MAKALAH INTERFACE & EXCEPTION DALAM

BAHASA JAVA



Oleh :

Diki Candra

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

DAFTAR ISI

[BAB I 2](#_Toc137902532)

[PENDAHULUAN 2](#_Toc137902533)

[A. Latar Belakang 2](#_Toc137902534)

[B. Tujuan dan Ruang Lingkup 2](#_Toc137902535)

[C. Metodologi 3](#_Toc137902536)

[BAB II 5](#_Toc137902537)

[PENGENALAN INTERFACE DALAM JAVA 5](#_Toc137902538)

[A. Definisi Interface 5](#_Toc137902539)

[B. Perbedaan Antara Class dan Interface 7](#_Toc137902540)

[C. Mendeklarasikan Interface 8](#_Toc137902541)

[D. Implementasi Interface 10](#_Toc137902542)

[E. Keuntungan Penggunaan Interface 11](#_Toc137902543)

[BAB III 14](#_Toc137902544)

[MEMAHAMI EXCEPTION dalam JAVA 14](#_Toc137902545)

[A. Pengertian Exception 14](#_Toc137902546)

[B. Tipe-tipe Exception 15](#_Toc137902547)

[C. Mekanisme Exception Handling 16](#_Toc137902548)

[D. Blok Try-Catch-Finally 18](#_Toc137902549)

[E. Pernyataan throw dan throws 21](#_Toc137902550)

[F. Membuat Exception Kustom 22](#_Toc137902551)

[BAB IV 25](#_Toc137902552)

[PENGGUNAAN 25](#_Toc137902553)

[INTERFACE dalam EXCEPTION HEADING 25](#_Toc137902554)

[A. Menggunakan Interface sebagai Tipe Parameter 25](#_Toc137902555)

[B. Menangani Exception dengan Interface 26](#_Toc137902556)

[C. Contoh Implementasi Interface dalam Exception Handling 28](#_Toc137902557)

[BAB V 31](#_Toc137902558)

[KESIMPULAN 31](#_Toc137902559)

[A. Ringkasan 31](#_Toc137902560)

[B. Manfaat dan Kelemahan Penggunaan Interface dan Exception Handling 32](#_Toc137902561)

[DAFTAR PUSTAKA 35](#_Toc137902562)

# BAB I

# PENDAHULUAN

## A. Latar Belakang

membahas peran penting dari interface dan exception dalam bahasa pemrograman Java. Java adalah salah satu bahasa pemrograman yang sangat populer dan digunakan secara luas dalam pengembangan perangkat lunak, terutama dalam pembuatan aplikasi berbasis objek.

Interface adalah salah satu konsep utama dalam pemrograman berorientasi objek. Ini memungkinkan pengembang untuk mendefinisikan kontrak antara kelas-kelas yang berbeda, sehingga memungkinkan komunikasi dan interaksi yang efektif antara objek-objek tersebut. Melalui penggunaan interface, kita dapat mencapai abstraksi yang lebih tinggi dan polimorfisme dalam kode kita.

Exception handling (penanganan pengecualian) juga merupakan bagian penting dalam pemrograman Java. Saat menjalankan program, kita mungkin akan menghadapi situasi yang tidak terduga atau kesalahan yang dapat terjadi, seperti kesalahan pembagian oleh nol atau kesalahan saat membaca file. Dalam situasi-situasi tersebut, exception handling memungkinkan kita untuk mengenali, menangani, dan merespons kesalahan tersebut dengan cara yang terstruktur, sehingga program kita dapat berjalan dengan lebih andal dan terhindar dari kegagalan.

Mengerti konsep dan penggunaan yang benar dari interface dan exception handling dalam bahasa Java sangat penting bagi pengembang perangkat lunak. Makalah ini bertujuan untuk memberikan pemahaman yang komprehensif tentang konsep-konsep ini, serta memberikan contoh-contoh implementasi yang berguna. Dengan memahami latar belakang dan konsep ini, pembaca akan dapat memanfaatkan interface dan exception handling dengan lebih efektif dalam pengembangan aplikasi Java mereka.

## B. Tujuan dan Ruang Lingkup

Tujuan Makalah:

1. Memberikan pemahaman yang mendalam tentang konsep dan penggunaan interface dalam bahasa pemrograman Java.
2. Menjelaskan perbedaan antara class dan interface, serta manfaat yang diperoleh dengan menggunakan interface.
3. Membahas tipe-tipe exception dalam Java dan bagaimana penanganan exception dilakukan.
4. Menggambarkan mekanisme exception handling dalam Java, termasuk penggunaan blok try-catch-finally.
5. Menjelaskan cara menggunakan interface dalam penanganan exception, serta memberikan contoh implementasi yang relevan.
6. Memberikan studi kasus dan contoh implementasi nyata dari penggunaan interface dan exception handling dalam aplikasi Java.
7. Membahas manfaat dan kelemahan penggunaan interface dan exception handling dalam pengembangan perangkat lunak Java.
8. Memberikan rekomendasi untuk pengembangan selanjutnya dalam penggunaan interface dan exception handling.

Ruang Lingkup Makalah:

1. Makalah ini akan fokus pada konsep interface dan exception handling dalam bahasa pemrograman Java.
2. Pemaparan akan mencakup penjelasan mendalam tentang konsep interface, termasuk deklarasi, implementasi, dan manfaatnya.
3. Makalah akan membahas tipe-tipe exception yang ada dalam Java, serta cara menangani dan meresponsnya.
4. Mekanisme exception handling, seperti blok try-catch-finally, juga akan dijelaskan secara rinci.
5. Penjelasan tentang penggunaan interface dalam penanganan exception akan dicakup, dengan contoh-contoh implementasi yang relevan.
6. Studi kasus dan contoh aplikasi nyata akan disajikan untuk memberikan pemahaman yang lebih baik tentang penggunaan interface dan exception handling dalam konteks nyata.
7. Manfaat dan kelemahan penggunaan interface dan exception handling dalam pengembangan perangkat lunak akan dibahas secara objektif.
8. Rekomendasi untuk pengembangan selanjutnya dalam penggunaan interface dan exception handling akan diberikan, jika relevan.

Penting untuk dicatat bahwa ruang lingkup ini dapat disesuaikan dengan kebutuhan dan batasan makalah yang ditentukan.

## C. Metodologi

Metodologi yang dapat digunakan dalam makalah ini adalah sebagai berikut:

1. Studi Pustaka: Dilakukan penelitian mendalam tentang konsep interface dan exception handling dalam bahasa pemrograman Java. Mengacu pada sumber-sumber teks, buku, artikel, dan dokumentasi resmi Java untuk memahami konsep-konsep tersebut secara komprehensif.
2. Analisis Konsep: Menganalisis konsep-konsep dasar seperti interface, class, objek, inheritance, polimorfisme, dan abstraksi. Menggali lebih dalam tentang peran interface dalam pengembangan aplikasi Java dan bagaimana kontrak antara kelas-kelas dapat ditentukan melalui interface. Selain itu, menganalisis konsep exception, tipe-tipe exception, dan mekanisme exception handling dalam Java.
3. Contoh Implementasi: Memberikan contoh-contoh implementasi untuk menggambarkan penggunaan interface dan exception handling dalam kode Java. Menggunakan contoh sederhana hingga kompleks untuk mengilustrasikan penggunaan interface sebagai tipe parameter, penanganan exception menggunakan blok try-catch-finally, serta contoh penggunaan interface dalam penanganan exception.
4. Studi Kasus: Memberikan studi kasus dan contoh aplikasi nyata yang menggunakan interface dan exception handling dalam pengembangan aplikasi Java. Menggambarkan bagaimana interface dapat digunakan untuk mempermudah pengembangan aplikasi yang kompleks dan bagaimana exception handling dapat membantu dalam mengatasi situasi yang tidak terduga dalam aplikasi.
5. Analisis Manfaat dan Kelemahan: Menganalisis manfaat dan kelemahan penggunaan interface dan exception handling dalam pengembangan perangkat lunak Java. Menyajikan argumen dan perbandingan yang objektif untuk mendapatkan pemahaman yang lebih lengkap tentang kapan dan bagaimana menggunakan konsep-konsep ini dengan tepat.
6. Kesimpulan dan Rekomendasi: Menyimpulkan makalah dengan merangkum poin-poin penting yang dibahas. Memberikan rekomendasi untuk pengembangan selanjutnya dalam penggunaan interface dan exception handling, serta saran untuk memaksimalkan manfaat dari kedua konsep ini dalam pengembangan aplikasi Java.

