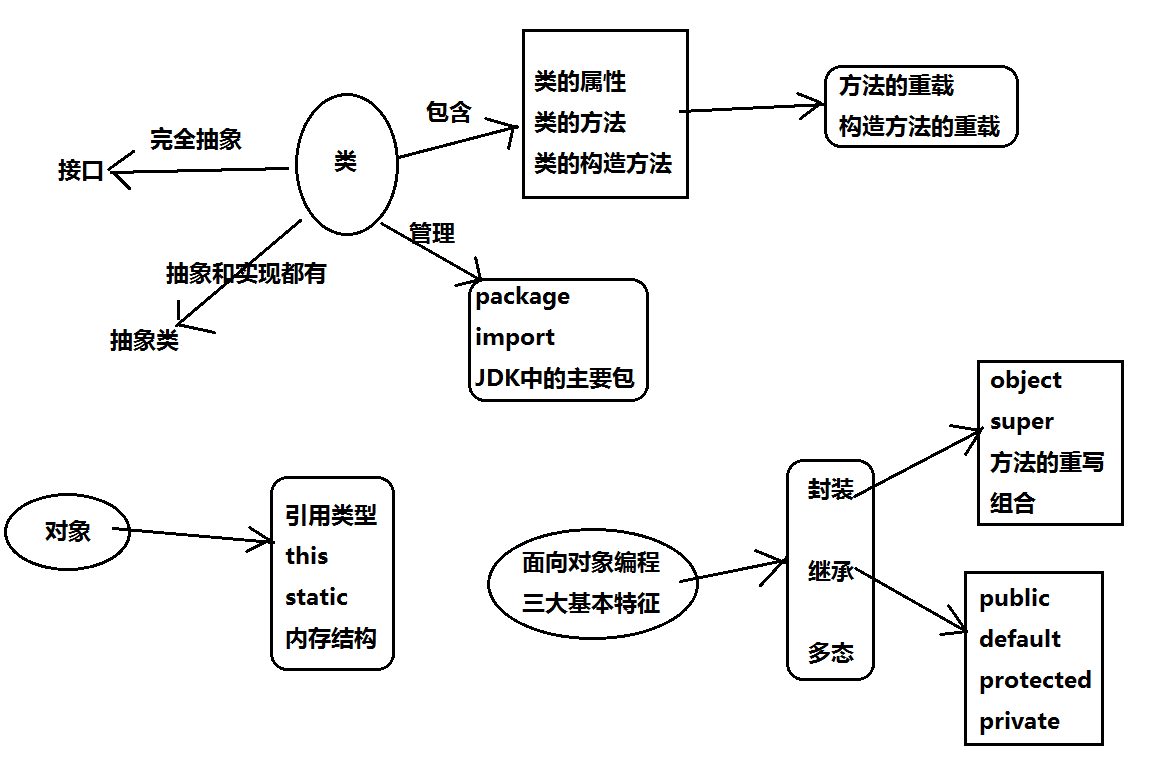
面向对象

## 本章内容概述：



# 第一章 面向过程与面向对象

## 1.1 面向过程思想到面向对象思想的演变过程：

|  |
| --- |
| 举例：  一个制造汽车的应用程序：  第一要素：  全局变量1; 全局变量2; 全局变量3; 。。。全局变量1000;  第二要素：  方法1(){aaaaa; 局部变量1；bbbb; ccccc };  方法2(){aaaaa; 局部变量1；bbbb; ccccc };  方法3(){aaaaa; 局部变量1；bbbb; ccccc };  ..........  方法1000(){aaaaa; 局部变量1；bbbb; ccccc };  在面向过程中，是以方法来进行组织代码的，然后方法之间的相互调用。这样场景在数据相对较少，方法相对较少的情况下，是没有问题的。  但当应用程序庞大复杂时，一个根本完成不了，需要多人协作来进行开发时，这时就有问题了。  别人在不清楚你所写的功能代码的情况下是很难上手的，因为我们是以方法来进行组织代码的，别人需要把整个流程熟悉后，才能进行在之前的基础上进行开发（别人需要把每一个方法内部的实现细节都要了解，才能进行修改，否则很有可能把之前程序的逻辑性打乱，从而引发BUG。比如，之前的逻辑是A方法中调用了B方法，但是经过修改之后，又调用了C方法，导致整个逻辑就出现了问题。）  所以，以面向过程的思维进行大型开发时，程序的管理比较复杂，不易于程序的扩充；对于程序的可维护性比较低；  但是，在C语言中，有一个结构体的特性，可以把每一类型的变量存放到每一个结构体中来进行管理。比如，把车相关的变量存放到车的结构体中，把行车导航仪的相关的变量存放到导航仪的变量中。当变量使用时，使用结构体.变量来调用对应的变量。方法也类似的操作，封装到一个个的结构体中。  但是，这样还是存在缺陷的，比如，车相关的方法：启动的方法，需要调用启动相关的变量时，必须要调用封装启动变量的结构体，才能获取数据。在此基础上，就在思考，为什么不把车启动的变量和方法的数据封装到一起呢，这样明显更加具有条理性。  于是，把每一类型的数据封装到一起，这时，类的概念就出现了（C++，在C语言的基础做了升级）。