软件设计模式课程项目总结报告

——摩尔庄园

**小组成员**

1952455 刘思远

1951120 罗 检

1951121 宁之恒

1951574 吴 渭

1952107 王子轩

1952221 杨一迪

1952377 蒋汶霖

1952540 万惟佳

1952635 凌亚楠

1952910 裴元昊

报告撰写人：蒋汶霖、罗检

撰写完成日期：2020年11月5日

**指导老师**

**冯巾松 侯捷**

目录

[1 **项目简介** 4](#_Toc87023186)

[2**设计模式汇总** 6](#_Toc87023187)

[3 **设计模式详述** 10](#_Toc87023188)

[**摩尔庄园——基础框架** 10](#_Toc87023189)

[**3.1 单例模式** 10](#_Toc87023190)

[**3.2 解释器模式** 11](#_Toc87023191)

[**3.3 组合模式** 12](#_Toc87023192)

[**3.4 黑板模式** 13](#_Toc87023193)

[**3.5 延迟初始化模式** 14](#_Toc87023194)

[**3.6 MVC模式** 15](#_Toc87023195)

[**摩尔庄园——农场** 17](#_Toc87023196)

[**3.7 工厂模式** 18](#_Toc87023197)

[**3.8 命令模式** 19](#_Toc87023198)

[**3.9 适配器模式** 20](#_Toc87023199)

[**3.10 观察者模式** 21](#_Toc87023200)

[**3.11 状态模式** 22](#_Toc87023201)

[**3.12 代理模式** 23](#_Toc87023202)

[**3.13 责任链模式** 24](#_Toc87023203)

[**3.14 建造者模式** 25](#_Toc87023204)

[**3.15 迭代器模式** 26](#_Toc87023205)

[**摩尔庄园——商店** 27](#_Toc87023206)

[**3.16 抽象工厂模式** 27](#_Toc87023207)

[**3.17 桥接模式** 28](#_Toc87023208)

[**3.18 装饰器模式** 29](#_Toc87023209)

[**3.19 过滤器模式** 31](#_Toc87023210)

[**3.20 策略模式** 32](#_Toc87023211)

[**3.21 访问者模式** 33](#_Toc87023212)

[**摩尔庄园——游乐园** 35](#_Toc87023213)

[**3.22 外观模式** 35](#_Toc87023214)

[**3.23 享元模式** 37](#_Toc87023215)

[**3.24 模板模式** 38](#_Toc87023216)

[**3.25 备忘录模式** 40](#_Toc87023217)

[**3.26 原型模式** 42](#_Toc87023218)

[**摩尔庄园——聊天室** 43](#_Toc87023219)

[**3.27 中介者模式** 43](#_Toc87023220)

[**3.28 互斥模式** 44](#_Toc87023221)

[**3.29 多例模式** 45](#_Toc87023222)

[4 **项目分工说明与成员贡献** 47](#_Toc87023223)

## 1 **项目简介**

本项目选题名称为摩尔庄园。该项目以一款已有的同名社区养成类游戏为原型，以现实生活中的城市为背景，虚构了一个梦幻、美好的虚拟世界。作为一款角色扮演类游戏，玩家将扮演摩尔角色在世界进行购物、娱乐、生产、交流等日常活动。

项目在原有摩尔庄园的基础上进行删减，保留了4个核心板块:商店、游乐园、农场和聊天室（见下图）。用户运行摩尔庄园游戏后，需要先为摩尔挑选初始属性。完成角色创建后将进入到整个系统的主菜单界面。小摩尔就可以正式开始体验庄园里丰富的板块内容啦。

在摩尔庄园的农场中，小摩尔将亲自体验播种与收获的快乐，你需要娴熟运用铲子、锄头和水壶等农具进行农耕；同时，面对影响作物健康的异常状态（如干旱、虫灾等），小摩尔也需要发挥才智，利用各种工具解决问题。当作物培育成熟，我们便可以将其送入仓库，并选择卖给商店，用自己的辛勤劳动成果换取摩尔豆！

在农场忙碌后当然要买买买来释放压力，商店模块为摩尔提供了丰富的商品，摩尔可以购买各种衣物装扮，可以购买香甜可口的食物……

闲暇之余，游乐园为摩尔提供了休闲的好去处。想要体验琴棋书画的风雅，摩尔将移步到井字棋小游戏；想要一展厨房风采，摩尔可步进做饭小游戏；想要拥抱速度与激情，摩尔能前往赛车小游戏。更多小游戏，等待您的探索。

仅仅只玩个人游戏是不是还不满足呢？摩尔庄园还为各位玩家提供了聊天室区域。在这里你可以和其它小摩尔一起畅所欲言，无话不谈。试着在聊天室里找到那个心仪的ta吧。

本游戏意在倡导健康、快乐、创造、分享的内涵，将带给用户良好的游戏体验，追忆儿时的回忆。感谢您游玩摩尔庄园，并成为庄园的一份子，为庄园的蓬勃发展贡献一份力。



## 2**设计模式汇总**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 设计模式 | 简介 | 源文件 | 备注说明 |
| 摩尔庄园——基础框架 | | | | |
| 1 | 单例模式  Singleton | 摩尔庄园在整个项目仅有唯一实例 | MoleManor.java | 23个经典设计模式之一 |
| 2 | 解释器模式  Interpreter | 小摩尔持有一定数量的钱，购买门票才能游玩摩尔庄园的游乐分区，对购买多少张门票扣除摩尔账户里多少钱进行解释。 | Expression.java  Mul.java  Number.java  Sub.java | 23个经典设计模式之一 |
| 3 | 组合模式  Composite | 菜单选项中的单选项以及二级菜单，模拟了文件与文件夹的组合模式 | Component.java  Menu.java  MenuList.java  MenuOption.java | 23个经典设计模式之一 |
| 4 | 黑版模式  Blackboard | 在框架端摩尔庄园的公告板采用此模式 | Blackboard.java  Control.java  Notice.java | 《设计模式之禅》作者: 秦小波 出版社: 机械工业出版社 ISBN: 9787111437871 |
| 5 | 延迟初始化模式  Lazy Initialization | 摩尔庄园各模块的加载，在用户选择进入后才进行 | MoleManor.java | 《Beginning Scala》 Author:Pollak, David ISBN:9781430219897. |
| 6 | MVC模式  Model-View-Controller | 背包不仅只有外观，还有它内部物品的增添的控制。  在MVC设计模式下，我们将背包分为Model-View-Controller三个部分，使得各个模块相对功能独立。 | Backpack.java  BackpackController.java  BackpackView.java | 《设计模式之禅》作者: 秦小波 出版社: 机械工业出版社 ISBN: 9787111437871 |
| 摩尔庄园——农场 | | | | |
| 7 | 工厂模式  Factory | 农场中种子、作物、肥料的创建操作统一由它们对应的具体工厂完成。 | SeedFactory.java  FertilizerFactory.java  Factory.java | 23个经典设计模式之一 |
| 8 | 命令模式Command | 请求以命令的形式包裹在对象中，将农场仓库对种子、肥料、作物的买卖操作封装起来。 | SeedCommand.java  FertilizerCommand.java  Command.java | 23个经典设计模式之一 |
| 9 | 适配器模式Adapter | 基础框架端提供关于天气相关的信息，对于农场无法直接使用，使用该模式进行适配。 | WeatherAdapter.java  Target.java | 23个经典设计模式之一 |
| 10 | 观察者模式Observe | 天气情况的变化时，由于所有农田块的干湿状态依赖于天气，它们将得到通知并被自动更新状态。 | WeatherObserver.java | 23个经典设计模式之一 |
| 11 | 状态模式  State | 农田块将根据自身当前干湿状态，决定被浇水时执行的相应动作。 | InsectDamageWeather.java  RanWeather.java  Context.java | 23个经典设计模式之一 |
| 12 | 代理模式  Proxy | 摩尔发出买卖种子、肥料的请求，由仓库代理去完成该请求。 | Proxy.java | 23个经典设计模式之一 |
| 13 | 职责链  Chain of Responsibility | 先从仓库中找寻种子。缺少种子时进一步，由仓库向商店要货。 | ShopHandler.java  WareHouseHandler.java  Handler.java | 23个经典设计模式之一 |
| 14 | 建造者模式Builder | 农场为用户提供了方便的一键操作。程序将用户播种、浇水、施肥等流程进行了封装 | Director.java  Builder.java  ConcreteBuilder1.java  ConcreteBuilder2.java | 23个经典设计模式之一 |
| 15 | 迭代器模式  Iterator | 用户查看农田状态时将会对农田块进行迭代遍历，隐藏对象的内部表示。 | Iterator.java  FarmIterator.java | 23个经典设计模式之一 |
| 摩尔庄园——商店 | | | | |
| 16 | 抽象工厂模式  Abstract Factory | 抽象工厂模式用于产生服装工厂，食品工厂；具体工厂再通过相应的抽象产品接口生成具体服装，种子和食品。 | AbstractClothes.java  AbstractFactory.java  ICommodity.java | 23个经典设计模式之一 |
| 17 | 桥接模式  Bridge | 服装的样式是多种的，服装的颜色也是多种的，将样式与颜色进行桥接。 | ClothesColor.java  Blue.java  Red.java  Yellow.java | 23个经典设计模式之一 |
| 18 | 装饰器模式Decorator | 通过口味装饰器对食品店的汉堡进行装饰，使得摩尔可以选择购买加了不同口味小料的汉堡。 | CondimentDercorator.java  Beef.java  Codfish.java  Shrimp.java | 23个经典设计模式之一 |
| 19 | 过滤器模式  Filter | 通过过滤器接口以及价格过滤器用于商场服装店按价格区间筛选服装。 | IFilter.java  priceFilter.java | 《Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML》Author：Mark Grand ISBN：0471449334, 9780471449331 |
| 20 | 策略模式Strategy | 对服装购买的结账使用正常付款、返利、打折三种策略。 | SaleSelect.java  CashContext.java  CashNormal.java  CashRebate.java | 23个经典设计模式之一 |
| 21 | 访问者模式Visitor | 对服装的操作包括购买和租赁两种操作，操作进行访问，被访问者是结账时的购物车。 | ClothesVisitor.java  ClothesPurchaseVisitor.java  ClothesLeaseVisitor.java | 23个经典设计模式之一 |
| 摩尔庄园——游乐园 | | | | |
| 22 | 外观模式  Facade | 定义了一个游戏的高层接口，子系统中包括井字棋游戏、做饭小游戏、赛车游戏。 | Game.java  GameMaker.java | 23个经典设计模式之一 |
| 23 | 享元模式  Flyweight | 棋子工厂是用来获得棋子对象实例的，如果缓存中存在棋子对象实例，则从缓存中读取获得，否则生成一个新对象存入缓存后返回。 | TictactoeFactory.java  BlackChess.java  WhiteChess.java | 23个经典设计模式之一 |
| 24 | 模板模式  Template | 漂移赛、障碍赛、竞速赛均包括检票、进行游戏、计算得分的过程，故而将这三个过程抽象成模板。 | AbstractRacing.java  DriftRace.java  ObstacleRace.java  TimeRace.java | 23个经典设计模式之一 |
| 25 | 备忘录模式  Memento | 将摩尔在游乐园所玩过的游戏名称、所获积分存入备忘录。 | RecordList.java  RecordMemento.java  ScoreOriginator.java | 23个经典设计模式之一 |
| 26 | 原型模式  Prototype | 做饭小游戏提供三道菜：宫保鸡丁、酸菜鱼、糖醋里脊，小摩尔可能重复做菜，采用原型设计模式对菜肴进行克隆，提高系统的性能。 | AbstractMeal.java  GongBaoJiDing.java  SuanCaiYu.java  TangCuLiJi.java | 23个经典设计模式之一 |
| 摩尔庄园——聊天室 | | | | |
| 27 | 中介者模式  Mediator | 多个用户可以向聊天室发送消息，聊天室向所有的用户显示消息。聊天室的角色就是中介者。 | ChatRoom.java | 23个经典设计模式之一 |
| 28 | 互斥模式  Mutex | 只有一个NPC能进入聊天室添加消息，给聊天室加上了锁。 | Lock.java  Mutex.java | 《Design Patterns in Java》 Author: Steven John Metsker， William C. Wake ISBN：0321629949, 9780321629944 |
| 29 | 多例模式  Multipleton | 聊天室有三个NPC和用户端参与，使用枚举类型实现多例模式创建了3个NPC：菩提大伯，么么公主，瑞琪。 | NPCList.java  Robot.java | 《Apex Design Patterns》 Author：Jitendra Zaa, Anshul Verma ISBN：1782173668, 9781782173663 |

