DEFENSIVE PROGRAMMING

Laporan ini disusun untuk memenuhi tugas mata kuliah Teknik Pemrograman

Oleh:

Zahra Hilyatul Jannah

231524031

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

PROGRAM STUDI SARJANA TERAPAN TEKNIK INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2024

Daftar Isi

Daftar Gambar

Case 1 : Exceptions Aren't Always Errors

Pada saat menjalankan program CountLetters dan memasukkan lebih dari satu frasa (memakai spasi atau tanda baca), program akan error dan menampilkan ArrayIndexOutOfBoundsException. Untuk menangani hal tersebut program dimodifikasi dengan menggunakan blok “try” dan “catch”.

```java
package case1;
//CountLetters.java

import java.util.Scanner;

public class CountLetters {
    public static void main(String[] args) {
        int[] counts = new int[26];
        Scanner scan = new Scanner(System.in);

        //get word from user
        System.out.print("Enter a single word (letters only, please): ");
        String word = scan.nextLine();

        //convert to all upper case
        word = word.toUpperCase();

        //count frequency of each letter in string
        for (int i = 0; i < word.length(); i++) {
            try {
                counts[word.charAt(i) - 'A']++;
            } catch (ArrayIndexOutOfBoundsException e) {
                // Pada langkah pertama, blok catch masih dikosongkan
            }
        }
        //print frequencies
        System.out.println();
        for (int i = 0; i < counts.length; i++) {
            if (counts[i] != 0) {
                System.out.println((char) (i + 'A') + ": " + counts[i]);
            }
        }
    }
}

```

Gambar 1 Source Code CountLetter.java 1

Pada langkah pertama program dimodifikasi dengan memasukkan badan loop pertama dalam blok “try” dan menambahkan “catch” yang menangkap *exceptions* tanpa melakukan apapun. Output dari program di atas sebagai berikut

```
Enter a single word (letters only, please): Hallo, Zahra!

A: 3
H: 2
L: 2
O: 1
R: 1
Z: 1
```

Gambar 2 Output CountLetter.java 1

Pada modifikasi kedua, badan blok “catch” dimodifikasi untuk mencetak pesan “Not a letter: ” diikuti dengan karakter yang menyebabkan masalah. Berikut merupakan kode program dan hasil outputnya

```java
package case1;
//CountLetters.java

import java.util.Scanner;

public class CountLetters {
    public static void main(String[] args) {
        int[] counts = new int[26];
        Scanner scan = new Scanner(System.in);
        //get word from user
        System.out.print("Enter a single word (letters only, please): ");
        String word = scan.nextLine();
        //convert to all upper case
        word = word.toUpperCase();
        //count frequency of each letter in string
        for (int i = 0; i < word.length(); i++) {
            try {
                counts[word.charAt(i) - 'A']++;
            } catch (ArrayIndexOutOfBoundsException e) {
                // Pada langkah pertama, blok catch masih dikosongkan
                // Pada langkah kedua, badan blok "catch" ditambahkan menambah pesan "Not a letter:
                // diikuti dengan karakter yang membuat indeks diluar batas
                System.out.println("Not a letter: " + word.charAt(i));
            }
        }
        //print frequencies
        System.out.println();
        for (int i = 0; i < counts.length; i++) {
            if (counts[i] != 0) {
                System.out.println((char) (i + 'A') + ": " + counts[i]);
            }
        }
    }
}
```

Gambar 3 Source Code CountLetter.java 2

```
Enter a single word (letters only, please): Hello, Zahra!
Not a letter: ,
Not a letter:
Not a letter: !

A: 2
E: 1
H: 2
L: 2
O: 1
R: 1
Z: 1
```

Gambar 4 Output CountLetter.java 2

Pada hasil modifikasi pertama, program akan tetap berjalan dan menangani karakter non-huruf tanpa memunculkan pesan kesalahan yang mengganggu. Akan tetapi, dengan mencetak karakter yang menyebabkan *exceptions*, kita dapat melihat karakter mana yang menyebabkan masalahnya dan mengoreksi masalahnya. Ini dapat meningkatkan pengalaman kita karena program memberikan informasi yang jelas dan memungkinkan kita untuk memperbaiki kesalahan dengan lebih baik.