这个就是面向对象的思维；  以类的方式组织代码后，之前1000个变量和1000个方法的代码结构就会变为n个类，这样管理程序的效率就能大大提高。同时，程序也变的更容易扩展。比如，在现在的功能之上，添加自动驾驶的功能时，只需要添加一个相关的类即可，原有的代码结构并不需要修改。其次，程序的开发分工更加合理，比如按照功能来进行分工，从而提高程序开发的效率；当出现人员变更时，新人只需要研究对应的功能即可，不需要把整个应用程序流程研究一遍才能上手，这样也大大降低程序的维护成本；  在选择面向过程开发，还是面向对象开发时，是分应用场景的，如果程序的逻辑比较庞大复杂时，选择面向对象。如果代码比较少，逻辑比较简单时，选择面向过程即可，没必要去选择面向对象。 |

## 1.2 面向过程思维和面向对象思维的对比：

|  |
| --- |
| 举例：  如何驾驶汽车？  面向过程： 面向对象：   1. 踩离合； 1、驾驶员； 2. 挂挡； 2、汽车； 3. 踩油门，放离合； 3、驾驶员开汽车； 4. 车走了；   面向过程属于线性思维，考虑的是第一步做什么，第二步做什么。。。。。，当事情比较简单时合适。但当事情变的复杂，需求他人的帮助下才能完成时，就需要面向对象的思维了。  面向对象思维来思考事情该如何做时，首先考虑这件事情中，包含有哪些对象，比如，有驾驶员和汽车两个对象，把如何驾驶汽车的多个方法封装到汽车对象当中，汽车直接调用即可实现。所以，在面向对象当中，是包含着面向过程，是对面向过程的高级封装；  面向过程：以事件为中心，强调的是做一件事的步骤；  面向对象：以事物为中心，强调的是由谁来做；  二者最终的目的一样，但方式不同；  举例：  如何造汽车：  面向过程： 面向对象；   1. 造车轮？ 车轮： 2. 造发动机？ 买橡胶 3. 造车皮？ 到马来西亚 4. 挡风玻璃？ 找到橡胶厂   掏钱买  很难决定上面的每一步步骤之间的关系 用船将橡胶运到国内  是先造发动机还是先造车轮？ 造磨具  将橡胶放入磨具  出车轮  发动机  车壳  座椅  挡风玻璃  最后将上面造出来的部件进行组装汽车就好了  制造汽车一定是一个非常复杂的工艺过程，参与在其中的工作人员非常多，此时如果用面向过程的思维来设计，很不现实，如果把车的各个部件拆分归类，来进行生产，就是一个很合理的设计。 |

