MAKALAH PENGENALAN BAHASA PEMROGRAMAN

JAVA



Oleh :

DIKI CANDRA

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

**Daftar Isi**

[BAB I 3](#_Toc1397661688)

[PENDAHULUAN 3](#_Toc1114412205)

[1.1 Latar Belakang 3](#_Toc329979951)

[1.2 Rumusan Masalah 3](#_Toc1076385426)

[1.3 Tujuan 4](#_Toc1819366816)

[1.3 Manfaat 4](#_Toc1060984969)

[BAB II 4](#_Toc697675981)

[PEMBAHASAN 4](#_Toc1831382666)

[2.1 Sintaks Dasar Bahasa Java 4](#_Toc426594798)

[2.2 Fitur JAVA 4](#_Toc2029085456)

[2.3 Percabangan Java 4](#_Toc1480884601)

[2.4 Akses Modifier Java 4](#_Toc1946760945)

[2.5 Kelas Abstrak dan Antarmuka 4](#_Toc2085544809)

[BAB III 4](#_Toc2039396160)

[PENUTUPAN 4](#_Toc1573470416)

[3. 1 Kelebihan dan Kekurangan Mempelajari Bahasa Java 4](#_Toc419546077)

[3.2 Kesimpulan 4](#_Toc2127384563)

[DAFTAR PUSTAKA 4](#_Toc2090759659)

# BAB I

# PENDAHULUAN

**1.1 Latar Belakang**

Bahasa Java adalah bahasa pemrograman modern yang dikembangkan oleh Sun Microsystems pada tahun 1995. Java adalah bahasa pemrograman yang terkenal karena kemampuannya untuk bekerja pada berbagai platform dan memiliki kesederhanaan pemrograman yang mudah dipelajari dan diterapkan.

Bahasa pemrograman Java adalah salah satu bahasa pemrograman yang paling populer saat ini. Dalam artikel ini, kami akan memberikan pengenalan tentang bahasa pemrograman Java, termasuk sejarah, fitur, dan keuntungan penggunaannya.

Bahasa Java banyak digunakan untuk pengembangan aplikasi web, game, dan mobile. Java juga digunakan dalam pengembangan aplikasi enterprise dan berbagai aplikasi bisnis.

Pada makalah ini, kami akan membahas tentang dasar-dasar Bahasa Java dan penggunaannya dalam pengembangan perangkat lunak. Kami juga akan membahas fitur-fitur yang membedakan Bahasa Java dengan bahasa pemrograman lain.

Bahasa Java pertama kali dikembangkan oleh James Gosling, Patrick Naughton, dan Mike Sheridan pada tahun 1991 di Sun Microsystems. Awalnya, bahasa ini dikenal dengan nama Oak. Pada awalnya, bahasa ini dikembangkan untuk memenuhi kebutuhan perangkat elektronik, seperti mesin cuci dan set-top box. Namun, dengan perkembangan teknologi, bahasa ini diarahkan pada pengembangan aplikasi web.

Java pertama kali dirilis pada tahun 1995 sebagai bagian dari platform Java Development Kit (JDK) oleh Sun Microsystems. Pada tahun 2009, Sun Microsystems diakuisisi oleh Oracle Corporation dan pengembangan Bahasa Java menjadi tanggung jawab Oracle.

Pada saat ini, Bahasa Java menjadi salah satu bahasa pemrograman paling populer di dunia dengan banyak digunakan di industri perangkat lunak dan aplikasi web.

**1.2 Rumusan Masalah**

Rumusan masalah pada materi ini dapat dirumuskan sebagai berikut:

* Apa itu bahasa pemrograman Java dan sejarah pengembangannya?
* Apa saja fitur-fitur yang dimiliki oleh bahasa pemrograman Java?
* Apa keuntungan penggunaan bahasa pemrograman Java?
* Bagaimana cara mempelajari bahasa pemrograman Java bagi pemula?
* Bagaimana penggunaan bahasa pemrograman Java dalam pengembangan aplikasi?

**1.3 Tujuan**

Tujuan mempelajari materi ini adalah sebagai berikut:

1. Mengetahui tentang bahasa pemrograman Java secara lebih mendalam, termasuk sejarah, fitur, dan keuntungan penggunaannya.
2. Mengetahui cara memulai belajar bahasa pemrograman Java bagi pemula dan sumber daya apa saja yang tersedia untuk membantu dalam proses belajar.
3. Mengetahui bagaimana penggunaan bahasa pemrograman Java dalam pengembangan aplikasi, termasuk aplikasi web, aplikasi desktop, dan aplikasi seluler.
4. Memahami keuntungan dan kelebihan penggunaan bahasa pemrograman Java, sehingga dapat membuat keputusan yang tepat dalam memilih bahasa pemrograman untuk digunakan dalam pengembangan aplikasi.

Dengan mempelajari materi ini, seseorang dapat memiliki pengetahuan yang lebih mendalam tentang bahasa pemrograman Java dan dapat menggunakannya untuk mengembangkan berbagai jenis aplikasi. Selain itu, pengetahuan tentang bahasa pemrograman Java juga dapat membantu seseorang dalam mencari pekerjaan di bidang teknologi informasi, karena Java adalah salah satu bahasa pemrograman yang sangat dicari oleh perusahaan-perusahaan besar saat ini.

## **1.3 Manfaat**

Mempelajari materi ini memiliki berbagai manfaat, antara lain:

1. Meningkatkan kemampuan dalam pemrograman: Dengan mempelajari bahasa pemrograman Java, seseorang dapat meningkatkan kemampuan dalam pemrograman secara keseluruhan. Java adalah salah satu bahasa pemrograman yang paling populer dan digunakan secara luas di seluruh dunia, sehingga mempelajarinya dapat membuka peluang untuk mempelajari bahasa pemrograman lainnya.
2. Membuka peluang kerja: Java adalah salah satu bahasa pemrograman yang sangat dicari oleh perusahaan-perusahaan besar, terutama di bidang teknologi informasi. Oleh karena itu, mempelajari Java dapat membuka peluang kerja yang lebih luas di bidang ini.
3. Mengembangkan aplikasi: Java adalah bahasa pemrograman yang sangat kuat dan fleksibel, sehingga dapat digunakan untuk mengembangkan berbagai jenis aplikasi, mulai dari aplikasi desktop hingga aplikasi seluler dan aplikasi web. Dengan mempelajari Java, seseorang dapat mengembangkan aplikasi yang lebih kompleks dan lebih fungsional.
4. Meningkatkan daya saing: Dalam dunia bisnis yang sangat kompetitif saat ini, keahlian dalam bahasa pemrograman Java dapat memberikan keunggulan yang lebih besar dan meningkatkan daya saing dalam mencari pekerjaan dan dalam membangun bisnis.
5. Menjadi programmer yang lebih baik: Mempelajari bahasa pemrograman Java juga dapat membantu seseorang menjadi programmer yang lebih baik dan lebih efisien. Java memiliki banyak fitur yang dapat membantu meningkatkan efisiensi dan efektivitas dalam pengembangan aplikasi, serta mempermudah pemeliharaan aplikasi yang sudah ada.

