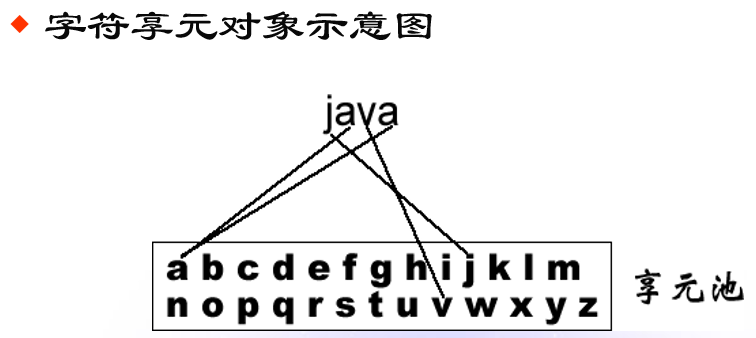
# 享元模式简介

## 概述

1. 当系统中存在大量相同或者相似的对象（这些对象通常是同类对象）时，享元模式是一种值得考虑的解决方案，它通过共享技术实现相同或相似的细粒度对象的复用，从而节约了内存空间、提高了系统性能。在享元模式中提供了一个享元池用于存储已经创建好的享元对象，并通过工厂类将享元对象提供给客户端使用。
2. 如果一个系统软件在运行时所创建的相同或相似对象（这些对象通常是同类对象）数量太多，将导致运行代价过高，带来系统资源浪费、性能下降等问题，而享元模式可以解决这一问题。享元模式通过共享技术实现相同或相似对象的重用，在享元模式中存储这些共享实例对象的地方称为享元池（Flyweight Pool），可以将对象放在享元池中，当需要时再从享元池中取出。
3. 享元模式以共享的方式高效地支持大量细粒度对象的重用，享元对象能做到共享的关键是区分了内部装填（Instrinsic State）和外部状态（Extrinsic State）。内部状态时存储在享元对象内部并且不会随环境影响改变而改变的状态，内部状态可以共享。外部状态是随环境改变而改变的、不可以共享的状态；享元对象的外部状态通常由客户端保存，并在享元对象被创建之后需要使用的时候再传入到享元对象内部，一个外部状态和另一个外部状态之间是相互独立的，相互之间没有影响，客户端可以在使用时将外部状态注入到享元对象中。
4. 享元模式正因为区分了外部状态和内部状态，可以将具有相同内部状态的对象存储在享元池中，享元池中的对象时可以实现共享的，当需要的时候通过注入不同的外部状态以此从享元池中取出不同的对象，而这些对象在内存中实际上只存储一份。
5. 享元模式（Flyweight Pattern）：运用共享技术有效地支持大量细粒度对象的复用。
6. 享元模式要求被共享的对象必须是细粒度对象，它又被称为轻量级模式，享元模式是一种对象结构型模式。



## 结构

享元模式的结构较为复杂，通常结合工厂模式一起使用，在它的结构图中包含了一个享元工厂类，其结构图如下：



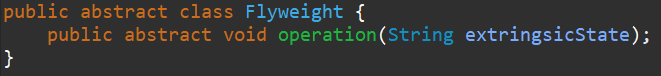
享元模式包含以下4个角色：

1. Flyweight（抽象享元类）：抽象享元类是一个接口或抽象类，在抽象享元类中声明了具体享元类公共的方法，这些方法可以向外界提供享元对象的内部数据（内部状态），同时也可以通过这些方法来设置外部数据（外部状态）。
2. ConcreteFlyweight（具体享元类）：具体享元类实现了抽象享元类，其实例称为享元对象；在具体享元类中为内部状态提供了存储空间，通常可以结合单例模式来设计具体享元类，为每一个具体享元类提供唯一的享元对象。
3. UnsharedConcreteFlyweight（非共享具体享元类）：并不是所有的抽象享元类的子类都需要被共享，不能被共享的子类可以设计为非共享具体享元类；当需要一个非共享具体享元类的对象时可以直接通过实例化创建。
4. FlyweightFactory（享元工厂类）：享元工厂类用于创建并管理享元对象，它针对抽象享元类编程，将各种类型的具体享元对象存储在一个享元池中，享元池一般设计为一个储存“键值对“的集合（也可以是其他类型的集合），可以结合工厂模式进行设计；当用户请求一个具体享元对象时，享元工厂提供一个存储在享元池中已创建的实例或者创建一个新的实例（如果不存在），返回新创建的实例并将其存储在享元池中。

