

Java面向对象编程5

- What?Why?How?



接口 interface

- 我们前面用继承关系，描述了动物、哺乳动物、爬行动物的各种关系。
- 现在我们要描述：
 - 飞机 导弹 子弹 篮球 石头的关系？



为什么使用接口

- 要求实现防盗门的功能
 - 门有“开”和“关”的功能，锁有“上锁”和“开锁”的功能
 - 将门和锁分别定义为抽象类
 - 将门定义为抽象类，锁定义为接口
 - 防盗门继承门，实现锁的接口

防盗门可以继承门的同时又继承锁吗？

如何解决这个问题呢？



什么是接口

- 认识一下接口

```
public interface MyInterface {  
    public void foo();  
    //其他方法  
}
```

所有方法都是：
public abstract

- 必须知道的接口特性

- 接口不可以被实例化
- 实现类必须实现接口的所有方法
- 实现类可以实现多个接口
- 接口中的变量都是静态常量

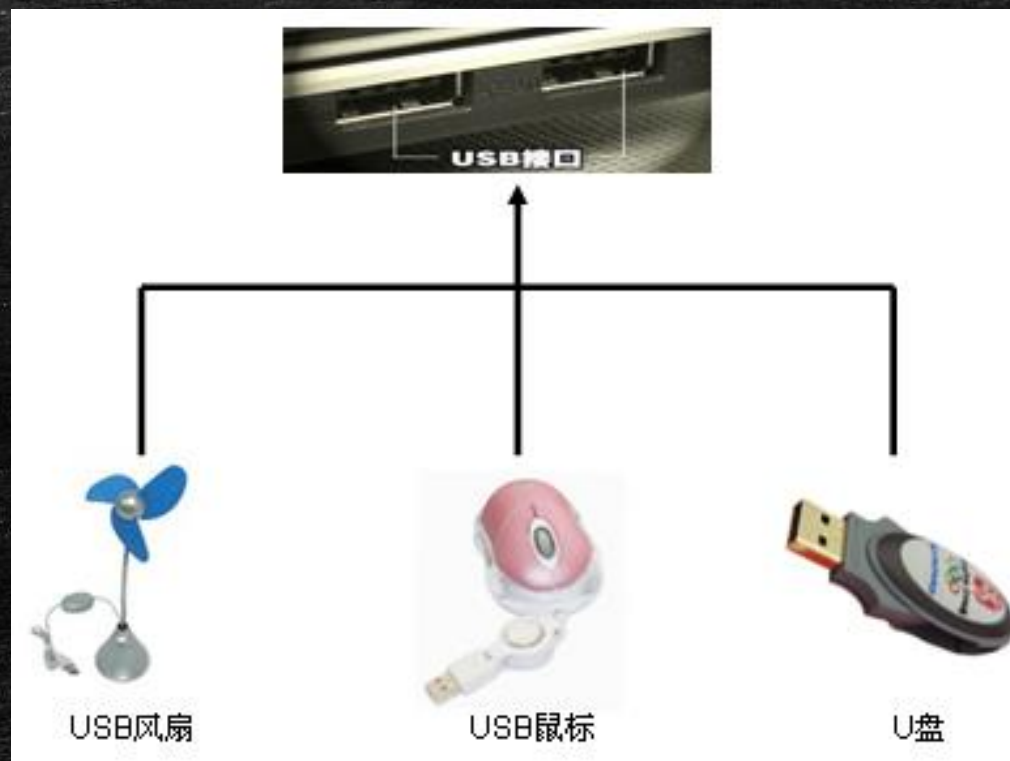
常作为类型使用

Java中的多继承



如何使用接口

- 用程序描述USB接口



如何使用接口

USB接口本身没有实现任何功能

USB接口规定了数据传输的要求

USB接口可以被多种USB设备实现

- 可以使用Java接口来实现

编写USB接口

→ 根据需求设计方法

实现USB接口

→ 实现所有方法

使用USB接口

→ 用多态的方式使用



如何使用接口

· 编码实现

编写
接口

```
public interface UsbInterface {  
    /**  
     * USB接口提供服务。  
     */  
    void service();  
}
```

