Tugas Pendahuluan: Modul 8

NIM: 105223006 **Nama: Florence Regis Lake**

Instruksi: Jawablah pertanyaan-pertanyaan berikut dengan jelas dan rinci. Gunakan contoh kode jika diperlukan untuk memperjelas jawaban Anda. Kumpulkan dengan format **TP8_NIM_PrakPBO.pdf.** Deadline, Kamis 15 Mei 2025, pukul 13:00 (sebelum praktikum). Link pengumpulan: https://forms.gle/AcPBWgx7is21tJUi7

Soal

1. Jelaskan konsep IS-A relationship dalam Java! Berikan contoh implementasinya dalam bentuk kode program sederhana. Apa manfaat dari penggunaan hubungan ini dalam pemrograman berorientasi objek?

   Jawab:

IS-A, atau bisa disebut juga konsep *inheritance*, adalah fitur dimana sebuah object dapat mempunyai object turunan [1]. Dalam Java, hubungan ini diimplementasikan menggunakan kata kunci extends (untuk inheritance antar class) atau implements (untuk implementasi interface). Manfaat dari penggunaan hubungan ini adalah dapat menjadikan program lebih efisien, terstruktur, dan fleksibel tergantung dari kasus yang diimplementasikan.

Contoh implementasi:

```
// Kelas induk (superclass)
class Hewan {
    void makan() {
        System.out.println("Hewan sedang makan...");
    }
}

// Kelas turunan (subclass)
class Kucing extends Hewan {
    void mengeong() {
```

```
        System.out.println("Kucing mengeong: Meow!");
    }
}

// Program utama
public class Main {
    public static void main(String[] args) {
        Kucing kucingku = new Kucing();
        kucingku.makan();      // Pewarisan dari Hewan
        kucingku.mengeong();   // Metode dari kelas Kucing
    }
}
```

2. Apa yang dimaksud dengan Has-A relationship dalam Java? Jelaskan dengan contoh kode dan bandingkan perbedaannya dengan IS-A relationship!

   Jawab:

Dalam Java, hubungan Has-A yang juga disebut komposisi digunakan untuk reusabilitas kode, yang berarti bahwa sebuah objek dari satu kelas memiliki referensi ke objek dari kelas lain atau objek lain dari kelas yang sama [2]. Perbandingannya dengan IS-A relationship adalah IS-A merupakan pewarisan seperti B adalah A, sedangkan HAS-A merupakan komposisi seperti B memiliki A.

Contoh implementasi:

```
        // Kelas Engine (komponen)
        class Engine {
            void start() {
                System.out.println("Mesin menyala...");
            }
        }

        // Kelas Mobil memiliki Engine → Has-A relationship
        class Mobil {
            private Engine mesin;
```

```
        Mobil() {
            mesin = new Engine();
        }

        void nyalakanMobil() {
            mesin.start();
            System.out.println("Mobil berjalan...");
        }
    }

    // Program utama
    public class Main {
        public static void main(String[] args) {
            Mobil avanza = new Mobil();
            avanza.nyalakanMobil();
        }
}
```

3. Kata kunci instanceof sering digunakan dalam Java. Jelaskan fungsi dari keyword ini dan berikan contoh penggunaannya! Dalam situasi seperti apa instanceof sebaiknya digunakan?

Jawab:

Dalam Java, instanceof adalah kata kunci yang digunakan untuk memeriksa apakah sebuah variabel referensi mengandung referensi objek dari tipe tertentu atau tidak [3]. Instanceof biasanya digunakan saat melakukan downcasting (dari superclass ke subclass) untuk menghindari ClassCastException dan menangani polimorfisme dalam hirarki kelas yang kompleks.

Contoh implementasi:

```
class Hewan {
   void suara() {
      System.out.println("Hewan bersuara");
   }
}
```

```
class Kucing extends Hewan {
    void suara() {
        System.out.println("Meow");
    }
}

public class Main {
    public static void main(String[] args) {
        Hewan hewan = new Kucing();

        // Mengecek apakah objek 'hewan' adalah instance dari Kucing
        if (hewan instanceof Kucing) {
            Kucing kucing = (Kucing) hewan; // Safe casting
            kucing.suara(); // Output: Meow
        }
    }
}
```

4. Java mendukung beberapa tipe pewarisan. Jelaskan tipe-tipe pewarisan dalam Java dan berikan contoh kode untuk masing-masingnya! Sertakan penjelasan tentang pewarisan tunggal, pewarisan multilevel, dan pewarisan hierarkis. Mengapa Java tidak mendukung pewarisan berganda secara langsung?

Jawab:

Java mendukung 3 tipe pewarisan yaitu pewarisan tunggal dimana subclass hanya dapat mewarisi dari satu superclass, pewarisan multilevel dimana subclass mewarisi dari superclass, dan subclass tersebut dapat menjadi superclass bagi subclass lainnya, dan pewarisan hierarkis dimana banyak subclass yang mewarisi dari satu superclass [4]. Sedangkan alasan mengapa pewarisan berganda tidak dapat dilakukan pada Java dikarenakan kompleksitas pada struktur hierarki dan dapat menimbulkan masalah Diamond Problem.

