TUGAS

“Exception”

Dibuat untuk Memenuhi Tugas Mata Kuliah Pemrograman Berbasis Objek (PBO) Teori

Disusun oleh

Sifa Wafiqna
231511092

2C-JTK

PROGRAM DIPLOMA III TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2024

1. **Perbedaan antara checked exception dan unchecked exception**

checked exception	unchecked exception
If a client can reasonably be expected to recover from an exception	If a client cannot do anything to recover from the exception,
Pengecekan saat compile time	Pengecekan saat runtime
Wajib ditangani atau di deklarasikan programmer menggunakan *throws* keyword atau blok try-catch	Penanganan opsional/Tidak wajib di tangani programmer
Biasanya error berasal dari sumber eksternal. (seperti: operasi file tidak ditemukan, operasi I/O, koneksi database)	Error sering kali terjadi karena kesalahan logika proram/dalam kode (seperti: mengakses elemen di luar batas array, membagi angka dengan 0, memanggil method objek nilai null)

2. **Tuliskan contoh implementasi JAVA untuk checked exception ?**

code > CheckedExceptionExample.java > ...

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class CheckedExceptionExample {

    // Metode ini harus mendeklarasikan "throws IOException" karena FileReader dan BufferedReader
    // dapat melempar checked exception, yaitu IOException.
    public static void readFile(String fileName) throws IOException {
        // Membuka file dengan FileReader (bisa melempar FileNotFoundException, turunan dari IOException)
        FileReader fileReader = new FileReader(fileName);

        // Membungkus FileReader dengan BufferedReader untuk membaca file baris demi baris
        BufferedReader bufferedReader = new BufferedReader(fileReader);

        String line;
        // Membaca file baris demi baris, ini juga bisa melempar IOException
        while ((line = bufferedReader.readLine()) != null) {
            System.out.println(line);
        }

        // Jangan lupa untuk menutup file setelah selesai
        bufferedReader.close();
    }

    Run | Debug
    public static void main(String[] args) {
        String filePath = "data.txt";

        try {
            // Memanggil metode readFile dan menangani checked exception di sini
            readFile(filePath);
        } catch (IOException e) {
            // Jika terjadi IOException (misalnya file tidak ditemukan), eksekusi akan masuk ke sini
            System.out.println("Terjadi kesalahan saat membaca file: " + e.getMessage());
        }
    }
}
```

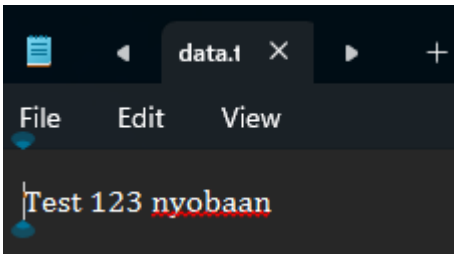
Penjelasan:

1. throws IOException: mendeklarasikan bahwa metode readFile bisa melempar IOException. Hal ini diperlukan karena operasi membaca file melibatkan checked exception.
2. try-catch Block: metode readFile dipanggil di dalam metode main. Karena metode ini melempar IOException, kita harus menangani pengecualian tersebut dengan blok try-catch. Jika file tidak ditemukan atau ada masalah membaca file, Java akan melempar IOException, dan pesan kesalahan akan dicetak di blok catch.
3. FileReader dan BufferedReader: Kelas FileReader digunakan untuk membuka file, dan BufferedReader digunakan untuk membaca isi file baris demi baris. Kedua kelas ini bisa melempar IOException saat ada masalah dengan operasi file, sehingga perlu penanganan.

Output saat file data.txt belum di buat

```
PS D:\MATKUL\semester3\pbot\code> javac CheckedExceptionExample.java
PS D:\MATKUL\semester3\pbot\code> java CheckedExceptionExample.java
Terjadi kesalahan saat membaca file: data.txt (The system cannot find the file specified)
```

Percobaan dengan membuat file data.txt



Output yang ditampilkan setelah file data.txt dibuat

```
PS D:\MATKUL\semester3\pbot\code> java CheckedExceptionExample.java
Test 123 nyobaan
PS D:\MATKUL\semester3\pbot\code>
```

3. **Tuliskan contoh implementasi JAVA untuk uncheked exception ?**
   Terdapat 2 opsi untuk contoh implementasi program unchecked exception yaitu dengan memakai try cacth (opsional) dan tidak memakai hanya menampilkan pesan error otomatis saja. Berikut contoh program nya yang memakai try catch dan outputnya:

code > UncheckedExceptionExample.java

```java
public class UncheckedExceptionExample {
    Run | Debug
    public static void main(String[] args) {
        try {
            int result = divide(a:10, b:0);
            System.out.println("Hasil: " + result);
        } catch (ArithmeticException e) {
            System.out.println(x:"Terjadi pengecualian: Pembagian dengan nol tidak diizinkan!");
        }
    }

    public static int divide(int a, int b) {
        return a / b;  // Potensi ArithmeticException
    }
}
```

```
PS D:\MATKUL\semester3\pbot\code> java UncheckedExceptionExample.java
Terjadi pengecualian: Pembagian dengan nol tidak diizinkan!
PS D:\MATKUL\semester3\pbot\code>
```

Berikut contoh program dan outputnya yang tidak menggunakan try catch

code > UncheckedExceptionExampleWithoutTryCatch.java > ...