实现接口使用的
关键字

多个接口使用 “,” 分隔

实现
接口

```
public class UDisk implements UsbInterface {  
    public void service() {  
        System.out.println("连接USB口，开始传输数据。");  
    }  
}
```

使用
接口

```
UsbInterface uDisk = new UDisk();  
uDisk.service();
```

用接口实现多态



接口表示一种能力

- “做这项工作需要一个钳工（木匠/程序员）”

钳工是一种“能力”，不关心具体是谁

- 接口是一种能力

体现在接口的方法上

- 面向接口编程

程序
设计时

关心实现类有何能力，而不关心实现细节

面向接口的约定而不考虑接口的具体实现



面向接口编程3-1

▪ 实现防盗门功能

轻轻拉门，门关上了。
插进钥匙，向左旋转钥匙三圈，锁上了，拔出钥匙。
插进钥匙，向右旋转钥匙三圈，锁打开了，拔出钥匙。
用力推，门打开了。

🔍 分析

- 防盗门是一个门
- 防盗门有一个锁
 - 上锁
 - 开锁

能力

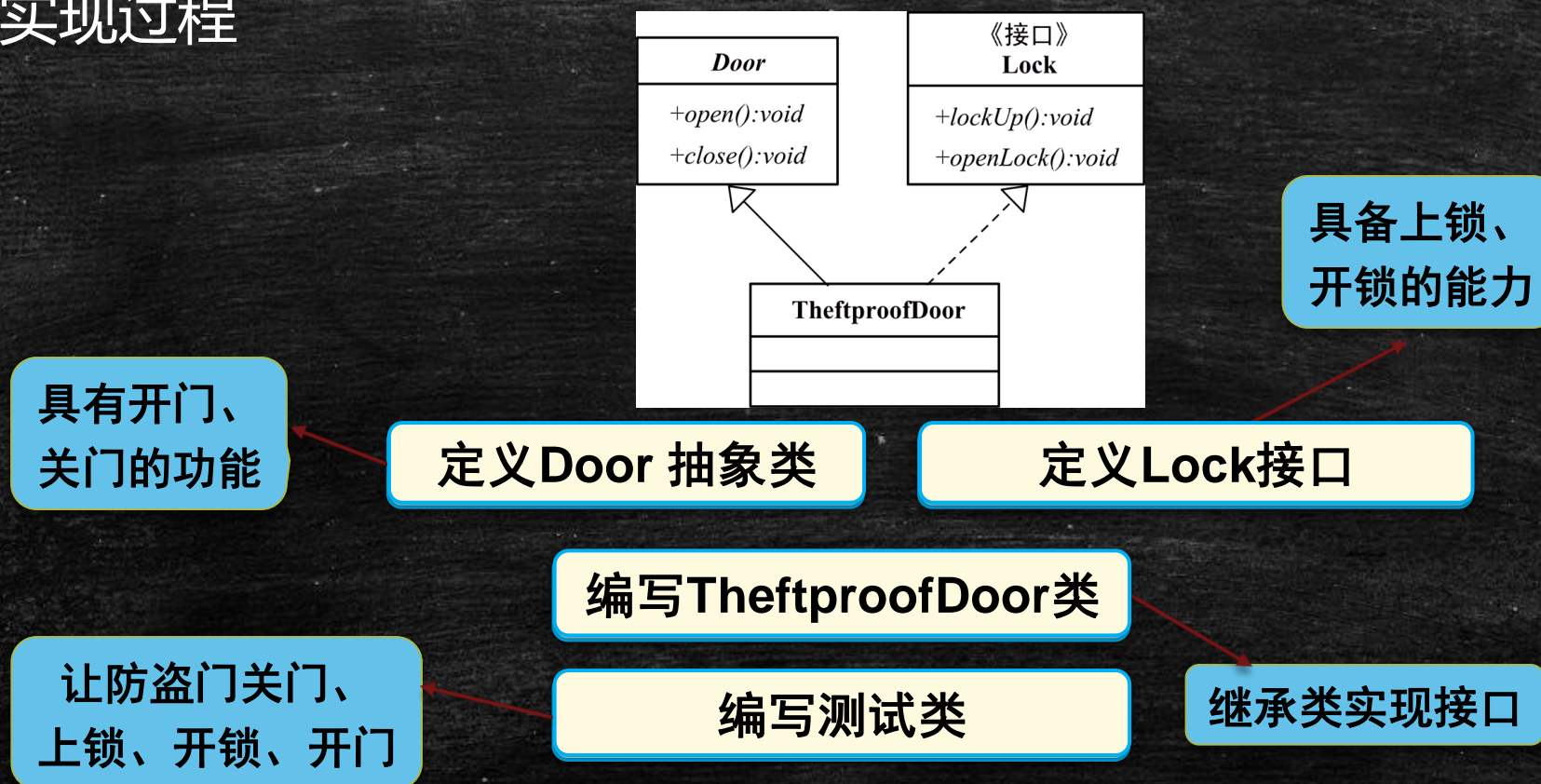
is a的关系

has a的关系



面向接口编程3-2

实现过程



面向接口编程3-3

一个人可以具有多项能力
一个类可以实现多个接口

