III.项目中所有设计模式的汇总表，格式如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 设计模式名称 | 实现个（套）数 | 何处体现 | 备注说明 |
| 1 | Template pattern | 1 |  |  |
| 2 | Command pattern | 1 | 运动会小卖部 ，把订单请求封装成一个对象，让下订单和执行订单的操作分隔开 |  |
| 3 | Composite pattern | 1 | 运动会赛事导航，运动会的项目可以分成许多种类，我们把这些运动项目组合成树状的层次结构，来表示整体与部分的关系，也就是组合模式 |  |
| 4 | FrontController | 1 | 运动会服务中心，提供一个集中的请求处理机制，动物们的请求被前端控制器统一处理，然后通过调度器调度到具体的处理程序。 |  |

III.设计模式详述，按下列内容给出详细说明

3.1 设计模式A

3.1.1 实现API 描述

3.1.2 类图（还可添加流程图、时序图、等等其他UML图辅助说明）

3.1.3 其他内容（如果有的话）

3.2 设计模式 B

……

3.1 Command Pattern

3.1.1 实现API描述

Order类为订单抽象类，相当于模式中的Command类，相当于声明对订单操作的接口，它有的成员变量name、num和receiver，分别表示订单的类型名称、订单编号（每个订单具有唯一的编号），成员函数ExecuteOrder表示执行订单，Command类有子类BandageCmd、DrinkCmd、FoodCmd、PosterCmd,把请求命令对象化，相当于模式中的ConcreteCommand，是具体的订单，实现了具体的函数ExecuteOrder表示具体的操作。Clerk类相当于Invoker，用于接收小动物们的订单，并且调用具体的命令，包含成员变量m\_commandList、order\_sum，表示生成的订单列表和订单总数，成员函数SetOrder、UndoOrder、Notify，表示下订单、撤回订单、通知执行。ShopAssitant类相当于模式中的Receiver，包含成员函数BuyDrink、BuyFood、BuyBandage、BuyPoster，表示最终处理具体的订单操作。

