MAKALAH

PEWARISAN DALAM BAHASA JAVA



Oleh :

Diki Candra

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEMAWE

2022/2023

DAFTAR ISI

[BAB I 3](#_Toc135476963)

[PENDAHULUAN 3](#_Toc135476964)

[**1.1. Latar Belakang** 3](#_Toc135476965)

[**1.2. Tujuan Penulisan Makalah** 4](#_Toc135476966)

[**1.3. Ruang Lingkup Makalah** 4](#_Toc135476967)

[**1.4. Metode Penulisan** 4](#_Toc135476968)

[BAB II 5](#_Toc135476969)

[KONSEP DASAR PEWARISAN 5](#_Toc135476970)

[**2.1. Pengertian Pewarisan dalam Pemrograman Berorientasi Objek (PBO)** 5](#_Toc135476971)

[**2.2. Tujuan dan Manfaat Pewarisan dalam PBO** 5](#_Toc135476972)

[**2.3. Contoh Kasus Penggunaan Pewarisan dalam Java** 6](#_Toc135476973)

[BAB III 10](#_Toc135476974)

[SYNTAX DAN NOTASI PEWARISAN 10](#_Toc135476975)

[**3.1. Keyword "extends" dan "super"** 10](#_Toc135476976)

[**3.2. Pembuatan dan Penggunaan Kelas Turunan (Subclass)** 12](#_Toc135476977)

[**3.3. Pewarisan Multiple dalam Java** 14](#_Toc135476978)

[BAB IV 17](#_Toc135476979)

[KONSTRUKTOR DALAM PEWARISAN 17](#_Toc135476980)

[**4.1. Konstruktor dalam Kelas Induk (Superclass)** 17](#_Toc135476981)

[**4.2. Konstruktor dalam Kelas Turunan (Subclass)** 19](#_Toc135476982)

[**4.3. Memanggil Konstruktor Kelas Induk dari Kelas Turunan** 21](#_Toc135476983)

[BAB V 24](#_Toc135476984)

[CONTOH IMPLEMENTASI PEWARISAN 24](#_Toc135476985)

[**5.1. Studi Kasus: Kelas Binatang dan Kelas Turunannya** 24](#_Toc135476986)

[**5.2. Analisis Implementasi Pewarisan dalam Kasus Tersebut** 27](#_Toc135476987)

[BAB VI 29](#_Toc135476988)

[KESIMPULAN 29](#_Toc135476989)

# BAB I

# PENDAHULUAN

## **1.1. Latar Belakang**

Pewarisan merupakan salah satu konsep utama dalam pemrograman berorientasi objek yang memungkinkan pengguna untuk membuat hierarki kelas dan mewarisi sifat dan perilaku dari kelas yang sudah ada.

Pemahaman yang baik tentang pewarisan dalam Java sangat penting bagi para pengembang perangkat lunak, karena dapat membantu dalam pengorganisasian kode, meningkatkan efisiensi, dan memungkinkan penggunaan kembali kode yang telah ada. Dalam makalah ini, akan dijelaskan secara rinci mengenai konsep dasar pewarisan, syntax dan notasi yang digunakan dalam pewarisan Java, konstruktor dalam pewarisan, serta konsep overriding dan polimorfisme.

Selain itu, makalah ini juga akan memberikan contoh implementasi pewarisan dalam Java melalui studi kasus tentang kelas binatang dan turunannya. Tujuan dari makalah ini adalah memberikan pemahaman yang komprehensif tentang pewarisan dalam pemrograman Java, memperjelas konsep, serta memberikan pemahaman yang praktis melalui contoh implementasi.

Dengan pemahaman yang baik tentang pewarisan dalam Java, para pengembang perangkat lunak dapat memanfaatkannya dengan efektif untuk meningkatkan kualitas dan efisiensi kode program yang mereka tulis.

## **1.2. Tujuan Penulisan Makalah**

1. Memberikan pemahaman yang komprehensif tentang konsep dan implementasi pewarisan dalam pemrograman Java.
2. Menjelaskan syntax, notasi, dan mekanisme yang digunakan dalam pewarisan Java.
3. Menggambarkan hubungan antara kelas induk (superclass) dan kelas turunan (subclass) dalam pewarisan.
4. Memahami penggunaan konstruktor dalam pewarisan Java dan cara memanggil konstruktor kelas induk dari kelas turunan.
5. Membahas konsep overriding dan polimorfisme dalam konteks pewarisan.
6. Menggambarkan contoh implementasi pewarisan dalam Java melalui studi kasus kelas binatang dan turunannya.
7. Menyoroti kelebihan dan kelemahan dari pewarisan dalam pemrograman Java.
8. Memberikan pemahaman tentang aksesibilitas atribut dan metode dalam pewarisan.
9. Menganalisis relevansi dan manfaat pewarisan dalam pemrograman Java.
10. Memberikan saran-saran untuk penerapan pewarisan yang efektif dalam pengembangan perangkat lunak.

Secara keseluruhan, tujuan utama dari makalah ini adalah memberikan pemahaman yang jelas dan mendalam tentang pewarisan dalam pemrograman Java, serta memberikan wawasan praktis dan contoh implementasi untuk membantu pembaca memahami dan memanfaatkannya dengan baik dalam pengembangan perangkat lunak.

## **1.3. Ruang Lingkup Makalah**

1. Konsep dasar pewarisan dalam pemrograman Java, termasuk pengertian, tujuan, dan manfaatnya.
2. Syntax dan notasi yang digunakan dalam pewarisan Java, termasuk penggunaan keyword "extends" dan "super".
3. Pembahasan mengenai kelas turunan (subclass) dan hubungannya dengan kelas induk (superclass).
4. Penjelasan mengenai konstruktor dalam pewarisan Java dan cara memanggil konstruktor kelas induk dari kelas turunan.
5. Pengertian dan penerapan overriding metode dalam pewarisan, serta penggunaan "super" dalam overriding.
6. Konsep polymorphism dalam konteks pewarisan dalam pemrograman Java.
7. Contoh implementasi pewarisan dalam Java melalui studi kasus kelas binatang dan turunannya.
8. Pembahasan mengenai aksesibilitas atribut dan metode dalam pewarisan.
9. Kelebihan dan kelemahan dari pewarisan dalam pemrograman Java.
10. Analisis relevansi dan manfaat pewarisan dalam pemrograman Java.

