简单工厂模式

public class Operation

{

public static double getResult(double numberA, double numberB,

String operator)

{

double result = 0;

if ("+".equals(operator))

result = numberA + numberB;

else if ("-".equals(operator))

result = numberA - numberB;

else if ("\*".equals(operator))

result = numberA \* numberB;

else if ("/".equals(operator))

result = numberA / numberB;

return result;

}

}

public class Main

{

public static void main(String[] args)

{

InputStreamReader stdin = null;

BufferedReader bufferReader = null;

stdin = new InputStreamReader(System.in);

bufferReader = new BufferedReader(stdin);

try

{

System.out.print("请输入数字A:");

String numberA = bufferReader.readLine();

System.out.print("请选择运算符(+、-、\*、/):");

String operator = bufferReader.readLine();

System.out.print("请输入数字B:");

String numberB = bufferReader.readLine();

String result = String.valueOf(Operation.getResult(Double

.parseDouble(numberA), Double.parseDouble(numberB),

operator));

System.out.println("结果是:" + result);

}

catch (Exception ex)

{

System.out.println("您输入有误:" + ex.getMessage());

ex.printStackTrace();

}

}

}

*Operation*运算基类

public class Operation

{

private double numberA = 0;

private double numberB = 0;

public double getResult() throws Exception

{

double result = 0;

return result;

}

public double getNumberA()

{

return numberA;

}

public void setNumberA(double numberA)

{

this.numberA = numberA;

}

public double getNumberB()

{

return numberB;

}

public void setNumberB(double numberB)

{

this.numberB = numberB;

}

}

加减乘除类

public class OperationAdd extends Operation

{

public double getResult()

{

double result = 0;

result = getNumberA() + getNumberB();

return result;

}

}

public class OperationSub extends Operation

{

public double getResult()

{

double result = 0;

result = getNumberA() - getNumberB();

return result;

}

}

public class OperationMul extends Operation

{

public double getResult()

{

double result = 0;

result = getNumberA() \* getNumberB();

return result;

}

}

public class OperationDiv extends Operation

{

public double getResult() throws Exception

{

double result = 0;

if (getNumberB() == 0)

{

throw new Exception("除数不能为0");

}

result = getNumberA() / getNumberB();

return result;

}

}

简单运算工厂类

public class OperationFactory

{

public static Operation createOperation(String operate)

{

Operation oper = null;

if ("+".equals(operate))

oper = new OperationAdd();

else if ("-".equals(operate))

oper = new OperationSub();

else if ("\*".equals(operate))

oper = new OperationMul();

else if ("/".equals(operate))

oper = new OperationDiv();

return oper;

}

}

客户端类

{

public static void main(String[] args)

{

InputStreamReader stdin = null;

BufferedReader buffer = null;

stdin = new InputStreamReader(System.in);

buffer = new BufferedReader(stdin);

try

{

System.out.print("请输入数字A:");

double numberA = Double.parseDouble(buffer.readLine());

System.out.print("请选择运算符(+、-、\*、/):");

String operator = buffer.readLine();

System.out.print("请输入数字B:");

double numberB = Double.parseDouble(buffer.readLine());

Operation oper = OperationFactory.createOperation(operator);

oper.setNumberA(numberA);

oper.setNumberB(numberB);

System.out.println("结果是:" + oper.getResult());

}

catch (NumberFormatException e)

{

e.printStackTrace();

}

catch (IOException e)

{

e.printStackTrace();

}

}

}

策略模式

public class CashContext

{

CashSuper cashSuper;

public CashContext(CashSuper cashSuper)

{

this.cashSuper = cashSuper;

}

public CashContext(String type)

{

if ("正常收费".equals(type))

{

cashSuper = new CashNormal();

}

else if ("满300返100".equals(type))

{

cashSuper = new CashReturn(300, 100);

}

else if ("打8折".equals(type))

{

cashSuper = new CashRebate(0.8);

}

}

public double acceptCash(double money)

{

return cashSuper.acceptCash(money);

}

}

//客户端代码

public class Main

{

private static double total = 0;

public static void main(String[] args)

{

consume("正常收费", 1, 1000);

consume("满300返100", 1, 1000);

consume("打8折", 1, 1000);

System.out.println("总计:" + total);

}

public static void consume(String type, int num, double price)