Metodologi di atas dapat disesuaikan dan dikembangkan lebih lanjut sesuai dengan kebutuhan dan tujuan makalah yang lebih spesifik.

# BAB II

# PENGENALAN INTERFACE DALAM JAVA

## A. Definisi Interface

Interface dalam pemrograman Java adalah suatu kontrak atau spesifikasi yang menggambarkan perilaku yang harus diikuti oleh kelas-kelas yang mengimplementasikannya. Secara sederhana, interface adalah himpunan metode yang tidak memiliki implementasi konkret, tetapi hanya mendefinisikan tanda tangan metode (nama, parameter, tipe kembalian) dan konstanta (variabel yang bernilai tetap). Interface menyediakan sebuah antarmuka yang memungkinkan komunikasi dan interaksi antara objek-objek yang berbeda.

Interface mendefinisikan apa yang harus dilakukan oleh suatu kelas yang mengimplementasikannya, tetapi tidak mengatur bagaimana implementasinya dilakukan. Ini memungkinkan pengembang untuk memisahkan antara definisi dan implementasi metode, sehingga menciptakan kelas yang lebih fleksibel dan modular.

Secara sintaksis, sebuah interface dideklarasikan dengan menggunakan kata kunci "interface" diikuti oleh nama interface dan blok kode yang berisi tanda tangan metode dan konstanta. Contoh deklarasi interface dalam Java adalah sebagai berikut:

public interface Drawable {

void draw();

double calculateArea();

String getName();

int MAX\_WIDTH = 800;

int MAX\_HEIGHT = 600;

}

Dalam contoh di atas, interface "Drawable" mendefinisikan tiga metode tanpa implementasi konkret yaitu "draw()", "calculateArea()", dan "getName()". Interface juga menyediakan dua konstanta "MAX\_WIDTH" dan "MAX\_HEIGHT" yang memiliki nilai tetap.

Ketika sebuah kelas mengimplementasikan sebuah interface, itu harus menyediakan implementasi konkret untuk semua metode yang didefinisikan dalam interface tersebut. Hal ini dicapai dengan menggunakan kata kunci "implements" diikuti oleh nama interface yang ingin diimplementasikan. Contoh implementasi interface dalam kelas Java adalah sebagai berikut:

public class Rectangle implements Drawable {

private double width;

private double height;

// Implementasi metode dari interface Drawable

public void draw() {

// Implementasi logika menggambar persegi panjang

}

public double calculateArea() {

// Implementasi logika menghitung luas persegi panjang

}

public String getName() {

// Implementasi logika mengambil nama

}

// Metode lain dari kelas Rectangle

// ...

}

Dalam contoh di atas, kelas "Rectangle" mengimplementasikan interface "Drawable" dengan memberikan implementasi konkret untuk metode "draw()", "calculateArea()", dan "getName()".

## B. Perbedaan Antara Class dan Interface

Perbedaan antara class dan interface dalam pemrograman Java adalah sebagai berikut:

1. Implementasi Metode:

Sebuah class dapat memiliki implementasi konkret untuk metode yang didefinisikan di dalamnya, sedangkan sebuah interface hanya mendefinisikan tanda tangan metode tanpa implementasi konkret. Class dapat memiliki metode yang dapat langsung dieksekusi, sementara interface hanya menyediakan kontrak tentang metode yang harus diimplementasikan oleh kelas-kelas yang mengimplementasikannya.

1. Inheritance:

Sebuah class dapat mewarisi atribut dan metode dari class lain menggunakan pewarisan (inheritance). Namun, sebuah class hanya dapat mewarisi satu class lain secara langsung. Sedangkan sebuah interface dapat diimplementasikan oleh banyak class dan dapat menerapkan multiple inheritance, yang berarti sebuah class dapat mengimplementasikan beberapa interface sekaligus.

1. Constructor:

Sebuah class dapat memiliki constructor (konstruktor) yang digunakan untuk membuat objek kelas tersebut. Namun, sebuah interface tidak dapat memiliki constructor, karena interface tidak dapat digunakan untuk membuat objek secara langsung.

1. Variabel:

Sebuah class dapat memiliki variabel instance dan variabel static yang dapat menyimpan nilai dan keadaan objek. Sebuah interface hanya dapat memiliki variabel static final (konstanta) yang bernilai tetap dan bersifat publik (public static final).

1. Instantiasi Objek:

Sebuah class dapat diinstantiasi menjadi objek yang dapat digunakan dalam program. Sementara itu, sebuah interface tidak dapat diinstantiasi menjadi objek, karena interface hanya berisi definisi metode dan konstanta.

1. Visibilitas:

Metode dan variabel dalam class dapat memiliki tingkat visibilitas (visibility) seperti public, private, atau protected, yang mengatur aksesibilitas dari luar class. Di sisi lain, metode dalam interface secara default bersifat publik (public) dan tidak dapat diberikan tingkat visibilitas lain.

1. Tujuan:

Class digunakan untuk membuat blueprint atau template untuk objek, sedangkan interface digunakan sebagai kontrak yang menentukan perilaku yang harus diikuti oleh class yang mengimplementasikannya.

Perbedaan-perbedaan ini mempengaruhi penggunaan dan desain kelas dan interface dalam pemrograman Java. Class biasanya digunakan untuk mewakili objek konkret dengan atribut dan perilaku tertentu, sementara interface digunakan untuk menyediakan kontrak dan memfasilitasi polimorfisme dan modularitas dalam kode.

## C. Mendeklarasikan Interface

Deklarasi interface dalam bahasa pemrograman Java melibatkan penggunaan kata kunci "interface" diikuti oleh nama interface dan blok kode yang berisi tanda tangan metode dan konstanta. Berikut adalah contoh sintaksis untuk mendeklarasikan sebuah interface:

public interface NamaInterface {

// Deklarasi metode

tipeKembalian namaMetode1(parameter1, parameter2, ...);

tipeKembalian namaMetode2(parameter1, parameter2, ...);

// Deklarasi konstanta

tipeData namaKonstanta1 = nilaiKonstanta1;

tipeData namaKonstanta2 = nilaiKonstanta2;

// ...