## 3 **设计模式详述**

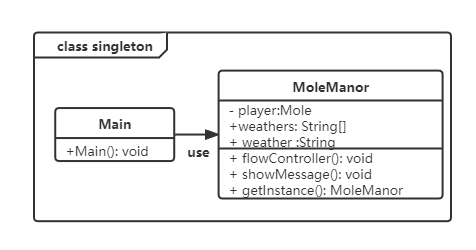
### **摩尔庄园——基础框架**

### **3.1 单例模式**

#### **3.1.1 实现API描述**

|  |  |
| --- | --- |
| API | MoleManor.getInstance () |
| 函数说明 | 获取摩尔庄园单例 |
| 参数 | 无 |
| 返回值 | MoleManor |

#### **3.1.2 类图**



#### **3.1.3 模式简介与应用**

单例模式是 Java 中最简单的设计模式之一。这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。摩尔庄园本身是独一无二的实例。在整个设计模式项目中存在且仅存在一个摩尔庄园。

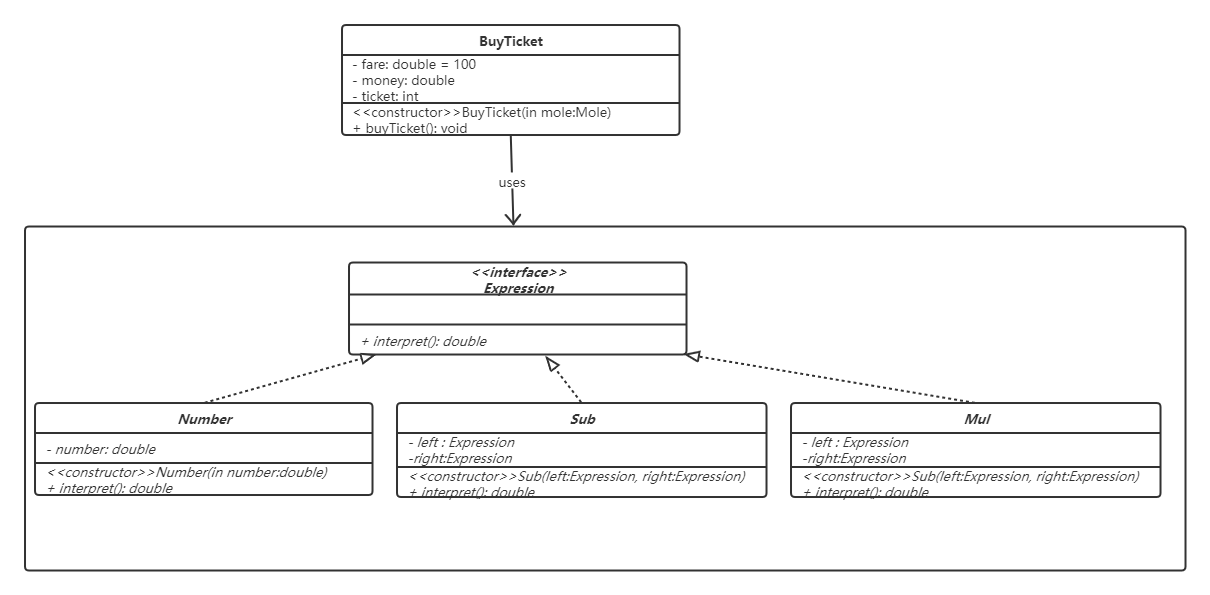
### **3.2 解释器模式**

#### **3.2.1 实现API描述**

|  |  |
| --- | --- |
| API | Expression.interpret () |
| 函数说明 | 表达式的解析过程，该接口的各实现类的解析规则各不相同 |
| 参数 | 无 |
| 返回值 | Double数据类型的解析结果 |

|  |  |
| --- | --- |
| API | BuyTicket.buyTicket () |
| 函数说明 | 用真实的购买门票场景测试解释器设计模式 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.2.2 类图**



#### **3.2.3 模式简介与应用**

小摩尔持有一定数量的钱，购买门票才能游玩摩尔庄园的游乐分区，对购买多少张门票扣除摩尔账户里多少钱进行解释。

创建一个表达式接口Expression，创建实现了上述接口的实体类Number、Sub、Mul，BuyTicket 使用Expression类来创建规则，并解析它们。

摩尔购买门票的事件发生的频率足够高，所以可能就值得将该事件的各个实例表述为一个简单语言中的句子。这样就可以构建一个解释器，该解释器通过解释这些句子来解决该问题。也就是把购买门票的数量Number、扣除账户金额的减法Sub、门票数量与门票单价之间的乘法Mul表述为简单语言中的句子，构建解释器。

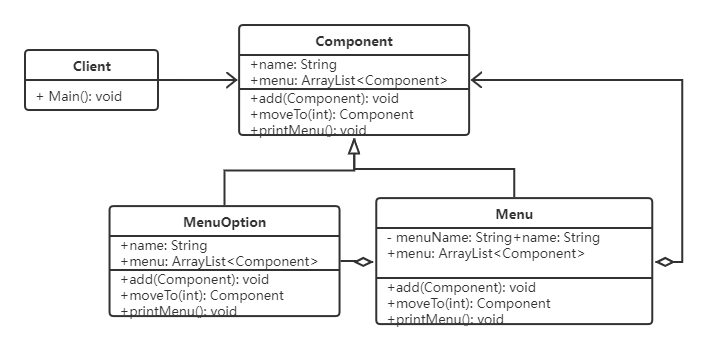
### **3.3 组合模式**

#### **3.3.1 实现API描述**

|  |  |
| --- | --- |
| API | Component.add (Component[ ]) |
| 函数说明 | 添加新的菜单选项 |
| 参数 | Component数组 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | Component.printMenu () |
| 函数说明 | 打印菜单 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.3.2 类图**



#### **3.3.3 模式简介与应用**

MenuOption和Menu两个类都继承Component类以体现菜单选项中的单选项以及二级菜单，模拟了文件与文件夹的方式。在组合模式类图中MenuOption中表示叶节点对象，叶节点没有子节点，Menu表示非叶节点，类比文件系统中的文件夹，二者共同继承Component后，将复杂且相似的对象组合在一起，有效防止在对外表现时的杂乱无章，方便对同一组对象进行管理与控制。

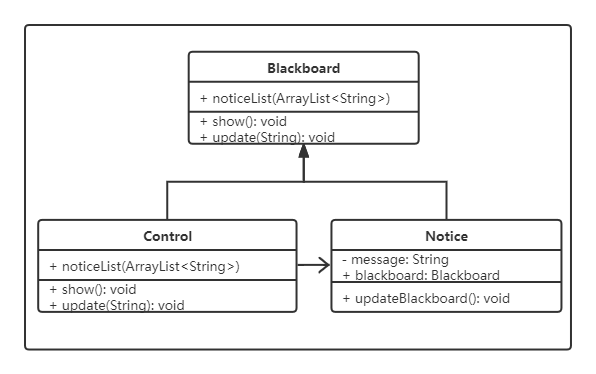
### **3.4 黑板模式**

#### **3.4.1 实现API描述**

|  |  |
| --- | --- |
| API | Blackboard.show () |
| 函数说明 | 展示黑板信息 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | Blackboard.update (String) |
| 函数说明 | 添加新公告 |
| 参数 | String |
| 返回值 | 空 |

#### **3.4.2 类图**



#### **3.4.3 模式简介与应用**

黑板类（Blackboard）是一个类，该类主要实现存储摩尔庄园内的公告和展示公告，还具有加入新公告的功能。

控制类（Control）是用于控制公告源类（Notice），主要控制公告源的加入和公告的添加。公告源类用于提供新的公告。

#### **3.4.4 出处**

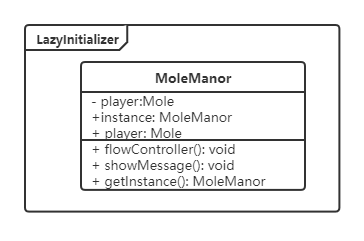
《设计模式之禅》作者: 秦小波 出版社: 机械工业出版社 ISBN: 9787111437871

### **3.5 延迟初始化模式**

#### **3.5.1 实现API描述**

|  |  |
| --- | --- |
| API | MoleManor.flowConctoller () |
| 函数说明 | 根据用户选择延迟加载对应模块 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.5.2 类图**



#### **3.5.3 模式简介与应用**

由于摩尔庄园各模块自身都较为庞大，如果程序在一开始就将所有模块一次性加载完毕，会导致过大的内存开销，可能会降低用户体验，因此在用户选择了需要进入的模块后，程序才会创建对应的实例。

#### **3.5.4 出处**

《Beginning Scala》 Author:Pollak, David ISBN:9781430219897.