- 扩展防盗门门铃功能，主要是实现拍照存档

轻轻拉门，门关上了。

插进钥匙，向左旋转钥匙三圈，锁上了，拔出钥匙。

铃.....咔嚓.....照片已存储

插进钥匙，向右旋转钥匙三圈，锁打开了，拔出钥匙。

用力推，门打开了。



上机练习5--使用接口实现防盗门功能

- 需求说明：
 - 使用面向接口编程实现防盗门的功能。
 - 开门、关门
 - 上锁、开锁
 - 拍照存档

定义Door抽象类
定义Lock、DoorBell接口

定义TheftproofDoor类

编写测试类

轻轻拉门，门关上了。
插进钥匙，向左旋转钥匙三圈，锁上了，拔出钥匙。
铃.....咔嚓.....照片已存储
插进钥匙，向右旋转钥匙三圈，锁打开了，拔出钥匙。
用力推，门打开了。



上机练习6--使用接口实现手机功能2-1

- 训练要点:

- 接口的基础知识。
- 接口表示一种能力。

- 需求说明:

- 原始的手机，可以发短信，通电话。随着发展，手机增加了功能：音频、视频播放、拍照、上网。

这是一款型号为G502c的索尼爱立信手机：

开始播放音乐《热雪》.....

发送文字信息.....

开始语音通话.....

这是一款型号为HTC的I9100手机：

已经启动移动网络.....

开始播放视频《小时代》.....

咔嚓.....拍照成功

发送带图片与文字的信息.....

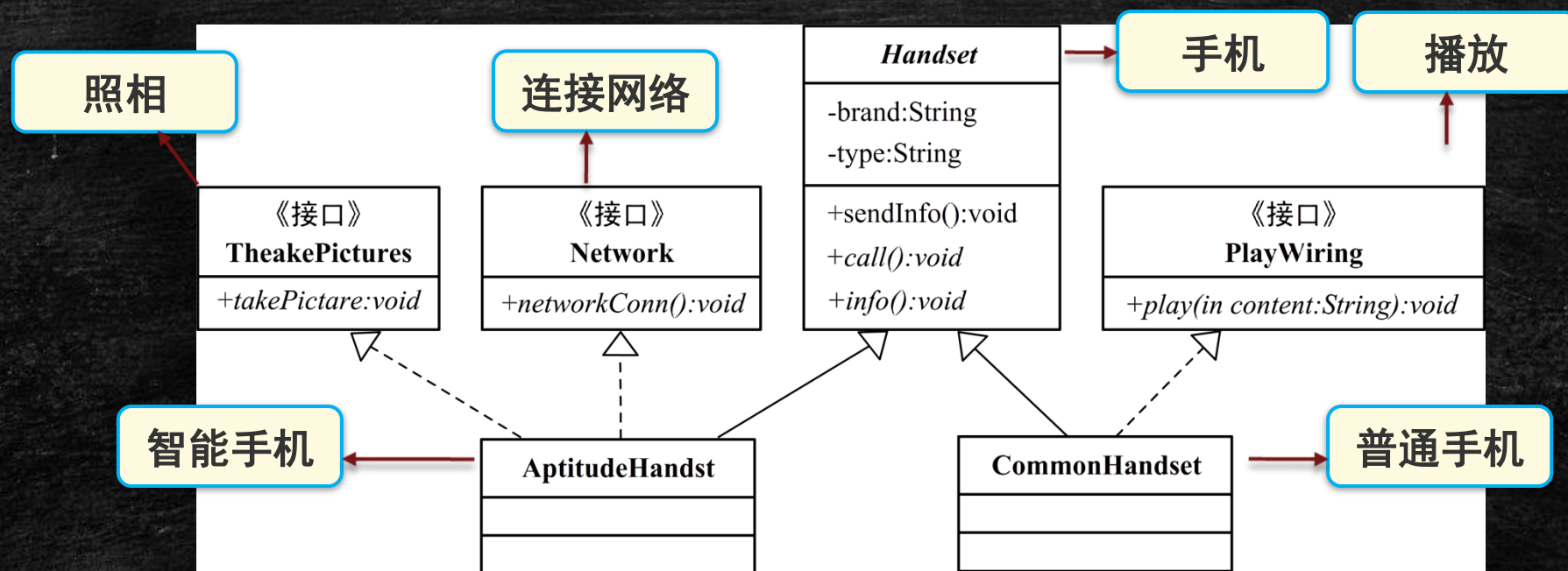
开始视频通话.....



