合照场景设计模式简述

包含Proxy/State/Abstract Factory/Bridge

27/10/2019

场景简介

本次设想的场景是迪士尼乐园中开利用Ar技术和虚拟电影角色合照的项目。允许创建用户、创建摄影棚，并由用户自定义同哪位电影的哪位虚拟角色进行合照，并在生成的照片附上定制的话语和Logo。

Abstract Factory设计模式

1. 设计模式简述

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。

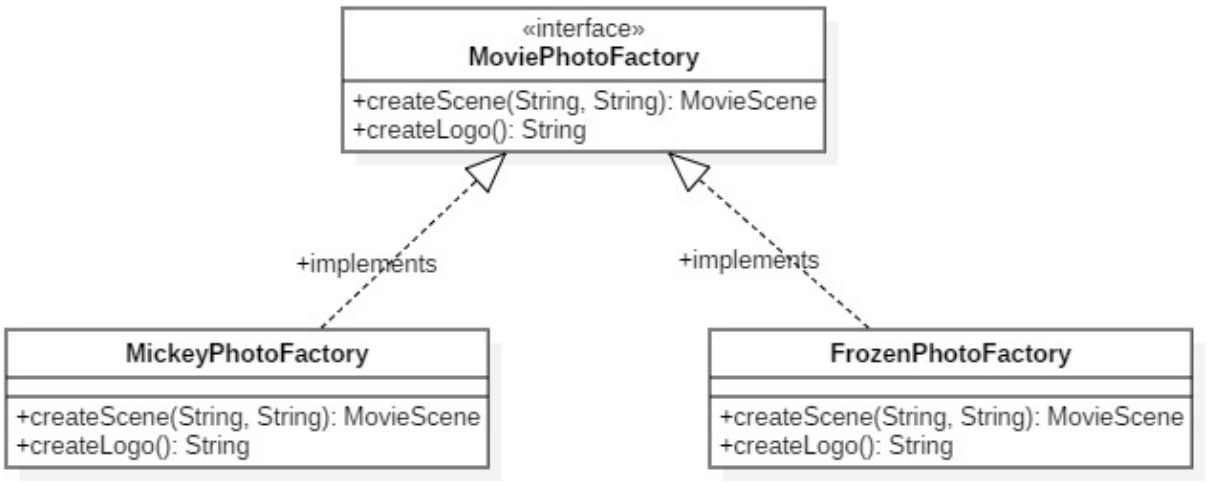
在本例中，用户创建的照片需要包含一位虚拟角色和电影Logo，而且他希望这位虚拟角色和Logo匹配正确。通过抽象工厂模式使米老鼠电影场景和冰雪奇缘电影场景同时继承合照工厂类，用户可以直接连续调用生成场景函数和生成Logo函数，而不必去在意他们是否匹配。