具体实现API的列表如下：

ShopAssitant类：

void BuyDrink(int num); //买饮料的具体执行函数

void BuyFood(int num); //买零食的具体执行函数

void BuyPoster(int num);//买横幅海报的具体执行函数

void BuyBandage(int num); //买绷带的具体执行函数

Clerk类：

void SetOrder(Order\* temp);//下订单操作

void UndoOrder(int num);//撤回订单操作

void Notify();// 通知执行

BandageCmd类：

virtual void ExecuteOrder()//执行具体命令

DrinkCmd类：

virtual void ExecuteOrder()//执行具体命令

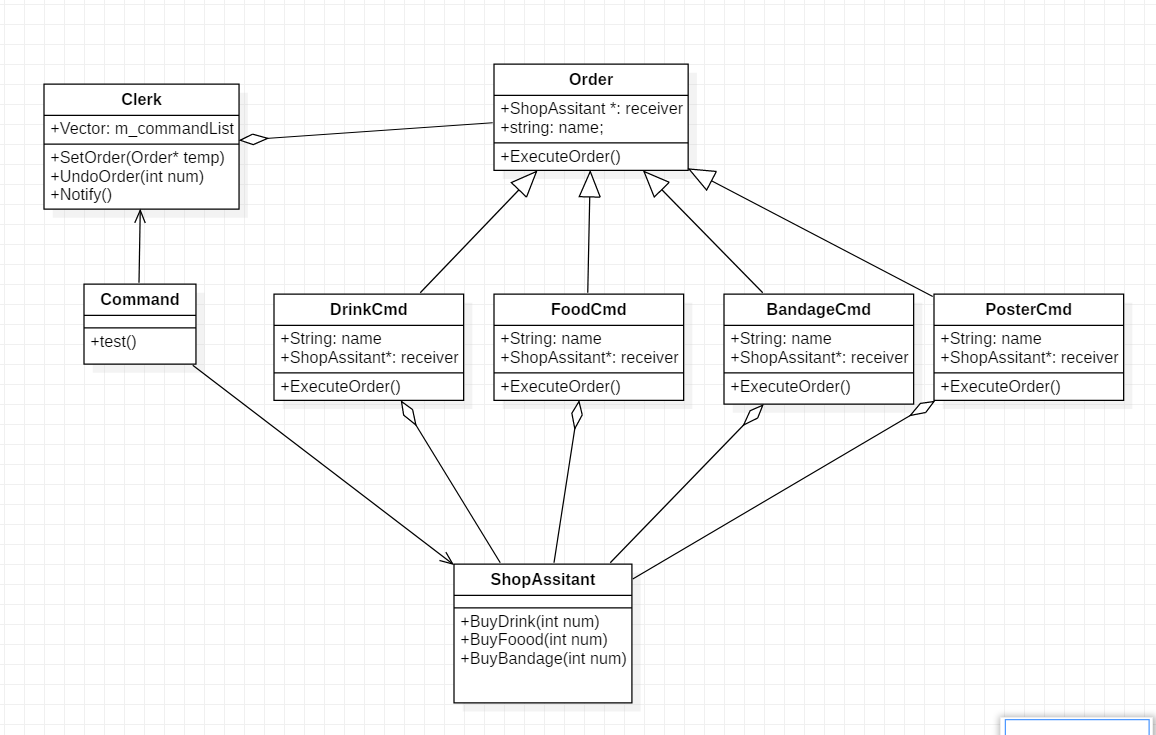
FoodCmd类：

virtual void ExecuteOrder()//执行具体命令

PosterCmd类：

virtual void ExecuteOrder()//执行具体命令

3.1.2 类图



3.1.3 故事背景

为了方便来观看比赛的小动物观众们更好地了解本次运动会的赛事流程，我们专门设置了赛事导航，里面包括本次运动会体育项目的时间、地点和介绍，帮助小动物们玩转本次运动会。

3.2 Composite Pattern

3.1.1 实现API描述

MenuComponent类相当于模式中的Component角色，是抽象基类，表示比赛项目菜单的基类，包含成员变量m\_name、m\_place、m\_time、m\_description,分别表示体育项目的名称、地点、时间、描述，包含成员函数Add、Remove、Show、Print，分别表示添加体育项目、移除体育项目、显示体育项目和打印所有的体育项目，Menu类相当于Composite角色，真正的菜单类，它维护了一个MenuComponent的vector数组，这个数组里面既可以有menu也可以有menuItem，它重写了基类的Add、Remove、Show、Print函数。menuitem类表示的是菜单中具体的比赛项目，比如陆地比赛的赛跑、爬树等等，不包含子菜单，它对基类的方法进行了重写，相当于composite模式里面的leaf。

具体实现API的列表如下：

MenuComponent类：

virtual void Add(MenuComponent \*pCom) {}//添加体育项目的函数

virtual void Remove(MenuComponent \*pCom) {}//移除体育项目的函数

virtual void Show(int depth) {}//显示体育项目的函数

virtual void print() {}//打印体育项目

MenuItem类：

void print();//打印体育项目

Menu类：

void Add(MenuComponent \*pCom);//添加体育项目的函数

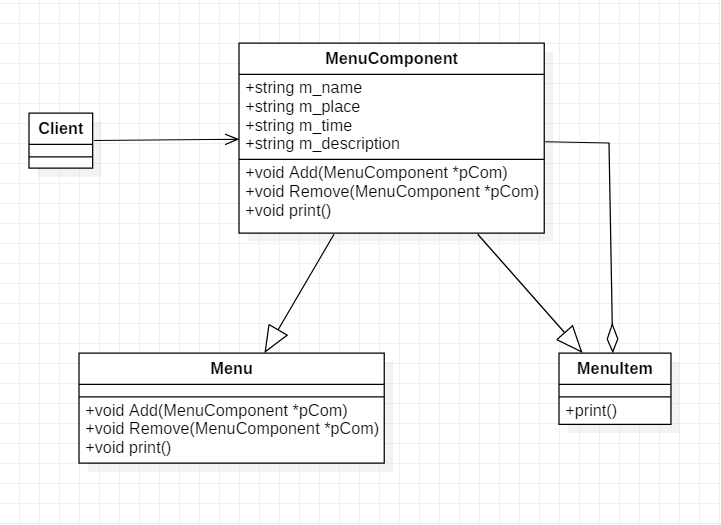
void Show(int depth); //显示体育项目的函数

void remove(MenuComponent \*pCom);//移除体育项目的函数

MenuComponent\* getChild(int i); //获取子菜单

void print();//打印体育项目

3.1.2 类图



3.1.3 故事背景

我们设置了体育场小卖部，运动员和观众可以在小卖部购买需要的物品，比如零食、饮料、应援物、创口贴绷带等等，因为来购买的小动物可能很多，而且有的小动物会来店铺里面买，有些小动物会希望小卖部把他们需要的物品送到他们手中，为避免产生混乱，每一个顾客会生成一个购物清单，客户把他们的需求写在清单上，这样店员就可以按顺序实现需求，客户也可以方便的撤销。

