# Jobsheet 7

# Interface

## A. Kompetensi

Setelah menyelesaikan lembar kerja ini mahasiswa diharapkan mampu:

1. Menjelaskan maksud dan tujuan penggunaan interface;
2. Menerapkan interface di dalam pembuatan program.

## B. Pendahuluan

*Interface* merupakan sekumpulan abstract method yang saling berkaitan

### 1. Karakteristik:

a. Umumnya terdiri dari abstract method

b. Selalu dideklarasikan dengan menggunakan kata kunci `interface`.

c. Diimplementasikan dengan menggunakan kata kunci `implements`

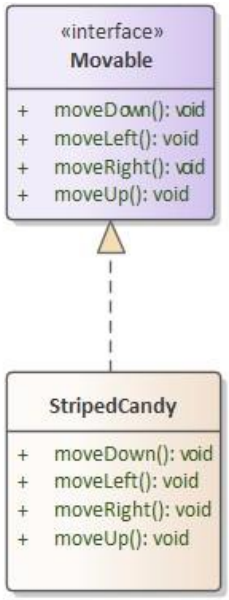
d. Interface tidak dapat diinstansiasi, hanya dapat diinstasiasi melalui class yang mengimplement interface tersebut

### 2. Kegunaan:

Bertindak sebagai **kontrak/syarat** yang berisi **sekumpulan behavior/method** yang saling terkait untuk memenuhi suatu **kapabilitas**. Dengan kata lain, interface memberikan panduan mengenai method apa saja yang perlu diimplementasikan untuk memenuhi kapabilitas tertentu.

### 3. Notasi Class Diagram Interface

- Nama interface **tidak** dicetak miring
- Keterangan <<interface>> di atas nama interface
- Nama method boleh dicetak miring atau tidak
- Implements dilambangkan dengan garis panah putus-putus



## 4. Sintaks Interface

- Untuk mendeklarasikan suatu interface:
  `public interface <NamaInterface>`
- Untuk mengimplementasikan interface:
  `public class <NamaClass> implements <NamaInterface>`
- Nama interface sebaiknya dalam bentuk **adjective/kata sifat** jika merepresentasikan kapabilitas. Dapat juga menggunakan **kata benda**
- Contoh:

```
public interface Movable {
    void moveLeft();
    void moveRight();
    void moveUp();
    void moveDown();
}
```

```
public class PlainCandy extends GameItem implements Movable{
    @Override
    public void moveLeft() {}
    @Override
    public void moveRight() {}
    @Override
    public void moveUp() {}
    @Override
    public void moveDown() {}
}
```

## 5. Implementasi Interface

Bila sebuah class mengimplementasikan suatu interface:

- **Seluruh konstanta** dari interface akan dimiliki oleh class tersebut
- **Seluruh method** pada interface harus diimplementasikan
- Bila class yang meng-implement interface **tidak mengimplementasikan semua method**, maka class tersebut harus dideklarasikan sebagai **abstract class**

## 6. Multiple Interface

- Suatu class dapat meng-implement multiple interface
- Bila suatu class merupakan subclass dan meng-implement interface, maka **keyword extends mendahului implements** → Contoh: `public class PlainCandy extends GameItem implements Crushable, Movable`

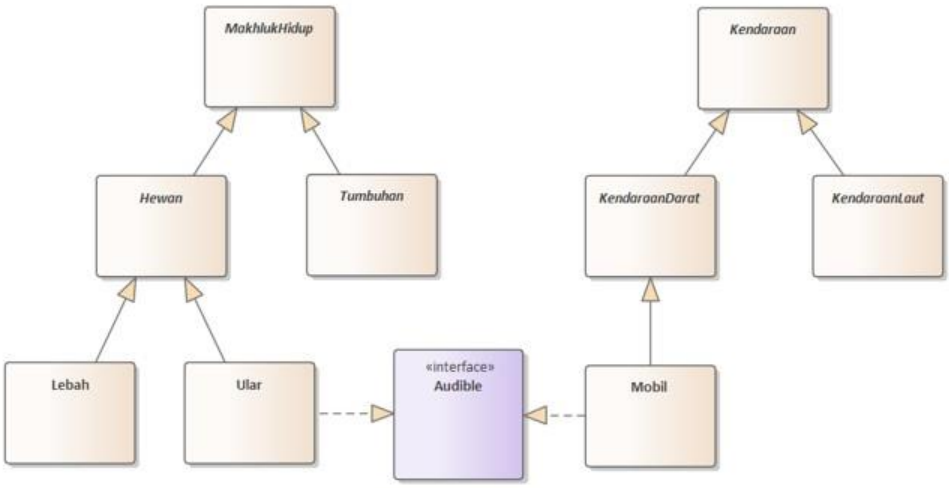
## 7. Perbedaan Abstract Class dan Interface