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）

2. 一套API

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

3. UML类图



Bridge设计模式

1. 设计模式简述

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。

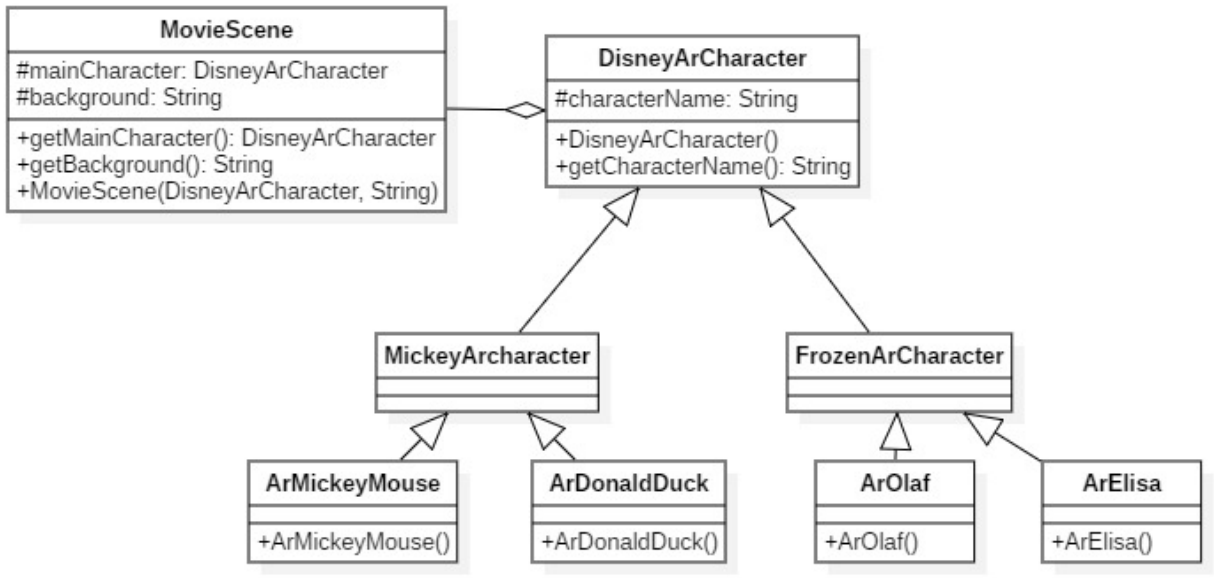
在本例中，一个电影场景包含一位Ar角色，此时如果将该角色的所有信息都封装到场景类当中，将会造成场景类的大量冗余。在后续新增角色时，也需要改动大量原有代码。

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。

2. 一套API

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

3. UML类图



Proxy设计模式

1. 设计模式简述

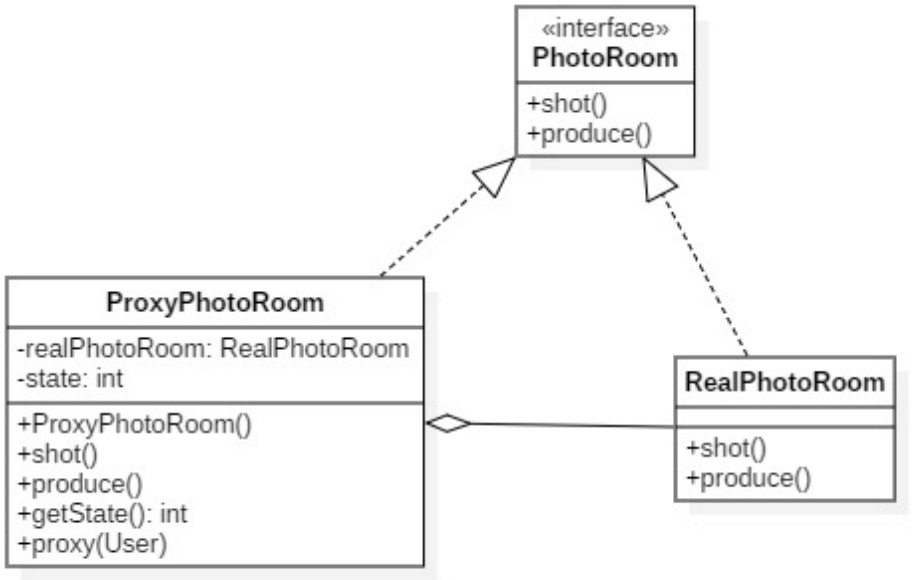
代理模式是一种结构型设计模式， 让你能够提供对象的替代品或其占位符。 代理控制着对于原对象的访问， 并允许在将请求提交给对象前后进行一些处理。

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。

2. 一套API

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

3. UML类图



Template设计模式

1. 设计模式简述

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

2. 一套API

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

3. UML类图
（等下就有）

Prototype设计模式

1. 设计模式简述

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。

2. 一套API

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

3. UML类图
（马上就有）

State设计模式

1. 设计模式简述

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。

2. 一套API

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

3. UML类图