

**实验报告**

|  |  |
| --- | --- |
| **课程名称**: | 设计模式 |
| **开课学期**: | 2020-2021学年 第2学期 |
| **专业**: | 软件工程 |
| **班级年级**: | 2018级 |
| **学生姓名**: | 宋行健 |
| **学号:** | 222018321062006 |
| **实验教师:** | 王晓蒙 |

**计算机与信息科学学院 软件学院**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | | 实验1 工厂方法、抽象工厂 | | | |
| 实验时间 | | 2021/3/22 | Type | □验证性 \*设计性 □综合性 | |
| 1. **实验1 基础实验**   实验要求：  问题描述：销售公司（Company）根据订单购置一批轿车（Car），目前订单上只有A（CarA）和B（CarB）两种品牌的车辆。假设车辆主要由车架（Frame）、车轮（Wheel）和发动机（Engine）三个部件组成。（1）现用工厂方法设计程序，模拟订单处理流程，设计类图和完成代码实现；（2）任意添加一种品牌汽车，分析新程序与原程序之间得区别，从设计图和代码两方面分析（第2小问，只需分析，不需要设计类图和实现代码）。  思路：  根据工厂方法，我先建造了一个IFactory接口，并使用FactoryA、FactoryB两个类去实现这个接口。另外构建了一个抽象的Car类，通过CarA、CarB两个类进行实现。  在Company类中进行测试，使用一个Map进行工厂的储存，从txt文件中读取信息后通过IFactory进行Car的建造。  类图：    代码：   * Company.java   package CarCompany**;**  **import** java**.**io**.**File**;**  **import** java**.**io**.**FileInputStream**;**  **import** java**.**io**.**FileNotFoundException**;**  **import** java**.**util**.\*;**  public class Company**{**  public static void main**(**String**[]** args**){**  // 配置工厂信息  Map**<**String**,** IFactory**>** factoryMap **=** **new** HashMap**<**String**,** IFactory**>();**  factoryMap**.**put**(**"A"**,** **new** FactoryA**());**  factoryMap**.**put**(**"B"**,** **new** FactoryB**());**  // 车辆仓库  List**<**Car**>** list **=** **new** ArrayList**<**Car**>();**  // 输入订单，根据订单选择工厂进行生产  **try(**Scanner input **=** **new** Scanner**(new** FileInputStream**(new** File**(**"E:\\设计模式\\0\_实验报告\\实验2\\FactoryMethod\\order.txt"**)))){**  **while(**input**.**hasNext**())** **{**  String typeToCreate **=** input**.**next**();**  IFactory selectedFactory **=** factoryMap**.**get**(**typeToCreate**);**  list**.**add**(**selectedFactory**.**createCar**());**  **}**  **}** **catch(**FileNotFoundException e**){**  e**.**printStackTrace**();**  **}**  // 打印车辆信息  **for(**Car c**:**list**)** **{**  c**.**getInfo**();**  **}**  **}**  **}**   * IFactory.java   package CarCompany**;**  public interface IFactory **{**  Car createCar**();**  **}**   * FactoryA.java   package CarCompany**;**  public class FactoryA **implements** IFactory **{**  @Override  public Car createCar**()** **{**  **return** **new** CarA**(**"FrameA"**,** "WheelA"**,** "EngineA"**);**  **}**  **}**   * FactoryB.java   package CarCompany**;**  public class FactoryB **implements** IFactory **{**  @Override  public Car createCar**()** **{**  **return** **new** CarB**(**"FrameB"**,** "WheelB"**,** "EngineB"**);**  **}**  **}**   * Car.java   package CarCompany**;**  abstract public class Car **{**  String Frame**;**  String Wheel**;**  String Engine**;**  abstract void getInfo**();**  **}**   * CarA.java   package CarCompany**;**  public class CarA **extends** Car **{**  private final String Frame**;**  private final String Wheel**;**  private final String Engine**;**  public CarA**(**String frameA**,** String wheelA**,** String engineA**)** **{**  **this.**Frame **=** frameA**;**  **this.**Wheel **=** wheelA**;**  **this.**Engine **=** engineA**;**  **}**  public String getEngine**()** **{**  **return** Engine**;**  **}**  public String getFrame**()** **{**  **return** Frame**;**  **}**  public String getWheel**()** **{**  **return** Wheel**;**  **}**  public void getInfo**()** **{**  String Info **=** "-- CarA: " **+** Frame **+** " " **+** Wheel **+** " " **+** Engine**;**  System**.**out**.**println**(**Info**);**  **}**  **}**   * CarB.java   package CarCompany**;**  public class CarB **extends** Car **{**  private final String Frame**;**  private final String Wheel**;**  private final String Engine**;**  public CarB**(**String frameB**,** String wheelB**,** String engineB**)** **{**  **this.**Frame **=** frameB**;**  **this.**Wheel **=** wheelB**;**  **this.