实验五 适配器模式的应用

1 实验目的

1) 掌握适配器模式（Adapter）的特点

2) 分析具体问题，使用适配器模式进行设计。

2 实验内容和要求

一个软件团队开发绘图系统，设计了圆对象(Circle)、矩形对象(Rectangle)，线对象(Line)都支持Draw()函数，即可以通过Draw()函数绘制图形。为了加快项目进度，将角度对象(Angle)绘制功能交给了合作团队实现。但合作团队将角度对象绘制函数定为了DrawAngle()。绘图系统提供给用户后，用户不满意，希望能统一的调用，不用记太多命令。应用适配器模式，用C#或Java控制台应用程序完善该设计。绘制该模式的UML图。

3 UML图



图5.3.1 类图

4 代码实现

interface View {  
 void draw();  
}

public class Rectangle implements View{  
 @Override  
 public void draw() {  
 System.*out*.println(this.getClass().getSimpleName()+"");  
 }  
}

public class Line implements View{  
 @Override  
 public void draw() {  
 System.*out*.println(this.getClass().getSimpleName()+"");  
 }  
}

public class Circle implements View{  
  
 @Override  
 public void draw() {  
 System.*out*.println(this.getClass().getSimpleName()+"");  
 }  
}

public class Angle {  
 void drawAngle(double angle){  
 System.*out*.println(this.getClass().getSimpleName()+""+angle);  
 }  
}

public class AngleAdapter extends Angle{  
 private Angle angle;  
  
 public AngleAdapter(Angle angle) {  
 this.angle = angle;  
 }  
  
 public void draw(double radian) {  
 angle.drawAngle(radian\*180);  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 new Line().draw();  
 new Circle().draw();  
  
 AngleAdapter angleAdapter=new AngleAdapter(new Angle());  
 angleAdapter.draw(1.0\*1/3);  
 }  
}

**测试结果**

Line

Circle

Angle60.0

5 要点总结

1. 适配器模式属于结构型模式，它作为两个不兼容接口之间的桥梁，结合了两个独立接口的功能，将一个类的接口转换成另外一个接口使得原本由于接口不兼容而不能一起工作的类可以一起工作。
2. 类适配器模式 ：适配器（Adapter）跟适配者（Adaptee）是一种继承关系
3. 对象适配器模式：适配器跟适配者是一种组合关系。
4. 这里不仅模拟了借口不一致的适配，也模拟了设计单位不一致的适配。

实验六 桥接模式的应用

1 实验目的

1) 掌握桥接模式（Bridge）的特点

2) 分析具体问题，使用桥接模式进行设计。

2 实验内容和要求

一个咖啡店可以提供大杯(JorumCoffee)、中杯(MediumCoffee)、小杯(SmallCoffee)的咖啡(Coffee)，为了满足不同用户的口味，在咖啡中可以添加牛奶(Milk)，或者糖(Sugar)，或者柠檬(Lemon)，提供给用户不同口味的组合，如大杯咖啡加牛奶，中杯咖啡加糖，小杯咖啡加柠檬，小杯咖啡加糖等。应用桥接模式，用C#或Java控制台应用程序实现该设计。绘制该模式的UML图。

3 UML图



图6.3.1 类图

4 代码实现

interface Taste {  
 void display();  
}

public class Sugar implements Taste{  
 @Override  
 public void display() {  
 System.*out*.println("糖");  
 }  
}

public class Milk implements Taste{  
 @Override  
 public void display() {  
 System.*out*.println("牛奶");  
 }  
}

public class Lemon implements Taste{  
 @Override  
 public void display() {  
 System.*out*.println("柠檬");  
 }  
}

public abstract class Coffee {  
 Taste taste;  
  
 void display(){  
 taste.display();  
 }  
  
 public void setTaste(Taste taste) {  
 this.taste = taste;  
 }  
}

public class SmallCoffee extends Coffee {  
 @Override  
 void display() {  
 System.*out*.print("小杯咖啡 配 ");  
 super.display();  
 }  
}

public class MediumCoffee extends Coffee{  
  
 @Override  
 void display() {  
 System.*out*.print("中杯咖啡 配 ");  
 super.display();  
 }  
}

public class JorumCoffee extends Coffee {  
 @Override  
 void display() {  
 System.*out*.print("大杯咖啡 配 ");  
 super.display();  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 Coffee coffee = new JorumCoffee();  
 coffee.setTaste(new Lemon());  
 coffee.display();  
 }  
}

**测试结果**

大杯咖啡 配 柠檬

5 要点总结

1. 桥接模式，将抽象部分与它的实现部分分离，使它们都可以独立地变化。