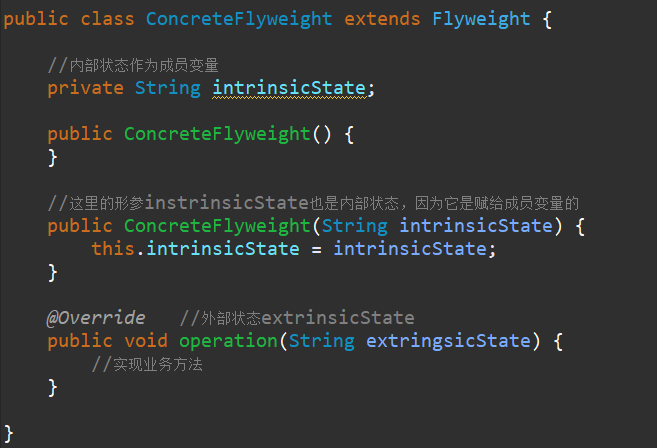
# 实现

## 实现原理

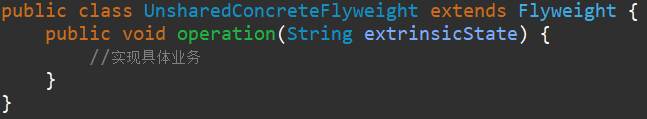
1. 为了提高系统的可扩展性，通常会定义一个抽象享元类作为所有具体享元类的公共父类，典型的抽象享元类代码如下：



1. 在具体享元类中要将内部状态和外部状态分开处理，通常将内部状态作为具体享元类的成员变量（属性），而外部状态通过注入的方式添加到具体享元类中，即将外部状态定义为具体享元类中方法的形参，且这个形参不是赋给内部状态的（成员变量）。典型的具体享元类代码如下：



除了可以共享的具体享元类以外，在使用享元模式时有时还需要处理那些不需要共享的抽象享元类，这些子类可以定义为非共享具体享元类，其典型代码如下：



1. 在享元模式中引入了享元工厂类（运用了简单工厂模式），享元工厂类的作用是提供一个用于存储享元对象的享元池，当用户需要对象时首先从享元池中获取，如果享元池中不存在，则创建一个新的享元对象返回给用户，并将其保存在享元池中。典型的享元工厂类代码如下：



## 实例

实例说明：某软件公司要开发一个围棋软件，该软件公司的开发人员通过对围棋软件进行分析发现，在围棋棋盘中包含大量的黑子和白子，它们的形状、大小一模一样，只是在棋盘上的位置不同而已。如果将每一个棋子作为一个独立的对象存储在内存中，将导致该围棋软件在运行时所需的内存空间较大，那么如何降低运行代价、提高系统性能是需要解决的一个问题。为了解决该问题，现使用享元模式来设计该围棋软件的棋子对象。

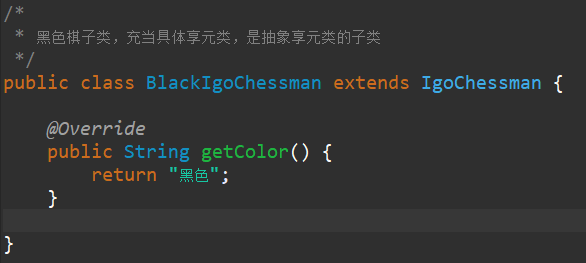
IgoChessman充当抽象享元类，BlackIgoChessman和WhiteIgoChessman充当具体享元类，IgoChessmanFactory充当享元工厂类。结构图如下：