| Abstract Class | Interface |
|---|---|
| Dapat memiliki **concrete** method atau **abstract** method | Hanya dapat memiliki **abstract** method |
| Level modifier atribut dan method: **public, protected, no-modifier, private** | Level modifier variable dan method hanya **public** (boleh tidak dituliskan) |
| Dapat memiliki **static/non-static, final/non final** variable | Hanya dapat memiliki **static dan final** variable |
| Method boleh bersifat **static/non-static dan final/non final** | Method **tidak** boleh bersifat static dan final |
| Digunakan untuk mendefinisikan hirarki class | Tidak mendefinisikan hirarki class/bukan bagian dari hirarki class |

## 8. Interface tidak terikat pada hirarki

Suatu class di java hanya dapat meng-extend atau menjadi subclass secara langsung dari **satu** superclass saja. Akibatnya class tersebut akan terikat pada suatu hirarki tertentu. Misalnya class Lebah merupakan subclass Hewan sedangkan class Hewan sendiri merupakan subclass MakhlukHidup. Pembatasan 1 parent class secara langsung ini menyebabkan class Lebah terikat pada hirarki makhluk hidup dan tidak bisa terkait dengan hirarki lainnya.

Sementara itu interface tidak terikat pada suatu hirarki. Interface dibuat "secara lepas" tanpa bergantung pada hirarki. Misalkan terdapat interface Audible, interface tersebut dapat diimplementasikan di class apapun dari hirarki manapun. Misal class Ular bisa bersuara, class ini dapat mengimplementasikan interface Audible. Begitu juga dengan class Mobil dari hirarki kendaraan dapat pula mengimplementasikan interface Audible.



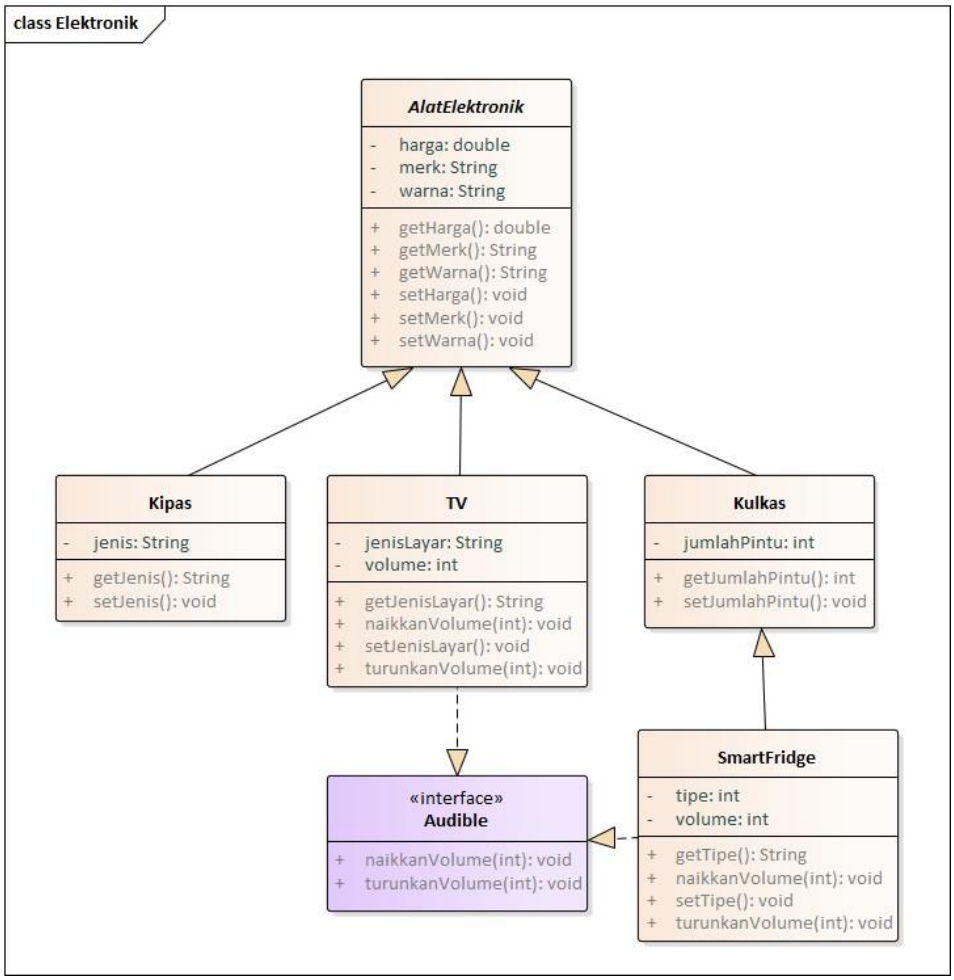
## 9. Penggunaan Abstract Class vs Interface