### **3.6 MVC模式**

#### **3.6.1 实现API描述**

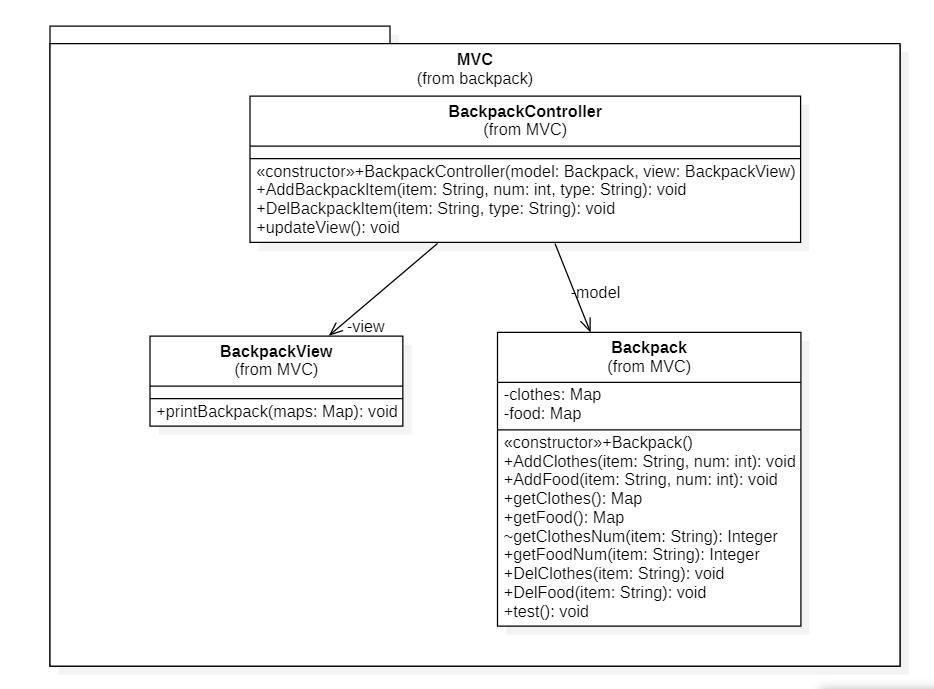
|  |  |
| --- | --- |
| API | BackpackView.printBackpack () |
| 函数说明 | 视图输出当前背包所含的模型信息 |
| 参数 | 衣服或者食物的键值对集合 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | BackpackController.updateView () |
| 函数说明 | 控制器更新背包视图 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | BackpackController.AddBackpackItem () |
| 函数说明 | 控制器增加背包内物品 |
| 参数 | 增加的物品名，数量，类型（衣服或者食物） |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | BackpackController.DelBackpackItem () |
| 函数说明 | 控制器删除背包内物品 |
| 参数 | 删除的物品名，类型（衣服或者食物） |
| 返回值 | 空 |

#### **3.6.2 类图**



#### **3.6.3 模式简介与应用**

小摩尔的背包内含许多物品，或是衣服，或是食物，或是一件，或是好几件。背包不仅只有外观，还有它内部物品的增添的控制。

在MVC设计模式下，我们将背包分为Model-View-Controller三个部分，使得各个模块相对功能独立。

创建一个作为模型的 Backpack 对象。BackpackView 是一个把背包详细信息输出到控制台的视图类，BackpackController 是负责存储数据到 Backpack 对象中的控制器类，并相应地更新视图 BackpackView。

#### **3.5.4 出处**

《设计模式之禅》作者: 秦小波 出版社: 机械工业出版社 ISBN: 9787111437871

### **摩尔庄园——农场**

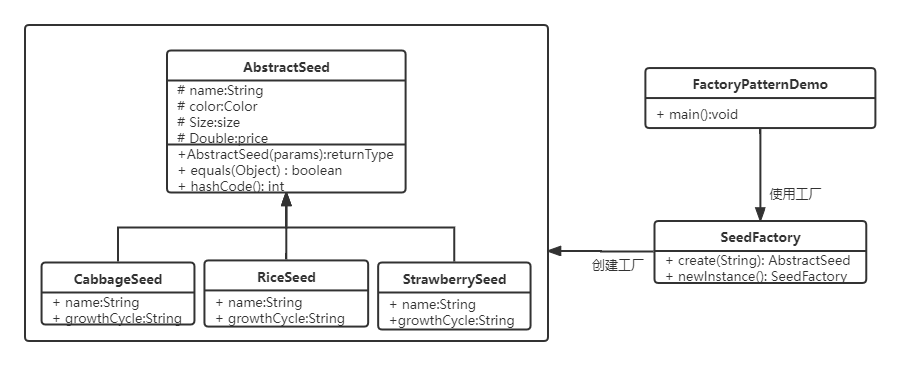
### **3.7 工厂模式**

#### **3.7.1 实现API描述**

|  |  |
| --- | --- |
| API | SeedFactory.newInstance () |
| 函数说明 | 获取种子工厂实例 |
| 参数 | 无 |
| 返回值 | SeedFactory |

|  |  |
| --- | --- |
| API | SeedFactory.creat (String) |
| 函数说明 | 利用工厂生产种子 |
| 参数 | String种子名称 |
| 返回值 | AbstractSeed具体种子对象 |

#### **3.7.2 类图**



#### **3.7.3 模式简介与应用**

工厂模式是 Java 中最常用的设计模式之一。当农场需要种子Seed、肥料Fertilizer等产品时，我们无需关心它们是如何生产出来的，只需要直接去工厂（SeedFactory、FertilizerFactory）提货即可。该模式的使用提高了程序的拓展性，若想要新增产品只要扩展一个工厂类就可以，并对调用者屏蔽了具体实现。

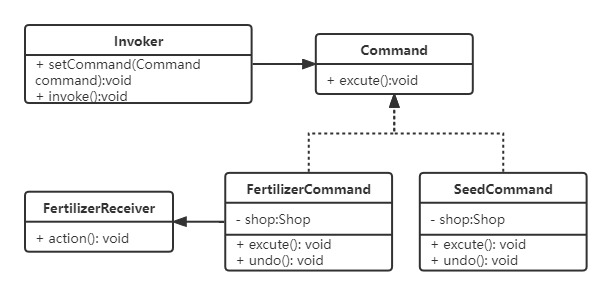
### **3.8 命令模式**

#### **3.8.1 实现API描述**

|  |  |
| --- | --- |
| API | FertilizerCommand.execute (String,int) |
| 函数说明 | 执行购买肥料命令 |
| 参数 | String种类，int数量 |
| 返回值 | Boolean是否购买成功 |

|  |  |
| --- | --- |
| API | SeedCommand.execute (String,int) |
| 函数说明 | 执行购买种子命令 |
| 参数 | String种类，int数量 |
| 返回值 | Boolean是否购买成功 |

#### **3.8.2类图**



#### **3.8.3 模式简介与应用**

该模式将请求以命令的形式包裹在对象中，并传给调用对象。调用对象寻找可以处理该命令的合适的对象，并把该命令传给相应的对象，由该对象执行命令。

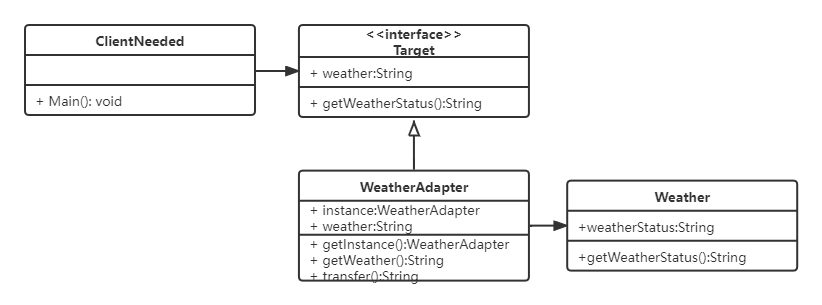
项目中摩尔可以发出购买种子、肥料的的命令，具体实现如下Command接口为所有命令声明了一个接口，FertilizerCommand和SeedCommand实现了该接口，调用者Invoker调用具体命令的excute方法就可以让对应命令的接收者执行相应动作。

### **3.9 适配器模式**

#### **3.9.1 实现API描述**

|  |  |
| --- | --- |
| API | Target.getWeather () |
| 函数说明 | 获取天气的接口 |
| 参数 | 无 |
| 返回值 | String天气情况 |

#### **3.9.2 类图**



#### **3.9.3 模式简介与应用**

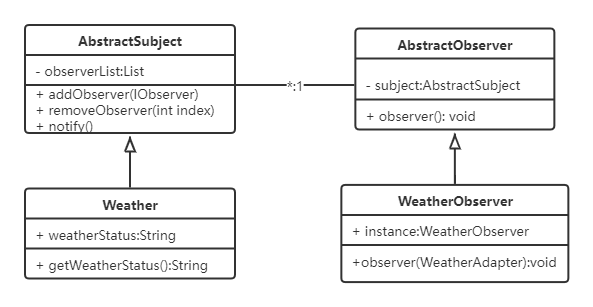
适配器模式作为两个不兼容的接口之间的桥梁，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作，它属于结构型模式。在本项目中，Target类是农场模块所期待的接口，Weather类是基础框架端提供的类，农场模块不能直接使用。通过在内部包装一个WeatherAdapter对象，把源接口转换成目标接口，从而农场模块可以使用框架端提供的天气。

### **3.10 观察者模式**

#### **3.10.1 实现API描述**

|  |  |
| --- | --- |
| API | WeatherObderver.observer (WeatherAdapter) |
| 函数说明 | 状态发生变化时调用观察者的更新函数 |
| 参数 | 天气 |
| 返回值 | 空 |

