第13章

享元（Flyweight）模式

享元模式在客户对象间提供共享对象，并且为共享对象创建职责，以便普通对象不需要考虑共享对象创建的问题。通常情况下，任何时候都只能有一个客户对象引用该共享对象。当某个客户对象改变该共享对象的状态时，该共享对象不需要通知其他客户对象，然而有时候可能需要让多个客户对象同时共享访问某个对象。

存在这么一种需要在多个客户对象间共享对象的场景：当你需要管理成千上万个小对象时，例如在线电子书中的字符对象。在这种场景下，出于性能的考虑，需要在不同的客户对象间安全地传输这些细粒度的对象。例如：某本书需要 A 对象，当有很多 A 对象时，就需要采用某种方法对其建模。

享元模式的意图是通过共享来有效地支持大量细粒度的对象。

不变性

享元模式让多个客户对象间共享访问限定数量的对象。为了实现这个目的，你必须要考虑到当某个对象改变了该共享对象的状态时，该状态变化会影响到每个访问它的对象。当多个客户对象要共享访问某个对象时，若要保证对象间不会相互影响，一种最简单而又常用的做法是，限制客户对象调用任何可能改变共享对象的方法。你可以通过将对象设置为不变（immutable）类型

来达到这一目的。然而，这样做会导致对象在创建后无法改变。Java 中最常见的不可变对象是 String 类。当创建了一个 String 对象后，无论你还是其他客户对象都无法改变该对象的字符。

> **挑战 13.1**
>
> 给出一个理由，表明 Java 语言的设计者为何将 String 对象设置为不可变？如果你认为这不是一种明智的做法，也请给出理由。
>
> 答案参见第 327 页

若存在大量相似对象，而你可能需要共享访问它们，但它们可能并非一成不变。在这种情况下，使用享元模式的第一步是将对象中不可变的部分抽取出来，共享它们。

抽取享元中不可变的部分

对于 Oozinoz 公司，化学药品的使用程度就像文档中的字符一样普遍。工厂中的采购部门、工程部门、制造部门、安全部门都会关心成千上万的化学药品的流动情况。成批的化学药品都被建模到 Substance 类的实例中，如图 13.1 所示。

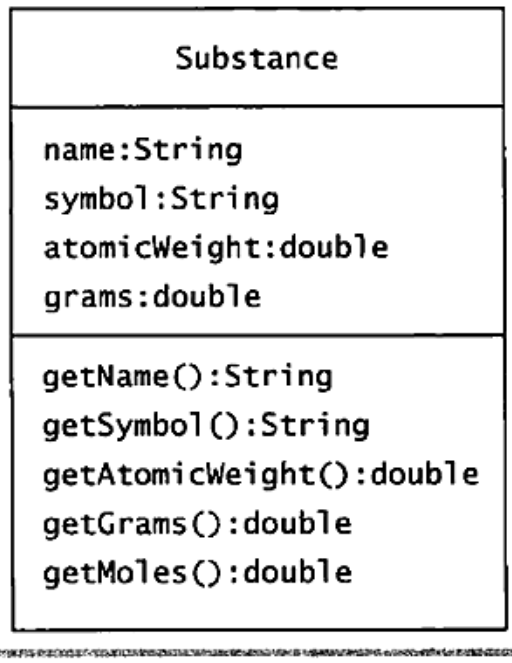


图 13.1　化学药品被建模成 Substance 类的实例

Substance 类有很多方法可以访问它的属性，还有 getMoles()方法用来返回其摩尔数——即化学成分的分子数量。一个 Substance 对象代表特定的摩尔数。Oozinoz 公司使用 Mixture 类来建模化学药品的组成成分。例如，图 13.2 展示了黑火药的对象图。

