import java.io.\*;

import java.sql.\*;

import java.util.Scanner;

public class ExceptionHandler {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

while (true) {

// Display menu options to trigger different exceptions

System.out.println("\n=== Exception Handling Tester ===");

System.out.println("Choose an exception to simulate:");

System.out.println("1. IOException (e.g., File Reading)");

System.out.println("2. FileNotFoundException (e.g., Missing File)");

System.out.println("3. EOFException (e.g., Unexpected End of File)");

System.out.println("4. SQLException (e.g., Database Connection)");

System.out.println("5. ClassNotFoundException (e.g., Missing Class)");

System.out.println("6. ArithmeticException (e.g., Divide by Zero)");

System.out.println("7. NullPointerException (e.g., Null Object Access)");

System.out.println("8. ArrayIndexOutOfBoundsException (e.g., Invalid Index)");

System.out.println("9. ClassCastException (e.g., Invalid Type Casting)");

System.out.println("10. IllegalArgumentException (e.g., Invalid Argument)");

System.out.println("11. NumberFormatException (e.g., Invalid Number Format)");

System.out.println("0. Exit");

// Get validated input for menu choice

choice = getValidatedInteger("Enter your choice: ",scanner);

// Handle the user's choice with appropriate methods

switch (choice) {

case 1: handleIOException(scanner); break;

case 2: handleFileNotFoundException(scanner); break;

case 3: handleEOFException(scanner); break;

case 4: handleSQLException(scanner); break;

case 5: handleClassNotFoundException(scanner); break;

case 6: handleArithmeticException(scanner); break;

case 7: handleNullPointerException(); break;

case 8: handleArrayIndexOutOfBoundsException(scanner); break;

case 9: handleClassCastException(); break;

case 10: handleIllegalArgumentException(scanner); break;

case 11: handleNumberFormatException(scanner); break;

case 0:

System.out.println("Exiting...");

System.exit(0);

break;

default:

System.out.println("Invalid choice! Try again.");

break;

}

}

}

/\*\*

\* Utility method to get validated integer input from the user.

\*/

private static int getValidatedInteger(String message,Scanner scanner) {

while (true) {

try {

System.out.print(message);

return Integer.parseInt(scanner.nextLine().trim());

} catch (NumberFormatException e) {

System.out.print("Invalid input! Please enter a valid number: ");

}

}

}

/\*\*

\* Utility method to get validated integer input from the user.

\*/

private static void printTitle(String title) {

System.out.print("\nHandling " + title);

System.out.print("\n==============================\n");

}

/\*\*

\* Approach: Reading a file and handling IOException if it occurs.

\* Example: Attempting to read a non-existent file can trigger IOException.

\*/

private static void handleIOException(Scanner scanner) {

printTitle("IOException");

try {

System.out.print("Enter the file name to read: ");

String filename = scanner.nextLine(); // Get the filename from the user input.

// Create a BufferedReader to read the file.

BufferedReader reader = new BufferedReader(new FileReader(filename));

// Read and print the first line of the file.

System.out.println("First line: " + reader.readLine());

reader.close(); // Close the file reader to release resources.

} catch (IOException e) {

// Handle IOException and print the error message.

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

}

}

/\*\*

\* Approach: Opening a file and handling FileNotFoundException if it occurs.

\* Example: Trying to open a file that does not exist.

\*/

private static void handleFileNotFoundException(Scanner scanner) {

printTitle("FileNotFoundException");

try {

System.out.print("Enter the file name to open: ");

String filename = scanner.nextLine(); // Get the filename from the user input.

// Attempt to open the file using FileInputStream.

FileInputStream fis = new FileInputStream(filename);

fis.close(); // Close the file input stream if gotten to release resources

} catch (FileNotFoundException e) {

// Handle FileNotFoundException and print the error message.

System.out.println("FileNotFoundException occurred: " + e.getMessage());

} catch (IOException e) {

// Handle IOException and print the error message.

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

}

}

/\*\*

\* Approach: Reading a file byte by byte and handling EOFException when reaching the end.

\* Example: Trying to read beyond the end of a file.

\*/

private static void handleEOFException(Scanner scanner) {

printTitle("EOFException");

try {

System.out.print("Enter the file name to read: ");

String filename = scanner.nextLine(); // Get the filename from the user input.

// Open the file using DataInputStream to read its contents.

DataInputStream dis = new DataInputStream(new FileInputStream(filename));

// Continuously read bytes until an EOFException is thrown.

while (true) {

dis.readByte(); // Read one byte at a time from the file.

}

} catch (EOFException e) {

// Handle EOFException and print the error message.

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

} catch (IOException e) {

// Handle general IOException and print the error message.

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

}

}

/\*\*

\* Approach: Attempting to connect to a database and handling SQLException.

\* Example: Providing invalid database credentials or URL.

\*/

private static void handleSQLException(Scanner scanner) {

printTitle("SQLException");

System.out.print("Enter database URL: ");

String dbUrl = scanner.nextLine(); // Get the database URL from the user.

System.out.print("Enter username: ");

String username = scanner.nextLine(); // Get the username from the user.