#### **3.10.2 类图**



#### **3.10.3 模式简介与应用**

当一个对象被修改时，如何自动通知依赖它的对象？观察者模式给出了解决方案，它降低了目标与观察者之间的耦合关系，在两者之间建立了一套触发机制，符合依赖倒置原则。

在具体实现里，每个观察者将会注册到目标的List链表里，调用notify()方法后将会依次调用每个观察它的Observer的observer方法，通知观察者自己发生的变化。项目中当天气状态发生变化时例如变为下雨，将通知每一个农田块，改变其干湿状态。

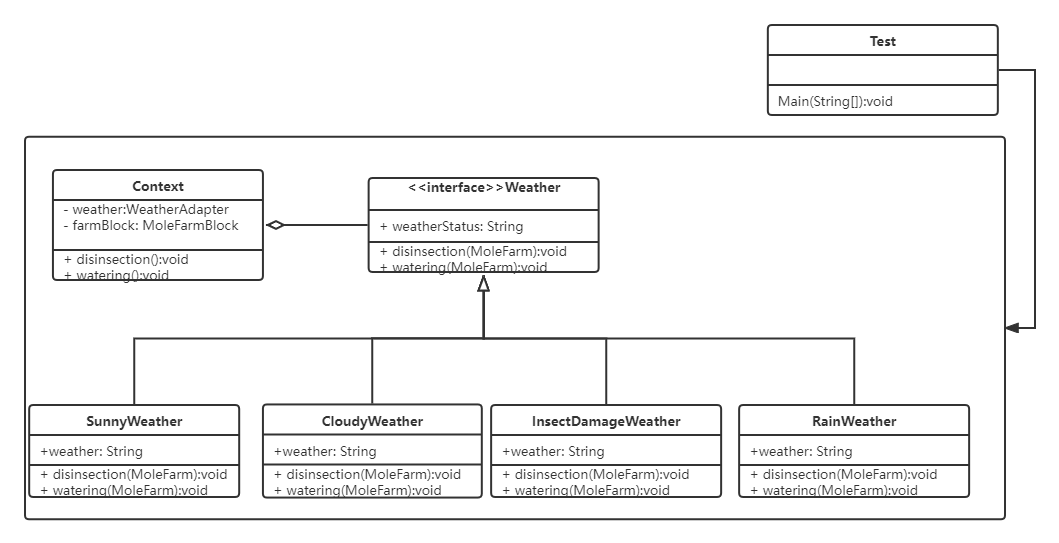
### **3.11 状态模式**

#### **3.11.1 实现API描述**

|  |  |
| --- | --- |
| API | Context.disInsection () |
| 函数说明 | 调用不同天气状态下的除虫方法 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | Context.watering () |
| 函数说明 | 调用不同天气状态下的浇水方法 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.11.2 类图**



#### **3.11.3 模式简介与应用**

在状态模式中，类的行为是基于它的状态改变的，我们需要允许对象在内部状态发生改变时改变它的行为。

农场将会面临不同的天气状态，包括SunnyWeather、CloudyWeather、InsectDamageWeather、RainWeather。在不同的天气状态下，农田块会具有不同的状态，在雨天农田块将不能被浇水，在虫灾天农田块可以被除虫。从而实现了不同状态下具有不同的行为。

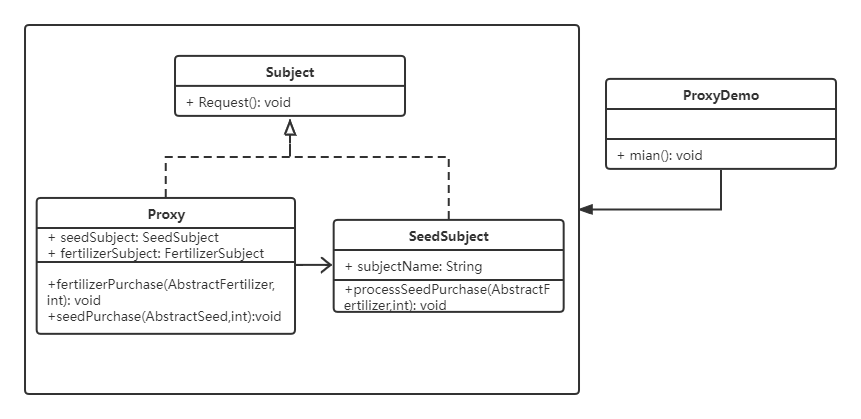
### **3.12 代理模式**

#### **3.12.1 实现API描述**

|  |  |
| --- | --- |
| API | Proxy.fertilizerPurchase (String,int) |
| 函数说明 | 让仓库代理去商店购买指定种类和数量的肥料 |
| 参数 | String种类，int数量 |
| 返回值 | boolean是否购买成功 |

|  |  |
| --- | --- |
| API | Proxy.seedPurchase (String,int) |
| 函数说明 | 让仓库代理去商店购买指定种类和数量的种子 |
| 参数 | String种类，int数量 |
| 返回值 | boolean是否购买成功 |

#### **3.12.2 类图**



#### **3.12.3 模式简介与应用**

由于某些原因直接访问某一对象会给使用者或者系统结构带来一些不便，该模式旨在为其他对象提供一种代理以控制对目标对象的访问。

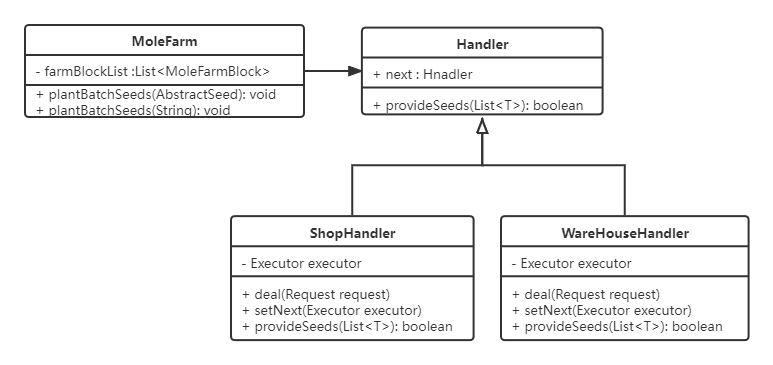
在本项目中，摩尔在农庄需要购买种子和肥料时，不是直接去执行操作，而是由仓库代理，由仓库去完成对应的操作，例如种子购买和肥料购买。

### **3.13 责任链模式**

#### **3.13.1 实现API描述**

|  |  |
| --- | --- |
| API | provideSeeds (List<T>) |
| 函数说明 | 比较List中需要播种的种子数量和仓库中已有种子数量，若不足则将请求传给商店购买新种子 |
| 参数 | 需要播种的种子列表List<T> |
| 返回值 | Boolean是否播种成功 |

#### **3.13.2 类图**



#### **3.13.3 模式简介与应用**

责任链模式为请求创建了一个接收者对象的链条，职责链上的处理者负责处理请求，客户只需要将请求发送到职责链上即可，无须关心请求的处理细节和请求的传递。请求将沿着这条链传递，直到有对象处理为止。

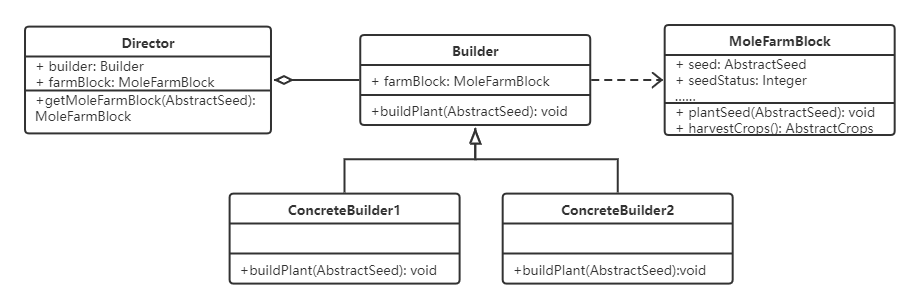
该模式实现了将请求的发送者和请求的处理者解耦的目标。项目中摩尔发出种植特定的作物种子的请求后，仓库将会处理其请求，即检查仓库中对应的种子数量是否充足，是则该请求在此处被处理完毕，否则将会发出购买种子的请求，传递给商店进行处理。商店会检查摩尔豆是否充足，是则完成购买，否则该请求在整个责任链上都无法完成，返回false。

### **3.14 建造者模式**

#### **3.14.1 实现API描述**

|  |  |
| --- | --- |
| API | Builder.buildPlant (AbstractSeed) |
| 函数说明 | 将给定的种子，按照生长过程进行建造 |
| 参数 | AbstractSeed种子 |
| 返回值 | 空 |

#### **3.14.2 类图**



#### **3.14.3 模式简介与应用**

该模式将一个复杂的构建拆分为多个简单的对象一步一步构建，这是由于某个复杂对象的各部分可能面临着剧烈的变化，但是将它们组合在一起的算法却相对稳定。

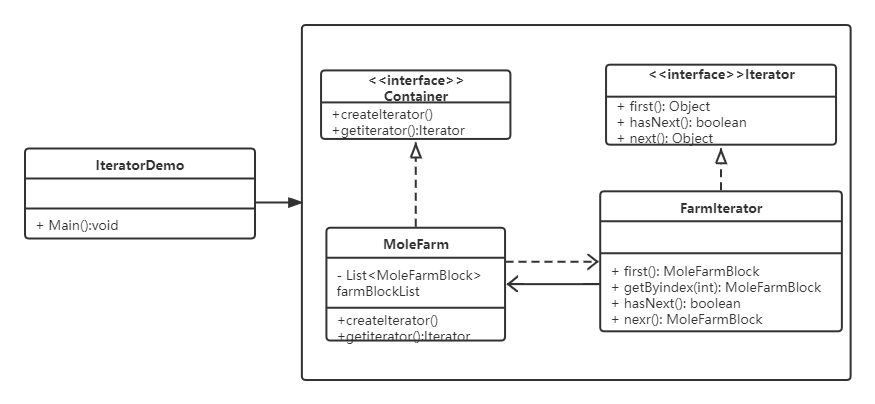
在农场中我们提供了一键种植、一键收获的方法。Director是指挥者，Builder是抽象的接口。两个具体的建造者ConcreteBuilder1和ConcreteBuilder2负责两种不同的建造方式，在MoleFarmBlock完成对种子的种植。

### **3.15 迭代器模式**

#### **3.15.1 实现API描述**

|  |  |
| --- | --- |
| API | FarmIterator.getByIndex (int) |
| 函数说明 | 给定指定的下标，返回对应农田块 |
| 参数 | int下标 |
| 返回值 | MoleFarmBlock农田块对象 |