Contoh implementasi:

Pewarisan tunggal

```
class Hewan {
    void makan() {
        System.out.println("Hewan sedang makan.");
    }
}

class Anjing extends Hewan {
    void menggonggong() {
        System.out.println("Anjing menggonggong.");
    }
}

public class Main {
    public static void main(String[] args) {
        Anjing anjing = new Anjing();
        anjing.makan();  // Menggunakan metode dari kelas Hewan
        anjing.menggonggong();  // Metode khusus untuk Anjing
    }
}
```

Pewarisan multilevel

```
class Hewan {
    void makan() {
        System.out.println("Hewan sedang makan.");
    }
}

class Anjing extends Hewan {
    void menggonggong() {
        System.out.println("Anjing menggonggong.");
    }
}
```

```
    class AnjingPeliharaan extends Anjing {
        void bermain() {
            System.out.println("Anjing peliharaan sedang bermain.");
        }
    }

    public class Main {
        public static void main(String[] args) {
            AnjingPeliharaan anjingPeliharaan = new AnjingPeliharaan();
            anjingPeliharaan.makan();  // Menggunakan metode dari kelas Hewan
            anjingPeliharaan.menggonggong();  // Menggunakan metode dari kelas
    Anjing
            anjingPeliharaan.bermain();        // Metode khusus dari kelas
    AnjingPeliharaan
        }
}
```

Pewarisan hierarkis

```
    class Hewan {
        void makan() {
            System.out.println("Hewan sedang makan.");
        }
    }

    class Anjing extends Hewan {
        void menggonggong() {
            System.out.println("Anjing menggonggong.");
        }
    }

    class Kucing extends Hewan {
        void mengeong() {
            System.out.println("Kucing mengeong.");
```

```
        }
    }

    public class Main {
        public static void main(String[] args) {
            Anjing anjing = new Anjing();
            anjing.makan();  // Menggunakan metode dari kelas Hewan
            anjing.menggonggong();  // Metode khusus Anjing

            Kucing kucing = new Kucing();
            kucing.makan();  // Menggunakan metode dari kelas Hewan
            kucing.mengeong();  // Metode khusus Kucing
        }
}
```

5. Jelaskan perbedaan antara agregasi dan komposisi dalam konteks relasi antar objek dalam Java! Berikan contoh implementasi kode untuk masing-masing relasi tersebut. Kapan sebaiknya menggunakan agregasi dan kapan menggunakan komposisi?

   Jawab:

Agregasi adalah bentuk dimana sebuah object memiliki lifecycle nya sendiri tapi dengan kepemilikan dan class anak tidak dapat memiliki class induknya, sedangkan relasi komposisi sama halnya dengan agregasi dimana suatu class merupakan bagian utuh dari class lainya namun pada hal ini satu bagian class tersebut akan sangat bergantung pada keberadaan class lainya [5]. Agregasi sebaiknya digunakan ketika objek bagian tidak eksklusif dimiliki oleh objek induk dan dapat dikelola atau digunakan bersama oleh objek lain. Sebaliknya, komposisi lebih tepat digunakan ketika objek bagian hanya dapat eksis dalam konteks objek induk dan sepenuhnya bergantung pada siklus hidup objek induk.

Contoh implementasi:

Agregasi

```
    class Mahasiswa {
        String nama;
```

```
    Mahasiswa(String nama) {
        this.nama = nama;
    }
}

class Dosen {
    String nama;
    // Agregasi: Dosen memiliki referensi ke Mahasiswa
    Mahasiswa mahasiswa;

    Dosen(String nama, Mahasiswa mahasiswa) {
        this.nama = nama;
        this.mahasiswa = mahasiswa;
    }

    void info() {
        System.out.println(nama + " membimbing " + mahasiswa.nama);
    }
}

public class Main {
    public static void main(String[] args) {
        Mahasiswa mhs = new Mahasiswa("Sinta");
        Dosen dosen = new Dosen("Pak Budi", mhs);
        dosen.info();
    }
}
```

Komposisi

```
class Ruangan {
    String nama;

    Ruangan(String nama) {
```

```
        this.nama = nama;
    }

    void tampil() {
        System.out.println("Ruangan: " + nama);
    }
}

class Rumah {
    // Komposisi: Ruangan diciptakan dan dimiliki penuh oleh Rumah
    private Ruangan ruangTamu;
    private Ruangan kamarTidur;

    Rumah() {
        ruangTamu = new Ruangan("Ruang Tamu");
        kamarTidur = new Ruangan("Kamar Tidur");
    }

    void tampil() {
        ruangTamu.tampil();
        kamarTidur.tampil();
    }
}

public class Main {
    public static void main(String[] args) {
        Rumah rumah = new Rumah();
        rumah.tampil();
    }
}
```

Referensi

[1]. Jusuf, H. (n.d.). *Konsep Pemrograman Berorientasi Objek (Modul 01)*. MSIM4301, Edisi 1. Universitas Terbuka.

[2]. GeeksforGeeks, "What is HAS-A relation in Java?," Aug. 2, 2022. [Online]. Available: https://www.geeksforgeeks.org/what-is-has-a-relation-in-java/

[3]. GeeksforGeeks, "instanceof keyword in Java," *GeeksforGeeks*, May 11, 2023. [Online]. Available: https://www.geeksforgeeks.org/instanceof-keyword-in-java/

[4]. RPL UPI, "Mendalami Konsep Inheritance dalam Bahasa Pemrograman Java," *Program Studi Rekayasa Perangkat Lunak UPI*, May 6, 2023. [Online]. Available: https://rpl.upi.edu/mendalami-konsep-inheritance-dalam-bahasa-pemrograman-java/

[5]. N. Devi, "Relasi Asosiasi, Komposisi, Agregasi," *Natalia Devi – Student Telkom University*, Mar. 5, 2020. [Online]. Available: https://nataliadevi.student.telkomuniversity.ac.id/2020/03/05/relasi-asosiasi-komposisi-agregasi/