# JOBSHEET W08 ABSTRACT CLASS

# Nama : Muhammad Abhinaya Zurfa

# Kelas : SIB 2F / 18

# NIM : 2341760186

1. **KOMPETENSI**
   1. Memahami konsep dasar dan tujuan abstract class
   2. Mampu menerapkan abstract class dalam suatu kode program
   3. Mampu membuat subclass yang meng-extend abstract class dengan mengimplementasikan seluruh abstract method-nya

# PENDAHULUAN

Abstract class merupakan class yang **tidak dapat diinstansiasi namun dapat di-extend**. Umumnya abstract class digunakan sebagai **generalisasi** atau **guideline** dari subclass dan hanya bisa digunakan lebih lanjut setelah **di-extend** oleh **concrete class** (class pada umumnya)

Abstract class memiliki karakteristik sebagai berikut:

* 1. Selalu dideklarasikan dengan menggunakan keyword “**abstract class**”
  2. Dapat memiliki atribut dan methods (yang bukan abstract method) seperti concrete class
  3. Umumnya memiliki **abstract method**, yaitu method yang hanya dideklarasikan tetapi tidak memiliki implementasi (body)
     + Abstract method mendifinisikan apa saja yang bisa dilakukan oleh sebuah class namun tidak ada detail bagaimana cara melakukannya
     + Untuk membuat abstract method, hanya menuliskan deklarasi method tanpa body dan menggunakan keyword abstract

Untuk mendeklarasikan abstract class:

# public abstract class <NamaClass>

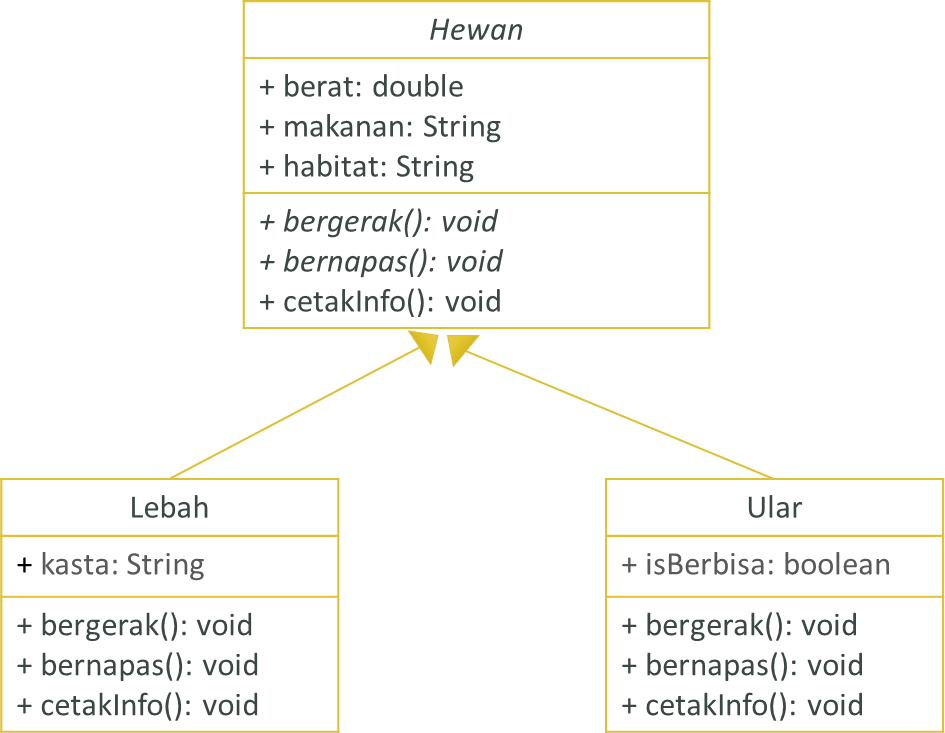
Untuk mendeklarasikan method abstract:

# public abstract <return\_type> <namaMethod>();

Contoh:



Secara umum, notasi class diagram untuk abstract class sama dengan concrete class, namun nama kelas dicetak miring atau ditambah anotasi <<abstract>> di atas nama kelas. Di samping itu abstract method harus dicetak miring juga seperti pada contoh berikut.