#### **3.15.2 类图**



#### **3.15.3 模式简介与应用**

许多时候用户需要顺序访问集合对象的元素，而底层的数据结构可能较为复杂，需要屏蔽集合对象的底层表示，迭代器模式可用于解决这样的需求。项目中创建一个Iterator接口和一个返回迭代器的Container接口。MoleFarm实现了Container接口，FarmIterator实现了Iterator接口，用户遍历农田状态信息的时候便可以通过使用迭代器来屏蔽底层信息。

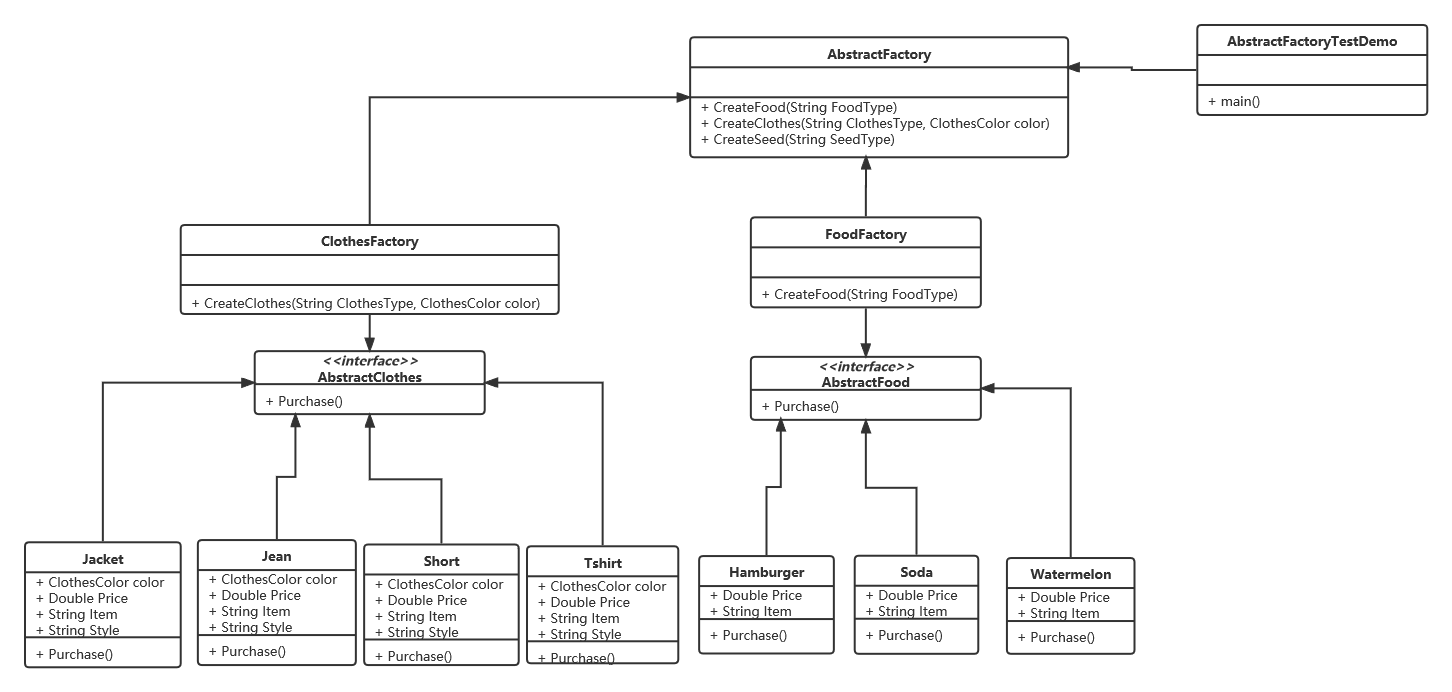
### **摩尔庄园——商店**

### **3.16 抽象工厂模式**

#### **3.16.1 实现API描述**

|  |  |
| --- | --- |
| API | Purchase () |
| 函数说明 | 购买商品 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.16.2类图**



#### **3.16.3 模式简介与应用**

在商场中有许多不同的商品会被出售，这些商品可以分为服装，食品；需要有不同的工厂产生这些产品，在项目中这三个不同的具体工厂继承自抽象工厂，服装食品种子则是具体工厂产生的抽象产品，包括夹克，T恤/汉堡，汽水等具体商品，这些商品分别具有getitem(),purchase()等方法可进行购买，获取服装信息等操作。

### **3.17 桥接模式**

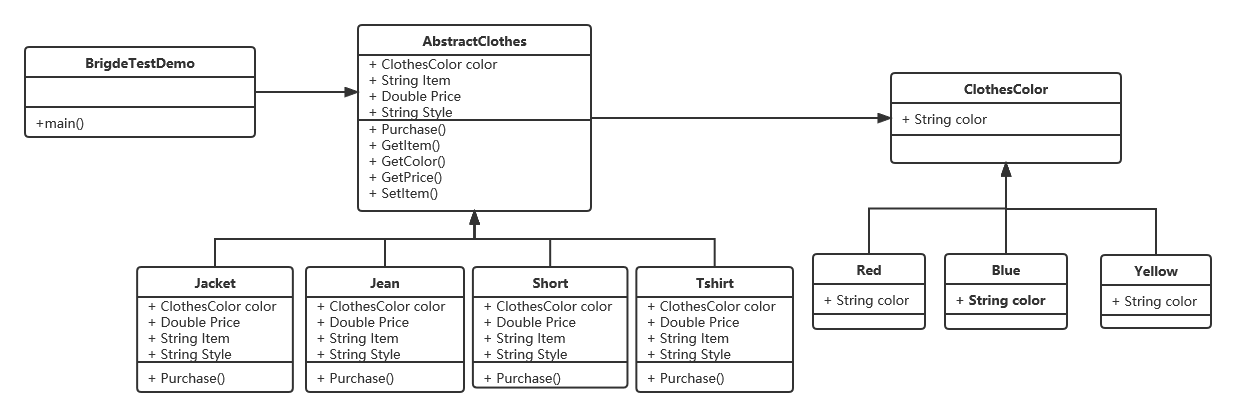
#### **3.17.1 实现API描述**

|  |  |
| --- | --- |
| API | AbstractClothes.purchase () |
| 函数说明 | 购买商品 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | AbstractClothes.getcolor () |
| 函数说明 | 获取颜色 |
| 参数 | 无 |
| 返回值 | 颜色 |

|  |  |
| --- | --- |
| API | AbstractClothes.setitem () |
| 函数说明 | 设置款式 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.17.2 类图**



#### **3.17.3 模式简介与应用**

商品的服装中拥有种类和颜色两种属性，如果将两种属性都作为成员变量放在类中，在新增颜色或种类时想要获得所有颜色所有款式的服装将需要创建很多类，使用桥接模式将颜色独立出来成为一个类，在服装类种加入对颜色的引用，可以大大减少需要创造的类的个数。

### **3.18 装饰器模式**

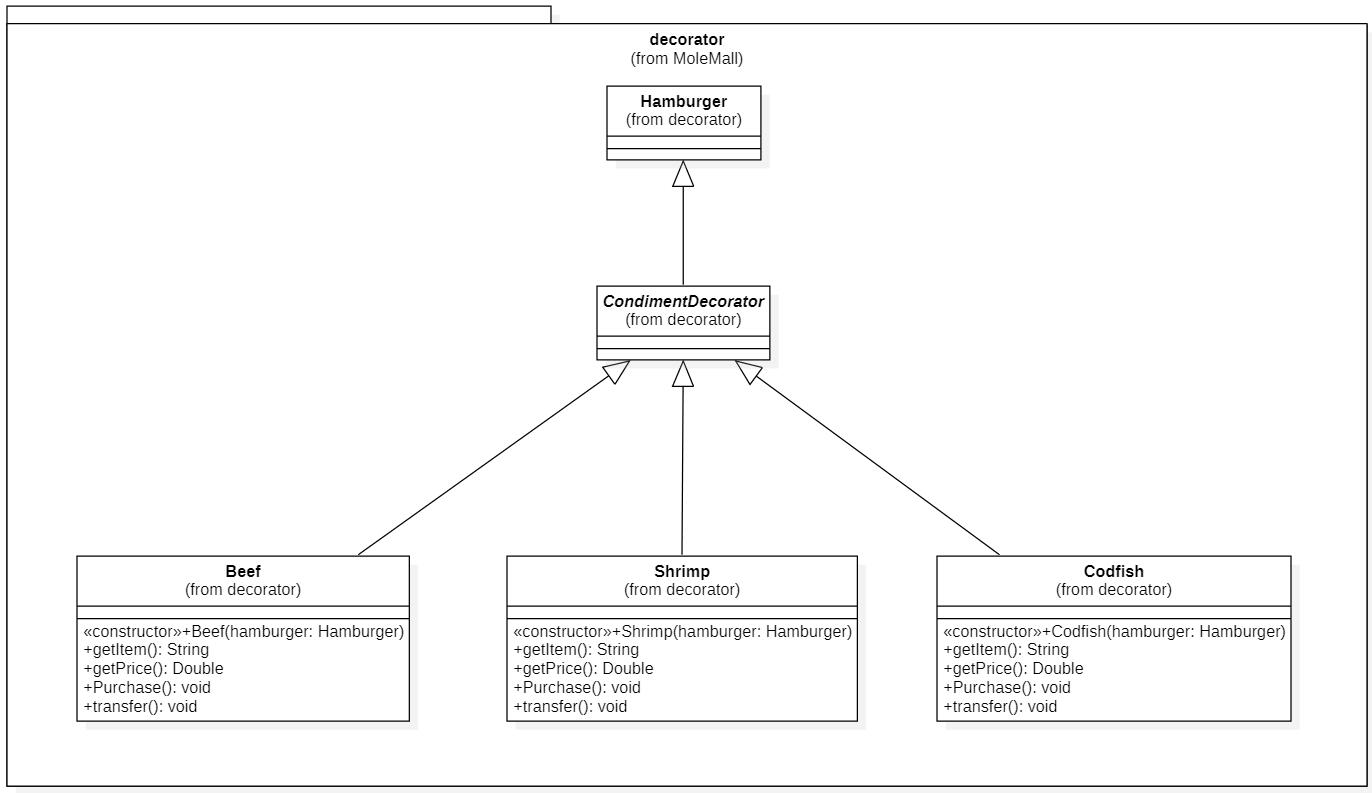
#### **3.18.1 实现API描述**

|  |  |
| --- | --- |
| API | getItem () |
| 函数说明 | 重写汉堡类中getItem()函数，以返回装饰后汉堡的名字 |
| 参数 | 无 |
| 返回值 | String，表示装饰后汉堡的名字 |

|  |  |
| --- | --- |
| API | getPrice () |
| 函数说明 | 重写汉堡类中getPrice()函数，以返回装饰后汉堡的价格 |
| 参数 | 无 |
| 返回值 | Double，表示装饰后汉堡的价格 |

|  |  |
| --- | --- |
| API | Purchase () |
| 函数说明 | 重写汉堡类中Purchase()函数 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.18.2 类图**



