迪士尼乐园

Document for Design Pattern Project

by
1751593 张亦驰
1751984 王舸飞
1752184 周晨曦
1752304 符其军
1752366 谭棋
1752662 龙宇
1753798 赵佳庆
1753910 马思腾

1 背景介绍

《迪士尼乐园》是我们小组设计的一套以迪士尼主题乐园为背景的API。使用者主要可分为主题乐园管理员和游客两种。乐园管理员可以设置游览指南、管理迪士尼大酒店信息、管理票务信息、筛选符合条件的预约、提醒游客娱乐设施状态变化、迪士尼演出人物角色管理、安检游客包中物品、检查安检部件、开展门票促销活动、管理AR合照功能、添加合照场景、迪士尼餐馆接受订餐订单。游客可以查看和听游览指南、经行走失儿童求助（孩子、家长皆可求助）、预定迪士尼大酒店、恢复预定酒店信息、购买各种类型的票、预约游乐设施、与虚拟角色合影、与Ar角色互动、更改合影场景背景、在迪士尼中点餐吃饭。

2 design pattern总览

编号	Design pattern name	实现个（套）数	sample programe 数目
1	Adapter	1	1
2	Mediator	1	1
3	Command	1	1
4	Memento	1	1
5	Visitor	1	1
6	Filter	2	2
7	Facade	1	1
8	Observer	1	1
9	Interpreter	1	1
10	Factory Method	1	1
11	Chain of Responsibility	1	1
12	Decorator	1	1
13	Iterator	1	1
14	Abstract Factory	1	1
15	Bridge	1	1
16	Proxy	1	1
17	Template	1	1
18	Prototype	1	1
19	State	1	1
20	Simple factory	1	1
21	Builder	1	1
22	Composite	1	1
23	Flyweight	1	1
24	Singleton	3	3

3 design pattern详述

1.Adapter

Adaptor模式能使接口不兼容的对象能够相互合作。适配器通过封装对象将复杂的转换过程隐藏，被封装的对象甚至察觉不到适配器的存在。客户端代码只需通过接口与适配器交互，获取不能直接取得的服务即可，无需与具体的适配器类耦合。因此，向程序中添加新类型的适配器时无需修改代码。

优点：单一职责原则（可将接口或数据转换代码从程序主要业务逻辑中分离）；开闭原则（只要客户端代码通过客户端接口与适配器进行交互， 就能不修改现有客户端代码而添加新的适配器）

缺点：代码整体复杂度增加，因为你需要新增一系列接口和类。有时直接更改服务类使其与其他代码兼容会更简单。

1. API描述

Adapter场景为：迪士尼公园在导航板上提供游览指南，大多数游客可以轻易从中获取信息，而视力障碍的游客则需要有声指南的帮助。有声指南适配器应能够从不同指南中读取信息并转化成声音信号输出。

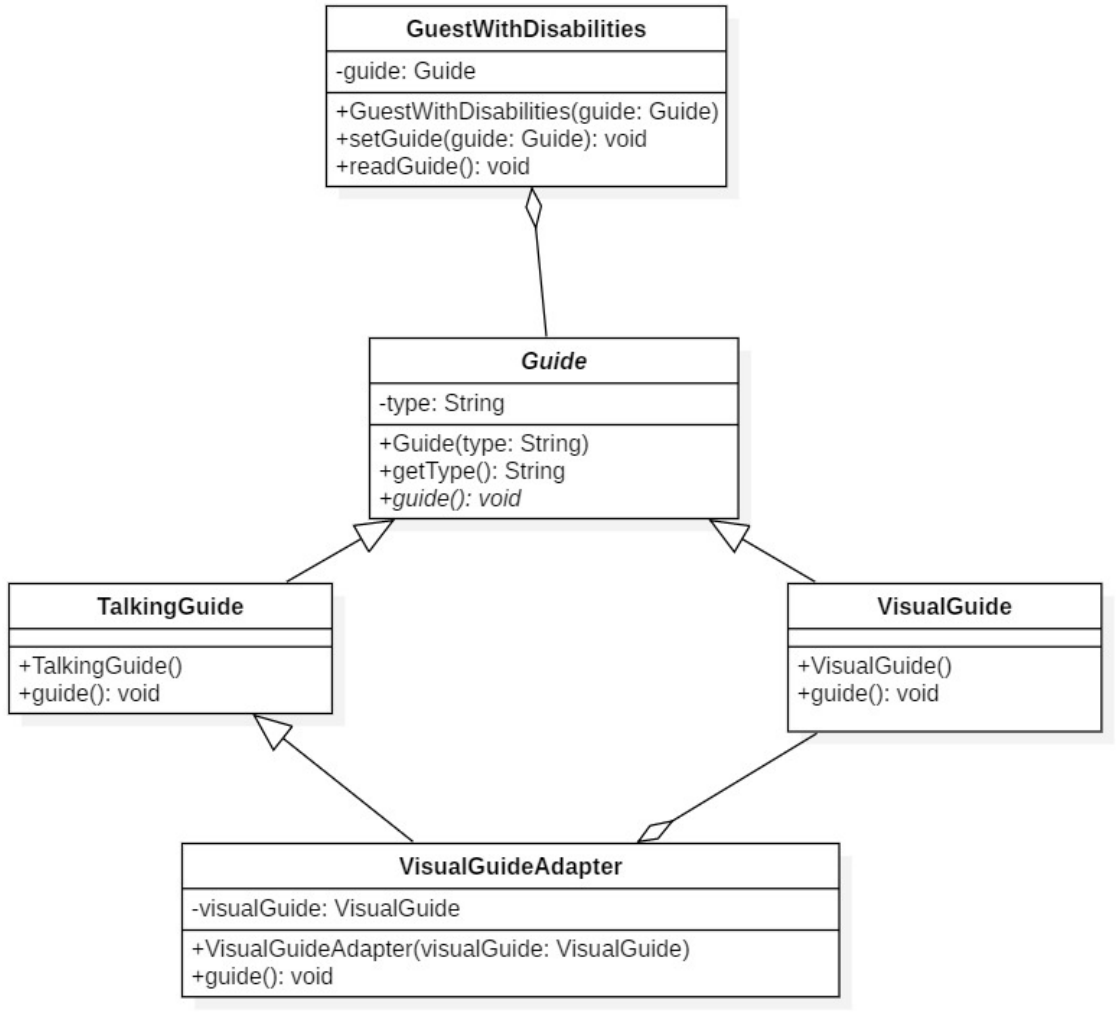
Guide是指南的抽象基类，声明了抽象方法guide()。TalkingGuide和VisualGuide继承了Guide，对父类guide()做各自实现。

GuestWithDisabilities是视障游客，声明了setGuide()、readGuide()方法。具体函数如下：

函数名	作用
setGuide(Guide guide)	设置指南种类
readGuide()	游客阅读地图

VisualGuideAdapter是适配器，持有visualGuide引用，通过继承目标类TalkingGuide，使得视障游客能够获取VisualGuide信息。

2. class diagram



2.Mediator

中介者模式能减少对象之间混乱无序的依赖关系，该模式会限制对象之间的直接交互，迫使它们通过一个中介者对象进行合作。

优点：降低了对象之间的耦合性，使得对象易于独立地被复用。

缺点：将对象间的一对多关联转变为一对一的关联，提高系统的灵活性，使得系统易于维护和扩展。

1. API描述

Mediator场景为：迪士尼吸引了众多游客前来体验丰富的游乐项目，尤其是节假日期间，场面异常火爆。为了避免孩子与家长走散所带来的麻烦，迪士尼设立了走失儿童认领处，如果发现自己和亲人失散，儿童和家长均可向其求助，为其尽快联系上家属。

Colleague：为每个ConcreteColleague提供与Mediator交互的接口。

ColleagueBase：每个Colleague的抽象基类。关键方法如下：

方法名	作用
getCount()	Mediator获取访问走失儿童认领处次数
setCount(int count)	计数，Mediator更新Colleague访问次数
setMediator(Mediator mediator)	设置Mediator
setColleagueState(ColleagueState colleagueState)	设置聚散状态
getColleagueState()	获取聚散状态
getMyName()	获取自己的姓名
getFamilyName()	获取家人姓名
seekHelp()	通过mediator，向BabyCareCenter寻求帮助
printResult()	根据不同状态输出结果

Baby：家长走失的孩子，在该设计模式中扮演(ConcreteColleague)角色，是ColleagueBase的具体实现。

Parent：与孩子失散的家长，在该设计模式中扮演(ConcreteColleague)角色，是ColleagueBase的具体实现。

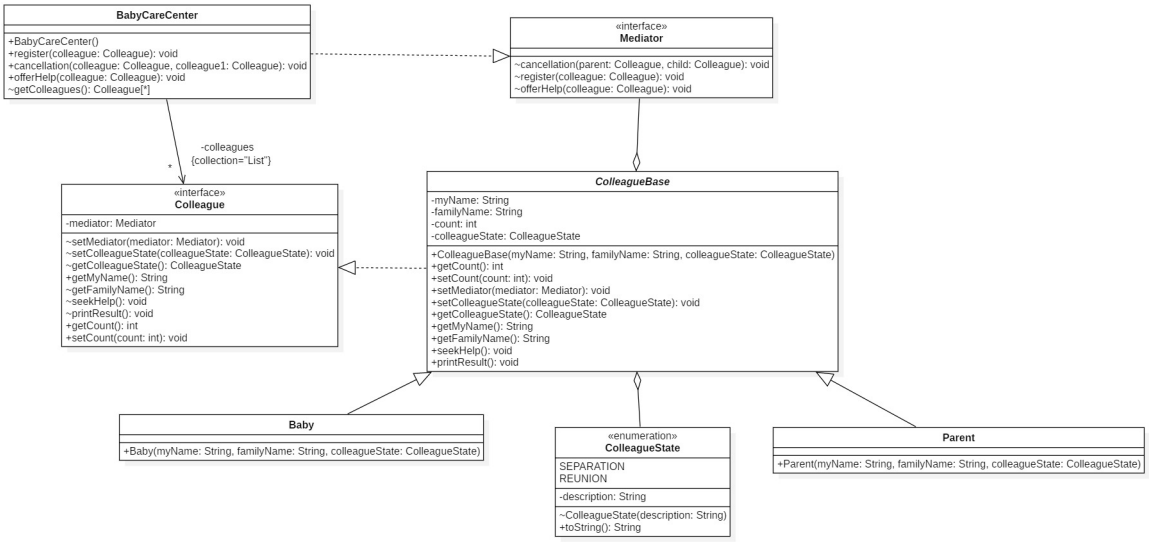
ColleagueState：枚举类，用于表示孩子与家长的聚散状态（separation和reunion）。

