



第 4 章

类、对象和方法

关键技能与概念

- 了解类的基础知识
- 了解如何创建对象
- 理解如何给引用变量赋值
- 创建方法，返回值，并使用形参
- 使用 `return` 关键字
- 从方法返回值
- 向方法添加形参
- 使用构造函数
- 创建带形参的构造函数
- 理解 `new` 运算符
- 理解垃圾回收
- 使用 `this` 关键字

在深入学习 Java 前,需要了解类。类是 Java 的精华,是整个 Java 语言的基础,因为类定义了对对象的本质。类形成了 Java 中面向对象程序设计的基础。类中定义了数据和操作这些数据的代码。代码包含在方法中。因为类、对象和方法是 Java 的基础,所以本章要介绍这些内容。对这些内容有了基本认识,才能编写出较复杂的程序,才能更好地理解下一章要介绍的 Java 的一些关键元素。

4.1 类的基础知识

因为所有 Java 程序活动都发生在类中,所以从本书的开始部分就已经在使用类了。当然,我们用到的只是极简单的类,而类的主要功能还没有用到。类所蕴含的功能比我们先前见到的有限示例要强大许多。

让我们从复习基础知识开始。类是定义对象形式的模板,指定了数据以及操作数据的代码。Java 使用类的规范来构造对象(object),而对象是类的实例(instance)。因此,类实质上是一系列指定如何构建对象的计划。类是逻辑抽象,搞清楚这个问题非常重要。直到类的对象被创建时,内存中才会有类的物理表示。

注意,组成类的方法和变量都称为类的成员(member)。数据成员也称为实例变量。

4.1.1 类的基本形式

当定义类时,要声明类确切的形式和特性。这是通过指定类所包含的实例变量和操作它们的方法来完成的。尽管简单的类可能只包含方法,或只包含实例变量,但大多数实际的类一般都包含这两者。

使用关键字 `class` 创建类。类定义的基本形式如下所示:

```
class classname {
    // declare instance variables
    type var1;
    type var2;
    // ...
    type varN;

    // declare methods
    type method1(parameters) {
        // body of method
    }
    type method2(parameters) {
        // body of method
    }
    // ...
    type methodN(parameters) {
        // body of method
    }
}
```

尽管类的定义没有严格的语法规则,但是设计良好的类应该只定义唯一的逻辑实体。例如,用于存储姓名和电话号码的类一般不用于存储股市信息、平均降雨量、太阳黑斑周期或其他无关信息。这里的要点是:设计良好的类只应该组织逻辑相关的信息。将无关信息放在同一个类中很快就会破坏代码。

直到现在,我们使用的类只用到一个方法: `main()`。但是,你很快就会明白如何创建其他方法。注意,类的基本形式中没有指定 `main()` 方法。只有当一个类是程序的运行起点时,才需要定义 `main()` 方法,而且某些类型的 Java 应用程序不需要 `main()` 方法。

4.1.2 定义类

为了说明类，我们将开发一个封装汽车(如轿车、敞篷货车和卡车)信息的类。该类名为 `Vehicle`，它存储了汽车的以下信息：载客数量、油箱容量和耗油均值(每加仑行驶的英里数)。

第一版的 `Vehicle` 如下所示。它定义了三个实例变量：`passengers`、`fuelcap` 和 `mpg`。注意，`Vehicle` 不包含任何方法。因此，它是一个只包含数据的类(后面将向其中添加方法)。

```
class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon
}
```

`class` 定义了一种新的数据类型。本例中，新的数据类型名为 `Vehicle`。可使用这个名称声明 `Vehicle` 类型的对象。切记，`class` 声明只是类型描述，不创建任何实际对象。因此，前面的代码不会创建任何 `Vehicle` 类型的对象。

创建 `Vehicle` 对象需要使用如下语句：

```
Vehicle minivan = new Vehicle(); // create a Vehicle object called minivan
```

该语句执行后，`minivan` 会成为 `Vehicle` 的一个实例。因此，它就有了真实性。这里先不要考虑语句的细节。

每次创建类的实例时，都是在创建一个对象，其中包含类定义的实例变量副本。因此，每个 `Vehicle` 对象都会包含实例变量 `passengers`、`fuelcap` 和 `mpg` 的副本。要访问这些变量，可使用点(.)运算符。点运算符将对象名和成员名链接在一起。点运算符的基本形式如下所示：

```
object.member
```

