设计模式是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。
使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性

第一种设计模式：

工厂模式:

java 中最常使用的模式之一，创建一个对象无需创建具体的类，通过工厂模式只用将类的实现封装进一个工厂中而无需客户端代码中实例化对象。

实例:

以汽车厂为例:

1.汽车制造：客户只需要从工厂提货而无需关心内部实现。

优点:

1.调用者只需要对象的名称即可创建对象。

2.扩展性高，要增加新产品，只需要扩展一个工厂类。

3.用户只关心接口，按需求取货而无需关心实现。

缺点:

每次增加一个新产品时都要增加一个新的类，系统中类的数量迅速增加，增加了系统复杂度和类的依赖。

主要结构:

1.抽象产品：定义了产品的共同接口或抽象类。它可以是具体产品类的父类或接口，规定了产品对象的共同方法。

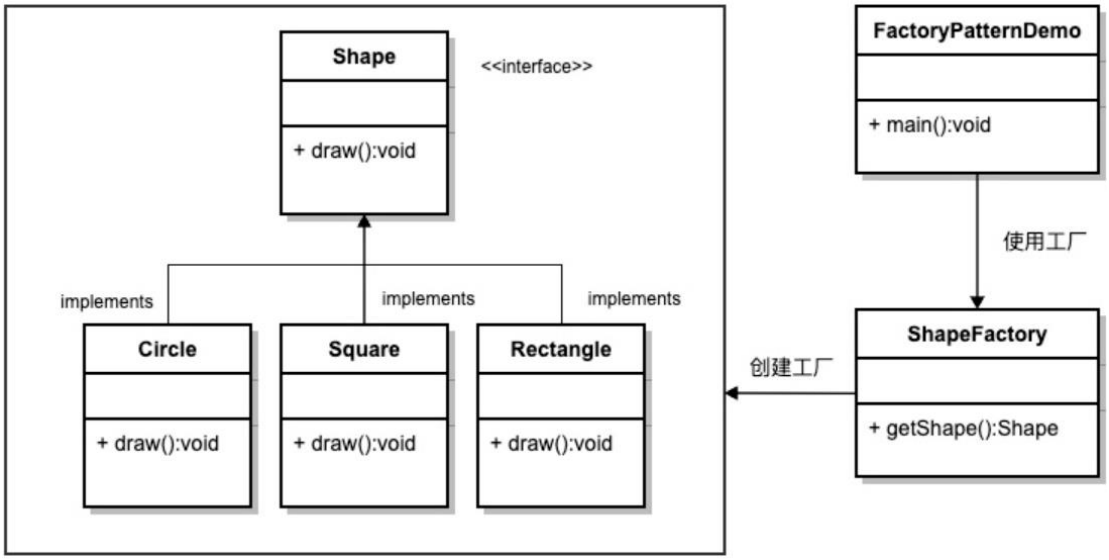
2.具体产品：实现了抽象产品接口，定义了具体产品的特定行为和属性。

3.抽象工厂：声明了创建产品的抽象方法，可以是接口或抽象类。它可以有多个方法用于创建不同类型的产品

4.具体工厂：实现了抽象工厂接口，负责实际创建具体产品的对象。

实现举例:

创建一个 Shape 接口和实现 Shape 接口的实体类。下一步是定义工厂类 ShapeFactory。FactoryPatternDemo 类使用 ShapeFactory 来获取 Shape 对象。它将向 ShapeFactory 传递信息（CIRCLE / RECTANGLE / SQUARE），以便获取它所需对象的类型。



第二种设计模式：

单例模式：

java 中最简单的设计模式之一，在这种模式下，每一个类创建一个对象，同时确保只有单个对象被创建，然后提供一个唯一访问该对象的方法，可以直接访问，无需实例化该类的对象。