Mediator：中介者接口，交由BabyCareCenter实现。

BabyCareCenter：走失儿童认领处，是Mediator类的具体实现。在该设计模式中扮演(ConcreteMediator)角色。关键方法如下：

方法名	作用
register(Colleague colleague)	登记Colleague，为每个Colleague提供Mediator
cancellation(Colleague colleague, Colleague colleague1)	在家长与其失散的孩子团聚后调用此方法，将家长及其孩子的状态设置为REUNION
offerHelp(Colleague colleague)	为每个需要寻找其亲人的Colleague提供帮助

2. class diagram



3.Command

命令模式是一种行为设计模式， 它可将请求转换为一个包含与请求相关的所有信息的独立对象。该转换让你能根据不同的请求将方法参数化、 延迟请求执行或将其放入队列中， 且能实现可撤销操作。

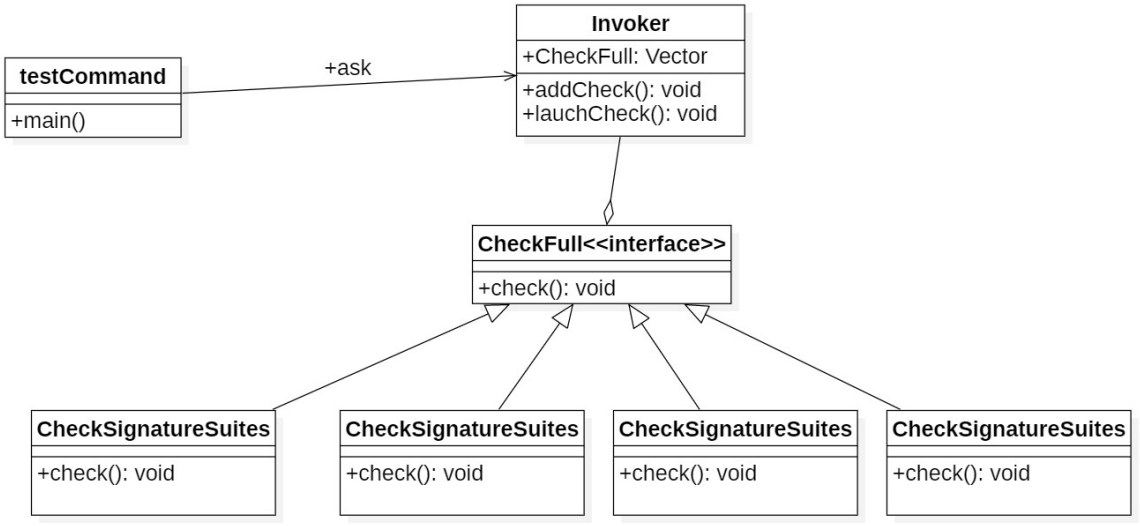
优点：你可以实现撤销和恢复功能。

1.API描述

Command场景：当游客在迪士尼乐园游玩时，游客可以选择住宿在迪士尼乐园大酒店。酒店需要查看还是否有房源，对于不同的房间类型要进行不同的检查：CheckClubLevelRooms、CheckDeluxeRooms、CheckJuniorSuites、CheckSignatureSuites。这些功能可在CheckFull中调用，具体的实现则被封装好。

方法名	作用
CheckClubLevelRooms()	查看行政房是否有空房
CheckDeluxeRooms()	查看豪华房是否有空房
CheckJuniorSuites()	查看小套房是否有空房
CheckSignatureSuites()	查看主题套房是否有空房

2. class diagram



4.Memento

备忘录模式是一种行为设计模式， 允许在不暴露对象实现细节的情况下保存和恢复对象之前的状态。

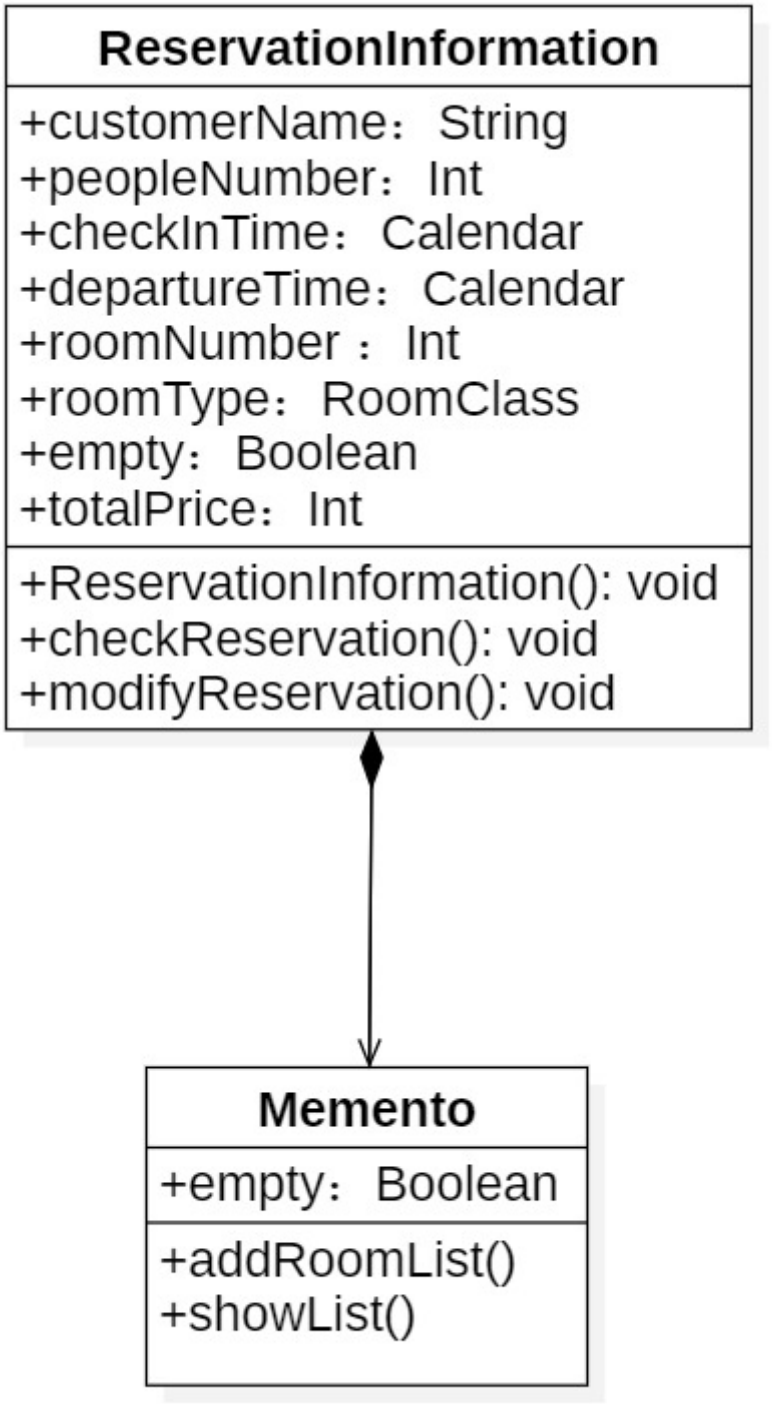
优点： 你可以在不破坏对象封装情况的前提下创建对象状态快照。

1.API描述

Memento场景：当有游客预定房间后，被预定的房间会被设置为不为空，并保存下预定前的信息（addRoomList）。若预定取消，则可以恢复此前状态（showList）。

方法名	作用
addRoomList（）	保存下预定前的信息
showList（）	恢复此前状态

2. class diagram



5.Visitor

访问者模式是一种行为设计模式， 它能将算法与其所作用的对象隔离开来。访问者模式通过在访问者对象中为多个目标类提供相同操作的变体， 让你能在属于不同类的一组对象上执行同一操作。该模式会将所有非主要的行为抽取到一组访问者类中， 使得程序的主要类能更专注于主要的工作。

