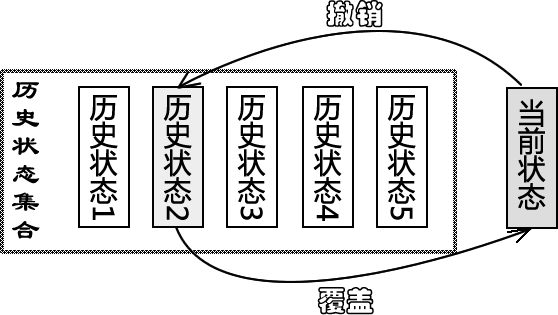
# 备忘录模式简介

备忘录模式是软件系统的“月光宝盒”，它提供了一种对象状态的撤销实现机制，当系统中的某个对象需要恢复到某一历史状态时可以使用备忘录模式进行设计。

## 概述

1. 在软件使用过程中用户难免会出现一些误操作，例如不小心删除了某些文字或文件，为了使软件的使用更加人性化，对于这些误操作需要提供一种类似“后悔药”的机制，让软件系统可以回到误操作前的状态，因此需要保存用户每一个操作时系统的状态，一旦出现误操作，把存储的历史状态取出即可回到之前的状态。现在大多数软件都有撤销（Undo）的功能，快捷键一般是Ctrl + Z，目的就是为了解决这个后悔的问题。备忘录模式正式为了解决此类撤销问题而诞生的，它为软件提供了“后悔药”，通过使用备忘录模式可以让系统恢复到某一特定的历史状态。在实现撤销时首先必须保存软件系统中的历史状态，当用户需要取消误操作并且返回到某个历史状态时可以取出事先保存的历史状态来覆盖当前状态。如下：



1. 备忘录模式提供了一种状态恢复的实现机制，使得用户可以方便地回到一个特定的历史步骤，当新的状态无效或者存在问题时可以使用暂时存储起来的备忘录将状态复原，当前在很多软件所提供的车撤销（Undo）操作中就使用了备忘录模式。
2. 备忘录模式（Memento Pattern）的定义如下：在不破坏封装的前提下捕获一个对象的内部状态，并在该对象之外保存这个状态，这样可以在以后将对象恢复到原先保存的状态。
3. 备忘录模式是一种对象行为型模式，其别名为标记（Token）模式。

## 结构

备忘录模式的核心是备忘录类（Memento）以及用于管理备忘录的负责人类（Caretaker）的设计。其结构如下：



由结构图可知，备忘录模式包含以下3个角色：

1. Originator（原发器）：原发器是一个普通类，它通过创建一个备忘录来存储当内部状态，也可以使用备忘录来恢复其内部状态，一般将系统中需要保存内部状态的类设计为原发器。
2. Memento（备忘录）：备忘录用于存储原发器的内部状态，根据原发器来决定保存哪些内部状态。备忘录的设计一般可以参考原发器的设计，根据实际需要确定备忘录类中的属性。需要注意的是，除了原发器与负责人类之外，备忘录对象不能直接提供其他类使用，原发器的设计在不同的编程语言中实现机制会有所不同。
3. Caretaker（负责人）：负责人又称为管理者，它负责保存备忘录，但是不能对备忘录的内容进行操作或检查。在负责人类中可以存储一个或多个备忘录对象，它只负责存储对象，而不能修改对象，也无须知道对象的实现细节。

# 实现

## 实现原理

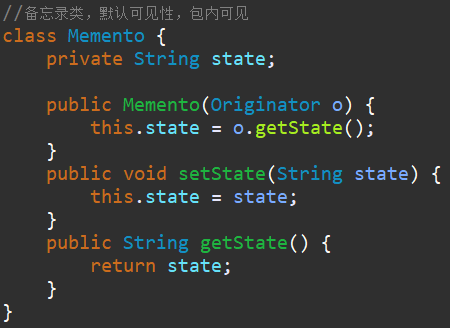
备忘录模式的关键在于如何设计备忘录类和负责人类。由于在备忘录中存储的是原发器的中间状态，因此需要防止原发器以外的其他对象访问备忘录，特别是不允许其他对象来修改备忘录。

1. 在使用备忘录模式时首先应该存在一个原发器类Originator。在真实业务中，原发器类是一个具体的业务类，它包含一些用于存储成员数据的属性。其典型代码如下：



