Experiment No 5

Bhuvi Ghosh
60009210191

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

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)

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

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.

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

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

Q1)

```java
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class SharedBuffer {
    private int[] buffer;
    private int size;
    private int count;
    private int in;
    private int out;

    private Lock lock = new ReentrantLock();
    private Condition notFull = lock.newCondition();
    private Condition notEmpty = lock.newCondition();

    public SharedBuffer(int size) {
        this.size = size;
        this.buffer = new int[size];
        this.count = 0;
        this.in = 0;
        this.out = 0;
    }

    public void produce(int item) {
        lock.lock();
        try {
            while (count == size) {
                try {
                    notFull.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
```

```
            }

            buffer[in] = item;
            in = (in + 1) % size;
            count++;

            System.out.println("Producing: " + item);

            notEmpty.signal();
        } finally {
            lock.unlock();
        }
    }

    public int consume() {
        lock.lock();
        try {
            while (count == 0) {
                try {
                    notEmpty.await();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }

            int item = buffer[out];
            out = (out + 1) % size;
            count--;

            System.out.println("Consuming: " + item);

            notFull.signal();

            return item;
        } finally {
            lock.unlock();
        }
    }
}

class Producer implements Runnable {
    private SharedBuffer buffer;

    public Producer(SharedBuffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            buffer.produce(i);
            try {
                Thread.sleep(100); // Simulate some work
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class Consumer implements Runnable {
    private SharedBuffer buffer;

```

```
92
93  class Consumer implements Runnable {
94      private SharedBuffer buffer;
95
96      public Consumer(SharedBuffer buffer) {
97          this.buffer = buffer;
98      }
99
100     @Override
101     public void run() {
102         for (int i = 0; i < 10; i++) {
103             int item = buffer.consume();
104             try {
105                 Thread.sleep(200); // Simulate some work
106             } catch (InterruptedException e) {
107                 Thread.currentThread().interrupt();
108             }
109         }
110     }
111 }
112
113 public class Main {
114     public static void main(String[] args) {
115         SharedBuffer buffer = new SharedBuffer(5);
116
117         Thread producerThread = new Thread(new Producer(buffer));
118         Thread consumerThread = new Thread(new Consumer(buffer));
119
120         producerThread.start();
121         consumerThread.start();
122
123         try {
```

```
99
100     @Override
101     public void run() {
102         for (int i = 0; i < 10; i++) {
103             int item = buffer.consume();
104             try {
105                 Thread.sleep(200); // Simulate some work
106             } catch (InterruptedException e) {
107                 Thread.currentThread().interrupt();
108             }
109         }
110     }
111 }
112
113 public class Main {
114     public static void main(String[] args) {
115         SharedBuffer buffer = new SharedBuffer(5);
116
117         Thread producerThread = new Thread(new Producer(buffer));
118         Thread consumerThread = new Thread(new Consumer(buffer));
119
120         producerThread.start();
121         consumerThread.start();
122
123         try {
124             producerThread.join();
125             consumerThread.join();
126         } catch (InterruptedException e) {
127             Thread.currentThread().interrupt();
128         }
129     }
130 }
```

input

```
Producing: 0
Consuming: 0
Producing: 1
Producing: 2
Consuming: 1
Producing: 3
Producing: 4
Consuming: 2
Producing: 5
Consuming: 3
Producing: 6
Producing: 7
Consuming: 4
Producing: 8
Producing: 9
Consuming: 5
Consuming: 6
Consuming: 7
Consuming: 8
Consuming: 9


...Program finished with exit code 0
Press ENTER to exit console.
```

Q2)

```java
import java.util.Scanner;

public class SystemInputValidation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int validCount = 0;
        int invalidCount = 0;

        System.out.println("Enter integers (one per line, or 'q' to quit):");

        while (scanner.hasNext()) {
            String input = scanner.next();

            if (input.equalsIgnoreCase("q")) {
                break;
            }

            try {
                int number = Integer.parseInt(input);
                System.out.println("Valid Integer: " + number);
                validCount++;
            } catch (NumberFormatException e) {
                System.out.println("Invalid Input: " + input);
                invalidCount++;
            }
        }

        System.out.println("Total Valid Integers: " + validCount);
        System.out.println("Total Invalid Inputs: " + invalidCount);
    }
}
```

Q3)

Main.java

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

class Result {
    private String name;
    private int seatNo;
    private String date;
    private int centerNumber;
    private int marks;

    public Result(String name, int seatNo, String date, int centerNumber) {
        this.name = name;
        this.seatNo = seatNo;
        this.date = date;
        this.centerNumber = centerNumber;
    }

    public void setMarks(int marks) throws MarksOutOfBoundsException {
        if (marks < 0 || marks > 100) {
            throw new MarksOutOfBoundsException("Marks out of bounds (0-100)");
        }
        this.marks = marks;
    }

    public void displayResult() {
        System.out.println("Name: " + name);
        System.out.println("Seat No: " + seatNo);
        System.out.println("Date: " + date);
        System.out.println("Center Number: " + centerNumber);
        System.out.println("Marks: " + marks);
    }
```