Dengan mempelajari materi ini, seseorang dapat memperoleh berbagai manfaat dalam karir dan kehidupan pribadi, serta meningkatkan kemampuan dalam dunia pemrograman.

# BAB II

# PEMBAHASAN

## **2.1 Sintaks Dasar Bahasa Java**

Java adalah bahasa pemrograman yang populer di kalangan pengembang dan programmer. Bahasa ini memiliki sintaks yang sangat penting untuk dipahami untuk membuat program yang efektif dan efisien. Berikut ini adalah sintaks dasar Java lengkap dengan penjelasan dan contoh.

1. **Struktur program Java**

Setiap program Java harus memiliki struktur tertentu yang terdiri dari tiga bagian utama:

a. Package Declaration

Package Declaration merupakan baris pertama dalam file Java yang mendefinisikan package dari program. Package adalah nama dari kumpulan class atau file Java yang ada di dalamnya. Package Declaration tidak selalu diperlukan, tetapi sangat disarankan karena membuat program lebih mudah diorganisir.

Contoh:

javaCopy code

package com.example.myprogram;

b. Import Statements

Import Statements digunakan untuk mengimpor class dari package tertentu ke program Java. Import Statement harus ditempatkan di bawah Package Declaration, tetapi sebelum class declaration.

Contoh:

javaCopy code

import java.util.ArrayList;

c. Class Declaration

Class Declaration merupakan bagian utama dari program Java. Ini adalah tempat di mana kode program yang sebenarnya ditulis. Setiap program Java harus memiliki setidaknya satu class declaration.

Contoh:

kotlinCopy code

public class MyClass { // kode program }

1. **Data Types Java memiliki beberapa tipe data, seperti:**

a. Primitive Data Types Primitive Data Types adalah tipe data yang didefinisikan oleh bahasa Java dan digunakan untuk menyimpan nilai dasar seperti angka, karakter, dan boolean.

Contoh:

javaCopy code

int myNumber = 42; char myChar = 'a'; boolean myBoolean = true;

b. Reference Data Types

Reference Data Types adalah tipe data yang digunakan untuk menyimpan referensi ke objek di Java. Contoh dari Reference Data Types termasuk String, ArrayList, dan lain-lain.

Contoh:

vbnetCopy code

String myString = "Hello, world!"; ArrayList<Integer> myList = new ArrayList<Integer>();

1. **Variables**

Variabel adalah tempat untuk menyimpan nilai di program Java. Setiap variabel harus memiliki tipe data dan nama.

Contoh:

javaCopy code

int myNumber = 42; String myName = "John";

1. **Operators**

Operator adalah simbol atau karakter yang digunakan untuk melakukan operasi matematika, logika, dan lain-lain pada nilai dalam program Java. Beberapa operator dasar di Java termasuk:

a. Arithmetic Operators

Arithmetic Operators digunakan untuk melakukan operasi matematika pada nilai. Contoh dari Arithmetic Operators termasuk +, -, \*, /, dan %.

Contoh:

javaCopy code

int x = 10; int y = 5; int z = x + y; // Hasilnya adalah 15

b. Assignment Operators

Assignment Operators digunakan untuk menetapkan nilai ke variabel. Contoh dari Assignment Operators termasuk =, +=, -=, \*=, /=, dan %=.

Contoh:

javaCopy code

int x = 10; x += 5; // Nilai x sekarang adalah 15

c. Comparison Operators

Comparison Operators digunakan untuk membandingkan nilai. Contoh dari Comparison Operators termasuk ==, !=, <, >, <=, dan >=.

Contoh:

sqlCopy code

int x = 10; int y = 5; boolean result = x > y; // Hasilnya adalah true

d. Logical Operators

Logical Operators digunakan untuk melakukan operasi logika pada nilai Contoh dari Logical Operators termasuk &&, ||, dan !.

Contoh:

int x = 10; int y = 5; boolean result = x > 5 && y < 10; // Hasilnya adalah true

1. **Control Structures**

Control Structures digunakan untuk mengontrol jalannya program. Ada tiga jenis Control Structures di Java:

a. Conditional Statements

Conditional Statements digunakan untuk mengontrol aliran program berdasarkan kondisi tertentu. Contoh dari Conditional Statements termasuk if, else if, dan switch.

Contoh:

goCopy code

int x = 10; if (x > 5) { System.out.println("x lebih besar dari 5"); } else if (x < 5) { System.out.println("x lebih kecil dari 5"); } else { System.out.println("x sama dengan 5"); }

b. Looping Statements

Looping Statements digunakan untuk mengulang kode program. Contoh dari Looping Statements termasuk for, while, dan do-while.

Contoh:

csharpCopy code

for (int i = 0; i < 10; i++) { System.out.println(i); } int i = 0; while (i < 10) { System.out.println(i); i++; } int j = 0; do { System.out.println(j); j++; } while (j < 10);

c. Jump Statements

Jump Statements digunakan untuk mengubah aliran program. Contoh dari Jump Statements termasuk break, continue, dan return.

Contoh:

cssCopy code

for (int i = 0; i < 10; i++) { if (i == 5) { break; } System.out.println(i); } for (int i = 0; i < 10; i++) { if (i == 5) { continue; } System.out.println(i); } public int addNumbers(int a, int b) { return a + b; }

1. **Methods**

Methods digunakan untuk mengelompokkan kode program dan menghindari duplikasi kode. Methods harus dideklarasikan dengan tipe data, nama, dan argumen.

