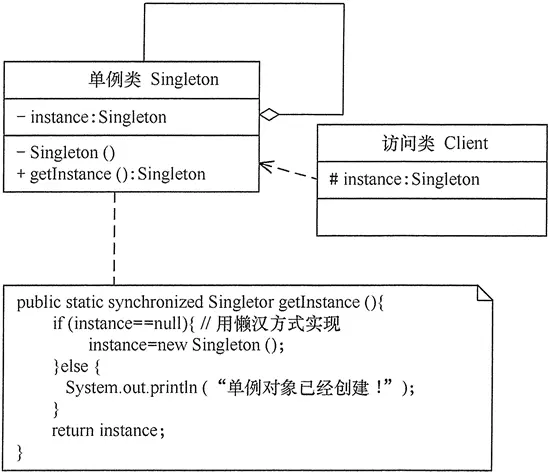
**1、单例模式**



应用场合：

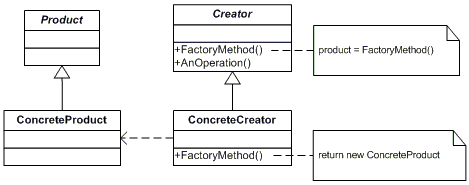
1. 系统中只需要有一个实例：在某些情况下，整个系统只需要有一个实例存在，使用单例模式可以确保只有一个实例被创建和使用。

2. 资源共享的情况下：当多个对象需要共享同一个资源时，使用单例模式可以保证该资源被正确地共享和管理。

3. 需要频繁创建和销毁对象的情况下：在某些情况下，创建和销毁对象的成本很高，因此需要尽量减少对象的创建和销毁次数。使用单例模式可以避免频繁创建和销毁对象，提高系统性能。

4. 控制全局唯一性：在某些情况下，需要确保某个类的实例在整个系统中是唯一的，以便于其他对象可以方便地访问和使用该实例。单例模式可以实现全局唯一性的控制。

**2、工厂方法模式**



应用场合：

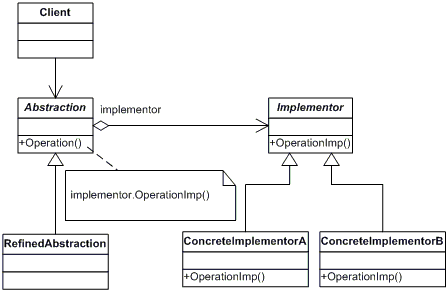
1. 对象的创建需要根据不同条件或参数进行区分：当需要根据不同的条件或参数创建不同类型的对象时，可以使用工厂方法模式。

2. 希望将对象的创建逻辑与使用逻辑分离：工厂方法模式可以将对象的创建过程封装在具体的工厂类中，而客户端只需要关注对象的使用即可。这样可以降低耦合性，并且使得代码更加易于维护和扩展。

3. 需要动态切换和扩展对象的创建逻辑：工厂方法模式可以通过添加新的具体工厂类来扩展对象的创建逻辑，而不需要修改已有的代码。这样可以方便地动态切换和扩展对象的创建方式，满足不同的需求。

4. 通过接口或抽象类来统一管理对象的创建：工厂方法模式可以通过接口或抽象类来定义工厂方法，从而统一管理对象的创建。这样可以提高代码的可读性和可维护性，同时也方便了代码的测试和重构。

**3、桥接模式**

****

应用场合：

1. 当一个类存在多个维度的变化时，使用桥接模式可以将各个维度的变化解耦，使得它们可以独立地变化。例如，当有多个不同类型的手机和多个不同类型的软件需要进行组合时，可以使用桥接模式将手机和软件分别抽象出来，使它们可以独立地变化，而不需要为每一个组合都创建一个新的类。

2. 当需要在运行时动态切换实现的时候，使用桥接模式可以方便地进行切换。例如，在一个系统中，有多种不同的颜色和形状需要进行组合，可以使用桥接模式将颜色和形状分离开来，从而可以在运行时动态地选择和切换不同的颜色和形状的组合。

3. 当希望通过继承来扩展功能时，使用桥接模式可以避免类爆炸的问题。例如，当有多个不同类型的电视和多个不同类型的遥控器需要进行组合时，可以使用桥接模式将电视和遥控器分别抽象出来，并通过继承来扩展不同类型的电视和遥控器的功能。

4. 当需要对抽象和实现进行解耦时，使用桥接模式可以将抽象和实现分离开来。这样可以使得抽象和实现可以独立地变化，而不会相互影响。例如，在一个系统中，有多个不同类型的数据库和多个不同类型的数据访问对象需要进行组合，可以使用桥接模式将数据库和数据访问对象分离开来，从而实现解耦。

**4、策略模式**



应用场合：

1. 当一个系统需要支持多种算法或行为，并且这些算法或行为可以在运行时动态切换时，可以使用策略模式。例如，一个电商平台的购物车结算功能可能需要支持不同的折扣策略，而这些策略可以根据用户的会员等级或促销活动进行动态选择。

2. 当一个类有多个行为或算法，而且不希望将这些行为或算法耦合在一起时，可以使用策略模式。例如，一个支付系统中的支付方式可能有多种，每种支付方式都有自己独特的实现逻辑，使用策略模式可以将每种支付方式封装成一个策略类，使得它们可以独立地变化和扩展。

3. 当多个相关的类只有行为或算法不同，但是其他部分的代码都相同或相似时，可以使用策略模式来避免重复代码。例如，一个游戏中的敌人角色可能有多种攻击方式，但是它们的移动、防御等行为都是相同的，使用策略模式可以将攻击方式封装成策略类，减少代码冗余。

4. 当一个类的行为或算法需要根据不同的条件进行选择时，可以使用策略模式。例如，一个电商平台中的商品排序功能可能根据用户的选择和搜索关键字来确定排序规则，使用策略模式可以将不同的排序规则封装成策略类，并根据条件动态选择合适的排序策略。