对象在点运算符左侧指定，成员则放在点运算符右侧。例如，将数值 16 赋给 `minivan` 的 `fuelcap` 变量，需要使用下面的语句：

```
minivan.fuelcap = 16;
```

总之，可使用点运算符来访问实例变量和方法。

下面是使用 `Vehicle` 类的完整程序：

```
// A program that uses the Vehicle class.

class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon
}

// This class declares an object of type Vehicle.
class VehicleDemo {
    public static void main(String args[]) {
        Vehicle minivan = new Vehicle();
        int range;

        // assign values to fields in minivan
        minivan.passengers = 7;
        minivan.fuelcap = 16; ← 注意使用点运算符来访问成员
        minivan.mpg = 21;
```

```

    // compute the range assuming a full tank of gas
    range = minivan.fuelcap * minivan.mpg;
    System.out.println("Minivan can carry " + minivan.passengers +
        " with a range of " + range);
}
}

```

为测试这个程序，可将 `Vehicle` 和 `VehicleDemo` 类放在同一个源文件中。例如，包含本程序的文件可以命名为 `VehicleDemo.java`，因为 `main()` 方法在名为 `VehicleDemo` 的类中，而不是在名为 `Vehicle` 的类中。这两个类都可以放在文件的开头。在使用 `javac` 编译该程序时，会创建两个扩展名为 `.class` 的文件，一个用于 `Vehicle`，另一个则用于 `VehicleDemo`。Java 编译器将自动把每个类放到各自的 `.class` 文件中。没必要将 `Vehicle` 和 `VehicleDemo` 类放入同一个源文件。可将这两个类分别放在名为 `Vehicle.java` 和 `VehicleDemo.java` 的文件中。若这么做，仍可通过编译 `VehicleDemo.java` 来编译程序。

要运行该程序，必须执行 `VehicleDemo.class`，输出如下所示：

```
Minivan can carry 7 with a range of 336
```

在继续学习前，先复习如下基本原理：因为每个对象都有自己的由类定义的实例变量副本，所以一个对象的变量内容与另一个对象的变量内容是不同的。两个对象除了都是同一类型的对象这一事实外，它们之间没什么联系。例如，如果有两个 `Vehicle` 对象，每个对象都可以有自己的 `passengers`、`fuelcap` 和 `mpg`，而且对象间变量的内容也可以不一样。以下程序就说明了这一点(注意带有 `main()` 的类现在名为 `TwoVehicles`)。

```

// This program creates two Vehicle objects.

class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon
}

// This class declares an object of type Vehicle.
class TwoVehicles {
    public static void main(String args[]) {
        Vehicle minivan = new Vehicle();
        Vehicle sportscar = new Vehicle();
        // 注意，minivan 和 sportscar
        // 代表独立对象

        int range1, range2;

        // assign values to fields in minivan
        minivan.passengers = 7;
        minivan.fuelcap = 16;
        minivan.mpg = 21;

        // assign values to fields in sportscar
        sportscar.passengers = 2;
        sportscar.fuelcap = 14;
        sportscar.mpg = 12;

        // compute the ranges assuming a full tank of gas
        range1 = minivan.fuelcap * minivan.mpg;
        range2 = sportscar.fuelcap * sportscar.mpg;

        System.out.println("Minivan can carry " + minivan.passengers +

```

```
        " with a range of " + range1);

    System.out.println("Sportscar can carry " + sportscar.passengers +
        " with a range of " + range2);
}
}
```

程序的输出如下所示：

```
Minivan can carry 7 with a range of 336
Sportscar can carry 2 with a range of 168
```

