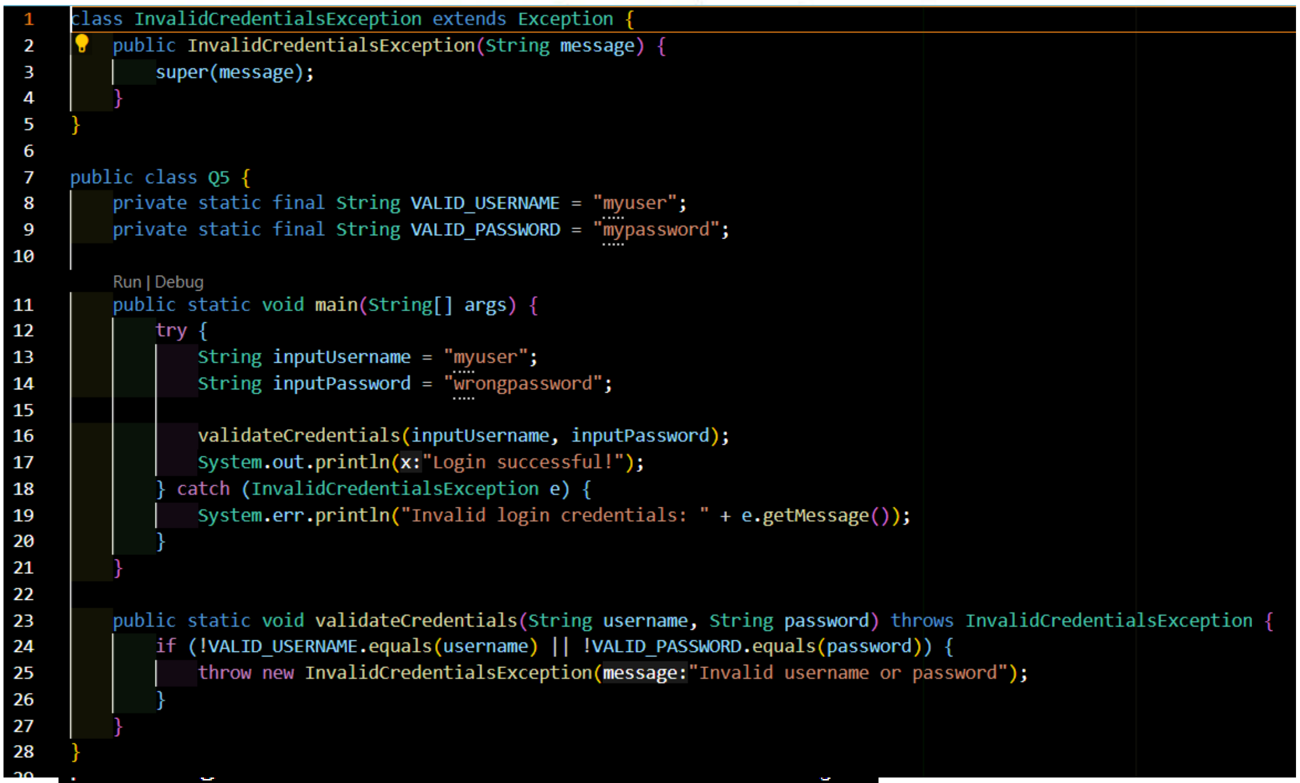


```
class Q4{
    Run | Debug
    public static void main(String args[]){
        four t1 = new four();
        five t2 = new five();
        thirteen t3 = new thirteen();
        t1.start();
        t2.start();
        t3.start();
        }
}
```

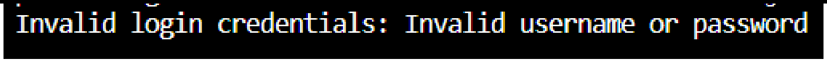


```
Table of 4:
Table of 5:
Table of 13:
13*1=13
13*2=26
4*1=4
5*1=5
4*2=8
13*3=39
13*4=52
13*5=65
13*6=78
13*7=91
4*3=12
4*4=16
5*2=10
4*5=20
13*8=104
4*6=24
5*3=15
5*4=20
4*7=28
13*9=117
4*8=32
5*5=25
4*9=36
13*10=130
```

v. Write a program Create User-Defined Exception in Java for Validating Login Credentials

