计算机学院 软件工程 课程实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：设计模块 | | 学号：201818130200 |
| 日期：2021年5月30日星期日 | 班级： 计科二班 | 姓名： 韦世强 |
| 实验目的：  1. 培养利用UML工具的能力  2. 练习设计模式应用，能在项目设计中运用设计模式进行面向对象设计；  3. 持续进行项目“持续规划与设计”和“持续开发与集成”的DevCloud开发。  4. 记录项目及小组的工作进度。 | | |
| 实验步骤与内容：   1. 继续完成凤凰商城系统的持续规划设计和CD、CI过程：      1. 查找工厂模式，策略模式：   简单工厂：  https://pic4.zhimg.com/80/v2-77280b58e8bde3a816e8da8d8eaba51f_720w.jpg  public interface ITea {      // 沏茶      public void makeTea();  }  public class LongjingTea implements ITea {      @Override      public void makeTea() {          System.out.println("西湖龙井");      }  }  public class BiluochunTea implements ITea{      @Override      public void makeTea() {          System.out.println("碧螺春");      }  }  /\*  准备沏一杯茶，根据传入的参数决定不同的茶叶类型  \*/      public ITea prepareTea(String type){          ITea tea = null;          if(type.equals("longjing")){              tea = new LongjingTea();          }else if(type.equals("biluochun")){              tea = new BiluochunTea();          }          if(tea != null){              tea.makeTea();          }          return tea;      }  public class TeaFactory {      public ITea createTea(String type){          ITea tea = null;          if(type.equals("longjing")){              tea = new LongjingTea();          }else if(type.equals("biluochun")){              tea = new BiluochunTea();          }          if(tea != null){              tea.makeTea();          }          return tea;      }  }  public static void main(String[] args) {          TeaFactory teaFactory = new TeaFactory();          ITea tea = teaFactory.createTea("longjing");      }  工厂方法：  preview  preview  /\*\*   \* 生产茶叶的统一接口   \*/  public interface ITeaFactory {      // 生产茶叶      public ITea createTea();  }  public class LongjingTeaFactory implements ITeaFactory{      @Override      public ITea createTea() {          return new LongjingTea();      }  }  public class BiluochunTeaFactory implements ITeaFactory{      @Override      public ITea createTea() {          return new BiluochunTea();      }  }  public class FactoryMethodTest {      public static void main(String[] args) {          ITeaFactory factory = new LongjingTeaFactory();          factory.createTea();          factory = new BiluochunTeaFactory();          factory.createTea();      }  }  抽象工厂：    /\*\*   \* 冰箱   \*/  public interface IFridge {      // 冷藏      void coldStorage();  }  /\*\*   \* 洗衣机   \*/  public interface IWasher {      void wash();  }  public class MeideFridge implements IFridge{      @Override      public void coldStorage() {          System.out.println("美的冰箱");      }  }  public class MeideWasher implements IWasher {      @Override      public void wash() {          System.out.println("美的洗衣机");      }  }  public class GeliFridge implements IFridge {      @Override      public void coldStorage() {          System.out.println("格力冰箱");      }  }  public class GeliWasher implements IWasher{      @Override      public void wash() {          System.out.println("格力洗衣机");      }  }  /\*\*   \* 抽象工厂接口，家用电器工厂，生产冰箱和洗衣机   \*/  public interface IHouseholdElectricFactory {      IFridge createFridge();      IWasher createWasher();  }  public class MeideHouseholdFactory implements IHouseholdElectricFactory{      @Override      public IFridge createFridge() {          return new MeideFridge();      }      @Override      public IWasher createWasher() {          return new MeideWasher();      }  }  public class GeliHouseholdFactory implements IHouseholdElectricFactory{      @Override      public IFridge createFridge() {          return new GeliFridge();      }      @Override      public IWasher createWasher() {          return new GeliWasher();      }  }  public class AbsFactoryTest {      public static void main(String[] args) {          IHouseholdElectricFactory factory = new MeideHouseholdFactory();          factory.createFridge().coldStorage();          factory.createWasher().wash();      }  }  策略模式  public interface FlyBehavior {      public void fly();  }  public class FlyWithWings implements FlyBehavior{      @Override      public void fly() {          System.out.println("用翅膀飞");      }  }  public class FlyNoWay implements FlyBehavior {      @Override      public void fly() {          System.out.println("不会飞");      }  }  public interface QuackBehaivor {      public void quack();  }  public class Quack implements QuackBehaivor {      @Override      public void quack() {          System.out.println("呱呱叫");      }  }  public class MuteQuack implements QuackBehaivor {      @Override      public void quack() {          System.out.println("不会叫");      }  }  public abstract class Duck {     FlyBehavior flyBehavior;     QuackBehaivor quackBehavior;       public abstract void display();       public void performFly() {         flyBehavior.fly();     }       public void performQuack() {         quackBehavior.quack();     }       public void swim() {         System.out.println("所有的鸭子都会漂浮");     }     // 动态设置飞行行为     public void setFlyBehavior(FlyBehavior flyBehavior) {         this.flyBehavior = flyBehavior;     }     //动态设置叫的行为     public void setQuackBehavior(QuackBehaivor quackBehavior) {         this.quackBehavior = quackBehavior;     }  }  public class MallardDuck extends Duck {      @Override      public void display() {          System.out.println("我是绿头鸭");      }  }  public class MiniDuckSimulator {        public static void main(String []args) {          Duck mallard = new MallardDuck();          mallard.setFlyBehavior(new FlyWithWings());//用翅膀飞          mallard.setQuackBehavior(new Quack());//呱呱叫            mallard.performFly();          mallard.performQuack();      }  } | | |