实例：

一个班只有一个班主任。

Windows 在多进程多线程环境下操作文件时，避免多个进程或线程同时操作同一个文件，需要通过唯一实例进行处理。

优点：

1.内存中只有一个实例，减少内存开销，尤其是频繁创建和销毁实例时

2.避免资源的多重占用

缺点：

1.没有接口，不能继承

2.与单一职责原则冲突，一个类应该只关心内部逻辑，而不应该关心实例化方式。

主要结构：

1.单例类：包含单例实例的类，通常将构造函数声明为私有。

2.静态成员变量：用于存储单例实例的静态成员变量。

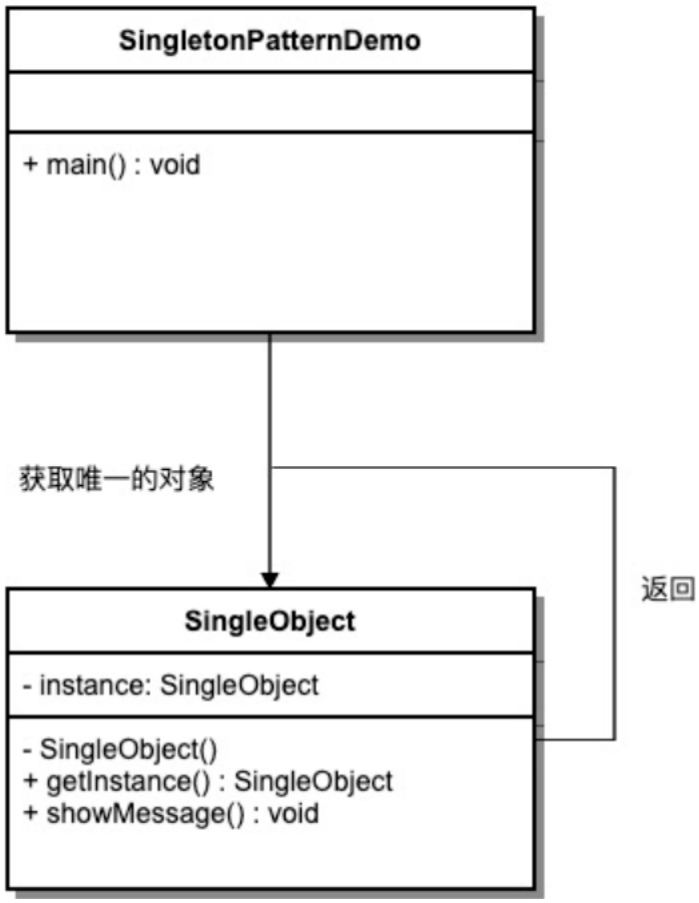
3.获取实例方法：静态方法，用于获取单例实例。

4.私有构造函数：防止外部直接实例化单例类。

5.线程安全处理：确保在多线程环境下单例实例的创建是安全的。

实现举例:

创建一个 SingleObject 类。SingleObject 类有它的私有构造函数和本身的一个静态实例。SingleObject 类提供了一个静态方法，供外界获取它的静态实例。SingletonPatternDemo 类使用 SingleObject 类来获取 SingleObject 对象。



第三种设计模式:

桥接模式:

桥接模式属于结构型模式，它通过提供抽象化和实现化之间的桥接结构，来实现二者的解耦。这种模式涉及到一个作为桥接的接口，使得实体类的功能独立于接口实现类，这两种类型的类可被结构化改变而互不影响。它通过组合的方式，而不是继承的方式，将抽象和实现的部分连接起来。

实例:

转世投胎：灵魂（抽象）和肉体（实现）的分离，允许灵魂选择不同肉体。

优点:

1.抽象与实现分离，提高了系统的灵活性和可维护性

2.扩展能力强，可以独立地扩展抽象和现实

3.用户不需要了解细节

缺点:

理解与设计难度很高，要求开发者在抽象层进行设计与编程。

主要结构:

