面向对象之继承

继承的概念：

让类与类之间产生父子关系。被继承的类叫做父类（基类、超类），继承的类叫做子类（派生类）。

格式（extends）：

Class父类{

.........

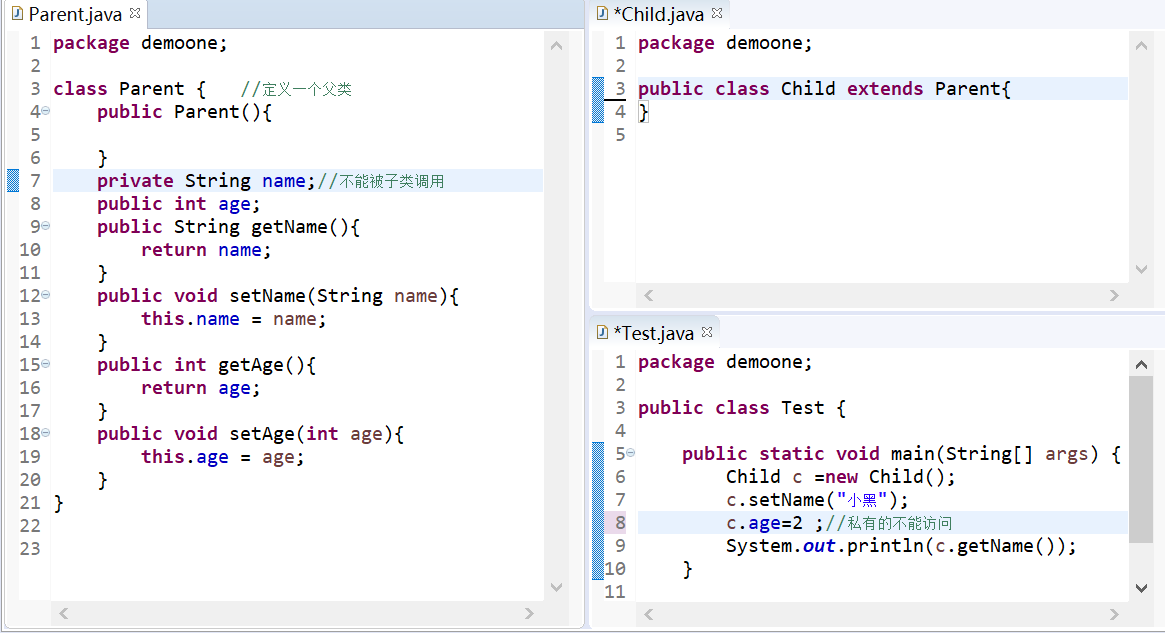
}

Class 子类 extends 父类{

........

}

继承之后子类可以拥有父类的非私有成员（成员变量、成员方法）



继承的使用场景：多个类存在相同的属性和行为时，可以将这些内容提取到一个新类中，让这些类和新类产生父子关系，实现代码复用

继承的优点：功能复用（直接将已有的属性和行为继承过来，实现了功能的复用，节省大量工作）、便于拓展新功能、结构清晰简化认识、易维护性（不同类之间的继承关系，让这些事物之间保持一定程度的一致性，大大降低了维护成本）

