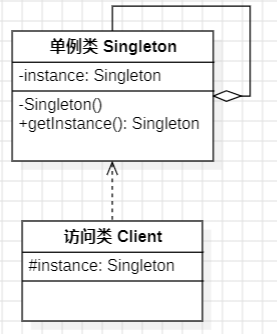
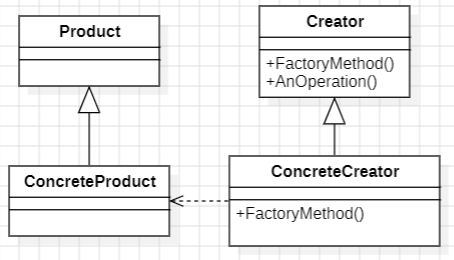
1. 单例模式：



应用场景：计算机系统、Windows的回收站、操作系统中的文件系统、多线程中的线程池、显卡的驱动程序对象、打印机的后台处理服务、应用程序的日志对象、数据库的连接池、网站的计数器、Web应用的配置对象、应用程序中的对话框、系统中的缓存等常常被设计成单例。

1. 工厂方法模式：



应用场景：

UI控件库：UI库中通常包含多种不同的控件，如按钮、文本框、标签等。工厂方法可以用来创建这些控件的实例，从而使用户能够轻松扩展控件库，添加新的控件类型。

数据库驱动程序：数据库访问框架通常需要支持不同种类的数据库，如MySQL、Oracle、SQL Server等。使用工厂方法模式，可以为每个数据库类型创建一个具体的数据库连接工厂，以便创建适当的数据库连接对象。

日志记录器：在应用程序中记录日志是一项常见任务。使用工厂方法，可以轻松切换不同的日志记录器（如文件日志、数据库日志、控制台日志等）而无需修改现有代码。

游戏开发：在游戏中，通常需要创建大量的游戏对象，如角色、武器、敌人等。工厂方法可以用于创建这些游戏对象，支持扩展游戏中的新角色和物品。

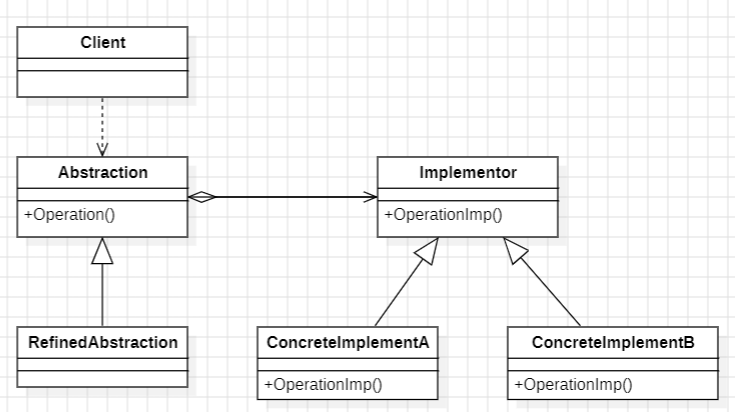
插件系统：应用程序和框架通常使用插件来扩展功能。工厂方法可用于创建插件实例，以便轻松添加新的插件类型。

图形和图像处理：在图形和图像处理应用中，可以使用工厂方法来创建各种图形对象和图像过滤器，以便支持不同类型的图形处理操作。

车辆制造：在汽车制造业中，工厂方法模式可以用于创建不同型号和品牌的汽车，使制造商能够根据市场需求扩展其产品线。

工厂方法设计模式在这些应用场景中有助于减少代码的耦合性，支持扩展和维护，以及提高代码的可读性。通过使用工厂方法，可以更容易地添加新的产品或对象类型，同时保持现有的客户端代码不变。

1. 桥接模式



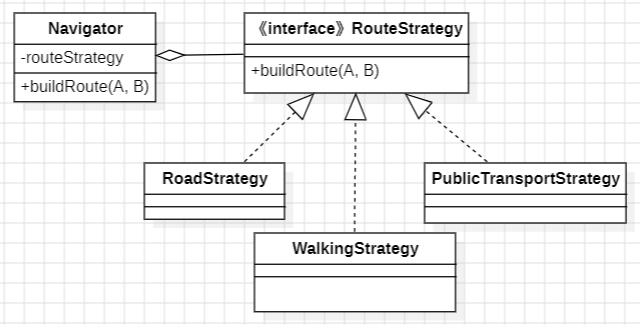
应用场景：

1、在抽象和具体实现之间需要增加更多的灵活性的场景。

2、一个类存在两个或多个独立变化的维度，而这两个或多个维度都需要独立进行扩展。

3、不希望使用继承，或因为多层继承导致系统类的个数剧增。

1. 策略模式



应用场景：

有一系列算法，可以封装每个算法并使它们可以互换的时候，可以使用策略模式。它使得算法可以独立于客户端而变化。