***Assignement***

1.This program demonstrates how an IOException is triggered by

attempting to read a non-existent file. It simulates a real-world scenario

where input/output operations may fail, such as reading or writing to files. The program handles the exception using a try-catch block and provides a

user-friendly message when the file cannot be read. Program

import java.io.\*;

/\*\*

\* Program to demonstrate IOException. \* This program attempts to read a non-existent file to trigger an

IOException

\* and handles the error gracefully using a try-catch block. \*/

public class IOExceptionExample {

public static void main(String[] args) {

System.out.println("=== IOException Example ===");

try {

// Attempting to read a file that does not exist

BufferedReader reader = new BufferedReader(new

FileReader("non\_existent\_file.txt"));

reader.readLine();

} catch (IOException e) {

// Catch block to handle IOException

System.out.println("IOException caught: " + e.getMessage());

} finally {

// Code in finally block runs regardless of exception occurrence

System.out.println("Finished attempting to read the file.");

}

}

}

2.This program focuses on a specific type of IOException, the

FileNotFoundException. It simulates an attempt to open a missing file, showing how such situations can be handled gracefully in a program. This is useful in scenarios where file operations depend on user input or

external resources

Program

import java.io.\*;

/\*\*

\* Program to demonstrate FileNotFoundException. \* This program tries to open a missing file to trigger a

FileNotFoundException

\* and handles it using a try-catch block. \*/

public class FileNotFoundExceptionExample {

public static void main(String[] args) {

System.out.println("=== FileNotFoundException Example ===");

try {

// Attempting to open a file that does not exist

FileInputStream file = new FileInputStream("missing\_file.txt");

} catch (FileNotFoundException e) {

// Catch block to handle FileNotFoundException

System.out.println("FileNotFoundException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting to open the file.");

}

}

}

3. This program demonstrates an EOFException, which occurs when a

program attempts to read beyond the end of a file. It highlights how to

handle unexpected situations in file streams, such as empty files or

improperly terminated data files. Program

import java.io.\*;

/\*\*

\* Program to demonstrate EOFException. \* This program tries to read beyond the end of an empty file using

ObjectInputStream

\* to trigger an EOFException. \*/

public class EOFExceptionExample {

public static void main(String[] args) {

System.out.println("=== EOFException Example ===");

try (ObjectInputStream ois = new ObjectInputStream(new

FileInputStream("empty\_file.txt"))) {

ois.readObject();

} catch (EOFException e) {

// Catch block to handle EOFException

System.out.println("EOFException caught: " + e.getMessage());

} catch (IOException | ClassNotFoundException e) {

System.out.println("Exception caught: " + e.getMessage());

} finally {

System.out.println("Finished attempting to read the file.");

}

}

}

4. The program simulates a database connection failure by attempting to

connect to a non-existent database. It demonstrates how to catch a

SQLException, which is critical in handling database errors such as

incorrect queries, unreachable servers, or invalid credentials. Program

import java.sql.\*;

/\*\*

\* Program to demonstrate SQLException. \* This program tries to connect to a non-existent database to trigger a

SQLException. \*/

public class SQLExceptionExample {

public static void main(String[] args) {

System.out.println("=== SQLException Example ===");

try {

// Attempting to connect to a non-existent database

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/nonexistentd

b", "user", "password");

} catch (SQLException e) {

// Catch block to handle SQLException

System.out.println("SQLException caught: " + e.getMessage());

} finally {

System.out.println("Finished attempting database connection.");

}

}

}

5.This program shows how a ClassNotFoundException is triggered when

the program tries to load a non-existent class dynamically. It's useful for

understanding runtime errors in scenarios involving reflection or external

libraries. Program

/\*\*

\* Program to demonstrate ClassNotFoundException. \* This program tries to load a non-existent class to trigger a

ClassNotFoundException. \*/

public class ClassNotFoundExceptionExample {

public static void main(String[] args) {

System.out.println("=== ClassNotFoundException Example ===");

try {

// Attempting to load a non-existent class

Class.forName("com.nonexistent.Class");

} catch (ClassNotFoundException e) {

// Catch block to handle ClassNotFoundException

System.out.println("ClassNotFoundException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting to load the class.");

}

}

}

6.The program demonstrates an ArithmeticException, triggered by

dividing a number by zero. It helps developers handle invalid arithmetic

operations that might crash a program. Program

/\*\*

\* Program to demonstrate ArithmeticException.

