第6课

面向对象(下)

SOFTCITS@2016

SoftCITS SOFT CULTURE IT SALON 软件园IT文化沙龙

抽象类的例子

复习抽象类的特性

通过下面的例子再次理解抽象类的必要性

示例代码 6.7

接口

跟抽象类相比，接口(interface)是一种更彻底的抽象形式

接口概念和特征

- 接口相当于定义一套规范(比如USB, PCI插槽)
- 这套规范可由一个或多个类共同遵守
- 接口只定义规范，要求某些类去遵守去提供某些方法，但接口并不关心类实现这些方法的细节 (只有Java8以后允许有默认方法)

接口的意义

体现了规范和实现分离的设计哲学

让软件系统的各个组件、模块之间面向接口耦合，是一种松耦合设计

接口的定义和使用

接口定义

[修饰符] interface 接口名 [extends 父接口1, 父接口2, 父接口3...] {

  常量定义

  抽象方法定义

  内部类、枚举...

  默认方法

}

类中实现接口

[修饰符] class 类名 [extends 父类] implements 接口1, 接口2... {

  类体部分

}

代码示例 6.8.1

接口的定义和使用

注意事项

- 一个接口的java文件中只能有一个接口，并且文件名必须与接口名相同
- Java8以后才可以在接口中定义default方法
- 接口中除了default默认方法，其他都是抽象方法（虽然并没有添加abstract关键字）
- 接口里只能定义常量，不能定义普通的成员变量
- 接口里的方法都是public的，类里面实现的接口方法也必须是public
- 一个类实现某个接口时，它就得到了该接口中定义的常量、方法等
- 一个类实现某个接口时，必须实现该接口中全部抽象方法
- 一个类可以同时实现多个接口，而且这个类的引用可以直接复制给这些接口类型，相当于模拟了多继承
- 一个类可以继承父类，同时去实现多个接口，语法上必须extends在前，implements在后
- 接口和抽象类一样，都不能被实例化

接口与抽象类的差别

- 接口是系统或模块与外界交互的窗口，接口体现的是一种规范

  对于实现者而言，接口规定了必须向外部提供哪些服务

  对于调用者而言，接口规定了可以使用哪些服务，如何调用这些服务
- 接口类似于系统的“总纲”，它制定了模块间甚至整个系统应该遵循的标准。因此一个系统的接口不应该经常改变，否则对整个系统甚至其他外部系统都有影响，可能很多东西都需要改写
- 抽象类体现的是一种模板的设计模式，是为了控制子类们必须要提供哪些功能