Abstract class dapat memiliki atribut (instance variable), yaitu suatu variable yang dimiliki oleh objek tertentu. Atribut dan method ini (jika access level modifier-nya sesuai) akan diwariskan terhadap subclass nya. Oleh karena itu, jika suatu class memiliki **common properties (dan method)** maka sebaiknya dibuat abstract class sebagai generalisasi. Misal ada beberapa class `PlainCandy, StripedCandy, RainbowChocoCandy, Wall` dll yang merupakan jenis item dalam game dengan atribut yang sama, misalnya `positionX, positionY, dan iconName`, sebaiknya kita buat abstract class GameItem sebagai generalisasi dari class-class tersebut.

Sementara itu, jika beberapa class memiliki **common behavior** (perilaku atau kapabilitas yang sama) kita bisa menggunakan interface untuk memberikan panduan mengenai method apa saja yang perlu diimplementasikan untuk memenuhi kapabilitas tertentu. Misalnya jika suatu class memiliki kapabilitas untuk dapat berpindah atau `Movable`, seharusnya dia memiliki method `moveLeft(), moveRight(), moveDown, moveUp`. Sekumpulan method dalam interface ini akan menjadi panduan atau pedoman, bahwa jika selanjutnya ada pengembangan atau penambahan game item lain dan item tersebut dapat bergerak juga maka method-method tersebut harus diimplementasikan dalam class nya.

## C. PERCOBAAN

Implementasikan class diagram berikut ke dalam kode program.



1. Buat project baru dengan nama InterfaceLatihan (boleh disesuaikan)
2. Pada sebuah package, buatlah abstract class AlatElektronik

```java
public class AlatElektronik {
    private double harga;
    private String warna;
    private String merk;

    public AlatElektronik(double harga, String warna, String merk) {
        this.harga = harga;
        this.warna = warna;
        this.merk = merk;
    }

    public double getHarga() {
        return harga;
    }

    public void setHarga(double harga) {
        this.harga = harga;
    }

    public String getWarna() {
        return warna;
    }

    public void setWarna(String warna) {
        this.warna = warna;
    }

    public String getMerk() {
        return merk;
    }

    public void setMerk(String merk) {
        this.merk = merk;
    }
```

3. Selanjutnya buatlah subclass dari AlatElektronik, yaitu Kipas, TV, dan Kulkas sebagai berikut.

```java
public class Kipas extends AlatElektronik{
    private String jenis;

    public Kipas (String jenis, double harga, String warna, String merk) {
        super(harga, warna, merk);
        this.jenis = jenis;
    }

    public String getJenis() {
        return jenis;
    }

    public void setJenis() {
        this.jenis = jenis;
    }
}
```

```java
public class TV extends AlatElektronik {
    private String jenisLayar;
    private int volume;

    public TV(String jenisLayar, int volume, double harga, String warna, String merk) {
        super(harga, warna, merk);
        this.jenisLayar = jenisLayar;
        this.volume = volume;
    }

    public String getJenisLayar() {
        return jenisLayar;
    }

    public void setJenisLayar(String jenisLayar) {
        this.jenisLayar = jenisLayar;
    }
}
```

```java
public class Kulkas extends AlatElektronik {
    private int jumlahPintu;

    public Kulkas(int jumlahPintu, double harga, String warna, String merk) {
        super(harga, warna, merk);
        this.jumlahPintu = jumlahPintu;
    }

    public void setJumlahPintu(int jumlahPintu) {
        this.jumlahPintu = jumlahPintu;
    }

    public int getJumlahPintu() {
        return jumlahPintu;
    }
}
```