}

Dalam deklarasi interface di atas:

* "NamaInterface" adalah nama yang diberikan untuk interface. Sebaiknya menggunakan kata benda yang menggambarkan entitas yang diwakili oleh interface tersebut.
* Di dalam blok kode interface, dideklarasikan tanda tangan metode yang harus diikuti oleh kelas-kelas yang mengimplementasikannya. Tanda tangan metode mencakup tipe kembalian (jika ada), nama metode, dan parameter yang diperlukan. Implementasi konkret metode tidak diberikan dalam interface, hanya tanda tangan metode yang ditentukan.
* Setelah deklarasi metode, kita dapat mendeklarasikan konstanta dalam interface. Konstanta adalah variabel dengan nilai tetap yang didefinisikan di dalam interface. Konstanta harus memiliki tingkat aksesibilitas publik (public) dan dinyatakan sebagai statis (static) dan final (final).
* Dalam deklarasi interface, metode dan konstanta tidak memerlukan kata kunci "public" karena secara default semua metode dan konstanta dalam interface bersifat publik.

Contoh deklarasi interface dalam Java:

public interface Drawable {

void draw();

double calculateArea();

int MAX\_WIDTH = 800;

int MAX\_HEIGHT = 600;

}

Dalam contoh di atas, interface "Drawable" mendeklarasikan dua metode, yaitu "draw()" dan "calculateArea()", yang harus diimplementasikan oleh kelas-kelas yang mengimplementasikannya. Juga, terdapat dua konstanta "MAX\_WIDTH" dan "MAX\_HEIGHT" yang dideklarasikan sebagai konstanta publik yang bersifat statis dan final.

Setelah sebuah interface dideklarasikan, interface tersebut dapat diimplementasikan oleh kelas-kelas lain menggunakan kata kunci "implements". Kelas-kelas yang mengimplementasikan interface harus memberikan implementasi konkret untuk semua metode yang didefinisikan dalam interface tersebut.

## D. Implementasi Interface

Implementasi interface dalam bahasa pemrograman Java melibatkan pembuatan kelas yang menggunakan kata kunci "implements" diikuti oleh nama interface yang ingin diimplementasikan. Implementasi ini memastikan bahwa kelas tersebut menyediakan implementasi konkret untuk semua metode yang didefinisikan dalam interface. Berikut adalah langkah-langkah untuk mengimplementasikan sebuah interface:

1. Deklarasikan kelas dengan menggunakan kata kunci "class" diikuti oleh nama kelas yang ingin diimplementasikan. Contoh: **public class MyClass implements MyInterface**.
2. Setelah deklarasi kelas, tambahkan kata kunci "implements" diikuti oleh nama interface yang ingin diimplementasikan. Jika ada beberapa interface yang ingin diimplementasikan, gunakan koma untuk memisahkan antara nama interface. Contoh: **public class MyClass implements MyInterface1, MyInterface2**.
3. Implementasikan semua metode yang didefinisikan dalam interface dengan menyediakan implementasi konkret untuk masing-masing metode. Pastikan tanda tangan metode yang diimplementasikan sama dengan yang didefinisikan dalam interface. Contoh:

public class MyClass implements MyInterface {

public void method1() {

// Implementasi metode method1

}

public void method2() {

// Implementasi metode method2

}

}

Dalam contoh di atas, kelas **MyClass** mengimplementasikan **MyInterface** dan memberikan implementasi konkret untuk metode **method1** dan **method2** yang didefinisikan dalam **MyInterface**.

1. Pastikan bahwa semua metode yang didefinisikan dalam interface telah diimplementasikan dengan benar di kelas yang mengimplementasikannya. Jika ada metode yang tidak diimplementasikan, akan terjadi error kompilasi.

Implementasi interface memungkinkan kelas-kelas untuk mematuhi kontrak yang diberikan oleh interface dan memastikan bahwa mereka memiliki perilaku yang diharapkan. Dengan mengimplementasikan interface, kelas dapat menggunakan polimorfisme untuk berinteraksi dengan objek-objek yang mengimplementasikan interface tersebut secara umum, tanpa harus mengetahui implementasi spesifik dari setiap kelas yang diimplementasikan.

## E. Keuntungan Penggunaan Interface

Penggunaan interface dalam pemrograman Java menawarkan beberapa keuntungan yang signifikan, di antaranya:

1. Abstraksi dan Pemisahan Kontrak: Interface memungkinkan pengembang untuk membuat abstraksi yang jelas dari perilaku yang diharapkan dari suatu objek. Dengan mendefinisikan tanda tangan metode dalam interface, interface memberikan kontrak yang jelas tentang metode yang harus diimplementasikan oleh kelas yang mengimplementasikannya. Ini memisahkan antara apa yang harus dilakukan oleh sebuah objek (interface) dan bagaimana hal itu dilakukan (implementasi di kelas-kelas).
2. Modularitas dan Reusabilitas: Interface mendukung prinsip modularitas dan reusabilitas dalam desain perangkat lunak. Dengan menggunakan interface, pengembang dapat membuat komponen yang terpisah dan independen yang dapat digunakan kembali dalam berbagai konteks. Interface juga memungkinkan untuk mengganti implementasi kelas yang satu dengan yang lain tanpa mengubah kode yang menggunakan interface tersebut.
3. Polimorfisme: Penggunaan interface memungkinkan penggunaan polimorfisme dalam program Java. Dengan menggunakan referensi interface, kita dapat memperlakukan objek-objek yang berbeda secara umum, tanpa harus bergantung pada implementasi spesifik dari setiap kelas. Hal ini memberikan fleksibilitas dan mengurangi ketergantungan pada implementasi spesifik.
4. Mendukung Multiple Inheritance: Java mendukung implementasi multiple interface, yang berarti suatu kelas dapat mengimplementasikan lebih dari satu interface sekaligus. Ini memungkinkan kelas untuk mewarisi perilaku dan kontrak dari beberapa sumber sekaligus, yang meningkatkan fleksibilitas dalam desain dan memungkinkan pengembang untuk menggabungkan fitur-fitur dari beberapa interface yang berbeda.
5. Kontrak yang Kuat: Interface memberikan sebuah kontrak yang kuat dan jelas antara pihak yang menggunakan interface (klien) dengan pihak yang mengimplementasikan interface (kelas-kelas). Hal ini memungkinkan tim pengembang yang berbeda untuk bekerja secara independen dalam mengimplementasikan kelas-kelas yang memenuhi kontrak interface yang telah ditetapkan.
6. Penggantian Fleksibel: Dengan menggunakan interface, kita dapat dengan mudah mengganti implementasi kelas yang satu dengan yang lain tanpa mengganggu klien yang menggunakannya. Jika ada kebutuhan untuk mengganti logika di balik suatu perilaku, kita hanya perlu mengimplementasikan ulang interface tersebut dengan logika baru tanpa mempengaruhi kode yang menggunakan interface tersebut.
7. Testing dan Debugging yang Lebih Mudah: Interface memungkinkan pengembang untuk menggunakan teknik pengujian dan debugging yang lebih mudah. Dengan menggunakan interface, kita dapat membuat stub atau mock object untuk mengisolasi dan menguji komponen perangkat lunak secara terpisah tanpa harus bergantung pada implementasi konkret dari kelas tersebut.

Penggunaan interface membantu dalam mencapai kode yang lebih modular, fleksibel, dan dapat dipelihara. Interface membantu dalam memisahkan perhatian antara klien dan implementasi, memungkinkan pengembang untuk berfokus pada Konsep interface memainkan peran penting dalam pemrograman berorientasi objek dengan memisahkan antara kontrak (interface) dan implementasi (kelas-kelas). Dengan menggunakan interface, kita dapat mencapai desain yang lebih fleksibel, modular, dan dapat diperluas. Beberapa keuntungan penggunaan interface yang telah dijelaskan sebelumnya, termasuk abstraksi yang baik, reusabilitas, polimorfisme, dan kemampuan untuk menggantikan implementasi dengan mudah.

