BAB V

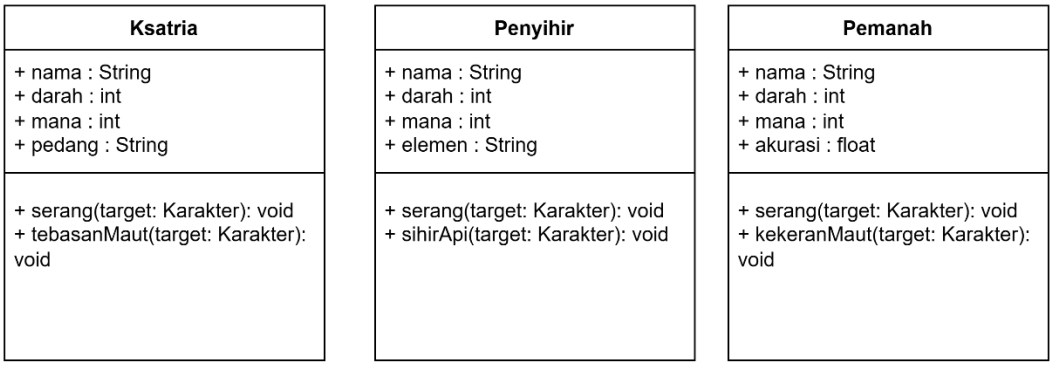
Inheritance (Pewarisan)

Pada materi kali ini, kita akan belajar tentang *Inheritance* (Pewarisan) pada pemrograman berorientasi objek (*OOP*). Apa arti Pewarisan dalam OOP sebenarnya? Mari kita belajar Bersama-sama. *OOP* adalah paradigma pemrograman yang menggunakan "Objek" untuk memodelkan data dan perilaku. Dalam *OOP,* terdapat beberapa konsep utama, yaitu:

1. Encapsulation : menyembunyikan detail implementasi dan hanya menyediakan antarmuka publik.
2. Abstraction : menyederhanakan kompleksitas dari suatu kode dengan menampilkan fitur yang penting saja
3. Inheritance : memungkinkan sebuah kelas (subclass) mewarisi properti dan metode dari kelas lain (superclass)
4. Polymorphism : memungkinkan objek untuk mengambil banyak bentuk, biasanya melalui method overriding atau interface

Modul ini akan lebih membahas pada bagian *Inheritance (Pewarisan)*

Mari misalkan bahwa sekarang kita adalah seorang pengembang game online, dan memulai untuk membuat karakter-karakter game kita terlebih dahulu ide awal kita, dan muncul dipikiran kita untuk membuat 3 karakter seperti di bawah ini,



Lalu tanpa basa-basi, kita langsung menulis seluruh kodenya berdasarkan dengan diagram kelas tersebut :

1. Ksatria.java

```java
class Ksatria {
    String nama;
    String pedang;
    int darah;
    int mana;

    void serang() {
        System.out.println("menyerang");
    }

    void tebasanMaut() {
        System.out.println("tebasan maut");
    }
}
```

2. Penyihir.java

```java
class Penyihir {
    String nama;
    String elemen;
    int darah;
    int mana;

    void serang() {
        System.out.println("menyerang");
    }

    void sihirApi(){
        System.out.println("menyihir!");
    }
}
```