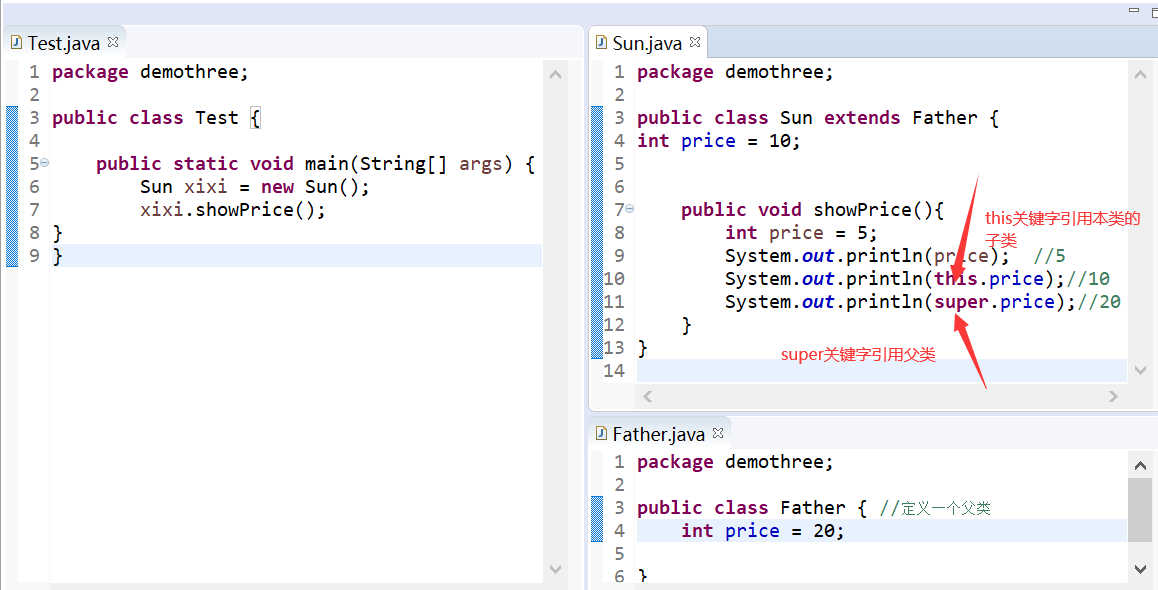
继承的缺点：打破了封装性（父类向子类暴露了实现细节，打破了父类对对象的封装性）、高耦合性（类与类之间相互依赖性较强）