```java
public class Main {
    public static void main(String[] args) {
        Result studentResult = new Result("John Doe", 123456, "2023-10-27", 7890);

        try {
            studentResult.setMarks(85);
            studentResult.displayResult();
        } catch (MarksOutOfBoundsException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        try {
            studentResult.setMarks(110);
            studentResult.displayResult();
        } catch (MarksOutOfBoundsException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

```
Name: John Doe
Seat No: 123456
Date: 2023-10-27
Center Number: 7890
Marks: 85
Exception: Marks out of bounds (0-100)


...Program finished with exit code 0
Press ENTER to exit console.
```

Q4)

TablePrinter.java

```java
public class TablePrinter extends Thread {
    private int number;

    public TablePrinter(int number) {
        this.number = number;
    }

    @Override
    public void run() {
        long startTime = System.currentTimeMillis();
        System.out.println("Table of " + number + ":");
        for (int i = 1; i <= 10; i++) {
            System.out.println(number + " x " + i + " = " + (number * i));
        }
        long endTime = System.currentTimeMillis();
        long totalTime = endTime - startTime;
        System.out.println("Thread for table of " + number + " executed in " + totalTime + " milliseconds.");
    }

    public static void main(String[] args) {
        TablePrinter tableOfFive = new TablePrinter(5);
        TablePrinter tableOfSeven = new TablePrinter(7);
        TablePrinter tableOfThirteen = new TablePrinter(13);

        long startTime = System.currentTimeMillis();

        tableOfFive.start();
        tableOfSeven.start();
        tableOfThirteen.start();

```

TablePrinter.java

```
14          }
15          long endTime = System.currentTimeMillis();
16          long totalTime = endTime - startTime;
17          System.out.println("Thread for table of " + number + " executed in " + totalTime + " milliseconds.");
18      }
19
20      public static void main(String[] args) {
21          TablePrinter tableOfFive = new TablePrinter(5);
22          TablePrinter tableOfSeven = new TablePrinter(7);
23          TablePrinter tableOfThirteen = new TablePrinter(13);
24
25          long startTime = System.currentTimeMillis();
26
27          tableOfFive.start();
28          tableOfSeven.start();
29          tableOfThirteen.start();
30
31          try {
32              tableOfFive.join();
33              tableOfSeven.join();
34              tableOfThirteen.join();
35          } catch (InterruptedException e) {
36              Thread.currentThread().interrupt();
37          }
38
39          long endTime = System.currentTimeMillis();
40          long totalTime = endTime - startTime;
41          System.out.println("Total time taken for all threads: " + totalTime + " milliseconds");
42      }
43  }
```

input

```
Table of 13:
Table of 7:
Table of 5:
13 x 1 = 13
5 x 1 = 5
7 x 1 = 7
13 x 2 = 26
13 x 3 = 39
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
5 x 2 = 10
7 x 5 = 35
13 x 4 = 52
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
13 x 5 = 65
5 x 6 = 30
13 x 6 = 78
13 x 7 = 91
5 x 7 = 35
13 x 8 = 104
13 x 9 = 117
5 x 8 = 40
13 x 10 = 130
5 x 9 = 45
5 x 10 = 50
Thread for table of 5 executed in 177 milliseconds.
Thread for table of 13 executed in 177 milliseconds.
Thread for table of 7 executed in 100 milliseconds.
Total time taken for all threads: 187 milliseconds


...Program finished with exit code 0
Press ENTER to exit console.
```

Q5)

```java
class InvalidCredentialsException extends Exception {
    public InvalidCredentialsException(String message) {
        super(message);
    }
}
class LoginValidator {
    private static final String validUsername = "myusername";
    private static final String validPassword = "mypassword";

    public static void validateCredentials(String username, String password) throws InvalidCredentialsException {
        if (!validUsername.equals(username) || !validPassword.equals(password)) {
            throw new InvalidCredentialsException("Invalid login credentials");
        }
        System.out.println("Login successful!");
    }
}

public class Main {
    public static void main(String[] args) {
        String username = "myusername";
        String password = "mypassword";

        try {
            LoginValidator.validateCredentials(username, password);
        } catch (InvalidCredentialsException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

input

```
Login successful!


...Program finished with exit code 0
Press ENTER to exit console.
```

Q6)

SynchronizationEx...

```java
class Counter {
    private int count = 0;

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

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

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

public class SynchronizationExample {
    public static void main(String[] args) {
        Counter counter = new Counter();

        Thread incrementThread = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

        Thread decrementThread = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.decrement();
            }
        });

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

        try {
            incrementThread.join();
            decrementThread.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

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

input

```
Final Count: 0


...Program finished with exit code 0
Press ENTER to exit console.
```