Ruang lingkup makalah ini terfokus pada pemahaman konsep dan implementasi pewarisan dalam pemrograman Java. Makalah ini tidak membahas konsep-konsep lain dalam pemrograman berorientasi objek, seperti enkapsulasi, polimorfisme, atau konsep lainnya yang tidak langsung terkait dengan pewarisan. Selain itu, ruang lingkup makalah ini juga tidak mencakup aplikasi pewarisan dalam konteks spesifik atau bahasan teknis terkait pemrograman Java yang lebih mendalam.

Dengan membatasi ruang lingkup pada topik-topik tersebut, makalah ini diharapkan dapat memberikan pemahaman yang komprehensif dan praktis tentang pewarisan dalam pemrograman Java kepada pembaca.

## **1.4. Metode Penulisan**

1. Studi Pustaka: Melakukan penelitian dan studi pustaka untuk memahami konsep dan teori pewarisan dalam pemrograman Java. Mengacu pada buku, jurnal, artikel, dan sumber-sumber tepercaya lainnya untuk mendapatkan landasan teoritis yang kuat.
2. Analisis Konsep: Menganalisis konsep dasar pewarisan dalam pemrograman Java, termasuk sintaksis, notasi, dan mekanisme yang digunakan. Menjelaskan secara rinci penggunaan keyword "extends" dan "super", serta konstruktor dalam pewarisan.
3. Studi Kasus: Menggunakan studi kasus kelas binatang dan turunannya untuk memberikan contoh implementasi pewarisan dalam Java. Menjelaskan bagaimana hierarki kelas dibentuk, bagaimana konstruktor dan metode didefinisikan, serta bagaimana pewarisan dan overriding dilakukan.
4. Penjelasan Praktis: Memberikan penjelasan praktis dan contoh implementasi untuk memperjelas konsep-konsep yang telah dijelaskan sebelumnya. Menggunakan kode program dan ilustrasi untuk membantu pembaca memahami penggunaan pewarisan dalam pemrograman Java.
5. Analisis dan Diskusi: Menganalisis kelebihan dan kelemahan dari pewarisan dalam pemrograman Java. Membahas aksesibilitas atribut dan metode dalam pewarisan, serta implikasi penggunaan pewarisan dalam pengembangan perangkat lunak.
6. Kesimpulan: Menyimpulkan temuan-temuan yang telah dijelaskan dalam makalah. Merangkum pentingnya pemahaman dan penerapan pewarisan dalam pemrograman Java, serta memberikan rekomendasi dan saran untuk penerapan yang efektif.

Dalam metode penulisan tersebut, penting untuk menyajikan informasi dengan jelas, logis, dan terstruktur. Menggunakan contoh, ilustrasi, dan kode program yang relevan juga dapat membantu pembaca dalam memahami konsep-konsep yang dijelaskan. Selain itu, mengacu pada sumber-sumber tepercaya dan menyertakan daftar pustaka yang lengkap adalah penting untuk menjaga kualitas dan keandalan makalah.

# BAB II

# KONSEP DASAR PEWARISAN

## **2.1. Pengertian Pewarisan dalam Pemrograman Berorientasi Objek (PBO)**

Pewarisan dalam pemrograman berorientasi objek (PBO) adalah konsep di mana sebuah kelas dapat mewarisi sifat dan perilaku dari kelas lain yang lebih umum atau induk. Dalam konteks pewarisan, kelas yang mewarisi disebut sebagai kelas turunan (subclass) atau kelas anak, sedangkan kelas yang memberikan warisan disebut sebagai kelas induk (superclass) atau kelas orangtua.

Dengan menggunakan pewarisan, kelas turunan dapat mengakses atribut dan metode dari kelas induk. Hal ini memungkinkan pembentukan hierarki kelas yang terstruktur dan memungkinkan penggunaan kembali kode program. Pewarisan memungkinkan kelas turunan untuk memiliki sifat dan perilaku yang sama dengan kelas induk, serta memperluas fungsionalitasnya dengan menambahkan atribut dan metode khusus.

Pewarisan juga mendukung konsep polimorfisme, di mana objek kelas turunan dapat dianggap sebagai objek kelas induk. Hal ini memungkinkan penggunaan pola desain seperti penggantian objek (object substitution) dan polimorfisme parameter (parameter polymorphism), yang meningkatkan fleksibilitas dan modularitas dalam pengembangan perangkat lunak.

Dengan pewarisan, pemrogram dapat mengorganisir kode secara hierarkis, meningkatkan efisiensi dan pemeliharaan kode, serta mengurangi duplikasi kode. Penggunaan pewarisan dalam pemrograman berorientasi objek (PBO) membantu dalam pengembangan aplikasi yang lebih terstruktur, modular, dan mudah dikelola.

## **2.2. Tujuan dan Manfaat Pewarisan dalam PBO**

1. Tujuan Pewarisan: Tujuan dari penggunaan pewarisan dalam PBO adalah:
   * Memahami konsep dasar pewarisan dalam PBO: Tujuan ini adalah memberikan pemahaman yang kuat tentang konsep dasar pewarisan, termasuk hubungan kelas turunan dengan kelas induk, dan bagaimana pewarisan membentuk hierarki kelas.
   * Mengjelaskan tujuan penggunaan pewarisan dalam PBO: Tujuan ini adalah menjelaskan mengapa pewarisan digunakan dalam pengembangan perangkat lunak, seperti memungkinkan penggunaan kembali kode, memperluas fungsionalitas kelas yang ada, dan membantu dalam pengaturan hierarki kelas yang terstruktur.
   * Menggambarkan pembentukan hierarki kelas melalui pewarisan: Tujuan ini adalah memberikan pemahaman tentang bagaimana kelas-kelas turunan dapat dibentuk melalui pewarisan dari kelas induk, membentuk struktur hierarkis yang terorganisir dalam pengembangan perangkat lunak.
2. Manfaat Pewarisan: Manfaat yang diperoleh dari penggunaan pewarisan dalam PBO adalah:
   * Mengurangi duplikasi kode: Dengan pewarisan, kita dapat menempatkan sifat dan perilaku umum dalam kelas induk, sehingga mengurangi duplikasi kode di dalam kelas turunan.
   * Meningkatkan efisiensi dan pemeliharaan kode: Dengan menggunakan pewarisan, kode program dapat dikembangkan dengan cara yang lebih efisien dan mudah dikelola. Jika ada perubahan yang perlu dilakukan, perubahan hanya perlu dilakukan pada kelas induk, dan perubahan tersebut akan diwariskan ke kelas-kelas turunan secara otomatis.
   * Meningkatkan fleksibilitas dan skalabilitas program: Dengan pewarisan, kita dapat memperluas fungsionalitas kelas yang ada melalui kelas turunan. Hal ini memberikan fleksibilitas dalam mengubah dan menyesuaikan perilaku kelas turunan tanpa mempengaruhi kelas induk atau kelas lainnya dalam hierarki.
   * Memungkinkan penggunaan kembali kode program: Dengan pewarisan, kode program yang sudah ada dalam kelas induk dapat digunakan kembali dalam kelas turunan, sehingga menghemat waktu dan upaya pengembangan.
   * Memperluas fungsionalitas kelas yang ada: Melalui pewarisan, kita dapat menambahkan atribut dan metode khusus dalam kelas turunan, sehingga memperluas fungsionalitas yang dimiliki oleh kelas induk.