在原发器类中，定义了用于备份当前对象给备忘录和根据指定的备忘录对象恢复当前对象状态的方法。

1. 对于备忘录类Memento而言，它通常提供了与原发器相对应的属性（可以是全部，也可以是部门）用于存储原发器的状态。典型的备忘录类代码如下：

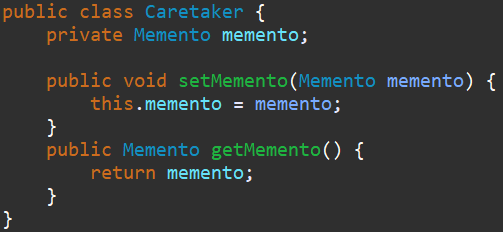


在设计备忘录类时需要考虑其封装性，除了原发器类Originator，不允许其他类来调用备忘录类Memento的构造函数与相关方法。如果不考虑封装性，允许其他类调用setState( )等方法，将导致在备忘录中保存的历史状态发生改变，通过撤销操作所恢复的状态就不再是真实的历史状态，备忘录也就失去了本身的意义。

为了实现对备忘录对象的封装，需要对备忘录的调用进行控制，对于原发器而言，它可以调用备忘录的所有信息，允许原发器访问返回到先前状态所需的所有数据；对于负责人而言，只负责备忘录的保存并将备忘录传递给其他对象；对于其他对象而言，只需要从负责人处取出备忘录对象并将原发器对象的状态恢复，而无须关心备忘录的保存细节。理想的情况是只允许生成该备忘录的那个原发器访问备忘录的内部状态。

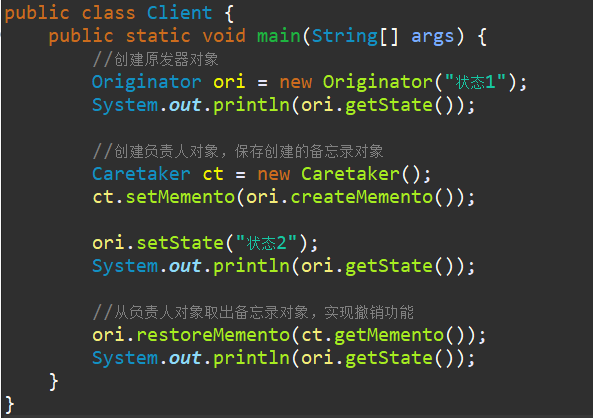
在使用Java语言实现备忘录模式时，一般将Memento类（备忘录）与Originator类（原发器）定义在同一个包（package）中来实现封装，在Java语言中可以使用默认访问标识符来定义Memento类，即保证其包内可见。只有Originator可以对Memento进行访问，而限制了其他类对Memento的访问。在Memento中保存了Originator的state值，如果Originator中的state值改变之后需要撤销，可以通过调用它的restoreMemento( )方法进行恢复。除此之外，还可以将备忘录类作为原发器类的内部类，使得只有原发器才可以访问备忘录中的数据，其他对象都无法使用备忘录中的数据。

1. 对于负责人类Caretaker，它用于保存备忘录对象，并提供getMemento( )方法用于返回一个备忘录对象，原发器通过使用这个备忘录对象可以回到某个历史状态。典型的负责人类的代码如下：

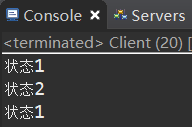


在负责人类（Caretaker）中不应该直接调用Memento中的状态改变方法，它的作用仅仅是存储备忘录对象。将原发器备份生成的备忘录对象存储在其中，当用户需要对原发器进行恢复时再将存储在其中的备忘录对象取出。

1. 在客户端代码中可以通过创建Memento对象来保存原发器的历史状态，在需要的时候再用历史状态来覆盖当前状态，客户端演示代码如下：



运行结果如下：



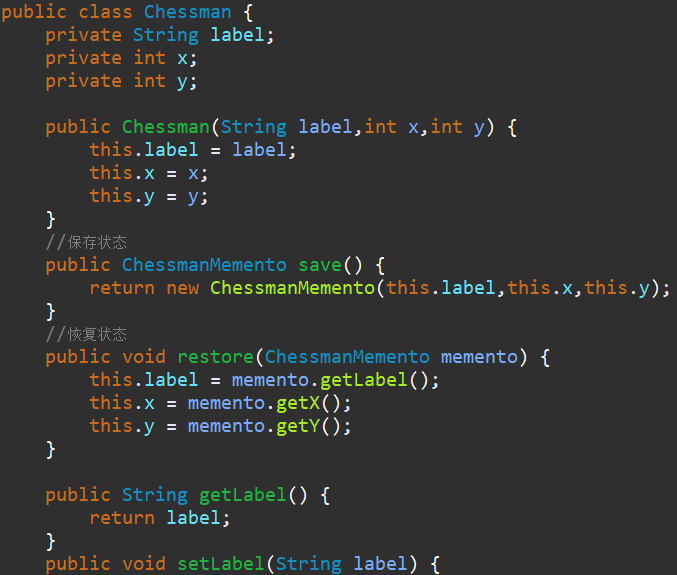
## 实例

实例说明：某软件公司要使用Java语言开发一款可以运行在Android平台的触摸式中国象棋软件，由于考虑到有些用户是菜鸟，经常不小心走错棋；还有些用户因为不习惯使用手指在手机屏幕上拖动棋子，常常出现操作失误，因此该中国象棋提供“悔棋”功能，用户走错棋或操作失误后可恢复到前一个步骤。为了实现悔棋功能，现使用备忘录模式来设计该中国象棋软件。通过分析，其结构图如下图所示：

