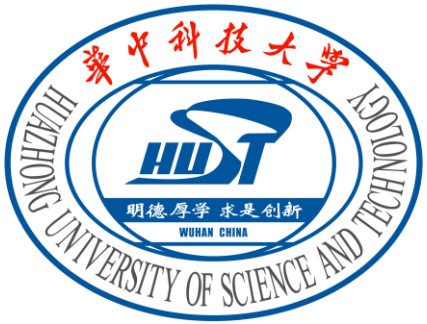
基于Java的面向对象程序设计

陈维亚

weiya_chen@hust.edu.cn

华中科技大学软件学院

第9讲：接口



目录

1. 概念

❑ 接口 interface

软件项目通常由团队完成，各成员独立完成不同子模块后再进行组合

为使组合顺利进行，开发成员之间需达成某种“协议”，我们称之为接口。

网上购物系统

商品管理、用户界面、售后客服、订单管理等；

汽车控制系统

汽车底层控制（启动、停止、转向等），汽车导航传感等；

1. 概念

❑ 接口的定义

```
[public] interface 接口名 [extends 其他的接口名列表] {
    [public static final] 数据类型 常量名 = 值;
    [public abstract] 返回类型 方法名(形参列表);
}
```

```
public interface GroupedInterface extends Interface1,
Interface2, Interface3 {
  // constant declarations
  double E = 2.718282;

  // method signatures
  void doSomething (int i, double x);
  int doSomethingElse(String s);
}
```

接口是一种不同于类(class)的引用类型，是抽象方法的集合。

1. 概念

❑ 接口的实现

```
[访问修饰符] class 类名 [extends 超类名] implements 接口名列表 {
        类体
}
```

```
public class SomeClass implements GroupedInterface {
  …
  // method definitions
  void doSomething (int i, double x){
    …
  }

  int doSomethingElse(String s){
    …
  }
}
```

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，除非声明为抽象类。

1. 概念

❑ 接口的特性

```
public interface Alarmable {
   int duration = 10;
   public void callAlarm();
   public void stopAlarm();
}
```

- 接口是隐式抽象的，当声明一个接口的时候，不必使用**abstract**关键字

- 接口中可以含有变量，变量会被隐式的指定为 **public static final**（用 private 修饰会报编译错误）

- 接口中的方法会被隐式的指定为 **public abstract**

接口练习

【练习1】如下接口存在什么问题？请改正之。

```
public interface SomethingIsWrong {
  void aMethod(int aValue){
    System.out.println("Hi Mom");
  }
}
```

接口练习

【练习2】如下接口可用吗？

```
public interface Marker {
}
```

接口练习

【练习3】以下是接口I的定义，哪些代码可以编译通过？

```
public interface I {
  void setValue(int val);
  int getValue();
}
```

```
class A extends I {
  int value;
  void setValue(int val){
    value = val;
  }
  int getValue(){
    return value;
  }
}
```

```
interface B extends I {
  void increment();
}
```

```
abstract class C implements I {
  int getValue(){
    return 0;
  }

  abstract void increment();
}
```

```
interface D implements I {
  void increment();
}
```

```
class E implements I {
  int value;
  public void setValue(int val){
    value = val;
  }
}
```

1. 概念

❑ 接口的功能

```
abstract class Door {
  public void open();
  public void close();
}
```

```
abstract class Car {
  public void run();
  public void stop();
}
```

```
abstract class Alarm {
  public void callAlarm();
  public void stopAlarm();
}
```

若我们需要一种带有报警功能的门，该怎么办？

```
public class AlarmDoor extends Door {
  …
  public void startAlarm(){}
}
```

同样的功能却有不同的函数名称!

若我们需要一种带有报警功能的车，该怎么办？

```
public class AlarmCar extends Car {
  …
  public void alarming(){}
}
```

1. 概念

❑ 接口的功能

```
public abstract class Door {
  public void open();
  public void close();
}
```

```
public interface Alarmable {
  public void callAlarm();
  public void stopAlarm();
}
```

```
public class AlarmDoor extends Door implements Alarmable {
  …
  public void callAlarm(){}

  public void stopAlarm(){}
}
```

1. 接口提供了一组功能的命名集合

一个类通过实现接口的方式，从而获得接口所声明的抽象方法。

1. 概念

❑ 接口的功能

```
public class Projectile {
  private int damage;
  public void collide(Zombie z){
    z.receiveDamage(this.damage);
  }
}
```