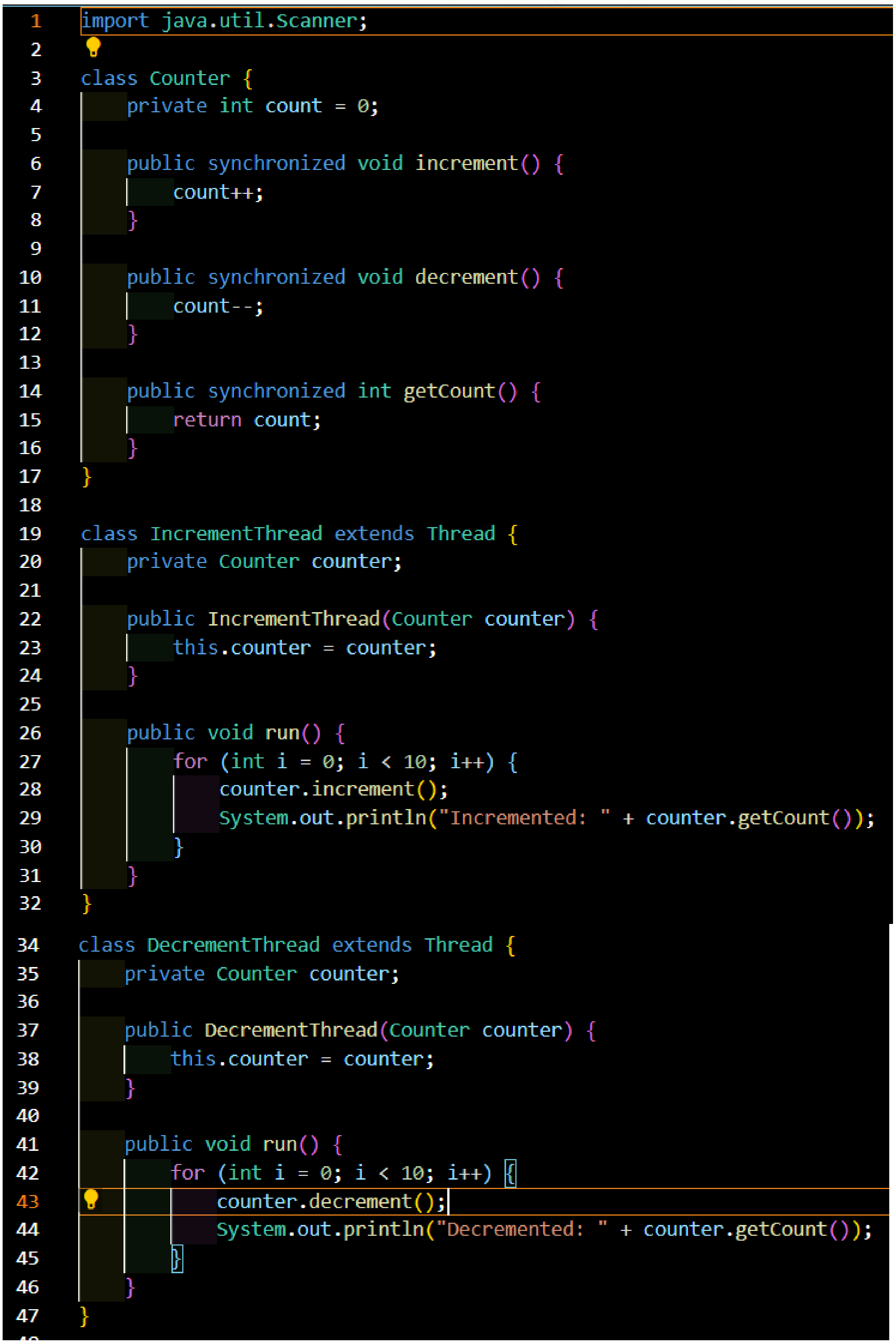


```java
class InvalidCredentialsException extends Exception {
    public InvalidCredentialsException(String message) {
        super(message);
    }
}

public class Q5 {
    private static final String VALID_USERNAME = "myuser";
    private static final String VALID_PASSWORD = "mypassword";

    Run | Debug
    public static void main(String[] args) {
        try {
            String inputUsername = "myuser";
            String inputPassword = "wrongpassword";

            validateCredentials(inputUsername, inputPassword);
            System.out.println(x:"Login successful!");
        } catch (InvalidCredentialsException e) {
            System.err.println("Invalid login credentials: " + e.getMessage());
        }
    }

    public static void validateCredentials(String username, String password) throws InvalidCredentialsException {
        if (!VALID_USERNAME.equals(username) || !VALID_PASSWORD.equals(password)) {
            throw new InvalidCredentialsException(message:"Invalid username or password");
        }
    }
}
```