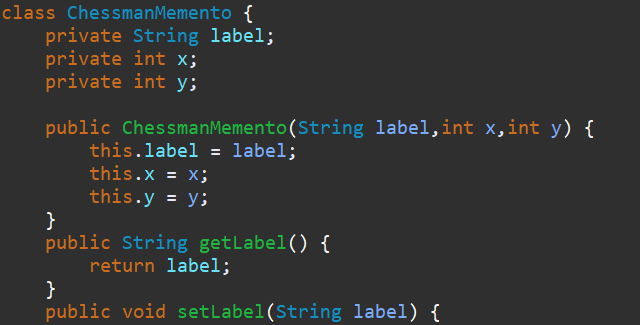


上图中，Chessman充当原发器，ChessmanMemento充当备忘录，MementoCaretaker充当负责人，在MementoCaretaker中定义了一个ChessmanMemento类型的对象，用于存储备忘录。

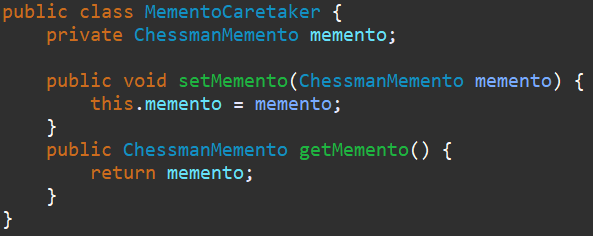
1. Chessman：象棋棋子类，充当原发器



1. ChessmanMemento：象棋棋子备忘录类，充当备忘录



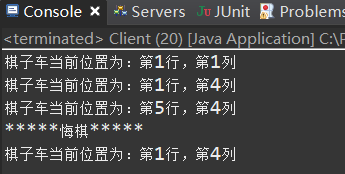
1. MementoCaretaker：象棋棋子备忘录管理类，充当负责人



1. Client：客户端测试类



运行结果如下：



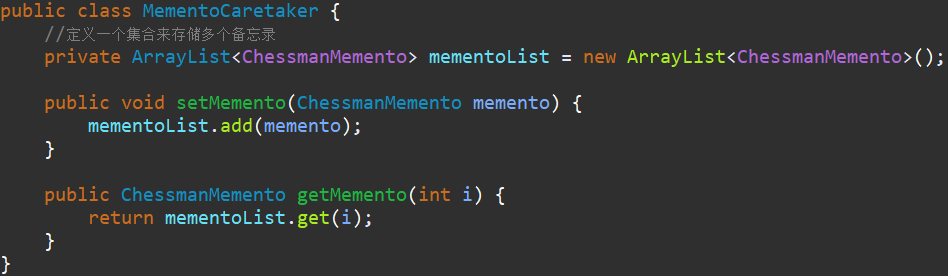
从运行结果可以看出，通过创建备忘录对象可以将象棋棋子的历史状态信息记录下来，在“悔棋”时再取出备忘录中的历史状态信息，用历史状态来覆盖当前状态，从而实现状态的撤销。

## 实现多次撤销

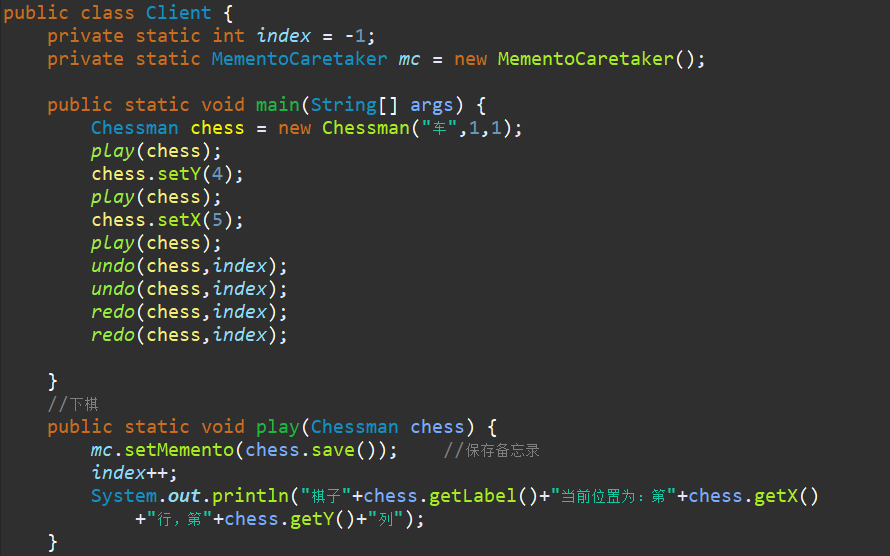
在上中国象棋棋子实例中只能实现一次撤销，即只能“悔棋”一次，因为在负责人类中只定义了一个备忘录对象来保存状态，后面保存的状态会将前一次保存的状态覆盖，但有时候用户需要撤销多步操作。如何实现多步撤销呢？那就是在负责人类中定义一个集合来存储多个备忘录，每个备忘录负责保存一个历史状态，在撤销时可以对备忘录集合进行逆向遍历，回到指定的历史状态，而且还可以对备忘录集合进行正向遍历，实现重做（Redo）或恢复操作，即取消撤销，让对象状态得到恢复。改进之后的中国象棋棋子撤销功能结构图如下所示：