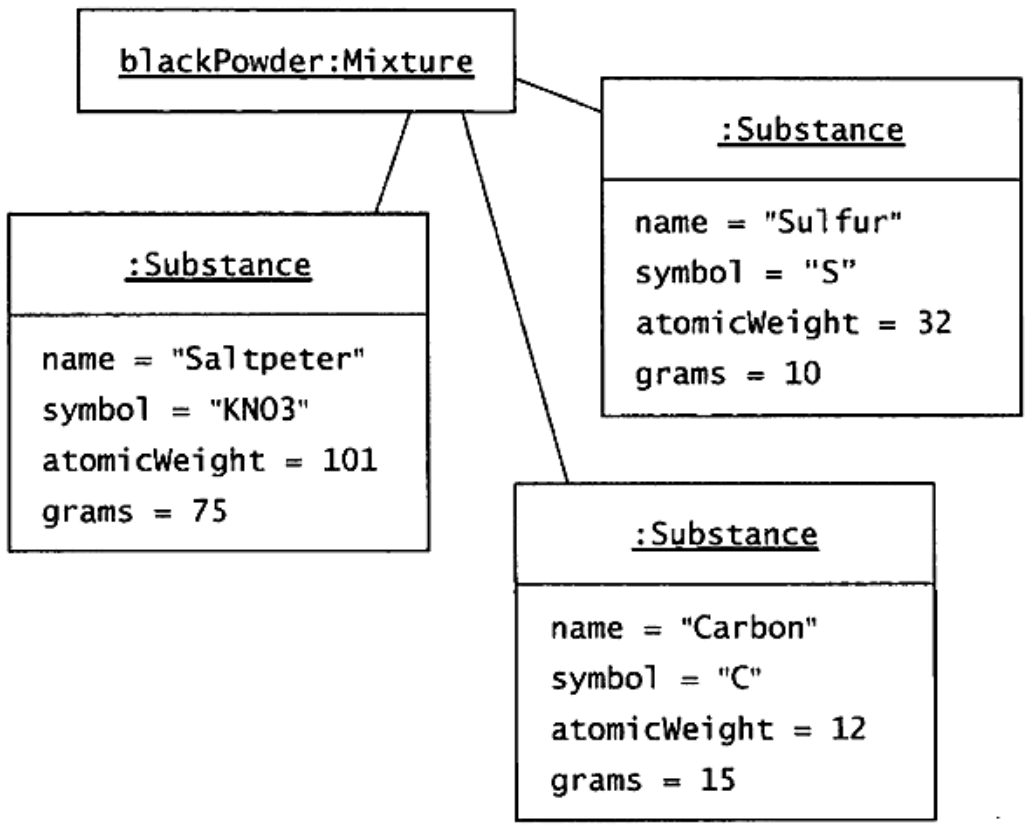


图 13.2　黑火药包含硝酸钾、硫黄以及碳粉

假设 Oozinoz 公司的化学药品越来越多，因而决定使用享元模式来减少程序中 Substance 对象的数量。为了将 Substance 对象变成享元模式，首先需要做的事情就是将类中不变的部分与变化的部分分离出来。假设你要重构 Substance 类，可以将不变的部分抽取到 Chemical 类中。

挑战 13.2

完成图 13.3 中的类图，给出重构后的 Substance2 类，以及一个新的、不可变的 Chemical 类。

答案参见第 327 页

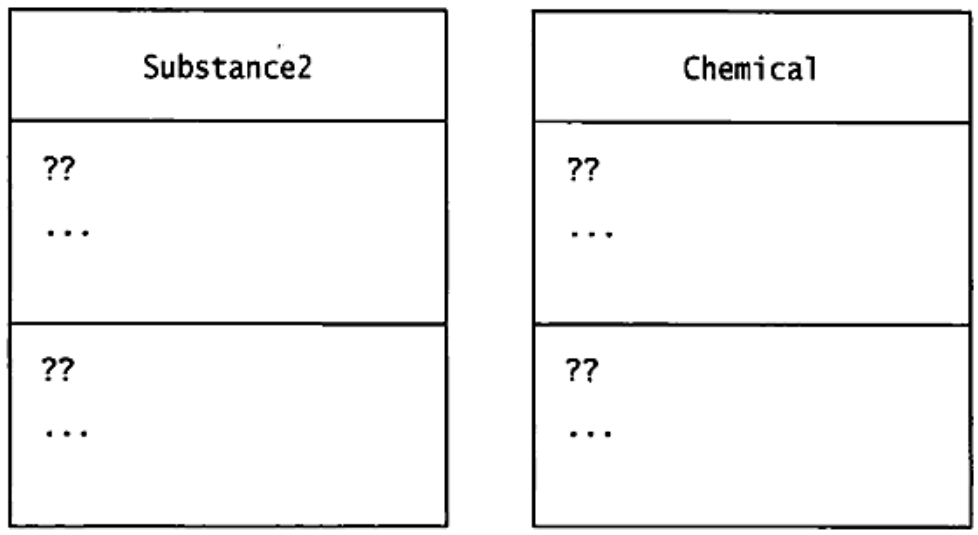


图 13.3　完成该图，将 Substance2 中不可变的部分抽取到 Chemical 类中

共享享元

抽出对象中不变的部分仅仅只完成了享元模式的一半。另一半包括创建享元工厂、实例化享元，以及让客户对象共享享元对象。我们还需要确保客户对象应使用享元工厂创建享元对象，而不是自己创建。

为了创建化学药品的享元对象，需要定义一个 ChemicalFactory 类作为享元工厂。该类包含一个静态方法，负责接收一个名称，返回对应的化学药品。你可以将化学药品存在一个哈希表中，在初始化工厂的时候创建它们。图 13.4 展示了 ChemicalFactory 类的设计。