## 1.3 面向对象思想编程 总结：

1. 编程的本质：

把现实中存在的逻辑，使用代码来实现；现实中逻辑，就是使用语言了进行表达的。

1）面向对象编程（OOP）的本质：

以类的方式组织代码，以对象的方式封装数据（以面向对象的思维组织代码，封装数据）。

1. 什么是对象：

对象是具体的事物，一类事物具体的一个实例；

举例：张三、李四就是人的对象；

什么是类：

是对对象的一种抽象（抽象，抽出像的部分），一类事物的一个抽象；

举例：人、动物、汽车，都是类；

在编程思想中，必须先定义类，才有对象，对象是java的核心；

抽象的两个方面，也就是一个类包含的两个部分；

1. 静态属性------>成员变量；
2. 动态行为------>成员方法；

每一个对象都有自己的属性，属性值一般和另一个对象的属性是不一样的。

调用一个方法，必须先new一个对象，然后对这个对象来调用这个方法。针对不同对象调用的方法，它的作用域是不一样的。

类是对象的模版，new出来的对象，都共享类的模版；

1. 面向对象也叫做面向类的编程：

作为面向对象，在处理业务逻辑时，不应该先考虑第一步做什么，第二部做什么；

考虑：

1. 在问题当中，有哪些类，哪些对象；
2. 在这些类和对象中，应该具有哪些属性和方法；
3. 类与类之间有怎么样的关系；

技巧：找类的方法：名词，判断名词是类，还是属性；

1. 如何在内存中区别类和对象：

类是静态的，在内存的代码区；

对象是new出来的，在堆内存；

1. 对象的创建和使用：
2. 必须使用new关键字来创建对象；
3. 使用对象名（引用）.成员变量来引用对象的成员变量；
4. 使用对象名（引用）.方法(形参列表)来调用对象的方法；
5. 操作对象，就是操作它的地址，对象名（引用）就是地址；
6. 对象和类 总结：

对象和类的关系：特殊到一般，具体到抽象；

类：class;

对象：Object(事务) ， instance（实例） ;

类可以看成一类对象的模版，对象可以看成该类的一个具体实例；

类是用于描述同一类型的对象的一个抽象的概念，类中定义了这一类对象所应具有的静态和动态属性。

对象是java程序的核心，在java程序中“万事万物皆对象”；

JDK提供了很多类供编程人员使用，编程人员也可以自定义类；

## 1.4 引用类型：

四类八种之外的所有类型，全部为引用类型。引用类型占两块内存，一小块内存，指向一大块内存，所以，所有的引用类型全是指针；

# 第二章 面向对象编程：

## 2.1 创建Java自定义类：

步骤：

1）定义类（考虑修饰符、类名）

2）编写类的属性（考虑修饰符、属性类型、属性名、初始化值）

3）编写类的方法（考虑修饰符、返回值类型、方法名、形参等）

## 2.2 类的属性、类的方法：

1）属性，field，属于成员变量；

2）属性用于定义一个类或一个类的对象所包含的数据，或者说静态属性；

3）属性的作用范围是整个类体；

4）在定义成员变量时，可以对其进行初始化，如果不对其初始化，Java使用默认值对其初始化；

5）属性定义格式：

[修饰符] 属性类型 属性名 = [默认值];

6）面向对象中，整个程序的基本单位是类，方法是从属于类的。

7）java中方法参数传递是：值传递；

## 2.3 JVM内存分析：

1. 桟：自动分配连续的空间，释放内存时，具有后进先出的特点；
2. 堆：凡是new出来的东西，都在堆中；分配不连续的空间，包含有方法区；
3. 方法区：包含：类的信息、static静态成员、常量池（字符串常量。。。）；