{

CashContext cashContext = new CashContext(type);

double totalPrices = cashContext.acceptCash(num \* price);

total += totalPrices;

System.out.println("单价:" + price + " 数量:" + num + "合计:" + totalPrices);

}

}

装饰模式

//Person类（ConcreteComponent）

public class Person

{

private String name;

public Person()

{

}

public Person(String name)

{

this.name = name;

}

public void show()

{

System.out.println("装扮的" + name);

}

}

//服饰类（Decorator）

public class Finery extends Person

{

protected Person component;

public void decorate(Person component)

{

this.component = component;

}

public void show()

{

if (null != component)

{

component.show();

}

}

}

//具体服饰类（ConcreteDecorator）

public class TShirts extends Finery

{

public void show()

{

super.show();

System.out.println("大T恤");

}

}

public class BigTrouser extends Finery

{

public void show()

{

super.show();

System.out.println("大裤衩");

}

}

public class Sneakers extends Finery

{

public void show()

{

super.show();

System.out.println("破球鞋");

}

}

public class Suit extends Finery

{

public void show()

{

super.show();

System.out.println("西装");

}

}

public class Tie extends Finery

{

public void show()

{

super.show();

System.out.println("领带");

}

}

public class LeatherShoes extends Finery

{

public void show()

{

super.show();

System.out.println("皮鞋");

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

Person person = new Person("小菜");

System.out.println("第一种装扮：");

Sneakers pqx = new Sneakers();

BigTrouser kk = new BigTrouser();

TShirts dtx = new TShirts();

pqx.decorate(person);

kk.decorate(pqx);

dtx.decorate(kk);

dtx.show();

System.out.println("第二种装扮：");

LeatherShoes px = new LeatherShoes();

Tie ld = new Tie();

Suit xz = new Suit();

px.decorate(person);

ld.decorate(px);

xz.decorate(ld);

xz.show();

}

}

代理模式

//代理接口

public interface GiveGift

{

void giveDolls();

void giveFlowers();

void giveChocolate();

}

//追求者类

public class Pursuit implements GiveGift

{

SchoolGirl mm;

public Pursuit(SchoolGirl mm)

{

this.mm = mm;

}

public void giveDolls()

{

System.out.println(mm.name + " 送你洋娃娃");

}

public void giveFlowers()

{

System.out.println(mm.name + " 送你鲜花");

}

public void giveChocolate()

{

System.out.println(mm.name + " 送你巧克力");

}

}

//代理类

public class Proxy implements GiveGift

{

Pursuit gg;

public Proxy(SchoolGirl mm)

{

gg = new Pursuit(mm);

}

public void giveChocolate()

{

gg.giveChocolate();

}

public void giveDolls()

{

gg.giveDolls();

}

public void giveFlowers()

{

gg.giveFlowers();

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

SchoolGirl jiaojiao = new SchoolGirl();

jiaojiao.setName("李娇娇");

Proxy daili = new Proxy(jiaojiao);

daili.giveDolls();

daili.giveFlowers();

daili.giveChocolate();

}

}

工厂方法模式

//雷锋工厂

public interface IFactory

{

LeiFeng createLeiFeng();

}

//学雷锋的大学生工厂

public class UndergraduateFactory implements IFactory

{

public LeiFeng createLeiFeng()

{

return new Undergraduate();

}

}

//社区志愿者工厂

public class VolunteerFactory implements IFactory

{

public LeiFeng createLeiFeng()

{

return new Volunteer();

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

IFactory factory = new UndergraduateFactory();

LeiFeng student = factory.createLeiFeng();

student.buyRice();

student.sweep();

student.wash();

}

}

外观模式

//四个子系统的类

public class SubSystemOne

{

public void methodOne()

{

System.out.println("子系统方法1");

}

}

public class SubSystemTwo

{

public void methodTwo()

{

System.out.println("子系统方法2");

}

}

public class SubSystemThree

{

public void methodThree()

{

System.out.println("子系统方法3");

}

}

public class SubSystemFour

{

public void methodFour()

{

System.out.println("子系统方法4");

}

}

//外观类

public class Facade

{

SubSystemOne one;

SubSystemTwo two;

SubSystemThree three;

SubSystemFour four;

public Facade()

{

one = new SubSystemOne();

two = new SubSystemTwo();

three = new SubSystemThree();

four = new SubSystemFour();