4. Buatlah class SmartFridge yang merupakan sublass dari class Kulkas

```java
public class SmartFridge extends Kulkas{
    private int volume;

    public SmartFridge(int volume, int jumlahPintu, double harga, String warna, String merk) {
        super(jumlahPintu, harga, warna, merk);
        this.volume = volume;
    }
}
```

5. Beberapa dari alat elektronik dapat mengeluarkan suara. Kapabilitas ini kita buat ke dalam kode program dengan interface Audible dengan method naikkanVolume() dan turunkanVolume() sebagai berikut

```java
public interface Audible {
    void naikkanVolume(int increment);
    void turunkanVolume(int decrement);
}
```

6. Ubah class TV untuk meng-impelement interface Audible

```java
public class TV extends AlatElektronik implements Audible{
    private String jenisLayar;
    private int volume;
```

7. Implementasi abstract method pada interface Audible pada class TV

```java
public class TV extends AlatElektronik implements Audible{
    private String jenisLayar;
    private int volume;

    public String getJenisLayar() {
        return jenisLayar;
    }

    public void setJenisLayar(String jenisLayar) {
        this.jenisLayar = jenisLayar;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }

    public TV(String jenisLayar, int volume, double harga, String warna, String merk) {
        super(harga, warna, merk);
        this.jenisLayar = jenisLayar;
        this.volume = volume;
    }

    @Override
    public void naikkanVolume(int increment) {
        volume += increment;
    }

    @Override
    public void turunkanVolume(int decrement) {
        volume += decrement;
    }
}
```

8. Lakukan hal yang sama pada class SmartFridge

```java
public class SmartFridge extends Kulkas implements Audible{
    private int volume;

    public SmartFridge(int volume, int jumlahPintu, double harga, String warna, String merk) {
        super(jumlahPintu, harga, warna, merk);
        this.volume = volume;
    }

    @Override
    public void naikkanVolume(int increment) {
        volume += increment;
    }

    @Override
    public void turunkanVolume(int decrement) {
        volume += decrement;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }
}
```

## D. PERTANYAAN 2

1. Mengapa terjadi error pada langkah 5?

   Karena kelas TV yang dibuat harus mengimplementasikan metode naikkanVolume(int) yang berasal dari interface Audible.

2. Mengapa Audible tidak dapat dibuat sebagai class?

   Interface seperti Audible digunakan untuk mendefinisikan kontrak yang harus diikuti oleh kelas-kelas lain. Jika Audible diubah menjadi kelas, maka TV harus mewarisi kelas Audible, yang akan membatasi fleksibilitas.

3. Jika access level modifier interface Audible tidak dituliskan, apa access level modifier defaultnya?

   Default access level-nya adalah public jika interface tersebut berada di dalam package yang sama, atau package-private (tanpa modifier) jika berada di luar package.

4. Access level modifier method-method dalam interface Audible tidak dituliskan, apa access level modifier sebenarnya?

   Dalam java, semua method dalam interface secara default adalah public

5. Method naikkanVolume() dan turunkanVolume() memilki implementasi yang sama pada TV dan SmartFridge(), mengapa tidak langsung diimplementasikan pada interface Audible()?

   Dikarenakan ada kode implementasi yang perlu di-share antar kelas tetapi membutuhkan akses ke data kelas tertentu

6. Method naikkanVolume() dan turunkanVolume() memilki implementasi yang sama pada TV dan SmartFridge(), mengapa tidak langsung diimplementasikan pada class AlatElektronik?

   Dikarenakan tidak semua jenis AlatElektronik memerlukan volume

7. Apakah method naikkanVolume() dan turunkanVolume() pada class TV dan SmartFridge() dapat memiliki implementasi yang berbeda?

   Ya, metode naikkanVolume() dan turunkanVolume() pada kelas TV dan SmartFridge() dapat memiliki implementasi yang berbeda.

