CodInClub
Powered By BridgeLabz

Practice Problems for Exception Handling

1. Checked Exception (Compile-time Exception)

💡 **Problem Statement:**

Create a Java program that reads a file named `"data.txt"`. If the file does not exist, handle the `IOException` properly and display a user-friendly message.

Expected Behavior:

- If the file exists, print its contents.
- If the file does not exist, catch the `IOException` and print `"File not found"`.

```java
import java.io.*;

public class FileRead {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("data.txt");
            int i;
            while ((i = fis.read()) != -1) {
                System.out.print((char) i);
            }
            fis.close();
        } catch (IOException e) {
            System.out.println("File not found");
        }
    }
}
```

2. Unchecked Exception (Runtime Exception)

💡 **Problem Statement:**

Write a Java program that asks the user to enter two numbers and divides them. Handle possible exceptions such as:

- **`ArithmeticException`** if division by zero occurs.
- **`InputMismatchException`** if the user enters a non-numeric value.

Expected Behavior:

- If the user enters valid numbers, print the result of the division.
- If the user enters `0` as the denominator, catch and handle `ArithmeticException`.
- If the user enters a non-numeric value, catch and handle `InputMismatchException`.

```
import java.util.*;

public class SafeDivision {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter numerator: ");
            int a = sc.nextInt();
            System.out.print("Enter denominator: ");
            int b = sc.nextInt();
            int result = a / b;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
```

```java
        } catch (InputMismatchException e) {
            System.out.println("Invalid input! Enter numbers only.");
        }
    }
}
```

---

3. Custom Exception (User-defined Exception)

💡 **Problem Statement:**

Create a **custom exception** called `InvalidAgeException`.

- Write a method `validateAge(int age)` that throws `InvalidAgeException` if the age is below 18.
- In `main()`, take user input and call `validateAge()`.
- If an exception occurs, display `"Age must be 18 or above"`.

Expected Behavior:

- If the age is `>=18`, print `"Access granted!"`.
- If age `<18`, throw `InvalidAgeException` and display the message.

```java
import java.util.*;

class InvalidAgeException extends Exception {
    public InvalidAgeException(String msg) {
        super(msg);
    }
}
```

```
public class AgeValidation {
    static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter age: ");
            int age = sc.nextInt();
            validateAge(age);
            System.out.println("Access granted!");
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

---

4. Multiple Catch Blocks

💡 **Problem Statement:**

Create a Java program that performs array operations.

- Accept an integer array and an index number.
- Retrieve and print the value at that index.
- Handle the following exceptions:
    - `ArrayIndexOutOfBoundsException` if the index is out of range.
    - `NullPointerException` if the array is `null`.

Expected Behavior:

- If valid, print `"Value at index X: Y"`.
- If the index is out of bounds, display `"Invalid index!"`.
- If the array is null, display `"Array is not initialized!"`.

```java
import java.util.*;

public class ArrayAccess {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = null;

        try {
            System.out.print("Enter array size: ");
            int size = sc.nextInt();
            arr = new int[size];
            System.out.println("Enter " + size + " elements:");
            for (int i = 0; i < size; i++) {
                arr[i] = sc.nextInt();
            }
            System.out.print("Enter index to access: ");
            int index = sc.nextInt();
            System.out.println("Value at index " + index + ": " +
arr[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Invalid index!");
        } catch (NullPointerException e) {
            System.out.println("Array is not initialized!");
        }
    }
}
```

5. try-with-resources (Auto-closing Resources)

💡 **Problem Statement:**

Write a Java program that reads the first line of a file named `"info.txt"` using `BufferedReader`.

- Use **`try-with-resources`** to ensure the file is automatically closed after reading.
- Handle any `IOException` that may occur.

Expected Behavior:

- If the file exists, print its first line.
- If the file does not exist, catch `IOException` and print `"Error reading file"`.

```
import java.io.*;

public class TryWithResources {
    public static void main(String[] args) {
        try (BufferedReader br = new BufferedReader(new
FileReader("info.txt"))) {
            String line = br.readLine();
            System.out.println("First line: " + line);
        } catch (IOException e) {
            System.out.println("Error reading file");
        }
    }
}
```

6. throw vs. throws (Exception Propagation)

💡 **Problem Statement:**

Create a method `calculateInterest(double amount, double rate, int years)` that:

- Throws `IllegalArgumentException` if `amount` or `rate` is negative.
- Propagates the exception using `throws` and handles it in `main()`.

Expected Behavior:

- If valid, return and print the calculated interest.
- If invalid, catch and display `"Invalid input: Amount and rate must be positive"`.

