使用设计模式的目的是为了可重用代码，提高代码的可扩展性和可维护性。

开闭原则：扩展开放，修改关闭。

里氏代换原则：使用父类的地方都能使用子类对象

# 创建型模式

创建型模式关注点是如何创建对象，其核心思想是要把对象的创建和使用相分离。

创建型模式包括：

* 工厂方法：Factory Method
* 抽象工厂：Abstract Factory
* 建造者：Builder
* 原型：Prototype
* 单例：Singleton

## 工厂方法

Integer n = Integer.valueOf(100);

Integer既静态工厂。它提供了静态方法valueOf()来创建Integer。那么这种方式和直接写new Integer(100)有何区别呢？我们观察valueOf()方法：

public final class Integer { public static Integer valueOf(int i) { if (i >= IntegerCache.low && i <= IntegerCache.high) return IntegerCache.cache[i + (-IntegerCache.low)]; return new Integer(i); } ... }

它的好处在于，valueOf()内部可能会使用new创建一个新的Integer实例，但也可能直接返回一个缓存的Integer实例。对于调用方来说，没必要知道Integer创建的细节。

 工厂方法可以隐藏创建产品的细节，且不一定每次都会真正创建产品，完全可以返回缓存的产品，从而提升速度并减少内存消耗。

## 抽象工厂

抽象工厂模式是为了让创建工厂和一组产品与使用相分离，并可以随时切换到另一个工厂以及另一组产品；

抽象工厂模式实现的关键点是定义工厂接口和产品接口，但如何实现工厂与产品本身需要留给具体的子类实现，客户端只和抽象工厂与抽象产品打交道

## 建造者：Builder

生成器模式（Builder）是使用多个“小型”工厂来最终创建出一个完整对象。

## 原型：Prototype

是指创建新对象的时候，根据现有的一个原型来创建。 复制过来

原型模式应用不是很广泛，因为很多实例会持有类似文件、Socket这样的资源，而这些资源是无法复制给另一个对象共享的，只有存储简单类型的“值”对象可以复制。

## 单例：Singleton

单例模式（Singleton）的目的是为了保证在一个进程中，某个类有且仅有一个实例。

public class Singleton { // 静态字段引用唯一实例: private static final Singleton INSTANCE = new Singleton(); // 通过静态方法返回实例: public static Singleton getInstance() { return INSTANCE; } // private构造方法保证外部无法实例化: private Singleton() { } }

那我们什么时候应该用Singleton呢？实际上，很多程序，尤其是Web程序，大部分服务类都应该被视作Singleton，如果全部按Singleton的写法写，会非常麻烦，所以，通常是通过约定让框架（例如Spring）来实例化这些类，保证只有一个实例，调用方自觉通过框架获取实例而不是new操作符：

@Component // 表示一个单例组件 public class MyService { ... }

# 结构型模式

结构型模式主要涉及如何组合各种对象以便获得更好、更灵活的结构。

* 适配器
* 桥接
* 组合
* 装饰器
* 外观
* 享元
* 代理

## 适配器模式（Adapter）

*将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作*

Arrays.asList(T[])就相当于一个转换器，它可以把数组转换为List

类似还有InputStreamReader就是Java标准库提供的Adapter，它负责把一个InputStream适配为Reader

客户端通过自己选择一个品牌，再配合一种引擎，得到最终的Car：

RefinedCar car = **new** BossCar(**new** HybridEngine());

car.drive();

## 桥接

*将抽象部分与它的实现部分分离，使它们都可以独立地变化*

客户端通过自己选择一个品牌，再配合一种引擎，得到最终的Car：

RefinedCar car = **new** BossCar(**new** HybridEngine());

car.drive();

使用桥接模式的好处在于，如果要增加一种引擎，只需要针对Engine派生一个新的子类，如果要增加一个品牌，只需要针对RefinedCar派生一个子类，任何RefinedCar的子类都可以和任何一种Engine自由组合,即一辆汽车的两个维度：品牌和引擎都可以独立地变化

## 组合

*将对象组合成树形结构以表示“部分-整体”的层次结构，使得用户对单个对象和组合对象的使用具有一致性。*

*如*XML或HTML的实现