#### **3.18.3 模式简介与应用**

装饰器模式是一种对象结构型模式，主要对现有的类对象进行包裹和封装，期望在不改变类对象及其类定义的情况下，为对象添加额外的功能，该过程是通过调用被包裹之后的对象完成功能添加的，而不是直接修改现有对象的行为。

商场的食品店提供各种各样的食品，其中汉堡具有各式各样的口味，例如牛肉汉堡、鳕鱼汉堡、鲜虾汉堡，为了实现汉堡的各种口味而不改变汉堡对象的类定义，采用了装饰器模式。

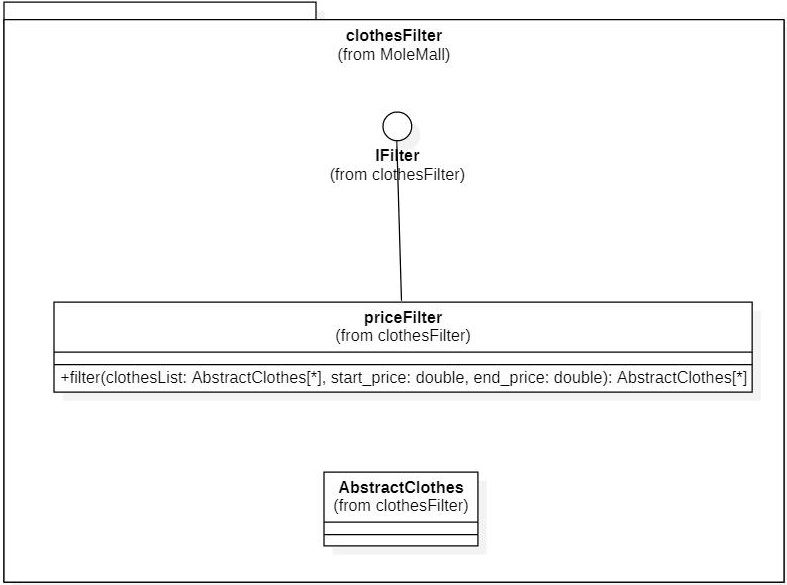
其中，抽象类CondimentDecrator继承实体类Hamburger，同时派生出3种口味Beef、Shrimp、Codfish，通过重写类内函数，实现不同口味汉堡的价格不同、名称不同等。

### **3.19 过滤器模式**

#### **3.19.1 实现API描述**

|  |  |
| --- | --- |
| API | filter () |
| 函数说明 | 得到过滤后的服装列表 |
| 参数 | 服装列表List<AbstractClothes> clothesList  最低价格double start\_price  最高价格double end\_price |
| 返回值 | List<AbstractClothes> |

#### **3.19.2 类图**



#### **3.19.3 模式简介与应用**

过滤器模式属于结构型模式，它允许开发人员使用不同的标准来过滤一组对象，通过逻辑运算以解耦的方式把它们连接起来。当摩尔在商场服装店购买服装时，可能有按照价格对服装进行筛选的需求，因此采用了过滤器模式来实现这种筛选。

其中IFilter是过滤器的接口，priceFilter是价格过滤器，对服装列表按照价格区间进行过滤。

#### **3.19.4 出处**

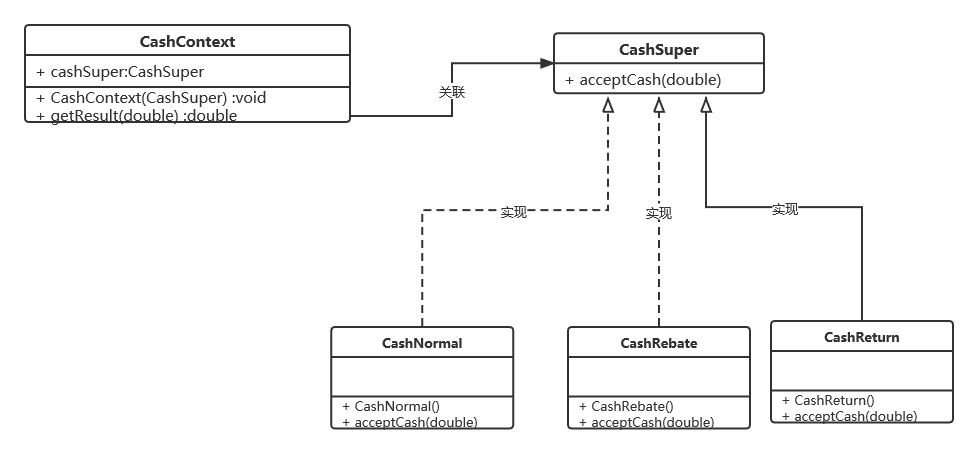
《Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML》Author：Mark Grand ISBN：0471449334, 9780471449331

### **3.20 策略模式**

#### **3.20.1 实现API描述**

|  |  |
| --- | --- |
| API | acceptCash () |
| 函数说明 | 对总价格进行优惠处理得到应付价格 |
| 参数 | double |
| 返回值 | double |

#### **3.20.2 类图**



#### **3.20.3 模式简介与应用**

策略模式属于行为型模式，它定义了一系列的算法，并将每一个算法封装起来，使每个算法可以相互替代，使算法本身和使用算法的客户端分割开来，相互独立。

当摩尔在商场服装店进行购买结算时会有不同优惠活动，使用策略过滤器模式可以避免if else的嵌套。

其中CashContext是来获取bean，CashNormal、CashRebate、CashReturn是策略的具体实现。

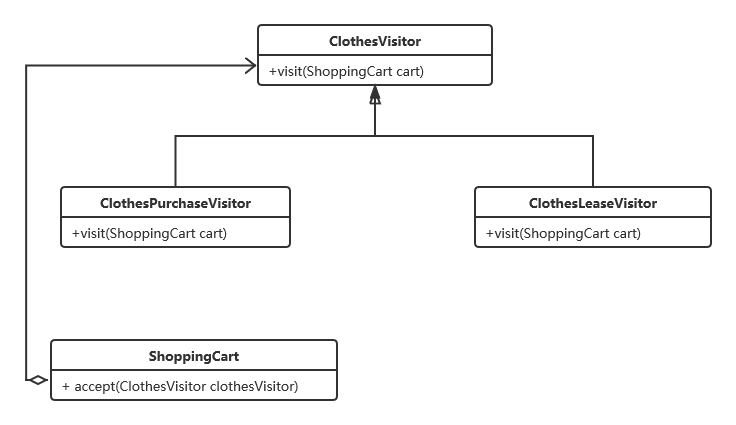
### **3.21 访问者模式**

#### **3.21.1 实现API描述**

|  |  |
| --- | --- |
| API | Visit () |
| 函数说明 | 传入被访问者cart，进行购买或租赁操作 |
| 参数 | T t（T extends ShoppingCart） |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | Accept () |
| 函数说明 | 根据传入的参数的不同类型，决定使用不同的重写方法 |
| 参数 | ClothesVisitor |
| 返回值 | 空 |

#### **3.21.2 类图**



#### **3.21.3 模式简介与应用**

访问者模式是一种结构型模式，它把数据结构和作用于结构上的操作之间的耦合解脱开，使得操作集合可以相对自由地演化。数据结构的每一个节点都可以接受一个访问者的调用，此节点向访问者对象传入节点对象，而访问者对象则反过来执行节点对象的操作。

商场的服装店中的服装可以支持两种操作——租赁和购买，这两种操作和商品本身解耦，作为操作集合。购物车接受两种操作作为访问者参数，而访问者反过来对购物车执行两种操作。

其中，ClothesPurchaseVisitor和ClotherLeaseVisitor实现接口ClothesVisitor的方法，两个Visitor类中具体实现visit操作——输出购买小票或租赁票据，购物车类接受访问。

### **摩尔庄园——游乐园**

### **3.22 外观模式**

#### **3.22.1 实现API描述**

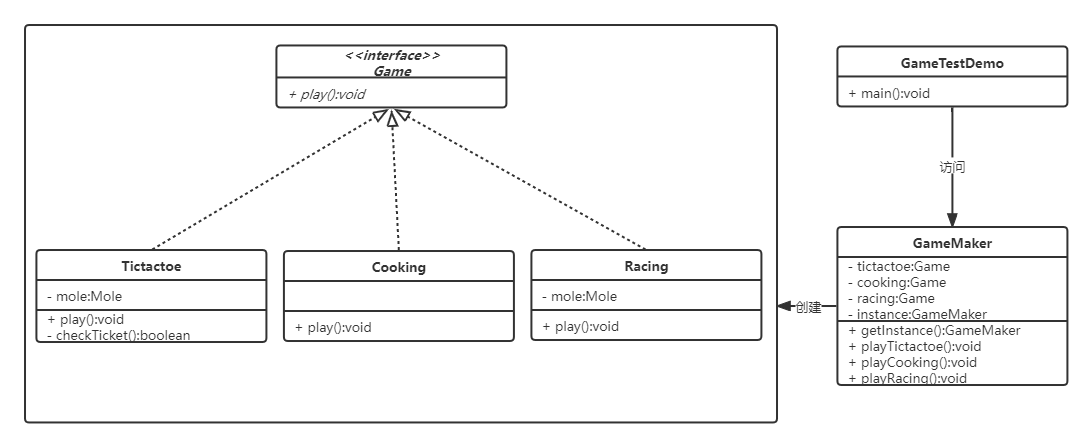
|  |  |
| --- | --- |
| API | GameMaker.getInstance () |
| 函数说明 | 获取gameMaker单例 |
| 参数 | 无 |
| 返回值 | GameMaker.instance |

|  |  |
| --- | --- |
| API | GameMaker.playTictactoe () |
| 函数说明 | 进行井字棋游戏 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | GameMaker.playCooking () |
| 函数说明 | 进行做饭游戏 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | GameMaker.playRacing () |
| 函数说明 | 进行赛车游戏 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.22.2 类图**



