Name : Faatihurrizki Prasojo

NIM : 244107020142

Class : TI_1H

Laporan Tugas 3

Pemrograman Berorientasi Objek (PBO)

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.**

   - Kelas
     Kucing memanfaatkan konsep inheritance (pewarisan) dari kelas **Hewan** dengan menggunakan *keyword* **extends** (class Kucing extends Hewan)[1]. Hubungan ini adalah hubungan "**is-a**" (Kucing *adalah sebuah* Hewan).
   - Sebagai *subclass* , Kucing mewarisi semua atribut dan metode dari *superclass* Hewan.
   - **Atribut yang diwarisi:** String nama[6].
   - **Metode yang diwarisi:** void bersuara()[7].

2. **Pada program latihan kedua, apa yang dimaksud dengan method overriding dan bagaimana cara mengimplementasikannya? Berikan contoh dari program tersebut.**

   - **Method Overriding** adalah kemampuan untuk **mengganti implementasi metode** yang diwarisi dari *superclass* di dalam *subclass*.

   - **Cara Mengimplementasikan:**

     - Anda mendeklarasikan ulang metode yang sama (nama, parameter, dan tipe kembalian harus sama) di dalam *subclass*.

     - Disarankan menggunakan *keyword* **@Override** di atas deklarasi metode di *subclass* untuk menandai dan memastikan bahwa metode tersebut benar-benar menggantikan metode dari *superclass*.

   - **Contoh dari program (Percobaan 2, Subclass Kucing):**

```
@Override
void bersuara() {
    System.out.println("Meong!"); // Meng-override metode bersuara()
dari Hewan [cite: 163, 164, 166]
}
```

   Metode bersuara() pada kelas Kucing menggantikan implementasi default dari Hewan yang hanya mencetak "Suara hewan...".

Name : Faatihurrizki Prasojo

NIM : 244107020142

Class : TI_1H

3. **Pada program latihan ketiga, mengapa kita perlu menggunakan keyword super dalam konstruktor subclass? Jelaskan bagaimana super digunakan dalam program tersebut.**

   - **Mengapa perlu super():** Konstruktor tidak dapat diwariskan, sehingga *subclass* harus memiliki konstruktornya sendiri. Kita perlu menggunakan *keyword* **super()** dalam konstruktor *subclass* untuk **memanggil konstruktor yang sesuai dari *superclass***. Pemanggilan super() harus menjadi **pernyataan pertama** dalam konstruktor *subclass*.

   - **Tujuan:** Memastikan bahwa **inisialisasi anggota *superclass*** (seperti nama dan umur pada kelas Hewan) dilakukan dengan benar sebelum *subclass* melanjutkan inisialisasi anggota spesifiknya.

   - **Contoh penggunaan dalam program (Percobaan 3, Subclass Kucing):**

```
class Kucing extends Hewan {
    public Kucing(String nama, int umur) {
        super(nama, umur); // Memanggil konstruktor Hewan
[cite: 203, 204]
        System.out.println("Konstruktor Kucing dipanggil");
    }
}
```

   Dalam contoh ini,
   super(nama, umur); memanggil konstruktor Hewan(String nama, int umur) untuk menginisialisasi atribut nama dan umur yang diwarisi[16161616].

4. **Pada program latihan keempat, jelaskan perbedaan antara single inheritance dan multilevel inheritance. Berikan contoh implementasi keduanya dari program tersebut.**

   - **Single Inheritance:** Terjadi ketika sebuah *subclass* **hanya mewarisi dari satu *superclass***[17]. Ini adalah bentuk pewarisan yang paling sederhana.
   - **Multilevel Inheritance:** Terjadi ketika sebuah *subclass* **mewarisi dari *subclass* lain**, membentuk **rantai pewarisan**[18181818181818].
   - **Contoh Implementasi dari Percobaan 4:**
     - **Single Inheritance:** Kelas **Mamalia** mewarisi langsung dari **Hewan** (class Mamalia extends Hewan)[19].
     - **Multilevel Inheritance:** Kelas **Kucing** mewarisi dari **Mamalia**, yang kemudian mewarisi dari **Hewan** (class Kucing extends Mamalia). Ini membentuk rantai: **Hewan Mamalia Kucing**.