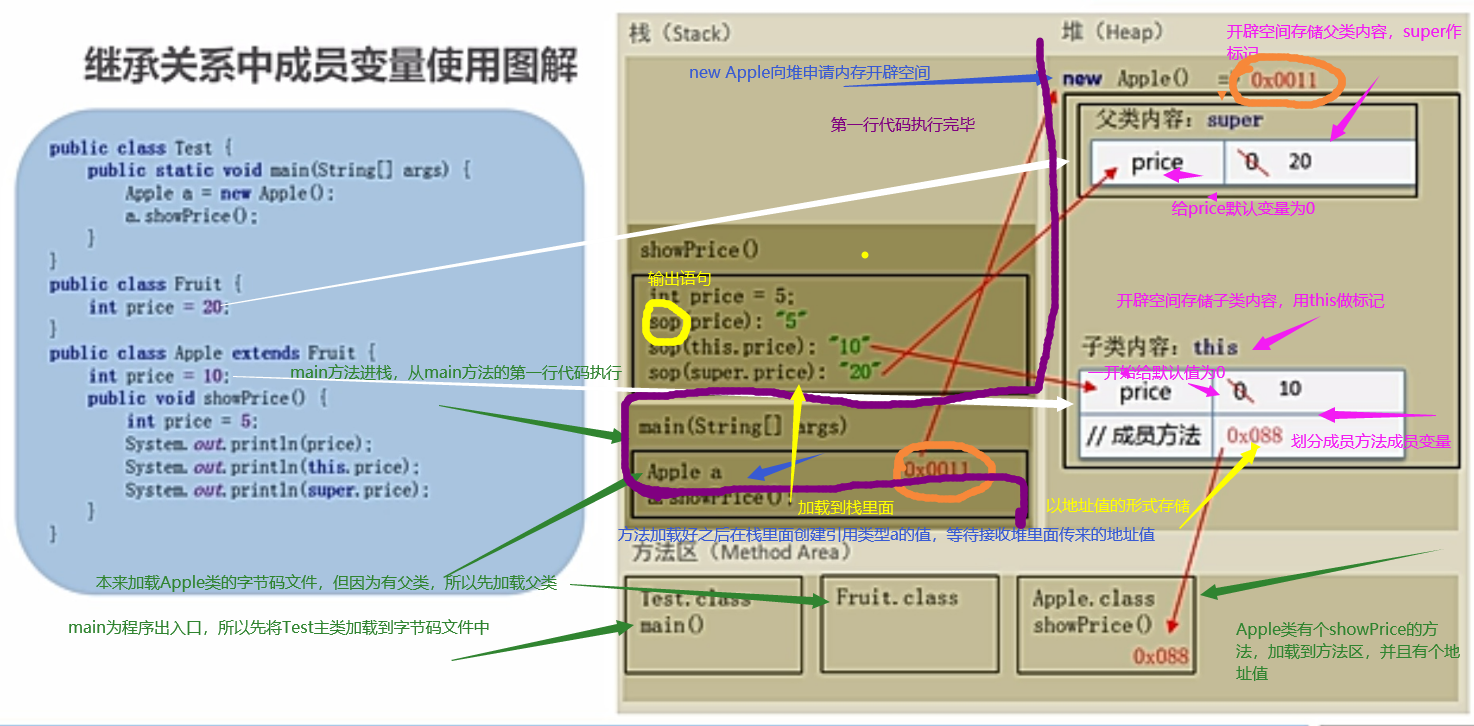
类似于大齿轮和小齿轮 一个坏零一个坏，所以代码讲究低耦合高内聚

继承关系类成员的使用：子父类定义了同名的成员变量

遵循就近原则，局部变量先使用，没有就去子类的成员变量找，有就使用，再没有就去父类的成员变量找，有就使用

局部变量>>成员变量>>父类>>更高的父类.......object





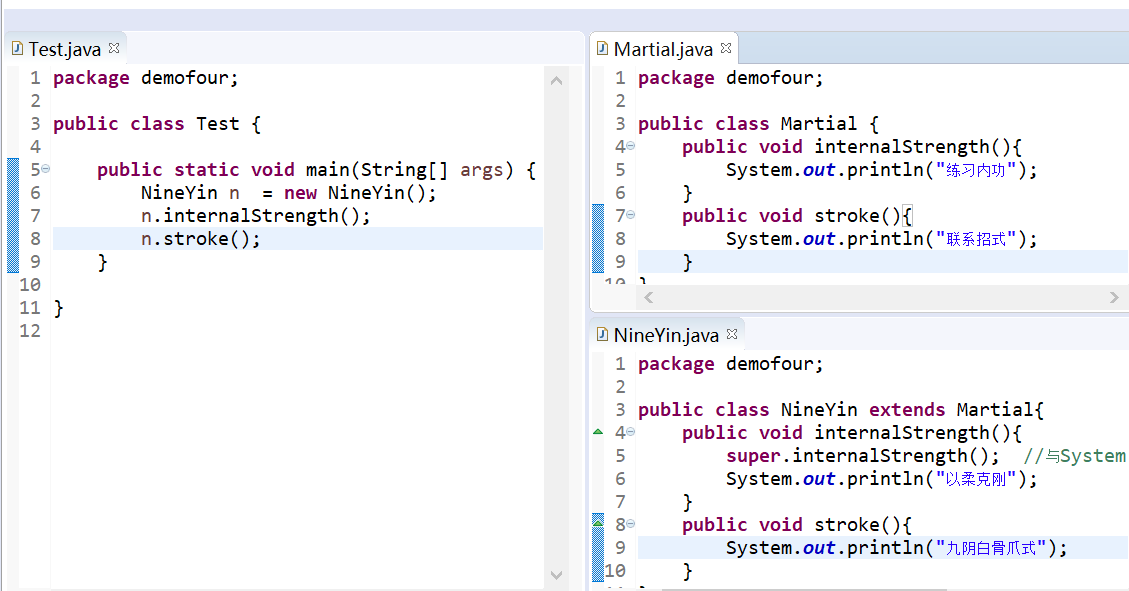
this和super区别

this 本质：对象 可理解为在堆内存中new出来的对象总的地址值，而这块空间要根据需求划分成若干个，其中有一个空间就是super，代表父类内存空间，用来初始化父类内容