#### **3.22.3 模式简介与应用**

游乐园提供多样的有趣的游戏，从而造成游乐园变成一个复杂的系统。需要提供一个游戏的高层接口，这个接口使得这游乐园子系统更加容易使用。游乐园子系统包括井字棋(Tictactoe)游戏、做饭(Cooking)游戏、赛车(Racing)游戏。

在上述场景中可以抽象出Tictactoe类，Cooking类和Racing类，这三个类实现了Game的接口，GameMaker 类使用实体类来代表用户对这些类的调用，GameTestDemo 类使用 GameMaker 类来显示结果。

外观模式为复杂的游乐园子系统提供外界访问的模块，减少系统相互依赖，提高了灵活性和安全性。

### **3.23 享元模式**

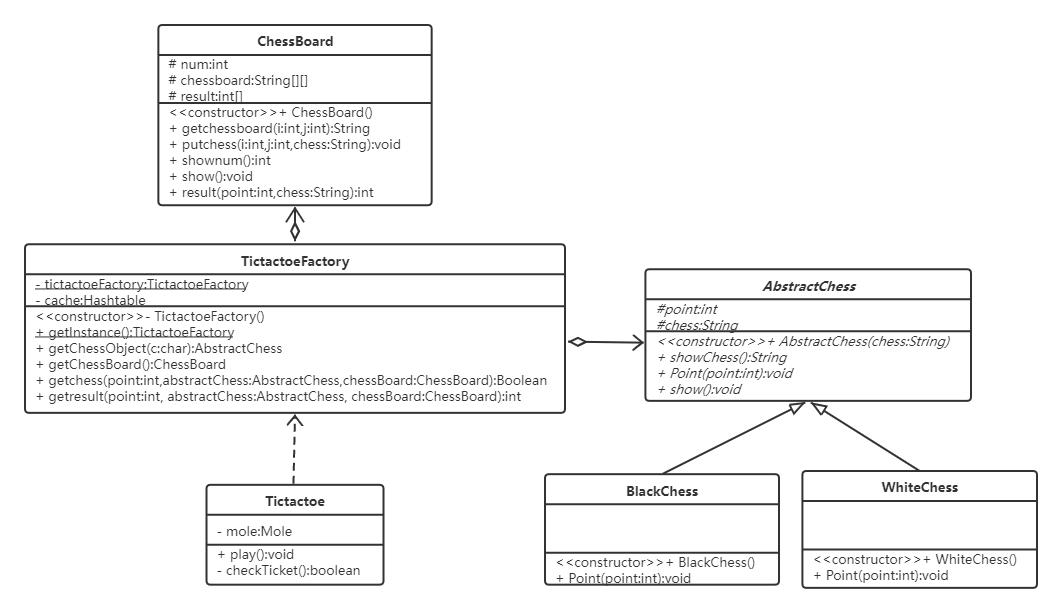
#### **3.23.1 实现API描述**

|  |  |
| --- | --- |
| API | TictactoeFactory.getChessObject () |
| 函数说明 | 从缓存中获得棋子对象实例 |
| 参数 | 字符c |
| 返回值 | 棋子对象 |

|  |  |
| --- | --- |
| API | TictactoeFactory.getchess () |
| 函数说明 | 填入棋子 |
| 参数 | 位置point, 棋子abstractChess, 棋盘chessBoard |
| 返回值 | 布尔值，表示填入棋子成功与否 |

|  |  |
| --- | --- |
| API | AbstractChess.Point () |
| 函数说明 | 在外部设置棋子坐标信息 |
| 参数 | 外部输入的棋子位置 |
| 返回值 | 空 |

#### **3.23.2 类图**



#### **3.23.3 模式简介与应用**

享元模式，以共享的方式高效地支持大量的细粒度对象。通过复用内存中已存在的对象，降低系统创建对象实例的性能消耗。井字棋游戏中的棋子的黑白色或者说形态（\*或者O）是对象的内蕴状态，它们在棋盘上的位置即为外蕴状态，外蕴状态是通过客户端来保存传入的，是可能会发生变化的，故而对井字棋游戏采用享元模式。

棋子工厂TictactoeFactory是用来获得棋子对象实例的，如果缓存中存在棋子对象实例，则从缓存中读取获得，否则生成一个新对象存入缓存后返回。享元模式的要义就在于“避免产生大量的细粒度对象，共享元对象”。AbstractChess是抽象棋子，黑白棋子均继承自抽象棋子。

### **3.24 模板模式**

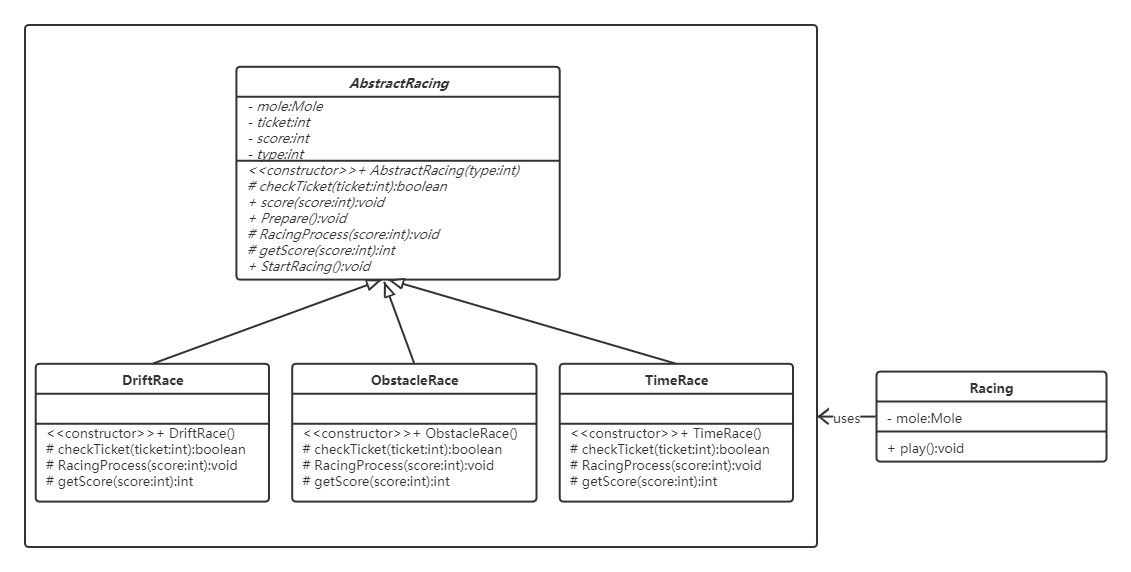
#### **3.24.1 实现API描述**

|  |  |
| --- | --- |
| API | AbstractRacing.checkTicket () |
| 函数说明 | 检票，不同的赛车小赛事对应不同的具体检票方法 |
| 参数 | 票ticket |
| 返回值 | Boolean，检票成功与否 |

|  |  |
| --- | --- |
| API | AbstractRacing.RacingProcess () |
| 函数说明 | 进行小赛事，不同的小赛事对应不同的具体进行赛事方法 |
| 参数 | 得分 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | AbstractRacing.checkTicket () |
| 函数说明 | 记分，不同的赛车小赛事对应不同的具体记分方法 |
| 参数 | 得分 |
| 返回值 | 得分 |

#### **3.24.2 类图**



#### **3.24.3 模式简介与应用**

在模板模式中，一个抽象类公开定义了执行它的方法的模板。它的子类可以按需要重写方法实现，但调用将以抽象类中定义的方式进行。游乐园中的赛车游戏分为漂移赛、障碍赛、竞速赛，这三种小赛事均包含检票、进行比赛、计算得分的过程，故而可以定义一个抽象类AbstractRacing，定义检票、进行比赛、计算得分这三个方法的模板，对赛车小游戏使用模板模式。

应用模板模式，提取公共代码，便于维护。另外，赛车行为由父类AbstractRacing控制，子类DriftRace、ObstacleRace、TimeRace实现。

### **3.25 备忘录模式**

#### **3.25.1 实现API描述**

|  |  |
| --- | --- |
| API | RecordMemento.getScore () |
| 函数说明 | 获取分数的状态 |
| 参数 | 无 |
| 返回值 | 得分 |

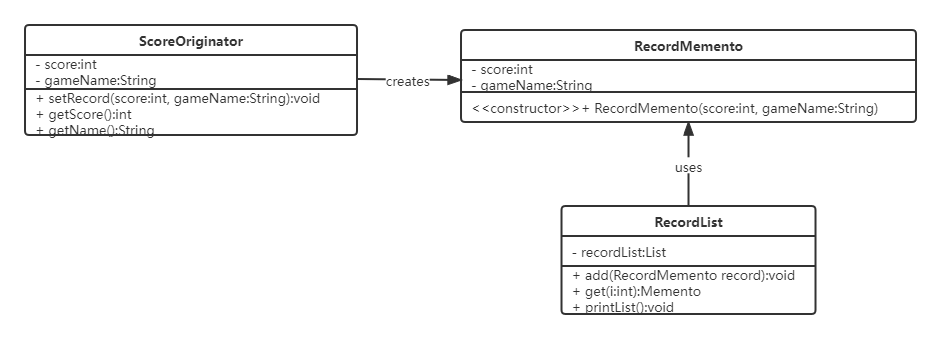
|  |  |
| --- | --- |
| API | RecordMemento.getName () |
| 函数说明 | 获取游戏名称 |
| 参数 | 无 |
| 返回值 | 游戏名称 |

|  |  |
| --- | --- |
| API | ScoreOriginator.saveRecordToMemento () |
| 函数说明 | 将当前记录存储到RecordMemento中 |
| 参数 | 无 |
| 返回值 | 备忘录 |

|  |  |
| --- | --- |
| API | RecordList.add () |
| 函数说明 | 添加积分记录 |
| 参数 | RecordMemento类 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | RecordList.get () |
| 函数说明 | 获取某一游戏积分记录并输出 |
| 参数 | 记录序号 |
| 返回值 | 空 |