设计模式介绍

设计模式

GOF23重设计模式是一套被广泛接受、反复使用的优秀代码设计经验的总结

设计模式充分考虑了复用性、可靠性和扩展性，而且使代码更容易被他人理解

设计模式是架构设计的精华，是软件大厦的基石脉络

模板模式

一个巧妙利用了抽象类的设计模式

特点

定义一个操作中的算法的骨架，而将步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义算法的某些特定步骤

模式中的角色

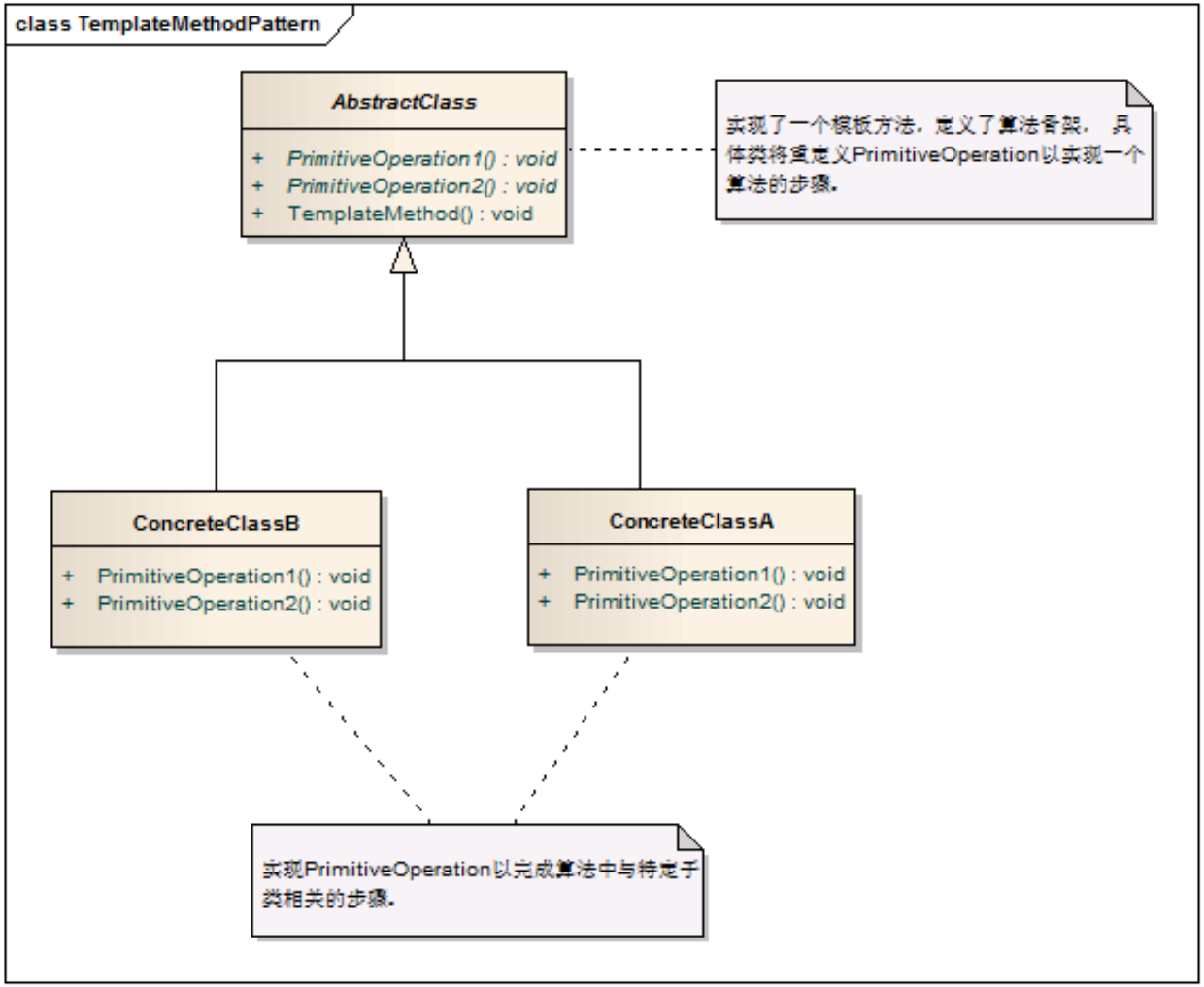
抽象类（AbstractClass）：实现了模板方法，定义了算法的骨架

具体类（ConcreteClass)：实现抽象类中的抽象方法，已完成完整的算法

模式基础代码 6.9.1

模板模式

UML图

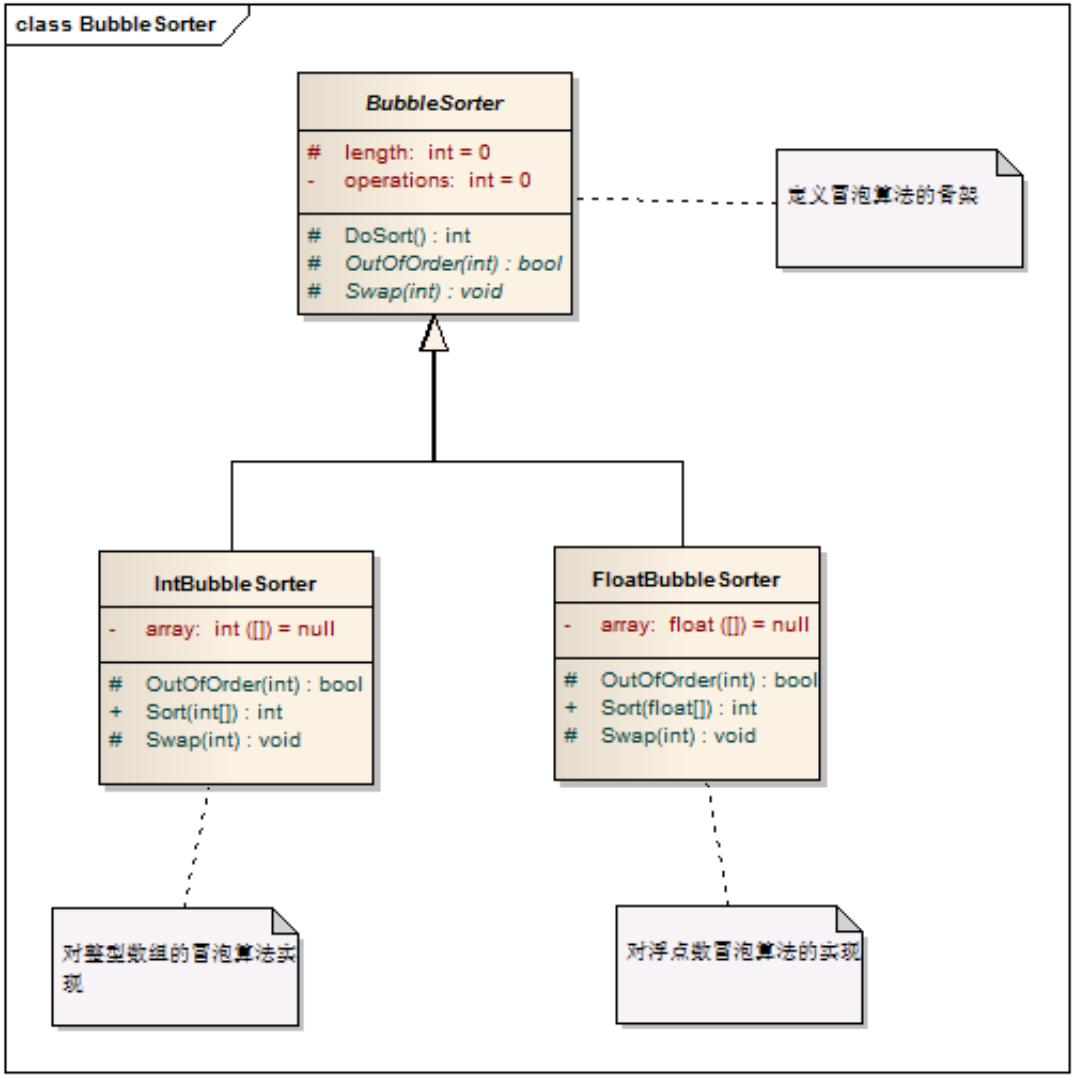

class TemplateMethodPattern
AbstractClass
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void
+ TemplateMethod() : void
实现了一个模板方法，定义了算法骨架，具体类将重定义PrimitiveOperation以实现一个算法的步骤。
ConcreteClassB
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void
ConcreteClassA
+ PrimitiveOperation1() : void
+ PrimitiveOperation2() : void
实现PrimitiveOperation以完成算法中与特定子类相关的步骤。

模板模式

实际应用例子

用冒泡算法非别对整型数组、浮点数数组、日期型数组... 实现排序

模式基础代码 6.9.2



模板模式

优点

模板方法模式通过把不变的行为搬移到超类，去除了子类中的重复代码

子类实现算法的某些细节，有助于算法的扩展

缺点

每个不同的实现都需要定义一个子类，这会导致类的个数的增加

适用场景

在某些类的算法中，用了相同的方法，造成代码的重复

控制子类扩展，子类必须遵守算法规则

简单工厂模式

简单工厂模式并不属于23种设计模式之一，是一种简化的模式。这里想通过简单工厂模式的一个片段让大家更好地理解**面向接口编程**

再回头看看示例代码6.8.1，想象着把Printer类换成VGA类(显卡)，然后思考下面的场景：
一家公司生产电脑，ComputerA类需要组合一个显示卡，有两个选择：VGA类和Output，采取哪种更好？