用法：从本类开始找

Super：本质:父类内存空间的标识

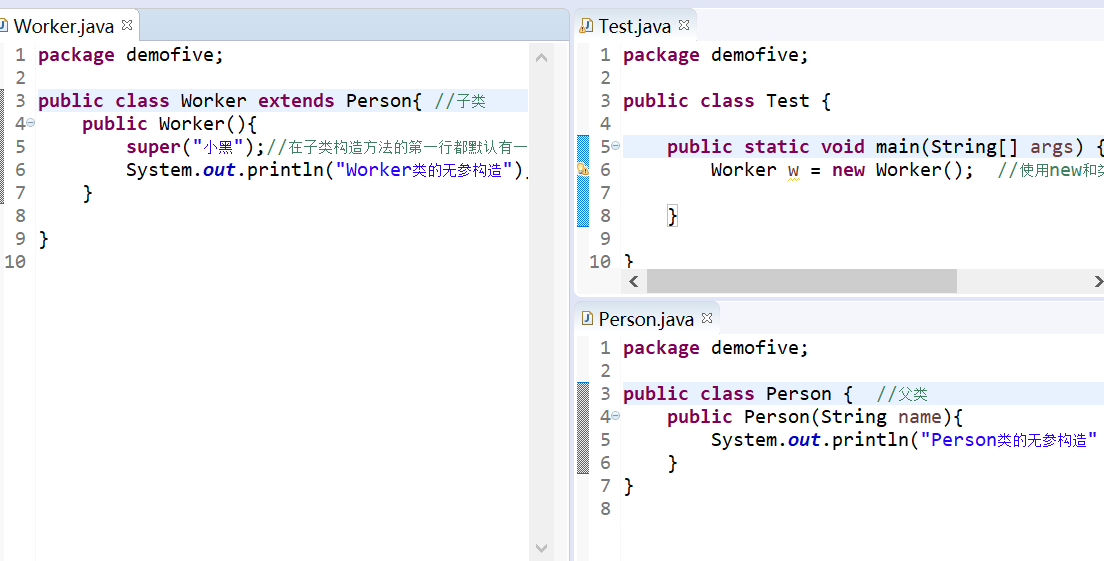
用法：从父类开始找



继承关系中构造方法的使用

1.创建子类时，优先调用父类构造方法

2.子类构造方法的第一行，隐含语句super（），用于调用父类默认无参构造，若在父类中不存在默认无参构造，需手动调用父类其他构造



使用new操作符和类的构造方法为声明的对象分配变量，即创建对象

方法重写（Overrride)