```java
public class UncheckedExceptionExampleWithoutTryCatch {

    Run | Debug
    public static void main(String[] args) {
        // Unchecked exception: ArithmeticException (pembagian dengan nol)
        int result = divide(a:10, b:0);  // Tidak ada try-catch, akan crash saat runtime
        System.out.println("Hasil: " + result);

        // Unchecked exception: ArrayIndexOutOfBoundsException (akses elemen di luar batas array)
        int[] numbers = {1, 2, 3};
        System.out.println("Angka pada indeks ke-3: " + numbers[3]);  // Akan error saat runtime
    }

    // Method untuk pembagian
    public static int divide(int a, int b) {
        return a / b;  // Berpotensi melempar ArithmeticException jika b adalah nol
    }
}
```

```
PS D:\MATKUL\semester3\pbot\code> java UncheckedExceptionExampleWithoutTryCatch.java
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at UncheckedExceptionExampleWithoutTryCatch.divide(UncheckedExceptionExampleWithoutTryCatch.java:15)
        at UncheckedExceptionExampleWithoutTryCatch.main(UncheckedExceptionExampleWithoutTryCatch.java:5)
PS D:\MATKUL\semester3\pbot\code>
```

4. **Buatlah sebuah contoh class exception yang anda definisikan sendiri (user defined exception) ?**

code > J UserDefinedExceptionExample.java > ...

```java
// Custom exception yang mewarisi dari Exception (checked exception)
class InvalidAgeException extends Exception {

    // Konstruktor dengan pesan kesalahan
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class UserDefinedExceptionExample
{
    // Method untuk memeriksa usia
    public static void checkAge(int age) throws InvalidAgeException
    {
        if (age < 18) {
            // Melempar custom exception jika usia kurang dari 18
            throw new InvalidAgeException(message:"Usia tidak valid! Usia minimal untuk mendaftar adalah 18 tahun.");
        } else {
            System.out.println("Pendaftaran berhasil. Usia: " + age);
        }
    }

    Run | Debug
    public static void main(String[] args)
    {
        try
        {
            // Uji dengan usia valid
            checkAge(age:20);  // Tidak ada pengecualian, akan mencetak pesan sukses

            // Uji dengan usia tidak valid
            checkAge(age:15);  // Akan melempar InvalidAgeException
        }
        catch (InvalidAgeException e)
        {
            // Menangani custom exception
            System.out.println("Terjadi pengecualian: " + e.getMessage());
        }
    }
}
```

Penjelasan Program:

1. Kelas InvalidAgeException adalah user-defined exception yang dibuat dengan mewarisi kelas Exception. Konstruktornya menerima pesan kesalahan dan meneruskannya ke kelas induk Exception.
2. Method validateAge menerima parameter age dan memvalidasi apakah usia tersebut di bawah 18 tahun. Jika iya, maka method akan melempar InvalidAgeException dengan pesan kesalahan.
3. Blok try-catch menangkap pengecualian yang dilempar oleh validateAge. Jika pengecualian terjadi, pesan kesalahan akan ditampilkan.

5. **Buatlah kendala dan lesson learnednya dari tugas ini**

   Kendala:

   1. Awalnya, sulit memahami perbedaan antara checked dan unchecked exceptions, terutama dalam konteks kapan perlu menggunakan try-catch dan kapan pengecualian tersebut tidak dipaksa untuk ditangani.
   2. Membuat custom exception pertama kali membutuhkan waktu untuk memahami bagaimana inheritance bekerja dalam hierarki exception di Java.
   3. Pengujian error handling: Kesulitan muncul saat mencoba menguji berbagai skenario tanpa try-catch, terutama untuk memastikan program gagal dengan cara yang diprediksi dan bagaimana penanganan pengecualian mempengaruhi jalannya program.

   Lesson Learned:

   1. Pentingnya Exception Handling: Exception handling sangat penting untuk mencegah program crash saat runtime, khususnya pada checked exceptions yang wajib ditangani agar program lebih aman dan andal.
   2. Fleksibilitas Unchecked Exception: Meski unchecked exceptions tidak harus ditangani, lebih baik tetap menerapkan try-catch untuk meningkatkan stabilitas program, terutama di bagian-bagian yang berpotensi menyebabkan kesalahan runtime.
   3. Custom Exception: Membuat custom exception memberi fleksibilitas untuk menangani kondisi-kondisi spesifik dalam aplikasi, memungkinkan kita memberi pesan kesalahan yang lebih deskriptif dan relevan bagi pengguna.
   4. Perencanaan Pengecualian: Saat menulis program, penting merencanakan bagian mana yang mungkin menyebabkan error dan memastikan pengecualian tersebut ditangani dengan baik, baik itu checked atau unchecked exceptions.