可以看出，minivan 中的数据与包含在 sportscar 中的数据是完全分开的。图 4-1 对此进行了说明。

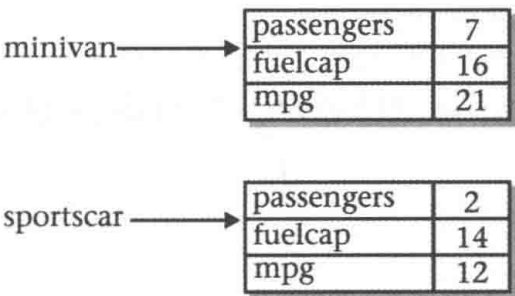


图 4-1 数据是完全分开的

4.2 如何创建对象

下面这一行代码在前面的程序中用于声明一个 `Vehicle` 类型的对象：

```
Vehicle minivan = new Vehicle();
```

该声明完成两个功能。首先，它声明一个名为 `minivan` 的 `Vehicle` 类型的变量。该变量没有定义对象，它只是一个可以引用对象的变量。其次，它创建了一个对象的实际副本，并把对象的引用赋给 `minivan`。这些都是由 `new` 运算符完成的。

`new` 运算符为对象动态分配内存(即在运行时分配内存)，并返回一个对它的引用。该引用是由 `new` 分配的对象在内存中的地址。然后，把引用存储在变量中。因此，在 `Java` 中，必须动态分配所有类对象。

前面语句中包含的两个步骤可分开重写，如下所示：

```
Vehicle minivan; // declare reference to object
minivan = new Vehicle(); // allocate a Vehicle object
```

第一行把 `minivan` 声明为对 `Vehicle` 类型对象的引用。因此，`minivan` 是一个可引用对象的变量，但它本身不是对象。此时，`minivan` 不引用对象。第二行创建了一个新的 `Vehicle` 对象，并把对它的引用赋给 `minivan`。现在，`minivan` 就与一个对象链接在一起了。

4.3 引用变量和赋值

在赋值运算中，对象引用变量与基本类型变量(如 `int` 变量)的工作方式不同。将一个基本类型的变量赋给另一个变量时，情况很简单。等号左边的变量接收等号右边变量值的副本。当把一个对象引用变量赋给其他对象引用变量时，情况就有些复杂了，因为是在改变引用变量所引用的对象。这一不同可能导致出乎意料的结果。例如，考虑下面的代码段：

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
```

乍一看，很容易想到 `car1` 和 `car2` 引用的是不同对象，但情况并非如此。相反，`car1` 和 `car2` 引用的是同一个对象。把 `car1` 赋给 `car2` 使 `car2` 也指向 `car1` 指向的对象。因此，既可通过 `car1` 也可通过 `car2` 来操作对象。例如，下面的赋值语句：

```
car1.mpg = 26;
```

执行之后，下面两条 `println()` 语句：

```
System.out.println(car1.mpg);
System.out.println(car2.mpg);
```

都显示相同的值——26。

尽管 `car1` 和 `car2` 都引用相同的对象，但它们之间没有关系。例如，下面对 `car2` 的赋值只是改变了 `car2` 所引用的对象：

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
Vehicle car3 = new Vehicle();
```

```
car2 = car3; // now car2 and car3 refer to the same object.
```

执行这一系列代码后，`car2` 与 `car3` 指向同一个对象。由 `car1` 引用的对象没有变化。

4.4 方法

如前所述，实例变量和方法组成了类。到目前为止，`Vehicle` 类只包含数据，却没有方法。尽管只有数据的类也是完全有效的，但多数类具有方法。方法是子例程，它操作类定义的数据，多数情况下会提供对数据的访问。多数情况下，程序的其他部分都是通过类的方法与类进行交互的。

一个方法包含一条或多条语句。在编写完善的 Java 代码中，每个方法只执行一项任务。每个方法只有一个名称，而方法也是通过使用这个名称被调用的。总之，可以对方法任意命名。但是，要切记 `main()` 是为程序开始执行准备的方法，而且也不能使用 Java 关键字作为方法名。

当用文本表示方法时，本书使用编写 Java 的常用惯例，即在方法名后面使用圆括号。例如，如果方法名为 `getval`，那么在句子中使用它的名称时，就会写为 `getval()`。这种表示方式有助于将书中的方法名与变量名区分开来。