Dengan memahami tujuan dan manfaat pewarisan dalam PBO, pemrogram dapat memanfaatkan konsep ini secara efektif dalam pengembangan perangkat lunak, meningkatkan kualitas, dan efisiensi.

## **2.3. Contoh Kasus Penggunaan Pewarisan dalam Java**

Contoh Implementasi:

Dalam sistem pengolahan data karyawan, kita akan menggunakan pewarisan untuk mengorganisir kelas-kelas yang terlibat, seperti karyawan tetap, karyawan kontrak, dan karyawan magang.

1. Pembentukan Hierarki Kelas:
   * Kita akan memiliki kelas induk bernama "Karyawan" yang berisi atribut dan metode umum yang dimiliki oleh semua jenis karyawan.
   * Selanjutnya, kita akan membuat kelas-kelas turunan seperti "KaryawanTetap", "KaryawanKontrak", dan "KaryawanMagang". Setiap kelas turunan ini akan mewarisi atribut dan metode dari kelas "Karyawan", serta memiliki atribut dan metode tambahan yang spesifik untuk jenis karyawan tersebut.
2. Penggunaan Kembali Kode:
   * Dalam kelas "KaryawanTetap", kita dapat menggunakan atribut dan metode dari kelas "Karyawan" seperti nama dan alamat.
   * Hal yang sama berlaku untuk kelas-kelas "KaryawanKontrak" dan "KaryawanMagang" dalam menggunakan atribut dan metode yang diwarisi dari kelas "Karyawan".
3. Memperluas Fungsionalitas:
   * Dalam kelas "KaryawanTetap", kita dapat menambahkan metode tambahan seperti menghitung total pendapatan yang mencakup gaji, tunjangan, dan bonus.
   * Dalam kelas "KaryawanKontrak", kita dapat menambahkan metode untuk menghitung total pendapatan berdasarkan durasi kontrak dan honorarium.
   * Dalam kelas "KaryawanMagang", kita dapat menambahkan metode khusus untuk melacak perkembangan mahasiswa selama masa magang.

Dengan menggunakan pewarisan dalam contoh ini, kita dapat membangun hierarki kelas yang terstruktur dan mengelompokkan atribut serta metode yang relevan ke dalam kelas-kelas yang sesuai. Hal ini membantu mengurangi duplikasi kode, meningkatkan efisiensi dan pemeliharaan kode, serta memungkinkan penambahan fungsionalitas yang spesifik untuk setiap jenis karyawan.

Implementasi yang sebenarnya akan melibatkan lebih banyak atribut dan metode yang spesifik, serta logika bisnis yang lebih kompleks sesuai dengan kebutuhan sistem pengolahan data karyawan yang sebenarnya.

Contoh implementasi lebih lanjut dari contoh kasus penggunaan pewarisan dalam Java ini dapat meliputi:

1. Menambahkan metode getGaji() di kelas "KaryawanTetap" yang menghitung total gaji karyawan tetap berdasarkan gaji pokok, tunjangan, dan bonus.

public class KaryawanTetap extends Karyawan {

private double tunjangan;

private double bonus;

public KaryawanTetap(String nama, String alamat, double gajiPokok, double tunjangan, double bonus) {

super(nama, alamat, gajiPokok);

this.tunjangan = tunjangan;

this.bonus = bonus;

}

public double getGaji() {

return super.getGaji() + tunjangan + bonus;

}

}

1. Menambahkan metode getGaji() di kelas "KaryawanKontrak" yang menghitung total gaji karyawan kontrak berdasarkan honorarium.

public class KaryawanKontrak extends Karyawan {

private double honorarium;

public KaryawanKontrak(String nama, String alamat, double honorarium) {

super(nama, alamat);

this.honorarium = honorarium;

}

public double getGaji() {

return honorarium;

}

}

1. Menambahkan metode untuk melacak perkembangan mahasiswa magang di kelas "KaryawanMagang".

public class KaryawanMagang extends Karyawan {

private String universitas;

private int jangkaWaktuMagang;

public KaryawanMagang(String nama, String alamat, String universitas, int jangkaWaktuMagang) {

super(nama, alamat);

this.universitas = universitas;

this.jangkaWaktuMagang = jangkaWaktuMagang;

}

public void cekPerkembangan() {

// Implementasi logika untuk melacak perkembangan mahasiswa magang

// ...

}

}

Dalam contoh ini, pewarisan memungkinkan kita untuk mengelompokkan atribut dan metode yang relevan ke dalam kelas-kelas yang sesuai, serta menambahkan logika bisnis spesifik sesuai dengan jenis karyawan yang ada. Penggunaan pewarisan dalam implementasi ini membantu dalam mengorganisir kode, mengurangi duplikasi, serta mempermudah pemeliharaan dan perluasan fungsionalitas sistem pengolahan data karyawan.

Dengan demikian, contoh kasus ini memberikan ilustrasi tentang bagaimana pewarisan dapat diterapkan dalam pemrograman berorientasi objek (PBO) menggunakan bahasa Java.

# BAB III

# SYNTAX DAN NOTASI PEWARISAN

## **3.1. Keyword "extends" dan "super"**

1. "extends":

Keyword "extends" digunakan dalam deklarasi kelas turunan untuk menunjukkan bahwa suatu kelas adalah turunan dari kelas lain (kelas induk atau superclass). Dengan menggunakan "extends", kelas turunan akan mewarisi semua atribut dan metode yang ada pada kelas induk. Ini memungkinkan kelas turunan untuk memiliki fitur dan perilaku yang sama dengan kelas induk, serta menambahkan fitur-fitur yang spesifik.