System.out.print("Enter password: ");

String password = scanner.nextLine(); // Get the password from the user.

// Try to establish a database connection using the provided credentials.

// The try-with-resources ensures the connection is automatically closed.

try (Connection conn = DriverManager.getConnection(dbUrl, username, password)) {

// If the connection is successful, print a success message.

System.out.println("Connection successful!");

} catch (SQLException e) {

// Handle any SQL exceptions that occur during the connection attempt.

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

}

}

/\*\*

\* Approach: Attempting to load a class and handling ClassNotFoundException.

\* Example: Providing an invalid class name to load.

\*/

private static void handleClassNotFoundException(Scanner scanner) {

printTitle("ClassNotFoundException");

try {

System.out.print("Enter the class name to load: ");

String className = scanner.nextLine(); // Get the class name from the user input.

// Attempt to load the specified class using Class.forName().

Class.forName(className);

System.out.println("Class loaded successfully."); // Inform the user if the class loads without error.

} catch (ClassNotFoundException e) {

// Handle ClassNotFoundException and print the error message.

System.out.println("ClassNotFoundException occurred: " + e.getMessage());

}

}

/\*\*

\* Approach: Dividing two integers and handling ArithmeticException for division by zero.

\* Example: Entering zero as the divisor.

\*/

private static void handleArithmeticException(Scanner scanner) {

printTitle("ArithmeticException");

try {

int dividend = getValidatedInteger("Enter dividend: ",scanner); // Get the dividend from the user and validate input.

int divisor = getValidatedInteger("Enter divisor: ",scanner); // Get the divisor from the user and validate input.

// Perform division and print the result.

int result = dividend / divisor;

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

// Handle ArithmeticException and print the error message (e.g., division by zero).

System.out.println("ArithmeticException occurred: " + e.getMessage());

}

}

/\*\*

\* Approach: Accessing a method on a null object reference and handling NullPointerException.

\* Example: Calling a method on a null string.

\*/

private static void handleNullPointerException() {

printTitle("NullPointerException");

try {

String str = null; // Assign null to the string variable.

// Attempt to call a method on the null string object (this will throw a NullPointerException).

str.length();

} catch (NullPointerException e) {

// Handle NullPointerException and print the error message.

System.out.println("NullPointerException occurred: " + e.getMessage());

}

}

/\*\*

\* Approach: Accessing an invalid array index and handling ArrayIndexOutOfBoundsException.

\* Example: Entering an index outside the array bounds.

\*/

private static void handleArrayIndexOutOfBoundsException(Scanner scanner) {

printTitle("ArrayIndexOutOfBoundsException");

try {

int size = getValidatedInteger("Enter array size: ",scanner); // Get the array size from the user and validate input.

// Create an array with the specified size.

int[] arr = new int[size];

int index = getValidatedInteger("Enter index to access: ",scanner); // Get the index to access from the user.

// Attempt to access the array at the specified index and print the value.

System.out.println("Value: " + arr[index]);

} catch (ArrayIndexOutOfBoundsException e) {

// Handle ArrayIndexOutOfBoundsException and print the error message (invalid index).

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

}

}

/\*\*

\* Approach: Casting an object to an incompatible type and handling ClassCastException.

\* Example: Casting an instance of one class to an unrelated class.

\*/

private static void handleClassCastException() {

printTitle("ClassCastException");

try {

// Create an object of ExceptionHandler class.

Object obj = new ExceptionHandler();

// Attempt to cast the object to a String (which is an invalid cast).

String str = (String) obj;

} catch (ClassCastException e) {

// Handle ClassCastException and print the error message (invalid cast).

System.out.println("ClassCastException occurred: " + e.getMessage());

}

}

/\*\*

\* Approach: Providing an invalid argument and handling IllegalArgumentException.

\* Example: Providing a negative sleep time.

\*/

private static void handleIllegalArgumentException(Scanner scanner) {

printTitle("IllegalArgumentException");

try {

System.out.print("Enter sleep time in milliseconds: ");

long time = scanner.nextLong(); // Get the sleep time from the user.

scanner.nextLine();

// Attempt to put the current thread to sleep for the given time.

Thread.sleep(time);

} catch (IllegalArgumentException | InterruptedException e) {

// Handle IllegalArgumentException and InterruptedException and print the error message.

System.out.println("IllegalArgumentException occurred: " + e.getMessage());

}

}

/\*\*

\* Approach: Parsing an invalid number string and handling NumberFormatException.

\* Example: Entering a non-numeric string when a number is expected.

\*/

private static void handleNumberFormatException(Scanner scanner) {

printTitle("NumberFormatException");

try {

System.out.print("Enter a number: ");

String input = scanner.nextLine(); // Get the user input (expected to be a number).

// Attempt to parse the input string into an integer.

int num = Integer.parseInt(input);

// Print the parsed number if successful.

System.out.println("Parsed number: " + num);

} catch (NumberFormatException e) {

// Handle NumberFormatException and print the error message (invalid format).

System.out.println("NumberFormatException occurred: " + e.getMessage());

}

}

}