Thinking in Java chapter8 类重用

emacsun

目录

1 简介

Java的一个重要的特性就是可以方便的实现类重用。重用代码的概念在编程语言里早有体现：把常用的相对独立的功能以函数的形式保存，以备以后调用而不是重新编写。在Java语言中，重用代码的概念通过类重用得到了进一步的延伸。

今天我们学习两种代码重用的技巧： composition 和 inheritance 。我们简单的把 composition 翻译成为组合，把 inheritance 翻译为继承。大多时候我都倾向于保持使用 composition 和 inheritance ，这样一来虽然避免技术词汇翻译带来的不同意，但是却带来了语言的不中不洋。考虑到我们以技术讨论为主，我觉得保持技术文献的不中不洋没有什么不妥。

2 composition

我们之前就接触过 composition 这种代码重用方式。简而言之， composition 就是在一个类中生成另一个类的对象。

我们用一个例子来说明这个问题：

```
1 //: reusing/SprinklerSystem.java
```

```
2  // Composition for code reuse.
3
4  class WaterSource {
5    private String s;
6    WaterSource() {
7      System.out.println("WaterSource()");
8      s = "Constructed";
9    }
10   public String toString() { return s; }
11 }
12
13 public class SprinklerSystem {
14   private String valve1, valve2, valve3, valve4;
15   private WaterSource source = new WaterSource();
16   private int i;
17   private float f;
18   public String toString() {
19     return
20       "valve1 = " + valve1 + " " +
21       "valve2 = " + valve2 + " " +
22       "valve3 = " + valve3 + " " +
23       "valve4 = " + valve4 + "\n" +
24       "i = " + i + " " + "f = " + f + " " +
25       "source = " + source;
26   }
27   public static void main(String[] args) {
28     SprinklerSystem sprinklers = new SprinklerSystem();
29     System.out.println(sprinklers);
30   }
31 } /* Output:
```

在上面的例子中类 SprinklerSystem 具有几个私有变量，除了 privimitive 类型外还有一个 WaterSource 类型的类索引。这个类索引指向另一个类 WaterSource 。这就是 composition 。

在上面的例子中，我们还可以挖掘更多的东西。首先 SprinklerSystem 和 WaterSource 都定义了一个方法 toString 。当编译器需要一个 String 输入，但是给定的输入确实一个 object时，这个方法就会被调用。比如，在上面的例子中， SprinklerSystem 类中有一语句：

```
"source = " + source;
```

编译器发现，你想把一个 WaterSource 对象与一个 String 相加。但是，我们知道只有 String 类型的才能相加。这个时候编译器就会得到一个消息：“调用 toString 把 source 变成 String ”。然后把 toString 的输出和 "source = " 相加。

任何时候，如果你想要实现这种字符串和对象的相加，都需要在类定义中定义方法 toString 。

上面一段代码的输出是：

```
WaterSource()
valve1 = null valve2 = null valve3 = null valve4 = null
i = 0 f = 0.0 source = Constructed
```

我们看到字符串初始化为 `null` ，浮点数和整数初始化为 0.

3 inheritance

`inheritance` 是面向对象语言重要组成部分。实际上在Java中，你无时无刻不在使用继承。即使不是显示的实用，你也在隐式的实用。Java的所有对象都之间或者间接的继承于一个类 `Object` 。在Java中使用 `extends` 来扩展原来的类（我们称之为基类）。

接下来我们通过一个例子来了解 `inheritance` 的语法。

```
1  //: reusing/Detergent.java
2  // Inheritance syntax & properties.
3  package reusing;
4  import static net.mindview.util.Print.*;
5
6  class Cleanser {
7      private String s = "Cleanser␣comes␣first";
8      private String t = "␣Cleanser␣comes␣second";
9      private String u = "␣Cleanser␣comes␣third";
10     public void append(String a) { s += t;s+=a; }
11     public void dilute() { append("␣dilute()"); }
12     public void apply() { append("␣apply()"); }
13     public void scrub() { append("␣scrub()"); }
14     public String toString() { return s; }
15     public static void main(String[] args) {
16         Cleanser x = new Cleanser();
17         x.dilute(); x.apply(); x.scrub();
18         print(x);
19     }
20 }
21
22 public class Detergent extends Cleanser {
23     // Change a method:
24     public void scrub() {
25         append("␣Detergent.scrub()");
26         super.scrub(); // Call base-class version
27     }
28     // Add methods to the interface:
29     public void foam() { append("␣foam()"); }
30     // Test the new class:
31     public static void main(String[] args) {
32         Detergent x = new Detergent();
```

```
33          x.dilute();
34          x.apply();
35          x.scrub();
36          x.foam();
37          print(x);
38          print("Testing base class:");
39          Cleanser.main(args);
40      }
41  } /* Output:
42       Cleanser dilute() apply() Detergent.scrub() scrub() foam()
43       Testing base class:
44       Cleanser dilute() apply() scrub()
45    *///:~
```

在这个例子中 Cleanser 是基类， Detergent 是对基类的扩展。注意关键词 extends 。

上面的例子中有几点需要我们特别注意。首先在 Cleanser append() 中 String 类型通过操作符 += 来实现连接，这对操作符 += 的重载。

第二，在 Cleanser 和 Detergent 中都有 main 函数。这种在每个类中放一个 main 的做法有助于测试每一个类的功能。如果在一个java程序中有很多的类，意味着可能有多个 main 函数。只有命令行调用的那个类能够执行。在上面的代码中，只有 Detergent.main() 会被调用。

第三，在 Detergent.main() 中显示的调用了 Cleanser.main() 。传给 Cleanser.main() 的参数，和传给 Detergent.main() 的参数是一样的。

在 Cleanser 中的所有方法都是 public 的。如果对这些函数不指定 public 或者 private 这样的约束，这些函数的默认访问方式是 package access .也就是说这些函数只允许 package 的成员访问，也就是说在这个 package 中，任何一个类都可以访问这些成员函数。 Detergent 类在访问 Cleanser 类的方法过程中，即使 Cleanser 类中的方法没有指定 public 也不要紧，因为可以通过 package access 访问。但是假设如果其他 package 中的类（继承自 Detergent ）想要访问 Detergent 类中的方法，它只能访问 Detergent 中带有 public 标识的方法。一个通用的规则是，把一个类中的数据私有化（打上 private 标签），把方法公有化(打上 public 标签)。

在 Detergent 类中，重新定义了 scrub 函数（），注意到在基类 Cleanser 中也定义了 scrub 。那么在 Detergent 访问的 scrub 函数是在 Detergent 类中定义的。如果此时我想访问基类中的 scrub 函数怎么办？有个关键词 super .就像在 Detergent 的 scrub 函数中那样 super.scrub() . 显然，通过使用 inheritance ，在继承类中，我们不仅可以使用基类中的方法，也可以重新定义相同名字的方法，并通过 super 来访问基类中定义的方法。