计算机学院

北華航天工業學院
NORTH CHINA INSTITUTE OF AEROSPACE ENGINEERING

第7章 抽象类、接口、包和常用API

2022-2023-1

回顾上堂课的内容

- 方法重载与方法重写
- this关键字与super关键字
- final关键字
- 继承
- 多态

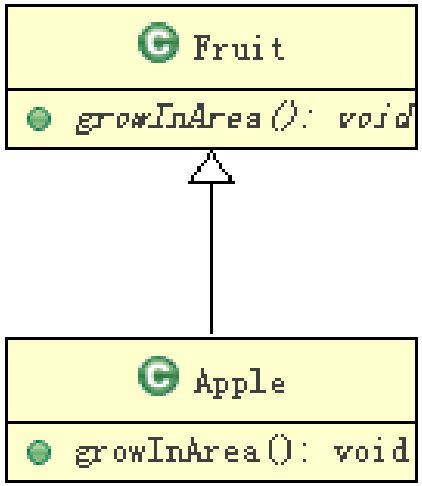
本章目标

- **抽象类**
  - 抽象类的概念、特征、作用及意义
- **接口**
  - 接口的概念、特征、作用及意义
- **包**
  - 包的作用、语法格式以及如何引入包
- **常用API**
  - **包装类**
  - String、StringBuffer、StringBuilder类

抽象类

- 描述：苹果是一种水果

水果实体在现实中并不存在，也就是不能创建Fruit对象。



用**继承**来模拟苹果是一种水果

抽象类

◆**抽象类概念：**

- 用abstract修饰的类就是抽象类。抽象类中可以有用abstract修饰的抽象方法，也可以没有抽象方法。

◆**语法：**

```
abstract class abstractClass{//抽象类
   …
   abstract returnType abstractMethod([paramlist])
   //抽象方法
}
```

抽象方法

- 在了解抽象类之前，先来了解一下抽象方法。抽象方法是一种特殊的方法：它只有声明，而没有具体的实现。抽象方法的声明格式为：

- ```
  abstract void fun();
  ```
  ```

为什么要设计抽象类

- 在某些情况下，某个父类只是知道其子类应该包含怎样的方法，但无法准确知道这些子类应该如何实现这些方法，于是就定义一些抽象方法，抽象方法是没有方法体的，没有具体的实现，必须要由子类来重写这个方法。例如有个Animal类，它有一个方法eat()，但是不同的动物进食的方式是不同的，所以Animal无法准确知道其子类怎么进食。 那有人会有疑问了，如果父类不知道怎么实现，就不要管这个方法了，让子类自己去扩展好了。但这样会存在一个问题，如果变量的引用类型是父类Animal，实际指向的是子类Dog的实例，那这个类没法调用eat()方法了。

抽象类

- 用abstract关键字来修饰一个类时，这个类叫做抽象类；用abstract来修饰一个方法时，该方法叫做抽象方法。
- 含有抽象方法的类必须被声明为抽象类，抽象类必须被继承，抽象方法必须被重写。
- 抽象类不能被实例化。
- 抽象方法只需声明，而不需实现。

```
abstract class Animal {
    private String name;
    Animal(String name) {
        this.name = name;
    }
    public abstract void enjoy();
}

class Cat extends Animal {
    private String eyesColor;
    Cat(String n,String c) {
        super(n); eyesColor = c;
    }
    public void enjoy() {
        System.out.println
            ("猫叫声.....");
    }
}
```

抽象类

- **抽象类特征：**
  - 抽象类不能被实例化，也就是不能用new关键字去产生对象（它的引用变量可以使用任何的派生类对象。但这个引用只能使用抽象类中声明的方法。不能使用派生类自定义的方法）。
  - 抽象方法只需声明，而不需实现（无方法体）。
  - 抽象类被子类继承，子类必须实现抽象方法。
  - 包含抽象方法的类肯定是抽象类，但抽象类中不一定包含抽象方法。

abstract不能和final、static、private修饰符共存来修饰方法

- abstract修饰的方法是要被子类重写的，如果一个方法是final的，那这个方法就不能被重写，所以abstract和final不能共存。
- 如果一个方法是static修饰的，那么就可以通过类名来直接调用，但是抽象的方法是没有方法体的，这样就会引发错误，所以abstract和static不能共存。
- 如果父类的方法是private修饰的，那么这个方法是不能被子类访问的，更不谈重写了，所以abstract和private不能共存。

抽象类和普通类的主要有三点区别：

1）抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。

2）抽象类不能用来创建对象；

3）如果一个类继承于一个抽象类，则子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。

抽象类

◆抽象类的作用及意义

- 作用：被子类继承

抽象类是从多个具有相同特征的类中抽象出来的一个类，这个抽象类作为父类被继承，其实是子类的一个模板，避免了子类设计的随意性。

抽象类是模板设计的一种体现，子类在抽象类的基础上进行扩展。

- **意义：**

1. 抽象类的抽象方法约定了多个子类共用的方法声明。
2. 不同的子类可以有不同的抽象方法实现，体现了oo多态性
3. 可以使子类都在同一个规范下使用