Cara menggunaan abstract class:

* + - Abstract class tidak dapat diinstansiasi (tidak dapat dibuat objectnya). Baris kode berikut akan memunculkan *compilation error* “Hewan is abstract; cannot be instantiated”

Hewan hewan1 = new Hewan();

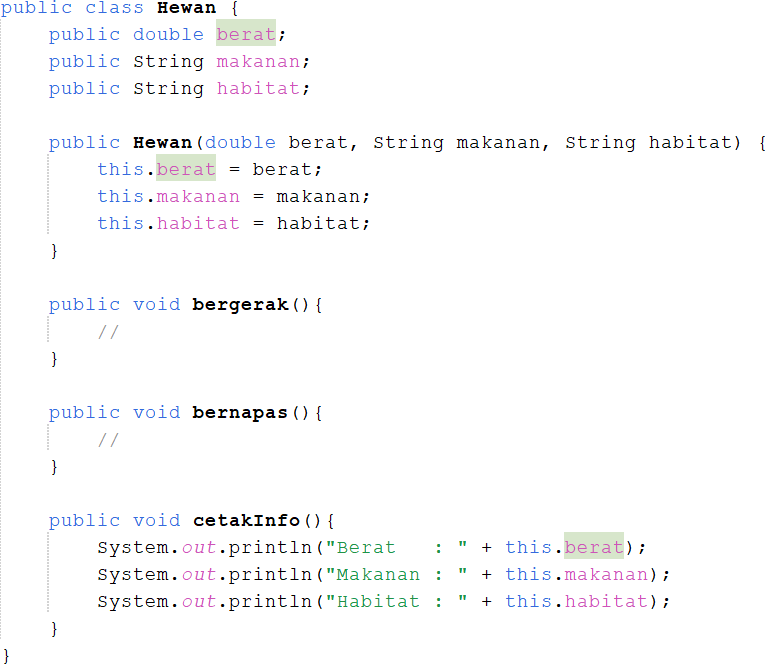
* + - Untuk menggunakan abstract class, dibuat concrete class yang meng-extend abstract class tersebut
      * concrete class menggunakan extends keyword
      * concrete class harus mengimplementasi semua abstract method
    - Class yang menge-extend abstract class tetapi tidak mengimplementasi seluruh abstract method nya maka harus dideklarasikan sebagai abstract class juga

Fungsi abstract class:

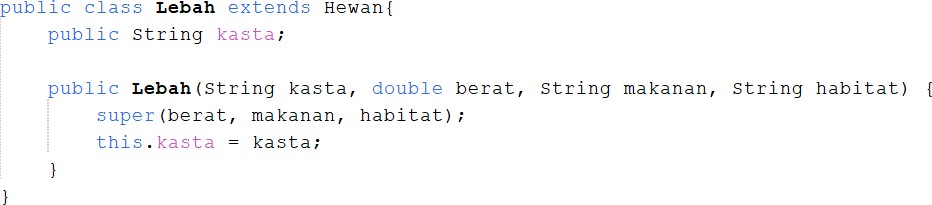
* + - Mencegah suatu class diinstansiasi atau dibuat objeknya
    - Sebagai generalisasi/superclass pada class hierarki
    - Sebagai guideline untuk subclass dengan cara memaksa subclass untuk mengimplementasikan abstract method

# PERCOBAAN

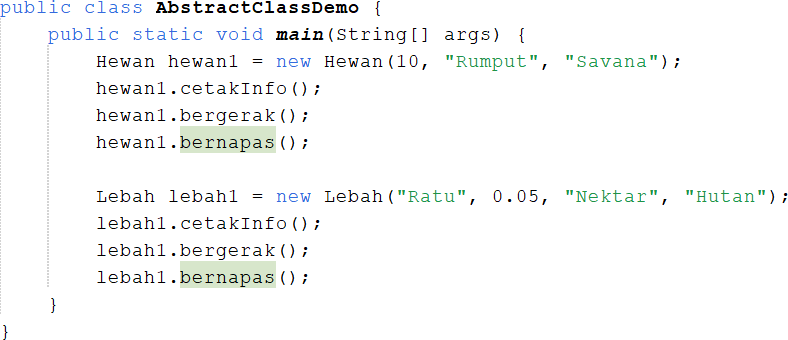
1. **PERCOBAAN 1**
   1. Buatlah project baru dengan nama Praktikum08 kemudian buat class baru dengan nama Hewan. Method bernapas dan bergerak tidak memiliki statement atau baris kode.



