# 3.12.Decorator

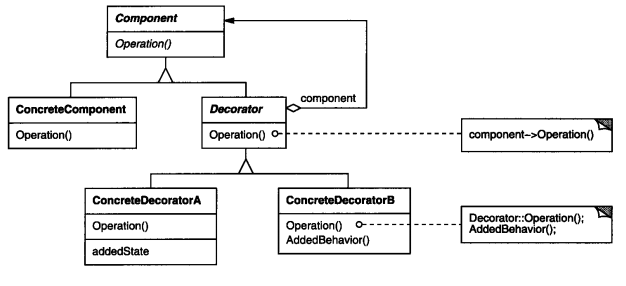
## 3.12.1设计模式简介：

该设计模式的意图是动态地给一个对象添加一些额外的职责。就增加功能来说， Decorator模式相比生成子类更为灵活。

Decorator模式适用于以下场景：

1. 在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。
2. 处理那些可以撤消的职责。
3. 当不能采用生成子类的方法进行扩充时。一种情况是，可能有大量独立的扩展，为支持每一种组合将产生大量的子类，使得子类数目呈爆炸性增长。另一种情况可能是因为类定义被隐藏，或类定义不能用于生成子类。

Decorator模式的结构如下：



上图结构中有Component，ConcreComponent，Decorator和ConcreteDecorator四个参与者。Component定义一个对象接口，可以给这些对象动态地添加职责; ConcreComponent定义一个对象，可以给这个对象添加一些职责; Decorator维持一个指向Component对象的指针，并定义一个与 Component接口一致的接口；ConcreteDecorator向组件添加职责。

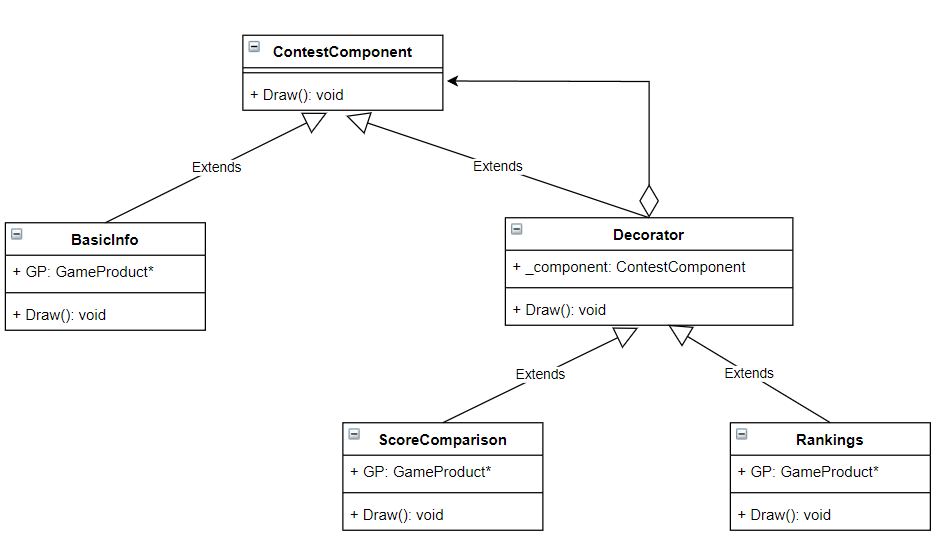
## 3.12.2在项目中的应用：

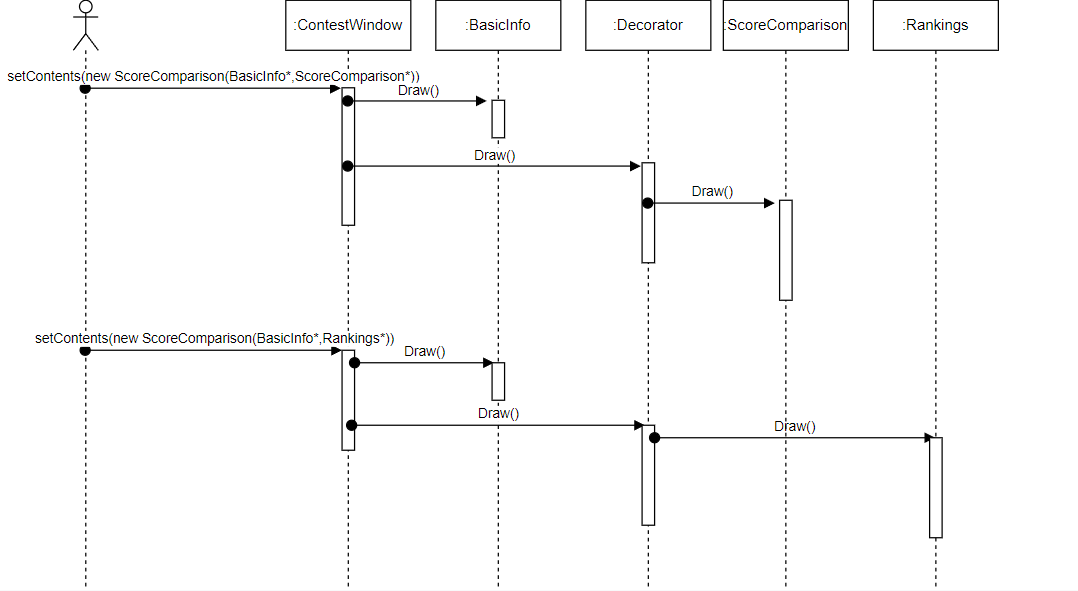
在本项目中，需要实时展示比赛场景。而比赛场景可以分成基本信息的打印（比赛名字，时间，场地，选手名称、国籍等）、以及该场比赛特有的信息——比如乒乓球比赛需要打印双方选手的比分；跑步需要记录时间和各个选手的排名。可以用decorator很方便地把基本信息和扩展信息组合起来。

## 3.12.3 api描述：

|  |  |  |
| --- | --- | --- |
| 类名 | 函数声明 | 作用 |
| BasicInfo | virtual void **Draw**(); | 打印基本信息 |
| Decorator | virtual void **Draw**(); | 由组合类来决定Draw()函数地功能。 |
| ScoreComparison | virtual void **Draw**(); | 实时展示乒乓球比赛中的比分。 |
| Rankings | virtual void **Draw**(); | 实时展示跑步比赛中的排名、各个选手已经跑过的距离和跑步速度。 |
| ContestWindow | void **SetContents**(**ContestComponent**\* contents); | 按照传入的参数来决定如何设置画布内容 |

## 3.12.4类图和时序图：





# 3.27.Immutable

## 3.27.1设计模式简介：

一个对象的状态在对象创建之后就不再变化，这就是所谓的immutable模式。Immutable模式具于以下优点：

1. 因为不能修改一个不变对象的状态，所以可以避免由此引起的不必要的程序错误；换言之，一个不变对象要比可变对象更加容易维护。
2. 因为没有任何一个线程能够修改不变对象的内部状态，一个不变对象自动就是线程安全的，这样就可以省掉处理同步化的开销。

## 3.27.2在本项目中的应用：

在本项目中，需要展示介绍展馆，而这一部分信息是固定的，不会改变的。

## 3.27.3 api描述：

|  |  |  |
| --- | --- | --- |
| 类名 | 函数声明 | 作用 |
| Stadiums | void **show**(); | 从整体、竞价和设计等方面对体育馆进行介绍 |

## 3.27.4类图：