8. Semua yang Audible seharusnya memiliki nilai volume, mengapa atribut volume tidak dideklarasikan dalam interface Audible()?

   Atribut volume tidak dideklarasikan dalam interface Audible karena interface hanya mendefinisikan perilaku yang diharapkan, bukan data atau atribut.

9. Ubah implementasi method naikkanVolume() dan turunkanVolume() pada class TV sebagai berikut:

```
@Override
public void naikkanVolume() {
    System.out.println("Klik tombol volume sisi atas");
}

@Override
public void turunkanVolume() {
    System.out.println("Klik tombol volume sisi bawah");
}
```

   Compile dan run program. Apakah terjadi error? Mengapa?

   Kelas TV mencoba mengimplementasikan metode naikkanVolume() yang tidak sesuai dengan metode yang ada di **interface** Audible

10. Kembalikan method naikkanVolume() dan turunkanVolume() pada class TV seperti semula

11. Apa fungsi dari interface?

    Untuk mendefinisikan kontrak atau perilaku yang harus diikuti oleh kelas yang mengimplementasikannya.

12. Buat method getInfo() untuk setiap class. Instansiasi objek dari setiap concrete class pada main class, kemudian tampilkan infonya.

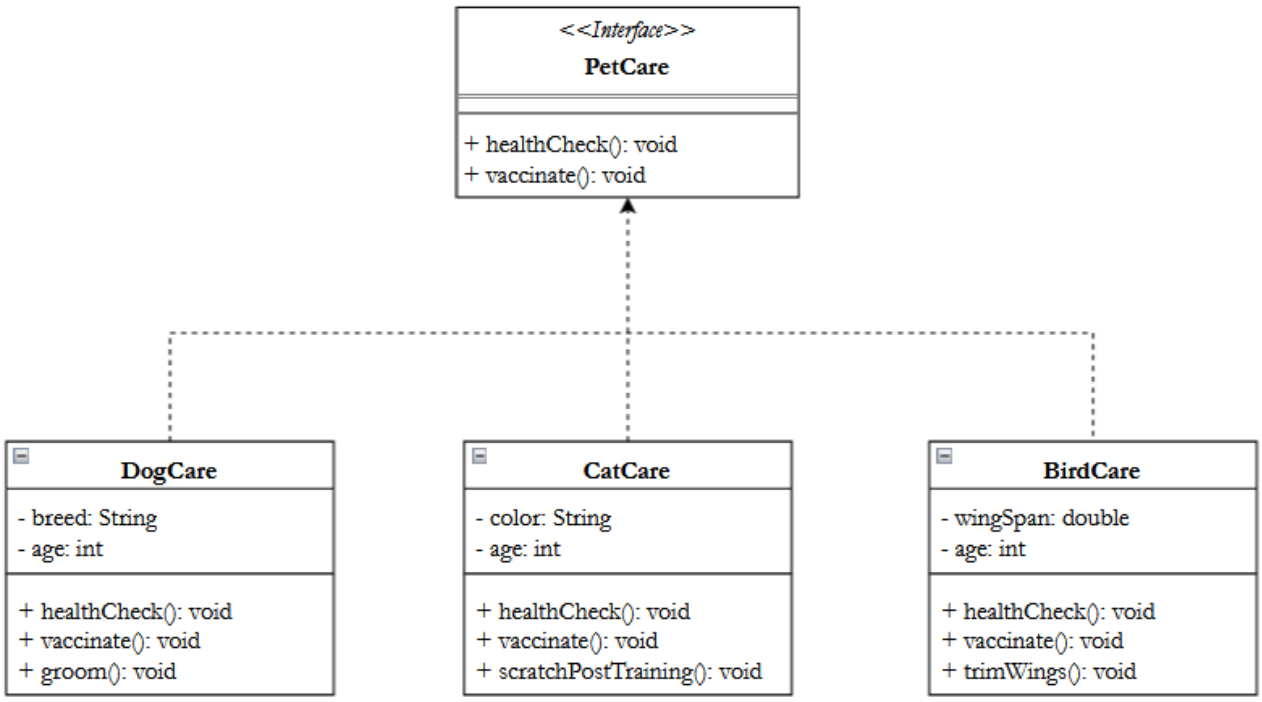
```java
public class MainElektronik {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        Kipas kipas = new Kipas(jenis:"Kipas dinding", harga:500000, warna:"Putih", merk:"Maspion");
        Kulkas kulkas = new Kulkas(jumlahPintu:2, harga:3000000, warna:"Hitam", merk:"Sharp");
        TV tv = new TV(jenisLayar:"LED", volume:10, harga:5000000, warna:"Hitam", merk:"Samsung");
        SmartFridge smartfridge = new SmartFridge(volume:100, jumlahPintu:2, harga:9999000, warna:"Silver", merk:"LG");

        System.out.println(kipas.getInfo());
        System.out.println(kulkas.getInfo());
        System.out.println(tv.getInfo());
        System.out.println(smartfridge.getInfo());
    }
}
```

```
Kipas [Jenis: Kipas dinding, Harga: 500000.0, Warna: Putih, Merk: Maspion]
SmartFridge [Harga: 3000000.0, Warna: Hitam, Merk: Sharp]
TV [Jenis Layar: LED, Volume: 10, Harga: 5000000.0, Warna: Hitam, Merk: Samsung]
SmartFridge [Volume: 100, Harga: 9999000.0, Warna: Silver, Merk: LG]
```

