1) Exercise -1

Create an Customer class with the following fields

a. id
b. firstName
c. lastName
d. age
e. mobile

- accept the data from the keyboard and throw user defined exception if the firstName and lastName (or either) is null or empty
- The valid age of the customer is greater than or equal to 15. If the age is < 15 year another user defined exception to be thrown.
- Throw another user defined exception if the salary of an employee is below 3000
- Handle all the exceptions in the main method.

2) Exercise -2

Write a program that divides two numbers provided by the user. Use a try-catch block to catch any ArithmeticException that occurs when dividing by zero. Display a custom error message in the catch block.

3) Exercise – 3

Modify the program from Exercise 1 to include multiple catch blocks to handle different exceptions, such as `NumberFormatException` when the input is not a valid number. Ensure each catch block handles a specific exception and displays an appropriate error message.

4) Exercise – 4

Extend the program from Exercise 2 by adding a finally block. Use this block to display a message that always executes, regardless of whether an exception was caught or not. This could be a simple "Program Ended" message or similar.

5) Exercise – 5

Write a program that reads a file name from the user and prints the contents of the file to the console. Use a `try-with-resources` statement to automatically close the file reader. Ensure your program gracefully handles `FileNotFoundException` and `IOException` and displays an appropriate message if the file does not exist or an IOException occurs.

6) Exercise – 6

Write a method named `processFile` that throws an IOException. This method should simulate reading a file by simply throwing the exception. Call `processFile` from

another method named `readFile`, which handles the IOException with a try-catch block. Ensure `readFile` provides a clear message to the user about the error.

7) Exercise – 7

Modify the program from Exercise 6 by removing the try-catch block from the `readFile` method. Instead, use the `throws` keyword to indicate that `readFile` can throw the IOException. Then, handle this exception in the `main` method or another wrapper method, providing detailed error handling and user feedback.

8) Exercise – 8

Create a custom exception class named `InvalidInputException`. Then, write a program that checks if the user input is a positive number. If the input is negative or zero, throw an `InvalidInputException` with a suitable error message. Catch this exception and display the message to the user.

9) Exercise 9: Creating and Running Threads

10) Objective: Learn how to create and run threads using two approaches: extending the Thread class and implementing the Runnable interface.
11)
12) Tasks:
13)
14) Extend the Thread class:
15)
16) Create a class that extends Thread.
17) Override the run() method to print "Hello from <YourClassName> thread!".
18) In your main method, create an instance of your class and start the thread using the start() method.
19) Implement the Runnable interface:
20)
21) Create a class that implements Runnable.
22) Implement the run() method to print "Hello from <YourClassName> thread!".
23) In your main method, create an instance of your class, pass it to a Thread object, and start the thread.

10)Exercise 10: Understanding Thread Lifecycle

Objective: Familiarize yourself with the lifecycle of a thread by observing different states.

Tasks:

Create a thread that sleeps for a few seconds using Thread.sleep(3000) inside the run() method.

Before starting the thread, print its state.

After starting the thread, print its state again.

After the thread has finished executing, print its state once more.

Use comments in your code to note the observed states at each step.

---

11)Exercise 11: Synchronization

Objective: Understand the importance of synchronization in multithreading to prevent thread interference.

Tasks:

Create a shared resource class (e.g., a counter) with a method to increment its value.

Create multiple threads that increment the counter's value.

Run your program without synchronization and observe the result.

Modify the increment method to be synchronized and observe the changes.

12)Exercise 12: Implement a Generic Method

Create a generic method named swap in a utility class, which swaps two elements in an array. The array and the indices of the elements to be swapped should be passed as parameters.

```
public static <T> void swap(T[] array, int i, int j) {
    // Implement this method
}
```

13) Exercise 13: Generic Calculator class

Create a generic calculator class which does the following operations on different data types

Addition

AreEqual