上机练习6--使用接口实现手机功能2-2

实现思路

- 编写类及接口，参照以下类的结构图
- 编写测试类，让普通手机播放音频、发信息和通电话，让智能手机上网、播放视频、照相、发信息和通电话



接口是一种约定

- 生活中，我们使用的两相电源插座，规定了：
 - 两个接头间的额定电压
 - 两个接头间的距离
 - 接头的形状
- 接口是一种约定

体现在接口名称和注释上

有些接口只有名称

方法的实现方式
要通过注释来约定

- 面向接口编程

程序设计时面向接口的约定而不考虑具体实现



面向接口编程

- 开发打印机
 - 墨盒：彩色、黑白
 - 纸张类型：A4、B5
 - 墨盒和纸张都不是打印机厂商提供的
 - 打印机厂商要兼容市场上的墨盒、纸张

使用黑白墨盒在A4纸张上打印。

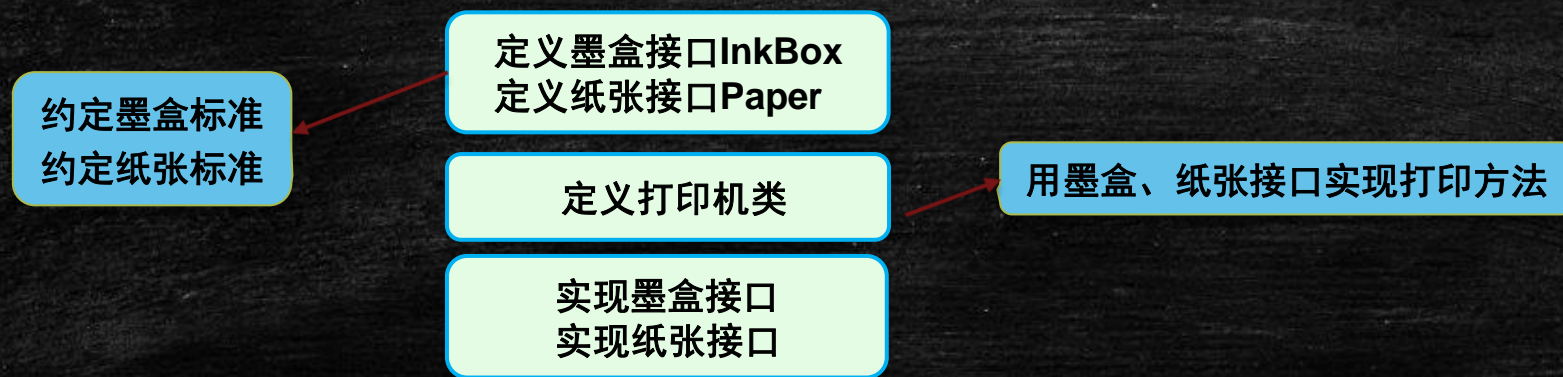
使用彩色墨盒在B5纸张上打印。

使用彩色墨盒在A4纸张上打印。



面向接口编程

- 墨盒和纸张的规格是一种约定
- 打印机需要遵守这些约定
- 用面向接口编程的方式开发
 - 制定墨盒、纸张的约定或标准
 - 打印机厂商使用墨盒、纸张的标准开发打印机
 - 其他厂商按照墨盒、纸张的标准生产墨盒、纸张



上机练习7--组装一台计算机2-1

- 训练要点：
 - 接口的基础知识
 - 理解接口表示一种约定
- 需求说明：
 - 采用面向接口编程思想组装一台计算机。
 - 计算机的主要组成部分有：
 - CPU
 - 硬盘
 - 内存

计算机的信息如下：

CPU的品牌是：Intel, 主频是：3.8GHz

硬盘容量是：3000GB

内存容量是：4GB



上机练习7--组装一台计算机2-2

▪ 实现思路:

定义计算机组成部分

- 定义CPU的接口CPU，返回CPU品牌和主频
- 定义内存的接口EMS，返回容量。
- 定义硬盘的接口HardDisk，返回容量。

实现计算机各组件信息

- 编写各组件厂商分别实现CPU、EMS、和HardDisk接口编写计算机类，组装计算机并显示相关信息
- 编写测试类运行



小结

- 编写和实现接口的语法是什么？
- 接口有哪些特性？（说出3个）
- 阅读代码，找出错误

```
public interface MyInterface {  
    public MyInterface();  
    public void method1();  
    public void method2(){ }  
    private void method3();  
    void method4();  
