**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，它们都有可以飞的特点

* 使用接口的**原因**：

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

防盗门可以继承门的同时又继承锁么？如何解决这个问题呢？用接口实现

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

**package** interfacedemo;  
  
**public abstract class** Door {  
 **public abstract void** openDoor();  
 **public abstract void** closeDoor();  
}

**package** interfacedemo;  
  
**public interface** Lock {  
 **void** openLock(); **//可以写作public abstract void openLock();** **void** closeLock(); **//可以写作public abstract void closeLock();**  
}

**package** interfacedemo;  
  
**public class** LockDoor **extends** Door **implements** Lock{  
  
 @Override  
 **public void** openDoor() {  
 System.***out***.println(**"开门"**);  
 }  
  
 @Override  
 **public void** closeDoor() {  
 System.***out***.println(**"关门"**);  
 }  
  
 @Override  
 **public void** openLock() {  
 System.***out***.println(**"开锁"**);  
 }  
  
 @Override  
 **public void** closeLock() {  
 System.***out***.println(**"关锁"**);  
 }  
}

**package** interfacedemo;  
  
**public class** TestLockDoor {  
 **public static void** main(String[] args) {  
 LockDoor lockDoor = **new** LockDoor();  
 lockDoor.openDoor();  
 lockDoor.openLock();  
 lockDoor.closeDoor();  
 lockDoor.closeLock();  
 }  
}

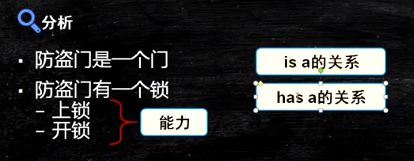
执行结果：

开门

开锁

关门

关锁



这个例子里面同样可以把Door也定义成接口，这样**public class** LockDoor **implements** Lock, Door 即可。但是按照现实生活中的逻辑来说有问题。可以说防盗门是一个门，有一个锁；但是不能说防盗门是一个锁，有一个门。这时可以用is a和has a来进行判断用哪个。如果是is a就用继承，如果是has a就用接口

Java中的继承是单继承，如果**拥有多个父类**的时候可以考虑使用**接口**进行实现

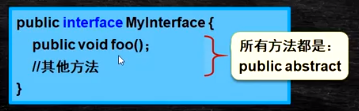
* Java中接口具备广泛的**使用**：

1.使用interface来修饰

2.接口中可以包含多个方法，且方法跟抽象类中的抽象方法一致，可以不写实现，子类

在实现的时候必须要实现代码逻辑

3.子类实现接口使用implements关键字



* **接口的特性**：

1. 接口不可以被实例化(因为接口中的所有方法都是抽象方法，不能包含方法的实现)

2. 接口中的所有方法的访问修饰权限都是public，不写并不是默认访问权限，而是public

3. 接口的子类必须要实现接口中的所有方法，跟抽象类有所不同。抽象类中的抽象方法要

被子类实现，非抽象方法不需要必须实现

4. 实现类可以实现多个接口

5. 接口中的变量都是静态常量，如果变量没有使用static关键字修饰，它也表示静态常量（是一个类变量），必须给一个初始值。不用final关键字修饰也是常量

比如在接口Lock里↓：

**int *a*** = 100; **//也可以写作static final int a=100;**

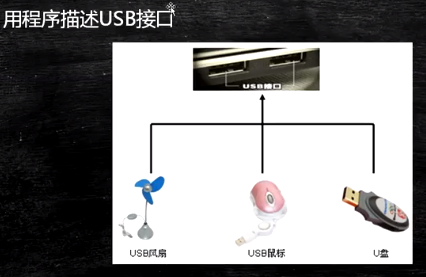
在TestLockDoor里↓：

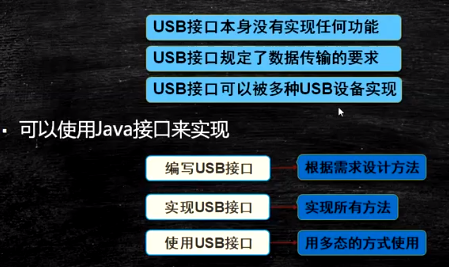
System.***out***.println(LockDoor.***a***);

这里的LockDoor.a是类方法，即一个静态类，是静态常量

6. 接口中的方法和常量无论是否添加public修饰，默认的权限有且只有一个，就是public

* **如何使用**接口：





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

接口表示一种能力，接口中可以定义N多个方法，子类在进行实现的时候必须要实现这些方法。将这些方法进行实现就意味着具备了方法的能力。

面向接口编程，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

**public interface** Usb {  
 **public void** dataTransfer();  
}

**public class** Mouse **implements** Usb{  
 @Override  
 **public void** dataTransfer() {  
 System.***out***.println(**"鼠标来点击屏幕"**);  
 }  
}

**public class** Upan **implements** Usb{  
 @Override  
 **public void** dataTransfer() {  
 System.***out***.println(**"U盘可以传输数据"**);  
 }  
}

**public class** Test {  
 **public static void** main(String[] args) {  
 Mouse mouse = **new** Mouse();  
 mouse.dataTransfer();  
 Upan upan = **new** Upan();  
 upan.dataTransfer();  
 }  
}

