INHERITANCE (PEWARISAN) PADA BAHASA JAVA

Tugas 2 Program Lanjutan

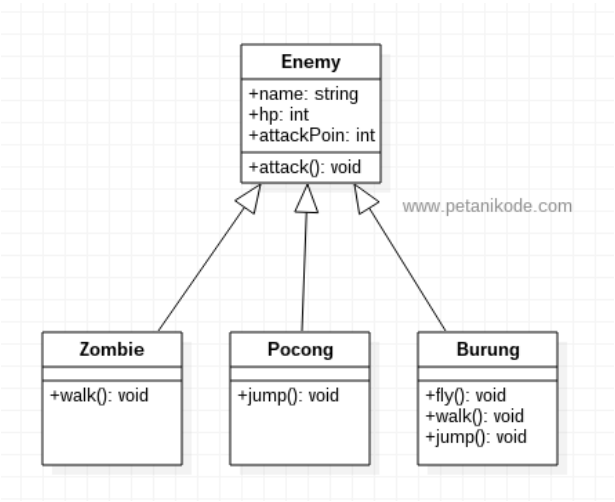
Nama Anggota Kelompok :

1. Abmi Sukma Edri (12250120341)
2. Daffa Ikhwan Nurfauzan (12250110979)
3. Dwi Jelita Adhliyah (12250120331)

1. Dasar – dasar Inheritance (pewarisan) pada java

Inheritance merupakan proses pewarisan data dan method dari suatu class yang telah ada kepada suatu class baru**.** Sebuah class di Java, bisa memiliki satu atau lebih keturunan atau class anak. Class anak akan memiliki warisan properti dan method dari class ibu.

a) **Class** : Inheritance dilakukan dengan membuat dua kelas, masing-masing sebagai kelas induk dan kelas turunan..
b) **Superclass** (induk) : Class yang menjadi dasar / induk bagi subclass.
c) **Subclass** (turunan) : Class baru yang dihasilkan dari pewarisan Superclass.
d) **Keyword** ‘extends’ : menunjukkan bahwa sebuah class adalah turunan dari superclass tertentu.
e) **Keyword** ‘super’ : Digunakan untuk memanggil konstruktor dan method dari superclass.
f) **Method** Overriding : Proses menimpa atau mengganti implementasi sebuah method pada kelas induk atau superclass oleh kelas turunan atau subclass.
g) **Method Overloading** : proses membuat beberapa method dengan nama yang sama pada suatu kelas, namun dengan parameter yang berbeda.
h) **Konstruktor** : metode khusus dalam class turunan atau subclass yang digunakan untuk menginisialisasi objek class turunan dan class induk atau superclass.
i) **Kontrol Akses** : Kkonsep yang digunakan untuk mengatur aksesibilitas atau visibilitas atribut dan method pada class turunan yang diwarisi dari class induk.



Contoh pemakaian inheritance dalam membuat game untuk membuat mthod dan property yang sama agar tidak berulang-ulang pembuatannya .

(Sumber :www.petanikode.com).

2. Membuat Kelas Turunan

ICara membuat kelas turunan menggunakan kata kunci **‘extends’**. Kata kunci ini digunakan untuk menunjukkan bahwa sebuah kelas (kelas turunan/subclass) mewarisi sifat dan karakteristik dari kelas lain (kelas induk/superclass).

```
class NamaTurunan extends NamaInduk {
```

3. Aturan Inheritance

a) Sebuah kelas hanya dapat mewarisi properti dari satu kelas induk, ditentukan dengan kata kunci '**extends**' yang diikuti dengan nama kelas induknya.

```
public class Kucing extends animal {
    // Kucing merupakan turunan dari kelas animal
}
```

b) Sebuah kelas turunan akan mewarisi semua atribut dan method yang ada di kelas induknya, kecuali atribut dan method yang bersifat private (tersembunyi).

```
public class animal {
    public String jenis;
    protected int umur;
    private String makanan;

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

    public void bergerak() {
        System.out.println("Hewan sedang bergerak");
    }
}
```

```
public class Kucing extends animal {                    ⚠ 1  ✓ 9
    // Kucing akan mewarisi atribut jenis dan umur dari Hewan,
    // serta method makan() dan bergerak()
}
```