    int method5();  
    int TYPE = 1;  
}
```



接口 interface

▪ 为什么需要接口?接口和抽象类的区别?

- 接口就是比“抽象类”还“抽象”的“抽象类”，可以更加规范的对子类进行约束。
全面地专业地实现了：规范和具体实现的分离。
- 接口就是规范，定义的是一组规则，体现了现实世界中“如果你是...则必须能...”的思想。如果你是天使，则必须能飞。如果你是汽车，则必须能跑。如果你好人，则必须干掉坏人；如果你是坏人，则必须欺负好人。
- 接口的本质是契约，就像我们人间的法律一样。制定好后大家都遵守。
- 项目的具体需求是多变的，我们必须以不变应万变才能从容开发，此处的“不变”就是“规范”。因此，我们开发项目往往都是面向接口编程！



接口 interface

- 接口相关规则

- 接口中所有方法都是抽象的。
- 即使没有显式的将接口中的成员用public标示，也是public访问类型的
- 接口中变量默认用 public static final标示，所以接口中定义的变量就是全局静态常量。
- 可以定义一个新接口，用extends去继承一个已有的接口
- 可以定义一个类，用implements去实现一个接口中所有方法。
- 可以定义一个抽象类，用implements去实现一个接口中部分方法。



接口 interface

▪如何定义接口?

-格式:

```
▪[访问修饰符] interface 接口名 [extends 父接口1, 父接口2...] {  
    -常量定义    //总是public static final  
    -方法定义    //总是: public abstract  
    ▪}
```

▪如何实现接口

- 子类通过**implements**来实现接口中的规范
- 接口不能创建实例，但是可用于声明引用变量类型。
- 一个类实现了接口，必须实现接口中所有的方法，并且这些方法只能是public的。
- Java的类只支持单继承，接口支持多继承



接口 interface

- C++支持多重继承，Java支持单重继承
- C++多重继承的危险性在于一个类可能继承了同一个方法的不同实现，会导致系统崩溃。
- Java中，一个类只能继承一个类，但同时可以实现多个接口，既可以实现多重继承的效果和功能，也避免的多重继承的危险性。
- `class Student extends Person implements Runner, Flyer`
- `{...}`
- 注意：extends 必须位于implements之前