Selain keuntungan-keuntungan tersebut, penggunaan interface juga memfasilitasi kolaborasi dalam tim pengembang. Interface memberikan dokumen yang jelas tentang kontrak yang harus dipenuhi oleh kelas-kelas yang mengimplementasikannya. Tim pengembang dapat bekerja secara terpisah dan paralel, dengan satu tim bertanggung jawab untuk mengimplementasikan interface dan tim lain menggunakan interface tersebut dalam kode mereka. Ini memungkinkan tim untuk saling bekerja mandiri dengan memastikan bahwa kelas-kelas yang diimplementasikan oleh satu tim akan sesuai dengan ekspektasi tim lain yang menggunakannya.

Selain itu, penggunaan interface juga memungkinkan pengembangan dengan pola desain seperti Dependency Injection (DI) dan Inversion of Control (IoC). Dalam pola-pola tersebut, interface digunakan sebagai kontrak yang diikuti oleh komponen-komponen yang terlibat dalam injeksi dependensi atau kontrol inversi. Hal ini memungkinkan fleksibilitas dalam penggantian komponen, pengujian unit yang lebih mudah, dan pengembangan kode yang lebih terpisah dan modular.

Namun, penting untuk diingat bahwa penggunaan interface haruslah sesuai dengan prinsip-prinsip desain yang baik. Terlalu banyak interface yang kompleks atau terlalu spesifik dapat menyulitkan pemahaman dan pemeliharaan kode. Selain itu, jika sebuah interface mengalami perubahan, maka semua kelas yang mengimplementasikannya juga harus diperbarui. Oleh karena itu, perlu mempertimbangkan dengan baik desain dan hierarki interface yang akan digunakan dalam proyek yang sedang dikerjakan.

Secara keseluruhan, penggunaan interface dalam pemrograman Java memberikan keuntungan signifikan dalam hal abstraksi, modularitas, reusabilitas, polimorfisme, dan fleksibilitas. Interface memungkinkan pemisahan antara kontrak dan implementasi, yang memungkinkan tim pengembang untuk bekerja secara independen, kolaboratif, dan efisien.

# BAB III

# MEMAHAMI EXCEPTION dalam JAVA

## A. Pengertian Exception

Dalam pemrograman, exception (atau pengecualian) merujuk pada situasi atau kondisi yang tidak biasa atau tidak terduga yang terjadi saat program sedang berjalan. Exception dapat terjadi ketika ada kesalahan dalam logika program, operasi yang tidak valid, kegagalan sumber daya eksternal, atau kondisi lain yang menyebabkan program tidak dapat melanjutkan eksekusi normalnya.

Secara umum, exception digunakan untuk menangani situasi yang tidak terduga dan mengubah alur eksekusi program. Ketika exception terjadi, program akan menghentikan eksekusinya pada titik tersebut dan mencari penanganan exception yang sesuai.

Pada pemrograman Java, exception diwakili oleh objek dari kelas-kelas yang turunan dari kelas Exception atau RuntimeException. Ketika sebuah exception terjadi, suatu objek exception dibuat dan dilemparkan (thrown) oleh program. Program kemudian mencari blok penanganan exception yang sesuai (exception handling) untuk mengatasi atau merespons exception tersebut.

Exception handling terdiri dari dua bagian utama:

1. Melempar (Throwing) Exception: Ketika suatu exception terjadi dalam program, kita menggunakan kata kunci "throw" untuk melemparkan (throw) exception tersebut. Ini menghasilkan pembuatan objek exception dan menghentikan alur eksekusi normal program. Contohnya seperti berikut:

throw new Exception("Pesan exception");

1. Menangani (Handling) Exception: Untuk menangani (handle) exception, kita menggunakan blok try-catch. Di dalam blok try, kita menempatkan kode yang mungkin menyebabkan exception terjadi. Di dalam blok catch, kita menangkap dan merespons exception yang terjadi dengan melakukan tindakan tertentu. Contohnya seperti berikut:

try {

// Kode yang mungkin menyebabkan exception

} catch (ExceptionType exception) {

// Penanganan exception

}

Dalam blok catch, kita menentukan tipe exception yang ingin ditangkap. Jika exception dengan tipe tersebut terjadi, blok catch akan dieksekusi dan penanganan exception dilakukan di dalamnya.

Penggunaan exception memungkinkan program untuk merespons kondisi-kondisi yang tidak terduga dengan cara yang terkontrol dan memberikan pesan yang informatif tentang kesalahan atau situasi yang terjadi. Ini membantu dalam penanganan kesalahan, pemulihan dari kegagalan, dan menjaga kestabilan program secara keseluruhan.

## B. Tipe-tipe Exception

Dalam pemrograman Java, exception dikelompokkan ke dalam beberapa tipe berdasarkan hierarki kelas-kelas exception yang ada. Berikut ini adalah beberapa tipe exception yang umum digunakan:

1. Checked Exception: Exception yang harus diperiksa (checked) oleh kompiler pada saat kompilasi. Ini adalah exception yang diharapkan terjadi dan harus ditangani oleh program. Jika kita menggunakan metode yang melempar checked exception, kita harus menangani exception tersebut atau mendeklarasikan penanganan exception tersebut dengan kata kunci **throws**. Beberapa contoh checked exception adalah:
   * **IOException**: Terjadi ketika ada masalah dengan operasi I/O.
   * **SQLException**: Terjadi ketika ada kesalahan dalam operasi database.
   * **ClassNotFoundException**: Terjadi ketika kelas yang diminta tidak ditemukan.
2. Unchecked Exception (RuntimeException): Exception yang tidak perlu diperiksa (unchecked) oleh kompiler pada saat kompilasi. Unchecked exception biasanya terjadi karena kesalahan dalam logika program atau situasi yang tidak terduga. Unchecked exception tidak harus ditangani secara paksa oleh program, tetapi sangat dianjurkan untuk menanganinya. Beberapa contoh unchecked exception adalah:
   * **NullPointerException**: Terjadi ketika terdapat penggunaan objek yang bernilai **null**.
   * **ArrayIndexOutOfBoundsException**: Terjadi ketika indeks array diluar rentang yang valid.
   * **ArithmeticException**: Terjadi ketika terjadi kesalahan aritmatika, seperti pembagian dengan nol.
3. Error: Kondisi yang menunjukkan kegagalan serius pada lingkungan runtime. Error biasanya tidak dapat ditangani secara efektif oleh program dan mengindikasikan situasi yang tidak dapat diperbaiki atau kelalaian dalam konfigurasi sistem. Beberapa contoh error adalah:
   * **OutOfMemoryError**: Terjadi ketika tidak ada cukup memori yang tersedia untuk menjalankan program.
   * **StackOverflowError**: Terjadi ketika stack memori penuh karena rekursi tak terbatas.

Selain tipe-tipe exception di atas, terdapat juga kelas-kelas exception khusus yang merupakan turunan dari **Exception** atau **RuntimeException**, seperti **IllegalArgumentException**, **NullPointerException**, dan sebagainya. Kita juga dapat membuat kelas exception kustom yang merupakan turunan dari kelas **Exception** atau **RuntimeException** untuk menangani kondisi khusus dalam program kita.

Penting untuk menangani exception dengan benar untuk memastikan program kita dapat merespons dengan baik terhadap kondisi yang tidak terduga dan memberikan pesan yang informatif kepada pengguna.