1.抽象（Abstraction）：定义抽象接口，通常包含对实现接口的引用。

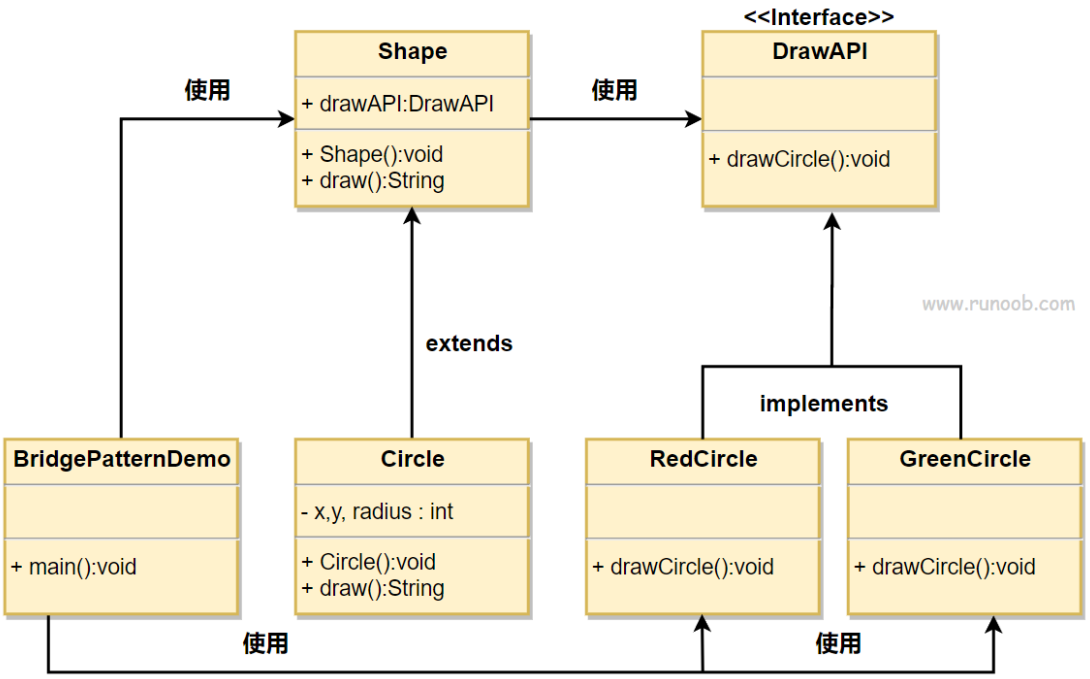
2.扩展抽象（Refined Abstraction）：对抽象的扩展，可以是抽象类的子类或具体实现类。

3.实现（Implementor）：定义实现接口，提供基本操作的接口。

4.具体实现（Concrete Implementor）：实现实现接口的具体类。

实现举例:

我们有一个作为桥接实现的 DrawAPI 接口和实现了 DrawAPI 接口的实体类 RedCircle、GreenCircle。Shape 是一个抽象类，将使用 DrawAPI 的对象。BridgePatternDemo 类使用 Shape 类来画出不同颜色的圆。



第四种设计模式：

装饰器模式：

允许向一个现有的对象添加新的功能，同时又不改变其结构，装饰器模式通过将对象包装在装饰器类中，以便动态地修改其行为。

实例：

画框装饰画：一幅画可以通过添加玻璃和画框来增强其展示效果。

优点：

1.低耦合：装饰类和被装饰类可以独立变化互不影响。

2.灵活：可以动态地撤销和添加功能。

3.替代继承：提供了一种继承之外的扩展对象功能的方式

缺点：

多层装饰可能导致系统复杂性增加。

主要结构：

1.抽象组件（Component）：定义了原始对象和装饰器对象的公共接口或抽象类，可以是具体组件类的父类或接口。

2.具体组件（Concrete Component）：是被装饰的原始对象，它定义了需要添加新功能的对象。

3.抽象装饰器（Decorator）：继承自抽象组件，它包含了一个抽象组件对象，并定义了与抽象组件相同的接口，同时可以通过组合方式持有其他装饰器对象。

4.具体装饰器（Concrete Decorator）：实现了抽象装饰器的接口，负责向抽象组件添加新的功能。具体装饰器通常会在调用原始对象的方法之前或之后执行自己的操作。

实现举例:

创建一个 Shape 接口和实现了 Shape 接口的实体类。然后我们创建一个实现了 Shape 接口的抽象装饰类 ShapeDecorator，并把 Shape 对象作为它的实例变量 RedShapeDecorator 是实现了 ShapeDecorator 的实体类。DecoratorPatternDemo 类使用 RedShapeDecorator 来装饰 Shape 对象。