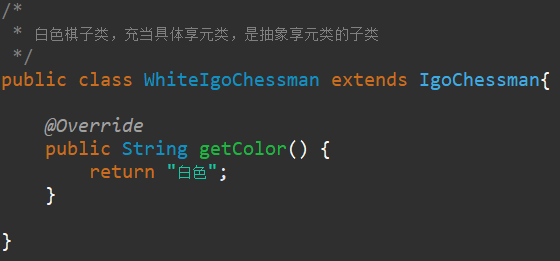
1. IgoChessman：围棋棋子类，充当抽象享元类。



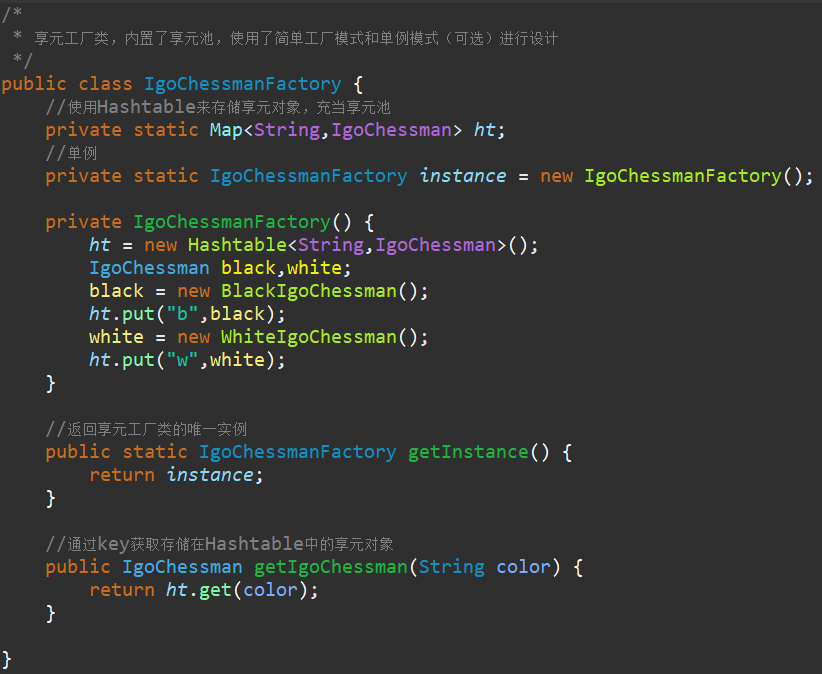
1. BlackIgoChessman：黑色棋子类，充当具体享元类。



1. WhiteIgoChessman：白色棋子类，充当具体享元类。



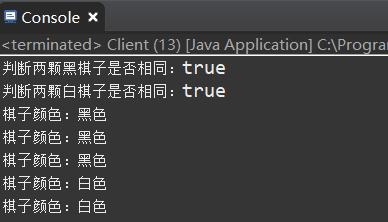
1. IgoChessmanFactory：围棋棋子工厂类（一个简单工厂），充当享元工厂类，使用单例模式（可选）对其进行设计。



1. Client：客户端测试类，测试享元模式。



运行结果：



# 单纯享元模式与复合享元模式

在标准的享元模式结构图中既包含可以共享的具体享元类，也包含不可以共享的非共享具体享元类。但是在实际使用过程中有时候会用到两种特殊的享元模式，即单纯享元模式和复合享元模式。

## 单纯享元模式

在单纯享元模式中所有的具体享元类都是可以共享的，不存在非共享具体享元类。单纯享元模式的结构图如下：



## 复合享元模式

将一些单纯享元对象使用组合模式加以组合还可以形成复合享元对象，这样的复合享元对象本身不能共享，但是它们可以分解成单纯享元对象，而后可以共享。复合享元模式的结构图如下：



通过使用复合享元模式可以让复合享元类CompositeConcreteFlyweight中所包含的每个单纯享元类ConcreteFlyweight都具有相同的外部状态，而这些单纯享元的内部状态往往可以不同。如果希望为多个内部状态不同的享元对象设置相同的外部状态，可以考虑使用复合享元模式。

# 优缺点及适用环境

当系统中存在大量相同或相似的对象时，享元模式是一种较好的解决方案，它通过共享技术实现相同或相似的细粒度对象的复用，从而节约了内存空间，提供了系统性能。相比其他结构型设计模式，享元模式的使用频率并不算太高。

## 优点

享元模式的优点主要如下：

1. 享元模式可以减少内存中对象的数量，使得相同或者相似对象在内存中只保存一份，从而可以节约系统资源，提高系统性能。
2. 享元模式的外部状态相对独立，而且不会影响其内部状态，从而使享元对象可以在不同环境中被共享。

## 缺点

享元模式的缺点主要如下：

1. 享元模式使系统变得复杂，需要分离出内部状态和外部状态，这使得程序的逻辑复杂化。
2. 为了使对象可以共享，享元模式需要将享元对象的部分状态外部化，而读取外部状态将使运行时间变长。
3. 在使用享元模式时需要维护一个存储享元对象的享元池，而这需要耗费一定的系统资源。

## 适用环境

在以下情况下可以考虑适用享元模式：

1. 一个系统有大量相同或者相似的对象，造成内存的大量耗费。
2. 对象的大部分状态都可以外部化，可以将这些外部状态传入对象中。
3. 在使用享元模式时需要维护一个存储享元对象的享元池，而这需要耗费一定的系统资源，因此应当在需要多次重复使用享元对象时才使用享元模式。

## 小结

1. 享元模式运用共享技术有效地支持大量细粒度对象的复用。通过使用享元模式系统只需使用很少量的对象，而这些对象都很相似，状态变化很小，因此可以实现对象的多次复用，享元模式是一种对象结构型模式。
2. 享元模式包含抽象享元类、具体享元类、非共享具体享元类和享元工厂类这4个角色。其中，在抽象享元类中声明了具体享元类公共的方法；具体享元类实现了抽象享元类，为内部状态提供了存储空间；非共享具体享元类是不能被共享的抽象享元类的子类；享元工厂类用于创建并管理享元对象，它针对抽象享元类编程，将各种类型的具体享元对象存储在一个享元池中。
3. 享元模式的优点主要是可以极大地减少内存中对象的数量，使得相同或相似对象在内存中只保存一份，从而可以节约系统资源，提高系统性能。其缺点主要是使系统变得复杂，需要分离出内部状态和外部状态，这使得程序的逻辑复杂化；此外，享元模式需要将享元对象的部分状态外部化，而读取外部状态使得运行时间变长。
4. 享元模式适用于以下环节：一个系统中有大量相同或者相似的对象，造成内存的大量耗费；对象的大部分状态都可以外部化，可以将这些外部状态传入对象中；需要多次重复使用享元对象。
5. 享元模式以共享的方式高效地支持大量细粒度对象的重用，享元对象能做到共享的关键是区分了内部状态和外部状态。内部状态是存储在享元对象内部并且不会随环境改变而改变的状态，通常将其定义为具体享元类的属性，内部状态可以共享；外部状态是随环境改变而改变的、不可共享的状态。
6. 在单纯享元模式中所有具体享元类都是可以共享的，不存在非共享具体享元类。将一些单纯享元对象使用组合模式加以组合还可以形成复合享元模式。