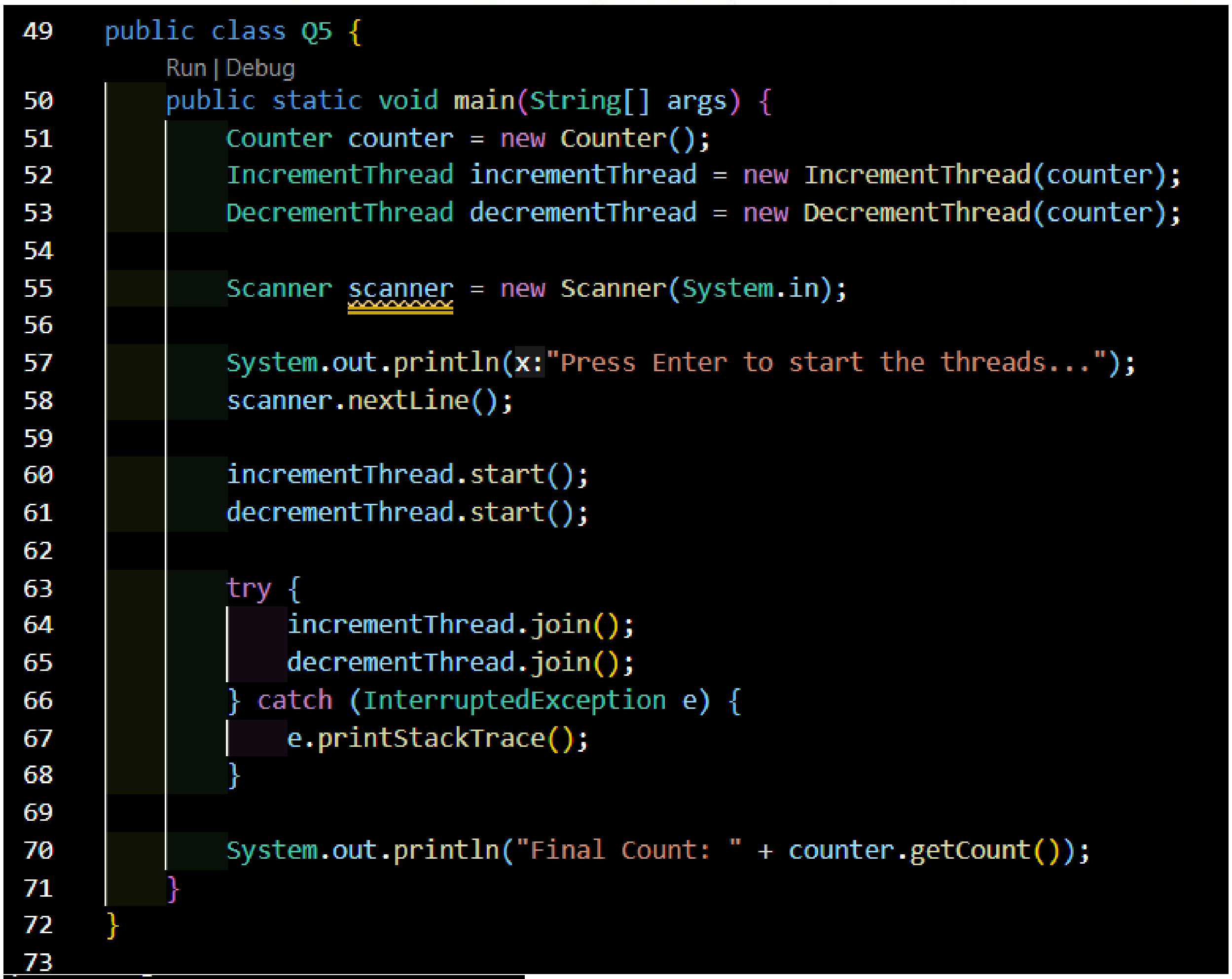


```
Invalid login credentials: Invalid username or password
```