优点：

1、开闭原则。 你可以引入在不同类对象上执行的新行为， 且无需对这些类做出修改。

2、单一职责原则。可将同一行为的不同版本移到同一个类中。

3、访问者对象可以在各种对象交互时收集一些有用的信息。当你想要遍历一些复杂的对象结构（例如对象树），并在结构中的每个对象上应用访问者时，这些信息可能会有所帮助。

缺点：1、每次在元素层次结构中添加或移除一个类时， 你都要更新所有的访问者。

2、在访问者同某个元素进行交互时， 它们可能没有访问元素私有成员变量和方法的必要权

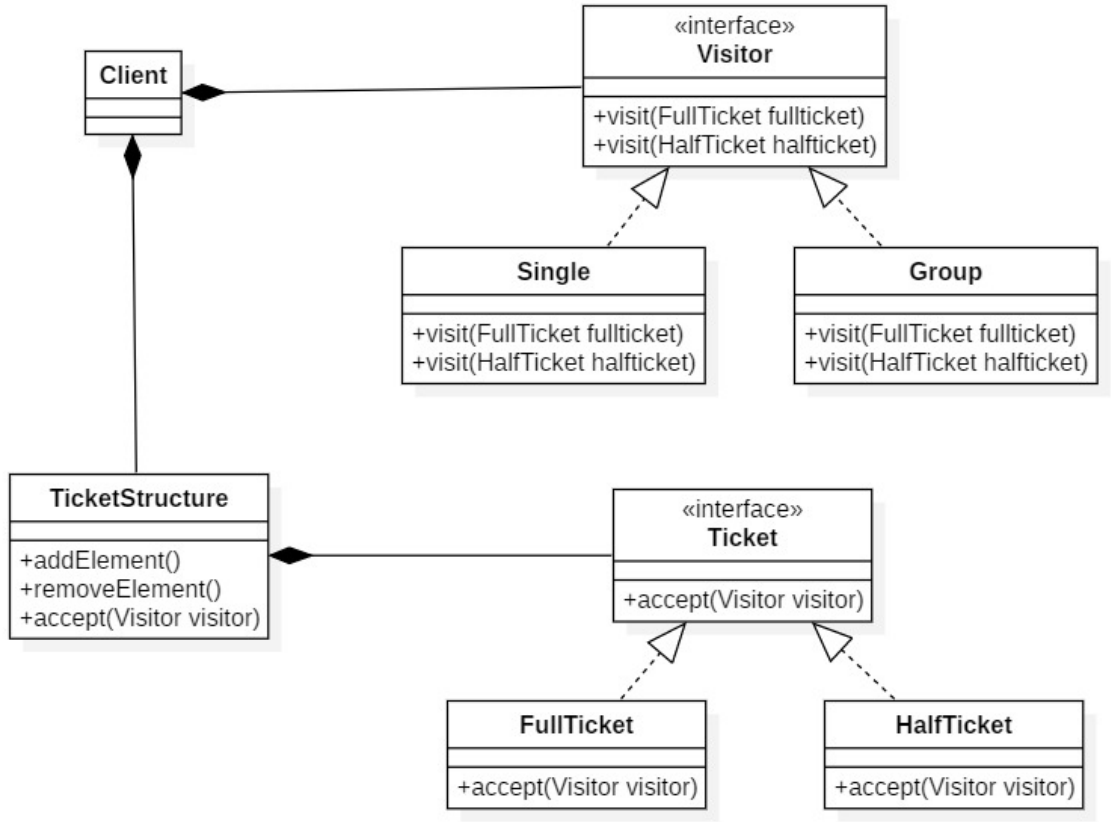
限。

1.API描述

Visitor场景为：迪士尼乐园会接受各种各样的游客，其中因为游客团体的不同，迪士尼的票务系统也提供不同的种类的票供游客选择，也提供不同的票务窗口。因为游客的数量不同，门票分为团体窗口和个人窗口，因为游客身份的不同，门票分为全价票和半价票。

EventListener是用于处理Editor产生的状态改变，进而向特定的EventListener发送值得关注的事件，且允许新预约者加入和当前预约者离开列表的订阅框架。EventListener接口声明了通知接口，仅包含一个update()方法。RollerCosterListener和CarrouselListener分别对应了预约摩天轮和预约旋转木马的预约者。

2. class diagram



6.Filter

Filter Pattern（过滤器模式）又被称为Criteria Pattern（标准模式 ）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。这种类型的设计模式属于结构型模式，它结合多个标准来获得单一标准。

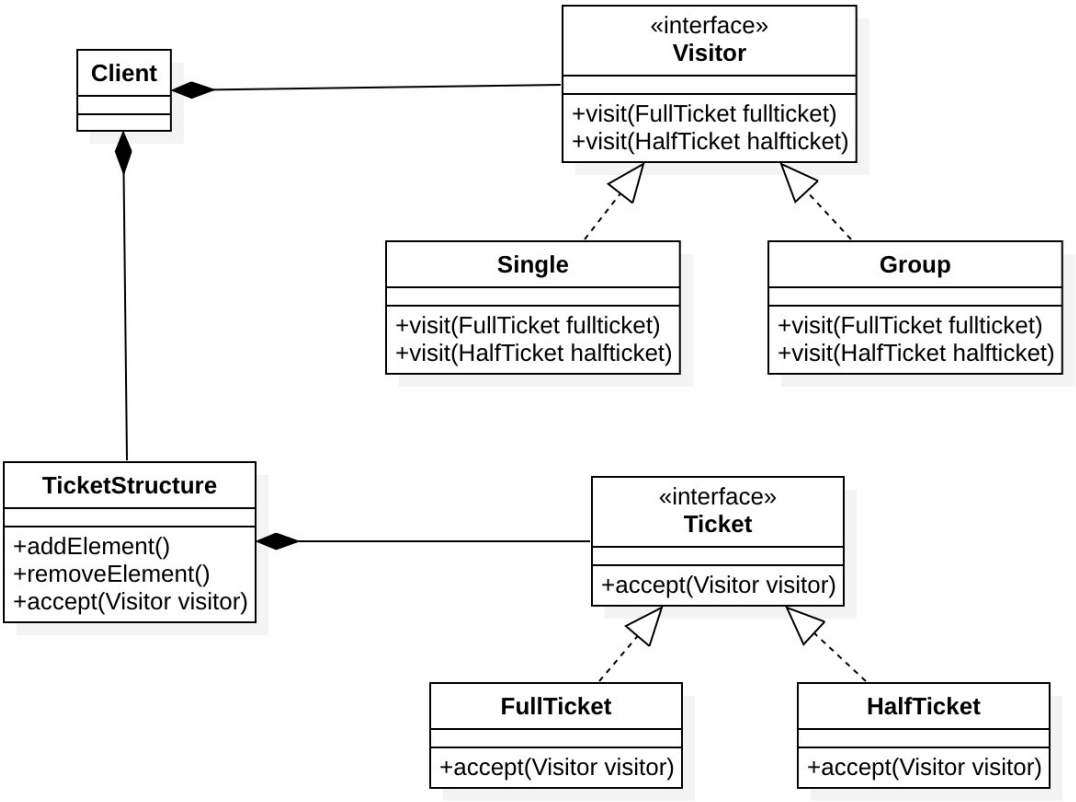
优点：简单，解耦，使用方便。

缺点：在构建过滤规则的时候会有些繁琐。

1.API描述

Filter场景为：迪士尼乐园有游乐设施如摩天轮和旋转木马，每个游乐设施都对游客的年龄或身高有一定限制，在已预约该项目的游客中，需要以不同的条件筛选出符合游玩条件的游客并显示。

2. class diagram

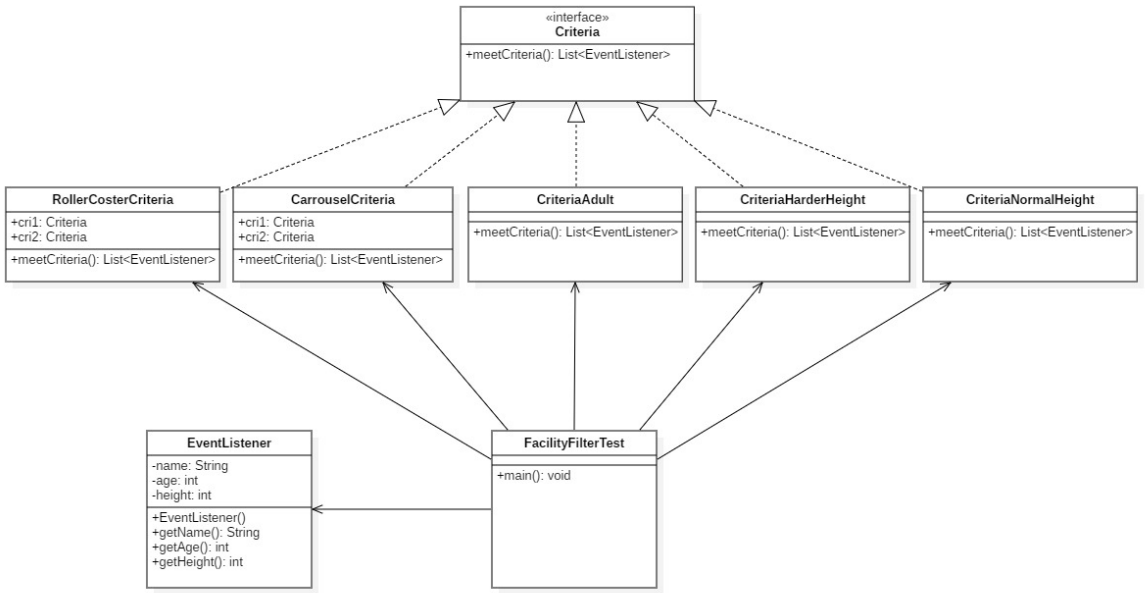


3.API描述

Filter场景为：迪士尼乐园有游乐设施如摩天轮和旋转木马，每个游乐设施都对游客的年龄或身高有一定限制，在已预约该项目的游客中，需要以不同的条件筛选出符合游玩条件的游客并显示。

Criteria接口声明了标准接口，仅包含一个meetCriteria()方法。CriteriaAdult、CriteriaHarderHeight、CriteriaNormalHeight都定义了具体的简单标准，分别筛选年龄和不同标准的身高。RollerCosterCriteria和CarrouselCriteria分别定义了摩天轮和旋转木马游玩条件的复杂标准，用于筛选能够游玩该游乐设施的游客。

4.class diagram



7.Facade

Facade（外观模式）是一种结构型设计模式， 能为程序库、 框架或其他复杂类提供一个简单的接口。提供了一种访问特定子系统功能的便捷方式， 其了解如何重定向客户端请求， 知晓如何操作一切活动部件。

