INHERITANCE

1. Kompetensi

Secara keseluruhan, latihan praktikum ini akan membekali mahasiswa dengan kompetensi yang dibutuhkan untuk mengembangkan program Java yang lebih terstruktur, mudah dikelola, dan dapat digunakan kembali, serta mempersiapkan mereka untuk menghadapi tantangan pemrograman yang lebih kompleks di masa depan. Setelah menempuh materi percobaan ini, mahasiswa mampu:

1. Memahami konsep dasar inheritance, termasuk bagaimana subclass mewarisi atribut dan metode dari superclass, serta bagaimana hubungan "is-a" diterapkan dalam Pemrograman Berorientasi Objek (PBO)
2. Mengimplementasikan inheritance dalam program Java, termasuk mendeklarasikan subclass, menggunakan keyword `extends`, dan memanggil konstruktor superclass menggunakan `super()`
3. Memahami konsep single dan multilevel inheritance, di mana sebuah subclass dapat mewarisi dari subclass lain, dan membentuk rantai pewarisan.
4. Memahami dan menerapkan method overriding, yaitu kemampuan untuk mengganti implementasi metode yang diwarisi dari superclass di subclass
5. Menggunakan keyword `super` untuk mengakses anggota (atribut dan metode) dari superclass, terutama saat terjadi konflik nama atau saat ingin memanggil metode dari superclass yang telah di-override

2. Pendahuluan

2.1 Konsep dasar

Inheritance atau pewarisan adalah salah satu konsep fundamental dalam pemrograman berorientasi objek (OOP). Inheritance memungkinkan kita untuk membuat kelas baru (subclass atau child class) berdasarkan kelas yang sudah ada (superclass atau parent class). Subclass mewarisi semua atribut dan metode dari superclass, sehingga kita dapat menggunakan kembali kode yang sudah ada dan menghindari duplikasi

Beberapa terminologi umum yang dikenal dalam Inheritance atau pewarisan adalah sebagai berikut;

1. **Superclass (Parent Class)**
   - **Definisi:** Superclass, juga dikenal sebagai parent class atau base class, adalah kelas yang menjadi dasar atau template untuk kelas-kelas lain. Ini adalah kelas yang sifat-sifatnya akan diwariskan kepada kelas-kelas turunannya.
   - **Analogi:** Anggaplah superclass sebagai cetakan kue. Cetakan ini memiliki bentuk dasar yang akan menentukan bentuk kue yang dihasilkan.
2. **Subclass (Child Class)**
   - **Definisi**: Subclass, juga dikenal sebagai child class atau derived class, adalah kelas yang dibuat berdasarkan superclass. Subclass mewarisi semua atribut (variabel) dan metode (fungsi) yang dimiliki oleh superclass-nya, dan juga dapat memiliki atribut dan metode tambahan yang spesifik untuk dirinya sendiri.
   - **Analogi**: Subclass seperti kue yang dibuat menggunakan cetakan kue (superclass). Kue ini memiliki bentuk dasar yang sama dengan cetakan, tetapi bisa memiliki dekorasi atau rasa tambahan yang membuatnya unik.
3. **Hubungan "is-a":** Subclass "is a" superclass. Contoh: Mobil "is a" Kendaraan.

- **Definisi**: Hubungan "is-a" menggambarkan hubungan antara superclass dan subclass. Subclass "is a" (adalah sebuah) jenis khusus dari superclass-nya.
- **Analogi**: Mobil "is a" Kendaraan. Ini berarti mobil adalah jenis kendaraan tertentu, dan memiliki semua karakteristik dasar sebuah kendaraan (roda, mesin, dll.), tetapi juga memiliki karakteristik tambahan yang spesifik untuk mobil (jumlah pintu, jenis transmisi, dll.).
- **Pentingnya**: Hubungan "is-a" adalah kunci untuk menentukan apakah inheritance tepat digunakan atau tidak. Jika hubungan ini tidak jelas atau dipaksakan, maka inheritance mungkin bukan solusi yang tepat.