* 1. Buat class Lebah sebagai subclass dari class Hewan sebagai berikut

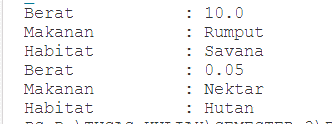


* 1. Buat class main dengan nama AbstractClassDemo lalu instansiasi objek dari class Hewan dan class Lebah. Run program kemudian amati hasilnya.



# PERTANYAAN

* 1. Bagaimana hasil pada langkah 3? Apakah objek hewan1 dan lebah1 berhasil diinstansiasi?



Pada hewan1 keluar seperti biasa sesuai dengan methodnya cetakInfo tetapi pada lebah1 yang muncul hanya berat, makanan dan habitat karena method print mengikuti parent classnya sehingga print kasta “ratu” tidak muncul

* 1. Menurut Anda, mengapa tidak ada baris program pada method bergerak() dan bernapas() pada class Hewan()?

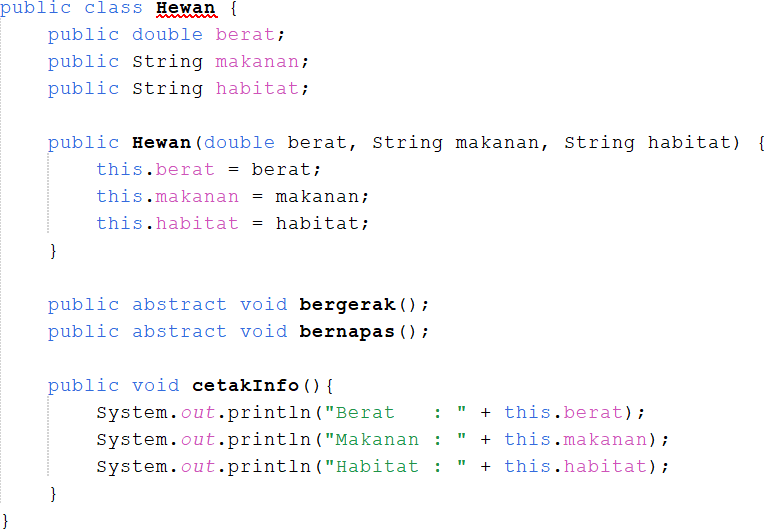
Karena tidak diimplementasikan di parent class methodnya (kosong) sehingga pada main ketika dipanggil tidak muncul apa-apa

* 1. Class Lebah tidak memiliki method bergerak(), bernapas(), dan cetakInfo(), mengapa tidak terjadi error pada AbstractClassDemo?

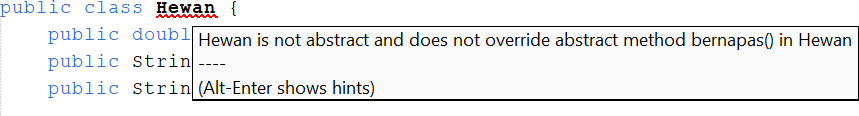
Karena class lebah merupakan child class dari class hewan yang otomatis bisa mengakses method dari parent classnya

