【保存对象状态】

我们在使用文本编辑器编写文件时，如果不小心删除了某句话，可以通过撤销（undo）功能将文件恢复至之前的状态。有些文本编辑器甚至支持多次撤销，能够恢复至很久之前的版本。使用面向对象编程的方式实现撤销功能时，需要事先保存实例的相关状态信息。然后，在撤销时，还需要根据所保存的信息将实例恢复至原来的状态。

要想恢复实例，需要一个可以自由访问实例内部结构的权限。但是，如果稍不注意，又可能会将依赖于实例内部结构的代码分散地编写在程序中的各个地方，导致程序变得难以维护。这种情况就叫作“破坏了封装性”。

通过引入表示实例状态的角色，可以在保存和恢复实例时有效地防止对象的封装性遭到破坏。

这就是我们在本章中要学习的Memento模式。

使用Memento模式可以实现应用程序的以下功能。

·Undo（撤销）

·Redo（重做）

·History（历史记录）

·Snapshot（快照）

Memento有“纪念品”“遗物”“备忘录”的意思。

当大家从抽屉中拿出让人怀念的照片时，肯定会感慨万千，感觉仿佛又回到了照片中的那个时候。Memento模式就是一个这样的设计模式，它事先将某个时间点的实例的状态保存下来，之后在有必要时，再将实例恢复至当时的状态。

|18.2示例程序

下面我们来看一段使用了Memento模式的示例程序。这是一个收集水果和获取金钱数的掷骰子游戏，游戏规则很简单，具体如下。

·游戏是自动进行的

·游戏的主人公通过掷骰子来决定下一个状态

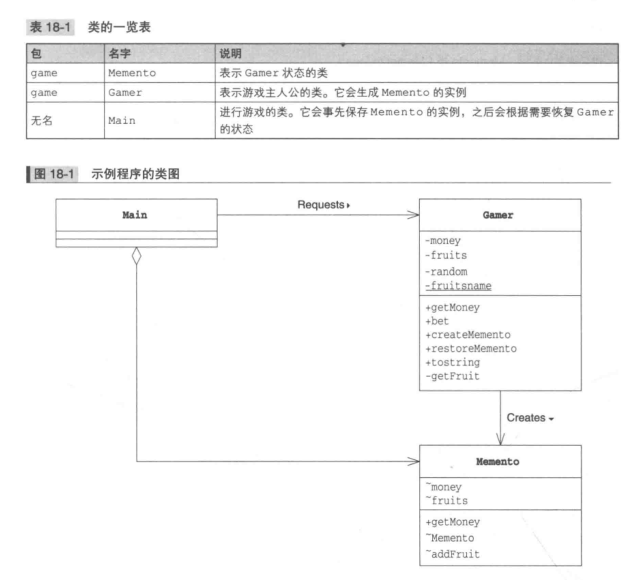
·当骰子点数为1的时候，主人公的金钱会增加·当般子点数为2的时候，主人公的金钱会减少

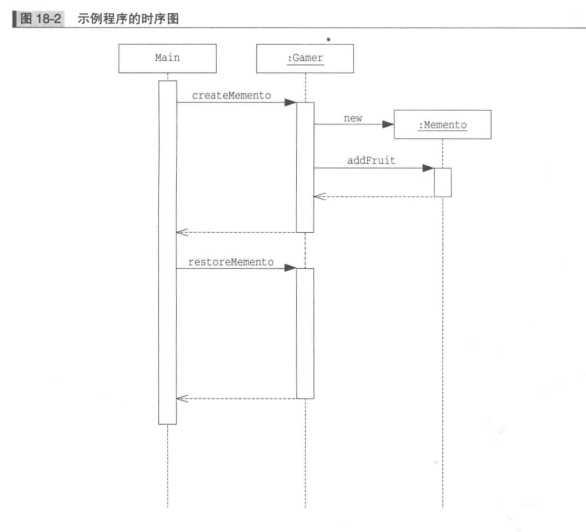
·当般子点数为6的时候，主人公会得到水果

·主人公没有钱时游戏就会结束

在程序中，如果金钱增加，为了方便将来恢复状态，我们会生成Memento类的实例，将现在的状态保存起来。所保存的数据为当前持有的金钱和水果。如果不断掷出了会导致金钱减少的点数，为了防止金钱变为0而结束游戏，我们会使用Memento的实例将游戏恢复至之前的状态。

下面，我们将通过这个小游戏来学习Memento模式。





|18.3 Memento模式中的登场角色

在Memento模式中有以下登场角色。

◆Originator（生成者）Originator角色会在保存自己的最新状态时生成Memento角色。当把以前保存的Memento角色传递给Originator角色时，它会将自己恢复至生成该Memento角色时的状态。在示例程序中，由Gamer类扮演此角色。◆Memento（纪念品）Memento角色会将Originator角色的内部信息整合在一起。在Memento角色中虽然保存了Originator角色的信息，但它不会向外部公开这些信息。

Memento角色有以下两种接口（APl）。

·wide interface——宽接口（APl）Memento角色提供的“宽接口（APl）”是指所有用于获取恢复对象状态信息的方法的集合。由于宽接口（APl）会暴露所有Memento角色的内部信息，因此能够使用宽接口（API）的只有Originator角色。

·narrow interface——窄接口（APl）Memento角色为外部的Caretaker角色提供了“窄接口（API”。可以通过窄接口（APl）获取的Memento角色的内部信息非常有限，因此可以有效地防止信息泄露。