Contoh penggunaan "extends":

class Karyawan {

// atribut dan metode kelas Karyawan

}

class KaryawanTetap extends Karyawan {

// atribut dan metode kelas KaryawanTetap

}

Dalam contoh di atas, kelas "KaryawanTetap" menggunakan "extends" untuk mewarisi atribut dan metode dari kelas "Karyawan". Sebagai hasilnya, kelas "KaryawanTetap" akan memiliki atribut dan metode yang sama dengan kelas "Karyawan" serta atribut dan metode tambahan yang dimilikinya sendiri.

2. "super":

Keyword "super" digunakan dalam kelas turunan untuk merujuk ke atribut dan metode dari kelas induk (superclass). Dengan menggunakan "super", kita dapat mengakses dan menggunakan atribut dan metode dari kelas induk dalam implementasi kelas turunan. Ini berguna ketika kita ingin memodifikasi atau menambahkan perilaku khusus pada kelas turunan sambil mempertahankan perilaku yang ada dari kelas induk.

Contoh penggunaan "super":

class Karyawan {

String nama;

void displayInfo() {

System.out.println("Nama: " + nama);

}

}

class KaryawanTetap extends Karyawan {

double gaji;

void displayInfo() {

super.displayInfo(); // memanggil metode displayInfo() dari kelas induk

System.out.println("Gaji: " + gaji);

}

}

Dalam contoh di atas, kelas "KaryawanTetap" menggunakan "super" untuk memanggil metode "displayInfo()" dari kelas induk ("Karyawan"). Dengan demikian, metode "displayInfo()" di kelas "KaryawanTetap" akan menampilkan informasi dari kelas induk serta informasi tambahan yang dimiliki oleh kelas "KaryawanTetap".

Dengan pemahaman tentang penggunaan "extends" dan "super", Anda dapat memanfaatkan konsep pewarisan dengan lebih baik dalam pemrograman berorientasi objek (PBO) dengan bahasa Java.

## **3.2. Pembuatan dan Penggunaan Kelas Turunan (Subclass)**

Pembuatan dan penggunaan kelas turunan (subclass) adalah salah satu poin yang terdapat dalam daftar isi yang telah disebutkan sebelumnya. Berikut adalah penjelasan mengenai pembuatan dan penggunaan kelas turunan dalam konteks pewarisan dalam bahasa Java:

Pembuatan dan Penggunaan Kelas Turunan (Subclass):

1. Pembuatan Kelas Turunan:
   * Dalam konsep pewarisan, kelas turunan dibuat dengan menggunakan keyword "extends" setelah deklarasi kelas turunan.
   * Kelas turunan mewarisi atribut dan metode dari kelas induk (superclass) yang ditentukan setelah keyword "extends".
   * Kelas turunan memiliki semua atribut dan metode yang dimiliki oleh kelas induk, serta dapat menambahkan atribut dan metode tambahan yang spesifik.
2. Penggunaan Kelas Turunan:
   * Kelas turunan dapat digunakan untuk membuat objek seperti halnya kelas biasa.
   * Objek yang dibuat dari kelas turunan memiliki akses ke atribut dan metode yang diwarisi dari kelas induk, serta atribut dan metode tambahan yang dimiliki oleh kelas turunan.
   * Dalam penggunaan kelas turunan, kita dapat memanipulasi dan mengakses data serta perilaku objek sesuai dengan kelas turunan tersebut.

Contoh Penggunaan Kelas Turunan:

Misalkan kita memiliki kelas induk "Karyawan" dan kelas turunan "KaryawanTetap" yang mewarisi dari kelas "Karyawan".

class Karyawan {

String nama;

void displayInfo() {

System.out.println("Nama: " + nama);

}

}

class KaryawanTetap extends Karyawan {

double gaji;

void displayInfoGaji() {

System.out.println("Gaji: " + gaji);

}

}

Dalam contoh di atas, kelas "KaryawanTetap" adalah kelas turunan yang mewarisi atribut dan metode dari kelas "Karyawan". Kelas "KaryawanTetap" memiliki atribut "gaji" yang merupakan atribut tambahan yang khusus untuk kelas tersebut. Kelas "KaryawanTetap" juga memiliki metode "displayInfoGaji()" yang merupakan metode tambahan yang spesifik untuk kelas tersebut.

Kita dapat membuat objek dari kelas "KaryawanTetap" dan menggunakannya sebagai berikut:

public class Main {

public static void main(String[] args) {

KaryawanTetap karyawan = new KaryawanTetap();

karyawan.nama = "John Doe";

karyawan.gaji = 5000000;

karyawan.displayInfo(); // Memanggil metode displayInfo() dari kelas Karyawan

karyawan.displayInfoGaji(); // Memanggil metode displayInfoGaji() dari kelas KaryawanTetap

}

}

Dalam contoh di atas, kita membuat objek "karyawan" dari kelas "KaryawanTetap" dan mengakses atribut "nama" yang diwarisi dari kelas "Karyawan". Kita juga memanggil metode "displayInfo()" yang diwarisi dari kelas "Karyawan" dan metode "displayInfoGaji()" yang spesifik untuk kelas "KaryawanTetap".

Dalam contoh di atas, pemanggilan metode "displayInfo()" akan menampilkan informasi "Nama: John Doe" yang berasal dari kelas "Karyawan". Sedangkan pemanggilan metode "displayInfoGaji()" akan menampilkan informasi "Gaji: 5000000" yang merupakan atribut tambahan dari kelas "KaryawanTetap".

Penggunaan kelas turunan memungkinkan kita untuk mengelompokkan atribut dan metode yang relevan ke dalam kelas-kelas yang sesuai, serta menambahkan fitur-fitur tambahan yang spesifik. Dengan adanya pewarisan, kita dapat menghindari duplikasi kode yang tidak perlu dan mempermudah pemeliharaan serta perluasan fungsionalitas sistem.

Selain itu, pewarisan juga memungkinkan kita untuk menggunakan polimorfisme, di mana objek kelas turunan dapat digunakan sebagai objek kelas induk. Misalnya, objek "karyawan" dalam contoh di atas dapat dianggap sebagai objek "Karyawan" atau objek "KaryawanTetap", tergantung pada konteks penggunaannya.