}

public void methodA()

{

System.out.println("方法组A");

one.methodOne();

two.methodTwo();

four.methodFour();

}

public void methodB()

{

System.out.println("方法组B");

two.methodTwo();

three.methodThree();

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

Facade facade = new Facade();

facade.methodA();

facade.methodB();

}

}

建造者模式

//Product类—产品类，由多个部件组成

public class Product

{

List<String> parts = new ArrayList<String>();

public void add(String part)

{

parts.add(part);

}

public void show()

{

System.out.println("产品 创建----");

for (String part : parts)

{

System.out.println(part);

}

}

}

//Builder类—抽象建造者类，确定产品由两个部件PartA和PartB组成，并声明一个得到产品建造后结果的方法getResult

public abstract class Builder

{

public abstract void buildPartA();

public abstract void buildPartB();

public abstract Product getResult();

}

//ConcreteBuilder1类—具体建造者类

public class ConcreteBuilder1 extends Builder

{

private Product product = new Product();

public void buildPartA()

{

product.add("部件A");

}

public void buildPartB()

{

product.add("部件B");

}

public Product getResult()

{

return product;

}

}

// ConcreteBuilder2类—具体建造者类

public class ConcreteBuilder2 extends Builder

{

private Product product = new Product();

public void buildPartA()

{

product.add("部件X");

}

public void buildPartB()

{

product.add("部件Y");

}

public Product getResult()

{

return product;

}

}

//Director类—指挥类

public class Director

{

public void construct(Builder builder)

{

builder.buildPartA();

builder.buildPartB();

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

Builder builder1 = new ConcreteBuilder1();

Builder builder2 = new ConcreteBuilder2();

Director director = new Director();

director.construct(builder1);

director.construct(builder2);

Product product1 = builder1.getResult();

Product product2 = builder2.getResult();

product1.show();

product2.show();

}

}

观察者模式

//Subject类，可翻译为主题或抽象通知者，一般用一个抽象类或一个接口实现。它把所有对观察者对象的引用保存在一个聚集里面，每个主题都可以有任何数量的观察者。抽象主题提供一个接口，可以拉回和删除观察者对象。

public abstract class Subject

{

private List<Observer> observers = new ArrayList<Observer>();

public void attach(Observer observer)

{

observers.add(observer);

}

public void detach(Observer observer)

{

observers.remove(observer);

}

public void announce()

{

for (Observer obj : observers)

{

obj.update();

}

}

}

//Observer类，抽象观察者，为所有的具体观察者定义一个接口，在得到主题的通知时更新自己，这个接口叫做更新接口。抽象观察者一般用一个抽象类或者一个接口实现。更新接口通常包含一个update()方法，这个方法叫做更新方法。

public abstract class Observer

{

public abstract void update();

}

//ConcreteSubject类，叫做具体主题或具体通知者，将有关状态存入具体观察者对象，在具体主题的内部状态改变时，给所有登记过的观察者发出通知。具体主题角色通常用一个具体子类实现。

public class ConcreteSubject extends Subject

{

// 具体被观察者状态

private String subjectState;

public String getSubjectState()

{

return subjectState;

}

public void setSubjectState(String subjectState)

{

this.subjectState = subjectState;

}

}

//ConcreteObserver类，具体观察者，实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。具体观察者角色可以保存一个指向具体主题对象的引用。具体观察者角色通常用一个具体子类实现。

public class ConcreteObserver extends Observer

{

private String name;

private String observerState;

private ConcreteSubject subject;

public ConcreteObserver(String name, ConcreteSubject subject)

{

this.name = name;

this.subject = subject;

}

public void update()

{

observerState = subject.getSubjectState();

System.out.println("观察者" + name + "的新状态是" + observerState);

}

public ConcreteSubject getSubject()

{

return subject;

}

public void setSubject(ConcreteSubject subject)

{

this.subject = subject;

}

}

//客户端代码

public class Main

{

public static void main(String[] args)

{

ConcreteSubject s = new ConcreteSubject();

s.attach(new ConcreteObserver("X", s));

s.attach(new ConcreteObserver("Y", s));

s.attach(new ConcreteObserver("Z", s));

s.setSubjectState("ABC");

s.announce();

}

}

结果显示：

观察者X的新状态是ABC

观察者Y的新状态是ABC

观察者Z的新状态是ABC