**Engine **=** engineB**;**  **}**  public String getEngine**()** **{**  **return** Engine**;**  **}**  public String getFrame**()** **{**  **return** Frame**;**  **}**  public String getWheel**()** **{**  **return** Wheel**;**  **}**  public void getInfo**()** **{**  String Info **=** "-- CarB: " **+** Frame **+** " " **+** Wheel **+** " " **+** Engine**;**  System**.**out**.**println**(**Info**);**  **}**  **}**  运行结果：   |  |  | | --- | --- | | 图 1 order.txt文件 | 图 2 程序输出 |   总结：  该项目是一个工厂方法模式，满足开闭原则。定义了抽象类（Car）和接口（IFactory）。生产具体型号的Car，只需要添加相应的实体类和Factory类即可。对生产的汽车配置没有做限制，且添加型号时不需要改动原始模块的代码。  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  **2. 实验2角色动画**  思路：  首先是对讲义中MyApp.java的代码进行研究，理解每一句代码实现的功能，对代码进行分块，比如哪一块是构建什么的。将分块后的代码进行封装，使用要求的抽象工厂模式进行封装。  构建一个IFactory接口，使用EnemyFactory和PlayerFactory进行接口的实现。另外构建两个抽象类Building和Rider，然后使用EnemyBuilding和PlayerBuilding去实现Building这个抽象类，对于Rider抽象类的实现同理。这就构建了一个抽象工厂模式。至于传参，我在IFactory中留了一个Point2D类型的参数，作为构造Building和Rider的位置参数。至于图片的裁剪等操作都放在了具体的实现类中，简化了构建。  类图：    代码：  说明：因为Enemy和Player的代码类似，我在实验报告中只摘取Player的代码作为示例。   * IFactory.java   package factory**;**  **import** javafx**.**geometry**.**Point2D**;**  **import** javafx**.**scene**.**image**.**Image**;**  public interface IFactory **{**  Rider createRider**(**Point2D pos**);**  Building createBuilding**(**Point2D pos**);**  **}**   * PlayerFactory.java   package factory**;**  **import** game**.**ImageTool**;**  **import** javafx**.**geometry**.**Point2D**;**  **import** javafx**.**scene**.**image**.**Image**;**  public class PlayerFactory **implements** IFactory **{**  Image resource **=** **new** Image**(**"file:" **+** System**.**getProperty**(**"user.dir"**)** **+** "\\image\\res\\res01.png"**);**  @Override  public PlayerRider createRider**(**Point2D pos**)** **{**  Image**[]** images **=** **new** Image**[**3**];**  **for** **(**int i **=** 0**;** i **<** 3**;** i**++)** **{**  images**[**i**]** **=** ImageTool**.**clipImage**(**resource**,** 540 **+** i **\*** 110**,** 370**,** 110**,** 80**);**  **}**  **return** **new** PlayerRider**(**images**,** pos**);**  **}**  @Override  public PlayerBuilding createBuilding**(**Point2D pos**)** **{**  Image house **=** ImageTool**.**clipImage**(**resource**,** 600**,** 0**,** 240**,** 256**);**  **return** **new** PlayerBuilding**(**house**,** pos**);**  **}**  **}**   * PlayerRider.java   package factory**;**  **import** javafx**.**geometry**.**Point2D**;**  **import** javafx**.**scene**.**image**.**Image**;**  public class PlayerRider **extends** Rider **{**  public PlayerRider**(**Image**[]** imgs**,** Point2D pos**)** **{**  **super(**imgs**,** pos**);**  **}**  @Override  public void move**()** **{**  setCurrentIndex**((**getCurrentIndex**()** **+** 1**)** **%** getImage**().**length**);**  **if(**getPosition**().**getX**()** **>=** 800**)** **{**  setPosition**(**getPosition**().**add**(-**600**,** 0**));**  **}else** **{**  setPosition**(**getPosition**().**add**(**10**,** 0**));**  **}**  **}**  **}**   * PlayerBuiding.java   package factory**;**  **import** javafx**.**geometry**.**Point2D**;**  **import** javafx**.**scene**.**image**.**Image**;**  public class PlayerBuilding **extends** Building **{**  public PlayerBuilding**(**Image img**,** Point2D pos**)** **{**  **super(**img**,** pos**);**  **}**  **}**   * MyApp.java   package factory**;**  **import** game**.**GameLoopTimer**;**  **import** javafx**.**application**.**Application**;**  **import** javafx**.**geometry**.**Point2D**;**  **import** javafx**.**scene**.**Group**;**  **import** javafx**.**scene**.**Scene**;**  **import** javafx**.**scene**.**canvas**.**Canvas**;**  **import** javafx**.**scene**.**canvas**.**GraphicsContext**;**  **import** javafx**.**scene**.**image**.**Image**;**  **import** javafx**.**stage**.