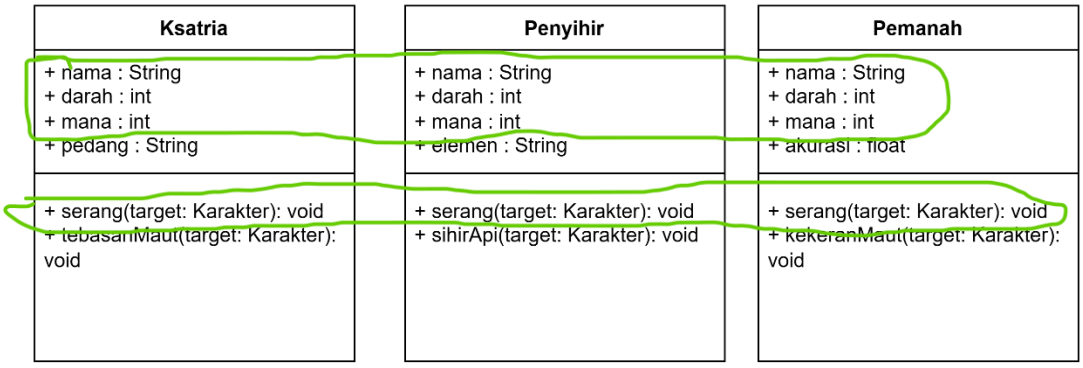
3. Pemanah.java

```java
class Pemanah {
    String nama;
    int darah;
    int mana;
    float akurasi;

    void serang() {
        System.out.println("menyerang");
    }

    void kekeranMaut() {
        System.out.println("panah api");
    }
}
```

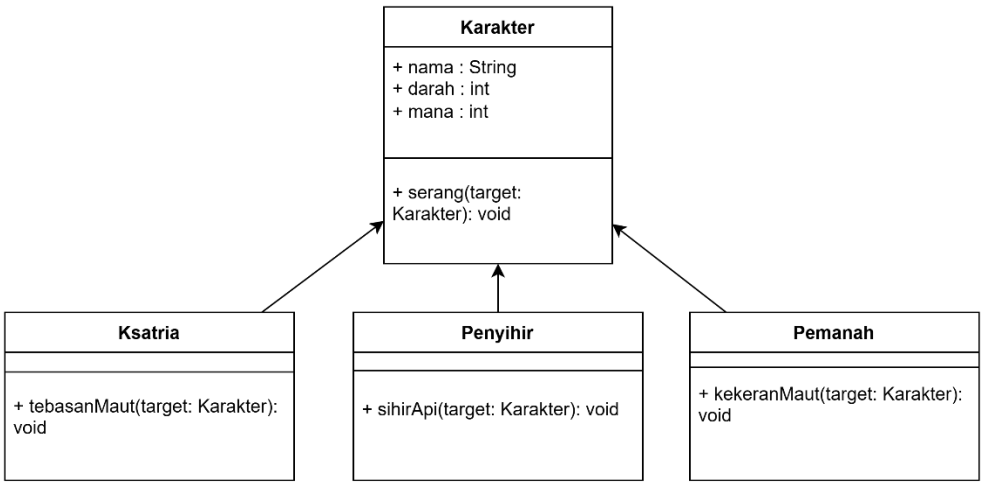
Apa boleh seperti ini? Jawabannya boleh-boleh saja, akan tetapi tidak efektif dalam menulis kode. Dalam *Programming* ini disebut dengan *WET (We Enjoy Typing)* sebagai programmer kita seharusnya selalu menerapkan konsep *DRY (Don't Repeat Yourself)*.

Disinilah Inheritance berfungsi penting.

Pertama mari kita telaah dan lihat hal-hal apa saja yang sama.



Setelah dilihat bahwa ada beberapa hal yang sama, maka bisa dibuatkan seperti di bawah ini



Terus bagaimana kalau implementasi kodenya untuk karakter jika bentuk nya seperti ini? jawabannya sama seperti biasa dan ditulis seperti biasa, hanya saja berbeda untuk kelas childnya

1. Karakter.java

```java
class Karakter {
    String nama;
    int darah;
    int mana;

    void serang(Karakter target) {
        System.out.println("menyerang");
    }
}
```

Sebelum membahasnya lebih dalam mari kita lanjutkan ke definisinya terlebih dahulu.