内部类—成员内部类

- 把一个类定义在另一个类的内部称为**内部类**

```
package com.mashibing.demo;//声明包
class Outer//声明类
{
    private String info="hello World";//声明私有属性
    class Inner//声明类
    {
        public void print(){//打印输出的方法
            System.out.println(info);
        }
    }
    public void fun(){//定义方法
        new Inner().print();//通过内部类调用方法
    }
}
public class TestOuterAndInner2//测试类
{
    public static void main(String [] args){
        new Outer().fun();//调用方法
    }
}
```

内部类轻松访问外部类的私有属性



内部类—成员内部类

- 注意事项:
- (1) 外部类不能直接使用内部类的成员和方法
- (2) 如果外部类和内部类具有相同的成员变量或方法，内部类默认访问自己的成员变量或方法，如果要访问外部类的成员变量，需使用this关键字

```
public class Outer//外部类
{
    public class Inner//内部类
    {
        public void print(){
            System.out.println("helloWorld")
        }
    };
    public void show(){//外部类的成员方法
        print();//调用内部类的成员方法
    }
};
```

```
public class Outer//外部类
{
    String name="张三";
    public class Inner//内部类
    {
        String name="李四";
        public void print(){
            System.out.println("外部类:name"+Outer.this.name);
            System.out.println("内部类:name"+name);
        }
    };
    public static void main(String [] args){
        Outer o=new Outer();//创建外部类的对象
        o.new Inner().print();
    }
};
```



在外部访问内部类

- 语法

- 外部类 外部类对象=new 外部类();

- 外部类.内部类 内部类对象=外部类对象.new 内部类 ();

```
public static void main(String [] args){  
    Outer out=new Outer();//创建外部类的对象  
    Outer.Inner inner=out.new Inner();//创建内部类的对象  
    inner.print();//访问内部类的方法  
}
```

如果主方法在外部类内部，则可以省略Outer
Inner inner=out.new Inner();



内部类--静态内部类

- 语法
- new 外部类类名.内部类().方法名
- 外部类类名.内部类 内部类对象名=new 外部类类名.内部类类名();

```
package cn.bjsxt.demo;//声明包
class Outer//声明类
{
    private static String info="hello World";//声明私有属性
    static class Inner//声明静态内部类
    {
        public void print()//打印输出的方法
        {
            System.out.println(info);
        }
    }

    public static void main(String [] args){
        new Outer. Inner().print();//调用方法
    }
}
```

使用static声明的内部类不能访问非static的外部属性



内部类--匿名内部类

- 适合只需要使用一次的类，安卓中使用的比较多

```
this.addWindowListener(new WindowAdapter() {  
      
        @Override  
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
    }  
});
```



内部类—方法内部类

- 方法内部类是指：将内部类定义在外部类的方法中。
- 注意事项：
 - (1) 方法内部类不能在外部类的方法以外的地方使用，所以
- 方法内部类不能使用访问控制符和static修饰符

```
class Outer//声明类
{
    private String info="hello World";//声明私有属性
    public void function(final int temp)//声明方法
    {
        class Inner//方法中的内部类
        {
            public void print()//打印输出的方法
            {
                System.out.println("类中的属性:"+info);
                System.out.println("方法中的参数:"+temp);
            }
        }
        new Inner().print();
    }
}

public class TestOuterAndInner2//测试类
{
    public static void main(String [] args){
        Outer out=new Outer();//创建外部类的对象
        out.function(30);
    }
}
```



总结2-1

·继承

- 符合is-a关系
- 使用extends关键字
- 代码复用

·方法重写的规则

- 方法名相同
- 参数列表相同
- 返回值类型相同或者是其子类
- 访问权限不能严于父类

■super关键字来访问父类的成员

- super只能出现在子类的方法和构造方法中
- super调用构造方法时，只能是第一句
- super不能访问子类的private成员



总结2-2

- 抽象类和抽象方法
 - 抽象类不能被实例化
 - 可以有0~多个抽象方法
 - 非抽象类必须重写父类的所有抽象方法
 - final修饰符
 - 修饰的类，不能再被继承
 - 修饰的方法，不能被子类重写
 - 修饰的变量将变成常量，只能在初始化时进行赋值
- Object类
是所有类的父类