2. 接口定义了不同类交互的标准

日常生活中同样如此，两个实体之间进行连接的部分被称为接口。

```
public interface Damageable{
  public void receiveDamage(int d);
}
```

```
public abstract class Zombie implements Damageable {
  public void receiveDamage(int d){
    …
  }
}
```

1. 概念

❑ 接口的功能

```
public interface Alarmable {
  public void callAlarm();
  public void stopAlarm();
}
```

1. 接口提供了一组功能的命名集合

2. 接口定义了不同类交互的标准

实现了“多继承”

分离了方法的声明和方法的实现

1. 概念

❑ 接口 v.s. 类

- **接口与类相似点：**

  接口文件保存在 .java 结尾的文件中，文件名使用接口名。

  接口的字节码文件保存在 .class 结尾的文件中。

- **接口与类的区别：**

  接口不能用于实例化对象。

  接口没有构造方法。

  接口中所有的方法必须是抽象方法。

  接口只能包含static final成员变量。

  接口不是被类继承了，而是要被类实现。

  接口支持多继承。

1. 概念

❑ 接口 v.s. 抽象类 - 多态的重要实现者

1、 抽象层次不同

抽象类是对类抽象，而接口是对行为的抽象。抽象类是对整个类整体进行抽象，包括属性、行为，但是接口却是对类局部（行为）进行抽象。

2、 跨域不同

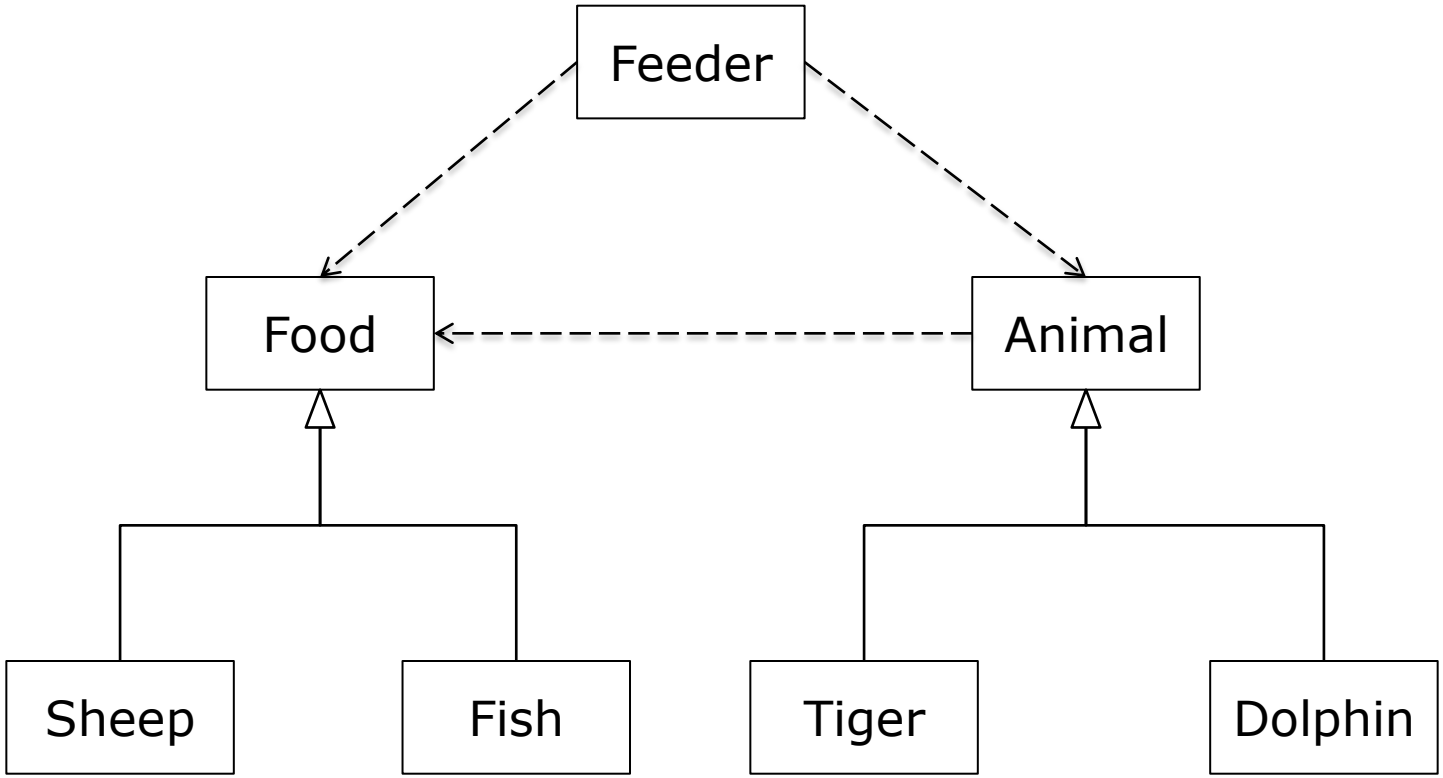
抽象类所体现的是一种继承关系，父类和派生类之间必须存在"is-a" 关系，即父类和派生类在概念本质上应该是相同的。对于接口并不要求接口的实现者和接口定义在概念本质上是一致的， 仅仅是实现了接口定义的契约而已。