## 2.4 Package 和 import :

1）功能：为了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。提供类的多重命名空间。

2）package语句作为java源文件的第一条语句；若缺少该语句，则指定为无名包；

3）编辑出来的.class文件必须位于正确目录（和包的层次完全一致）；

4）如果在另外的类中，要用这个包下的其他类，必须把全路径写完整或者import引入；

5）class文件的最上层包位于classpath下；

6）不同项目设置不同的classpath;

# 第三章 构造方法：

1. 构造器：也叫构造方法，constructor;
2. 构造器的作用：（1）用于构造该类的实例（对象），（2）用来初始化对象的属性；
3. 语法格式：

[修饰符] 类名（形参列表）｛

代码块；

｝

1. 构造器是一种特殊的方法：

A ) 使用new + 构造方法，来创建一个类的新对象；

B ) 构造器虽然有返回值，但是不能定义返回类型（返回值的类型肯定是本类对象），不能在构造器里调用return;

C ) 如果没有定义构造器，则编译器会自动定义一个无参的构造方法。如果已经定义则编译器不会添加；

D ) 构造器的方法名必须和类名一致；

E ) 构造方法可以有多个，构造方法的重载；

F ) 最好同时定义无参和有参构造方法；

# 第四章 方法的重载[overload]：

1. 方法的重载是指一个类中可以定义有相同的名字，但参数不同的多个方法。调用时，会根据不同的参数表选择对应的方法。
2. 两同三不同：

同一个类，同一个方法名；

不同：参数列表不同（类型，个数，顺序不同）；

1. 只有返回值不同不构成方法的重载：

|  |
| --- |
| Int a(String str){}  Void a(String i){}  调用：a() 不能确定要调用哪个方法 |

4）与普通方法一样，构造方法也可以重载

# 第五章 static 关键字：

1. 表示static修饰的成员，属于静态成员，所有对象共享一份，它存放在方法区里；
2. 用static声明的方法是静态方法，在调用该方法时，不会将对象的引用传递给它，所以在static 方法中不可访问非static的成员；（静态方法不再是针对于某个对象调用，所以不能访问非静态成员）

静态方法在类加载时就被初始化了；

在没有任何对象产生下，静态方法不能调用非静态成员；

静态方法可以调用静态成员；

1. Static修饰的成员，属于类级别的，可以用类名直接调用；
2. 也可以用对象名调用，但建议用类名调用；
3. 如果某个对象修改了static静态变量的值，其他对象共享改变；
4. Static的作用：
5. 在堆里，每个对象都会有一个相同的空间，浪费内存；
6. 如果修改值，需要每一个对象一个个的改；

# 第六章 this 关键字：

1. 在普通方法中，this指向调用该方法的对象；
2. 在构造方法中，this指向正要初始化的对象；
3. This不能用于static方法中；
4. 区别成员变量和局部变量；当在一个方法内时，局部变量与成员变量重名时，为了区分，成员变量要加this ; 如果没有重名的情况，this可以省略（加上也可以）；
5. 使用this来调用方法；方法没有重名时，this可以省略；
6. This可以调用构造方法：（语法要求：必须是第一句， this()）
7. This在内存中的表示：