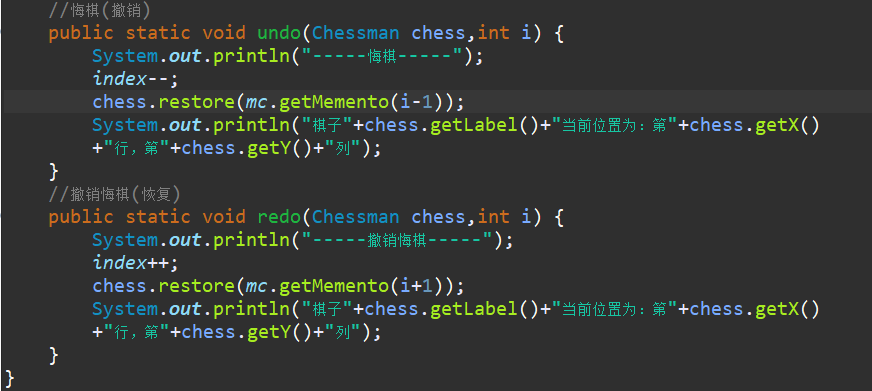


图中，对负责人类MementoCaretaker进行了修改，在其中定义了一个ArrayList类型的集合对象来存储多个备忘录。其代码如下：

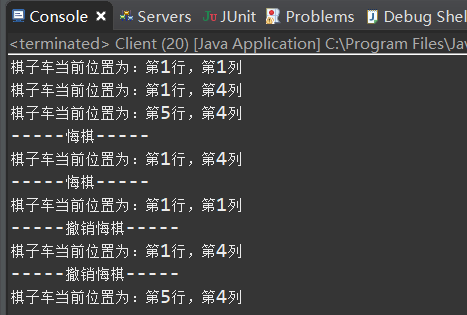


编写以下客户端测试代码：





运行结果如下：



虽然本实现实现了多次撤销（Undo）和恢复（Redo）操作，但并未考虑对象状态在操作过程中出现分支的情况。如果在撤销到某个历史状态之后用户再修改对象状态，此后执行撤销操作时可能会发送对象状态错误。在实际开发中可以使用链表或者堆栈来处理有分支的对象状态改变，读者可以通过链表或者堆栈对本实例进行改进。

# 优缺点及适用环境

备忘录模式在很多软件的使用过程中普遍存在，但是在应用软件开发中它的使用频率并不太高，因为现在很多基于窗体和浏览器的应用软件并没有提供撤销操作。如果需要为软件提供撤销功能，备忘录模式无疑是一种很好的解决方案。在一些字处理软件、图像编辑软件、数据库管理系统等软件中备忘录都得到了很好的应用。

## 优点

备忘录模式的优点主要如下：

1. 提供了一种状态恢复的实现机制，使得用户可以方便地回到一个特定的历史步骤，当新的状态无效或者存在问题时可以使用暂时存储起来的备忘录将状态复原。
2. 备忘录实现了对信息的封装，一个备忘录对象是一种原发器对象状态的表示，不会被其他代码所改动。备忘录保存了原发器的状态，采用列表、堆栈等集合来存储备忘录对象可以实现多次撤销操作。

## 缺点

备忘录模式的缺点主要如下：资源消耗过大，如果需要保存的原发类的成员变量太多，就不可以避免地需要占用大量的存储空间，每保存一次对象的状态都需要消耗一定的系统资源。

## 适用环境

在以下情况下可以考虑使用备忘录模式：

1. 保存一个对象在某一个时刻的全部状态或部分状态，这样以后需要时它能够恢复放到先前的状态，实现撤销操作。
2. 防止外界对象破坏一个对象历史状态的封装性，避免将对象历史状态的实现细节暴露给外界对象。