Contoh:

csharpCopy code

public int addNumbers(int a, int b) { return a + b; } public void printMessage(String message) { System.out.println(message); }

1. **Arrays**

Arrays digunakan untuk menyimpan kumpulan nilai dalam satu variabel. Arrays dapat dideklarasikan dengan tipe data dan ukuran tertentu.

Contoh:

int[] numbers = {1, 2, 3, 4, 5}; String[] names = {"John", "Mary", "Bob"}; int[] numbers = new int[5]; numbers[0] = 1; numbers[1] = 2; numbers[2] = 3; numbers[3] = 4; numbers[4] = 5;

1. **Classes and Objects**

Classes dan Objects adalah konsep yang sangat penting di Java. Class adalah blueprint untuk membuat objek, sementara objek adalah instansi dari sebuah class.

Contoh:

csharpCopy code

public class Person { String name; int age; public Person(String name, int age) { this.name = name; this.age = age; } public void printDetails() { System.out.println("Name: " + name); System.out.println("Age: " + age); } } Person person1 = new Person("John", 35); person1.printDetails()

1. **Inheritance**

Inheritance digunakan untuk membuat class baru dari class yang sudah ada. Class yang sudah ada disebut superclass atau parent class, sedangkan class baru disebut subclass atau child class.

Contoh:

public class Animal { String name; int age;

public Animal(String name, int age) { this.name = name; this.age = age; }

public void makeSound() { System.out.println("The animal makes a sound"); } }

public class Dog extends Animal { public Dog(String name, int age) { super(name, age); }

public void makeSound() { System.out.println("The dog barks"); } }

Dog dog1 = new Dog("Buddy", 3); dog1.makeSound();

10. **Polymorphism**

Polymorphism digunakan untuk memungkinkan subclass untuk menggunakan metode yang didefinisikan di superclass. Hal ini memungkinkan subclass untuk mengganti perilaku metode yang didefinisikan di superclass.

Contoh:

public class Animal { public void makeSound() { System.out.println("The animal makes a sound"); } }

public class Dog extends Animal { public void makeSound() { System.out.println("The dog barks"); } }

public class Cat extends Animal { public void makeSound() { System.out.println("The cat meows"); } }

Animal animal1 = new Dog(); animal1.makeSound();

Animal animal2 = new Cat(); animal2.makeSound();

## **2.2 Fitur JAVA**

Java adalah bahasa pemrograman yang berorientasi objek dan dianggap sebagai bahasa pemrograman yang sangat populer di dunia saat ini. Dalam artikel ini, kita akan membahas secara rinci tentang fitur-fitur bahasa Java yang membedakannya dari bahasa pemrograman lainnya.

1. Portabilitas

Salah satu fitur paling menonjol dari bahasa Java adalah portabilitasnya. Program yang ditulis dalam bahasa Java dapat dijalankan pada berbagai sistem operasi tanpa perlu melakukan perubahan pada kode sumber. Hal ini dicapai dengan menggunakan JVM (Java Virtual Machine), yang dapat diinstal pada berbagai sistem operasi.

1. Pemrograman Berorientasi Objek

Java sepenuhnya berorientasi objek dan menyediakan berbagai fitur pemrograman berorientasi objek yang kompleks. Pemrograman berorientasi objek memungkinkan pengembang untuk membangun aplikasi yang dapat dipecah menjadi beberapa objek atau entitas yang dapat berinteraksi satu sama lain untuk mencapai tujuan tertentu.

1. Memori Manajemen

Manajemen memori di Java dilakukan oleh JVM yang mengelola memori dan menghapus objek yang tidak lagi digunakan. Hal ini mengurangi risiko memory leak dan memungkinkan program Java untuk mengoptimalkan penggunaan memori.

1. Exception Handling

Java memiliki mekanisme exception handling yang memungkinkan program untuk menangani kesalahan dan kondisi yang tidak diinginkan pada waktu eksekusi. Dengan exception handling, program dapat menangani situasi yang tidak diharapkan dengan cara yang terkontrol dan menghindari program berhenti atau mengalami kerusakan.

1. Multithreading

Java menyediakan kemampuan untuk membuat program multithreaded. Hal ini memungkinkan program untuk menjalankan beberapa tugas secara bersamaan, sehingga meningkatkan kinerja dan efisiensi program. Fitur multithreading ini sangat berguna dalam aplikasi yang membutuhkan pengolahan data yang besar.

1. API dan Libraries

Java memiliki banyak API dan libraries yang tersedia, yang memungkinkan pengembang untuk mengembangkan aplikasi dengan lebih cepat dan efisien. Library tersebut mencakup berbagai jenis, seperti GUI (Graphical User Interface), pengolahan data, koneksi jaringan, pengolahan gambar, dan banyak lagi.

1. Keamanan

Java dirancang dengan keamanan sebagai salah satu fiturnya. Java memiliki mekanisme keamanan yang kuat yang dapat mencegah program dari penyalahgunaan atau serangan hacker. Java juga memiliki kemampuan untuk mengenkripsi data yang sensitif untuk melindunginya dari penyerang.

1. Polymorphism

Polymorphism adalah konsep dalam pemrograman berorientasi objek yang memungkinkan objek dari kelas yang berbeda untuk memiliki perilaku yang sama. Java mendukung polimorfisme, sehingga objek dari kelas yang berbeda dapat memiliki perilaku yang sama.

1. Interface

Java juga mendukung penggunaan interface. Interface adalah kelas abstrak yang hanya menyediakan deklarasi metode tanpa implementasi. Interface memungkinkan pengembang untuk membuat kode yang lebih bersih dan mudah dipelihara, karena memisahkan deklarasi dari implementasi.

1. Garbage Collection

Java memiliki fitur garbage collection yang menjaga memori agar tetap bersih. Dalam bahasa Java, setiap kali sebuah objek tidak lagi digunakan, garbage collector menghapusnya secara otomatis untuk membebaskan memori. Hal ini sangat membantu untuk mencegah memory leak atau kebocoran memori yang dapat menyebabkan aplikasi gagal.

1. Networking

Java memiliki fitur networking yang kuat dan memungkinkan pengembang untuk membuat aplikasi yang berkomunikasi melalui jaringan. Java menyediakan API yang mudah digunakan untuk membuat protokol jaringan seperti HTTP, FTP, dan SMTP.

