LAPORAN PRAKTIKUM XII

FUNDAMENTAL OF PROGRAMMING WORKSHOP



DOSEN PENGAJAR

Indrawati, SST.MT

**DISUSUN OLEH**

**Kelompok I:**

DIKI CANDRA

**Kelas : TRKJ-1B**

POLITEKNIK NEGERI LHOKSEUMAWE

JURUSAN TEKNOLOGI KOMUNIKASI DAN KOMPUTER

PRODI D-IV TEKNOLOGI REKAYASA KOMPUTER DAN JARINGAN

2022/2023

1. **TUJUAN PRAKTIKUM**

1. Memahami tentang Interface pada Bahasa Java

2. Memahami tentang Exception pada Bahasa Java

1. **DASAR TEORI**

Interface dalam pemrograman berorientasi objek adalah sebuah kontrak atau spesifikasi yang menguraikan metode yang harus diimplementasikan oleh kelas-kelas yang menggunakannya. Dalam Java, interface digunakan untuk mendefinisikan metode-metode tanpa implementasi yang berfungsi sebagai kontrak bagi kelas-kelas yang ingin mengadopsi (implement) interface tersebut.

Penggunaan interface memberikan beberapa keuntungan, di antaranya:

1. Mendorong polimorfisme: Dengan menggunakan interface, kita dapat memperlakukan objek dari berbagai kelas yang mengimplementasikan interface yang sama sebagai objek dari tipe interface tersebut. Hal ini memungkinkan penggunaan polimorfisme, di mana objek dapat dianggap sebagai objek dari tipe interface yang lebih umum, sehingga memudahkan dalam mengelola objek dengan fleksibilitas lebih tinggi.
2. Membantu dalam membangun hubungan lemah: Interface memungkinkan kita untuk membangun hubungan antara kelas-kelas yang tidak memiliki hubungan langsung melalui pewarisan (inheritance). Dengan menggunakan interface, kelas-kelas yang tidak berhubungan secara hierarkis dapat berinteraksi melalui kontrak yang ditentukan oleh interface tersebut. Hal ini memungkinkan pengembangan aplikasi dengan desain yang lebih fleksibel dan terlepas dari hubungan langsung antara kelas-kelas.
3. Mendukung desain modular: Interface memisahkan kontrak atau spesifikasi metode dari implementasinya. Dengan demikian, interface memungkinkan pemisahan antara bagaimana metode harus berperilaku (kontrak) dan bagaimana metode tersebut diimplementasikan. Hal ini mendukung desain modular yang lebih baik, karena memungkinkan pengembang untuk fokus pada kontrak dan spesifikasi metode tanpa harus terikat pada implementasi yang spesifik.

Exception (pengecualian) adalah situasi atau kondisi yang mengganggu alur normal program. Dalam Java, exception digunakan untuk menangani kesalahan atau kondisi yang tidak terduga selama eksekusi program. Ketika sebuah kesalahan terjadi, exception dapat dibuat (thrown) dan kemudian ditangkap (caught) serta dikelola oleh kode pemrograman.

Java memiliki beberapa jenis exception bawaan, yaitu:

1. Checked Exception: Exception yang harus ditangkap (caught) atau dideklarasikan (declared) dalam bagian tanda tangan metode. Ini berarti bahwa saat menggunakan metode yang dapat melemparkan checked exception, kita harus menggunakan blok try-catch untuk menangkapnya atau deklarasikan exception tersebut dalam tanda tangan metode. Contoh checked exception adalah IOException yang terjadi saat terjadi kesalahan I/O.
2. Unchecked Exception: Exception yang tidak diharuskan untuk ditangkap atau dideklarasikan secara eksplisit. Unchecked exception terjadi pada saat runtime dan tidak memerlukan penanganan khusus oleh kode pemrograman. Contoh dari unchecked exception adalah NullPointerException yang terjadi saat mengakses objek yang bernilai null, atau ArrayIndexOutOfBoundsException saat mengakses indeks array yang tidak valid.
3. Error: Exception yang terjadi ketika terjadi situasi yang tidak dapat diperbaiki oleh program. Error biasanya terjadi pada tingkat yang lebih rendah dalam struktur JVM (Java Virtual Machine) dan seringkali menunjukkan kondisi serius yang tidak dapat dipulihkan, seperti OutOfMemoryError yang terjadi ketika program kehabisan memori, atau StackOverflowError yang terjadi ketika terlalu banyak rekursi yang mengakibatkan stack penuh.
4. Dalam pemrograman, penanganan exception dapat dilakukan dengan menggunakan blok try-catch, di mana kita menempatkan kode yang mungkin memunculkan exception di dalam blok try, dan kemudian menangkap exception tersebut di dalam blok catch. Dengan menangkap exception, kita dapat melakukan tindakan yang sesuai, seperti memberikan pesan kesalahan kepada pengguna, melakukan logging, atau mengambil langkah pemulihan yang relevan.
5. Exception handling memungkinkan program untuk melanjutkan eksekusi meskipun terjadi kesalahan, sehingga memungkinkan penanganan yang lebih baik dan pemulihan dari situasi yang tidak terduga. Selain itu, exception handling juga membantu dalam menjaga keandalan dan stabilitas program, serta memberikan informasi yang berguna dalam proses debugging.

