

JOBSHEET W08

ABSTRACT CLASS

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

2. 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 mendefinisikan 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>

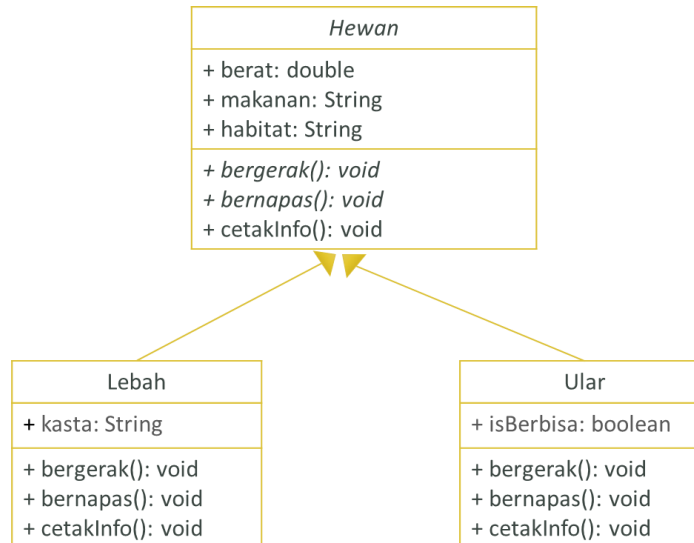
Untuk mendeklarasikan method abstract:

public abstract <return_type> <namaMethod>();

Contoh:

```
public abstract class Hewan {  
    public abstract void bergerak();  
    public abstract void bernapas();  
}
```

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 menggunakan 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 meng-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

3. PERCOBAAN

A. 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.

```
public class Hewan {
    public double berat;
    public String makanan;
    public String habitat;

    public Hewan(double berat, String makanan, String habitat) {
        this.berat = berat;
        this.makanan = makanan;
        this.habitat = habitat;
    }

    public void bergerak() {
        //
    }

    public void bernapas() {
        //
    }

    public void cetakInfo() {
        System.out.println("Berat : " + this.berat);
        System.out.println("Makanan : " + this.makanan);
        System.out.println("Habitat : " + this.habitat);
    }
}
```

2. Buat class Lebah sebagai subclass dari class Hewan sebagai berikut

```
public class Lebah extends Hewan{
    public String kasta;

    public Lebah(String kasta, double berat, String makanan, String habitat) {
        super(berat, makanan, habitat);
        this.kasta = kasta;
    }
}
```

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

```
public class AbstractClassDemo {  
    public static void main(String[] args) {  
        Hewan hewan1 = new Hewan(10, "Rumput", "Savana");  
        hewan1.cetakInfo();  
        hewan1.bergerak();  
        hewan1.bernapas();  
  
        Lebah lebah1 = new Lebah("Ratu", 0.05, "Nektar", "Hutan");  
        lebah1.cetakInfo();  
        lebah1.bergerak();  
        lebah1.bernapas();  
    }  
}
```

B. PERTANYAAN

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

gagal dan menghasilkan error "**Cannot instantiate the type Hewan**". Ini karena `Hewan` adalah **abstract class**, yang berarti Anda tidak bisa langsung membuat objek dari kelas tersebut. Anda hanya bisa menginstansiasi subclass yang mengimplementasikan semua method abstrak (seperti `bergerak()` dan `bernapas()`).

Jadi, untuk `hewan1` akan terjadi error saat kompilasi, dan program tidak akan bisa dijalankan dengan objek `Hewan` langsung. Anda harus menginstansiasi subclass yang mengimplementasikan method `bergerak()` dan `bernapas()` (seperti `Lebah`).

Namun, `lebah1` akan berhasil diinstansiasi karena `Lebah` adalah subclass dari `Hewan` yang mengimplementasikan semua method abstrak yang ada di `Hewan`.

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

Method yang bersifat **abstract** tidak memerlukan implementasi di class **abstract** itu sendiri. Sebaliknya, subclass yang **non-abstract** diharuskan untuk mengimplementasikan method-method tersebut.

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

Ini karena `Lebah` **mewarisi** method `cetakInfo()` dari class `Hewan` dan **diwajibkan** untuk mengimplementasikan `bergerak()` dan `bernapas()` karena kedua method tersebut dideklarasikan sebagai **abstract** di class `Hewan`.