1. Dynamic Binding

Java memiliki fitur dynamic binding atau late binding yang memungkinkan program untuk memilih implementasi yang tepat untuk metode yang dipanggil pada waktu eksekusi. Hal ini berguna dalam situasi di mana implementasi metode yang tepat mungkin tidak diketahui pada saat kompilasi.

1. Generic

Java memiliki fitur generic yang memungkinkan pengembang untuk membuat kelas dan metode yang dapat digunakan untuk berbagai jenis objek. Fitur ini memungkinkan pengembang untuk membuat kode yang lebih bersih dan efisien, dan mengurangi jumlah kode yang harus ditulis.

1. Reflection

Java juga memiliki fitur reflection, yang memungkinkan program untuk mengakses informasi pada waktu eksekusi tentang kelas, metode, dan variabel dalam program. Hal ini memungkinkan pengembang untuk membuat kode yang lebih dinamis dan fleksibel.

1. Annotations

Java memiliki fitur annotations yang memungkinkan pengembang untuk menambahkan metadata ke kelas dan metode dalam program. Metadata ini dapat digunakan untuk menjelaskan cara penggunaan kelas atau metode atau menambahkan informasi tambahan tentang program.

1. Lambda Expression

Java 8 memperkenalkan fitur lambda expression yang memungkinkan pengembang untuk menulis kode yang lebih pendek dan bersih. Lambda expression memungkinkan pengembang untuk menuliskan fungsi anonim yang dapat digunakan sebagai parameter dalam metode.

1. Stream API

Java 8 juga memperkenalkan fitur Stream API yang memungkinkan pengembang untuk memproses data secara fungsional dan paralel. Stream API menyediakan metode untuk mengakses, memanipulasi, dan mengelola data yang lebih mudah dan lebih efisien.

1. Modularitas

Java 9 memperkenalkan fitur modularitas yang memungkinkan pengembang untuk memecah program menjadi modul yang lebih kecil dan lebih mudah dipelihara. Fitur ini membantu mengurangi kompleksitas program dan memungkinkan pengembang untuk mengembangkan aplikasi dengan lebih cepat dan efisien.

1. JShell

Java 9 juga memperkenalkan JShell, yang merupakan lingkungan pemrograman interaktif yang memungkinkan pengembang untuk menulis dan menjalankan kode Java secara langsung. JShell memungkinkan pengembang untuk mengeksplorasi dan menguji kode dengan cepat dan mudah.

1. Machine Learning

Java juga telah memperkenalkan beberapa fitur baru dalam mendukung pengembangan aplikasi machine learning. Java menyediakan library seperti Deeplearning4j, Weka, dan Mahout yang memungkinkan pengembang untuk mengembangkan aplikasi machine learning yang kuat dan efisien menggunakan bahasa Java.

1. JAR

Java memiliki fitur JAR (Java Archive) yang memungkinkan pengembang untuk menggabungkan file kelas, sumber daya, dan metadata dalam satu file. JAR dapat digunakan untuk mendistribusikan aplikasi atau library dengan mudah dan efisien.

1. Debugging

Java menyediakan dukungan yang kuat untuk debugging, termasuk integrasi dengan lingkungan pengembangan terpadu (IDE) seperti Eclipse dan NetBeans. Debugging sangat penting dalam pengembangan aplikasi karena memungkinkan pengembang untuk mengidentifikasi dan memperbaiki kesalahan dalam kode.

1. Security

Java memiliki fitur keamanan yang kuat, termasuk sandboxing dan kebijakan keamanan. Sandbox adalah lingkungan terisolasi yang membatasi akses aplikasi ke sumber daya sistem dan meminimalkan risiko dari kode yang tidak dipercayai. Kebijakan keamanan memungkinkan pengembang untuk menentukan tingkat akses yang diperbolehkan untuk aplikasi.

1. Platform-Independent

Java dirancang untuk menjadi platform-independent, yang berarti bahwa kode Java dapat dijalankan pada berbagai platform seperti Windows, Mac, dan Linux tanpa perlu mengubah kode. Hal ini memudahkan distribusi dan penggunaan aplikasi, karena pengguna dapat menginstal aplikasi pada platform apa pun tanpa perlu melakukan pengaturan khusus.

1. Community

Java memiliki komunitas pengembang yang besar dan aktif. Komunitas ini berkontribusi pada pengembangan Java, menulis library dan framework, dan memberikan dukungan dan sumber daya untuk pengembang Java. Komunitas Java juga memungkinkan pengembang untuk berbagi pengetahuan dan pengalaman, memecahkan masalah, dan meningkatkan keterampilan mereka dalam pengembangan Java.

## **2.3 Percabangan Java**

Percabangan (branching) adalah salah satu konsep dasar dalam pemrograman yang memungkinkan program untuk memilih tindakan yang berbeda berdasarkan kondisi tertentu. Dalam bahasa pemrograman Java, ada beberapa jenis percabangan yang dapat digunakan, termasuk percabangan if-else, percabangan switch-case, dan percabangan operator ternary. Dalam penjelasan ini, kami akan membahas masing-masing jenis percabangan dengan lebih detail.

1. Percabangan if-else

Percabangan if-else digunakan untuk mengambil keputusan berdasarkan nilai dari kondisi tertentu. Bentuk dasar dari percabangan if-else adalah sebagai berikut:

if (kondisi) {

// blok kode dijalankan jika kondisi benar

} else {

// blok kode dijalankan jika kondisi salah

}

Kondisi dapat berupa ekspresi boolean atau hasil dari pembandingan variabel atau ekspresi. Jika kondisi benar, maka blok kode di dalam if akan dijalankan, sedangkan jika kondisi salah, maka blok kode di dalam else akan dijalankan.