1. Definisi *Inheritance*

Inheritance (Pewarisan) adalah salah satu pilar utama dalam OOP di mana sebuah kelas baru (biasa disebut dengan child class atau subclass) dapat mewarisi atribut dan metode dari kelas yang sudah ada (disebut superclass atau parent class). Konsep ini membantu dalam:

- Menghindari Redundansi Kode: Kode yang sama tidak perlu ditulis berulang di setiap kelas.
- Mempermudah Pemeliharaan: Perubahan di superclass otomatis diterapkan pada subclass
- Hierarki Kelas: Membentuk hubungan "is-a" (misalnya, seekor anjing adalah seekor hewan)

Contoh Perumpamaan :

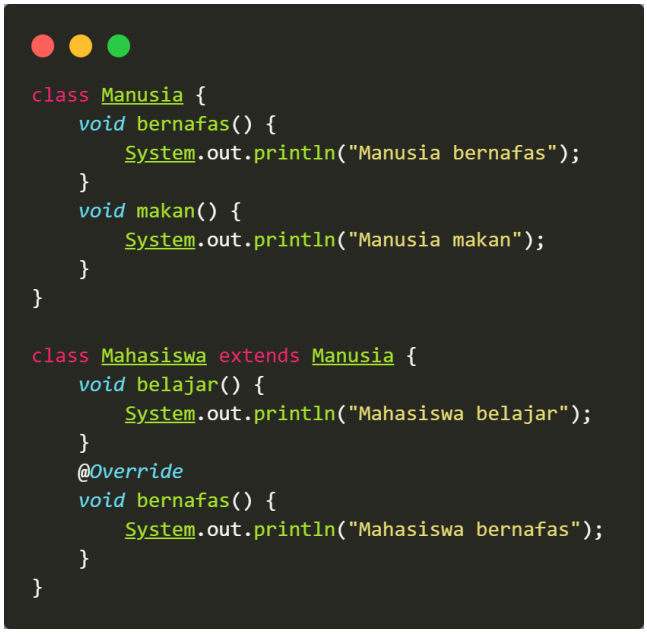
Bayangkan keluarga di mana sifat-sifat seperti warna mata, bentuk wajah, atau kebiasaan tertentu diwariskan dari orang tua ke anak-anak. Begitu juga dengan kelas di Java, dimana subclass "mewarisi" properti dan perilaku dari superclass. Atau jika kurang jelas dapat dilihat pada contoh diatas sebelum definisi dari *Inheritance*

2. Jenis-jenis *Inheritance*

Inheritance sendiri terdiri dari 3 jenis yaitu :

A. Single *Inheritance*

Ketika suatu kelas mewarisi kelas lain, itu dikenal dengan Single *Inheritance.* Dalam contoh yang diberikan di bawah ini, kelas Mahasiswa mewarisi kelas Manusia, jadi ada *Inheritance* atau pewarisan didalamnya.



```java
class Manusia {
    void bernafas() {
        System.out.println("Manusia bernafas");
    }
    void makan() {
        System.out.println("Manusia makan");
    }
}

class Mahasiswa extends Manusia {
    void belajar() {
        System.out.println("Mahasiswa belajar");
    }
    @Override
    void bernafas() {
        System.out.println("Mahasiswa bernafas");
    }
}
```

B. Multilevel *Inheritance*

Multilevel *inheritance* adalah sebuah konsep PBO di mana suatu class mewarisi class lain, yang kemudian diwarisi oleh class lain lagi, membentuk rantai pewarisan.

Contoh Sederhana :

- Kelas Hewan sebagai superclass atau parent class
- Kelas Mamalia sebagai subclass yang mewarisi kelas hewan
- Kelas Sapi sebagai subclass yang mewarisi kelas mamalia

Sama seperti pada kehidupan manusia yang dimana ada kakek lalu ibu hingga diri kita sendiri.