Dalam pemrograman berorientasi objek, pembuatan dan penggunaan kelas turunan sangat penting untuk memanfaatkan konsep pewarisan. Hal ini memungkinkan kita untuk memodelkan hubungan "is-a" antara kelas-kelas dan memperluas fungsionalitas sistem dengan cara yang lebih terorganisir dan efisien.

## **3.3. Pewarisan Multiple dalam Java**

Pewarisan multiple (multiple inheritance) adalah salah satu topik yang terdapat dalam daftar isi yang telah disebutkan sebelumnya. Namun, perlu diperhatikan bahwa Java tidak mendukung pewarisan multiple langsung, yang berarti kelas tidak dapat mewarisi dari beberapa kelas secara langsung.

Java menerapkan model pewarisan tunggal (single inheritance), di mana sebuah kelas dapat mewarisi atribut dan metode dari satu kelas induk saja. Namun, ada cara lain untuk mencapai efek yang mirip dengan pewarisan multiple dalam Java menggunakan konsep "interface" dan "implementasi antarmuka" (interface implementation).

Berikut adalah penjelasan singkat tentang implementasi antarmuka dalam Java:

1. Antarmuka (Interface):
   * Antarmuka adalah kumpulan deklarasi metode yang tidak memiliki implementasi.
   * Antarmuka dapat didefinisikan dengan menggunakan keyword "interface" di Java.
   * Antarmuka mendefinisikan perilaku yang harus diimplementasikan oleh kelas-kelas yang mengimplementasikannya.
   * Sebuah kelas dapat mengimplementasikan satu atau lebih antarmuka.
2. Implementasi Antarmuka:
   * Ketika sebuah kelas mengimplementasikan sebuah antarmuka, kelas tersebut harus memberikan implementasi untuk semua metode yang dideklarasikan dalam antarmuka tersebut.
   * Implementasi metode antarmuka dilakukan dengan menggunakan keyword "implements" setelah deklarasi kelas.
   * Kelas yang mengimplementasikan antarmuka dapat memiliki perilaku khususnya sendiri, selain perilaku yang ditentukan oleh antarmuka.

Contoh Implementasi Antarmuka dalam Java:

interface Interface1 {

void method1();

}

interface Interface2 {

void method2();

}

class MyClass implements Interface1, Interface2 {

public void method1() {

// Implementasi metode dari Interface1

}

public void method2() {

// Implementasi metode dari Interface2

}

// Metode kelas MyClass

}

Dalam contoh di atas, kita memiliki dua antarmuka, yaitu "Interface1" dan "Interface2". Kemudian kita memiliki kelas "MyClass" yang mengimplementasikan kedua antarmuka tersebut menggunakan keyword "implements". Kelas "MyClass" harus memberikan implementasi untuk metode "method1()" dari "Interface1" dan metode "method2()" dari "Interface2".

Dengan menggunakan implementasi antarmuka seperti ini, kita dapat mencapai efek yang mirip dengan pewarisan multiple. Kelas "MyClass" memiliki perilaku yang didefinisikan oleh kedua antarmuka yang diimplementasikan, serta perilaku khususnya sendiri yang didefinisikan dalam kelas tersebut.

Dengan demikian, meskipun Java tidak mendukung pewarisan multiple secara langsung, kita dapat menggunakan konsep implementasi antarmuka untuk mencapai tujuan yang serupa.

# BAB IV

# KONSTRUKTOR DALAM PEWARISAN

## **4.1. Konstruktor dalam Kelas Induk (Superclass)**

Konstruktor dalam kelas induk (superclass) adalah salah satu poin yang terdapat dalam daftar isi yang telah disebutkan sebelumnya. Berikut adalah penjelasan mengenai konstruktor dalam kelas induk (superclass) dalam konteks pewarisan dalam bahasa Java:

Konstruktor dalam Kelas Induk (Superclass):

1. Konstruktor Kelas Induk:
   * Konstruktor adalah sebuah metode khusus yang digunakan untuk menginisialisasi objek dari suatu kelas.
   * Kelas induk (superclass) juga dapat memiliki konstruktor seperti kelas biasa.
   * Konstruktor dalam kelas induk dipanggil saat objek kelas turunan (subclass) dibuat.
2. Penggunaan Konstruktor Kelas Induk:
   * Ketika kelas turunan dibuat, konstruktor kelas induk dipanggil terlebih dahulu untuk menginisialisasi atribut dari kelas induk.
   * Konstruktor kelas turunan dapat menggunakan kata kunci "super" untuk memanggil konstruktor dari kelas induk.
   * Jika tidak ada pemanggilan konstruktor kelas induk secara eksplisit, maka konstruktor default dari kelas induk akan dipanggil secara implisit.

Contoh Penggunaan Konstruktor dalam Kelas Induk:

Misalkan kita memiliki kelas induk "Kendaraan" yang memiliki konstruktor dan kelas turunan "Mobil" yang mewarisi dari kelas "Kendaraan".

class Kendaraan {

String merek;

Kendaraan(String merek) {

this.merek = merek;

System.out.println("Konstruktor Kendaraan dipanggil.");

}

}

class Mobil extends Kendaraan {

String warna;

Mobil(String merek, String warna) {

super(merek); // Memanggil konstruktor Kendaraan dengan menggunakan "super"

this.warna = warna;

System.out.println("Konstruktor Mobil dipanggil.");

}

}

Dalam contoh di atas, kelas "Kendaraan" memiliki konstruktor yang menginisialisasi atribut "merek" dengan menggunakan parameter. Kelas "Mobil" adalah kelas turunan dari "Kendaraan" dan memiliki konstruktor yang menginisialisasi atribut "warna" dengan menggunakan parameter.

Pada konstruktor kelas "Mobil", kita menggunakan kata kunci "super" di baris **super(merek);** untuk memanggil konstruktor kelas induk "Kendaraan" dan mengirimkan argumen "merek". Dengan ini, konstruktor kelas "Mobil" akan memanggil konstruktor kelas induk terlebih dahulu sebelum melanjutkan dengan inisialisasi atribut "warna" dan pesan yang dicetak.