执行结果：

鼠标来点击屏幕

U盘可以传输数据

功能的实现：只要有花括号，里面即使什么也不写也叫功能的实现。

* **上课练习题：**

需求说明：

原始的手机，可以发短信，通电话。随着发展增加了新功能：音频，视频播放，拍照，上网

要实现的结果：

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：

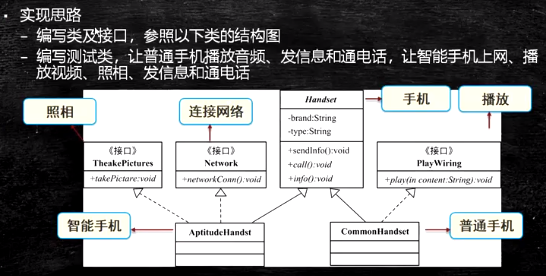
已经启动移动网络

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



将Handset定义成父类，普通手机和只能手机都继承Handset

而其他的功能设成接口

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

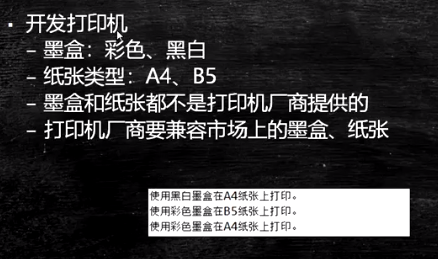
**接口是一种约定**，体现在接口名称和注释上。有些接口只有名称，方法的实现方式要通过注释来约定

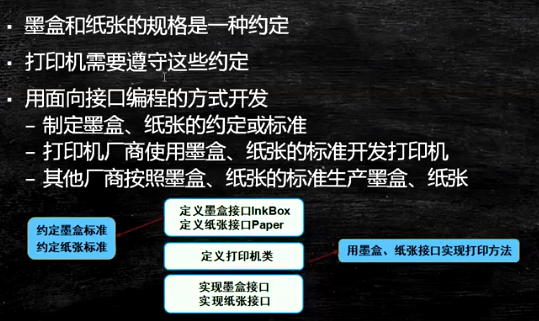
面向接口编程，程序设计时面向接口的约定而不考虑具体实现

**public interface** InterfaceDemo {  
 **/\*此方法表示可以实现展示A的功能  
 \*参数要求，必须是整型和String类型  
 \*/  
 public void** A(**int** a,String b);  
}

这时在使用这个接口的子类里就必须要实现接口里的方法

* 上机练习1：





**public interface** InkBox {  
 **public void** getWhiteAndBlack();  
 **public void** getColor();  
}

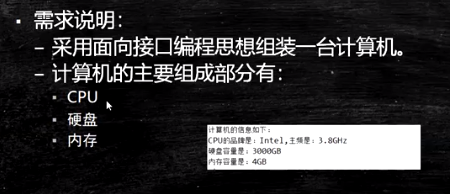
**public interface** Paper {  
 **public void** getA4();  
 **public void** getB5();  
}

**public class** Printer **implements** InkBox,Paper{  
 @Override  
 **public void** getWhiteAndBlack() {  
 System.***out***.print(**"使用黑白墨盒"**);  
 }  
  
 @Override  
 **public void** getColor() {  
 System.***out***.print(**"使用彩色墨盒"**);  
 }  
  
 @Override  
 **public void** getA4() {  
 System.***out***.print(**"在A4纸张上打印。"**);  
 }  
  
 @Override  
 **public void** getB5() {  
 System.***out***.print(**"在B5纸张上打印。"**);  
 }  
  
 **public static void** main(String[] args) {  
 Printer printer = **new** Printer();  
 printer.getWhiteAndBlack();  
 printer.getA4();  
 }  
}

执行结果：

使用黑白墨盒在A4纸张上打印。

* [上机练习2](#Computer组装实例常量添加)：





代码实现：

定义三个接口CPU，HardDisk，Memory：

**public interface** CPU {  
 **public** String getBrand();  
 **public** String getHZ();  
}

**public interface** HardDisk {  
 **public** String getVolumn();  
}

**public interface** Memory {  
 **public** String getCapacity();  
}

接口实现：

**public class** InterCpu **implements** CPU{  
   
 @Override  
 **public** String getBrand() {  
 **return "因特尔"**;  
 }  
  
 @Override  
 **public** String getHZ() {  
 **return "2000HZ"**;  
 }  
}

**public class** AmdCpu **implements** CPU{  
  
 @Override  
 **public** String getBrand() {  
 **return "Amd"**;  
 }  
  
 @Override  
 **public** String getHZ() {  
 **return "1000HZ"**;  
 }  
}

**public class** JSDHardDisk **implements** HardDisk{  
  
 @Override  
 **public** String getVolumn() {  
 **return "金士顿"**;  
 }  
}

**public class** SXHardDisk **implements** HardDisk{  
 @Override  
 **public** String getVolumn() {  
 **return "三星"**;  
 }  
}

**public class** Memory4G **implements** Memory{  
 @Override  
 **public** String getCapacity() {  
 **return "4GB"**;  
 }  
}