```
public abstract  class Fruit{
    public  abstract  void growInArea();
}

public  class  Apple extends  Fruit{
   public void growInArea() {
                    System.out.println("南北方都可以种植我。");
   }
}

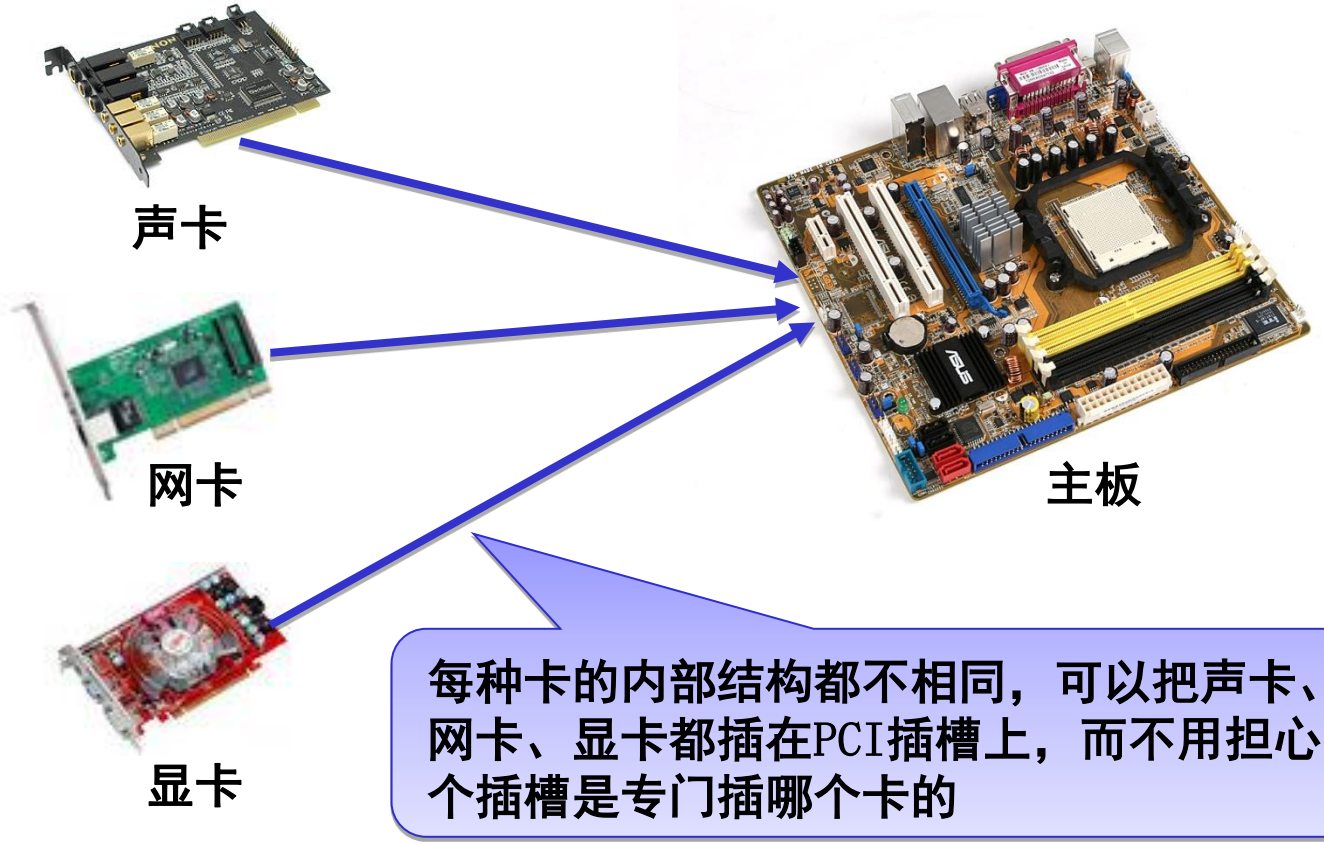
public  class  Pineapple extends  Fruit{
   public void growInArea() {
                    System.out.println("我喜欢温暖，只能在南方看
到我。");
   }
}
```

Lab7-1、实现输出三角形、长方形的面积计算。

提示：两者都要有两个数据计算面积。可以将这种共通行为提出到抽象类中。

生活中的接口

- 电脑主板上的PCI插槽的规范就类似于Java接口



接口(interface)

- 一个Java接口是一些方法特征的集合，但没有方法的实现