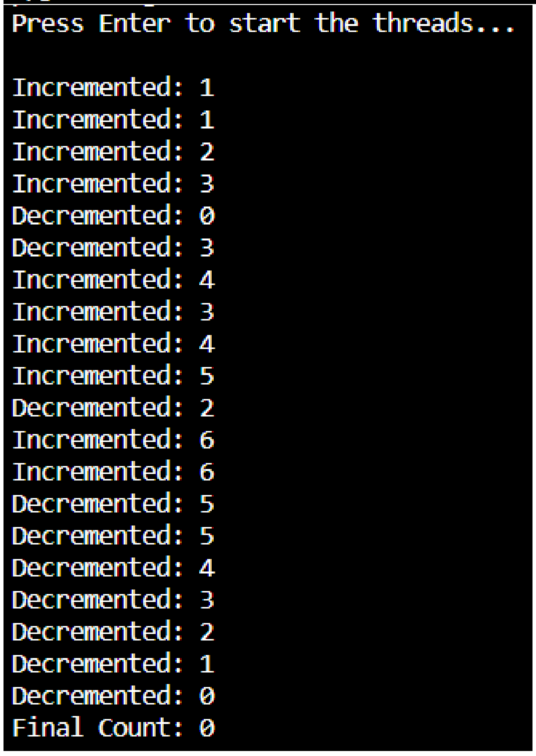
vi. Write java program to implement the concept of Thread Synchronization



```java
import java.util.Scanner;

class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public synchronized void decrement() {
        count--;
    }

    public synchronized int getCount() {
        return count;
    }
}

class IncrementThread extends Thread {
    private Counter counter;

    public IncrementThread(Counter counter) {
        this.counter = counter;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            counter.increment();
            System.out.println("Incremented: " + counter.getCount());
        }
    }
}
class DecrementThread extends Thread {
    private Counter counter;

    public DecrementThread(Counter counter) {
        this.counter = counter;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            counter.decrement();
            System.out.println("Decremented: " + counter.getCount());
        }
    }
}
```



```java
public class Q5 {
    public static void main(String[] args) {
        Counter counter = new Counter();
        IncrementThread incrementThread = new IncrementThread(counter);
        DecrementThread decrementThread = new DecrementThread(counter);

        Scanner scanner = new Scanner(System.in);

        System.out.println(x:"Press Enter to start the threads...");
        scanner.nextLine();

        incrementThread.start();
        decrementThread.start();

        try {
            incrementThread.join();
            decrementThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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



```
Press Enter to start the threads...

Incremented: 1
Incremented: 1
Incremented: 2
Incremented: 3
Decremented: 0
Decremented: 3
Incremented: 4
Incremented: 3
Incremented: 4
Incremented: 5
Decremented: 2
Incremented: 6
Incremented: 6
Decremented: 5
Decremented: 5
Decremented: 4
Decremented: 3
Decremented: 2
Decremented: 1
Decremented: 0
Final Count: 0
```