## C. Mekanisme Exception Handling

Mekanisme exception handling dalam pemrograman Java digunakan untuk menangani dan merespons exception yang terjadi saat program sedang berjalan. Exception handling memungkinkan program untuk mengatasi kesalahan atau situasi yang tidak terduga dengan cara yang terkontrol, sehingga meminimalkan dampak buruk pada alur eksekusi program.

Berikut ini adalah mekanisme exception handling dalam Java:

1. Blok **try**: Blok try digunakan untuk menempatkan kode yang berpotensi menyebabkan exception terjadi. Blok ini berisi satu atau lebih pernyataan yang dapat melemparkan exception.
2. Blok **catch**: Blok catch digunakan untuk menangkap dan menangani exception yang dilemparkan oleh blok try. Blok catch berisi pernyataan yang akan dieksekusi jika exception dengan tipe yang sesuai terjadi. Blok catch dapat memiliki beberapa blok catch yang berbeda untuk menangani tipe-tipe exception yang berbeda secara terpisah.

Contoh:

try {

// Kode yang mungkin menyebabkan exception

} catch (ExceptionType exception) {

// Penanganan exception

}

Di blok catch, kita menentukan tipe exception yang ingin ditangkap. Jika exception dengan tipe tersebut terjadi, blok catch akan dieksekusi dan penanganan exception dilakukan di dalamnya.

1. Blok **finally**: Blok finally adalah opsional dan dapat digunakan setelah blok catch. Blok ini berisi pernyataan yang akan dieksekusi setelah blok try atau blok catch selesai, baik exception terjadi atau tidak. Blok finally berguna untuk melakukan pembersihan atau penutupan sumber daya yang digunakan dalam blok try.

Contoh:

try {

// Kode yang mungkin menyebabkan exception

} catch (ExceptionType exception) {

// Penanganan exception

} finally {

// Pembersihan atau penutupan sumber daya

}

1. Klausul **throws**: Klausul throws digunakan dalam deklarasi metode untuk menunjukkan bahwa metode tersebut dapat melemparkan exception tertentu. Dengan menggunakan klausul throws, kita memberikan informasi kepada pemanggil metode tentang exception yang mungkin terjadi sehingga pemanggil metode dapat menanganinya dengan benar.

Contoh:

public void metode() throws ExceptionType {

// Kode yang mungkin melemparkan exception

}

Dalam exception handling, ketika sebuah exception terjadi, program akan mencari blok catch yang sesuai dengan tipe exception yang dilemparkan. Jika tidak ada blok catch yang sesuai ditemukan, exception akan diteruskan ke pemanggil metode yang lebih tinggi dalam hierarki pemanggilan (call stack). Jika exception tidak ditangani secara keseluruhan, program akan berhenti dan memberikan informasi mengenai exception tersebut.

Dengan menggunakan mekanisme exception handling, kita dapat merespons secara terkontrol terhadap exception yang terjadi dalam program.

## D. Blok Try-Catch-Finally

Blok try-catch-finally adalah bagian penting dalam mekanisme exception handling di Java. Blok ini digunakan untuk menangani exception yang terjadi dalam program dan memberikan respons yang sesuai. Berikut adalah penjelasan tentang setiap bagian blok try-catch-finally:

Blok try-catch-finally adalah bagian penting dalam mekanisme exception handling di Java. Blok ini digunakan untuk menangani exception yang terjadi dalam program dan memberikan respons yang sesuai. Berikut adalah penjelasan tentang setiap bagian blok try-catch-finally:

1. Blok **try**: Blok try digunakan untuk menempatkan kode yang berpotensi menyebabkan exception terjadi. Kode yang ditempatkan di dalam blok try akan dieksekusi secara normal. Jika exception terjadi dalam blok try, eksekusi kode akan berhenti di titik tersebut dan program akan mencari blok catch yang sesuai untuk menangani exception tersebut.

Contoh:

try { // Kode yang mungkin menyebabkan exception }

1. Blok **catch**: Blok catch digunakan untuk menangkap dan menangani exception yang dilemparkan oleh blok try. Blok catch berisi pernyataan yang akan dieksekusi jika exception dengan tipe yang sesuai terjadi. Blok catch dapat memiliki beberapa blok catch yang berbeda untuk menangani tipe-tipe exception yang berbeda secara terpisah. Pada umumnya, blok catch diikuti oleh parameter yang menunjukkan tipe exception yang ditangkap.

Contoh:

catch (ExceptionType exception) { // Penanganan exception }

Di dalam blok catch, kita dapat menulis kode untuk menangani exception tersebut. Kode ini bisa berupa penanganan exception, seperti mencetak pesan kesalahan atau melakukan tindakan tertentu sesuai dengan situasi yang dihadapi.

1. Blok **finally**: Blok finally adalah opsional dan digunakan setelah blok try atau blok catch. Blok ini berisi pernyataan yang akan dieksekusi setelah blok try atau blok catch selesai, baik exception terjadi atau tidak. Blok finally berguna untuk melakukan pembersihan atau penutupan sumber daya yang digunakan dalam blok try.

Contoh:

finally { // Pembersihan atau penutupan sumber daya }

Di dalam blok finally, kita dapat menulis kode yang akan dijalankan terlepas dari apakah exception terjadi atau tidak. Kode ini sering digunakan untuk memastikan bahwa sumber daya yang digunakan dalam blok try tetap ditutup atau dibersihkan dengan benar, bahkan jika terjadi exception.

Pada umumnya, blok try-catch-finally digunakan bersama-sama untuk menangani exception dan memastikan bahwa program tetap berjalan dengan baik meskipun terjadi exception. Ketika sebuah exception terjadi dalam blok try, program akan mencoba mencocokkan exception tersebut dengan blok catch yang sesuai. Jika exception cocok, kode di dalam blok catch akan dieksekusi. Setelah itu, blok finally akan dieksekusi terlepas dari apakah exception terjadi atau tidak.

Contoh lengkap penggunaan blok try-catch-finally:

try { // Kode yang mungkin menyebabkan exception }

catch (ExceptionType exception) { // Penanganan exception }

finally { // Pembersihan atau penutupan sumber daya }

Dalam blok try-catch-finally, kita dapat menggunakan salah satu atau kombinasi dari beberapa blok catch untuk menangani tipe-tipe exception yang berbeda secara terpisah. Ini memungkinkan kita untuk merespons dengan tepat terhadap jenis exception yang terjadi.

Berikut adalah contoh penggunaan multiple catch blocks:

try {

// Kode yang mungkin menyebabkan exception

} catch (ExceptionType1 exception1) {

// Penanganan exception1

} catch (ExceptionType2 exception2) {

// Penanganan exception2

} catch (ExceptionType3 exception3) {

// Penanganan exception3

} finally {

// Pembersihan atau penutupan sumber daya

}

Dalam contoh di atas, setiap blok catch menangani tipe exception yang berbeda. Jika exception dengan tipe **ExceptionType1** terjadi, blok catch pertama akan dieksekusi. Jika exception dengan tipe **ExceptionType2** terjadi, blok catch kedua akan dieksekusi, dan begitu seterusnya. Jika tidak ada blok catch yang sesuai dengan exception yang terjadi, exception akan diteruskan ke blok catch yang berada di level pemanggilan yang lebih tinggi atau program akan berhenti jika tidak ada penanganan exception yang sesuai.