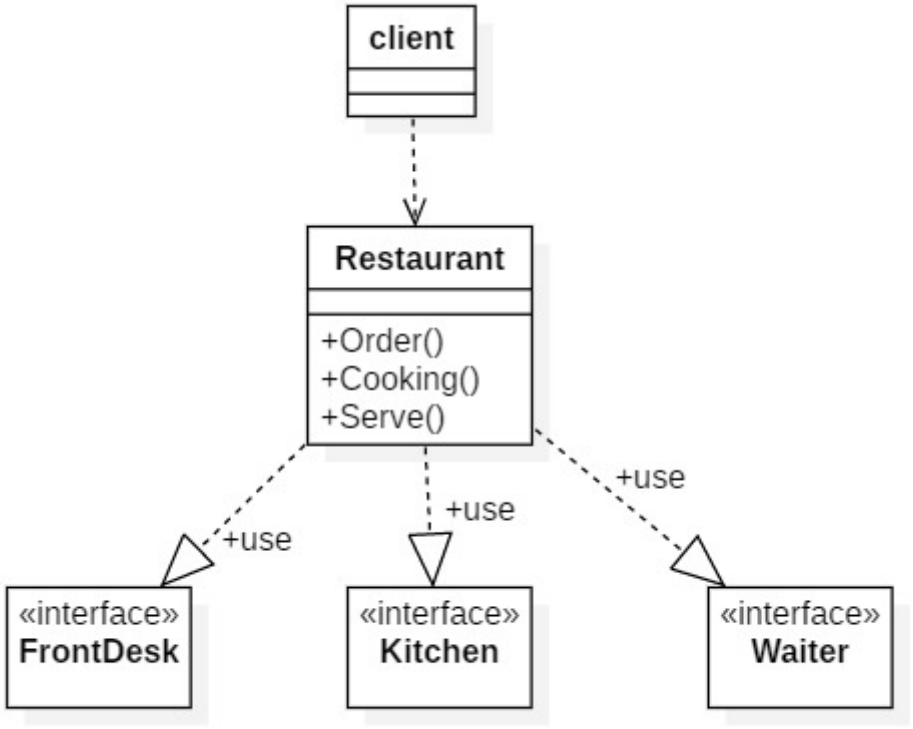
优点：你可以让自己的代码独立于复杂子系统。

缺点：外观可能成为与程序中所有类都耦合的上帝对象。

1.API描述

迪士尼内部的餐厅有着明确的分工，分为前台、厨房、服务员三个部门，分别负责点单、烹饪、上菜。因此有三个子系diar统FrontDesk、Kitchen、Waiter,有各自的成员函数。

2.Class diagram



8.Observer

Observer Pattern定义了一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。主要解决一个对象状态改变给其他对象通知的问题，而且要考虑到易用和低耦合，保证高度的协作的问题。观察者与被观察者之间是属于轻度的关联关系，并且是抽象耦合的。

优点:

1、观察者和被观察者是抽象耦合的。

2、建立一套触发机制。

缺点:

1、如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。

2、如果观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃。

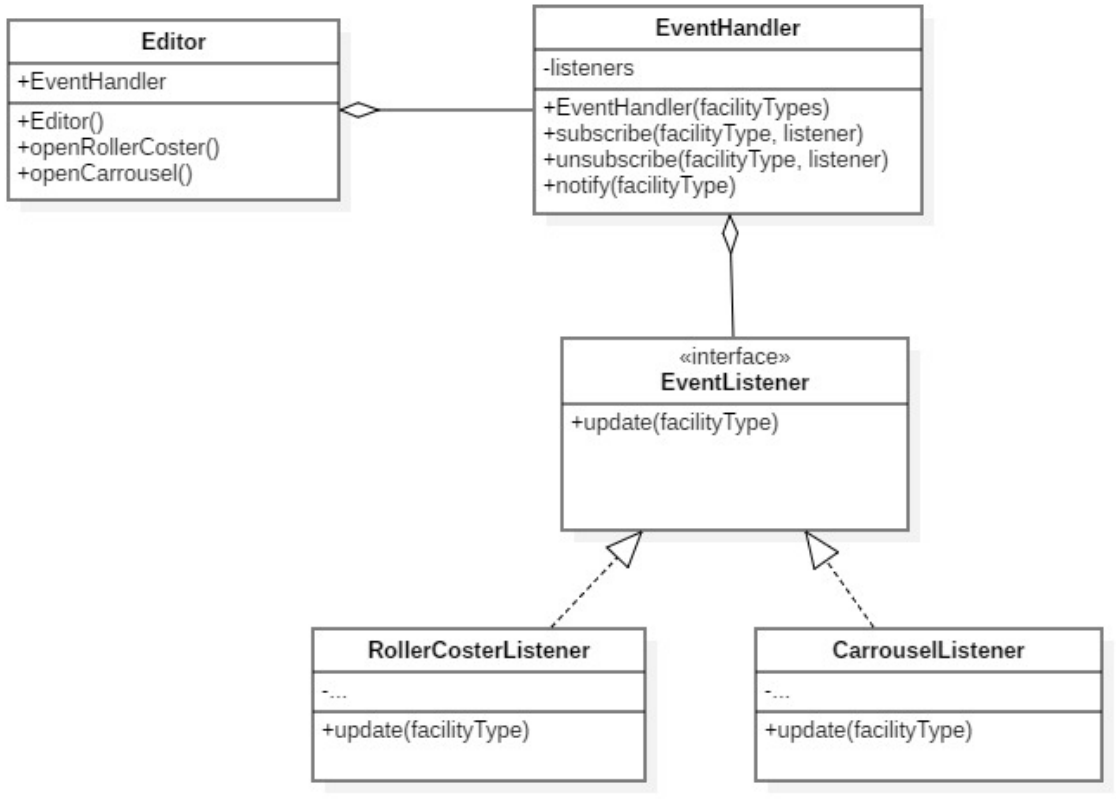
3、观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

1.API描述

Observer场景为：迪士尼乐园有游乐设施如摩天轮和旋转木马，它们都还未开放，对这些设施感兴趣的游客们可以提前预约，当它们其中的项目开放或状态产生变化时，就会通知所有的预约者。

EventListener是用于处理Editor产生的状态改变，进而向特定的EventListener发送值得关注的事件，且允许新预约者加入和当前预约者离开列表的订阅框架。EventListener接口声明了通知接口，仅包含一个update()方法。RollerCosterListener和CarrouselListener分别对应了预约摩天轮和预约旋转木马的预约者。

2.class diagram



9.Interpreter

Interpreter Pattern（解释器模式）提供了评估语言的语法或表达式的方式，属于行为型模式。它给分析对象定义一个语言，并定义该语言的文法表示，再设计一个解析器来解释语言中的句子。也就是说，用编译语言的方式来分析应用中的实例。这种模式实现了文法表达式处理的接口，该接口解释一个特定的上下文。

优点:

1、扩展性好。由于在解释器模式中使用类来表示语言的文法规则，因此可以通过继承等机制来改变或扩展文法。

2、容易实现。在语法树中的每个表达式节点类都是相似的，所以实现其文法较为容易。

缺点：

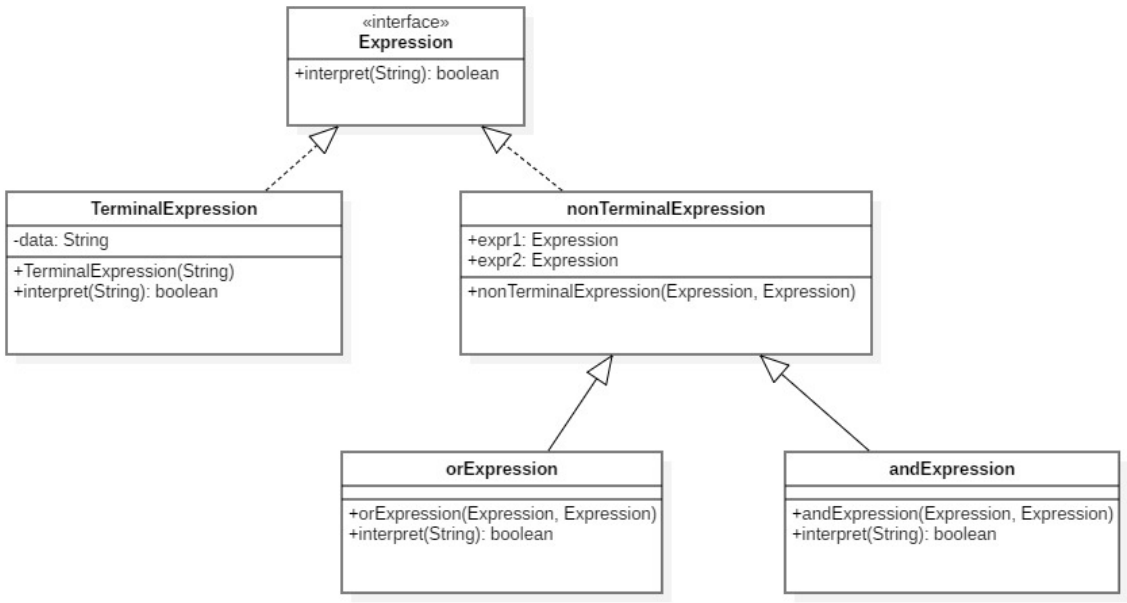
1、会引起类膨胀。解释器模式中的每条规则至少需要定义一个类，当包含的文法规则很多时，类的个数将急剧增加，导致系统难以管理与维护。

2、可应用的场景比较少。在软件开发中，需要定义语言文法的应用实例非常少，所以这种模式很少被使用到。

1.API描述

Interpreter场景为：定义数个文法规则，如"迪士尼人物有micky和donald"，在询问系统micky是否是迪士尼人物时，系统即会识别句法，并解析终结符和非终结符，最终返回判断值。实现一个简单的查询功能。