定义：子类中出现和父类方法定义相同的方法现象

方法复写也叫方法的重写、覆盖，方法名、参数列表、返回值类型都相同

注意事项：父类私有方法无法重写

子类方法访问权限不能小于父类方法

子类不能抛出比父类方法更大的异常

使用场景：父类功能过时时，重新实现父类功能，拓展父类功能



四大权限 本类 同包下的类 不同包下的子类 不同包下的无关类

Private T

默认 T T

Protected T T T

Public T T T T

方法重写和方法重载的区别

重载 方法名相同

参数列表不同（个数或对应位置类型不同）

返回值类型相同

修饰符无关

定义同一个类中

重写 方法名相同

参数列表相同

返回值类型相同

修饰符：访问权限不小于被重写方法

定义子父类之间

Java只支持类的单继承，但支持多层（重）继承

Public class Fruit{

}

Public class Apple extends Fruit {

}

Public class Orange extends Fruit{

}

Public class Fuji extends Apple{

}

苹果继承水果 橘子继承水果 腹肌继承苹果

Java只继承父类的非私有成员

构造方法不继承 用于初始化本类对象，创建子类对象时，需要调用父类构造初始化

继承体现了“is a”的关系 如 苹果is a 水果 可使用继承

面向对象之多态

多态性就是指父类的某个方法被其子类重写时，可以各自产生自己的功能行为，类似于方法的重载（方法的重载指的是类当中有若个同名的方法，但方法的参数不同）

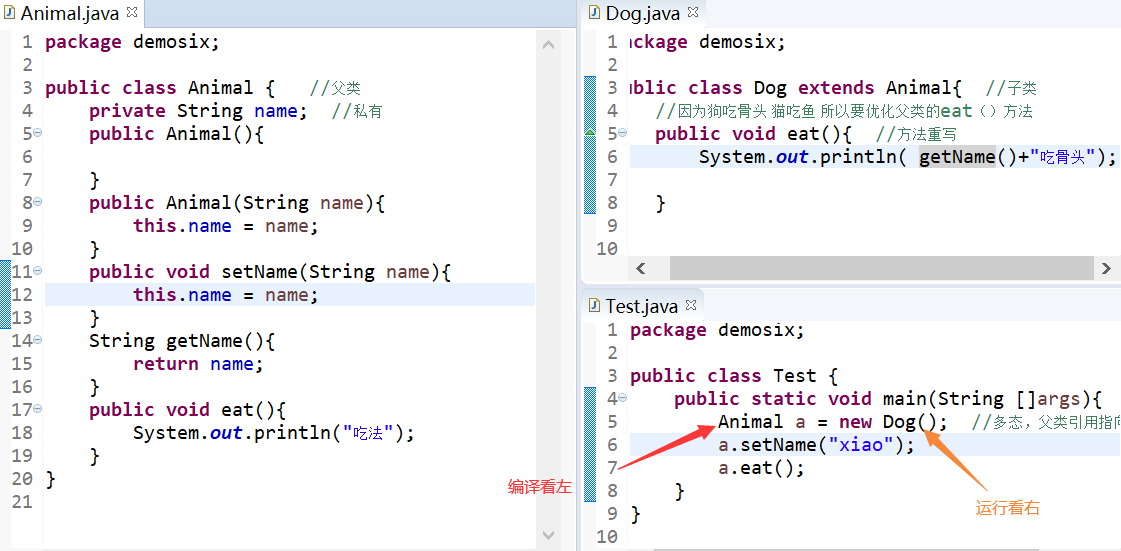
多种状态，同一对象在不同情况下表现出的不同状态或行为

实现多态步骤： 要有继承（或实现 类与接口关系）

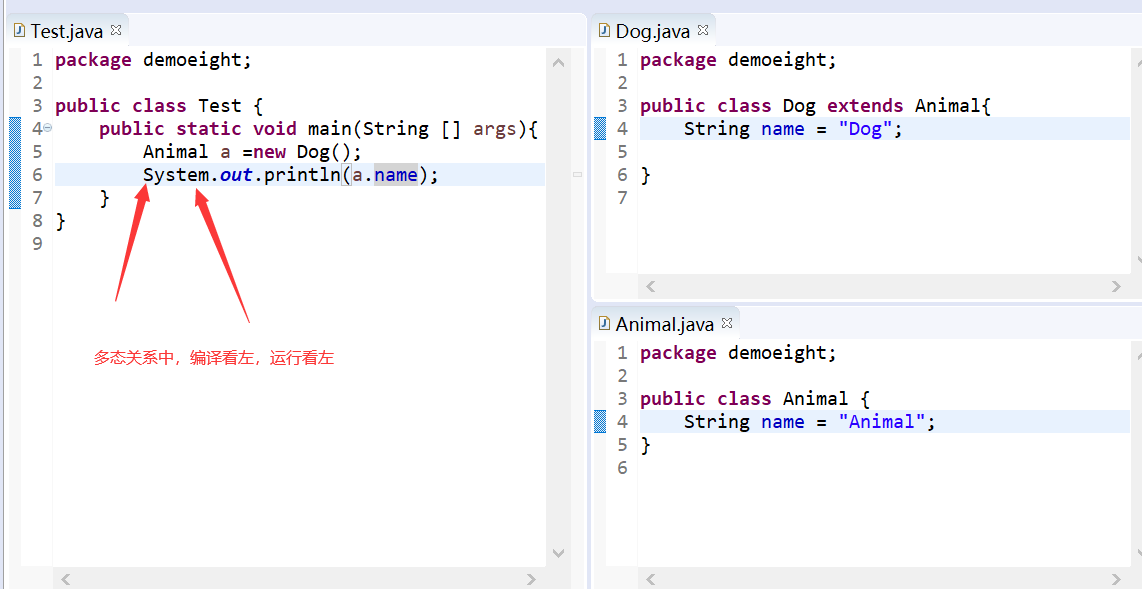
要有方法的重写

父类引用指向子类对象（is a）关系

多态运行成员方法编译看左（看父类有没有这个方法），运行看右（运行子类的方法）



多态运行成员变量编译看左（看父类有没有这个变量），运行看左（运行父类的变量）



多态的好处

可维护性，提高代码的复用性

可拓展性，把不同子类的对象当作父类看待，疲弊了不同子类对象间的差异

弊端：不能使用子类特有的成员

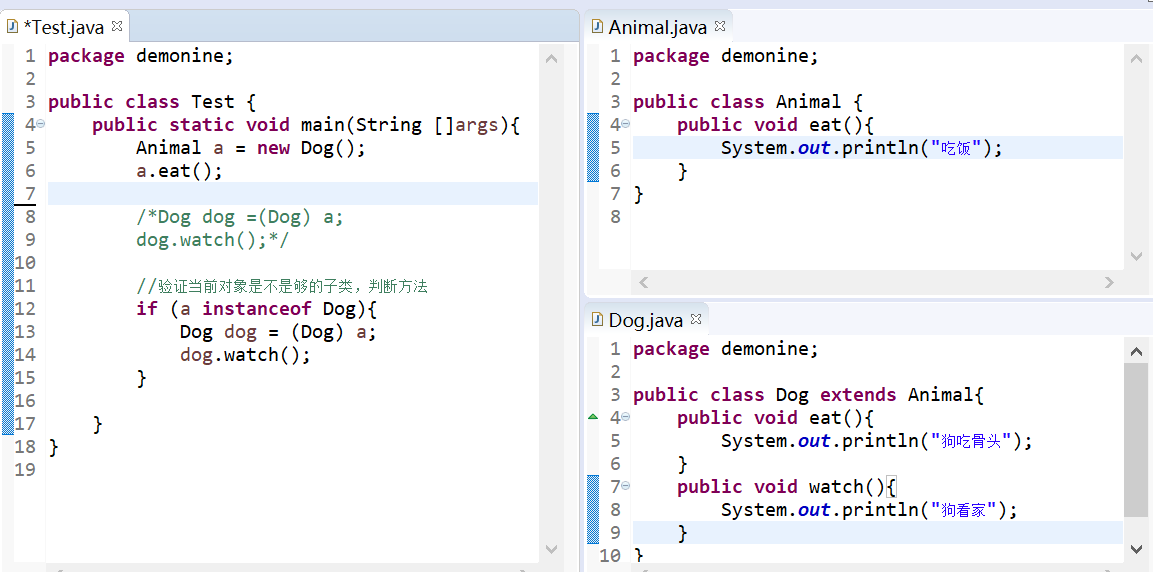
类型转换：向上转型——子类型转换成父类型

Animal animal = new dog();