内部类

- 将一个类定义置入另一个类定义中就叫作“内部类”
- 类中定义的内部类特点
 - 内部类作为外部类的成员，可以[直接访问](#)外部类的成员（包括private成员），反之则不行。
 - 内部类做为外部类成员，可声明为[private](#)、[默认](#)、[protected](#)或[public](#)。
 - 内部类成员只有在内部类的范围之内是有效的。
 - 用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的private还要小的访问权限。
 - 编译后生成两个类： OuterClass.class 和OuterClass\$InnerClass.class
- 内部类分类
 - 成员内部类 静态内部类 方法内部类 匿名内部类



内部类

- 匿名内部类Anonymous
 - 可以实现一个接口，或者继承一个父类
 - 只能实现一个接口
 - 适合创建那种只需要一次使用的类，不能重复使用。比较常见的是在图形界面编程GUI里用得到。
 - 匿名内部类要使用外部类的局部变量，必须使用final修饰该局部变量



内部类

```
class Outer{
    int outer_i = 100;
    void test(){
        Inner in = new Inner();
        in.display();
        System.out.println(in.a);
    }

    class Inner{
        int a=5;
        void display(){
            System.out.println("display: outer_i = " + outer_i);
        }
    }
}
```

- 1、Inner类是在Outer内部定义的
- 2、在Inner类中可以访问Outer类中的成员属性outer_i;
- 3、在Outer类中可在方法test()中创建内部类Inner的对象;
- 4、通过Outer类的对象调用test()方法最终就可以执行Inner类中的方法



垃圾回收机制

- 对象空间的分配：

- 使用new关键字创建对象即可

- 对象空间的释放：

- 传统的C/C++语言，需要程序员负责回收已经分配内存。显式回收垃圾回收的缺点：

- 程序忘记及时回收，从而导致内存泄露，降低系统性能。

- 程序错误回收程序核心类库的内存，导致系统崩溃。

- Java语言不需要程序员直接控制内存回收，是由JRE在后台自动回收不再使用的内存，称为垃圾回收机制(Garbage Collection)。

- 可以提高编程效率。

- 保护程序的完整性。

- 其开销影响性能。Java虚拟机必须跟踪程序中有用的对象，确定哪些是无用的。



垃圾回收机制关键点

- 垃圾回收机制只回收JVM堆内存里的对象空间。
- 对其他物理连接，比如数据库连接、输入流输出流、Socket连接无能为力
- 现在的JVM有多种垃圾回收实现算法，表现各异。
- 垃圾回收发生具有不可预知性，程序无法精确控制垃圾回收机制执行。
- 可以将对象的引用变量设置为null，暗示垃圾回收机制可以回收该对象。
- 程序员可以通过System.gc()或者Runtime.getRuntime().gc()来通知系统进行垃圾回收，会有一些效果，但是系统是否进行垃圾回收依然不确定。
- 垃圾回收机制回收任何对象之前，总会先调用它的finalize方法（如果覆盖该方法，让一个新的引用变量重新引用该对象，则会重新激活对象）。
- 永远不要主动调用某个对象的finalize方法，应该交给垃圾回收机制调用。



学习面向对象章节的注意事项

- 大家不要希望学到现在就精通面向对象，这只是开始。
- 事实上，很多人如果要对面向对象很精通，至少工作两年之后。



总结

- 封装
 - 成员权限修饰符 private 默认 protected public
 - 类权限修饰符 默认 public
- 继承
 - 方法重写
 - 构造方法执行过程 super关键字
 - Object类 重写toString() equals();
- 多态
 - 使用父类做形参 使用父类做方法返回值
 - 向上转型 向下转型
- final
 - final修饰类、成员变量、成员方法



总结

- 抽象类和抽象方法
 - abstract
- 接口
 - interface
 - implements
 - 抽象类和接口的联系和区别
 - 模拟实现Comparable接口和Comparator接口
- 内部类
 - 成员内部类 静态内部类 方法内部类 匿名内部类
- 垃圾回收机制
 - System.gc() 重写finalize()