This是一个引用变量，指向的是当前对象自身；

每个对象都有一个this变量，this成员变量；

# 第七章 代码块：

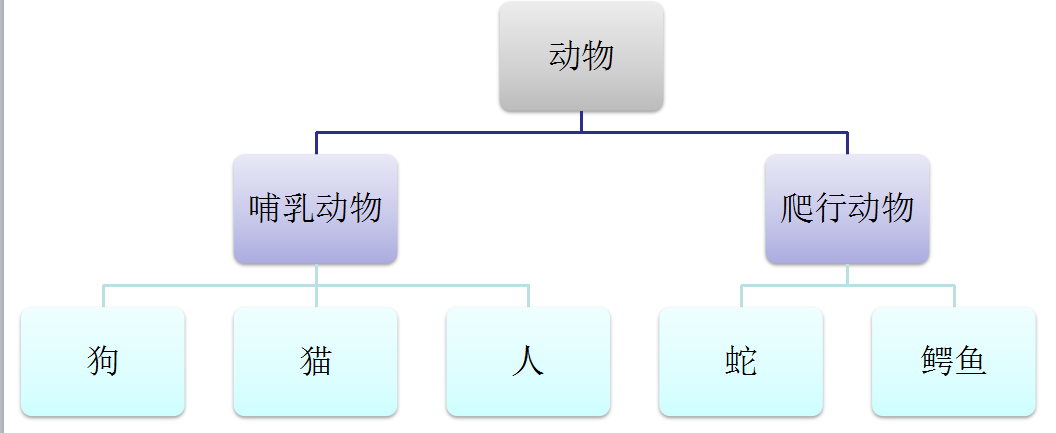
# 第八章 面向对象三大特性之封装：

1. 封装，也叫做隐藏；把成员变量封装起来，隐藏内部实现的细节，通过访问修饰符来实现不同的封装程度；
2. java中的四个访问修饰符；
3. private : 只能在本类中被访问；
4. 默认的（访问修饰符缺省）：只能在本包中直接被访问；
5. Protected : 本包可以直接访问，外包子类可以直接访问；
6. Public : 项目下，所用包都可以访问；
7. 不同的访问修饰符代表着不同的封装程度；
8. 四种访问修饰符都可以修饰成员变量和成员方法；

但修饰类时，只能用public和默认得来修饰；

1. 封装的好处：
2. 安全、对程序来说；
3. 简单、对用户来说；

# 第九章 面向对象三大特性之继承：



1. 继承：子类继承父类，父类也叫基类、超类；

子类从父类继承过来所用的成员（属性、方法）；

1. 作用：

1、类是对对象的抽象，继承是对某一批类的抽象，从而实现对显示世界更好的建模；

2、通过继承可以简化类的定义，提高代码的复用性；

3、父类修改了属性，子类自动的获得修改；

1. 语法：public class 子类名 extends 父类名；extands的意思是“扩展”。子类是父类的扩展
2. 如果一个类没有表示继承哪个类，默认继承Object类；
3. 如果子类里有和父类同名的成员变量；使用this和super来分别表示；

在每个普通的方法中，都包含两个隐式参数：this , super;

1. 调用成员方法：
2. 子类里如果没有定义这个方法，就调用从父类继承过来的；
3. 子类如果重写了父类的方法，就调用子类重写过的；
4. 子类特用的方法；
5. 子类继承父类的成员变量和成员方法，但构造方法不能继承：
6. 在java中，一个类只能继承一个（直接）父类，不能有多继承，但可以多层继承（通过接口实现）；

多继承：

优点：子类同时拥有多个父类的属性和方法；

缺点：当多个父类中用同名的属性或方法，子类引用时会产生歧义；

1. 从代码的重用的角度来看，组合完全可以替代继承，组合比继承更加灵活；
2. 继承条件下，代码块和构造方法的执行顺序：

父类静态代码块 -> 子类静态代码块 -> 父类非静态代码块和构造方法 -> 子类非静态代码块和构造方法。

1. 继承的语义：

“is a”:是一种；子类是父类的一种；

例：Dog extends Animal ： Dog是一种Animal类型的对象；

## 9.1 方法的重写（override）：

1. 必须是继承自父类的方法，并且父类的方法已经不适用与子类；

2）重写方法必须和被重写方法具有相同方法名称、参数列表和返回类型。

重写方法不能使用比被重写方法更严格的访问权限。（由于多态）

## 9.2 super关键字 ：

1. 作用：

执行当前对象的父类对象，每一对象都有一个supser属性，创建对象时分配空间；

1. 使用：
2. 修饰父类的成员变量：super.属性名；
3. 修饰父类的成员方法：super.方法名；
4. 修饰父类的构造方法：super.构造器；<在子类的构造器中，调用父类的构造器，要求：必须是第一句>
5. 注意：