C. PERCOBAAN 2

1. Ubah method bergerak dan bernapas menjadi abstract method.

```
public class Hewan {  
    public double berat;  
    public String makanan;  
    public String habitat;  
  
    public Hewan(double berat, String makanan, String habitat) {  
        this.berat = berat;  
        this.makanan = makanan;  
        this.habitat = habitat;  
    }  
  
    public abstract void bergerak();  
    public abstract void bernapas();  
  
    public void cetakInfo(){  
        System.out.println("Berat    : " + this.berat);  
        System.out.println("Makanan : " + this.makanan);  
        System.out.println("Habitat : " + this.habitat);  
    }  
}
```

2. Akan muncul error sebagai berikut

```
public class Hewan {  
    public double berat;  
    public String makanan;  
    public String habitat;  
}
```

Hewan is not abstract and does not override abstract method bernapas() in Hewan

(Alt-Enter shows hints)

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

```
public abstract class Hewan {  
    public double berat;  
    public String makanan;  
    public String habitat;  
  
    public Hewan(double berat, String makanan, String habitat) {  
        this.berat = berat;  
        this.makanan = makanan;  
        this.habitat = habitat;  
    }  
  
    public abstract void bergerak();  
    public abstract void bernapas();  
  
    public void cetakInfo(){  
        System.out.println("Berat : " + this.berat);  
        System.out.println("Makanan : " + this.makanan);  
        System.out.println("Habitat : " + this.habitat);  
    }  
}
```

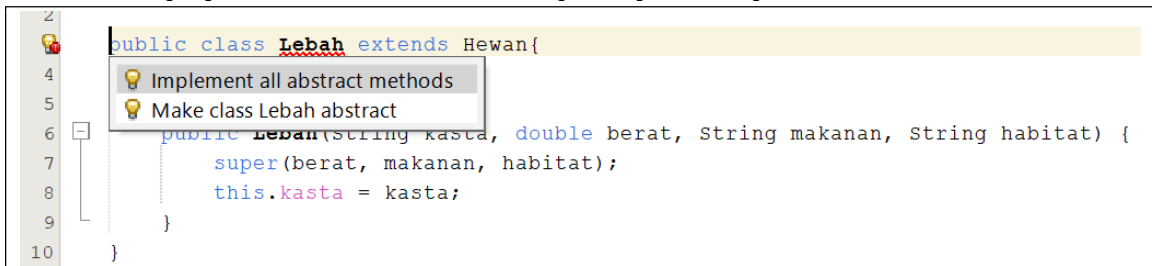
4. Ubah class demo sebagai berikut. Run program kemudian amati hasilnya

```
public class AbstractClassDemo {  
    public static void main(String[] args) {  
        Hewan hewan1 = new Hewan(10, "Rumput", "Savana");  
        hewan1.cetakInfo();  
        hewan1.bergerak();  
        hewan1.bernapas();  
    }  
}
```

5. Ubah class demo sebagai berikut. Run program kemudian amati hasilnya

```
public class AbstractClassDemo {  
    public static void main(String[] args) {  
        Lebah lebah1 = new Lebah("Ratu", 0.05, "Nektar", "Hutan");  
        lebah1.cetakInfo();  
        lebah1.bergerak();  
        lebah1.bernapas();  
    }  
}
```

6. Klik icon lampu pada class Lebah, kemudian pilih option "Implement all abstract method"



```
public class Lebah extends Hewan {  
    public Lebah(String kasta, double berat, String makanan, String habitat) {  
        super(berat, makanan, habitat);  
        this.kasta = kasta;  
    }  
}
```

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

```
@Override  
public void bernapas() {  
    System.out.println("Otot perut mengendur, udara masuk melalui lubang di segmen tubuh");  
    System.out.println("Trakea mengirimkan oksigen");  
    System.out.println("Otot perut berkontraksi, udara dikeluarkan");  
}  
  
@Override  
public void bergerak() {  
    System.out.println("Mengepakkan sayap ke depan");  
    System.out.println("Memutar sayap hampir 90 derajat");  
    System.out.println("Mengepakkan sayap ke belakang");  
}
```

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

```
@Override  
public void cetakInfo() {  
    super.cetakInfo();  
    System.out.println("Kasta : " + this.kasta);  
}
```

9. Buat class Ular kemudian sebagai berikut. Instansiasi objek bertipe Ular pada class AbstractClassDemo. Eksekusi ketiga method untuk object tersebut.

```
public class Ular extends Hewan{
    public boolean isBerbisa;

    public Ular(boolean isBerbisa, double berat, String makanan, String habitat) {
        super(berat, makanan, habitat);
        this.isBerbisa = isBerbisa;
    }

    @Override
    public void bergerak(){
        System.out.println("Mengerutkan otot dari segala sisi hingga membentuk lengkungan");
        System.out.println("Menemukan titik penahan seperti batu atau pohon");
        System.out.println("Menggunakan sisik untuk mendorong tubuh ke depan");
    }

    @Override
    public void bernapas(){
        System.out.println("Otot tulang rusuk kontraksi, udara masuk lewat hidung");
        System.out.println("Trakea mengirimkan udara ke paru-paru");
        System.out.println("Otot tulang rusuk relaksasi, udara dikeluarkan lewat hidung");
    }

    @Override
    public void cetakInfo() {
        super.cetakInfo();
        System.out.println("Berbisa : " + (this.isBerbisa ? "Ya" : "Tidak"));
    }
}
```

D. PERTANYAAN

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

Method bergerak() dan bernapas() sebaiknya dideklarasikan sebagai **abstract** karena perilaku **bergerak** dan **bernapas** dapat berbeda tergantung pada jenis hewan. Dengan mendeklarasikannya sebagai abstract, kita memberi kesempatan kepada setiap subclass dari Hewan (misalnya Lebah, Kucing, Ikan, dll.) untuk menyediakan implementasi spesifik yang sesuai dengan jenis hewan tersebut. Ini menghindari implementasi yang tidak relevan di class induk dan memastikan fleksibilitas untuk implementasi yang berbeda pada setiap subclass.