向下转型——父类型转换成子类型

Dog dog = (Dog) animal

PS:只能在继承层次内进行转换；将父类换成子类之前，用instanceof进行检查



抽象类

abstract类和abstract方法 （不能够实例化 不能生成对象 有属性方法可以）

用关键字abstract修饰的类称为abstract类（抽象类）

如 abstract class A（）{

}

抽象类中有抽象方法

用关键字abstract修饰的方法称为abstract方法（抽象方法）

如 Abstract int min(int x,int y); 没有方法体 没有大括号

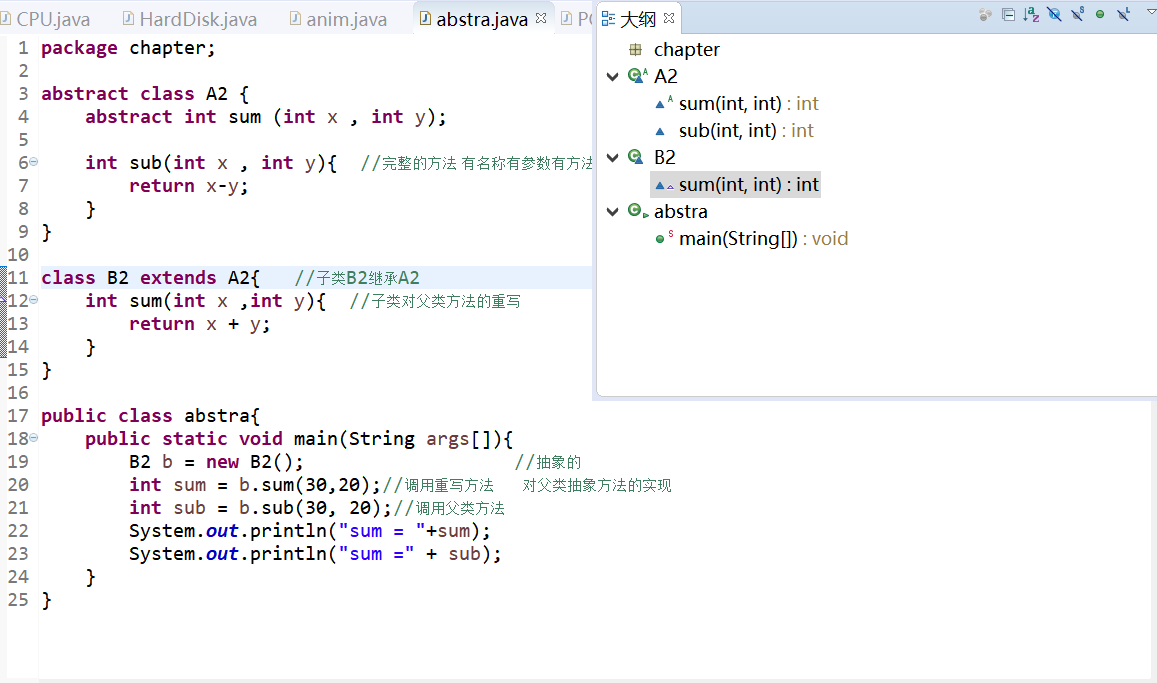
抽象类可包含抽象方法、实例方法

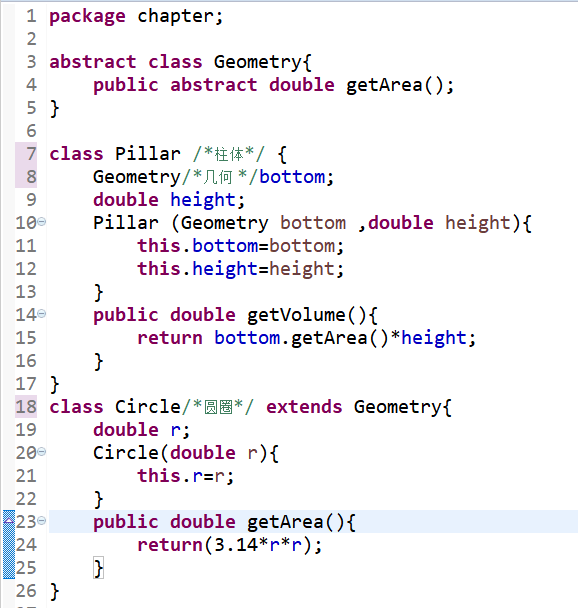
作用：对问题的抽象，对步骤不太精确，需要子类精确实现

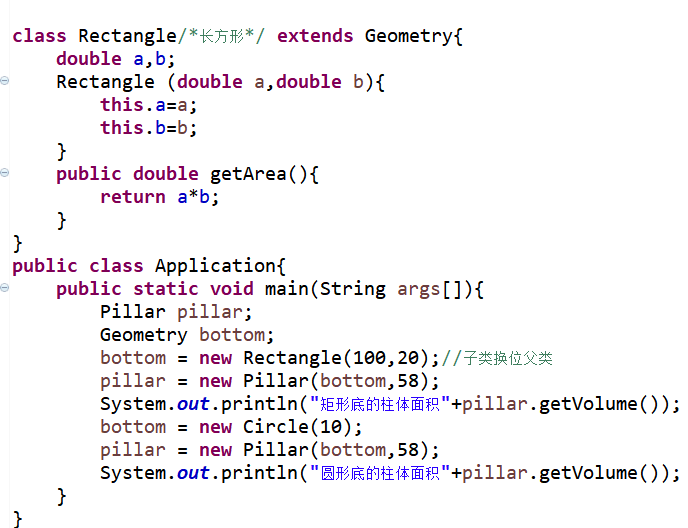
面向抽象编程： 抽象类与接口比较相似，抽象类可以包含实例方法，接口实现的子类相隔比较远，抽象都比较高，接口方法全为抽象方法