Ketika objek "Mobil" dibuat, misalnya **Mobil mobil = new Mobil("Honda", "Merah");**, konstruktor kelas "Kendaraan" akan dipanggil terlebih dahulu dan mencetak pesan "Konstruktor Kendaraan dipanggil." diikuti oleh konstruktor kelas "Mobil" yang mencetak pesan "Konstruktor Mobil dipanggil.".

Dengan menggunakan konstruktor dalam kelas induk, kita dapat melakukan inisialisasi atribut pada kelas induk sebelum melanjutkan dengan inisialisasi atribut kelas turunan. Hal ini memungkinkan kita untuk mengatur keadaan awal objek dengan benar dan memastikan bahwa semua atribut yang diperlukan telah diinisialisasi.

Selain itu, penggunaan kata kunci "super" untuk memanggil konstruktor kelas induk juga memberikan fleksibilitas dalam hal penanganan penurunan (inheritance) yang lebih kompleks. Kita dapat memanggil konstruktor kelas induk dengan berbagai argumen yang sesuai dengan kebutuhan aplikasi kita.

Penting untuk dicatat bahwa jika tidak ada pemanggilan konstruktor kelas induk secara eksplisit menggunakan "super", maka secara implisit akan dipanggil konstruktor default dari kelas induk. Namun, jika kelas induk tidak memiliki konstruktor default dan kita tidak memanggil konstruktor kelas induk dengan argumen yang sesuai, maka akan terjadi kesalahan kompilasi.

Dengan demikian, penggunaan konstruktor dalam kelas induk (superclass) memungkinkan kita untuk melakukan inisialisasi yang tepat pada kelas induk sebelum melanjutkan dengan inisialisasi kelas turunan. Hal ini membantu dalam membangun hierarki kelas yang lebih kompleks dan memastikan bahwa objek yang dibuat memiliki keadaan awal yang konsisten.

## **4.2. Konstruktor dalam Kelas Turunan (Subclass)**

Konstruktor dalam kelas turunan (subclass) adalah salah satu poin yang terdapat dalam daftar isi yang telah disebutkan sebelumnya. Berikut adalah penjelasan mengenai konstruktor dalam kelas turunan (subclass) dalam konteks pewarisan dalam bahasa Java:

Konstruktor dalam Kelas Turunan (Subclass):

1. Konstruktor Kelas Turunan:
   * Konstruktor adalah sebuah metode khusus yang digunakan untuk menginisialisasi objek dari suatu kelas.
   * Kelas turunan (subclass) dapat memiliki konstruktor sendiri yang dapat melakukan inisialisasi khusus untuk atribut atau perilaku tambahan yang dimilikinya.
   * Konstruktor dalam kelas turunan dapat memanggil konstruktor dari kelas induk (superclass) menggunakan kata kunci "super".
2. Pemanggilan Konstruktor Kelas Induk:
   * Dalam konstruktor kelas turunan, pemanggilan konstruktor kelas induk dilakukan dengan menggunakan kata kunci "super" di baris pertama konstruktor.
   * Pemanggilan konstruktor kelas induk harus dilakukan sebelum inisialisasi khusus kelas turunan.
   * Pemanggilan konstruktor kelas induk dapat menggunakan argumen yang sesuai dengan konstruktor yang didefinisikan dalam kelas induk.

Contoh Penggunaan Konstruktor dalam Kelas Turunan:

Misalkan kita memiliki kelas induk "Hewan" yang memiliki konstruktor dan kelas turunan "Kucing" yang mewarisi dari kelas "Hewan".

class Hewan {

String jenis;

Hewan(String jenis) {

this.jenis = jenis;

System.out.println("Konstruktor Hewan dipanggil.");

}

}

class Kucing extends Hewan {

String nama;

Kucing(String jenis, String nama) {

super(jenis); // Memanggil konstruktor Hewan dengan menggunakan "super"

this.nama = nama;

System.out.println("Konstruktor Kucing dipanggil.");

}

}

Dalam contoh di atas, kelas "Hewan" memiliki konstruktor yang menginisialisasi atribut "jenis" dengan menggunakan parameter. Kelas "Kucing" adalah kelas turunan dari "Hewan" dan memiliki konstruktor yang menginisialisasi atribut "nama" dengan menggunakan parameter.

Pada konstruktor kelas "Kucing", kita menggunakan kata kunci "super" di baris **super(jenis);** untuk memanggil konstruktor kelas induk "Hewan" dan mengirimkan argumen "jenis". Dengan ini, konstruktor kelas "Kucing" akan memanggil konstruktor kelas induk terlebih dahulu sebelum melanjutkan dengan inisialisasi atribut "nama" dan pesan yang dicetak.

Ketika objek "Kucing" dibuat, misalnya **Kucing kucing = new Kucing("Anggora", "Tom");**, konstruktor kelas "Hewan" akan dipanggil terlebih dahulu dan mencetak pesan "Konstruktor Hewan dipanggil." diikuti oleh konstruktor kelas "Kucing" yang mencetak pesan "Konstruktor Kucing dipanggil.".

Dengan menggunakan konstruktor dalam kelas turunan, kita dapat melakukan inisialisasi khusus pada kelas turunan sesuai dengan kebutuhan aplikasi kita. Konstruktor kelas turunan memungkinkan kita untuk menambahkan logika inisialisasi tambahan atau mengatur keadaan awal kelas turunan dengan cara yang sesuai.

## **4.3. Memanggil Konstruktor Kelas Induk dari Kelas Turunan**

Memanggil Konstruktor Kelas Induk dari Kelas Turunan adalah salah satu poin yang terdapat dalam daftar isi yang telah disebutkan sebelumnya. Berikut adalah penjelasan mengenai cara memanggil konstruktor kelas induk (superclass) dari kelas turunan (subclass) dalam konteks pewarisan dalam bahasa Java:

Memanggil Konstruktor Kelas Induk dari Kelas Turunan:

1. Penggunaan Kata Kunci "super":
   * Dalam kelas turunan, penggunaan kata kunci "super" digunakan untuk memanggil konstruktor dari kelas induk.
   * Pemanggilan konstruktor kelas induk dilakukan dalam konstruktor kelas turunan.
   * Pemanggilan konstruktor kelas induk harus dilakukan pada baris pertama konstruktor kelas turunan.
2. Pemilihan Konstruktor Kelas Induk yang Tepat:
   * Kelas induk (superclass) dapat memiliki beberapa konstruktor dengan parameter yang berbeda.
   * Dalam kelas turunan, kita harus memilih konstruktor kelas induk yang sesuai dengan kebutuhan dan parameter konstruktor kelas turunan.
   * Pemilihan konstruktor dilakukan dengan menggunakan kata kunci "super" diikuti oleh argumen yang sesuai.