2.class diagram



10.Factory Method

Factory Method（工厂方法模式），又称工厂模式，是一种创建型设计模式，它在父类中提供一个创建对象的接口，允许子类决定实例化对象的类型。

优点：

1、可以避免创建者和具体产品之间的紧密耦合。

2、单一职责原则。可以将产品创建代码放在程序的单一位置，从而使得代码更容易维护。

3、开闭原则。无需更改现有客户端代码，就可以在程序中引入新的产品类型。

缺点：

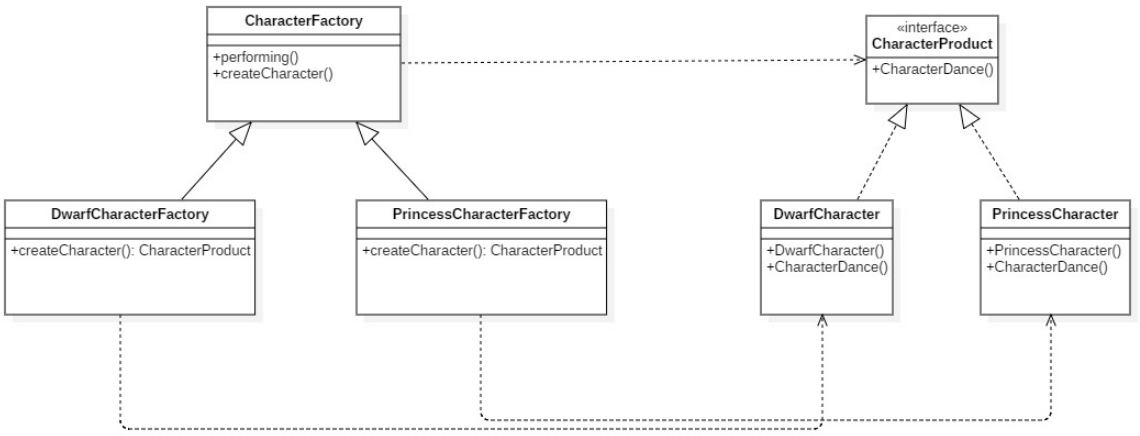
1、应用工厂方法模式需要引入许多新的子类，代码可能会因此变得更复杂。

1.API描述

Factory Method场景为：迪士尼乐园有一个演出需要设计多个小矮人和多名公主出演，也许未来也会加入新的角色，需要多次创建一个人物（小矮人、公主等）。

CharacterFactory是一个抽象的工厂，它定义了一个performing函数用于调用所有角色的CharacterDance操作。创建角色操作为一个抽象方法，由其子类创建不同角色。CharacterProduct对接口进行了声明。而每个具体角色都是对角色接口的不同实现。

2.class diagram



11.Chain of Responsibility

责任链模式是一种行为设计模式， 允许你将请求沿着处理者链进行发送。 收到请求后， 每个处理者均可对请求进行处理， 或将其传递给链上的下个处理者。

优点

1. 你可以控制请求处理的顺序。
2. 单一职责原则。 你可对发起操作和执行操作的类进行解耦。
3. 开闭原则。 你可以在不更改现有代码的情况下在程序中新增处理者。

缺点

1. 部分请求可能未被处理。

1.API描述

迪士尼乐园为了准确地检测出游客的包里到底放了什么东西，引进了一套安检机器。这个安检机器可以自由组合自己的安检部件，构成一个安检链，链上的任何一个部件发现不符合乐园规定的东西，就会宣告无法通过安检。反之则将包裹传递给链条上的下一个部件。

- SecurityCheckProcess

  安检流程的抽象类。

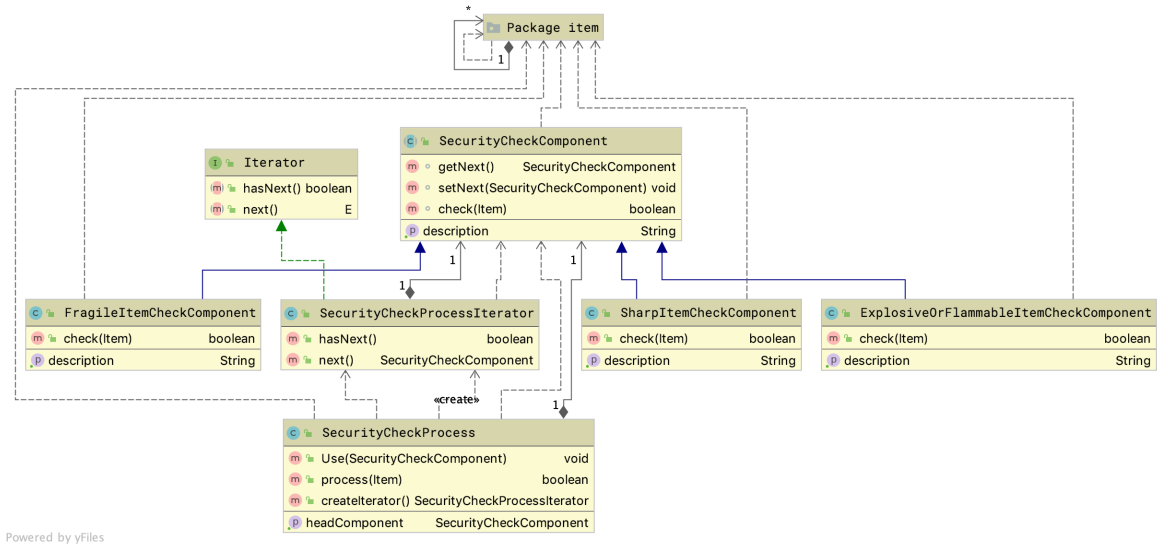
方法名	作用
Use	在流程中添加一个安检部件
process	安检一个物品
createIterator	创造一个迭代器用来迭代自身的安检部件

- SecurityCheckComponent

  安检部件类。可以独立地处理一个物品，并判断它是否可以通过自己的检查。

方法名	作用
getNext()	取得链上的下一个安检部件
setNext()	设置链上的下一个部件
check（）	检查一个物品，判断其是否符合自己的标准

2.class Diagram



12.Decorator

装饰模式是一种结构型设计模式， 允许你通过将对象放入包含行为的特殊封装对象中来为原对象绑定新的行为。

优点

1. 你无需创建新子类即可扩展对象的行为。
2. 你可以在运行时添加或删除对象的功能。
3. 你可以用多个装饰封装对象来组合几种行为。
4. 单一职责原则。 你可以将实现了许多不同行为的一个大类拆分为多个较小的类

缺点

1. 在封装器栈中删除特定封装器比较困难。
2. 实现行为不受装饰栈顺序影响的装饰比较困难。
3. 各层的初始化配置代码看上去可能会很糟糕。

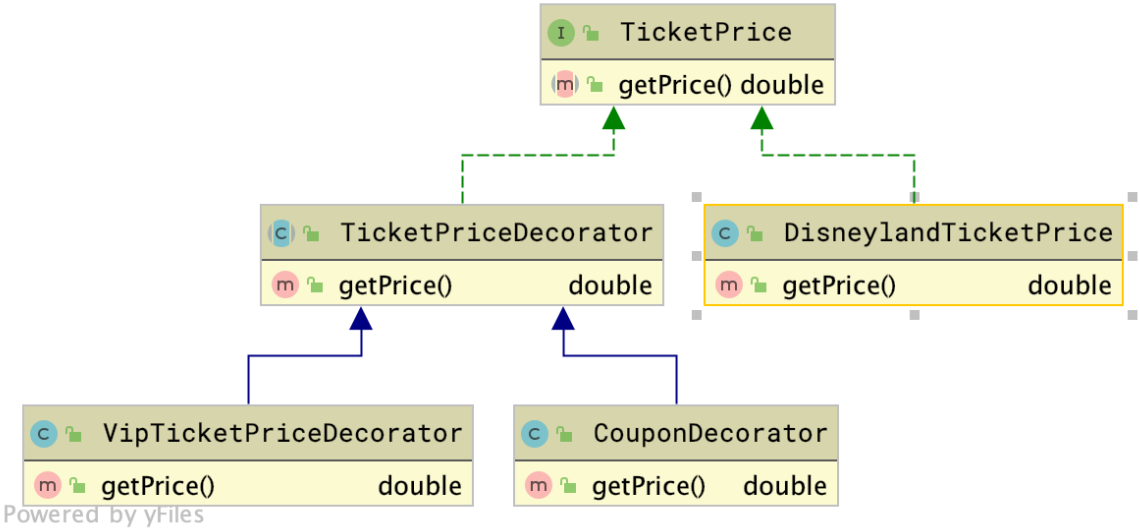
1.API描述

迪士尼乐园的门票有很多促销活动。为了扩展促销的方式，对各种促销活动进行任意组合得到门票价格，使用装饰者模式完成门票价格的计算。

有一个抽象的接口类TicketPrice，和一个真实的实习类DisneylandTicketPrice，以及一个装饰类TicketPriceDecorator。使用装饰类修饰迪士尼门票的价格，达到任意组合折扣的效果。

getPrice（）返回的是门票此时的价格。

2.class diagram



13.Iterator

迭代器模式是一种行为设计模式， 让你能在不暴露集合底层表现形式 （列表、 栈和树等） 的情况下遍历集合中所有的元素。

优点

1. 单一职责原则 通过将体积庞大的遍历算法代码抽取为独立的类， 你可对客户端代码和集合进行整理。
2. 开闭原则。 你可实现新型的集合和迭代器并将其传递给现有代码， 无需修改现有代码。
3. 你可以并行遍历同一集合， 因为每个迭代器对象都包含其自身的遍历状态。
4. 相似的， 你可以暂停遍历并在需要时继续。