#### **3.25.2 类图**



#### **3.25.3 模式简介与应用**

备忘录模式保存一个对象的某个状态，以便在适当的时候恢复对象。小摩尔在游乐园可以玩许多游戏，需要有备忘录记录小摩尔玩过游戏所获积分，故而对保存游戏积分场景应用备忘录模式。

RecordMemento包含了要保存对象的状态，ScoreOriginator 创建并在RecordMemento对象中存储状态，RecordList 对象负责从RecordMemento 中获得对象的状态。

当用户希望保存某个分数时，系统先调用scoreOriginator创建分数与游戏名称，并用RecordList.add()保存即可；另一方面，如果用户希望获得某个存档点前的状态，输入分数序号并调用RecordList.get()即可。

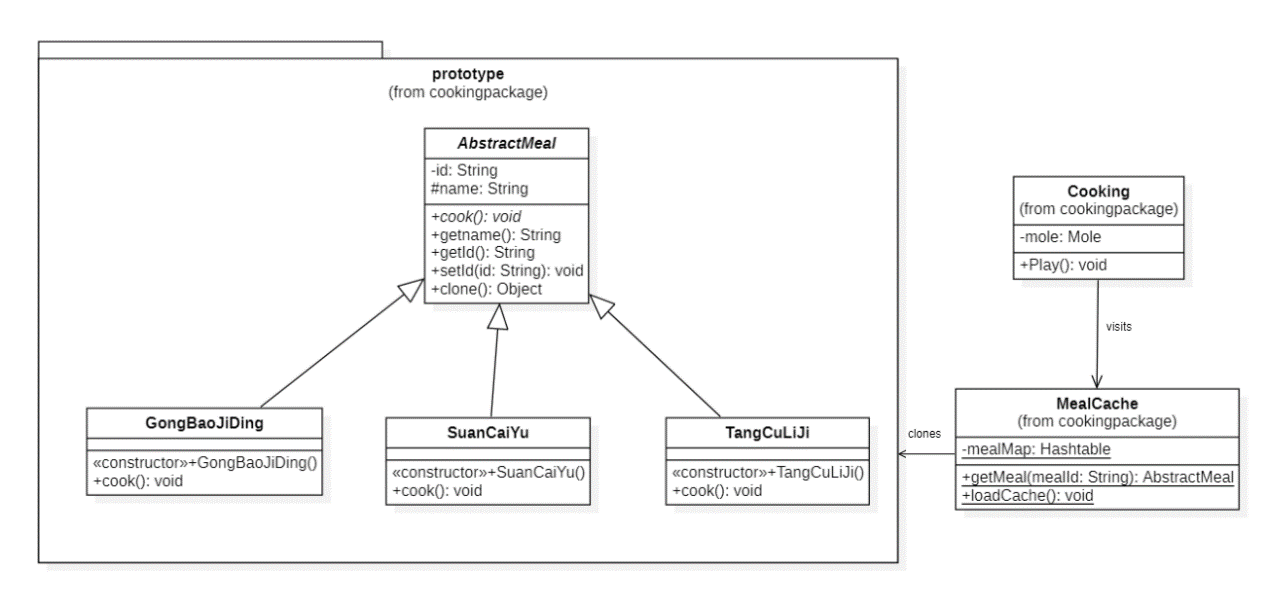
### **3.26 原型模式**

#### **3.26.1 实现API描述**

|  |  |
| --- | --- |
| API | AbstractMeal.cook () |
| 函数说明 | 抽象菜肴做菜接口 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | AbstractMeal.clone () |
| 函数说明 | 克隆菜肴对象 |
| 参数 | 无 |
| 返回值 | 克隆出来的新的菜肴对象 |

#### **3.26.2 类图**



#### **3.26.3 模式简介与应用**

小摩尔可能会重复做菜肴，在这个场景中，菜肴就不断地需要被重复创造出来。通过原型模式，快速克隆菜肴，从而逃避构造函数的约束，性能提高。

创建一个实现了 Cloneable 接口的抽象类AbstractMeal，创建扩展了上面抽象类的实体类GongBaoJiDing、SuanCaiYu、TangCuLiJi，创建一个类MealCache，从数据库获取实类，并把它们存储在一个 Hashtable 中，Cooking 使MealCache 类来获取存储在 Hashtable 中的形状的克隆。

### **摩尔庄园——聊天室**

### **3.27 中介者模式**

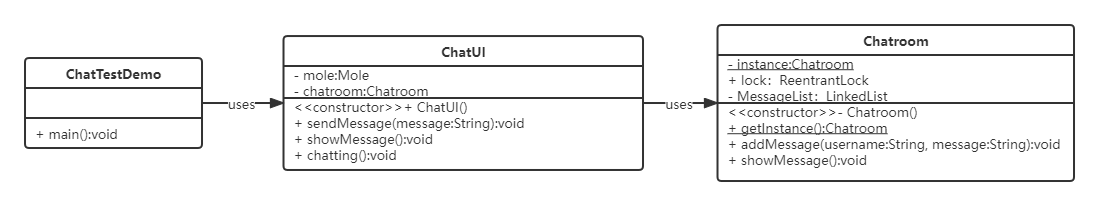
#### **3.27.1 实现API描述**

|  |  |
| --- | --- |
| API | Chatroom.addMessage () |
| 函数说明 | 添加消息 |
| 参数 | 用户名、消息 |
| 返回值 | 空 |
| API | **Chatroom.showMessage ()** |
| 函数说明 | 显示消息 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | ChatUI.sendMessage () |
| 函数说明 | 客户端发送消息 |
| 参数 | 消息 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | ChatUI.chatting () |
| 函数说明 | 聊天界面接口 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.27.2 类图**



#### **3.27.3 模式简介与应用**

中介者模式是用来降低多个对象和类之间的通信复杂性。这种模式提供了一个中介类，该类通常处理不同类之间的通信，并支持松耦合，使代码易于维护。游乐园聊天室就是这么一个多对象聊天的中介者。多个用户可以向聊天室发送消息，聊天室向所有的用户显示消息。

创建两个类 ChatRoom 和 ChatUI。ChatUI 相关的对象，比如客户端以及另外创建的机器人类，使用 ChatRoom 方法来分享他们的消息。ChatTestDemo类中定义了一个客户端myChatUI以及两只机器人，进而演示它们通过聊天室进行交流。

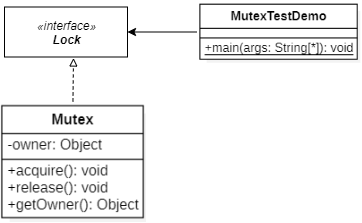
### **3.28 互斥模式**

#### **3.28.1 实现API描述**

|  |  |
| --- | --- |
| API | Mutex.acquire () |
| 函数说明 | 获得锁 |
| 参数 | 无 |
| 返回值 | 空 |

|  |  |
| --- | --- |
| API | Mutex.release () |
| 函数说明 | 释放锁 |
| 参数 | 无 |
| 返回值 | 空 |

#### **3.28.2 类图**



#### **3.28.3 模式简介与应用**

锁（Lock）是一个接口，具有释放和获取操作。互斥锁（Mutex）是一个具体实现，其作用是为聊天室提供一个互斥锁，使得只有一个NPC能进入聊天室添加消息，避免多个NPC冲突。

使用互斥设计模式来严格控制对聊天室的访问，限制对聊天室资源的使用。在本场景中应用互斥设计模式是为了保证功能的正常使用。

#### **3.28.4 出处**

《Design Patterns in Java》 Author: Steven John Metsker， William C. Wake ISBN：0321629949, 9780321629944

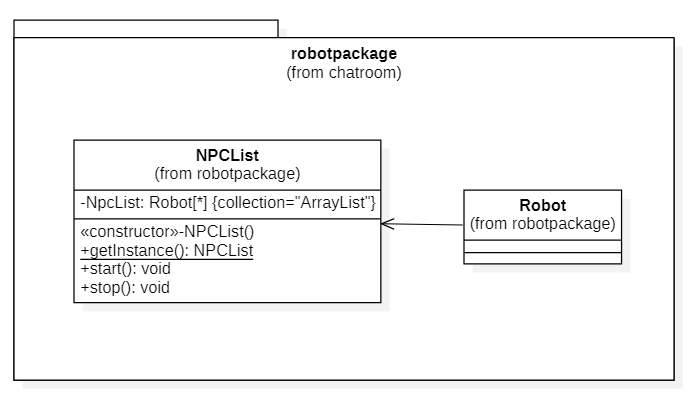
### **3.29 多例模式**

#### **3.29.1 实现API描述**

|  |  |
| --- | --- |
| API | NPCList.getInstance () |
| 函数说明 | 获取所有的NPC |
| 参数 | 无 |
| 返回值 | NPCList |

|  |  |
| --- | --- |
| API | Robot.getInstance () |
| 函数说明 | 根据输入的NPC名称返回对应NPC的单例 |
| 参数 | NPC名称 |
| 返回值 | NPC单例 |

#### **3.29.2 类图**



#### **3.29.3 模式简介与应用**

聊天室中存在三个NPC与客户端交流，它们分别是菩提大伯，么么公主，瑞琪。 每个NPC都是独一无二的实例，各个NPC并不相同。

本场景中，使用枚举类型来实现多例模式。在NPCList枚举类型中共有三个枚举值，分别对应聊天室中的三个NPC。在调用静态方法Robot.getInstance(String robotName)时传入的字符串参数会决定返回的NPC单例。如果对应的NPC不存在，则会抛出异常。

#### **3.29.4 出处**

《Apex Design Patterns》 Author：Jitendra Zaa, Anshul Verma ISBN：1782173668, 9781782173663

## 4 **项目分工说明与成员贡献**

|  |  |  |
| --- | --- | --- |
| 人员 | 分工 | 贡献度 |
| 1952455刘思远 | 框架搭建，代码整合、演讲 | 100% |
| 1951120罗检 | 游乐园、聊天室 | 100% |
| 1951121宁之恒 | 农场代码与文档 | 100% |
| 1951574吴渭 | 农场代码与文档 | 100% |
| 1952107王子轩 | 游乐园、聊天室 | 100% |
| 1952221杨一迪 | 商店代码与文档 | 100% |
| 1952377蒋汶霖 | 农场代码与文档 | 100% |
| 1952540万惟佳 | 商店代码与文档 | 100% |
| 1952635凌亚楠 | 商店代码与文档 | 100% |
| 1952910裴元昊 | 游乐园、聊天室 | 100% |

原型游戏（摩尔庄园）部分截图：