Case 2: Placing Exception Handlers

Pada kasus kedua, program dimodifikasi dengan menambahkan pernyataan “try” yang mencakup seluruh loop “while”. “try” diletakkan sebelum “while” dan “catch” setelah badan loop.

```java
package case1;

//ParseInts.java

import java.util.Scanner;

public class ParseInts {
    public static void main(String[] args) {
        int sum = 0;
        Scanner scan = new Scanner(System.in);
        String line;
        System.out.println("Enter a line of text");
        line = scan.nextLine();
        Scanner scanLine = new Scanner(line);

        try {
            while (scanLine.hasNext()) {
                int val = Integer.parseInt(scanLine.next());
                sum += val;
            }
        } catch (NumberFormatException e) {
            // mengosongkan blok catch
        }

        System.out.println("The sum of the integers on this line is " + sum);
    }
}
```

Gambar 5 Source Code ParseInts.java 1

```
Enter a line of text
10 dan 20
The sum of the integers on this line is 10
```

Gambar 6 Output ParseInts.java 1

Pada program di atas, program akan berhenti menjumlahkan begitu bertemu dengan yang bukan merupakan bilangan bulat. Sehingga pada modifikasi selanjutnya, blok try dan catch dipindahkan ke dalam loop menjadi seperti pada gambar di bawah.

```java
package case1;

//ParseInts.java

import java.util.Scanner;

public class ParseInts {
    public static void main(String[] args) {
        int sum = 0;
        Scanner scan = new Scanner(System.in);
        String line;
        System.out.println("Enter a line of text");
        line = scan.nextLine();
        Scanner scanLine = new Scanner(line);

        while (scanLine.hasNext()) {
            try {
                int val = Integer.parseInt(scanLine.next());
                sum += val;
            } catch (NumberFormatException e) {
                // mengosongkan blok catch
            }
        }

        System.out.println("The sum of the integers on this line is " + sum);
    }
}
```

Gambar 7 Source Code ParseInts.java 2

```
Enter a line of text
10 20 dan 30
The sum of the integers on this line is 60
```

Gambar 8 Output ParseInts.java 2

Dengan menempatkan blok try dan catch di dalam loop, kita memastikan bahwa exception tidak menghentikan eksekusi loop. Ini memungkinkan program untuk terus memproses instruksi berikutnya dalam loop setelah menangkap exception, sehingga seluruh instruksi dapat diproses dengan benar. Dengan cara ini, program dapat beroperasi lebih baik dan menangani input yang beragam dengan lebih baik.

Case 3: Throwing Exceptions

Pada langkah pertama kasus 3, metode factorials pada class MathUtils dimodifikasi agar memeriksa nilai parameter. Jika nilai paramter negatif, maka program akan melemparkan “IllegalArgumentException” dengan pesan yang memberikan penjelasan masalahnya. Hal tersebut disebabkan karena faktorial tidak didefinisikan untuk bilangan negatif sehingga menghitung faktorial dari bilangan negatif tidaklah mungkin. Oleh karena itu, lebih baik untuk menghentikan eksekusi program dan memberikan pesan yang jelas kepada pengguna bahwa input tidak valid. Dengan melemparkan IllegalArgumentException, kita memberikan pesan yang jelas tentang masalahnya dan mencegah program melanjutkan eksekusi dengan hasil yang tidak valid.

```
public class MathUtils {
    //-----------------------------------------------------------
    // Returns the factorial of the argument given
    //-----------------------------------------------------------
    public static int factorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Factorial is not defined for negative numbers.");
        }

        int fac = 1;
        for (int i = n; i > 0; i--) {
            fac *= i;
        }
        return fac;
    }
}
```

Gambar 9 Source Code MathUtils.java 2

```
Enter an integer: -5
Factorial is not defined for negative numbers.
Another factorial? (y/n)
```

Gambar 10 Output Factorials.java 1

Pada langkah kedua, program yang sama dimodifikasi dengan menambahkan baris code untuk memeriksa parameter yang lebih dari 16.

```java
public class MathUtils {
    //-------------------------------------------------------------
    // Returns the factorial of the argument given
    //-------------------------------------------------------------
    public static int factorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Factorial is not defined for negative numbers.");
        }

        if (n > 16) {
            throw new IllegalArgumentException("Factorial is too large to be represented by an int.");
        }

        int fac = 1;
        for (int i = n; i > 0; i--) {
            fac *= i;
        }
        return fac;
    }
}
```

Gambar 11 Source Code MathUtils.java 2

```
Enter an integer: 17
Factorial is too large to be represented by an int.
Another factorial? (y/n)
```

Gambar 12 Output Factorials.java 2

Faktorial dari bilangan yang besar (>16) dapat menghasilkan nilai yang sangat besar, yang mungkin melampaui rentang yang dapat direpresentasikan oleh tipe data int. Oleh karena itu, dengan melemparkan IllegalArgumentException untuk kasus overflow, kita memastikan bahwa program berhenti eksekusi saat terjadi situasi yang tidak dapat ditangani.