Selain itu, blok finally juga dapat digunakan tanpa blok catch. Ini berguna jika kita ingin melakukan pembersihan atau penutupan sumber daya tanpa memerlukan penanganan exception secara khusus. Contoh penggunaan blok finally tanpa catch adalah sebagai berikut:

try {

// Kode yang mungkin menyebabkan exception

} finally {

// Pembersihan atau penutupan sumber daya

}

Dalam contoh di atas, blok finally akan selalu dieksekusi, terlepas dari apakah exception terjadi atau tidak. Hal ini memastikan bahwa pembersihan sumber daya dilakukan dengan benar setelah blok try selesai dieksekusi.

Mekanisme try-catch-finally dalam exception handling sangat berguna untuk mengelola exception dengan baik dan memastikan bahwa program tetap berjalan dengan benar dalam situasi yang tidak terduga. Dengan menggunakan kombinasi blok try-catch-finally, kita dapat menangani exception, meresponsnya, dan melakukan pembersihan yang diperlukan dengan cara yang terkontrol dan dapat diandalkan.

## E. Pernyataan throw dan throws

Pernyataan **throw** dan klausul **throws** adalah bagian penting dalam mekanisme exception handling di Java. Keduanya digunakan untuk mengelola dan memberikan informasi tentang exception yang mungkin terjadi dalam program.

1. Pernyataan **throw**: Pernyataan **throw** digunakan untuk melemparkan atau melempar sebuah exception secara manual dalam sebuah blok kode. Ketika kita menemui situasi yang membutuhkan pemrosesan exception, kita dapat menggunakan pernyataan **throw** untuk melemparkan exception yang sesuai.

Contoh penggunaan pernyataan **throw**:

throw new ExceptionType("Pesan exception");

Dalam contoh di atas, kita menggunakan pernyataan **throw** untuk melemparkan instance dari **ExceptionType** dengan pesan yang ditentukan. Setelah pernyataan **throw** dieksekusi, kontrol akan keluar dari blok kode saat ini dan mencari blok catch yang sesuai untuk menangani exception yang dilemparkan.

1. Klausul **throws**: Klausul **throws** digunakan dalam deklarasi metode untuk menunjukkan bahwa metode tersebut dapat melemparkan atau melempar exception tertentu. Dengan menggunakan klausul **throws**, kita memberikan informasi kepada pemanggil metode bahwa exception tertentu mungkin terjadi dan pemanggil metode tersebut harus menanganinya.

Contoh penggunaan klausul **throws**:

public void methodName() throws ExceptionType1, ExceptionType2 {

// Kode yang mungkin melemparkan exception

}

Dalam contoh di atas, klausul **throws** digunakan dalam deklarasi metode **methodName** untuk menunjukkan bahwa metode tersebut dapat melemparkan **ExceptionType1** dan **ExceptionType2**. Pemanggil metode **methodName** harus menangani exception tersebut dengan menggunakan blok try-catch atau meneruskan klausul **throws** pada metode mereka sendiri.

Pernyataan **throw** digunakan ketika kita ingin secara manual melemparkan sebuah exception dalam blok kode kita. Hal ini berguna ketika kita ingin mengatasi situasi tertentu dan memicu terjadinya exception dengan informasi yang sesuai.

Klausul **throws** digunakan untuk memberikan informasi tentang exception yang mungkin terjadi dalam suatu metode. Dengan menggunakan klausul **throws**, kita memberi tahu pemanggil metode tentang kemungkinan exception yang dapat dilemparkan oleh metode tersebut. Pemanggil metode harus menangani exception tersebut dengan menggunakan blok try-catch atau meneruskan klausul **throws** pada metode mereka sendiri.

Kedua fitur ini, **throw** dan **throws**, memungkinkan kita untuk mengelola exception dengan lebih baik dalam program Java dan memastikan bahwa exception ditangani secara efektif sesuai kebutuhan aplikasi.

## F. Membuat Exception Kustom

Dalam bahasa Java, kita dapat membuat exception kustom atau exception khusus yang sesuai dengan kebutuhan kita. Exception kustom dapat digunakan untuk menangani situasi yang spesifik dalam program kita. Untuk membuat exception kustom, kita perlu mengikuti langkah-langkah berikut:

1. Membuat kelas exception baru: Buat kelas baru yang mewarisi kelas **Exception** atau salah satu subkelas **Exception** yang ada, seperti **RuntimeException** atau **IOException**. Kita dapat membuat kelas tersebut dengan menentukan nama kelas dan menggunakan kata kunci **extends** untuk mewarisi kelas **Exception**.

Contoh:

public class CustomException extends Exception {

// Constructor dan kode lainnya

}

1. Menentukan constructor: Dalam kelas exception kustom, kita perlu menentukan satu atau lebih constructor. Constructor ini digunakan untuk menginisialisasi exception dengan pesan khusus atau argumen tambahan yang diperlukan.

Contoh:

public class CustomException extends Exception {

public CustomException() {

super("Pesan exception kustom");

}

public CustomException(String message) {

super(message);

}

}

Dalam contoh di atas, kita mendefinisikan dua constructor. Constructor pertama mengatur pesan default untuk exception kustom, sedangkan constructor kedua memungkinkan kita untuk menentukan pesan exception secara khusus saat membuat instance exception.

1. Menggunakan exception kustom: Setelah exception kustom dibuat, kita dapat menggunakannya di dalam kode program kita dengan cara yang sama seperti menggunakan exception bawaan Java. Kita dapat melemparkan exception kustom dengan pernyataan **throw** atau menangkapnya dengan blok try-catch.

Contoh penggunaan exception kustom:

public void throwCustomException() throws CustomException {

throw new CustomException("Exception kustom terjadi");

}

public void handleCustomException() {

try {

throwCustomException();

} catch (CustomException e) {

System.out.println(e.getMessage());

}

}

Dalam contoh di atas, kita mendefinisikan dua metode. Metode **throwCustomException()** melemparkan instance exception kustom **CustomException** dengan pesan khusus. Metode **handleCustomException()** menangkap exception kustom tersebut dengan blok catch dan mencetak pesan exception yang ditangkap.

Dengan membuat exception kustom, kita dapat mengendalikan dan menangani situasi-situasi spesifik dalam program kita. Exception kustom memungkinkan kita untuk memberikan informasi yang lebih spesifik tentang kesalahan atau kondisi yang terjadi, sehingga memudahkan pemecahan masalah dan pemeliharaan program kita.

# BAB IV

# PENGGUNAAN

# INTERFACE dalam EXCEPTION HEADING

## A. Menggunakan Interface sebagai Tipe Parameter

Dalam Java, kita dapat menggunakan interface sebagai tipe parameter dalam deklarasi metode. Ini memungkinkan kita untuk mengirimkan objek yang mengimplementasikan interface tersebut ke dalam metode. Dengan menggunakan interface sebagai tipe parameter, kita dapat membuat metode yang lebih fleksibel dan dapat digunakan dengan berbagai implementasi yang berbeda. Berikut adalah penjelasan lebih lanjut tentang penggunaan interface sebagai tipe parameter:

1. Deklarasi metode dengan parameter interface: Untuk menggunakan interface sebagai tipe parameter, kita perlu menyertakan nama interface tersebut dalam deklarasi metode sebagai tipe parameter. Ini memungkinkan kita untuk mengirimkan objek yang mengimplementasikan interface tersebut ke dalam metode.

Contoh:

public void processData(DataProcessor processor) {

// Kode untuk memproses data menggunakan objek processor

}

Dalam contoh di atas, **DataProcessor** adalah nama interface yang digunakan sebagai tipe parameter dalam metode **processData()**. Metode ini akan menerima objek apa pun yang mengimplementasikan interface **DataProcessor** sebagai argumen.