This和super不能同时出现在子类的构造器中；

1. this和super都必须是构造方法中的第一条语句，如果存在，谁是第二条语句都是错误；
2. 继承条件下，对象的构建顺序：子类会调用父类的构造器；

A ) 子类构造方法，默认会调用父类无参构造方法；

1. 如果在子类构造方法中，显示指定了父类的构造方法时，就调用指定的父类构造方法；
2. 父类使用private修饰的成员，子类可以继承，但是无法访问；如果访问，在父类中设置set方法，在子类中通过super.set方法来调用；
3. 构建子类对象时的内存图：

Dog dog = new Dog();

创建一个Dog子类对象，会自上而下分别创建Object、Animal、Dog三个对象，每个对象除了具有自身的成员变量外，还分配this和super两个属性空间，this指向当前对象自身，super指向当前对象的直接父类对象；

Object、Animal、Dog三个对象是依赖于super关键词联系起来的，最后，将Dog子类的内存起始位置赋给dog引用变量；

注意：分配内存时，父类对象Animal的color, age属性并不会在Dog子类中分配空间。

子类对象要比父类对象大，因为子类对象包含了父类对象；

## 9.3 Object 类：

Object类是所有Java类的根基类

如果在类的声明中未使用extends关键字指明其基类，则默认基类为Object类；

## 9.4 Final 关键字：

语义： 最终结果；

1. 修饰类：不能被继承；

|  |
| --- |
| *Public final class 类名{}* |

1. 修饰方法：不能被重写，但可以重载；

|  |
| --- |
| *Public final void 方法{}* |

1. 修饰变量：不可变的变量（必须在声明时、构造器、初始化块中赋初始值）
2. 不能与abstract、private使用，与static使用；

# 第十章 面向对象三大特性之多态：

1. 多态：多种状态，同一个方法，多种结果；

父类引用实际上指向的是哪个子类对象，就调用哪个子类重写过的方法；

父类引用只能调用子类重写过的方法，不能调用子类独有的方法。

多态只是方法的多态，属性没有多态；

2、多态的必要条件:

1. 继承；
2. 重写；
3. 多态之前，本类类型的引用指向本类对象。多态之后，父类引用指向子类对象；

3、多态的作用：

A）代码灵活、美观、可读性高、避免代码重复；

B）使程序具有良好的可扩展性，程序设计开闭原则：对扩展开发，对修改关闭；

4、多态的使用：

A）用父类引用作为方法的形参，去接收不同的子类对象；特点：节省代码，减少方法重载的数量，提高代码的可维护性；

B）用父类类型作为方法的返回值类型，来返回不同的子类对象；

5、多态的转型：

A）向上转型：

自动转型，把子类引用转成父类引用； Animal dog = new Dog();

B）向下转型：

1、强制转换，把父类引用转成子类引用：Dog dog = (Dog)dog;