Superclass, subclass, dan hubungan "is-a" adalah konsep dasar yang penting dalam inheritance. Dengan memahami konsep-konsep ini, Anda dapat memanfaatkan inheritance untuk membuat kode yang lebih terstruktur, mudah dikelola, dan dapat digunakan kembali

2.2 Manfaat Inheritance

Inheritance memungkinkan kita untuk memanfaatkan kode yang sudah ada, membangun struktur yang lebih teratur, dan menciptakan program yang lebih fleksibel. Pada dunia pemrograman, 'kerja cerdas' seringkali lebih baik daripada 'kerja keras'. Inheritance adalah salah satu kunci untuk bekerja cerdas dengan memungkinkan kita menggunakan kembali kode yang sudah ada. Berikut ini beberapa manfaat inheritance membantu kita menghindari duplikasi kode dan meningkatkan produktivitas.

- **Reusabilitas Kode:** Menghindari duplikasi kode dengan menggunakan kembali kode dari superclass di subclass.
- **Pemeliharaan Kode:** Membuat kode lebih mudah dikelola dan diubah karena perubahan pada superclass akan otomatis mempengaruhi subclass.
- **Polimorfisme:** Kemampuan objek dari berbagai kelas untuk merespons pesan yang sama dengan cara yang berbeda, dimungkinkan oleh inheritance.

2.2 Implementasi Inheritance di Java

Berikut ini beberapa keyword yang dapat digunakan dalam menimplementasikan inheritance

- **Keyword `extends`:** Digunakan untuk mendeklarasikan inheritance

```java
class Hewan {
    String nama;

    public Hewan(String nama) {
        this.nama = nama;
    }

    void bersuara() {
        System.out.println("Suara hewan...");
    }
}

class Kucing extends Hewan {
    public Kucing(String nama) {
        super(nama); // Memanggil konstruktor superclass
    }

    void mengeong() {
        System.out.println("Meong!");
    }
}
```

- **Keyword `@Override`:** Digunakan untuk menandai metode yang di-overrid atau untuk mengganti implementasi metode yang diwarisi dari superclass di subclass

```java
class Kucing extends Hewan {
    // ...

    @Override
    void bersuara() {
        System.out.println("Meong!");
    }
}
```

- **Keyword `super`:** Digunakan untuk memanggil konstruktor superclass dari subclass. Harus menjadi pernyataan pertama dalam konstruktor subclass. Karena konstruktor tidak dapat diwariskan, maka subclass harus memiliki konstruktor sendiri.

```java
class Kucing extends Hewan {
    public Kucing(String nama) {
        super(nama); // Memanggil konstruktor Hewan
    }
}
```

2.3 Access Control

- **`public`:** Dapat diakses dari mana saja.
- **`protected`:** Dapat diakses dalam package yang sama dan oleh subclass, bahkan jika subclass berada di package yang berbeda.
- **`private`:** Hanya dapat diakses dari dalam kelas itu sendiri.
- **(default/package-private)**: Dapat diakses dalam package yang sama.

2.4 Jenis-jenis Inheritance di Java

- **Single Inheritance:** Subclass hanya mewarisi dari satu superclass.
- **Multilevel Inheritance:** Subclass mewarisi dari subclass lain, membentuk rantai pewarisan.
- **Hierarchical Inheritance:** Beberapa subclass mewarisi dari satu superclass.
- **Hybrid Inheritance:** Kombinasi dari multiple inheritance dan hierarchical inheritance (tidak didukung langsung di Java, bisa diimplementasikan menggunakan interface).

2.5 Abstract Class dan Method

- **Abstract Class:** Kelas yang tidak bisa diinstansiasi secara langsung, hanya bisa dijadikan superclass.
- **Abstract Method:** Metode yang hanya memiliki deklarasi, tanpa implementasi. Harus diimplementasikan oleh subclass.