Jadi, exception dalam Java digunakan untuk menangani kesalahan atau kondisi yang tidak terduga selama eksekusi program, dan Java menyediakan mekanisme exception handling yang memungkinkan kita untuk menangkap, mengelola, dan merespons exception tersebut dengan cara yang sesuai.

1. **ALAT DAN BAHAN**

Untuk melakukan praktikum ini dibutuhkan alat dan bahan sebagai berikut:

1. Personal Computer / Laptop
2. Apache NetBeans IDE
3. **LANGKAH KERJA**

Berikut ini adalah 3 contoh program Java yang menggambarkan konsep Interface :

1. Contoh Interface Shape (Bentuk)

public interface Shape {

double getArea();

double getPerimeter();

}

Penjelasan:

Interface **Shape** merupakan sebuah contoh interface dalam Java yang memiliki dua metode tanpa implementasi yaitu **getArea()** dan **getPerimeter()**. Interface ini digunakan untuk mendefinisikan kontrak atau spesifikasi untuk objek-objek geometri yang memiliki luas (area) dan keliling (perimeter).

Dengan menggunakan interface **Shape**, kelas-kelas yang ingin mewakili objek-objek geometri dapat mengimplementasikan interface ini dan memberikan implementasi konkret untuk metode-metode yang didefinisikan dalam interface. Setiap kelas yang mengimplementasikan interface **Shape** harus mengimplementasikan kedua metode **getArea()** dan **getPerimeter()**.

Contoh penggunaan interface **Shape** dapat melibatkan kelas-kelas seperti **Circle**, **Rectangle**, atau **Triangle**. Setiap kelas tersebut akan mengimplementasikan interface **Shape** dan memberikan implementasi yang sesuai untuk metode **getArea()** dan **getPerimeter()**. Dengan demikian, kita dapat memperlakukan objek-objek dari kelas-kelas tersebut sebagai objek **Shape** secara umum, sehingga memungkinkan polimorfisme dan penggunaan yang fleksibel dalam konteks yang membutuhkan objek-objek geometri.

Dengan menggunakan interface seperti **Shape**, kita dapat mengabstraksi konsep geometri secara umum dan memastikan bahwa setiap objek yang mengklaim sebagai objek geometri harus menyediakan implementasi yang sesuai untuk metode yang didefinisikan dalam interface tersebut.

1. Contoh Program Interface List (Daftar) :

public interface List<T> {

void add(T element);

T get(int index);

int size();

}

Penjelasan:

Interface **List<T>** adalah contoh interface dalam Java yang menggunakan tipe parameter (generics) untuk mendefinisikan kontrak umum untuk koleksi berurutan dari elemen-elemen bertipe **T**. Interface ini memiliki tiga metode utama: **add()**, **get()**, dan **size()**.

Metode **add(T element)** digunakan untuk menambahkan elemen bertipe **T** ke dalam daftar. Implementasi konkret dari interface **List** akan menentukan bagaimana elemen ditambahkan, misalnya, apakah elemen ditambahkan di akhir daftar atau di posisi tertentu.

Metode **get(int index)** digunakan untuk mengambil elemen dari daftar berdasarkan indeks yang diberikan. Implementasi konkret akan menentukan cara mendapatkan elemen dengan memperhatikan indeks yang valid dalam rentang ukuran daftar.