Contoh:

int x = 5;

if (x > 10) {

System.out.println("x is greater than 10");

} else {

System.out.println("x is less than or equal to 10");

}

Output:

x is less than or equal to 10

1. Percabangan switch-case

Percabangan switch-case digunakan untuk mengambil keputusan berdasarkan nilai dari sebuah variabel atau ekspresi. Bentuk dasar dari percabangan switch-case adalah sebagai berikut:

switch (variabel) {

case nilai1:

// blok kode dijalankan jika variabel sama dengan nilai1

break;

case nilai2:

// blok kode dijalankan jika variabel sama dengan nilai2

break;

...

default:

// blok kode dijalankan jika variabel tidak sama dengan semua nilai di atas

}

Setiap nilai yang ingin diuji harus ditentukan dalam blok case. Jika variabel sama dengan nilai tertentu, maka blok kode di dalam case tersebut akan dijalankan. Jika tidak ada case yang cocok, maka blok kode di dalam default akan dijalankan.

Contoh:

char grade = 'B';

switch (grade) {

case 'A':

System.out.println("Excellent!");

break;

case 'B':

case 'C':

System.out.println("Well done");

break;

case 'D':

System.out.println("You passed");

break;

case 'F':

System.out.println("Better try again");

break;

default:

System.out.println("Invalid grade");

}

Output:

Well done

1. Percabangan operator ternary

Percabangan operator ternary merupakan singkat dan mudah digunakan, dan digunakan untuk membuat keputusan berdasarkan kondisi yang sama seperti if-else. Bentuk dasar dari percabangan operator ternary adalah sebagai berikut:

variabel = (kondisi) ? nilai\_jika\_benar : nilai\_jika\_salah;

Jika kondisi benar, maka variabel akan diisi dengan nilai\_jika\_benar, sedangkan jika kondisi salah, maka variabel akan diisi dengan nilai jika salah.

Contoh:

int x = 5;

String result = (x > 10) ? "x is greater than 10" : "x is less than or equal to 10";

System.out.println(result);

Output:

x is less than or equal to 10

Keuntungan dari penggunaan operator ternary adalah dapat menghasilkan kode yang lebih singkat dan mudah dibaca. Namun, operator ternary sebaiknya digunakan hanya untuk kondisi yang sederhana dan tidak terlalu kompleks, karena dapat menyulitkan pembaca kode.

Selain jenis percabangan yang sudah dijelaskan di atas, ada juga beberapa konsep yang berhubungan dengan percabangan, seperti nested if-else dan penggunaan operator logika. Nested if-else digunakan untuk mengkombinasikan beberapa percabangan if-else dalam satu blok kode, sedangkan operator logika digunakan untuk mengkombinasikan kondisi boolean.

Contoh penggunaan nested if-else:

int x = 5;

int y = 10;

if (x > 0) {

if (y > 0) {

System.out.println("x and y are both positive");

} else {

System.out.println("x is positive but y is not");

}

} else {

System.out.println("x is not positive");

}

Output:

x and y are both positive

Contoh penggunaan operator logika:

int x = 5;

int y = 10;

if (x > 0 && y > 0) {

System.out.println("x and y are both positive");

} else if (x > 0 || y > 0) {

System.out.println("at least one of x and y is positive");

} else {

System.out.println("neither x nor y is positive");

}

Output:

x and y are both positive

Dalam pemrograman Java, percabangan sangat penting untuk membuat program yang dapat mengambil keputusan berdasarkan kondisi tertentu. Penggunaan percabangan yang tepat akan membuat program menjadi lebih efisien, singkat, dan mudah dipahami. Namun, penggunaan percabangan yang berlebihan atau tidak tepat dapat membuat program menjadi sulit dipahami dan sulit dipelihara. Oleh karena itu, pemrogram harus selalu memperhatikan penggunaan percabangan dalam kode mereka dan memastikan bahwa kode tersebut mudah dipahami dan efisien.

## **2.4 Akses Modifier Java**

Akses Modifier di Java adalah suatu fitur yang digunakan untuk mengatur tingkat akses dari class, method, dan variable dalam suatu program. Terdapat empat jenis akses modifier di Java, yaitu public, private, protected, dan default. Setiap akses modifier memiliki aturan tertentu yang perlu diperhatikan dalam penggunaannya.

1. Akses Modifier Public

Akses modifier public memungkinkan class, method, dan variable dapat diakses dari seluruh class dalam program. Dalam penggunaannya, class, method, dan variable yang diberi akses modifier public dapat diakses dari class lain dengan cara memanggil namanya. Selain itu, class yang diberi akses modifier public juga dapat diakses oleh class yang berada di luar package.

Contoh penggunaan akses modifier public adalah sebagai berikut:

public class MyClass {

public int myPublicInt = 10;

public void myPublicMethod() {

System.out.println("This is a public method.");

}

}

Dalam contoh di atas, class MyClass diberi akses modifier public sehingga dapat diakses dari class lain. Selain itu, variabel myPublicInt dan method myPublicMethod juga diberi akses modifier public, sehingga dapat diakses dari class lain dengan cara memanggil namanya. Sebagai contoh, jika terdapat class lain yang ingin mengakses variabel myPublicInt dan method myPublicMethod, dapat dilakukan dengan cara sebagai berikut:

public class MyOtherClass {

public static void main(String[] args) {

MyClass obj = new MyClass();

System.out.println(obj.myPublicInt);

obj.myPublicMethod();

}

}

Dalam contoh di atas, class MyOtherClass dapat mengakses variabel myPublicInt dan method myPublicMethod dari class MyClass karena keduanya diberi akses modifier public. Variabel myPublicInt dapat diakses dengan cara memanggil namanya melalui objek dari class MyClass, sedangkan method myPublicMethod dapat dipanggil dengan cara yang sama.

Penggunaan akses modifier public sangat berguna ketika variabel atau method perlu diakses dari class lain dalam program. Namun, perlu diingat bahwa penggunaan akses modifier public juga dapat menyebabkan kerentanan terhadap kesalahan dalam program, terutama jika variabel atau method yang diberi akses public digunakan secara tidak tepat oleh class lain. Oleh karena itu, perlu dipertimbangkan dengan hati-hati dalam penggunaannya.

1. Akses Modifier Private

Akses modifier private memungkinkan class, method, dan variable hanya dapat diakses oleh class yang sama. Dalam penggunaannya, class, method, dan variable yang diberi akses modifier private tidak dapat diakses oleh class lain, kecuali dengan bantuan dari method yang berada di dalam class tersebut. Sehingga, akses modifier private memberikan perlindungan terhadap variabel dan method dari perubahan yang tidak diinginkan.