```
import java.util.*;

public class InterestCalc {
    static double calculateInterest(double amount, double rate, int
years) throws IllegalArgumentException {
        if (amount < 0 || rate < 0) {
            throw new IllegalArgumentException();
        }
        return amount * rate * years / 100;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter amount: ");
            double amt = sc.nextDouble();
            System.out.print("Enter rate: ");
            double rate = sc.nextDouble();
            System.out.print("Enter years: ");
            int years = sc.nextInt();
            double interest = calculateInterest(amt, rate, years);
            System.out.println("Interest: " + interest);
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid input: Amount and rate must
be positive");
        }
    }
}
```

---

7. finally Block Execution

💡 **Problem Statement:**

Write a program that performs **integer division** and demonstrates the **`finally` block execution**.

- The program should:
    - Take two integers from the user.
    - Perform division.
    - Handle `ArithmeticException` (if dividing by zero).
    - Ensure `"Operation completed"` is always printed using `finally`.

Expected Behavior:

- If valid, print the result.
- If an exception occurs, handle it and still print `"Operation completed"`.

```java
import java.util.*;

public class FinallyExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter numerator: ");
            int a = sc.nextInt();
            System.out.print("Enter denominator: ");
            int b = sc.nextInt();
            int result = a / b;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        } finally {
            System.out.println("Operation completed");
        }
    }
}
```

---

8. Exception Propagation in Methods

💡 **Problem Statement:**

Create a Java program with three methods:

- `method1()`: Throws an `ArithmeticException` (`10 / 0`).
- `method2()`: Calls `method1()`.

- `main()`: Calls `method2()` and handles the exception.

Expected Behavior:

- The exception propagates from `method1()` → `method2()` → `main()`.
- Catch and handle it in `main()`, printing `"Handled exception in main"`.

```java
public class ExceptionPropagation {
    static void method1() {
        int x = 10 / 0;
    }

    static void method2() {
        method1();
    }

    public static void main(String[] args) {
        try {
            method2();
        } catch (ArithmeticException e) {
            System.out.println("Handled exception in main");
        }
    }
}
```

9. Nested try-catch Block

💡 **Problem Statement:**

Write a Java program that:

- Takes an **array** and a **divisor** as input.
- Tries to access an element at an index.
- Tries to divide that element by the divisor.
- Uses **nested try-catch** to handle:
    - `ArrayIndexOutOfBoundsException` if the index is invalid.
    - `ArithmeticException` if the divisor is zero.

Expected Behavior:

- If valid, print the division result.
- If the index is invalid, catch and display `"Invalid array index!"`.
- If division by zero, catch and display `"Cannot divide by zero!"`.

```java
import java.util.*;

public class NestedTryCatch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter array size: ");
            int size = sc.nextInt();
            int[] arr = new int[size];
            System.out.println("Enter array elements:");
            for (int i = 0; i < size; i++) {
                arr[i] = sc.nextInt();
            }

            try {
                System.out.print("Enter index: ");
                int index = sc.nextInt();
                System.out.print("Enter divisor: ");
                int divisor = sc.nextInt();
                int result = arr[index] / divisor;
                System.out.println("Result: " + result);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Invalid array index!");
            } catch (ArithmeticException e) {
                System.out.println("Cannot divide by zero!");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input!");
        }
    }
}
```

---

10. Bank Transaction System (Checked + Custom Exception)

💡 **Problem Statement:**

Develop a **Bank Account System** where:

- `withdraw(double amount)` method:
  - Throws `InsufficientBalanceException` if withdrawal amount exceeds balance.
  - Throws `IllegalArgumentException` if the amount is negative.
- Handle exceptions in `main()`.

Expected Behavior:

- If valid, print `"Withdrawal successful, new balance: X"`.
- If balance is insufficient, throw and handle `"Insufficient balance!"`.
- If the amount is negative, throw and handle `"Invalid amount!"`.

```
import java.util.*;

class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String msg) {
        super(msg);
    }
}

public class Bank {
    static double balance = 5000;

    static void withdraw(double amount) throws
InsufficientBalanceException {
        if (amount < 0) {
            throw new IllegalArgumentException("Invalid amount!");
        } else if (amount > balance) {
            throw new InsufficientBalanceException("Insufficient
balance!");
        }
        balance -= amount;
        System.out.println("Withdrawal successful, new balance: " +
balance);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.print("Enter withdrawal amount: ");
            double amount = sc.nextDouble();
            withdraw(amount);
        } catch (InsufficientBalanceException e) {
            System.out.println(e.getMessage());
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```