2.6 Interface

- **Definisi:** Sebuah kontrak yang mendefinisikan sekumpulan metode yang harus diimplementasikan oleh kelas yang mengimplementasikan interface tersebut.
- **Keyword `implements`:** Digunakan untuk mengimplementasikan interface.
- **Manfaat:** Mendukung polimorfisme dan memungkinkan sebuah kelas untuk mengimplementasikan beberapa interface.

2.7 Contoh Kasus dan Penerapan

- **Sistem Kendaraan:**
  - Superclass: Kendaraan (atribut: merk, tahunProduksi; metode: jalankan())
  - Subclass: Mobil, Motor, Truk (masing-masing memiliki atribut dan metode spesifik)
- **Sistem Informasi Akademik:**
  - Superclass: Orang (atribut: nama, alamat; metode: tampilkanInfo())
  - Subclass: Mahasiswa, Dosen, Staff (masing-masing memiliki atribut dan metode spesifik)

3. Percobaan

Berikut adalah 6 program Java untuk bahan **Latihan/Percobaan** sekaligus **Tugas Praktikum** OOP tentang inheritance, yang mencakup penerapan single inheritance, multilevel inheritance, override dan penggunaan keyword `super`:

3. 1 Percobaan 1: Dasar-Dasar Inheritance

- **Tujuan**: Memahami konsep dasar inheritance, bagaimana subclass mewarisi atribut dan metode dari superclass.
- Buat project baru dengan nama **HewanOOP1**

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Milo");
        kucing.bersuara(); // Output: Suara hewan... (diwarisi dari Hewan)
        kucing.mengeong();
    }
}
```

- Buat superclass **Hewan**

```
class Hewan {
    String nama;

    public Hewan(String nama) {
        this.nama = nama;
    }

    void bersuara() {
        System.out.println("Suara hewan...");
    }
}
```

- Buat subclass **Kucing**

```
class Kucing extends Hewan {
    public Kucing(String nama) {
        super(nama);
    }

    void mengeong() {
        System.out.println("Meong!");
    }
}
```

- **Tugas Praktikum**: Buatlah subclass lain, misalnya Anjing, yang mewarisi dari Hewan dan memiliki metode menggonggong().

3. 2 Percobaan 2: Menggunakan Method Overriding

- **Tujuan**: Memahami konsep method overriding, bagaimana subclass dapat mengganti implementasi metode yang diwarisi dari superclass.
- Perbarui project **HewanOOP1**

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Milo");
        kucing.bersuara(); // Output: Meong! (karena metode di-override)
        kucing.mengeong();
    }
}
```

- Superclass **Hewan**

```
class Hewan {
    String nama;

    public Hewan(String nama) {
        this.nama = nama;
    }

    void bersuara() {
        System.out.println("Suara hewan...");
    }
}
```

- Subclass **Kucing**

```
class Kucing extends Hewan {
    public Kucing(String nama) {
        super(nama);
    }

    @Override
    void bersuara() {
        System.out.println("Meong!"); // Meng-override metode bersuara() dari Hewan
    }

    void mengeong() {
        System.out.println("Meong!");
    }
}
```

- Subclass **Anjing**

```
class Anjing extends Hewan {
    public Kucing(String nama) {
        super(nama);
    }

    void menggonggong() {
        System.out.println("Guk...Guuk!");
    }
}
```

- **Tugas Praktikum**: Òverride metode bersuara() di kelas Anjing agar menghasilkan output "Guk!".

