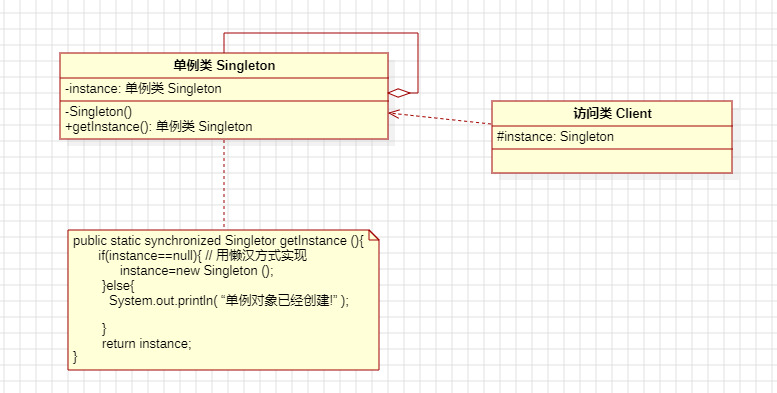
**单例模式**



使用场合：

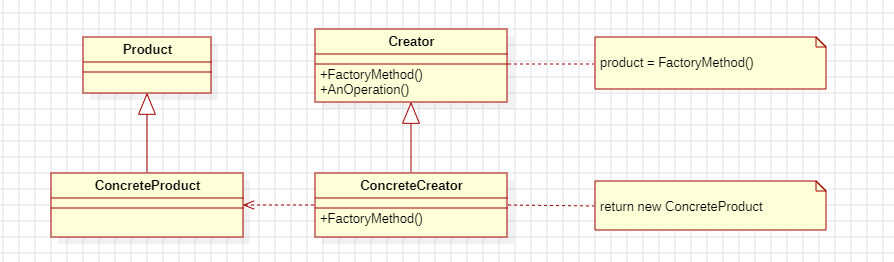
**1.全局访问点**：当一个对象需要在系统中的多个部分之间提供一个唯一的访问点时，可以使用单例模式。这有助于确保所有的代码都使用同一个实例，避免了多个实例之间的状态不一致问题。

**2.资源共享**：当有一些共享的资源，例如数据库连接、文件系统等，需要在整个系统中被共享时，单例模式可以确保这些资源只被实例化一次，避免资源浪费。

**3.控制实例数量**：在某些情况下，系统要求一个类只能有一个实例，例如线程池、缓存管理器等，这时可以使用单例模式来控制实例数量。

总体而言，单例模式适用于那些需要全局唯一实例，并且在系统的整个生命周期内保持一致性的情况。

**工厂方法（factory method)模式**

****

使用场合：

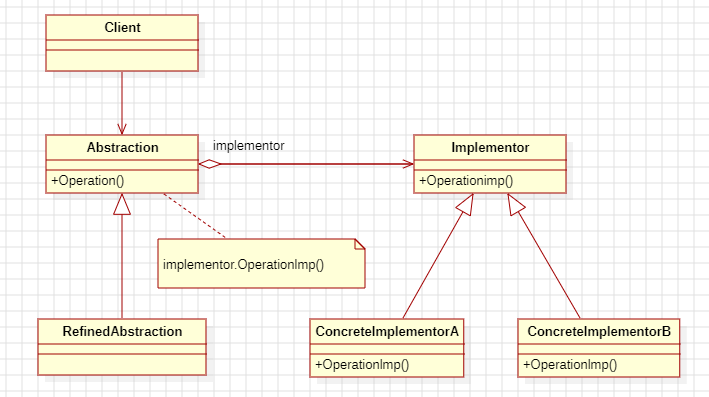
**1.对象创建延迟到子类**：当一个类无法预知它需要创建哪个具体子类的实例时，可以使用工厂方法模式。这样，具体的子类可以通过重写工厂方法来决定实例化哪个类。

**2.消除对象的直接依赖**：工厂方法模式允许客户端代码依赖于抽象的接口或类，而不是具体的类实例。这样可以减少代码的耦合度，使系统更易于扩展和维护。

**3.创建对象的过程需要灵活性**：如果创建对象的过程涉及到复杂逻辑、条件分支或需要动态确定对象类型，工厂方法模式可以提供灵活性，让子类决定实例化哪个对象。

总体而言，工厂方法模式适用于需要在运行时选择对象创建逻辑，以及需要将对象创建和使用代码解耦的情况。

**桥接模式（Bridge Pattern）**



使用场合：

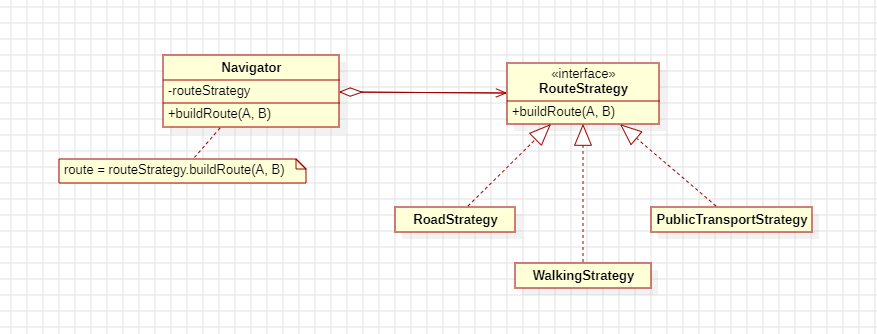
**1.抽象部分和实现部分需要独立扩展**：如果系统中有两个独立变化的维度，且希望能够在运行时组合这两个维度，桥接模式是一个合适的选择。例如，图形界面系统中，抽象部分可以是不同的窗口，实现部分可以是不同的操作系统。

**2.抽象部分和实现部分都需要可以扩展**：如果希望能够独立地扩展抽象部分和实现部分，而不影响彼此，桥接模式是一个有力的工具。

**3.需要避免静态绑定**：如果在编译时希望避免抽象部分与实现部分的静态绑定，桥接模式可以在运行时进行动态绑定，提高灵活性。

总体而言，桥接模式适用于希望将抽象部分和实现部分解耦，使它们可以独立变化，并且可以在运行时动态地将不同的抽象部分和实现部分组合在一起的情况。

**策略模式（Strategy Pattern）**



使用场合：

**1.需要在运行时动态选择算法**：如果一个系统有多个算法，而且需要在运行时根据具体情况选择其中之一，策略模式是一个合适的选择。

**2.算法经常发生变化**：如果系统中的算法经常发生变化，而且这种变化不应该影响到使用算法的客户端，策略模式可以使得算法的变化独立于客户端。

**3.避免使用条件语句进行算法选择**：如果希望避免在代码中使用大量的条件语句来选择算法，策略模式可以通过封装不同的算法并使其互换，使代码更加清晰、灵活。

总体而言，策略模式适用于需要定义一系列算法、并且能够在运行时动态选择其中一个算法的情况。