如果让ComputerA类组合一个VGA类，如果有一天产品升级，换更好的显卡BetterVGA，就不得不对ComputerA类进行修改。
如果只有ComputerA类组合了VGA类还好，一旦其他系列的产品ComputerB、ComputerC...都组合了VGA类呢？所有都需要修改

为了避免这个问题，工厂模式建议让ComputerA类组合一个Output类型对象，将ComputerA类与VGA类分离。 ComputerA类组合的对象到底是VGA还是BetterVGA， ComputerA类并不关心，它只知道那是“显卡”，具体什么型号则不关心。此时，当把VGA换成BetterVGA时， ComputerA类本身并不需要修改
示例代码 6.9.3

内部类

某些情况下，允许把一个类放在另外一个类的内部定义，这就是内部类，也称为嵌套类

这种做法会给内部类提供更好的封装，例如Cow类，需要组合一个CowLeg对象。CowLeg类只有在Cow类中才有意义，而且预计不会再有其他类要使用CowLeg类。此时就可以把CowLeg类定义成Cow类的内部类，不允许其他类去访问CowLeg类。

示例代码 6.9.4 Cow

访问规则

内部类可以直接访问外部类的私有成员，因为内部类本身就是外部类的类成员

外部类如果想访问内部类的成员，则必须通过内部类的对象才能够访问

内部类

假如外部类成员和内部类成员同名了，该怎么办？

如何使用this来解决？

示例代码 6.9.4 DiscernVariable

内部类

静态内部类

即用static修饰的内部类

#因为内部类本身就是外部类的成员，所以可以用static修饰，外部类则不可以

示例代码 6.9.4 StaticInnerClassTest

静态内部类的访问方法

示例代码 6.9.4 AccessStaticInnerClass （了解内容）

内部类

匿名内部类

匿名内部类适合于创建仅需要使用一次的类

继承父类添加方法

示例代码 6.9.4 Anonymous

创建实现接口的类

示例代码 6.9.4 AnonymousTest（了解内容）

抽象类和接口的Final练习

匿名内部类

参考Ex_6_1的代码，完成下面的问题

1. 在Ccircle类中添加适当代码，使程序执行结果如下：

球心: (8,6)

半径: 2

球体积: 33.4933333333333333

#注意：球体积计算公式为$4/3*\pi*r^3$

2. 在抽象类Csphere并没有看到接口iVolume中定义的两个方法，猜猜为什么编译不会出错？

谢谢观看

SeeYou！

SoftCITS 软件园IT文化沙龙

SOFT CULTURE IT SALON