Contoh penggunaan akses modifier private adalah sebagai berikut:

public class MyClass {

private int myPrivateInt = 10;

private void myPrivateMethod() {

System.out.println("This is a private method.");

}

public void accessMyPrivateMembers() {

System.out.println(myPrivateInt);

myPrivateMethod();

}

}

Dalam contoh di atas, variabel myPrivateInt dan method myPrivateMethod diberi akses modifier private, sehingga hanya dapat diakses dari dalam class MyClass. Namun, terdapat method baru bernama accessMyPrivateMembers yang memiliki akses modifier public dan dapat digunakan untuk mengakses variabel dan method yang bersifat private. Method accessMyPrivateMembers dapat diakses dari class lain dan digunakan untuk mengakses variabel myPrivateInt dan method myPrivateMethod dengan cara sebagai berikut:

public class MyOtherClass {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.accessMyPrivateMembers();

}

}

Dalam contoh di atas, class MyOtherClass dapat mengakses method accessMyPrivateMembers dari class MyClass karena method tersebut diberi akses modifier public. Dalam method accessMyPrivateMembers, variabel myPrivateInt dan method myPrivateMethod yang awalnya bersifat private dapat diakses karena method tersebut berada dalam class yang sama.

Penggunaan akses modifier private sangat berguna untuk menghindari kesalahan dalam program dan memperkuat enkapsulasi dalam sebuah class. Dengan menggunakan akses modifier private, variabel atau method yang tidak perlu diakses dari luar class dapat disembunyikan dan hanya dapat diakses oleh class tersebut. Namun, perlu diingat bahwa akses modifier private juga dapat membatasi fleksibilitas dalam program, terutama jika variabel atau method yang bersifat private kemudian diperlukan dalam pengembangan selanjutnya.

1. Akses Modifier Protected

Akses modifier protected memungkinkan class, method, dan variable hanya dapat diakses oleh class yang sama dan subclass-nya. Dalam penggunaannya, class, method, dan variable yang diberi akses modifier protected dapat diakses oleh class yang berada di dalam package yang sama atau subclass-nya. Akses modifier protected memungkinkan penggunaan kelas dan method secara terbatas dan memberikan kontrol yang lebih besar atas variabel dan method.

Contoh penggunaan akses modifier protected adalah sebagai berikut:

public class Person {

protected String name;

protected int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

protected void sayHello() {

System.out.println("Hello, my name is " + name);

}

}

public class Student extends Person {

private int studentID;

public Student(String name, int age, int studentID) {

super(name, age);

this.studentID = studentID;

}

public void introduce() {

sayHello();

System.out.println("I am a student with ID " + studentID);

}

}

Dalam contoh di atas, class Person memiliki akses modifier protected pada variabel name dan age, serta method sayHello. Ini berarti variabel name dan age dapat diakses oleh class yang diturunkan (subclass), tetapi tidak dapat diakses oleh class yang tidak terkait (non-subclass). Method sayHello juga diberi akses modifier protected, yang berarti method tersebut hanya dapat diakses oleh class yang diturunkan dan class yang berada dalam package yang sama.

Class Student diturunkan dari class Person dan memiliki akses ke variabel name dan age melalui konstruktor super(), serta dapat memanggil method sayHello melalui method introduce(). Method introduce() diberi akses modifier public sehingga dapat diakses oleh class lain, tetapi method sayHello hanya dapat diakses oleh class yang diturunkan dari class Person atau class yang berada dalam package yang sama.

Penggunaan akses modifier protected berguna untuk memungkinkan penggunaan variabel atau method dari class yang diturunkan, tetapi tidak ingin membuat variabel atau method tersebut menjadi public. Akses modifier protected juga berguna untuk meningkatkan enkapsulasi dengan membatasi akses ke variabel dan method tertentu, dan juga memudahkan pengembangan selanjutnya dengan memberikan akses khusus ke class yang diturunkan.

1. Akses Modifier Default

Akses modifier default adalah akses modifier yang tidak ditentukan secara eksplisit oleh pengembang. Dalam penggunaannya, class, method, dan variable yang tidak diberi akses modifier akan otomatis dianggap memiliki akses modifier default. Akses modifier default memungkinkan class, method, dan variable dapat diakses dari class yang berada dalam package yang sama. Namun, class, method, dan variable yang diberi akses modifier default tidak dapat diakses dari package yang berbeda.

Contoh penggunaan akses modifier default (atau tanpa akses modifier) adalah sebagai berikut:

package com.example;

class Person {

String name;

int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

void sayHello() {

System.out.println("Hello, my name is " + name);

}

}

public class Main {

public static void main(String[] args) {

Person person = new Person("John", 25);

person.sayHello();

}

}

Dalam contoh di atas, class Person didefinisikan tanpa akses modifier, yang berarti class tersebut memiliki akses modifier default. Ini berarti class Person hanya dapat diakses oleh class-class yang berada dalam package yang sama. Karena class Main berada dalam package yang sama dengan class Person, maka class Main dapat mengakses class Person dan membuat objek dari class tersebut.

Variabel name dan age dalam class Person didefinisikan tanpa akses modifier, sehingga variabel-variabel tersebut juga memiliki akses modifier default. Ini berarti variabel-variabel tersebut hanya dapat diakses oleh class-class yang berada dalam package yang sama.

Method sayHello() dalam class Person didefinisikan dengan akses modifier default, sehingga method tersebut hanya dapat diakses oleh class-class yang berada dalam package yang sama. Method ini dipanggil di dalam method main() pada class Main dan dapat diakses karena class Main berada dalam package yang sama dengan class Person.

Penggunaan akses modifier default berguna untuk membatasi akses ke variabel atau method tertentu hanya untuk class-class yang berada dalam package yang sama. Akses modifier ini membantu meningkatkan enkapsulasi dan mencegah penggunaan variabel atau method oleh class-class yang tidak berhubungan.

Penggunaan akses modifier sangat penting dalam pengembangan program dengan Java, karena dapat membantu mengatur tingkat akses dari class, method, dan variable sehingga dapat diakses dengan cara yang diinginkan. Selain itu, penggunaan akses modifier juga dapat membantu mencegah perubahan yang tidak diinginkan pada variabel dan method dari class yang berbeda. Dalam penggunaannya, pengembang harus memahami dengan baik aturan dan konsep di balik setiap akses modifier untuk memastikan program dapat berjalan dengan benar.