Contoh Penggunaan Memanggil Konstruktor Kelas Induk dari Kelas Turunan:

Misalkan kita memiliki kelas induk "Hewan" yang memiliki beberapa konstruktor, termasuk konstruktor dengan parameter dan konstruktor default, serta kelas turunan "Kucing" yang mewarisi dari kelas "Hewan".

class Hewan {

String jenis;

Hewan(String jenis) {

this.jenis = jenis;

System.out.println("Konstruktor Hewan dengan parameter dipanggil.");

}

Hewan() {

jenis = "Tidak Diketahui";

System.out.println("Konstruktor Hewan tanpa parameter dipanggil.");

}

}

class Kucing extends Hewan {

String nama;

Kucing(String jenis, String nama) {

super(jenis); // Memanggil konstruktor Hewan dengan parameter menggunakan "super"

this.nama = nama;

System.out.println("Konstruktor Kucing dipanggil.");

}

}

Dalam contoh di atas, kelas "Hewan" memiliki dua konstruktor: satu dengan parameter dan satu tanpa parameter. Kelas "Kucing" adalah kelas turunan dari "Hewan" dan memiliki konstruktor yang mengambil parameter "jenis" dan "nama".

Pada konstruktor kelas "Kucing", kita menggunakan kata kunci "super" di baris **super(jenis);** untuk memanggil konstruktor kelas induk "Hewan" yang mengambil parameter "jenis". Dengan menggunakan "super(jenis)", kita memilih konstruktor yang sesuai dengan parameter kelas turunan.

Ketika objek "Kucing" dibuat, misalnya **Kucing kucing = new Kucing("Anggora", "Tom");**, konstruktor kelas "Hewan" dengan parameter akan dipanggil terlebih dahulu dan mencetak pesan "Konstruktor Hewan dengan parameter dipanggil.", kemudian konstruktor kelas "Kucing" akan dipanggil dan mencetak pesan "Konstruktor Kucing dipanggil.".

Dengan menggunakan pemanggilan konstruktor kelas induk menggunakan kata kunci "super", kita dapat memastikan bahwa konstruktor kelas induk dijalankan terlebih dahulu sebelum melanjutkan dengan inisialisasi kelas turunan. Ini memungkinkan kita untuk melakukan inisialisasi yang diperlukan pada kelas induk sebelum melakukan inisialisasi khusus pada kelas turunan.

# BAB V

# CONTOH IMPLEMENTASI PEWARISAN

## **5.1. Studi Kasus: Kelas Binatang dan Kelas Turunannya**

Studi Kasus: Kelas Binatang dan Kelas Turunannya

Untuk memperjelas konsep pewarisan dalam Java, kita dapat menggunakan studi kasus tentang kelas "Binatang" dan kelas turunannya sebagai contoh. Berikut adalah penjelasan mengenai studi kasus ini berdasarkan daftar isi yang telah disebutkan sebelumnya:

1. Pembuatan Kelas Binatang:
   * Kita akan membuat kelas "Binatang" sebagai kelas induk atau superkelas.
   * Kelas "Binatang" akan memiliki atribut "nama" untuk menyimpan nama binatang.
2. Pembuatan Kelas Turunan: Kucing dan Anjing:
   * Kita akan membuat dua kelas turunan dari kelas "Binatang", yaitu kelas "Kucing" dan kelas "Anjing".
   * Kelas "Kucing" dan "Anjing" akan mewarisi atribut dan metode dari kelas "Binatang".
3. Penggunaan Kata Kunci "extends":
   * Untuk membuat kelas turunan, kita menggunakan kata kunci "extends" diikuti dengan nama kelas induk.
   * Misalnya, kelas "Kucing" akan dideklarasikan dengan class Kucing extends Binatang.
   * Ini menunjukkan bahwa kelas "Kucing" merupakan turunan dari kelas "Binatang".
4. Penambahan Atribut dan Metode Khusus:
   * Setiap kelas turunan dapat memiliki atribut dan metode khusus yang unik.
   * Misalnya, kelas "Kucing" dapat memiliki atribut "ras" dan metode "meow()", sedangkan kelas "Anjing" dapat memiliki atribut "berat" dan metode "gonggong()".

Dalam studi kasus ini, kita menggunakan kelas "Binatang" sebagai kelas induk yang memiliki atribut "nama". Kemudian, kita membuat dua kelas turunan, yaitu "Kucing" dan "Anjing", yang mewarisi atribut "nama" dari kelas "Binatang". Kedua kelas turunan ini juga memiliki atribut dan metode khusus mereka sendiri yang menambahkan perilaku spesifik.

Contoh Implementasi Kelas Binatang, Kucing, dan Anjing:

class Binatang {

String nama;

public Binatang(String nama) {

this.nama = nama;

}

public void makan() {

System.out.println(nama + " sedang makan.");

}

}

class Kucing extends Binatang {

String ras;

public Kucing(String nama, String ras) {

super(nama);

this.ras = ras;

}

public void meow() {

System.out.println(nama + " sedang meow.");

}

}

class Anjing extends Binatang {

double berat;

public Anjing(String nama, double berat) {

super(nama);

this.berat = berat;

}

public void gonggong() {

System.out.println(nama + " sedang gonggong.");

}

}

public class Main {

public static void main(String[] args) {

Kucing kucing = new Kucing("Tom", "Anggora");

kucing.makan();

kucing.meow();

Anjing anjing = new Anjing("Bruno", 15.5);

anjing.makan();

anjing.gonggong(); } }

Dalam contoh di atas, kita membuat objek "Kucing" dengan nama "Tom" dan ras "Anggora". Kemudian, kita memanggil metode "makan()" yang diwarisi dari kelas "Binatang" untuk memanggil perilaku umum binatang saat makan. Selain itu, kita juga memanggil metode khusus "meow()" yang hanya dimiliki oleh kelas "Kucing".

Kemudian, kita membuat objek "Anjing" dengan nama "Bruno" dan berat 15.5. Kita juga memanggil metode "makan()" yang diwarisi dari kelas "Binatang" dan metode khusus "gonggong()" yang hanya dimiliki oleh kelas "Anjing".