```
public interface PCI {
    void start();
    void stop();
}
```

这是Java接口，相当于主板上的PCI插槽的规范

- Java接口中定义的方法在不同的地方被实现，可以具有完全不同的行为

```
class SoundCard implements PCI {
   public void start()  {
      System.out.println("Du du...");
   }
   public void stop()  {
      System.out.println("Sound stop!");
   }
}
```

```
class NetworkCard implements PCI {
   public void start()  {
      System.out.println("Send...");
   }
   public void stop()  {
      System.out.println("Network stop!");
   }
}
```

声卡、网卡都实现了PCI插槽的规范，但行为完全不同

接口(interface)

◆接口概念：

- 如果一个抽象类中的所有方法都是抽象的，我们就可以将这个类用另外一种方式来定义，也就是接口定义。接口是抽象方法和常量值的定义的集合，从本质上讲，**接口是一种特殊的抽象类**，这种抽象类中只**包含常量和方法的定义**，而**没有变量和方法的实现**。
- 语法：

```
[public|default] interface interfaceName[extends listOfSuperInterface] {
    type NAME=value; //该常量被实现该接口的多个类共享;
    具有public ,final, static的属性。
    returnType methodName([paramlist]);  //(具有 public和abstract属性)
}
```

extends 子句与类声明的extends子句基本相同，不同的是一个接口可有多个父接口，用逗号隔开，而一个类只能有一个父类。

```
public interface Runner {
    public static final int id = 1;

    public void start();
    public void run();
    public void stop();
}
```

为使用引入接口

- 为了从一个类调用父类的方法，编译时两个类都要声明，以便java编译器检查方法是否兼容。不可避免的会使类的层次变得越来越强，越来越多的子类可以使用利用这种机制。接口可以解决这种问题，把方法的定义和层次分开。不相关的类可以实现同一个接口

接口(interface)

- **接口特征：**
  - 接口是只包含常量和抽象方法的特殊抽象类：
    常量默认的修饰符是public static final
    抽象方法默认的修饰符是 public abstract
  - 接口可以多继承，即一个接口可以通过extends关键字继承一个或者多个接口；
  - 子类通过implements实现一个或者多个接口
  - 一个类可以继承一个父类的同时，实现一个或多个接口，extends关键字必须位于implements关键字之前 。
  - 多个无关的类可以实现同一个接口；
  - 一个类可以实现多个无关的接口；
  - 与继承关系类似，接口与实现类之间存在多态性。

接口(interface)

◆接口的作用及意义

●作用

1. 用来解决多继承的问题。

2. 接口是用来规范类的，它可以避免类在设计上的不一致，这在多人合作的开发中犹为重要。

通过接口实现不相关类的相同行为,而无需考虑这些类之间的关系。

通过接口指明多个类需要实现的方法。

通过接口了解对象的交互界面,而无需了解对象所对应的类

●意义：

1. 体现了oo多态性

2. 可以使子类都在同一个规范下使用

接口(interface)