方法的基本形式如下所示：

```
ret-type name( parameter-list ) {
    // body of method
}
```

其中，**ret-type** 指定方法返回的数据类型。返回类型可以是任何有效类型，包括创建的类类型。如果方法不返回值，返回类型必须为 `void`。方法名由 `name` 指定。该名称可以是任何合法的，并且是当前作用域中没有被其他方法使用的标识符。`parameter-list` 是一系列用逗号隔开的类型与标识符对。形参(`parameter`)本质上是变量，它在调用方法时接收传递到方法的实参(`argument`)的值。如果方法没有形参，形参列表就为空。


向 Vehicle 类添加方法

如前所述，类的方法通常操作或访问类的数据。知道这一点后，回顾一下，前面示例的 `main()` 通过将耗油率和油箱容量相乘来计算汽车的行驶里程。尽管技术上这是正确的，但这并非处理这种计算的最佳办法。计算汽车的行驶里程最好由 `Vehicle` 类自行处理。这一结论的原因是易于理解的：汽车的行驶里程与油箱容量以及燃油率有关，这两个数据都被封装在 `Vehicle` 类中。通过向 `Vehicle` 类添加一个计算行驶里程的方法，可增强它的面向对象结构。要向 `Vehicle` 添加一个方法，在 `Vehicle` 的声明内指定它即可。例如，下面的 `Vehicle` 版本就包含一个名为 `range()` 的方法，它显示汽车行驶的里程。

```
// Add range to Vehicle.
```

```
class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon

    // Display the range.
    void range() {
        System.out.println("Range is " + fuelcap * mpg);
    }
}
```



注意 `fuelcap` 和 `mpg` 被直接使用，没有利用点运算符

```
class AddMeth {
    public static void main(String args[]) {
        Vehicle minivan = new Vehicle();
        Vehicle sportscar = new Vehicle();

        int rangel, range2;

        // assign values to fields in minivan
        minivan.passengers = 7;
        minivan.fuelcap = 16;
        minivan.mpg = 21;

        // assign values to fields in sportscar
        sportscar.passengers = 2;
        sportscar.fuelcap = 14;
        sportscar.mpg = 12;

        System.out.print("Minivan can carry " + minivan.passengers +
            ". ");

        minivan.range(); // display range of minivan

        System.out.print("Sportscar can carry " + sportscar.passengers +
            ". ");

        sportscar.range(); // display range of sportscar.
    }
}
```

程序的输出如下所示：


```
Minivan can carry 7. Range is 336
Sportscar can carry 2. Range is 168
```

我们从 `range()` 方法开始查看该程序的关键元素。`range()` 的第一行代码是：

```
void range() {
```

这一行声明了一个名为 `range` 的无形参方法。它返回的类型是 `void`。因此，`range()` 不向调用者返回值。这一行以方法体的左花括号结尾。

`range()` 的主体由下面这行代码组成：

```
System.out.println("Range is " + fuelcap * mpg);
```

该语句显示了 `fuelcap` 与 `mpg` 相乘得到的汽车行驶里程。因为在调用 `range()` 时，`Vehicle` 类型的每个对象都有自己的 `fuelcap` 和 `mpg` 的副本，所以行驶里程的计算使用了调用对象的这些变量的副本。

`range()` 方法在遇到右花括号时结束。这会导致程序的控制权返回给调用者。

下面详细介绍 `main()` 中的这行代码：

```
minivan.range();
```

该语句调用 `minivan` 的 `range()` 方法。即它在对象后面使用点运算符，调用与 `minivan` 对象相关的 `range()`。当调用方法时，程序控制权被转移到方法。当方法结束时，控制权转移回调用者，从调用后的一行代码继续执行。

本例中，调用 `minivan.range()` 用于显示 `minivan` 定义的汽车行驶里程。与之相似，调用 `sportscar.range()` 用于显示 `sportscar` 定义的汽车行驶里程。每次调用 `range()` 时，都显示特定对象的行驶里程。

在 `range()` 方法中有一个很重要的细节值得注意：没有在实例变量 `fuelcap` 和 `mpg` 的前面添加对象名或点运算符就直接引用了它们。当一个方法使用由它的类定义的实例变量时，可以直接使用，无须显式引用对象，也无须使用点运算符。思考一下，就会发现这一点是很容易理解的。方法总被与其相关的类的对象调用。一旦调用发生，对象就是已知的。因此，在方法中，没必要第二次指定对象。这就意味着 `range()` 中的 `fuelcap` 和 `mpg` 隐式引用了调用 `range()` 的对象中的变量副本。