# PERCOBAAN 2

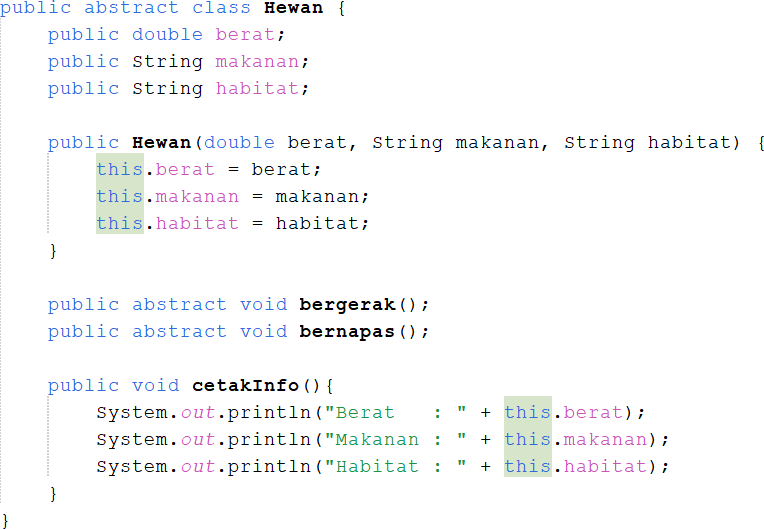
* 1. Ubah method bergerak dan bernapas menjadi abstract method.



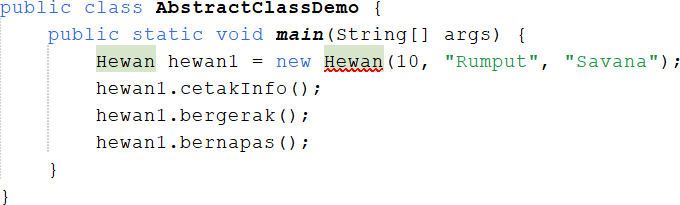
* 1. Akan muncul error sebagai berikut



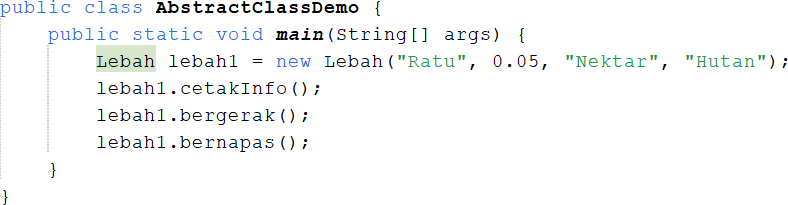
* 1. Ubah class Hewan menjadi abstract Class. Jalankan program kemudian amati hasilnya.



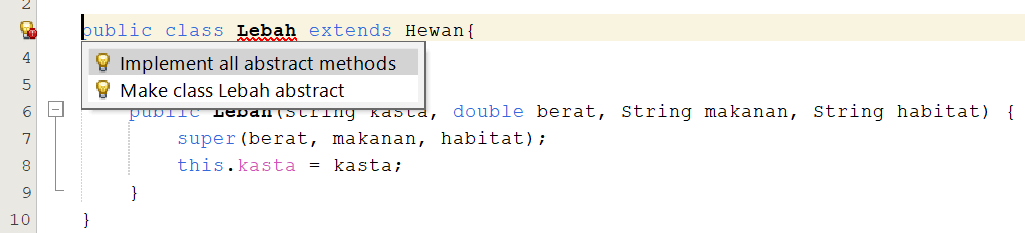
* 1. Ubah class demo sebagai berikut. Run program kemudian amati hasilnya



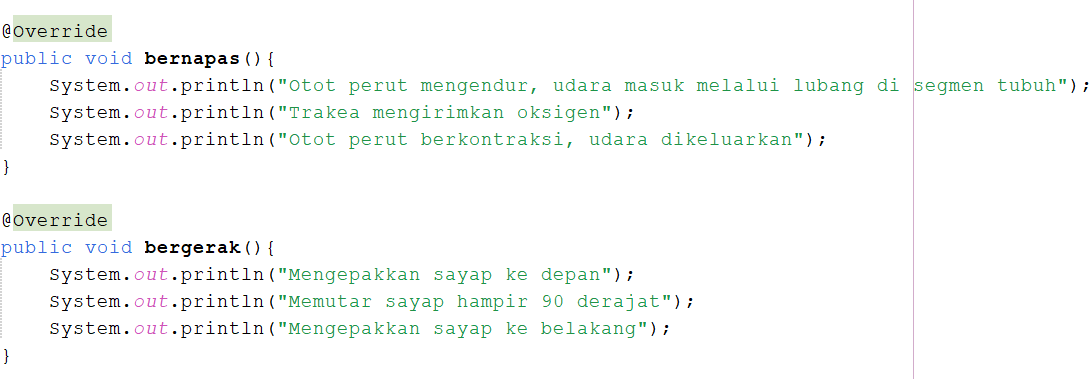
* 1. Ubah class demo sebagai berikut. Run program kemudian amati hasilnya