```
interface Singer {
    public void sing();
    public void sleep();
}
class Student implements Singer {
    private String name;
    Student(String name) {
        this.name = name;
    }
    public void study(){
        System.out.println("studying");
    }
    public String getName(){return name;}
    public void sing() {
        System.out.println("student is singing");
    }
    public void sleep() {
        System.out.println("student is sleeping");
    }
}
```

接口(interface)

```
interface Singer {
    public void sing();
    public void sleep();
}
interface Painter {
    public void paint();
    public void eat();
}

class Student implements Singer {
    private String name;
    Student(String name) {this.name = name;}
    public void study(){System.out.println("studying");}
    public String getName(){return name;}
    public void sing() {System.out.println
                        ("student is singing");}
    public void sleep() {System.out.println
                        ("student is sleeping");}
}
```

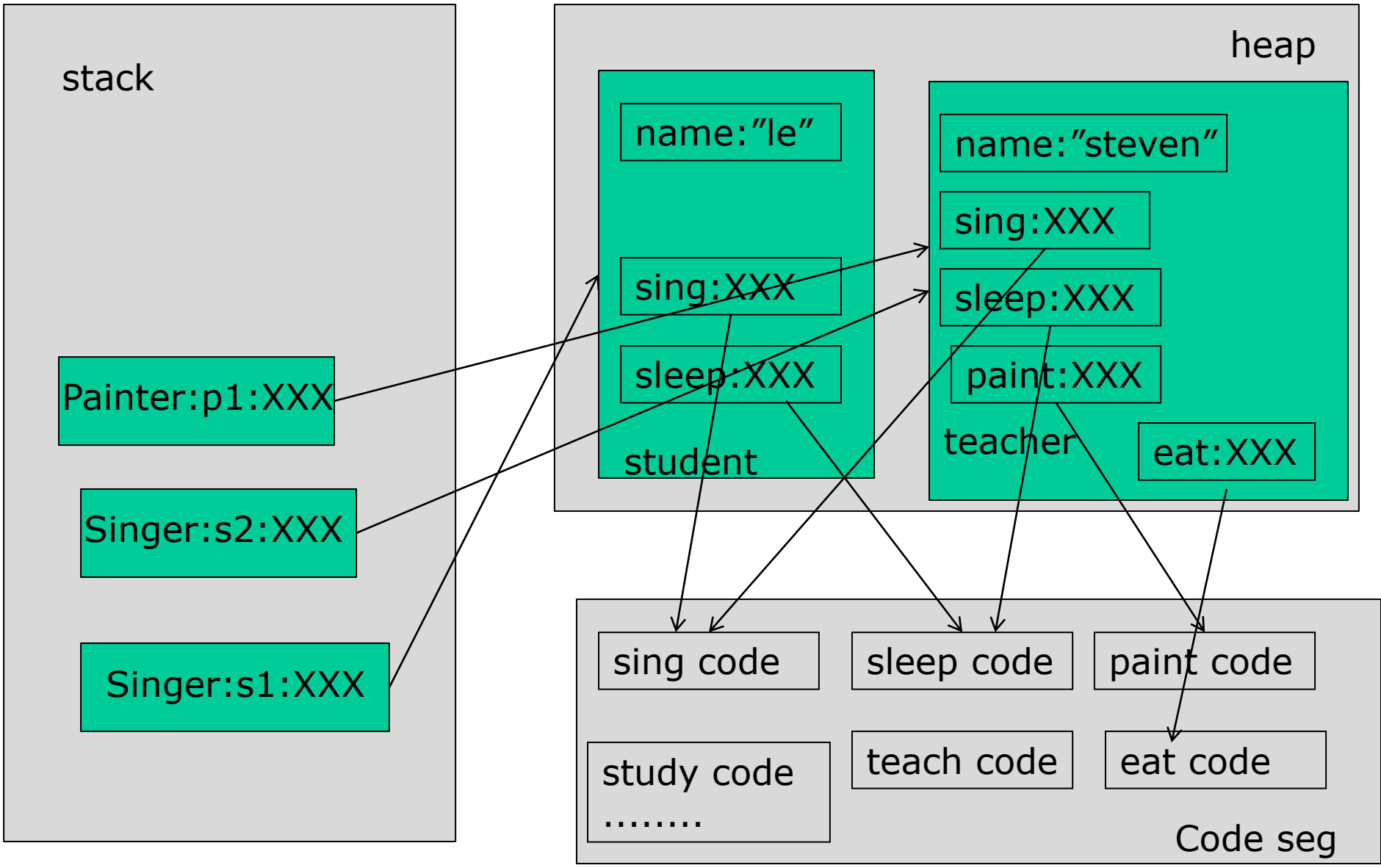
接口(interface)