3.1 FrontController Pattern

3.1.1 实现API描述

OlympicServiceCenter类是模式的最顶级类，也是用户使用界面，相当于模式中的Front Controller角色，观众或运动员向运动会服务中心发出一些请求，我们将请求用我们的请求调度器调度给相应的请求处理器来解决问题，并且我们能够记录下所有的请求，实现日志的功能。RequestDispatcher类相当于模式中的Dispatcher角色，它负责根据对应请求的种类，找到合适的处理器，并将请求转交给它来进行处理。 RequestSolvable类相当于模式中的View角色，是请求处理器的基类，相当于JAVA的接口。它们都具有一个处理请求的方法，并且可以获得它自身的请求种类。FacilityProblemSolver类、FoodProblemSolver类、SafeProblemSolver类、SafeProblemSolver类表示具体的请求处理器，分别解决具体的请求，包括设施问题、食品问题、安全问题、比赛问题。

具体实现API的列表如下：

OlympicServiceCenter：

void trackRequest(string requestType, string requestContent);// 记录请求，实现日志功能。

void dispatchRequest(string requestType, string requestContent);// 分配请求，唯一的共有函数，是所有外界用户使用它的入口。

RequestDispatcher:

void dispatch(string requestType, string requestContent); //分配器主函数,它负责根据对应请求的种类，找到合适的处理器，并将请求转交给它来进行处理。

RequestSolvable:

virtual void solveRequest(string request)=0;//解决实际请求。

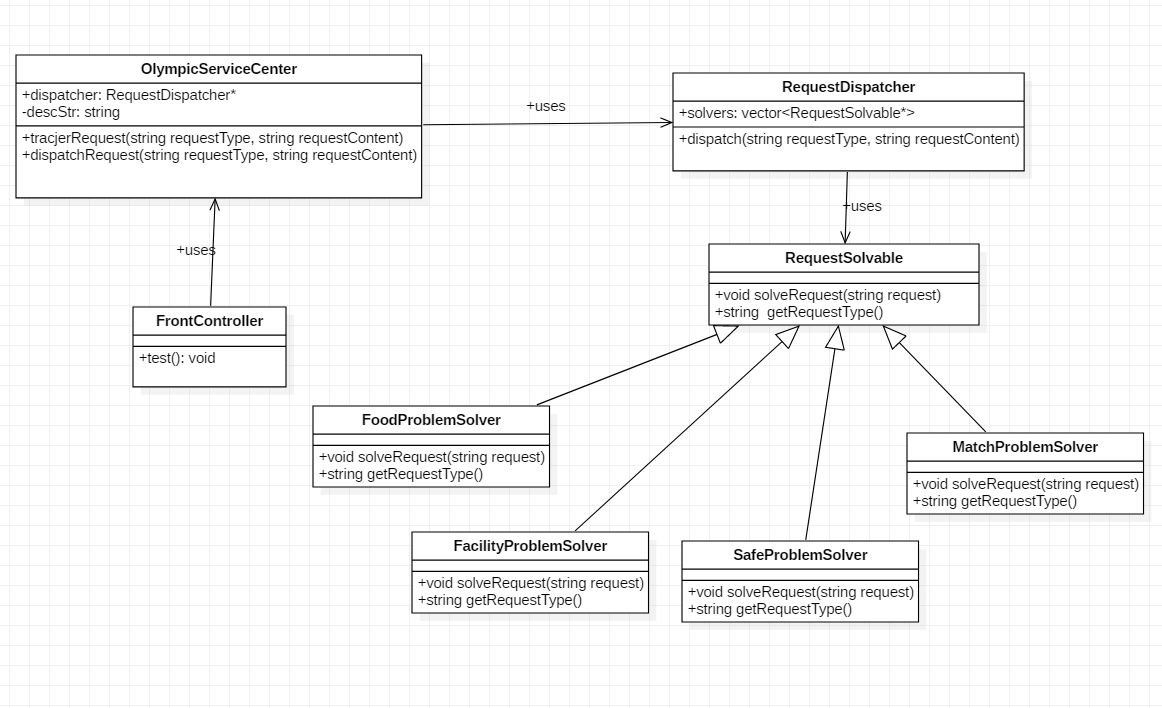
virtual string getRequestType() = 0; //获取自身的请求类型。

FacilityProblemSolver\FoodProblemSolver\SafeProblemSolver\SafeProblemSolver:

virtual string getRequestType();//获取自身的请求类型。

virtual void solveRequest(string request); //解决实际请求。

3.1.2 类图



3.1.3故事背景

在运动会进行的过程中，无论是来观看比赛的观众、还是运动员、赛训组，都可能会遇到各种各样的问题，因此，运动会主办方设置了动物运动会服务中心，专门为小动物门提供各种服务以及问题解决方案。小动物们如果在运动会开展期间遇到什么问题，都可以来服务中心咨询和寻求帮助，服务中心会根据小动物们问题的不同，把他们的请求分类，然后采取不同的措施解决。