Name : Faatihurrizki Prasojo

NIM : 244107020142

Class : TI_1H

5. **Pada program latihan kelima, jelaskan peran access control (public, private, protected) dalam inheritance. Bagaimana cara super digunakan untuk mengakses anggota dari superclass?**

- **Peran Access Control dalam Inheritance:** Access control (*public*, *private*, *protected*, *default*) mengatur tingkat akses terhadap atribut dan metode, yang sangat penting dalam menjaga enkapsulasi.
  - **public**: Dapat diakses dari mana saja, termasuk *subclass* di *package* berbeda.
  - **protected**: Dapat diakses dalam *package* yang sama **dan oleh *subclass***, bahkan jika *subclass* berada di *package* yang berbeda. Ini memungkinkan akses terbatas yang diperlukan untuk inheritance, seperti yang terjadi pada atribut umur di Percobaan 5.
  - **private**: **Hanya dapat diakses dari dalam kelas itu sendiri**. Atribut *private* (*e.g.,* nama di Hewan) tidak dapat diakses langsung oleh *subclass*.
- **Cara super digunakan untuk mengakses anggota dari *superclass***:
  - *Keyword* **super** digunakan untuk mengakses anggota (atribut dan metode) dari *superclass*.
  - Ini sangat berguna ketika *subclass* memiliki metode yang **meng-*override*** metode dari *superclass*, tetapi tetap perlu memanggil implementasi *superclass*.
  - **Contoh dalam program (Percobaan 5, Subclass Kucing):**

```
public void info() {
    super.info(); // Menggunakan 'super' untuk mengakses metode info() dari
parent class (Hewan) [cite: 287, 288]
    System.out.println("Warna bulu: " + warnaBulu);
}
```

Dengan super.info(), kelas Kucing memanggil metode info() milik Hewan terlebih dahulu untuk menampilkan nama dan umur, kemudian menambahkan informasi spesifiknya (warnaBulu).

Name : Faatihurrizki Prasojo

NIM : 244107020142

Class : TI_1H

6. **Pada program latihan keenam, bagaimana konsep abstract class dan method overriding digunakan untuk menciptakan struktur yang fleksibel dalam memodelkan berbagai jenis kendaraan?**

- **Abstract Class (Kendaraan):**
  - Kelas
    Kendaraan dideklarasikan sebagai **abstract class**. Ini berarti kelas ini **tidak dapat diinstansiasi secara langsung** (objek Kendaraan tidak bisa dibuat), tetapi hanya bisa dijadikan *superclass*.
  - Tujuannya adalah untuk mendefinisikan
    template atau kontrak umum untuk semua jenis kendaraan (memiliki merk, tahunProduksi, dan dapat jalankan()).

- **Abstract Method (jalankan()):**
  - Metode
    jalankan() dideklarasikan sebagai **abstract void jalankan()**. Ini berarti metode tersebut **hanya memiliki deklarasi tanpa implementasi**.
  - Tujuannya adalah
    memaksa setiap *subclass* non-abstrak (seperti Mobil dan Motor) **untuk mengimplementasikan** metode jalankan() dengan caranya masing-masing.

- **Method Overriding:**
  - Setiap
    subclass (Mobil, Motor) **wajib meng-*override*** dan memberikan implementasi spesifik untuk metode jalankan().
  - Mobil mengimplementasikan jalankan() dengan mencetak "Mobil [merk] berjalan...".
  - Motor mengimplementasikannya dengan mencetak "Motor [merk] berjalan...".
  - Struktur ini **fleksibel** karena:
    1. **Konsistensi:** Semua kendaraan pasti memiliki metode jalankan() (dijamin oleh *abstract method*).
    2. **Kekhususan:** Implementasi jalankan() berbeda untuk setiap jenis kendaraan (dicapai melalui *method overriding*).
    3. **Penggunaan Ulang:** Atribut umum (merk, tahunProduksi) dan metode umum (info()) diwarisi dari Kendaraan, menghindari duplikasi kode.