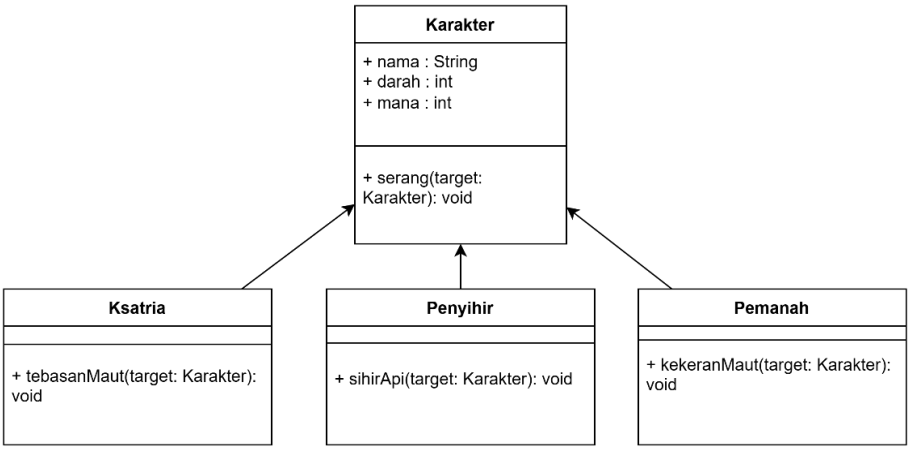
Implementasi dalam kode :

```
class Hewan {
    void eat(){
        System.out.println("Hewan Makan");
    }
}

class Mamalia extends Hewan {
    void jalan(){
        System.out.println("Mamalia Jalan");
    }
}

class Sapi extends Mamalia {
    void suara(){
        System.out.println("Sapi bersuara");
    }
}
```

C. Hierarchical Inheritance

Konsep terakhir dalam *Inheritance* yang di mana satu superclass atau parent class diwarisi oleh beberapa atau banyak subclass atau child class.

Contoh singkatnya seperti yang telah diterangkan di atas dengan diagram class nya seperti di bawah ini



Implementasi dalam kode :

```java
class Karakter {
    String nama;
    int darah;
    int mana;

    void serang(Karakter target) {
        System.out.println("menyerang wak");
    }
}
```

Figure 1 : Class Karakter

```java
class Ksatria extends Karakter {

    void tebasanMaut(Karakter target){
        System.out.println("Kena tebasan gw cik");
    }
}

class Penyihir extends Karakter {
    void sihirApi(Karakter target){
        System.out.println("PANASSS PANASS!!");
    }
}

class Pemanah extends Karakter {
    void kekeranMaut(Karakter target){
        System.out.println("IHH TAKOTNYEEE!!");
    }
}
```

Figure 2 : Class Anakannya

3. Keyword "*this*" dan "*super*" dalam *Inheritance*

1. "*this*"

   Digunakan untuk merujuk ke objek saat ini dalam class yang sama.
   Fungsi utama kata kunci *"this"* :

   - Membedakan variabel instance dan parameter jika namanya sama
   - Memanggil konstruktor lain dalam class yang sama (*'this( )'*)
   - Mengakses metode atau variabel instance dari objek saat ini

   Contoh implementasi *'this'* dalam Java :

```java
class Student {
    String name;
    int age;

    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + this.name + ", Age: " + this.age);
    }

    Student() {
        this("John Doe", 18);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Alice", 20);
        s1.display();

        Student s2 = new Student();
        s2.display();
    }
}
```

   Output yang dihasilkan :

```
Name: Alice, Age: 20
Name: John Doe, Age: 18
```

2. "*super*"
   Digunakan untuk merujuk ke superclass (parent class) dalam *Inheritance*.
   Fungsi utama kata kunci "*super*":
   - Memanggil konstruktor superclass ('*super( )*')
   - Mengakses metode superclass yang telah di-overridden.
   - Mengakses variabel superclass jika nama variabel di subclass yang sama

   Contoh implementasi "*super*" dalam Java :

```java
class Hewan {
    String type = "Hewan";

    Hewan() {
        System.out.println("Hewan constructor called");
    }

    void makeSound() {
        System.out.println("Hewan make sounds");
    }
}

class Kucing extends Hewan {
    String type = "Kucing";

    Kucing() {
        super();
        System.out.println("Kucing constructor called");
    }

    void displayType() {
        System.out.println("Type in child class: " + this.type);
        System.out.println("Type in parent class: " + super.type);
    }


    @Override
    void makeSound() {
        super.makeSound();
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Kucing myDog = new Kucing();
        myDog.displayType();
        myDog.makeSound();
    }
}
```