```
class Teacher implements Singer,Painter {
    private String name;
    public String getString() {
        return name;
    }
    Teacher(String name){this.name = name;}
    public void teach(){System.out.println("teaching");}
    public void sing(){System.out.println
                    ("teacher is singing");}
    public void sleep(){System.out.println
                    ("teacher is sleeping");}
    public void paint(){System.out.println
                    ("teacher is painting");}
    public void eat(){System.out.println
                    ("teacher is eating");}
}
```

接口(interface)

```
public class Test {
    public static void main(String args[]){
        Singer s1 = new Student("le");
        s1.sing(); s1.sleep();
        Singer s2 = new Teacher("steven");
        s2.sing(); s2.sleep();
        Painter p1 = (Painter)s2;
        p1.paint(); p1.eat();
    }
}
```

```
student is singing
student is sleeping
teacher is singing
teacher is sleeping
teacher is painting
teacher is eating
```


stack
heap
name:"le"
sing:XXX
sleep:XXX
student
name:"steven"
sing:XXX
sleep:XXX
paint:XXX
teacher
eat:XXX
Painter:p1:XXX
Singer:s2:XXX
Singer:s1:XXX
sing code
sleep code
paint code
study code
.....
teach code
eat code
Code seg

Lab7-2、使用抽象类、接口完成一个饲养员给动物喂食物的例子。

步骤一：定义Animal接口，包含方法eat(Food food)；

步骤二：定义类Dog和Cat让其实现接口Animal；

步骤三：定义抽象类Food，包含属性name及setName()、getName（）方法

步骤四：定义实现类Fish及Boner实现Food类

步骤五：定义Feeder类，包含方法feed(Animal animal, Food food)

步骤六：定义TestFeeder类（该类测试饲养员给动物喂食物）

接口和抽象类的区别

- **相同点**
  - 不能实例化；
  - 包含未实现的方法声明；
  - 派生类必须实现未实现的方法；
- **不同点**
  - 继承性；

    抽象类与子类之间采用单继承机制

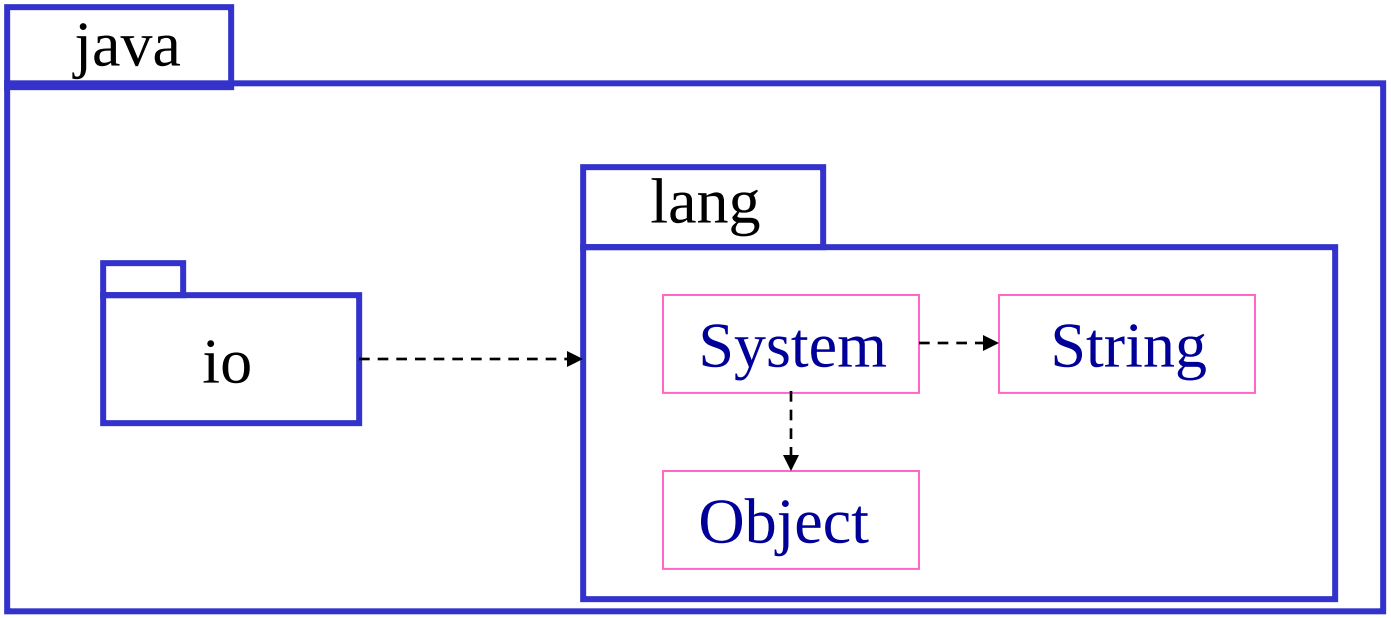
    一个类实现多个接口则可以实现多重继承
  - 成员变量；

    抽象类中可以声明成员变量，子类可以对该成员变量赋值，接口只能是常量。

接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

包(package)

- 作用：
  - 为便于管理大型软件系统中数目众多的类，解决类的命名冲突问题，Java引入包(package)机制，提供类的多重类命名空间。



包(package)

- 包是相关的类型的组，提供访问控制，命名空间管理。注意类型是类，接口等。

包(package)

◆语法：

package语句作为Java源文件的第一条语句，指明该文件中定义的类所在的包。(若缺省该语句，则指定为无名包)。它的格式为：

- package pkg1[.pkg2[.pkg3…]];

举例：Test.java