3、 设计层次不同

抽象类是自底向上抽象而来的，接口是自顶向下设计出来的。

接口练习

【练习4】动物饲养员建立了如下的关系模型，请指出其中的不合理之处。



接口练习

【练习5】设计一个接口Relatable，使得实现了它的几何形状具有和同类形状比较面积大小的功能。

```
public class Rectangle {
  public int width = 0;
  public int height = 0;
  public Point origin;

  public Rectangle(Point p, int w, int h) {
    origin = p;
    width = w;
    height = h;
  }
  public int getArea() {
    return width * height;
  }
}
```

1. 概念

❑ 接口名作为类型

定义新接口 = 定义新的引用数据类型

```
public interface Foodable {}
```

```
public class Fish implements Foodable {}
```

以接口名为声明类型的变量，可以被实现了该接口的任一类的对象赋值

```
Foodable food = new Fish();
```

可以把使用实现了某一接口的类创建的对象的引用赋给该接口声明的接口变量，那么该接口变量就可以调用被类实现的接口的方法。

1. 概念

- 接口名作为类型

```
public interface Relatable {
  public int isLargerThan(Relatable other);
}
```

```
public Object findLargest(Object object1, Object object2) {
  // if(object1 instanceof Relatable)
  Relatable obj1 = (Relatable)object1;
  Relatable obj2 = (Relatable)object2;
  if ((obj1).isLargerThan(obj2) > 0)
    return object1;
  else
    return object2;
}
```

❑ 接口继承

```
public interface DoIt {
  void doSomething(int i, double x);
  int doSomethingElse(String s);
}
```

若希望修改该接口，增加一个新方法，则接口会变成

```
public interface DoIt {
  void doSomething(int i, double x);
  int doSomethingElse(String s);
  boolean didItWork(int I, double x, String s);
}
```

若不希望修改接口，则可借助继承

```
public interface DoItPlus extends DoIt {
  boolean didItWork(int I, double x, String s);
}
```

❑ 默认方法

接口中也可以放入方法的实现，避免修改实现该接口的类的代码。

```
public interface DoIt {
  void doSomething(int i, double x);
  int doSomethingElse(String s);
  default boolean didItWork(int i, double x, String s) {
    // Method body
  }
}
```

❑ 静态方法

接口中也可以放入静态方法，以增加一些帮助性质的工具函数。

```
public interface DoIt {
   void doSomething(int i, double x);
   int doSomethingElse(String s);
   static int toolfunc(int x) {
     // Method body
   }
}
```

3. 应用

❑ 标准库中的接口

接口提供了一组功能的命名集合
接口定义了不同类交互的标准

java.lang.Comparable, 可以通过Collections.sort或Arrays.sort进行自动排序

```
public interface Comparable<T> {
  public int compareTo(T o);
}
```

```
public class Person implements Comparable<Person> {
  int age;

  public int compareTo(Person p) {
    return this.age-p.getAge();
  }
}
```

java.io.Serializable, 可以启用其序列化功能

3. 应用

❑ 构建程序构架

接口提供了一组功能的命名集合
接口定义了不同类交互的标准

设计连接和关闭数据库时，由于数据库不同，相对应的实现方式也会有所差异。

我们在设计用户系统时，可以设计一个接口，包含openDB()和closeDB()，无具体实现。

```
public interface DB {
  Connection openDB(String url, String user, String pw);
  void closeDB();
}
```

```
public class mysqlDB implements DB {
  Connection openDB(String url, String user, String pw){}
  void closeDB(){}
}
```

接口练习

【练习6】假设你写了一个时间服务器，可以周期性地把当前日期和时间报给客户端。编写一个客户端需实现的接口，使其可以与服务器正确通信。

```
public interface TimeClient { }
```