\* This program performs division by zero to trigger an

ArithmeticException. \*/

public class ArithmeticExceptionExample {

public static void main(String[] args) {

System.out.println("=== ArithmeticException Example ===");

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

// Catch block to handle ArithmeticException

System.out.println("ArithmeticException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting the calculation.");

}

}

}

7. This program triggers a NullPointerException by attempting to access a

method on a null object reference. It illustrates how to identify and handle

cases where objects may not be properly initialized. Program

/\*\*

\* Program to demonstrate NullPointerException. \* This program accesses a null reference to trigger a

NullPointerException. \*/

public class NullPointerExceptionExample {

public static void main(String[] args) {

System.out.println("=== NullPointerException Example ===");

try {

String str = null;

str.length(); // This will throw NullPointerException

} catch (NullPointerException e) {

// Catch block to handle NullPointerException

System.out.println("NullPointerException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting to access null

reference.");

}

}

}

8. This program demonstrates an ArrayIndexOutOfBoundsException, which occurs when attempting to access an array index outside its bounds. It highlights the importance of validating array indices in loops or data

processing

Program

/\*\*

\* Program to demonstrate ArrayIndexOutOfBoundsException. \* This program accesses an invalid array index to trigger an

ArrayIndexOutOfBoundsException. \*/

public class ArrayIndexOutOfBoundsExceptionExample {

public static void main(String[] args) {

System.out.println("=== ArrayIndexOutOfBoundsException

Example ===");

try {

int[] arr = {1, 2, 3};

int invalidElement = arr[5]; // Accessing invalid index

} catch (ArrayIndexOutOfBoundsException e) {

// Catch block to handle ArrayIndexOutOfBoundsException

System.out.println("ArrayIndexOutOfBoundsException caught: " + e.getMessage());

} finally {

System.out.println("Finished attempting to access invalid array

index.");

}

}

}

9.This program triggers a ClassCastException by performing an invalid

type cast. It demonstrates the need for type safety when working with

objects and how to handle such errors

Program

/\*\*

\* Program to demonstrate ClassCastException.

\* This program performs an invalid type cast to trigger a

ClassCastException. \*/

public class ClassCastExceptionExample {

public static void main(String[] args) {

System.out.println("=== ClassCastException Example ===");

try {

Object obj = new Integer(42);

String str = (String) obj; // Invalid type cast

} catch (ClassCastException e) {

// Catch block to handle ClassCastException

System.out.println("ClassCastException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting invalid type cast.");

}

}

}

10.The program showcases an IllegalArgumentException by passing an

invalid argument (negative value) to the Thread.sleep() method. It

emphasizes the importance of validating method inputs before calling

them. Program

/\*\*

\* Program to demonstrate IllegalArgumentException. \* This program passes an invalid argument to a method to trigger an

IllegalArgumentException. \*/

public class IllegalArgumentExceptionExample {

public static void main(String[] args) {

System.out.println("=== IllegalArgumentException Example ===");

try {

Thread.sleep(-1000); // Invalid argument

} catch (IllegalArgumentException | InterruptedException e) {

// Catch block to handle IllegalArgumentException

System.out.println("IllegalArgumentException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting to pass invalid

argument.");

}

}

}

11.This program demonstrates a NumberFormatException by attempting

to convert an invalid string into a number. It highlights how to manage

data conversion errors in scenarios such as user input or file parsing

Program

/\*\*

\* Program to demonstrate NumberFormatException. \* This program converts an invalid string to a number to trigger a

NumberFormatException. \*/

public class NumberFormatExceptionExample {

public static void main(String[] args) {

System.out.println("=== NumberFormatException Example ===");

try {

int number = Integer.parseInt("not\_a\_number"); // Invalid string

} catch (NumberFormatException e) {

// Catch block to handle NumberFormatException

System.out.println("NumberFormatException caught: " +

e.getMessage());

} finally {

System.out.println("Finished attempting invalid number

conversion.");

}

}

}