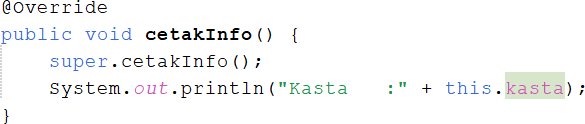
* 1. Klik icon lampu pada class Lebah, kemudian pilih option “Implement all abstract method”



* 1. Implementasi method bergerak dan bernapas pada class Lebah sebagai berikut. Run program kemudian amati hasilnya.

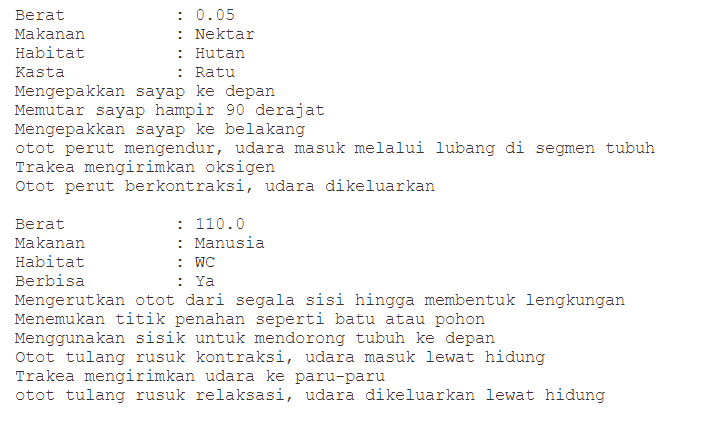


* 1. Tambahkan method cetakInfo() pada class Lebah. Run program kemudian amati hasilnya.



* 1. Buat class Ular kemudian sebagai berikut. Instansiasi objek bertipe Ular pada class AbstractClassDemo. Ekseksusi ketiga method untuk object tersebut.





# PERTANYAAN

* 1. Pada langkah 1, mengapa sebaiknya method bergerak() dan bernapas() dideklarasikan sebagai abstract method?

Karena method tersebut masih belum jelas untuk implementasi dan detailnya, jadi di generalisasikan pada Langkah ke-1

* 1. Mengapa pada langkah 2 muncul error?

Karena pada class hewan terdapat method yang diubah menjadi abstract, jadi classnya harus diubah menjadi abstract juga, sedangkan pada Langkah 2 tidak diubah sehingga eror

* 1. Apakah sebuah class yang memiliki abstract method harus dideklarasikan sebagai abstract class?

Harus, bila terdapat abstract method class harus dideklerasikan juga menjadi abstract, bila tidak akan terjadi eror

* 1. Sebaliknya, apakah abstract class harus memiliki abstract method?

Tidak harus, program tetap berjalan tidak eror meskipun classnya dideklerasikan abstract tapi methodnya tidak

* 1. Mengapa muncul error pada langkah 4?

Karena kelas yang sudah diabstract an tidak bisa dijadikan / instansikan objek

* 1. Apakah abstract class dapat memiliki constructor?

Dapat, tetapi pada program ini implementasikannya pada child class

* 1. Apakah constructor abstract class dapat dipanggil?

Dapat, pada class Lebah dan Ular memanggil consctructor pada Parent classnya (Hewan)

* 1. Pada langkah 6-8, mengapa method bergerak() dan bernapas() **harus** di-override, namun method cetakInfo() **tidak harus** di-override?