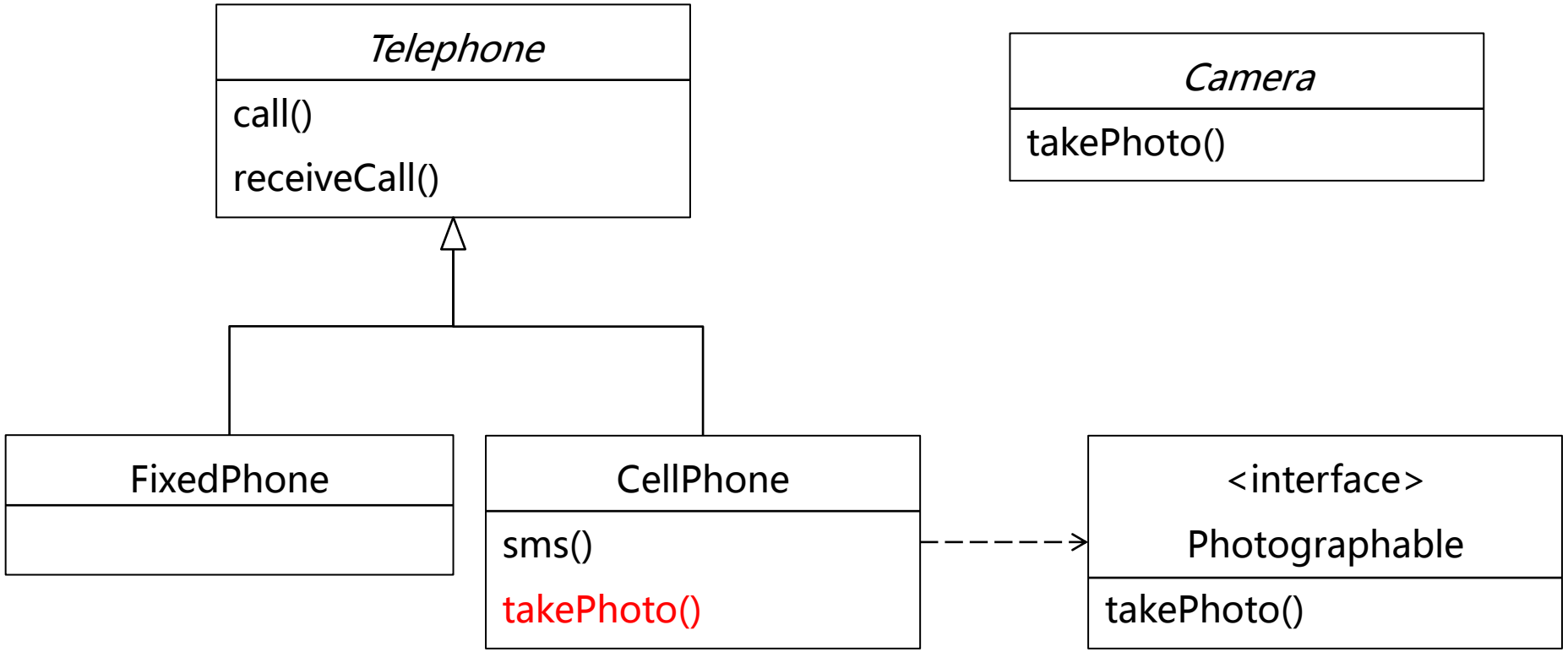
```
public class SimpleTimeClient implements TimeClient {}
```

```
public interface TimeClient {
  void setTime(int hour, int minute, int second);
  void setDate(int day, int month, int year);
  void setDateAndTime(int day, int month, int year,
int hour, int minute, int second);
}
```

3. 应用

- 实现多继承

接口提供了一组功能的命名集合
接口定义了不同类交互的标准



4. 章节小结

PS:

ISP（Interface Segregation Principle）：面向对象的一个核心原则。

它表明使用多个专门的接口比使用单一的总接口要好。

一个类对另外一个类的依赖性应当是建立在最小的接口上的。

一个接口代表一个角色，不应当将不同的角色都交给一个接口。没有关系的接口合并在一起，形成一个臃肿的大接口，这是对角色和接口的污染。

接口（interface）

- 概念
- Java的新特性
- 应用举例

接口所代表的是“like-a”的关系

只有对问题域的本质有良好的理解，才能做出正确、合理的设计。

下节预告

多态