Metode **size()** digunakan untuk mendapatkan jumlah elemen dalam daftar. Implementasi konkret akan menghitung dan mengembalikan jumlah elemen yang ada dalam daftar.

Dengan menggunakan interface **List<T>**, kita dapat membuat implementasi konkret seperti **ArrayList<T>** atau **LinkedList<T>**, di mana kita mengimplementasikan metode-metode tersebut sesuai dengan struktur data yang digunakan.

1. Contoh Program Interface Runnable:

public interface Runnable {

void run();

}

Penjelasan:

Interface **Runnable** adalah contoh interface dalam Java yang digunakan untuk mendefinisikan kontrak bagi objek yang dapat dijalankan (runnable). Interface ini memiliki satu metode yaitu **run()**, yang tidak mengembalikan nilai (void).

Metode **run()** digunakan untuk mendefinisikan tindakan yang akan dilakukan saat objek dijalankan. Implementasi konkret dari interface **Runnable** akan memberikan implementasi khusus untuk metode ini, yang akan dijalankan ketika objek tersebut dieksekusi.

Interface **Runnable** sangat berguna dalam konteks multithreading, di mana kita ingin menjalankan tindakan tertentu secara konkuren (concurrently) dalam thread terpisah. Dengan menggunakan interface **Runnable**, kita dapat membuat objek yang mewakili tugas yang ingin dijalankan dalam thread terpisah, dan menjalankan tugas tersebut menggunakan mekanisme thread yang disediakan oleh Java.

Berikut ini adalah 3 contoh program Java yang menggambarkan konsep Exception :

* 1. Contoh Program ArrayIndexOutOfBoundsException

public class ArrayExceptionExample {

public static void main(String[] args) {

int[] numbers = {1, 2, 3};

try {

// Mengakses indeks ke-3 pada array yang hanya memiliki 3 elemen

int value = numbers[3];

System.out.println(value);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Terjadi kesalahan: ArrayIndexOutOfBoundsException");

}

}

}

Penjelasan:

Program di atas adalah contoh penggunaan exception handling dalam Java. Pada program tersebut, kita memiliki array **numbers** dengan tiga elemen.

Dalam blok **try**, kita mencoba mengakses elemen ke-3 pada array **numbers** menggunakan indeks 3. Namun, karena array **numbers** hanya memiliki tiga elemen dengan indeks 0, 1, dan 2, mengakses indeks 3 akan menghasilkan **ArrayIndexOutOfBoundsException**.

Untuk menangani exception tersebut, kita menggunakan blok **catch** yang memiliki parameter **ArrayIndexOutOfBoundsException**. Di dalam blok **catch**, kita menangkap exception tersebut dan mencetak pesan kesalahan yang sesuai.

Jika program dijalankan, maka akan terjadi **ArrayIndexOutOfBoundsException** ketika mencoba mengakses elemen dengan indeks 3, dan pesan "Terjadi kesalahan: ArrayIndexOutOfBoundsException" akan dicetak sebagai hasilnya.

Dengan menggunakan exception handling seperti ini, program dapat menangani kesalahan atau kondisi yang tidak terduga dengan cara yang terkontrol dan memberikan respons yang sesuai. Dalam contoh ini, kita mengatasi kesalahan yang terjadi ketika mengakses indeks di luar batas array, dan memberikan pesan kesalahan yang jelas kepada pengguna.

* 1. Contoh Program NullPointerException:

public class NullPointerExample {

public static void main(String[] args) {

String text = null;

try {

// Memanggil metode pada objek null

int length = text.length();

System.out.println(length);

} catch (NullPointerException e) {

System.out.println("Terjadi kesalahan: NullPointerException");

}

}

}

Penjelasan:

Program di atas adalah contoh penggunaan exception handling untuk mengatasi **NullPointerException** dalam Java. Pada program tersebut, kita memiliki variabel **text** yang diinisialisasi dengan **null**.

Dalam blok **try**, kita mencoba memanggil metode **length()** pada objek **text**, yang sebenarnya adalah objek **null**. Karena **text** memiliki nilai **null**, memanggil metode pada objek **null** akan menghasilkan **NullPointerException**.

Untuk menangani exception tersebut, kita menggunakan blok **catch** dengan parameter **NullPointerException**. Di dalam blok **catch**, kita menangkap exception tersebut dan mencetak pesan kesalahan yang sesuai.