3. 3 Percobaan 3: Konstruktor dan `super()`

- **Tujuan**: Memahami bagaimana menggunakan `super()` untuk memanggil konstruktor superclass dari subclass.
- Buat project baru dengan nama **HewanOOP2**

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Milo", 2);
    }
}
```

- Superclass **Hewan**

```
class Hewan {
    String nama;
    int umur;

    public Hewan(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
        System.out.println("Konstruktor Hewan dipanggil");
    }
}
```

- Subclass **Kucing**

```
class Kucing extends Hewan {
    public Kucing(String nama, int umur) {
        super(nama, umur); // Memanggil konstruktor Hewan
        System.out.println("Konstruktor Kucing dipanggil");
    }
}
```

- **Tugas Praktikum**: Tambahkan atribut `ras` di kelas `Kucing` dan modifikasi konstruktornya agar menerima parameter `ras`. Pastikan konstruktor `Kucing` memanggil konstruktor `Hewan` dengan benar.

3. 4 Percobaan 4: Multilevel Inheritance

- **Tujuan**: Memahami konsep multilevel inheritance, bagaimana subclass dapat mewarisi dari subclass lain.
- Buat project baru dengan nama **HewanOOP3**

```java
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Milo", 2);
        kucing.berjalan();
        kucing.menyusui();
    }
}
```

- Superclass **Hewan**

```java
class Hewan {
    String nama;
    int umur;

    public Hewan(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
        System.out.println("Konstruktor Hewan dipanggil");
    }
}
```

- Subclass **Mamalia**

```java
class Mamalia extends Hewan {
    public Mamalia(String nama, int umur) {
        super(nama, umur);
    }

    void menyusui() {
        System.out.println("Menyusui anaknya...");
    }
}
```

- Sub-Subclass **Kucing**

```java
class Kucing extends Mamalia {
    public Kucing(String nama, int umur) {
        super(nama, umur);
    }
}
```

- **Tugas Praktikum**: Buatlah subclass lain, misalnya `Anjing`, yang mewarisi dari `Mamalia`. Tambahkan metode atau atribut yang sesuai untuk `Anjing`.

3. 5 Percobaan 5: Access Control dan `super`

- **Tujuan**: Memahami access control (`public`, `private`, `protected`) dan bagaimana `super` digunakan untuk mengakses anggota protected dari parent class.
- Buat project baru dengan nama **HewanOOP4**

```
public class Main {
    public static void main(String[] args) {
        Kucing kucing = new Kucing("Milo", 2, "Oranye");
        kucing.info();
        kucing.berjalan();
        System.out.println(kucing.nama); // Error, 'nama' adalah private
        System.out.println(kucing.umur); // Valid, 'umur' adalah protected dan bisa diakses oleh subclass
    }
}
```

- Superclass **Hewan**

```
class Hewan {
    private String nama;
    protected int umur;

    public Hewan(String nama, int umur) {
        this.nama = nama;
        this.umur = umur;
    }

    public void berjalan() {
        System.out.println(nama + " berjalan...");
    }

    public void info() {
        System.out.println("Nama: " + nama);
        System.out.println("Umur: " + umur);
    }
}
```

- Subclass **Kucing**

```
class Kucing extends Hewan {
    private String warnaBulu;

    public Kucing(String nama, int umur, String warnaBulu) {
        super(nama, umur);
        this.warnaBulu = warnaBulu;
    }

    public void info() {
        super.info(); // Menggunakan 'super' untuk mengakses metode dari parent class
        System.out.println("Warna bulu: " + warnaBulu);
    }
}
```

- **Tugas Praktikum**: Buatlah subclass lain, misalnya `Anjing`, yang mewarisi dari `Hewan`. Tambahkan metode atau atribut yang sesuai untuk `Anjing`, misalkan atribut `jenisBulu` yang bersifat `private`, dan metode `getJenisBulu()` yang bersifat `public` untuk mengakses atribut tersebut.

3. 6 Percobaan 6: Abstract Class

- **Tujuan**: Menerapkan konsep inheritance, `super`, access control, method overriding, dan abstract class dalam kasus yang lebih kompleks.
- Buat project baru dengan nama **KendaraanOOP**

```
public class Main {
    public static void main(String[] args) {
        Mobil mobil = new Mobil("Toyota", 2022, 4);
        Motor motor = new Motor("Honda", 2023, 150);

        mobil.info();
        mobil.jalankan();