## **2.5 Kelas Abstrak dan Antarmuka**

Dalam bahasa pemrograman Java, terdapat dua konsep penting yaitu kelas abstrak (abstract class) dan antarmuka (interface). Kedua konsep ini memungkinkan programmer untuk melakukan abstraksi pada sebuah program dengan cara yang berbeda.

**Kelas Abstrak (Abstract Class)**

Kelas abstrak adalah kelas yang tidak dapat diinstansiasi, artinya kita tidak dapat membuat objek dari kelas tersebut. Kelas abstrak digunakan sebagai kerangka dasar untuk membuat kelas turunan (sub-class) yang lebih spesifik. Kelas abstrak sendiri tidak memiliki implementasi (kode) untuk semua method yang dimilikinya. Oleh karena itu, untuk membuat sebuah kelas abstrak, kita harus menyediakan setidaknya satu method abstrak (abstract method).

Pada umumnya, kelas abstrak digunakan ketika kita ingin membuat kerangka dasar (template) untuk kelas-kelas turunan yang akan dibuat. Contoh sederhana dari kelas abstrak adalah sebagai berikut:

abstract class Animal {

String name;

int age;

public Animal(String name, int age) {

this.name = name;

this.age = age;

}

abstract void makeSound();

}

Kelas Animal di atas adalah sebuah kelas abstrak yang memiliki dua variabel (name dan age) serta satu method abstrak (makeSound()). Method makeSound() tidak memiliki implementasi karena kelas Animal hanya bertindak sebagai kerangka dasar untuk kelas-kelas turunannya.

Selain memiliki method abstrak, kelas abstrak juga dapat memiliki method biasa (non-abstrak) yang memiliki implementasi. Method biasa tersebut dapat digunakan oleh kelas-kelas turunan jika diperlukan.

Untuk membuat kelas turunan dari sebuah kelas abstrak, kita harus meng-extend kelas abstrak tersebut dan menyediakan implementasi untuk semua method abstrak yang dimilikinya. Contoh implementasi kelas turunan dari kelas Animal di atas adalah sebagai berikut:

class Dog extends Animal {

public Dog(String name, int age) {

super(name, age);

}

void makeSound() {

System.out.println("Bark!");

}

}

Kelas Dog di atas merupakan kelas turunan dari kelas Animal dan harus menyediakan implementasi untuk method makeSound() karena method tersebut merupakan method abstrak dari kelas Animal. Dalam contoh di atas, kelas Dog memiliki implementasi untuk method makeSound() dengan mencetak "Bark!" ke layar.

**Antarmuka (Interface)**

Antarmuka adalah konsep yang mirip dengan kelas abstrak. Namun, antarmuka hanya menyediakan definisi (signature) dari method dan tidak memiliki implementasi (kode) dari method tersebut. Antarmuka digunakan untuk menentukan perilaku (behavior) dari kelas-kelas yang mengimplementasikan antarmuka tersebut.

Dalam Java, sebuah kelas dapat mengimplementasikan satu atau lebih antarmuka. Hal ini memungkinkan programmer untuk membuat kelas-kelas yang berbeda dengan perilaku yang sama. Sebagai contoh, kita dapat membuat kelas-kelas yang dapat disimpan dalam koleksi (collection) dengan berbagai macam perilaku yang berbeda.

Contoh sederhana dari sebuah antarmuka adalah sebagai berikut:

interface Shape {

double getArea();

double getPerimeter();

}

Sementara kelas abstrak menyediakan kerangka kerja umum, antarmuka (interface) menyediakan kontrak umum yang harus dipenuhi oleh kelas lain. Antarmuka pada dasarnya adalah himpunan metode dan konstanta yang dideklarasikan tanpa implementasi. Kelas yang mengimplementasikan suatu antarmuka harus menyediakan implementasi untuk semua metode yang dideklarasikan dalam antarmuka.

Dalam Java, sebuah antarmuka dideklarasikan dengan menggunakan kata kunci **interface**. Berikut adalah contoh deklarasi antarmuka:

public interface Printable {

void print();

}

Dalam contoh di atas, antarmuka **Printable** hanya memiliki satu metode **print()**. Kelas yang mengimplementasikan antarmuka **Printable** harus menyediakan implementasi untuk metode **print()**.

Perlu dicatat bahwa sebuah kelas dapat mengimplementasikan banyak antarmuka. Berikut adalah contoh kelas yang mengimplementasikan dua antarmuka:

public class MyClass implements Printable, Serializable {

public void print() {

// implementasi metode print

}

}

Dalam contoh di atas, kelas **MyClass** mengimplementasikan antarmuka **Printable** dan antarmuka **Serializable**. Kelas ini harus menyediakan implementasi untuk metode **print()** yang dideklarasikan dalam antarmuka **Printable**.

Saat menerapkan antarmuka, setiap metode yang dideklarasikan dalam antarmuka harus diimplementasikan. Jika suatu kelas tidak menyediakan implementasi untuk metode antarmuka, maka kelas tersebut harus dideklarasikan sebagai kelas abstrak.

Berikut adalah contoh penggunaan antarmuka dalam program Java:

public interface Shape {

void draw();

}

public class Rectangle implements Shape {

public void draw() {

System.out.println("Menggambar persegi panjang");

}

}

public class Circle implements Shape {

public void draw() {

System.out.println("Menggambar lingkaran");

}

}

public class Main {

public static void main(String[] args) {

Shape s1 = new Rectangle();

Shape s2 = new Circle();

s1.draw();

s2.draw();

}

}

Output program di atas akan menjadi sebagai berikut:

Menggambar persegi panjang

Menggambar lingkaran

Dalam contoh di atas, antarmuka **Shape** dideklarasikan dengan satu metode **draw()**. Kelas **Rectangle** dan **Circle** mengimplementasikan antarmuka **Shape** dan memberikan implementasi untuk metode **draw()**. Dalam metode **main()**, objek dari kelas **Rectangle** dan **Circle** diciptakan dan metode **draw()** dipanggil pada setiap objek.

# BAB III

# PENUTUPAN

**3. 1 Kelebihan dan Kekurangan Mempelajari Bahasa Java**

Java adalah bahasa pemrograman yang sangat populer di kalangan programmer dan digunakan dalam pengembangan aplikasi desktop, mobile, web, dan enterprise. Ada banyak kelebihan dan kekurangan mempelajari bahasa Java, dan dalam artikel ini kami akan membahas secara rinci dalam 500 kata.