1. Mengimplementasikan interface pada kelas: Untuk menggunakan interface sebagai tipe parameter, kita perlu membuat kelas yang mengimplementasikan interface tersebut. Kelas ini harus menyediakan implementasi untuk semua metode yang dideklarasikan dalam interface.

Contoh:

public class DataProcessorImpl implements DataProcessor {

// Implementasi metode yang dideklarasikan dalam interface

}

Dalam contoh di atas, **DataProcessorImpl** adalah kelas yang mengimplementasikan interface **DataProcessor**. Kelas ini harus memberikan implementasi untuk semua metode yang dideklarasikan dalam **DataProcessor**.

1. Menggunakan objek yang mengimplementasikan interface: Setelah kita memiliki kelas yang mengimplementasikan interface, kita dapat membuat objek dari kelas tersebut dan mengirimkannya ke metode yang menggunakan interface sebagai tipe parameter.

Contoh:

DataProcessorImpl processor = new DataProcessorImpl();

processData(processor);

Dalam contoh di atas, kita membuat objek **processor** dari kelas **DataProcessorImpl**, yang mengimplementasikan interface **DataProcessor**. Kemudian, objek **processor** tersebut dikirim ke metode **processData()** yang menerima **DataProcessor** sebagai tipe parameter.

Dengan menggunakan interface sebagai tipe parameter, kita dapat membuat metode yang lebih fleksibel dan independen terhadap implementasi kelas konkret. Ini memungkinkan kita untuk menggunakan berbagai implementasi interface yang berbeda dalam satu metode, sehingga meningkatkan fleksibilitas dan reusabilitas kode kita.

## B. Menangani Exception dengan Interface

Dalam Java, kita dapat menggunakan interface untuk menangani exception yang terjadi dalam metode. Dengan menggunakan interface sebagai bagian dari mekanisme exception handling, kita dapat mengabstraksi penanganan exception dan memisahkannya dari implementasi kelas konkret. Berikut adalah penjelasan tentang cara menangani exception dengan menggunakan interface:

1. Membuat interface exception handler: Pertama, kita perlu membuat sebuah interface yang akan digunakan untuk menangani exception. Interface ini harus mendefinisikan metode atau operasi yang akan dilakukan saat exception terjadi.

Contoh:

public interface ExceptionHandler {

void handleException(Exception e);

}

Dalam contoh di atas, kita membuat interface **ExceptionHandler** yang memiliki satu metode **handleException()**. Metode ini akan digunakan untuk menangani exception yang terjadi.

1. Mengimplementasikan interface exception handler: Selanjutnya, kita perlu membuat kelas yang mengimplementasikan interface exception handler. Kelas ini akan memberikan implementasi untuk metode **handleException()** yang didefinisikan dalam interface.

Contoh:

public class MyExceptionHandler implements ExceptionHandler {

@Override

public void handleException(Exception e) {

// Log atau penanganan lainnya untuk exception yang terjadi

}

}

Dalam contoh di atas, kita membuat kelas **MyExceptionHandler** yang mengimplementasikan **ExceptionHandler**. Kelas ini memberikan implementasi untuk metode **handleException()** sesuai dengan kebutuhan kita, seperti mencetak pesan exception atau melakukan tindakan penanganan lainnya.

1. Menangani exception dengan menggunakan interface: Setelah kita memiliki interface dan implementasi kelas, kita dapat menggunakan objek dari kelas yang mengimplementasikan interface untuk menangani exception dalam metode.

Contoh:

public void process() {

try {

// Kode yang mungkin menyebabkan exception

} catch (Exception e) {

ExceptionHandler exceptionHandler = new MyExceptionHandler();

exceptionHandler.handleException(e);

}

}

Dalam contoh di atas, dalam blok catch, kita membuat objek **MyExceptionHandler** dan menggunakan objek tersebut untuk memanggil metode **handleException()** dalam interface **ExceptionHandler**. Dengan demikian, kita memisahkan penanganan exception dari implementasi kelas konkret dan menggunakan abstraksi melalui interface.

Dengan menggunakan interface untuk menangani exception, kita dapat mencapai fleksibilitas dan abstraksi yang lebih tinggi dalam penanganan exception. Ini memungkinkan kita untuk dengan mudah mengganti atau mengubah metode penanganan exception dengan mengganti implementasi kelas yang mengimplementasikan interface. Selain itu, ini juga memungkinkan kita untuk mengatur strategi penanganan exception yang berbeda untuk kondisi yang berbeda dalam program kita.

## C. Contoh Implementasi Interface dalam Exception Handling

Salah satu contoh implementasi interface dalam exception handling adalah dengan menggunakan mekanisme callback. Dalam hal ini, kita dapat menggunakan interface sebagai tipe parameter untuk metode yang mengoperasikan logika bisnis dan menangani exception yang mungkin terjadi. Berikut adalah contoh implementasi interface dalam exception handling menggunakan mekanisme callback:

1. Membuat interface exception handler: Pertama, kita perlu membuat sebuah interface yang mendefinisikan metode untuk menangani exception.

public interface ExceptionHandler {

void handleException(Exception e);

}

Interface **ExceptionHandler** memiliki satu metode **handleException()** yang menerima objek exception sebagai parameter.

1. Implementasi kelas yang mengimplementasikan interface: Selanjutnya, kita perlu membuat kelas yang mengimplementasikan interface **ExceptionHandler** dan memberikan implementasi untuk metode **handleException()**.

public class MyExceptionHandler implements ExceptionHandler {

@Override

public void handleException(Exception e) {

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

// Logika penanganan exception lainnya

}

}

Dalam contoh ini, kelas **MyExceptionHandler** mengimplementasikan **ExceptionHandler** dan mencetak pesan exception yang terjadi.

1. Metode dengan parameter tipe interface exception handler: Selanjutnya, kita dapat membuat metode yang menerima parameter dengan tipe interface exception handler untuk menangani exception.

public void processData(ExceptionHandler exceptionHandler) {

try {

// Kode logika bisnis

throw new Exception("Something went wrong!");

} catch (Exception e) {

exceptionHandler.handleException(e);

}

}

Metode **processData()** menerima objek **ExceptionHandler** sebagai parameter dan menggunakannya untuk menangani exception yang terjadi dalam blok catch.

1. Penggunaan metode dengan implementasi interface exception handler: Terakhir, kita dapat menggunakan metode **processData()** dengan memberikan implementasi kelas exception handler sebagai argumen.

public static void main(String[] args) {

MyExceptionHandler exceptionHandler = new MyExceptionHandler();

processData(exceptionHandler);

}

Dalam contoh ini, kita membuat objek **MyExceptionHandler** dan mengirimkannya ke metode **processData()** untuk menangani exception.

Dengan menggunakan implementasi interface dalam exception handling, kita dapat memisahkan logika bisnis dari penanganan exception. Hal ini memungkinkan fleksibilitas dalam mengganti strategi penanganan exception dengan mengubah implementasi kelas yang mengimplementasikan interface exception handler. Selain itu, dengan menggunakan callback melalui interface, kita dapat memisahkan tanggung jawab antara logika bisnis dan penanganan exception.

# BAB V

# KESIMPULAN

## A. Ringkasan

Dalam rangka mengimplementasikan interface dan exception handling dalam bahasa Java, berikut adalah ringkasan dari topik yang telah dibahas:

Interface:

* Interface adalah kontrak yang menggambarkan perilaku atau metode yang harus diimplementasikan oleh kelas-kelas yang menggunakannya.
* Interface berisi deklarasi metode tanpa implementasi.
* Interface digunakan untuk mencapai abstraksi, fleksibilitas, dan reusabilitas kode.
* Interface dapat digunakan sebagai tipe parameter dalam deklarasi metode, memungkinkan penggunaan berbagai implementasi yang berbeda dalam satu metode.