## E. TUGAS

Implementasikan class diagram yang dibuat pada tugas PBO ke dalam kode program.




Interface PetCare

```java
public interface PetCare {
    void healthCheck();
    void vaccinate();
}
```

Class DogCare

```java
public class DogCare implements PetCare {
    private String breed;
    private int age;

    public DogCare(String breed, int age) {
        this.breed = breed;
        this.age = age;
    }

    @Override
    public void healthCheck() {
        System.out.println(x:"Performing health check for the dog");
        System.out.println(x:".....");
        System.out.println(x:"Health check has been done :)");
        System.out.println();
    }

    @Override
    public void vaccinate() {
        System.out.println(x:"Vaccinating the dog");
        System.out.println(x:".....");
        System.out.println(x:"Vaccination has been done :)");
        System.out.println();
    }

    public void groom() {
        System.out.println(x:"Grooming the dog");
        System.out.println(x:".....");
        System.out.println(x:"Grooming has been done :)");
        System.out.println();
    }
}
```

Class CatCare

```java
public class CatCare implements PetCare{
    private String color;
    private int age;

    public CatCare(String color, int age) {
        this.color = color;
        this.age = age;
    }

    @Override
    public void healthCheck() {
        System.out.println(x:"Performing health check for the cat");
        System.out.println(x:".....");
        System.out.println(x:"Health check has been done :)");
        System.out.println();
    }

    @Override
    public void vaccinate() {
        System.out.println(x:"Vaccinating the cat");
        System.out.println(x:".....");
        System.out.println(x:"Vaccination has been done :)");
        System.out.println();
    }

    public void scratchPostTraining() {
        System.out.println(x:"Training the cat to use the scratch post");
        System.out.println(x:".....");
        System.out.println(x:"Training has been done :)");
        System.out.println();
    }
}
```

Class BirdCare

```java
public class BirdCare implements PetCare {
    private double wingSpan;
    private int age;

    public BirdCare(double wingSpan, int age) {
        this.wingSpan = wingSpan;
        this.age = age;
    }

    @Override
    public void healthCheck() {
        System.out.println(x:"Performing health check for the bird");
        System.out.println(x:".....");
        System.out.println(x:"Health check has been done :)");
        System.out.println();
    }

    @Override
    public void vaccinate() {
        System.out.println(x:"Vaccinating the bird");
        System.out.println(x:".....");
        System.out.println(x:"Vaccination has been done :)");
        System.out.println();
    }

    public void trimWings() {
        System.out.println(x:"Trimming the bird's wings");
        System.out.println(x:".....");
        System.out.println(x:"Bird's wings have been trimmed :)");
        System.out.println();
    }
}
```

Class Main

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        DogCare dog = new DogCare(breed:"Siberian Husky", age:5);
        dog.healthCheck();
        dog.vaccinate();
        dog.groom();

        CatCare cat = new CatCare(color:"Black", age:3);
        cat.healthCheck();
        cat.vaccinate();
        cat.scratchPostTraining();

        BirdCare bird = new BirdCare(wingSpan:0.5, age:2);
        bird.healthCheck();
        bird.vaccinate();
        bird.trimWings();
    }
}
```