通过对外提供以上两种接口（API），可以有效地防止对象的封装性被破坏。

在示例程序中，由Memento类扮演此角色。

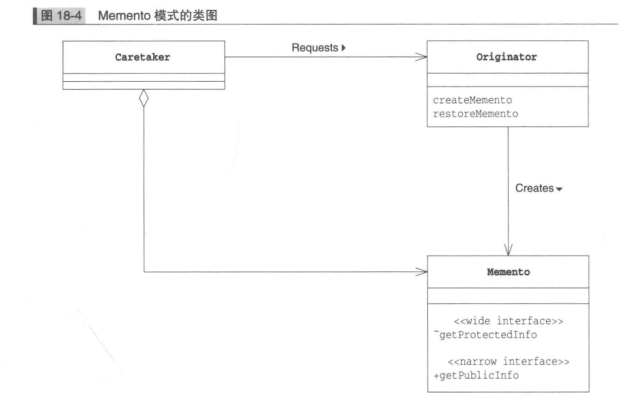
Originator角色和Memento角色之间有着非常紧密的联系。

◆Caretaker（负责人）当Caretaker角色想要保存当前的Originator角色的状态时，会通知Originator角色。Originator角色在接收到通知后会生成Memento角色的实例并将其返回给Caretaker角色。由于以后可能会用Memento 实例来将Originator恢复至原来的状态，因此Caretaker角色会一直保存Memento实例。

在示例程序中，由Main类扮演此角色。

不过，Caretaker角色只能使用Memento角色两种接口（APl）中的窄接口（APl），也就是说它无法访问Memento角色内部的所有信息。它只是将Originator 角色生成的Memento角色当作一个黑盒子保存起来。

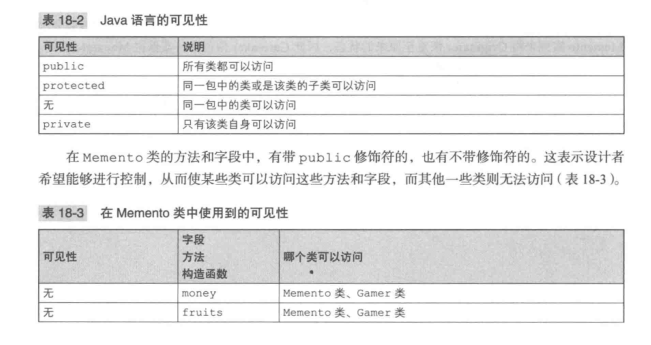
虽然Originator 角色和Memento角色之间是强关联关系，但Caretaker角色和Memento角色之间是弱关联关系。Memento角色对Caretaker角色隐藏了自身的内部信息。



|18.4拓展思路的要点

|两种接口（API）和可见性

为了能够实现Memento模式中的两套接口（AP1），我们利用了Java语言中的可见性。表18-2展示了Java语言提供的4种可见性。





在Memento类中，只有getMoney方法是public的，它是一个窄接口（API），因此该方法也可以被扮演Caretaker角色的Main类调用。

这里做一下补充说明。明明该方法带有修饰符public，但它却一个是窄接口（APl），这不免让人感到有些奇怪。其实，这里所说的“窄”是指外部可以操作的类内部的内容很少。在Memento类的所有方法中，只有getMoney的可见性是public的。也就是说，扮演Caretaker角色的Main类可以获取的只有当前状态下的金钱数目而已。像这种“能够获取的信息非常少”的状态就是本章中“窄”的意思。

由于扮演Caretaker角色的Main类并不在game包下，所以它只能调用public的getMoney方法。因此，Main类无法随意改变Memento类的状态。

还有一点需要注意的是，在Main类中Memento类的构造函数是无法访问的，这就意味着无法像下面这样生成Memento类的实例。

new Memento（100）

如果像这样编写了代码，在编译代码时编译器就会报错。如果Main类中需要用到Memento类的实例，可以通过调用Gamer类的createMemento方法告诉Gamer类“我需要保存现在的状态，请生成一个Memento类的实例给我”。

如果我们在编程时需要实现“允许有些类访问这个方法，其他类则不能访问这个方法”这种需求，可以像上面这样使用可见性来控制访问权限。

|需要多少个Memento在示例程序中，Main类只保存了一个Memento。如果在Main类中使用数组等集合，让它可以保存多个Memento类的实例，就可以实现保存各个时间点的对象的状态。

|Memento的有效期限是多久

在示例程序中，我们是在内存中保存Memento的，这样并没有什么问题。但是正如我们在后面习题18-4中所提到的，如果要将Memento永远保存在文件中，就会出现有效期限的问题了。

这是因为，假设我们在某个时间点将Memento保存在文件中，之后又升级了应用程序版本，那么可能会出现原来保存的Memento与当前的应用程序不匹配的情况。

划分Caretaker角色和Originator角色的意义读到这里，可能有读者会有这样的疑问：如果是要实现撤销功能，直接在Originator角色中实现不就好了吗？为什么要这么麻烦地引入Memento 模式呢？

Caretaker角色的职责是决定何时拍摄快照，何时撤销以及保存Memento角色。

另一方面，Originator角色的职责则是生成Memento角色和使用接收到的Memento角色来恢复自己的状态。

以上就是Caretaker角色与Originator角色的职责分担。有了这样的职责分担，当我们需要对应以下需求变更时，就可以完全不用修改Originator角色。

·变更为可以多次撤销

·变更为不仅可以撤销，还可以将现在的状态保存在文件中

|18.5相关的设计模式

◆Command模式（第22章）在使用Command模式处理命令时，可以使用Memento模式实现撤销功能。

◆Protype模式（第6章）在Memento模式中，为了能够实现快照和撤销功能，保存了对象当前的状态。保存的信息只是在恢复状态时所需要的那部分信息。

而在Protype模式中，会生成一个与当前实例完全相同的另外一个实例。这两个实例的内容完全一样。

◆State模式（第19章）在Memento模式中，是用“实例”表示状态。而在State模式中，则是用“类”表示状态。