调用方法根据类的层次调用 根据对象先找 有的话直接调用 然后再去找父类 子类方法和父类方法若名字相同 先调用子类







Pillar无法创建出其他类型底的柱体，改pillar类不满足用户的类型需求

final关键字

概念：最终的、最后的

作用：用于修饰类、方法和变量，该类不能继承（即该类不能有子类）

修饰的方法不能被重写（父类可以不被任意篡改）且不与

abstrate互存，因为abstract要有子类重写，修饰变量，即常 量，只能赋值一次



Static 关键字

概念：静态的

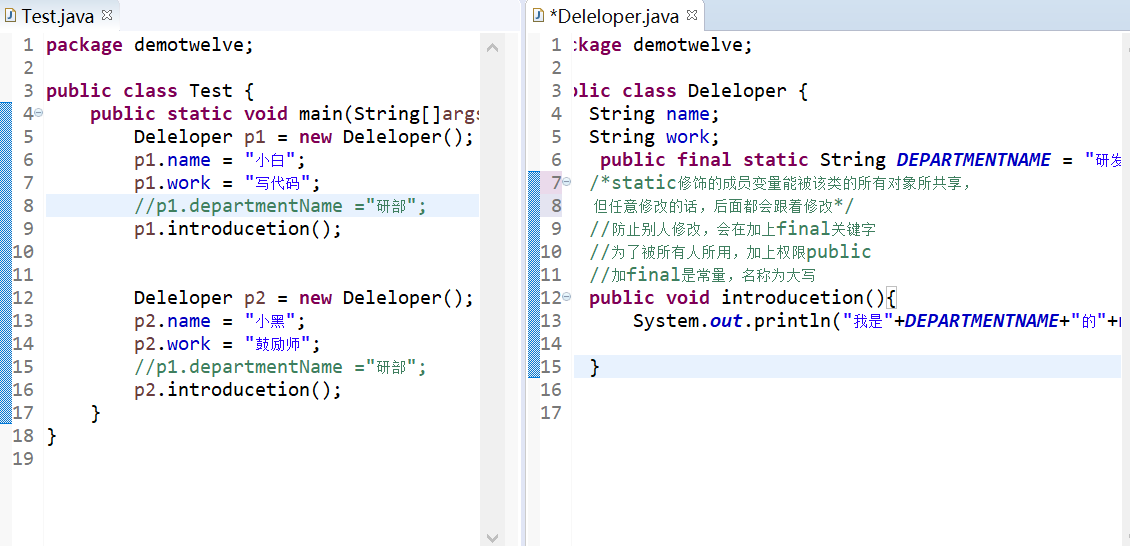
作用：用于修饰成员变量——类变量

用于修饰成员方法——类方法

调用方式：类名.成员变量名

类名.成员方法名()

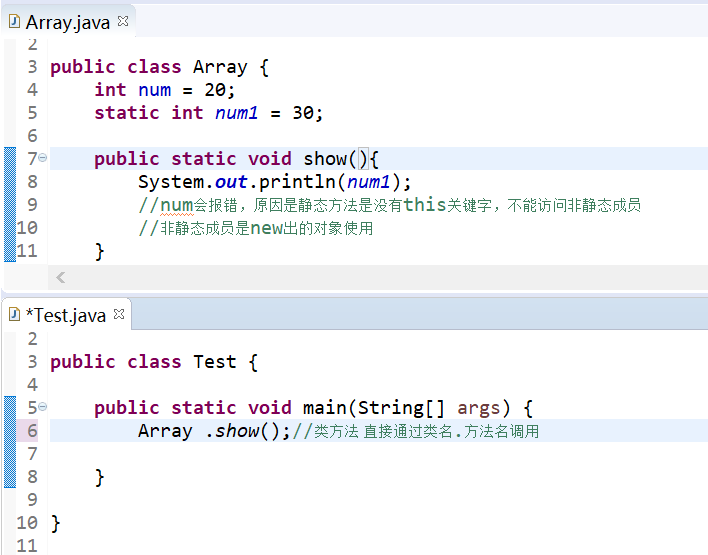
特点：能够被本类所有对象共享



Static修饰成分方法：静态方法中没有this，不能访问非静态成员

静态方法使用场景： 只需要访问静态成员

不需要访问静态对象，所需参数都由参数列表提供显示



上转型对象

对象a是父类的声明对象，对象b是子类的实例化对象，将b的引用赋值给a，这时，称对象a是b的上转型对象

示例代码：

Animal a;

a = new Tiger();

或者

Animal a;

Tiger b = new Tiger();

a=b;

上转型对象不能操作子类新增的属性和方法，但是可以访问子类继承和隐藏的成员变量

上转型对象可以访问子类继承或隐藏的成员变量，也可以调用子类继承的方法或子类重写的实例方法。上转型对象操作子类继承的方法或子类重写的实例方法，其作用等价于子类对象去调用这些方法。因此，如果子类重写了父类的某个实例方法后，当对象的上转型对象调用这个实例方法时一定是调用了子类重写的实例方法。

如果子类重写了父类的静态变量或者方法，那么子类对象的上转型对象调用重写了的静态方法或者静态变量时，调用的是父类的静态方法和静态变量（需要特别注意）