Exception Handling:

* Exception adalah kondisi yang abnormal atau kesalahan yang terjadi saat program sedang berjalan.
* Exception handling adalah mekanisme untuk menangani dan mengelola exception agar program tetap berjalan dengan benar.
* Exception handling menggunakan blok try-catch-finally untuk menangkap, menanganinya, dan memberikan penanganan alternatif ketika exception terjadi.
* Blok try digunakan untuk menempatkan kode yang berpotensi menghasilkan exception.
* Blok catch digunakan untuk menangkap dan menangani exception yang terjadi.
* Blok finally digunakan untuk menempatkan kode yang akan dieksekusi, baik terjadi exception atau tidak.
* Pernyataan throw digunakan untuk melemparkan atau melewatkan exception yang didefinisikan atau sudah ada.
* Pernyataan throws digunakan dalam deklarasi metode untuk menunjukkan bahwa metode tersebut mungkin melemparkan exception tertentu.

Menggunakan Interface dalam Exception Handling:

* Interface dapat digunakan untuk mengabstraksi penanganan exception dari implementasi kelas konkret.
* Dengan menggunakan interface, penanganan exception dapat dipisahkan dan digantikan dengan implementasi kelas yang mengimplementasikan interface tersebut.
* Menggunakan interface dalam exception handling dapat meningkatkan fleksibilitas, abstraksi, dan reusabilitas kode.
* Interface dapat digunakan sebagai tipe parameter dalam metode untuk menangani exception dan memisahkan logika bisnis dari penanganan exception.

Penggunaan interface dan exception handling dalam bahasa Java memungkinkan pengembang untuk mencapai pemisahan tanggung jawab yang baik antara logika bisnis dan penanganan exception, serta meningkatkan fleksibilitas dan reusabilitas kode dalam menghadapi situasi yang berbeda dalam program.

## B. Manfaat dan Kelemahan Penggunaan Interface dan Exception Handling

Penggunaan interface dan exception handling dalam pengembangan perangkat lunak memiliki manfaat dan kelemahan yang perlu dipertimbangkan. Berikut adalah beberapa manfaat dan kelemahan penggunaan interface dan exception handling:

Manfaat penggunaan Interface:

1. Abstraksi dan Polimorfisme: Interface memungkinkan kita untuk mencapai abstraksi, yaitu memisahkan implementasi dari perilaku yang diharapkan. Ini memungkinkan polimorfisme, di mana objek yang mengimplementasikan interface dapat digunakan secara generik dalam berbagai konteks.
2. Fleksibilitas dan Reusabilitas: Interface memungkinkan pengembangan yang lebih fleksibel dan reusabilitas kode. Dengan menggunakan interface, kita dapat mengganti implementasi kelas dengan yang baru tanpa mempengaruhi bagian lain dari kode yang menggunakannya.
3. Penggabungan Fungsionalitas: Interface memungkinkan penggabungan fungsionalitas dari beberapa kelas yang berbeda melalui implementasi interface yang sama. Ini memungkinkan kita untuk menggabungkan dan mengatur kode dengan lebih baik.

Kelemahan penggunaan Interface:

1. Kompleksitas Lebih Tinggi: Menggunakan interface dapat menambah kompleksitas kode karena memerlukan penambahan lapisan abstraksi dan pengaturan yang tepat untuk menghubungkan interface dengan implementasi kelasnya.
2. Pembatasan Fitur: Interface tidak dapat menyertakan implementasi metode, sehingga kita harus mengimplementasikan metode yang dideklarasikan dalam interface di kelas-kelas yang menggunakannya. Hal ini dapat membatasi beberapa fitur atau kemampuan yang mungkin diinginkan dalam implementasi kelas konkret.

Manfaat penggunaan Exception Handling:

1. Penanganan Exception yang Terstruktur: Exception handling memungkinkan penanganan yang terstruktur dan terorganisir terhadap exception yang mungkin terjadi dalam program. Ini membantu memisahkan logika normal dari penanganan exception.
2. Peningkatan Keandalan: Dengan menggunakan exception handling, kita dapat mengidentifikasi dan menangani exception secara terpusat, sehingga meningkatkan keandalan program. Hal ini memungkinkan kita untuk menangani dan memperbaiki situasi yang tidak terduga dengan baik.
3. Peningkatan Keterbacaan: Exception handling dapat meningkatkan keterbacaan kode karena memisahkan logika normal dengan penanganan exception. Kode menjadi lebih terstruktur dan lebih mudah dipahami.

Kelemahan penggunaan Exception Handling:

1. Overhead Kinerja:

Menggunakan exception handling dapat menambah overhead kinerja pada program karena melibatkan mekanisme penanganan exception yang memerlukan pengecekan dan transfer kontrol.

1. Pemecahan Alur Logika:

Terlalu banyak penanganan exception dalam kode dapat memecah alur logika dan membuatnya sulit untuk mengikuti jalannya program secara keseluruhan. Hal ini dapat mengaburkan logika bisnis yang sebenarnya.

1. Kesulitan Pemecahan Masalah:

Terkadang, penanganan exception yang buruk atau tidak memadai dapat menyulitkan pemecahan masalah dan debugging.

1. Missed Exception:

Terkadang, exception dapat terlewat dan tidak ditangkap atau ditangani dengan benar. Hal ini dapat menyebabkan bug yang sulit dilacak dan diperbaiki.

1. Ketergantungan pada Exception:

Terlalu banyak mengandalkan exception handling sebagai mekanisme penanganan error dapat membuat kode menjadi kurang prediktif dan sulit untuk diuji. Ketergantungan pada exception dapat menyebabkan kode yang sulit dijaga dan dipelihara.

1. Penggunaan Exception untuk Alur Kontrol:

Kadang-kadang, exception handling digunakan untuk mengontrol alur program secara keseluruhan, yang dapat menyebabkan kebingungan dan mempersulit pemahaman kode.

Penting untuk menggunakan exception handling secara bijak dan mempertimbangkan trade-off antara manfaat dan kerugian yang terkait. Pemilihan dan desain yang baik dalam penggunaan interface dan exception handling dapat membantu meningkatkan struktur dan keterbacaan kode, serta memfasilitasi pengembangan perangkat lunak yang lebih fleksibel dan mudah dikelola.

# DAFTAR PUSTAKA

* <https://stackoverflow.com/questions/15607060/java-interface-throws-an-exception-but-interface-implementation-does-not-throw-a>
* <https://levelup.gitconnected.com/a-couple-of-notes-about-exceptions-and-interfaces-in-java-164775a22abc>
* <https://www.tutorialspoint.com/can-the-abstract-methods-of-an-interface-throw-an-exception-in-java>
* <https://blog.eduonix.com/java-programming-2/learn-interfaces-exception-handling-java-programming/>
* <http://rifatshahriyar.github.io/files/CSE107/Java-Package_Interface_Exception.pdf>
* <https://coderanch.com/t/399874/java/Methods-throwing-Exception-Interface>
* <https://softwareengineering.stackexchange.com/questions/355184/what-should-i-do-with-implementation-specific-exceptions>
* <https://blogs.oracle.com/javamagazine/post/java-functional-interfaces-checked-exceptions>
* <https://dev.to/phouchens/handling-exceptions-in-java-streams-using-a-functional-interface-1p6e>