c) Sebuah kelas turunan dapat menambahkan atribut dan method sendiri, yang tidak ada di kelas induknya.

```
public class Kucing extends animal {
    public String warnaBulu;

    public void bersuara() {
        System.out.println("Meow!");
    }
    // Kucing memiliki atribut warnaBulu dan method bersuara() sendiri
}
```

d) Kelas turunan dapat mengganti metode yang ada di kelas induknya dengan membuat metode baru dengan nama yang sama dan parameter yang sama.

4. Konstruktor kelas turunan

Konstruktor kelas turunan (konstruktor subkelas) adalah konstruktor milik kelas turunan atau subkelas yang menginisialisasi objek dari subkelas. Konstruktor kelas turunan biasanya dipanggil setelah konstruktor kelas induk atau kelas induk (superclass constructor) dipanggil terlebih dahulu.

Di Java, saat membuat konstruktor pada kelas turunan, kita harus memanggil konstruktor kelas induk menggunakan kata kunci 'super'. Hal ini dilakukan untuk menginisialisasi semua properti dan atribut yang dimiliki oleh kelas induk sebelum digunakan oleh kelas turunan.

Contoh:

```
public class animal {                                    ⚠ 4
    1 usage
    private String jenis;

    1 usage
    public animal(String jenis) {
        this.jenis = jenis;
        System.out.println("Konstruktor Hewan dipanggil");
    }

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

```
public class Kucing extends animal {                     ⚠ 6
    1 usage
    private String warna;

    public Kucing(String jenis, String warna) {
        super(jenis);
        this.warna = warna;
        System.out.println("Konstruktor Kucing dipanggil");
    }

    public void bersuara() {
        System.out.println("Meow!");
    }
}
```

Pada contoh di atas, konstruktor pada kelas **animal** menerima satu parameter jenis dan dalam konstruktor itu menginisialisasi atribut jenis dengan nilai dari parameter tersebut. Sedangkan pada kelas **Kucing**, konstruktor menerima dua parameter jenis dan warna, dan memanggil konstruktor pada kelas **animal** dengan kata kunci **"String (jenis)",** sehingga atribut jenis pada kelas **animal** dapat diinisialisasi terlebih dahulu sebelum diinisialisasi atribut warna pada kelas **Kucing**. Setelah itu, konstruktor pada kelas **Kucing** dijalankan dan mencetak pesan **"Konstruktor Kucing dipanggil"**.

5. Pengaruh control akses pada pewarisan

Kontrol akses pada inheritance membuat konsep dalam pemrograman berorientasi objek yang mengatur akses bagaimana suatu kelas anak dapat membuat metode dari kelas induknya. Kontrol akses pada inheritance mempengaruhi keamanan dan fleksibilitas dari sebuah program.

Secara umum, ada tiga jenis control akses yang terdapat pada inheritance:

a) **Private** : Membatasi akses ke property atau method hanya untuk (kelas induk/superclass). Kelas turunan tidak dapat mengakses property atau metode private dari kelas induknya.
b) **Protected** : Membatasi akses ke property atau metode hanya untuk kelas induknya dan kelas anaknya. Kelas lain diluar inheritance tidak dapat mengakses property atau metode protected
c) **Public** : Memungkinkan akses ke properti atau metode dari kelas induk induk dan kelas turunan, serta kelas lain di luar lingkup inheritance.

Modifier	Class	Package	Subclass	World
Public	Ya	Ya	Ya	Ya
Protected	Ya	Ya	Ya	No
Private	Ya	No	No	No

Pada akhirnya, penggunaan control akses pada inheritance bisa mempengaruhi security atau keamanan dan fleksibilitas dari program. Penggunaan control akses yang tepat dapat menghindari perubahan dan kesalahan yang tidak disengaja pada kelas induk dan kelas turunan.

6. Penggunaan keyword ‘super’

Keyword "**super**" digunakan dalam Bahasa Java untuk mengakses konstruktor, method atau variabel yang punya kelas induk atau superclass dari kelas turunan atau subclass.

Ketika sebuah kelas turunan atau subclass dibuat, kelas tersebut mewarisi semua atribut dan metode dari kelas induknya. Namun, jika kelas turunan tersebut punya atribut atau metode dengan nama yang sama seperti kelas induk, maka penggunaan "**super**" diperlukan untuk membedakan antara atribut atau metode yang dari kelas turunan dan kelas induk.

Contoh penggunaan **"super"** pada konstruktor adalah sebagai berikut:

```
class Superclass {
    1 usage
    public Superclass(String arg) {
        System.out.println("Ini adalah konstruktor dari Superclass dengan argumen " + arg);
    }
}

class Subclass extends Superclass {
    public Subclass(String arg) {
        super(arg);
        System.out.println("Ini adalah konstruktor dari Subclass");
    }
}
```

Dari contoh ini, kelas Subclass mewarisi konstruktor dari Superclass dengan menggunakan keyword **"extends"**, karena kelas Subclass memiliki konstruktor dengan parameter yang sama dengan konstruktor milik Superclass, jadi keyword **"super"** digunakan saat memanggil konstruktor dari Superclass dan mengirimkan argumen ke konstruktor tersebut..

Contoh penggunaan "super" pada method adalah sebagai berikut:

```
class Superclass {
    1 usage  1 override
    public void print() {
        System.out.println("Ini adalah method dari Superclass");
    }
}

class Subclass extends Superclass {
    1 usage
    public void print() {
        super.print();
        System.out.println("Ini adalah method dari Subclass");
    }
}
```

Dalam contoh di atas, kelas Subclass mewarisi method **"print()"** dari Superclass. Kemudian, dalam method **"print()"** milik Subclass, kita ingin menambahkan perintah baru tetapi tetap mau menjalankan perintah yang sudah ada pada method **"print()"** dari Superclass. Untuk itu, kita menggunakan keyword **"super"** untuk memanggil method **"print()"** milik Superclass dan ditambah sebelum ada perintah baru pada method **"print()"** dari Subclass.

7. Method overriding

Method overriding pada Java adalah sebuah proses pergantian dari sebuah method untuk kelas induk atau superclass oleh kelas turunan atau subclass. Dalam method overriding, method pada kelas turunan memiliki nama, parameter, dan tipe kembalian yang sama dengan method pada kelas induk, tetapi penerapan yang berbeda.

Dalam proses method overriding, kelas turunan harus punya aksebilitas yang sama atau lebih bagus dari induknya. Misalnya, jika method dari kelas induk didefinisikan sebagai public, maka method di kelas turunan juga harus didefinisikan sebagai public atau protected. Tapi, jika method didefinisikan private, maka kelas turunannya tidak bisa di overriding.

Berikut adalah contoh implementasi method overriding pada Java:

```java
class Animal {
    2 overrides
    public void makeSound() {
        System.out.println("Hewan membuat suara");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }
}
```

Pada contoh di atas, kelas Cat dan Dog mewarisi method **"makeSound()"** dari kelas Animal dan meng-overriding method tersebut dengan implementasi yang berbeda. Ketika method **"makeSound()"** dipanggil pada objek Cat, maka akan mencetak "Meow", sedangkan ketika method **"makeSound()"** dipanggil pada objek Dog, maka akan mencetak "Woof".

8. Method overloading

Method overloading pada Java adalah proses membuat beberapa method dengan nama yang sama pada suatu kelas, namun dengan parameter yang berbeda dalam jumlah, tipe data, dan urutan parameter.

Dalam proses method overloading, Java dapat membedakan setiap method yang parameter berbeda dan memanggil method yang sesuai dengan parameter yang diberikan. Hal ini bisa memberi kita membuat method yang lebih fleksibel dan mudah digunakan oleh programmer.

Berikut adalah contoh implementasi method overloading pada Java:

```java
class Calculator {
    public int add(int num1, int num2) {
        return num1 + num2;
    }

    public int add(int num1, int num2, int num3) {
        return num1 + num2 + num3;
    }

    public double add(double num1, double num2) {
        return num1 + num2;
    }
}
```

Dalam contoh di atas, kelas Calculator memiliki tiga method yang bernama **"add"**, tapi dengan parameter yang berbeda. Method pertama menerima dua integer dan return hasil penjumlahan keduanya. Method kedua menerima tiga integer dan reurn hasil penjumlahan ketiganya. Method ketiga menerima dua double dan return hasil penjumlahan keduanya.

Ketika method **"add"** dipanggil pada objek Calculator, Java akan membedakan setiap method berdasarkan parameter yang diberikan dan memanggil method yang sesuai dengan parameter tersebut. Sebagai contoh, jika method **"add"** dipanggil dengan dua **integer**, maka method pertama akan dipanggil, jika panggil dua **double**, maka method ketiga akan dipanggil.

9. Bagaimana memanggil Method kelas induk (Superclass) ?

Dalam Java, untuk memanggil method kelas induk atau superclass dari kelas turunan atau subclass, kita dapat menggunakan kata kunci **"super"** untuk menuju method atau variable kelas induk dalam kelas turunan.

Untuk memanggil method kelas induk dari kelas turunan, kita dapat menggunakan **"super.namaMethod()"**. Dalam sintaksis tersebut, **"super"** menunjukan pada kelas induk, dan "namaMethod" menuju method mana yang ingin dipanggil.

Berikut adalah contoh implementasi memanggil method kelas induk dari kelas turunan pada Java:

```java
class Animal {
    1 usage  1 override
    public void makeSound() {
        System.out.println("Hewan membuat suara");
    }
}

class Cat extends Animal {
    1 usage
    @Override
    public void makeSound() {
        super.makeSound(); // memanggil method makeSound() dari kelas induk
        System.out.println("Meow");
    }
}
```

Pada contoh di atas, kelas Cat mewarisi method **"makeSound()"** dari kelas Animal dan meng-overriding method tersebut dengan pengunaan yang berbeda. Dalam method

"makeSound()" pada kelas Cat, kita memanggil method "makeSound()" dari kelas Animal dengan menggunakan keyword **"super"** sebelum nama method, dan kemudian mencetak "Meow" setelahnya. Dengan menggunakan "super", kita bisa memanggil method kelas induk dari kelas turunan dan mengakses pengunaannya dari kelas induk tersebut.

10. Penggunaan Keyword 'Final' Pada Inheritance

Dalam inheritance pada Java, pengunaan keyword "final" untuk membatasi pewarisan kelas dan method dari kelas induk ke kelas turunan. Ada dua penggunaan keyword "final" pada inheritance:

a) **Final class**: Ketika sebuah kelas dideklarasikan sebagai final class, kita **tidak** dapat membuat kelas turunan dari kelas tersebut. Contohnya:

```
final class Animal {
    // implementasi kelas Animal
}

// error, karena kelas Animal dideklarasikan sebagai final class
class Cat extends Animal {
    // implementasi kelas Cat
}
```

b) **Final method**: Ketika sebuah method dideklarasikan sebagai final method pada kelas induk, kita **tidak** dapat mengganti implementasi atau men override method tersebut pada kelas turunan. Contohnya:

```
class Animal {
    1 usage  1 override
    public void makeSound() {
        System.out.println("Hewan membuat suara");
    }
}

class Cat extends Animal {
    1 usage
    @Override
    public void makeSound() {
        super.makeSound(); // memanggil method makeSound() dari kelas induk
        System.out.println("Meow");
    }
}
```

Pada contoh syntax tersebu, method **"makeSound()"** pada kelas Animal dideklarasikan sebagai final method, sehingga **tidak** dapat di-overriding pada kelas turunan seperti kelas Cat. Jika kita mencoba meng-overriding method tersebut pada kelas Cat, maka akan terjadi error.

Penggunaan "final" pada inheritance dapat membantu kita untuk membatasi perubahan pada kelas dan method yang telah kita atur pada kelas induk, sehingga **mengurangi** kesalahan saat implementasi pada kelas turunan.