```
package package1;

public class Test {
    public void display() {
        System.out.println("in  method    display()");
    }
}
```

- Java编译器把包对应于文件系统的目录管理，package语句中，用‘.’ 来指明包(目录)的层次

例如，定义一个类Circ，将其放入com.wgh包中的代码如下：

- package com.wgh;
- public class Circ {
- final float PI=3.14159f;　　//定义一个用于表示圆周率的常量PI
- // 定义一个绘图的方法
- public void draw(){
- System.out.println("画一个圆形！");
- }
- }
- 说明：在Java中提供的包，相当于系统中的文件夹。例如，上面代码中的Circ类如果保存到C盘根目录下，那么它的实际路径应该为C:\com \wgh\Circ.java。

包(package)

◆导入包

为使用定义在不同包中的Java类，需用import语句来引入所需要的类。

- 语法格式：

```
import package1[.package2…]. (classname |*);
```

- 应用举例：

例如，引入com.wgh包中的Circ类的代码如下：

```
import com.wgh.Circ;

  public class TestPackage{
    public static void main(String args[]){
              Circ t = new Circ();
              t.draw();
    }
  }
```

- 包的出现是分层的，例如，java API包含java.awt包，java.awt.color包，java.awt.font包，还有很多其他的包以java.awt开头。但是，java.awt.color包，java.awt.font包,和其他java.awt.xxxx包不包含在java.awt包中。前缀java.awt(java抽象窗口工具)用来让一些相关的包，相关关系更加明显，但不是显示包含。

- 导入java.awt.*，导入java.awt包全部的类型，但是它不导入java.awt.color,java.awt.font,或者其他java.awt.xxx包的任何类。如果你打算像java.awt导入java.awt.color所有的类型，必须在源码文件导入两个包:

- import java.awt.*;import java.awt.color.*;

Java中提供的包主要有以下3种用途：

1）将功能相近的类放在同一个包中，可以方便查找与使用。

2）由于在不同包中可以存在同名类，所以使用包在一定程度上可以避免命名冲突。

3）在Java中，某次访问权限是以包为单位的。

Java包

- 见API中的介绍

包装类

- 基本数据类型只是纯粹的数据，除了自身数据信息之外，没有其他信息。包装类使他们可以以对象的形式存在，携带更多的信息并提供相对应的操作方法。

  - 基本数据类型：

    long、int、short、double、float、boolean、byte、char

  - 包装类：

    Long、Integer、Short、Double、Float、Boolean、Byte、Character。

包装类

◆常用包装类的操作方法：

- （除了Character之外，所有的包装类都有两个构造方法）带有基本值参数并创建包装类对象的构造函数.如可以利用Integer包装类创建对象,Integer obj=new Integer(145);
- 带有字符串参数并创建包装类对象的构造函数.如new Integer(“-45.36”);
- 生成字符串表示法的toString()方法,如obj.toString().
- 对同一个类的两个对象进行比较的equals()方法,如
  obj1.eauqls(obj2);
  obj.compareTo(obj1)
- 转换相应类型

  intValue(); doubleValue();byteValue();
  floatValue(); shortValue(); longValue();