缺点

1. 如果你的程序只与简单的集合进行交互， 应用该模式可能会矫枉过正。
2. 对于某些特殊集合， 使用迭代器可能比直接遍历的效率低。

1.API描述

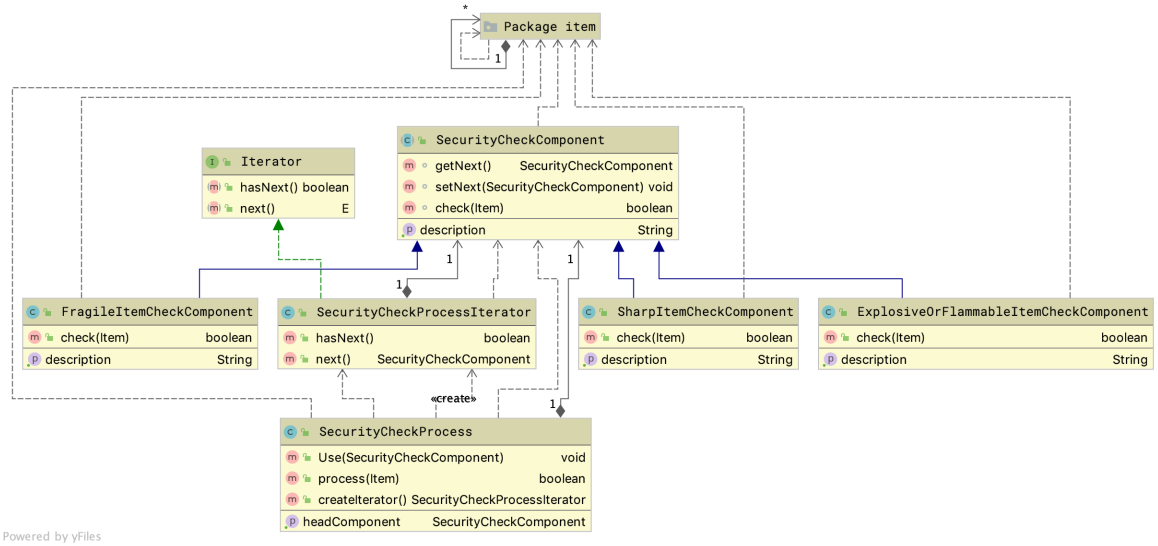
迪士尼的管理员有时需要检查一下安检流程中现在都有哪些部件。使用迭代器封装了部件链表的遍历操作，使得调用者可以轻松地查看安检流程中的部件。

接口类 Iterator 定义了迭代器需要有的函数

迭代器实现类 SecurityCheckProcessIterator。 实现了hasNext()和next()方法，用于遍历安检流程。

SecurityCheckProcessIterator不开放构造函数，只能通过SecurityCheckProcess创建。

2.class diagram



14.Abstract Factory

抽象工厂模式是一种创建型设计模式， 它能创建一系列相关的对象， 而无需指定其具体类。

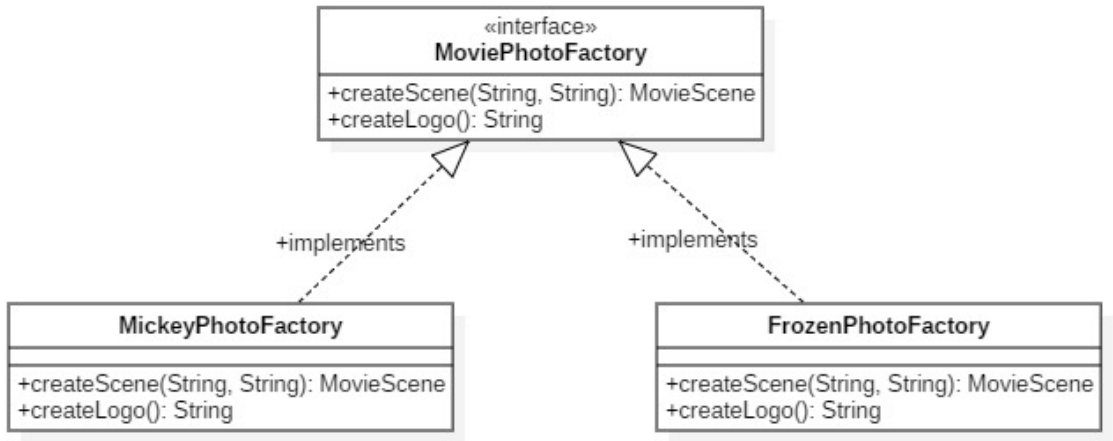
在本例中，用户创建的照片需要包含一位虚拟角色和电影Logo，而且他希望这位虚拟角色和Logo匹配正确。通过抽象工厂模式使米老鼠电影场景和冰雪奇缘电影场景同时继承合照工厂类，用户可以直接连续调用生成场景函数和生成Logo函数，而不必去在意他们是否匹配。

如果后续需要增加新电影场景，也只需继续继承合照工厂即可（OCP原则）

1.API描述

方法名	作用
createScene(String characterName, String background):MovieScene	根据角色名和背景文字生成一个场景
createLogo():String	返回一个Logo

2.Class diagram



15.Bridge

桥接模式是一种结构型设计模式， 可将一个大类或一系列紧密相关的类拆分为抽象和实现两个独立的层次结构， 从而能在开发时分别使用。

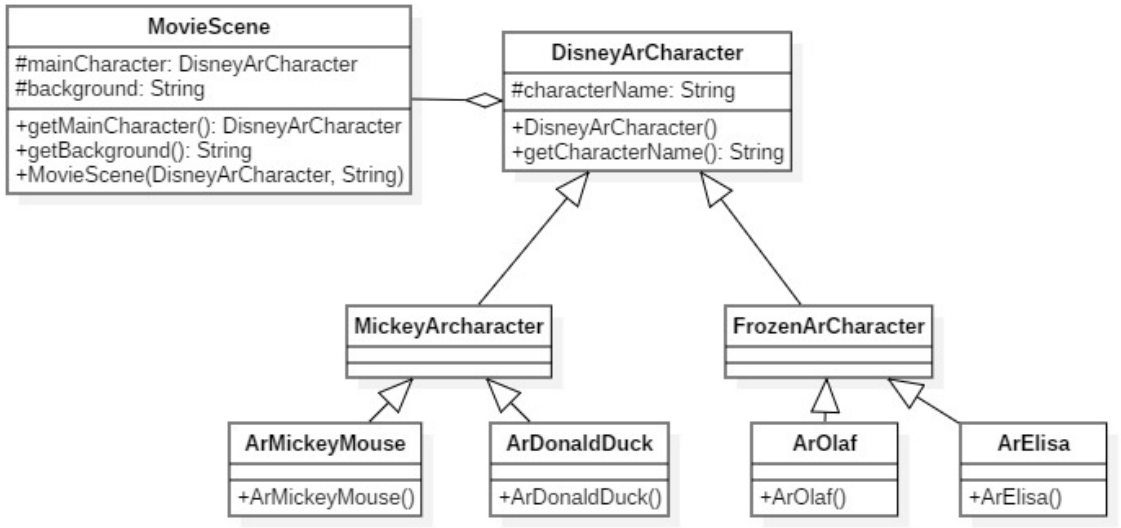
在本例中，一个电影场景包含一位Ar角色，此时如果将该角色的所有信息都封装到场景类当中，将会造成场景类的大量冗余。在后续新增角色时，也需要改动大量原有代码。

由于角色是信息量较大的相对独立个体，最好的办法就是利用桥接模式，将人物单独抽象成为一个类，这样无论是后续新增人物、还是想在场景中增加新元素，都可以在互不影响的情况下安全完成。

1.API描述

方法名	作用
MovieScene::getMainCharacter():DisneyArCharacter	返回一个Ar角色类
MovieScene::getBackground():String	返回该场景的背景文字
MovieScene::MovieScene()	场景的构造函数
DisneyArCharacter::getCharacterName():String	返回一个角色名

2.class diagram



16.Proxy

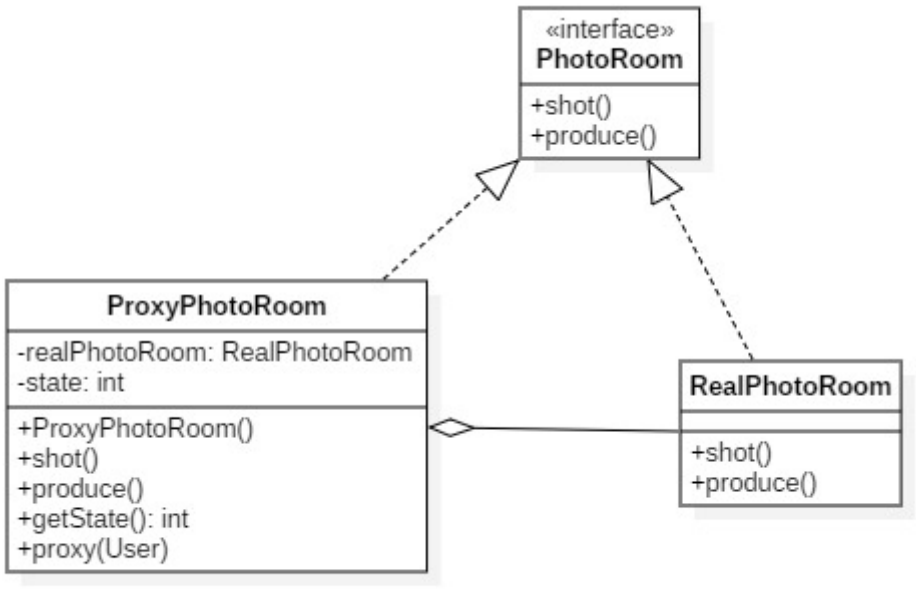
代理模式是一种结构型设计模式， 让你能够提供对象的替代品或其占位符。 代理控制着对于原对象的访问， 并允许在将请求提交给对象前后进行一些处理。

在本例当中，如果一个摄影棚已经被占用，那么它将无法继续提供服务。我们希望将摄影棚的本职功能和状态管理功能分别处理，以达到代码的解耦并提高可维护性。通过创建真实摄影棚的代理，可以成功将代理功能和摄像、照片生成功能分开。

1.API描述

方法名	作用
ProxyPhotoRoom::ProxyPhotoRoom()	代理影棚的构造函数
ProxyPhotoRoom::shot()	摄像函数
ProxyPhotoRoom::produce()	生产照片函数
ProxyPhotoRoom::getState():int	获取当前影棚状态，0代表空闲，1代表使用
ProxyPhotoRoom::proxy(User)	根据用户状态和影棚状态进行对应代理操作