**Kelebihan Mempelajari Bahasa Java**

1. Mudah Dipelajari

Java adalah bahasa pemrograman yang relatif mudah dipelajari. Bahasa ini dirancang untuk mudah dipahami dan digunakan oleh programmer yang baru memulai. Selain itu, Java memiliki banyak dokumentasi dan tutorial online yang tersedia, sehingga programmer dapat mempelajari bahasa ini secara mandiri.

1. Platform Independent

Salah satu kelebihan utama Java adalah dapat berjalan pada berbagai platform dan sistem operasi. Bahasa ini mengikuti konsep "write once, run anywhere", yang berarti kode Java dapat dijalankan pada berbagai platform tanpa perlu mengubah kode program. Hal ini membuat Java sangat fleksibel dan populer dalam pengembangan aplikasi yang memerlukan portabilitas dan interoperabilitas.

1. Keamanan Tinggi

Java memiliki keamanan yang tinggi dan dapat diandalkan. Dalam bahasa Java, terdapat mekanisme keamanan yang melindungi sistem dari program berbahaya, seperti sandboxing dan penggunaan class loader untuk membatasi akses ke kode program.

1. Kuat dan Scalable

Java memiliki fitur yang kuat dan lengkap, serta dapat digunakan dalam pengembangan aplikasi yang berskala besar. Bahasa ini mendukung pemrograman berorientasi objek, pemrograman berbasis komponen, dan penggunaan thread. Java juga memiliki dukungan yang kuat untuk database dan jaringan, sehingga dapat digunakan untuk pengembangan aplikasi enterprise.

1. Komunitas Pengembang yang Luas

Java memiliki komunitas pengembang yang luas dan aktif, serta banyak sumber daya online yang tersedia. Komunitas ini menyediakan dukungan dan bantuan untuk programmer pemula, serta berbagai komponen dan library yang dapat digunakan dalam pengembangan aplikasi.

**Kekurangan Mempelajari Bahasa Java**

1. Lambat

Salah satu kekurangan Java adalah lambatnya kinerja dibandingkan bahasa pemrograman lain seperti C atau C++. Meskipun ada beberapa upaya untuk meningkatkan kinerja Java, namun bahasa ini masih tidak secepat bahasa pemrograman lain dalam hal kinerja.

1. Memerlukan Memori yang Besar

Java memerlukan memori yang lebih besar dibandingkan dengan bahasa pemrograman lain, terutama ketika digunakan dalam pengembangan aplikasi berbasis web. Hal ini dapat mengakibatkan konsumsi memori yang tinggi dan dapat mempengaruhi kinerja aplikasi.

1. Kurangnya Dukungan untuk Pemrograman Low-Level

Java tidak mendukung pemrograman low-level seperti bahasa C atau assembly language. Hal ini membuat Java kurang cocok untuk pengembangan perangkat lunak tingkat rendah atau sistem operasi.

1. Kurangnya Dukungan untuk Pemrograman Sistem Operasi

Java tidak memiliki dukungan yang kuat untuk pemrograman sistem operasi seperti Windows atau Linux. Sebagai hasilnya, pengembangan aplikasi sistem operasi, driver perangkat keras, dan aplikasi yang sangat tergantung pada sistem operasi tidak dapat dilakukan dengan bahasa Java.

1. Sulit dalam Pengelolaan Memori

Java memiliki sistem manajemen memori yang sangat efisien dan mudah digunakan, yang dikenal sebagai garbage collector. Namun, penggunaan garbage collector ini dapat memakan waktu dan sumber daya CPU, dan juga dapat mempengaruhi kinerja aplikasi. Selain itu, penggunaan garbage collector dapat mempengaruhi kecepatan akses ke memori dan memori yang dibutuhkan untuk menjalankan aplikasi.

1. Sulit untuk Meningkatkan Kinerja pada Aplikasi yang Ada

Java adalah bahasa pemrograman interpretatif, yang berarti aplikasi dikompilasi menjadi bytecode dan dijalankan dalam lingkungan virtual. Hal ini membuatnya sulit untuk meningkatkan kinerja aplikasi yang ada, karena bahasa ini tidak mendukung pemrograman langsung ke mesin atau kernel sistem operasi.

**3.2 Kesimpulan**

Java adalah bahasa pemrograman yang populer dan banyak digunakan di berbagai industri. Ada banyak kelebihan dan kekurangan dalam mempelajari bahasa ini. Kelebihan utama Java adalah mudah dipelajari, platform independen, memiliki keamanan yang tinggi, kuat dan scalable, dan memiliki komunitas pengembang yang luas. Namun, Java juga memiliki beberapa kekurangan, seperti lambat, memerlukan memori yang besar, kurangnya dukungan untuk pemrograman low-level dan sistem operasi, serta sulit dalam pengelolaan memori dan meningkatkan kinerja aplikasi yang ada. Sebelum memulai mempelajari Java, penting untuk mempertimbangkan kelebihan dan kekurangan ini untuk memastikan bahasa ini sesuai dengan kebutuhan pengembangan aplikasi.

# DAFTAR PUSTAKA

* <https://www.petanikode.com/tutorial/java/>
* <https://glints.com/id/lowongan/bahasa-pemrograman-java/>
* <https://www.dicoding.com/academies/60>
* <https://www.duniailkom.com/tutorial-belajar-bahasa-pemrograman-java-untuk-pemula/>
* <https://aws.amazon.com/id/what-is/java/>
* <https://academy.alterra.id/blog/5-program-java-sederhana/>
* <https://www.qubisa.com/article/pengenalan-bahasa-pemrograman-java-untuk-pemula>
* <https://www.jagoanhosting.com/blog/java-adalah/>
* <https://azure.microsoft.com/id-id/resources/cloud-computing-dictionary/what-is-java-programming-language/>
* <https://surabaya.proxsisgroup.com/pengertian-java/>
* <https://toghr.com/bahasa-pemrograman-java-dan-contoh-programnya/>
* <http://repository.um-surabaya.ac.id/640/3/Bab_II.pdf>
* <https://www.niagahoster.co.id/blog/java-adalah/>