2. Mengapa pada langkah 2 muncul error?

Pada langkah 2, Anda mencoba untuk menginstansiasi objek dari class Hewan yang merupakan abstract class. Abstract class **tidak dapat diinstansiasi** langsung, karena tidak memiliki implementasi lengkap dari semua method yang dibutuhkan. Dalam hal ini, Hewan mendeklarasikan method bergerak() dan bernapas() sebagai abstract, yang mengharuskan subclass untuk mengimplementasikannya.

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

Ya, sebuah **class** yang memiliki **abstract method** harus dideklarasikan sebagai **abstract class**.

Hal ini karena class tersebut tidak dapat diinstansiasi secara langsung, dan method abstraknya harus diimplementasikan oleh subclass. Class yang memiliki setidaknya satu abstract method tidak dapat menjadi class konkret (class yang bisa diinstansiasi), oleh karena itu perlu dideklarasikan sebagai abstract class.

4. Sebaliknya, apakah abstract class harus memiliki abstract method?

Tidak, **abstract class** tidak selalu harus memiliki abstract method. Meskipun banyak abstract class yang memiliki abstract method, class ini juga bisa memiliki **method konkret** (method yang memiliki implementasi). Abstract class hanya perlu dideklarasikan sebagai abstract jika ada method yang bersifat abstract, tetapi jika seluruh implementasi sudah lengkap, abstract class tersebut tidak perlu memiliki abstract method.

5. Mengapa muncul error pada langkah 4?

Muncul error pada langkah 4 (saat mencoba menginstansiasi objek Hewan) karena **Hewan** adalah abstract class, yang tidak bisa diinstansiasi langsung. Untuk memperbaikinya, Anda harus membuat objek dari subclass konkret yang mengimplementasikan semua method abstract yang ada di class induk.

6. Apakah abstract class dapat memiliki constructor?

Ya, **abstract class** dapat memiliki constructor. Constructor pada abstract class digunakan untuk menginisialisasi atribut yang dimiliki oleh class induk (seperti atribut berat, makanan, dan habitat pada class Hewan). Meskipun abstract class tidak dapat diinstansiasi langsung, constructor dapat dipanggil oleh subclass yang menginstansiasi class tersebut.

7. Apakah constructor abstract class dapat dipanggil?

Constructor pada abstract class **tidak dapat dipanggil langsung** untuk membuat objek dari class tersebut. Namun, constructor tersebut **akan dipanggil secara otomatis** ketika subclass menginstansiasi dirinya sendiri dan memanggil constructor `super()` untuk mewarisi properti dari class induk (abstract class). Jadi, constructor di abstract class digunakan oleh subclass.

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

E. • **Method `bergerak()` dan `bernapas()`** harus di-override karena kedua method tersebut dideklarasikan sebagai **abstract** di class `Hewan`, sehingga setiap subclass yang konkret (seperti `Lebah`) **harus mengimplementasikannya**.

F. • **Method `cetakInfo()`** tidak harus di-override karena method ini sudah memiliki implementasi di class induk `Hewan`. Subclass dapat memilih untuk **menggunakan implementasi tersebut** atau mengubahnya jika diinginkan (overriding). Namun, jika subclass

tidak mengubahnya, maka implementasi di class induk tetap berlaku.

1. Simpulkan kegunaan dari abstract method

Kegunaan dari **abstract method** adalah untuk menyediakan **kerangka dasar** bagi subclass yang lebih spesifik. Dengan mendeklarasikan sebuah method sebagai abstract, class induk tidak menyediakan implementasi konkret, tetapi hanya mendeklarasikan bahwa method tersebut harus ada di subclass. Ini mendorong subclass untuk menyediakan implementasi sesuai dengan kebutuhan spesifik mereka, sehingga menciptakan **polimorfisme** dan fleksibilitas dalam hierarki kelas.

2. Simpulkan kegunaan dari abstract class

kegunaan dari **abstract class** adalah untuk menyediakan **kerangka dasar** dan **struktur umum** yang dapat diwarisi oleh subclass. Abstract class memungkinkan kita untuk mendefinisikan **implementasi yang umum** untuk beberapa method, sementara method yang lebih spesifik dapat didefinisikan oleh subclass. Ini juga memungkinkan kita untuk menghindari duplikasi kode dan memaksa subclass untuk mengimplementasikan method tertentu (jika diperlukan). Abstract class digunakan untuk menciptakan **keseragaman** dan **reusabilitas kode** dalam hierarki kelas.

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.

https://github.com/valinadz/ABSTRACT_JSPB023