2.class diagram



17.Template

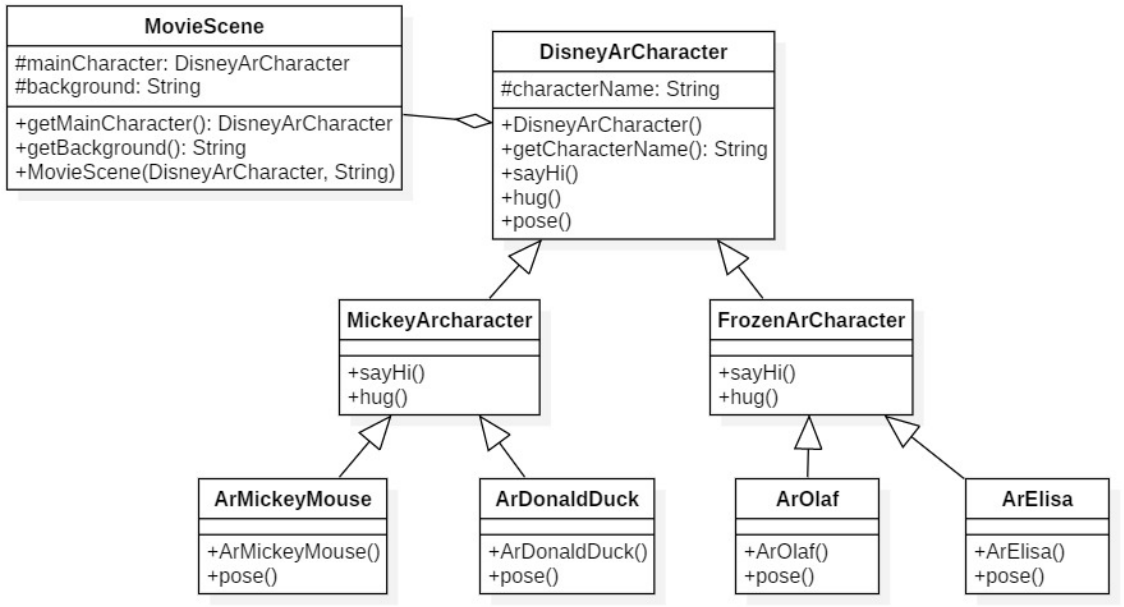
模板方法模式是一种行为设计模式， 它在超类中定义了一个算法的框架， 允许子类在不修改结构的情况下重写算法的特定步骤。

在本例中，不同的Ar角色虽然在打招呼、拥抱这两个行为上拥有相同的动作，但在招牌姿势上则有所不同。通过将这一函数作为模版留在子类函数中定义，使不同的父类对象在被调用时表现出不同的行为.

1.API描述

函数名	作用
DisneyArCharacter::sayHi()	直接调用进行问好
DisneyArCharacter::hug()	直接调用进行拥抱
DisneyArCharacter::pose()	根据子类定义的不同输出不同的招牌姿势

2.class diagram



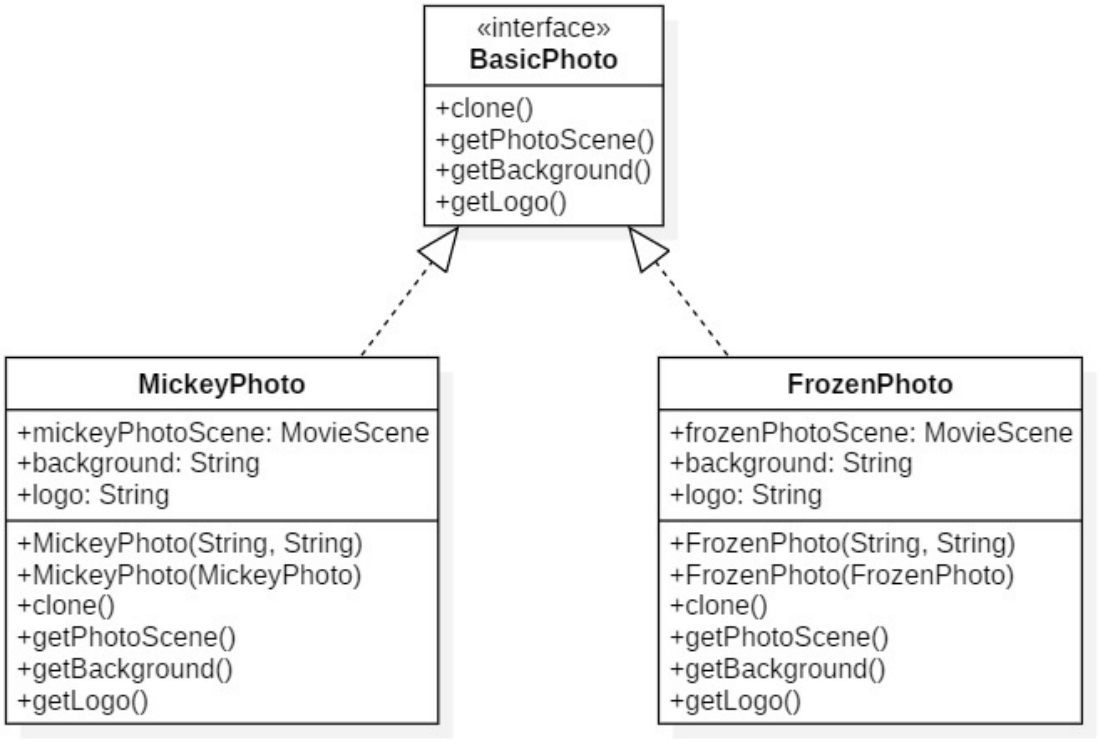
18.Prototype

原型模式是一种创建型设计模式， 使你能够复制已有对象， 而又无需使代码依赖它们所属的类。当用户想要复制一张照片时，如果没有采用原型设计模式，那么他就必须知道这张要复制的照片属于哪个类，只有这样他才能知道返回值时米老鼠主题的照片还是冰雪奇缘主题的照片。通过采用原型模式，用户只需将返回值定义为基础的照片类，而无需关心它的具体主题。

1.API描述

函数名	作用
BasicPhoto::clone():BasicPhoto	返回一个克隆照片
BasicPhoto::getPhotoScene:MovieScene	返回照片中的场景类
BasicPhoto::getBackground:String	返回照片的背景文字
BasicPhoto::getLogo:String	返回照片的主题Logo

2.class diagram



19.State

状态模式是一种行为设计模式， 让你能在一个对象的内部状态变化时改变其行为， 使其看上去就像改变了自身所属的类一样。

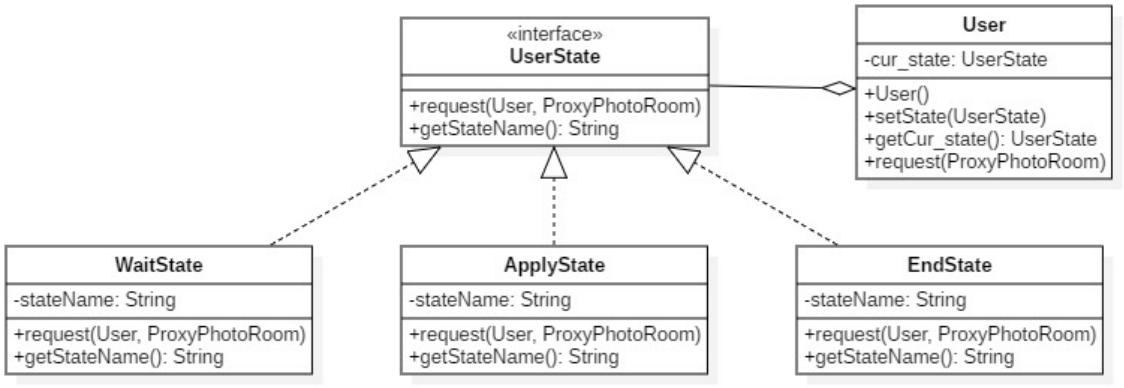
在本例中，用户在不同状态发出请求时希望得到不同的回复。通过单独设计状态类，并将其作为一个属性封装在用户类当中，可以实现状态间的切换，并对用户隐藏背后的复杂性。

在重新增加状态或修改状态时，也只需重新继承状态类即可。

1.API描述

函数名	作用
UserState::request(User, ProxyPhotoRoom)	对不同状态进行切换
UserState::getStateName():String	获取状态名
User::setState(UserState)	设置用户状态
User::getCur_state():User_State	获取用户状态，返回一个状态类
User::request(ProxyPhotoRoom)	用户对指定摄影棚发起一次请求

2.class diagram

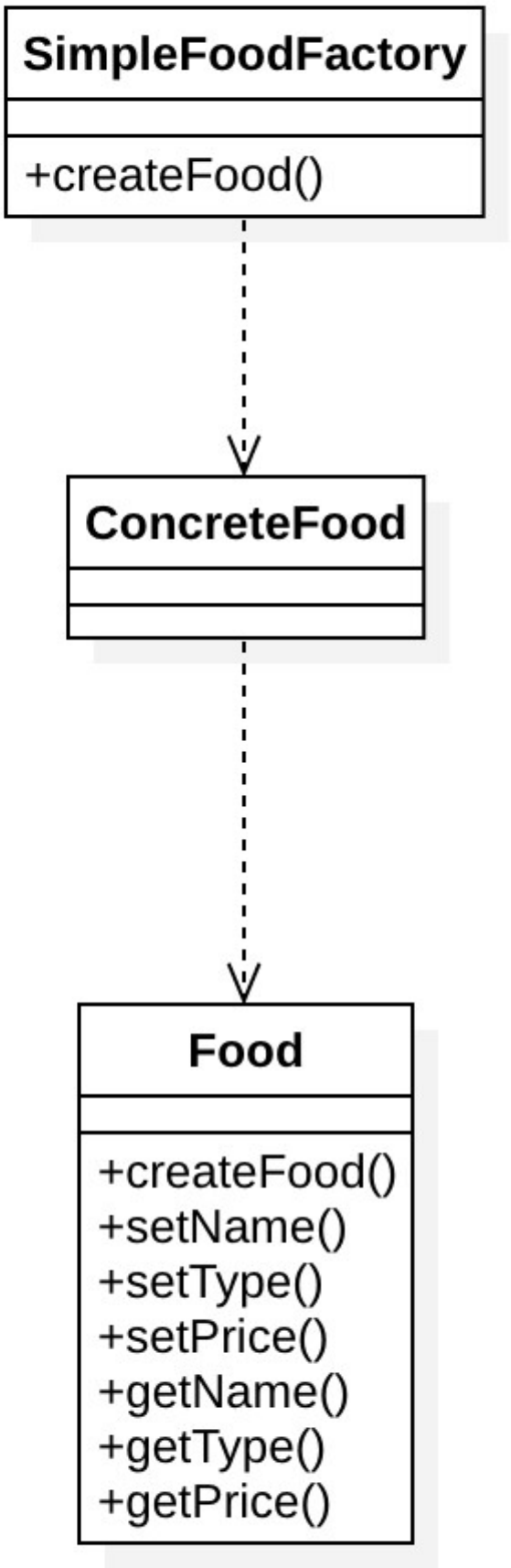


20.simple factory

1.API描述

函数名	作用
createFood()	提供了一个创造食物的接口
setPrice(int price)	设置食物价格

2.class diagram


SimpleFoodFactory
+createFood()
ConcreteFood
Food
+createFood()
+setName()
+setType()
+setPrice()
+getName()
+getType()
+getPrice()