```
public class Test{
      public static void main(String[] args) {
          boolean b1 = true;
          Boolean blobj = new Boolean(b1);//通过构造器把b1基本类型变量包装成包
 装类对象
          int it =5;
          Integer itobj = new Integer(it);//通过构造器把it基本类型变量包装成包
 装类对象
          Float f1 = new Float("4.56");//把一个字符串转换成Float 对象
          Boolean bobj = new Boolean("false");//把一个字符串转换成Boolean 对象
          boolean bb = bobj.booleanValue();//取出Boolean对象里的boolean变量
          int i = itobj.intValue();//取出Integer 对象里的int变量
          float f = f1.floatValue();//取出Float 对象里的float变量
        }
}
```

上面的程序分别可以把true,5等基本类型变量包装成包装类对象。

包装类

- 转换成相应的基本类型

  parseInt(String s); parseFloat(String s); parseByte(String s);

  parseShort(String s); parseLong(String s); parseDouble(String s);

- 创建一个包装类对象

  Integer.valueOf(int);
  Long.valueOf(long);Short.valueOf(short)

  Byte.valueOf(byte);
  Float.valueOf(float);Double.valueOf(double);

  Integer.valueOf(String); Long.valueOf(String);

  Short.valueOf(String) Byte.valueOf(String);
  Float.valueOf(String); Double.valueOf(String);

```
public class Primitive2String{
    public static void main(String[] args){
        String intStr = "123";
        int it = Integer.parseInt(intStr);//把一个特定字符串转换成int变量
        System.out.println("it result is:"+it);
        String floatStr = "4.56";
        float ft = Float.parseFloat(floatStr);//把一个特定字符串转换成float 变量
        System.out.println(ft);
        Integer i1 = Integer.valueOf(25);//创建一个包装类的对象，new Integer返回的永远是不同的对象，但是当整数范围在-128<i<=127时，Integer.valueOf返回的是同一个对象。
        Integer i2 = Integer.valueOf(25);
        if(i1==i2){
            System.out.println("i1 = i2");
        }else{
            System.out.println("i1 != i2");
        }
    }
}
```

包装类

◆自动装/拆箱

jdk1.5以后，提供了一个很有用的特征：自动的“装箱”和“拆箱”就在在int--integer之间，double和Double之间等，进行相互转换的时候，可以不用显示地写出代码，java会自动替我们完成转换。例如：

- 自动将匹配的基本类型包装成对象

  如Integer i=1;//装箱
- 自动将对象转化为基本类型数据

  如int j=i;//拆箱
- 便于批量转换数据和对象，提高开发效率
- 自动装箱和自动拆箱时必须注意类型匹配

```
public class AutoBoxingUnboxing
{
        public static void main(String[] args)
        {
                 //直接把一个基本类型变量赋给Integer对象
                Integer inobj = 5;
                 //直接把一个boolean类型变量赋给一个object类型的变量
                  Object boolobj = true;
                 //直接把一个Integer对象赋给int类型的变量
                int it = inobj;
               if(boolobj instanceof Boolean)
               {
            //先把object对象强制类型转换为Boolean类型，再赋给boolean变量
                        boolean b = (Boolean)boolobj;
                       System.out.println(b);
                }
          }
}
```

结果为：true

分析一下代码的输出结果：

```
Integer i=new Integer(0);
Integer j=new Integer(0);
System.out.println(i==j);
System.out.println(i.equals(j));
false  true
Integer i=0;
Integer j=0;
System.out.println(i==j);
System.out.println(i.equals(j));   true true
```

总结

- 抽象类的特征
- 接口的特征
- 包的概念与作用

谢谢！