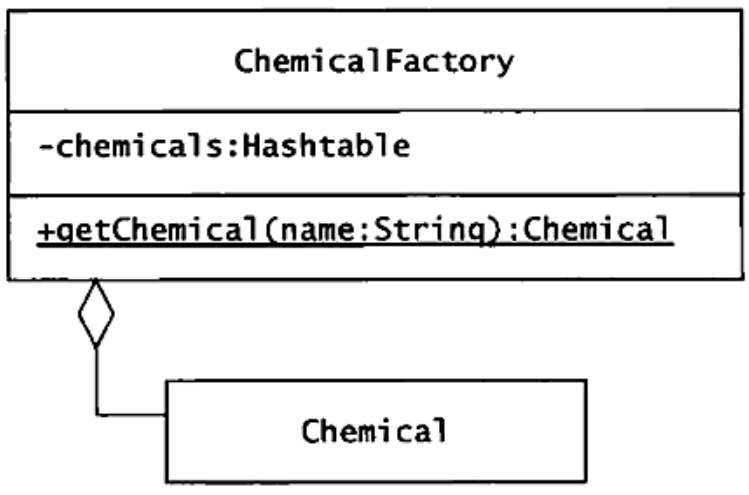


图 13.4　ChemicalFactory 类是一个返回 Chemical 对象的享元工厂

ChemicalFactory 类的代码使用静态构造函数将 Chemical 对象存储在哈希表中。

```
package com.oozinoz.chemical;
import java.util.*;

public class ChemicalFactory {
    private static Map chemicals = new HashMap();

    static {
      chemicals.put(
            "carbon", new Chemical("Carbon", "C", 12));
      chemicals.put(
                "sulfur", new Chemical("Sulfur", "S", 32));
      chemicals.put(
                "saltpeter", new Chemical("Saltpeter", "KNO3", 101));
      //...
    }

    public static Chemical getChemical(String name) {
      return (Chemical) chemicals.get(name.toLowerCase());
    }
}
```

创建完化学药品工厂后，需要执行一些步骤确保让开发人员使用工厂创建享元对象，而不是直接创建 Chemical 对象。一种简单的方式是借助 Chemical 类的可见性。

挑战 13.3

如何使用 Chemical 类的访问修饰符来限制其他开发人员初始化 Chemical 对象？

答案参见第 328 页

访问修饰符还不能完全控制享元对象的实例化，需要确保 ChemicalFactory 是唯一一个能创建 Chemical 实例的类。为了达到这一级别的控制，需要把 Chemical 类定义为 ChemicalFactory 的内部类（参见 com.oozinoz.chemical2 包）。

要访问内嵌的类型，客户对象必须使用如下代码：

```
ChemicalFactory.Chemical c =
    ChemicalFactory.getChemical("saltpeter");
```

可以通过将 Chemical 定义为接口，内嵌类定义为该接口的实现，从而简化对内嵌类型的使用。Chemical 接口可以定义如下三个访问方法：

```
package com.oozinoz.chemical2;
public interface Chemical {
    String getName();
    String getSymbol();
    double getAtomicWeight();
}
```

客户对象永远不会引用该内部类，因此，可以把它定义成私有类型，以确保只有 ChemicalFactory2 类才能访问它。

挑战 13.4

完成如下 ChemicalFactory2.java 的代码：

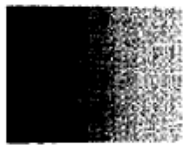
```
package com.oozinoz.chemical2;
import java.util.*;

public class ChemicalFactory2 {
   private static Map chemicals = new HashMap();

/* 挑战！ */ implements Chemical {
    private String name;
    private String symbol;
    private double atomicWeight;

    ChemicalImpl(
            String name,
            String symbol,
            double  atomicWeight) {
        this.name = name;
        this.symbol = symbol;
        this.atomicWeight = atomicWeight;
    }

    public String getName() {
        return name;
    }

    public String getSymbol() {
        return symbol;
```

```
        }

        public double getAtomicWeight() {
            return atomicWeight;
        }

        public String toString() {
            return name + "(" + symbol + ")[" +
                atomicWeight + "]";
        }
    }
    /* 挑战！ */ {
        chemicals.put("carbon",
            factory.new ChemicalImpl("Carbon", "C", 12));
        chemicals.put("sulfur",
            factory.new ChemicalImpl("Sulfur", "S", 32));
        chemicals.put("saltpeter",
            factory.new ChemicalImpl(
                "Saltpeter", "KNO3", 101));
        //...
    }

    public static Chemical getChemical(String name) {
        return /* 挑战！ */
    }
}
```

答案参见第 328 页

小结

享元模式可以使你共享地访问那些大量出现的细粒度对象，例如字符、化学药品以及边界等。享元对象必须是不可变的，可以将那些需要共享访问，并且不变的部分提取出来。为了确保你的享元对象能够被共享，需要提供并强制客户对象使用享元工厂来查找享元对象。访问修饰符对其他开发者进行了一定的限制，但是内部类的使用使限制更进一步，完全限制了该类仅能由其外部容器访问。在确保客户对象正确地使用享元工厂后，你就可以提供对大量细粒度对象的安全共享访问了。



第 3 部分

构造型模式