21.Builder

生成器模式是一种创建型设计模式， 使你能够分步骤创建复杂对象。 该模式允许你使用相同的创建代码生成不同类型和形式的对象。

优点

1.你可以分步创建对象， 暂缓创建步骤或递归运行创建步骤。

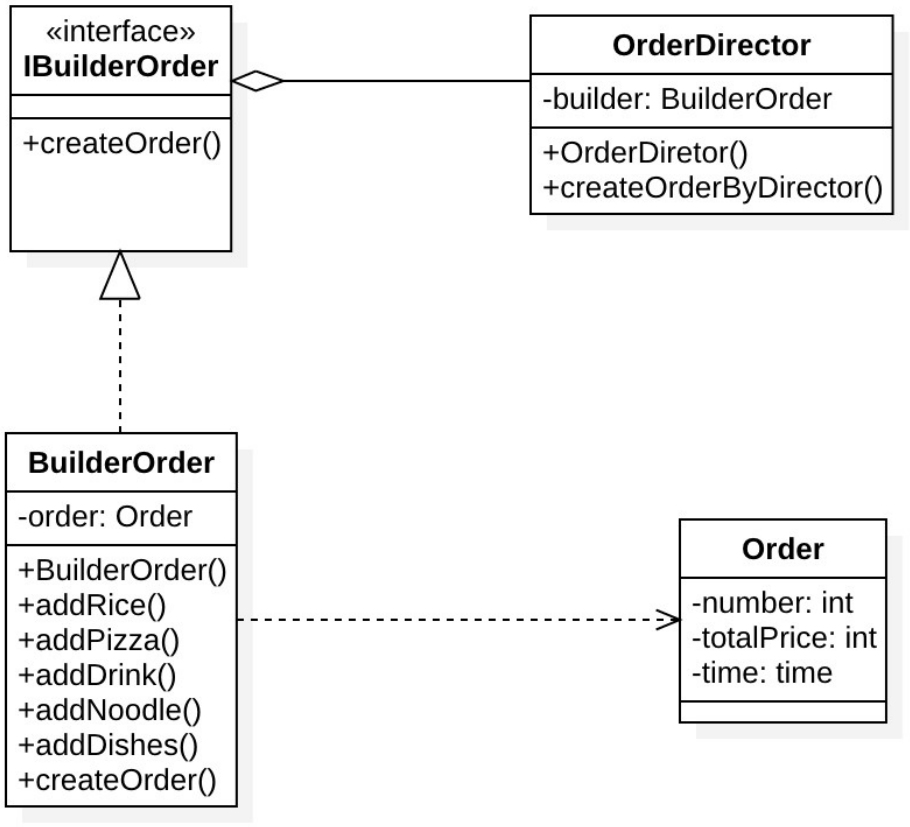
2.生成不同形式的产品时， 你可以复用相同的制造代码。

3. 单一职责原则。 你可以将复杂构造代码从产品的业务逻辑中分离出来。

1.API

函数名	作用
createOrder()	创建一个新的订单
addSth()	向订单增加内容
IBuilderOrder（）	提供创建订单的接口
display()	用来展示订单信息
createOderByDirector(int[] number)	指导建造者建造什么样的订单

2.class diagram



22.Composite

组合模式是一种结构型设计模式， 你可以使用它将对象组合成树状结构， 并且能像使用独立对象一样使用它们。

优点

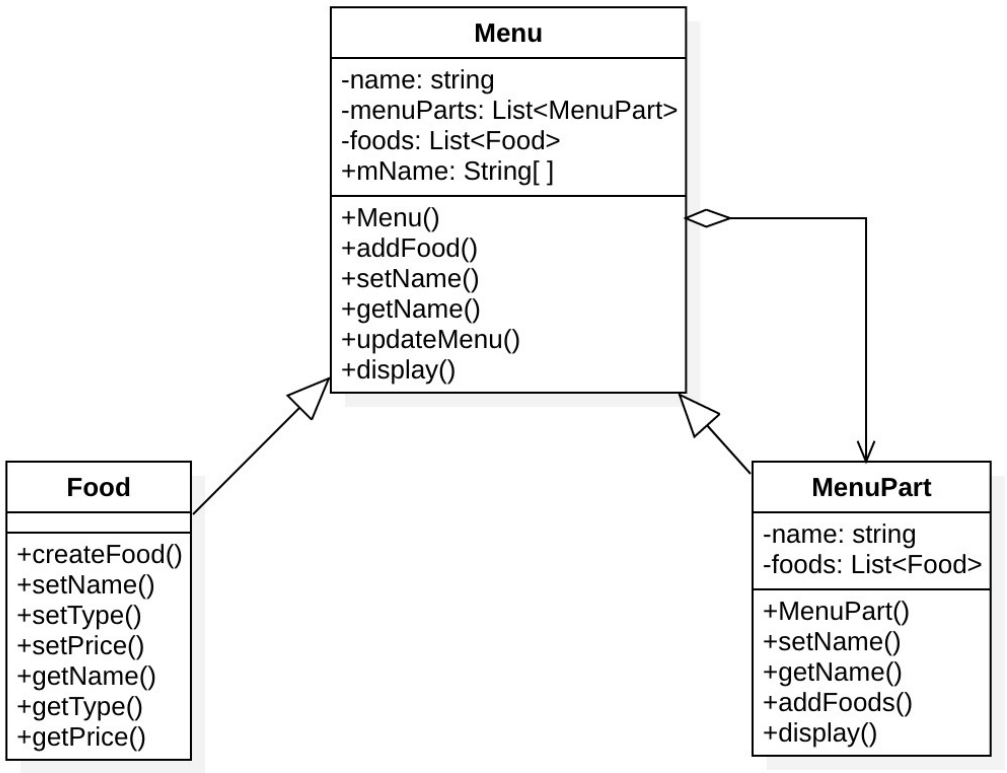
1.你可以利用多态和递归机制更方便地使用复杂树结构。

2.开闭原则。 无需更改现有代码， 你就可以在应用中添加新元素， 使其成为对象树的一部分。

1.API描述

函数名	作用
addFood(Food f)	方法向菜单中添加新的食物
updateMenu()	将新食物自动添加到分菜单中
display()	将当前菜单格式化输出

2.class diagram



23.Flyweight

享元模式是一种结构型设计模式， 它摒弃了在每个对象中保存所有数据的方式， 通过共享多个对象所共有的相同状态， 让你能在有限的内存容量中载入更多对象。

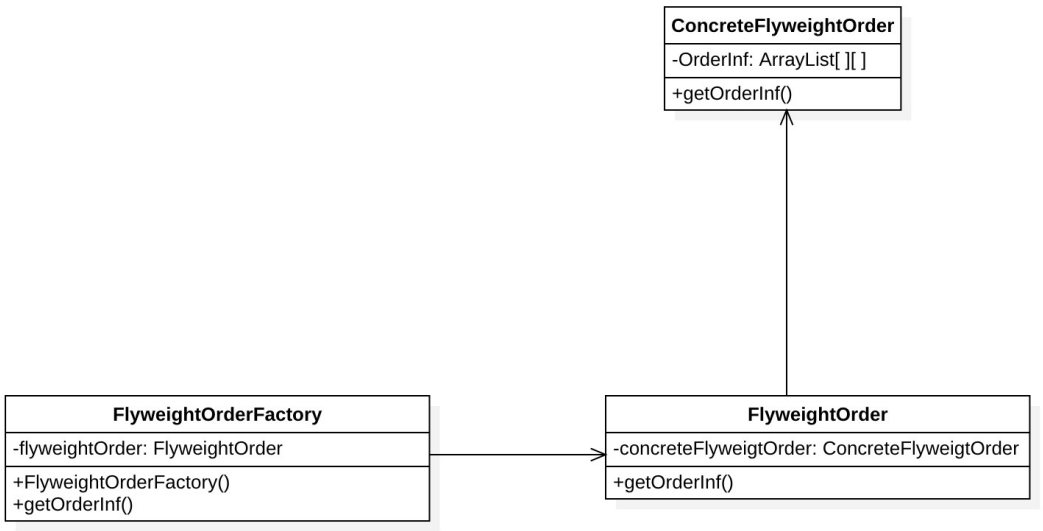
优点

如果程序中有很多相似对象， 那么你将可以节省大量内存。

1.API描述

函数名	作用
public int[] getOrderInf(int number)	返回套餐内食物的参数，以用来创建订单
public int[] getOrderInf(int number)	根据判断条件返回对应的参数。
public int[] getOrderInf(int number)	提供外界访问内部套餐数据的接口

2.class diagram



24.Singleton

单例模式是一种创建型设计模式， 让你能够保证一个类只有一个实例， 并提供一个访问该实例的全局节点。

优点

1.你可以保证一个类只有一个实例。

2.你获得了一个指向该实例的全局访问节点。

3.仅在首次请求单例对象时对其进行初始化。

1.API描述

函数名	作用
public Order createOderByDirector(int[] number)	完成全部对建造者的指导工作
public Food createFood(String name,int price,String type)	提供了一个创造食物的接口
public int[] getOrderInf(int number)	满足所有获取套餐信息的需求