Jika program dijalankan, maka akan terjadi **NullPointerException** ketika mencoba memanggil metode **length()** pada objek **null**, dan pesan "Terjadi kesalahan: NullPointerException" akan dicetak sebagai hasilnya.

Dengan menggunakan exception handling seperti ini, program dapat menangani kesalahan yang terjadi saat menggunakan objek yang bernilai **null** dan memberikan respons yang sesuai. Dalam contoh ini, kita mengatasi kesalahan yang terjadi ketika memanggil metode pada objek **null** dan memberikan pesan kesalahan yang jelas kepada pengguna.

* 1. Contoh Program ArithmeticException:

public class ArithmeticExceptionExample {

public static void main(String[] args) {

int dividend = 10;

int divisor = 0;

try {

// Pembagian dengan nol

int result = dividend / divisor;

System.out.println(result);

} catch (ArithmeticException e) {

System.out.println("Terjadi kesalahan: ArithmeticException");

}

}

}

Penjelasan:

Program di atas adalah contoh penggunaan exception handling untuk mengatasi **ArithmeticException** dalam Java. Pada program tersebut, kita memiliki dua variabel **dividend** dan **divisor**.

Dalam blok **try**, kita mencoba melakukan operasi pembagian **dividend** dibagi dengan **divisor**, di mana **divisor** memiliki nilai 0. Pembagian dengan nol akan menghasilkan **ArithmeticException**.

Untuk menangani exception tersebut, kita menggunakan blok **catch** dengan parameter **ArithmeticException**. Di dalam blok **catch**, kita menangkap exception tersebut dan mencetak pesan kesalahan yang sesuai.

Jika program dijalankan, maka akan terjadi **ArithmeticException** ketika mencoba melakukan pembagian dengan nol, dan pesan "Terjadi kesalahan: ArithmeticException" akan dicetak sebagai hasilnya.

Dengan menggunakan exception handling seperti ini, program dapat menangani kesalahan yang terjadi saat melakukan operasi matematika yang tidak valid, seperti pembagian dengan nol, dan memberikan respons yang sesuai. Dalam contoh ini, kita mengatasi kesalahan yang terjadi ketika melakukan pembagian dengan nol dan memberikan pesan kesalahan yang jelas kepada pengguna.

1. **KESIMPULAN**

Interface dan exception adalah konsep yang sangat penting dalam bahasa pemrograman Java. Interface digunakan untuk menetapkan kontrak atau spesifikasi untuk kelas-kelas yang ingin mengimplementasikannya. Dengan menggunakan interface, kita dapat memisahkan antara apa yang harus dilakukan oleh sebuah kelas (metode yang harus diimplementasikan) dengan bagaimana itu dilakukan (implementasi konkret). Ini mendorong polimorfisme, memungkinkan hubungan lemah antara kelas-kelas yang berbeda, dan mendukung desain modular yang lebih baik.

Sementara itu, exception digunakan untuk menangani situasi yang tidak terduga atau kesalahan dalam program. Dalam bahasa Java, exception digunakan untuk mengidentifikasi dan menangani kondisi yang tidak normal dalam program. Dengan menggunakan mekanisme exception handling, kita dapat mengatasi dan memberikan respons yang tepat terhadap kesalahan yang terjadi saat program dieksekusi. Exception membantu dalam menjaga alur program yang stabil dan memungkinkan pengembang untuk mengantisipasi dan menangani situasi yang tidak terduga dengan cara yang terkontrol.

Dengan memahami dan menguasai kedua konsep ini, pengembang dapat menghasilkan kode yang lebih terstruktur, modular, dan lebih tangguh. Menggunakan interface memungkinkan pengembangan aplikasi yang fleksibel dan memungkinkan penggunaan polimorfisme. Sedangkan penggunaan exception handling membantu dalam mengidentifikasi dan menangani kesalahan dengan baik, menjaga stabilitas program, dan memberikan respons yang tepat kepada pengguna.

Dalam kombinasi, penggunaan interface dan exception memberikan fondasi yang kuat untuk pengembangan perangkat lunak yang efektif dan dapat diandalkan. Keduanya adalah alat penting dalam kotak perangkat pengembang Java, yang membantu dalam mengorganisasikan kode, mengelola kesalahan, dan mencapai desain yang baik.