2、一定要转成dog所代表的实际对象的类型，否则会报错；Cat cat = (Cat)dog;报错

3、为了避免强转异常，在强转之前判断一下，父类引用指向的是哪个子类对象；

InstanceoOf 关键字：

例：dog instanceOf Cat : 引用变量dog所代表的实际类型是Cat吗？

1. 对象名（dog）所代表的编译类型（Animal）和类名或者接口名是继承数的上下级关系，所以，编译不会报错；

# 第十一章 抽象类 abstract：

1. 抽象类：包含抽象方法的类，叫做抽象类。包含抽象方法的类，必定是抽象类；

抽象方法：只有方法的声明，没有方法体；

抽象类是一个模版，避免了子类设计的随意性，很好的限制了子类的设计，使各子类之间更加通用。抽象类本身也是一个类，所以它的继承也要循序“is a”的语义；

1. 抽象类属于不完整的类，所以不能构建本类的对象；
2. 子类继承抽象类：
3. 实现：父类的抽象方法，子类需要实现；

父类的方法已经实现，子类可以重写；

1. 如果不实现父类的抽象方法时，需要把子类也定义成抽象类；

4、使用场景：父类的方法不能满足子类的需求，子类需要重写时，就把父类定义为抽象类；

5、语法：abstract 关键字

|  |
| --- |
| Public abstract void 方法名(); 方法后不加大括号，表示没有方法体，只提供了方法的定义； |

Abstract 不可以修饰变量，不能与private、final、static连用；

# 第十二章 接口interface：

本质上还是多态：

1. 多态的3大用法：
2. 接口引用指向实现类对象：
3. 接口引用作为方法的形参，来接收不同的实现类对象；
4. 接口类型作为方法的返回值类型，返回不同的实现类对象；
5. 接口，只包含常量和抽象方法；

接口的变量，默认是：public static final（可以省略）；

接口的方法，默认是：public abstract (可以省略)；

1. 接口的使用：

Interface 关键字；

1. 接口类型的引用变量指向实现类对象；
2. 调用的方法必须是接口里定义，实现类实现的方法；
3. 实际指向的是哪个实现类对象，执行的就是哪个对象的方法；
4. 一个类即能继承父类，又能实现接口；
5. 一个类可以同时实现多个接口；
6. 一个接口继承另一个接口；
7. 一个接口同时继承多个接口；
8. 接口的好处：
9. 可以实现多继承；
10. 运用多态，面向接口编程；
11. 类和接口的区别：

类和接口是同一级别的；

类是“is a”语义，是一种，子类必须是父类的一种；

接口是“has a”语义，有一种，实现类实现接口，实现类拥有一种接口里定义的功能；

1. 抽象类和接口的区别：

抽象类是类，“is a”语义，接口是”has a”语义；

抽象类是用来继承的，接口是用来实现的；

抽象类有构造方法，接口没有构造方法；

都不能创建自身的对象，都可以创建子类或者实现类的对象（多态）；

目的：设计和功能实现进行分离；功能实现的管理者，不关心功能的具体的实现；

## 12.1 Java 8中关于接口的改进：

Java 8中，你可以为接口添加静态方法和默认方法。从技术角度来说，这是完全合法的，只是它看起来违反了接口作为一个抽象定义的理念。

静态方法：使用 static 关键字修饰。可以通过接口直接调用静态方法，并执行其方法体。我们经常在相互一起使用的类中使用静态方法。你可以在标准库中找到像Collection/Collections或者Path/Paths这样成对的接口和类。

默认方法：默认方法使用 default 关键字修饰。可以通过实现类对象来调用。我们在已有的接口中提供新方法的同时，还保持了与旧版本代码的兼容性。

比如：java 8 API中对Collection、List、Comparator等接口提供了丰富的默认方法。

特点：

1. 静态方法和默认方法的访问修饰符默认是public,可以省略；
2. 默认方法可以被继承和重写，但静态方法不可以被实现类继承；

## 12.2 接口中的默认方法：

接口默认方法的“类优先”原则

若一个接口中定义了一个默认方法，而另外一个父类或接口中又定义了一个同名的方法时

选择父类中的方法。如果一个父类提供了具体的实现，那么接口中具有相同名称和参数的默认方法会被忽略。

接口冲突。如果一个父接口提供一个默认方法，而另一个接口也提供了一个具有相同名称和参数列表的方法（不管方法是否是默认方法），那么实现类必须覆盖该方法来解决冲突

# 第十三章 类包含的元素：

1. 成员变量：描述对象的静态属性，分为静态成员变量、非静态成员变量；
2. 代码块：
3. 构造方法；
4. 成员方法：描述对象的动态行为，分为静态成员方法、非静态成员方法；
5. 内部类；