Karena pada abstract class method bergerak dan bernapas ikut di abstract juga, berarti child classnya harus override methodnya juga, jika tidak maka akan error

* 1. Simpulkan kegunaan dari abstract method

Kegunaannya untuk menjelaskan apa saja yang bisa dilakukan oleh sebuah class namun tidak ada detail bagaimana cara melakukannya

* 1. Simpulkan kegunaan dari abstract class

Untuk mencegah suatu class diinstansiasi objeknya, dan sebagai generalisasi / superclass pada class hierarki

# 4. TUGAS

Implementasikan class diagram yang telah dirancang pada tugas PBO Teori ke dalam kode program. Selanjutnya buatlah instansiasi objek dari masing-masing subclass kemudian coba eksekusi method-method yang dimiliki.

public abstract class Hero {

    String nama;

    int hp;

    int walkingSpeed;

    public Hero(String nama, int hp, int walkingSpeed){

        this.nama = nama;

        this.hp = hp;

        this.walkingSpeed = walkingSpeed;

    }

    public abstract void menyerang();

    public abstract void bertahan();

    public void displayInfo(){

        System.out.println("Nama Hero\t\t: " + this.nama);

        System.out.println("Jumlah HP\t\t: " + this.hp);

        System.out.println("Kecepatan Berjalan\t: " + this.walkingSpeed);

    }

}

public class Marksman extends Hero{

    String senjata;

    int attackSpeed;

    public Marksman(String nama, int hp, int walkingSpeed, String senjata, int attackSpeed){

        super(nama, hp, walkingSpeed);

        this.senjata = senjata;

        this.attackSpeed = attackSpeed;

    }

    @Override

    public void menyerang(){

        if (this.senjata.equalsIgnoreCase("Sniper")) {

            System.out.println("Hero Marksman ini menyerang menggunakan Sniper yang mempunyai Jarak Range Tembak Jauh "

            + "dan sekali hit memberikan Damage yang sakit, tetapi lemah dalam Attack Speed"

            );

        } else if(this.senjata.equalsIgnoreCase("Panah")){

            System.out.println("Hero Marksman ini menyerang menggunakan Panah yang mempunyai Jarak Range Tembak Dekat "

            + "dan tidak terlalu mempunyai Damage yang sakit, tetapi mempunyai keunggulan Attack Speed");

        }

    }

    @Override

    public void bertahan(){

        System.out.println("Cara Defensif hero ini paling efektif adalah dibelakang Hero Tanker Karena mudah sekali diculik");

    }

    @Override

    public void displayInfo() {

        super.displayInfo();

        System.out.println("Senjata\t\t\t: " + this.senjata);

        System.out.println("Attack Speed\t\t: " + this.attackSpeed);

        menyerang();

        bertahan();

    }

}

public class Fighter extends Hero {

    String senjata;

    int spellVamp;

    String item;

    public Fighter(String nama, int hp, int walkingSpeed, String senjata, int spellVamp, String item){

        super(nama, hp, walkingSpeed);

        this.senjata = senjata;

        this.spellVamp = spellVamp;

        this.item = item;

    }

    @Override

    public void menyerang(){

        if(this.senjata.equalsIgnoreCase("Pedang")){

            System.out.println("Hero Fighter ini menyerang menggunakan Pedang dengan Range Jarak sangat Dekat"

            + " yang mempunyai fleksibilitas lebih cepat");

        } else if(this.senjata.equalsIgnoreCase("Palu Gada")){

            System.out.println("Hero Fighter ini menyerang dengan menggunakan Palu Gada dengan Range Jarak sangat Dekat"

            + " yang mempunyai Damage lebih besar, tetapi mempunyai kekurangan fleksibilitas");

        }

    }

    @Override

    public void bertahan(){

        System.out.println("Hero Fighter mempunyai HP Banyak dan Spell Vamp sehingga lebih tebal dan tidak mudah diculik");

    }

    @Override

    public void displayInfo() {

        super.displayInfo();

        System.out.println("Senjata\t\t\t: " + this.senjata);

        System.out.println("Spell Vamp\t\t: " + this.spellVamp);

        System.out.println("Item\t\t\t: " + this.item);

        menyerang();

        bertahan();

    }

}

public class Main {

    public static void main(String[] args) {

        Marksman marksman1 = new Marksman("Lesley", 500, 20, "Sniper", 30);

        marksman1.displayInfo();

        System.out.println();

        Fighter fighter1 = new Fighter("Lapu-Lapu", 1500, 20, "Pedang", 40, "Blade Of Despair");

        fighter1.displayInfo();

    }

}