Dengan menggunakan pewarisan dalam studi kasus ini, kita dapat membagi perilaku umum dari kelas "Binatang" ke kelas turunannya, seperti "Kucing" dan "Anjing", dan juga menambahkan perilaku unik yang sesuai dengan kelas turunan tersebut.

Melalui contoh ini, kita dapat melihat bagaimana pewarisan memungkinkan kita untuk mengorganisir dan memodelkan hierarki kelas dalam pemrograman berorientasi objek.

## **5.2. Analisis Implementasi Pewarisan dalam Kasus Tersebut**

Analisis Implementasi Pewarisan dalam Kasus Tersebut:

Implementasi pewarisan dalam kasus studi Binatang, Kucing, dan Anjing yang telah dijelaskan sebelumnya memiliki beberapa poin penting yang dapat dianalisis. Berikut adalah analisis implementasi pewarisan berdasarkan daftar isi yang telah disebutkan sebelumnya:

1. Hierarki Kelas:
   * Dalam kasus ini, kita memiliki kelas induk "Binatang" dan dua kelas turunan, yaitu "Kucing" dan "Anjing".
   * Hierarki kelas ini menggambarkan hubungan "is-a" antara kelas-kelas tersebut.
   * "Kucing" dan "Anjing" adalah jenis binatang, sehingga mereka dapat dianggap sebagai turunan dari kelas "Binatang".
2. Pewarisan Atribut:
   * Kelas turunan, seperti "Kucing" dan "Anjing", mewarisi atribut "nama" dari kelas induk "Binatang".
   * Ini berarti objek yang dibuat dari kelas turunan juga memiliki atribut "nama" yang sama seperti yang ada di kelas induk.
3. Pewarisan Metode:
   * Kelas turunan mewarisi metode yang telah didefinisikan dalam kelas induk.
   * Dalam kasus ini, kelas "Kucing" dan "Anjing" mewarisi metode "makan()" dari kelas "Binatang".
   * Kedua kelas turunan juga memiliki metode khusus mereka sendiri, yaitu "meow()" untuk kelas "Kucing" dan "gonggong()" untuk kelas "Anjing".
4. Penggunaan Konstruktor Kelas Induk:
   * Dalam konstruktor kelas turunan, kita menggunakan kata kunci "super" untuk memanggil konstruktor kelas induk.
   * Dalam contoh ini, konstruktor kelas "Kucing" dan "Anjing" memanggil konstruktor kelas "Binatang" menggunakan "super(nama)".
   * Ini memastikan bahwa konstruktor kelas induk dijalankan terlebih dahulu untuk menginisialisasi atribut "nama" sebelum melakukan inisialisasi kelas turunan.

Dengan menggunakan implementasi pewarisan dalam kasus ini, kita dapat memanfaatkan konsep pemrograman berorientasi objek untuk memodelkan hubungan hierarkis antara kelas-kelas tersebut. Pewarisan memungkinkan kita untuk membagikan atribut dan metode umum dari kelas induk ke kelas turunan, sambil memungkinkan kelas turunan untuk memiliki atribut dan metode khusus mereka sendiri.

Melalui analisis implementasi pewarisan dalam kasus ini, kita dapat melihat bagaimana pewarisan dapat membantu dalam mengorganisir kode, meminimalkan duplikasi, dan memungkinkan penggunaan kembali yang efisien.

# BAB VI

# KESIMPULAN

Kesimpulan dari makalah tentang pewarisan dalam pemrograman berorientasi objek (PBO) dengan bahasa Java dapat disimpulkan sebagai berikut:

1. Pewarisan merupakan konsep dalam PBO yang memungkinkan kelas untuk mewarisi atribut dan metode dari kelas lain yang menjadi kelas induk atau superkelas.
2. Tujuan dan manfaat pewarisan dalam PBO antara lain untuk mengorganisir hierarki kelas, menghindari duplikasi kode, memudahkan penggunaan kembali kode, dan meningkatkan fleksibilitas dan modularitas.
3. Ruang lingkup makalah meliputi pengertian pewarisan, penggunaan kata kunci "extends" dan "super", pembuatan dan penggunaan kelas turunan, pewarisan multiple, dan pengelolaan konstruktor dalam kelas induk dan turunan.
4. Metode penulisan yang dapat digunakan dalam makalah ini adalah metode deskriptif yang menjelaskan konsep, sintaksis, dan contoh implementasi pewarisan dalam Java.
5. Studi kasus tentang kelas Binatang dan kelas turunannya (Kucing dan Anjing) digunakan untuk menjelaskan konsep pewarisan dengan contoh nyata.
6. Analisis implementasi pewarisan dalam studi kasus tersebut mencakup hierarki kelas, pewarisan atribut dan metode, penggunaan konstruktor kelas induk, dan manfaat yang diperoleh dari pewarisan.

Makalah ini memberikan pemahaman yang mendalam tentang konsep pewarisan dalam PBO dengan menggunakan bahasa Java. Dengan memahami pewarisan, para pengembang perangkat lunak dapat memanfaatkannya untuk mengorganisir kode dengan lebih baik, meningkatkan efisiensi dan fleksibilitas, serta mempercepat pengembangan aplikasi.

DAFTAR PUSTAKA

* <https://www.petanikode.com/java-oop-inheritance/>
* <https://www.duniailkom.com/tutorial-oop-java-cara-membuat-pewarisan-class-inheritance/>
* <https://blog.rosihanari.net/konsep-pewarisan-inheritance-di-java/>
* <https://www.ardhi.web.id/2019/03/contoh-inheritance-pewarisan-di-java.html>
* <https://nextgen.co.id/inheritance-pada-java/#:~:text=Untuk%20melakukan%20inheritance%20dari%20sebuah%20class%2C%20gunakan%20keyword%20extends.&text=Pada%20contoh%20di%20atas%20kita%20menggunakan%20modifier%20protected%20dalam%20class%20Kendaraan>.
* <https://sis.binus.ac.id/2017/09/07/inheritance/>
* <https://triadinug89.wordpress.com/2010/05/16/inheritance-dalam-java/>
* <https://www.codeternity.com/id-id/article/java/java-inheritance>
* <https://runestone.academy/ns/books/published/csawesome/Unit9-Inheritance/topic-9-2-constructors.html>
* <https://www.geeksforgeeks.org/difference-between-super-and-super-in-java-with-examples/>