**Stage**;**  public class MyApp **extends** Application **{**  @Override  public void start**(**Stage primaryStage**)** **throws** Exception **{**  // 绘制JavaFX基本框架  Group group **=** **new** Group**();**  Scene scene **=** **new** Scene**(**group**,** 1200**,** 512**);**  Canvas gameCanvas **=** **new** Canvas**();**  GraphicsContext gc **=** gameCanvas**.**getGraphicsContext2D**();**  gameCanvas**.**widthProperty**().**bind**(**scene**.**widthProperty**());**  gameCanvas**.**heightProperty**().**bind**(**scene**.**heightProperty**());**  group**.**getChildren**().**add**(**gameCanvas**);**  primaryStage**.**setScene**(**scene**);**  // 录入地图  Image mapImage **=** **new** Image**(**"file:" **+** System**.**getProperty**(**"user.dir"**)** **+** "\\image\\maps\\map03.jpg"**);**  // 创建工厂  PlayerFactory playerFactory **=** **new** PlayerFactory**();**  EnemyFactory enemyFactory **=** **new** EnemyFactory**();**  // 调用工厂中的方法创建实例  Building playerBuilding **=** playerFactory**.**createBuilding**(new** Point2D**(**0**,** 0**));**  Building enemyBuilding **=** enemyFactory**.**createBuilding**(new** Point2D**(**944**,** 256**));**  Rider playerRider **=** playerFactory**.**createRider**(new** Point2D**(**200**,** 200**));**  Rider enemyRider1 **=** enemyFactory**.**createRider**(new** Point2D**(**600**,** 300**));**  Rider enemyRider2 **=** enemyFactory**.**createRider**(new** Point2D**(**630**,** 270**));**  Rider enemyRider3 **=** enemyFactory**.**createRider**(new** Point2D**(**630**,** 330**));**  // 时间计时器循环  GameLoopTimer timer **=** **new** GameLoopTimer**()** **{**  @Override  public void inputProcess**()** **{**  **}**  @Override  public void logicUpdate**()** **{**  playerRider**.**move**();**  enemyRider1**.**move**();**  enemyRider2**.**move**();**  enemyRider3**.**move**();**  **}**  @Override  public void displayUpdate**()** **{**  gc**.**clearRect**(**0**,** 0**,** 1200**,** 512**);**  gc**.**drawImage**(**mapImage**,** 0**,** 0**);**  gc**.**drawImage**(**playerBuilding**.**getImage**(),** playerBuilding**.**getPosition**().**getX**(),** playerBuilding**.**getPosition**().**getY**());**  gc**.**drawImage**(**enemyBuilding**.**getImage**(),** enemyBuilding**.**getPosition**().**getX**(),** enemyBuilding**.**getPosition**().**getY**());**  gc**.**drawImage**(**playerRider**.**getCurrentImage**(),** playerRider**.**getPosition**().**getX**(),** playerRider**.**getPosition**().**getY**());**  gc**.**drawImage**(**enemyRider1**.**getCurrentImage**(),** enemyRider1**.**getPosition**().**getX**(),** enemyRider1**.**getPosition**().**getY**());**  gc**.**drawImage**(**enemyRider2**.**getCurrentImage**(),** enemyRider2**.**getPosition**().**getX**(),** enemyRider2**.**getPosition**().**getY**());**  gc**.**drawImage**(**enemyRider3**.**getCurrentImage**(),** enemyRider3**.**getPosition**().**getX**(),** enemyRider3**.**getPosition**().**getY**());**  **}**  **};**  timer**.**start**();**  primaryStage**.**setResizable**(false);**  primaryStage**.**show**();**  **}**  public static void main**(**String**[]** args**)** **{**  launch**(**args**);**  **}**  **}**  运行结果：    总结：  本次实验复习了JavaFX的基本结构和抽象工厂模式，并将抽象工厂模式应用到实际的项目中，得到了更加深刻的理解。  通过研读老师提供的参考代码，先将代码分块，然后将代码块进行封装。通过构建一个IFactory接口进行工厂的构建，使用EnemyFactory和PlayerFactory进行接口的实现。另外构建两个抽象类Building和Rider，然后使用EnemyBuilding和PlayerBuilding去实现Building这个抽象类，对于Rider抽象类的实现同理。这就构建了一个抽象工厂模式。至于传参，我在IFactory中留了一个Point2D类型的参数，作为构造Building和Rider的位置参数。至于图片的裁剪等操作都放在了具体的实现类中，简化了构建。在最后，我构建了3个EnemyRider对象，并通过修改位置，使他们排成队列进行行军。通过传入每个士兵位置参数Point2D进行创建，十分方便。  实验中也遇到了一些问题，例如在开始时走了一些弯路，我为了对代码进行封装，将位置参数也封装进了实体类，这使得在工厂建造的时候不能有效控制对象的位置，建造多个对象的时候十分不方便，后来留了Point2D参数，使得构建不同位置的对象更加方便快捷。 | | | | | |
| 教师评阅 | 实验内容（60%）： | | | |  |
| 实验思路（20%）： | | | |  |
| 实验总结（20%）： | | | |  |
| 实验成绩： | | | | |