**public class** Memory8G **implements** Memory{  
 @Override  
 **public** String getCapacity() {  
 **return "8GB"**;  
 }  
}

Computer类实现组装功能：

**public class** Computer {  
 **public void** show(CPU cpu,HardDisk hardDisk,Memory memory){  
 System.***out***.println(**"计算机的组成如下："**);  
 System.***out***.println(**"cpu："**+cpu.getBrand()+**"主频是："**+cpu.getHZ());  
 System.***out***.println(**"硬盘容量是："**+hardDisk.getVolumn());  
 System.***out***.println(**"内存容量是："**+memory.getCapacity());  
 }  
}

Test类进行测试：

**public class** Test {  
 **public static void** main(String[] args) {  
 Computer computer = **new** Computer();  
 CPU cpu1 = **new** InterCpu();  
 HardDisk hardDisk1 = **new** JSDHardDisk();  
 Memory memory1 = **new** Memory4G();  
 computer.show(cpu1,hardDisk1,memory1);  
 }  
}

执行结果：

计算机的组成如下：

cpu：因特尔主频是：2000HZ

硬盘容量是：金士顿

内存容量是：4GB

找出下面接口中代码的错误：

**public interface** MyInterface {  
 **public** MyInterface(); ❌**//不允许定义构造方法，抽象类里可以定义构造方法  
 public void** method1(); ✔  
 **public void** method2(){} ❌**//不能加大括号  
 private void** method3(); ❌**//不能是private  
 void** method4();✔  
 **int** method5();✔  
 **int *TYPE*** = 1; ✔  
}

**总结：**

为什么需要接口？

接口就是比抽象类还抽象的抽象类，可以更加规范得对子类进行约束。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。例如如果你是天使，则必须能飞。如果你是汽车，则必须能跑。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。

如果接口实现里有相同的常量值，这时可以在接口实现里都写上这个常量值，但是麻烦，最好是定义到接口里，然后用这个 **接口的名称.常量名称** 来调用

[例如](#Computer组装实例)：

在每个接口实现(以CPU为例)里都写上这个常量值

**public class** AmdCpu **implements** CPU{  
 **public static final** String ***str*** = **"hehe"**;

}

**public class** InterCpu **implements** CPU{  
 **public static final** String ***str*** = **"hehe"**;

}

在Computer类里添加↓：

System.***out***.println(InterCpu.***str***); 或 System.***out***.println(AmdCpu.***str***);

Test里内容不变的情况下的执行结果：

计算机的组成如下：

**hehe**

cpu：因特尔主频是：2000HZ

硬盘容量是：金士顿

内存容量是：4GB

如果是在接口里定义常量：

**public interface** CPU {  
 **public static final** String ***str*** = **"hehe"**;

}

在Computer类里↓：

System.***out***.println(CPU.***str***); **//通过接口名.属性名**

在Test里执行的结果依然如上显示

1. **可以定义一个新接口，用extends去继承一个已有的接口**

**public interface** A {  
 **public void** show();  
}

**public interface** B **extends** A{  
  
}

1. **可以定义一个类，用implements去实现一个接口中所有方法**
2. **可以定义一个抽象类，用implements去实现一个接口中部分方法**

**public interface** A {  
 **public void** show();  
}

这时C可以实现接口A的方法，也可以不实现👇

**public abstract class** C **implements** A{  
 }

或者↓都可以

**public abstract class** C **implements** A{  
 @Override  
 **public void** show(){  
 System.***out***.println(**"show"**);  
 }  
 }

**接口A有个方法getName()，接口B继承接口A,不用复写接口A的方法，写接口B自己的东西(业务)即可,当一个类C去继承B接口的时候，是需要复写两个接口中的方法的实现的**

package practice;

public interface A

{

public void getName();

}

package practice;

public interface B extends A

{

public void getUser();

}

package practice;

public class C implements B

{

@Override

public void getName()

{

// TODO Auto-generated method stub

}

@Override

public void getUser()

{

// TODO Auto-generated method stub

}

}

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imjplements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

**class** Student **extents** Person **implements** Runner,Flyer{…}

注意extends必须位于implements之前

接口和抽象类的区别是什么？

1. 抽象类中方法可以有抽象方法，也可以有普通方法。但是接口中只能包含抽象方法
2. 抽象类需要使用abstract关键字来修饰，而接口使用interface关键字来修饰
3. 抽象类子类使用extends关键字来继承抽象类，使用implements来实现接口
4. 子类继承抽象类的时候必须要实现所有的抽象方法，普通方法可以不重写，而接口中的所有方法必须实现
5. 抽象类中可以定义成员变量，而接口中只能定义静态常量
6. 抽象类在子类实现的时候是单继承，而接口是多继承
7. 抽象类和接口都不能实例化，但是抽象类中可以有构造方法，而接口中不能有构造方法
8. 抽象类中可以实现接口，且不实现接口中的方法。而接口只能继承接口，不能实现接口

注意：在实际的项目开发过程中，如果可以使用接口尽量使用接口，将单继承的父类留在最关键的地方