        System.out.println("-----");

        motor.info();
        motor.jalankan();
    }
}
```

- Superclass **Kendaraan**

```
abstract class Kendaraan {
    String merk;
    int tahunProduksi;

    public Kendaraan(String merk, int tahunProduksi) {
        this.merk = merk;
        this.tahunProduksi = tahunProduksi;
    }

    abstract void jalankan();

    void info() {
        System.out.println("Merk: " + merk);
        System.out.println("Tahun produksi: " + tahunProduksi);
    }
}
```

- Subclass **Mobil**

```
class Mobil extends Kendaraan {
    int jumlahPintu;

    public Mobil(String merk, int tahunProduksi, int jumlahPintu) {
        super(merk, tahunProduksi);
        this.jumlahPintu = jumlahPintu;
    }

    @Override
    void jalankan() {
        System.out.println("Mobil " + merk + " berjalan...");
    }

    void info() {
        super.info();
        System.out.println("Jumlah pintu: " + jumlahPintu);
    }
}
```

- Subclass **Motor**

```java
class Motor extends Kendaraan {
    int kapasitasCC;

    public Motor(String merk, int tahunProduksi, int kapasitasCC) {
        super(merk, tahunProduksi);
        this.kapasitasCC = kapasitasCC;
    }

    @Override
    void jalankan() {
        System.out.println("Motor " + merk + " berjalan...");
    }

    void info() {
        super.info();
        System.out.println("Kapasitas CC: " + kapasitasCC);
    }
}
```

- **Tugas Praktikum**: Tambahkan subclass lain, misalnya `Truk`, yang mewarisi dari `Kendaraan`. Modifikasi kelas `Kendaraan` dan subclass lainnya sesuai kebutuhan.

4. Kesimpulan

Inheritance, konsep kunci dalam OOP, memungkinkan penggunaan kembali kode dari superclass, sehingga mengurangi duplikasi dan mempermudah pemeliharaan. Dengan `super`, subclass dapat mengakses anggota superclass dan memanggil konstruktornya.

Access control (`public`, `private`, `protected`) berperan penting dalam mengatur akses ke anggota kelas, menjaga enkapsulasi, dan mencegah perubahan yang tidak diinginkan. Inheritance sangat fleksibel, dapat diterapkan dalam berbagai kasus untuk memodelkan hubungan "is-a" antara kelas-kelas, menghasilkan kode yang lebih terstruktur, efisien, dan mudah dipahami.

5. Pertanyaan

Inheritance, konsep kunci dalam OOP, memungkinkan penggunaan kembali kode dari superclass, sehingga mengurangi duplikasi dan mempermudah pemeliharaan. Dengan `super`, subclass dapat mengakses anggota superclass dan memanggil konstruktornya.

1. Pada program latihan pertama, jelaskan bagaimana kelas `Kucing` memanfaatkan konsep inheritance dari kelas `Hewan`. Sebutkan atribut dan metode apa saja yang diwarisi oleh `Kucing`.
2. Pada program latihan kedua, apa yang dimaksud dengan method overriding dan bagaimana cara mengimplementasikannya? Berikan contoh dari program tersebut.
3. Pada program latihan ketiga, mengapa kita perlu menggunakan keyword `super` dalam konstruktor subclass? Jelaskan bagaimana `super` digunakan dalam program tersebut.
4. Pada program latihan keempat, jelaskan perbedaan antara single inheritance dan multilevel inheritance. Berikan contoh implementasi keduanya dari program tersebut.
5. Pada program latihan kelima, jelaskan peran access control (public, private, protected) dalam inheritance. Bagaimana cara `super` digunakan untuk mengakses anggota dari superclass?
6. Pada program latihan keenam, bagaimana konsep abstract class dan method overriding digunakan untuk menciptakan struktur yang fleksibel dalam memodelkan berbagai jenis kendaraan?