4.5 从方法返回值

从整体上讲，引起方法返回的条件有两个：第一个是当遇到方法的右花括号时，如前面示例中所示的 `range()` 方法；第二个是当执行 `return` 语句时。`return` 语句有两种形式：一种用在 `void` 方法(不返回值的方法)中，另一种则用于返回值的方法中。这里讨论第一种形式，4.6 节再介绍如何返回值。

在 `void` 方法中，可使用下面形式的 `return` 语句使方法立即结束：

```
return;
```

此语句执行时，程序控制权跳过方法中其余的代码返回给调用者。例如，考虑下面这个方法：

```
void myMeth() {
    int i;

    for(i=0; i<10; i++) {
        if(i == 5) return; // stop at 5
        System.out.println();
    }
}
```

这里，`for` 循环只在 `i` 为 0~5 时运行，因为一旦 `i` 等于 5，方法就会返回。一个方法可拥有多个 `return` 语句，

特别是当有两个或多个执行路径时，例如：

```
void myMeth() {
    // ...
    if(done) return;
    // ...
    if(error) return;
    // ...
}
```

这里，如果方法执行完毕或有错误发生，方法就返回。然而，一个方法中如果有太多出口点，就会破坏代码的结构，所以要小心使用 `return` 语句。设计良好的方法要有定义良好的出口点。

复习一下：`void` 方法的返回方式有两种——到达右花括号或执行 `return` 语句。

4.6 返回值

尽管带有 `void` 返回类型的方法并不少见，但大多数方法都会返回值。事实上，返回值是方法最有用的功能之一。前面介绍了一个返回值的示例：使用 `sqrt()` 函数来获得一个数的平方根。

在程序设计中，返回值可以用于不同目的。某些情况下，如 `sqrt()`，返回值包含一些计算的结果。在另一些情况下，返回值只用来指示成功或失败。还有一些情况，可能包含状态码。无论是何种目的，使用方法返回值都是 Java 程序设计的重要组成部分。

方法使用下面的 `return` 语句向调用例程返回值：

```
return value;
```

这里，`value` 是返回的值。这种形式的 `return` 语句只能用在返回类型不为 `void` 的方法中。而且，非 `void` 方法必须使用这种形式的 `return` 语句返回值。

可使用返回值来改进 `range()` 的实现方式。让 `range()` 计算并返回行驶里程比显示行驶里程更好。这种方法的优势是可将返回值用于其他计算。下例就不再让 `range()` 显示行驶里程，而是返回行驶里程：

```
// Use a return value.

class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon

    // Return the range.
    int range() {
        return mpg * fuelcap; ← 返回给定车辆的行驶里程
    }
}

class RetMeth {
    public static void main(String args[]) {
        Vehicle minivan = new Vehicle();
        Vehicle sportscar = new Vehicle();

        int range1, range2;

        // assign values to fields in minivan
        minivan.passengers = 7;
```

```

minivan.fuelcap = 16;
minivan.mpg = 21;

// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;

// get the ranges
range1 = minivan.range();
range2 = sportscar.range();

```

将返回值赋给变量

```

System.out.println("Minivan can carry " + minivan.passengers +
    " with range of " + range1 + " Miles");

```

```

System.out.println("Sportscar can carry " + sportscar.passengers +
    " with range of " + range2 + " miles");

```

程序的输出如下所示:

```

Minivan can carry 7 with range of 336 Miles
Sportscar can carry 2 with range of 168 miles

```

在程序中, 注意当 `range()` 被调用时, 是将它放在赋值语句的右边, 而左边是一个接收 `range()` 返回值的变量。因此, 在执行下列代码行后, `minivan` 对象的行驶里程就存储到 `range1` 中:

```
range1 = minivan.range();
```

注意, 现在 `range()` 有一个 `int` 类型的返回值。这就意味着要向调用者返回一个整数值。方法的返回类型之所以重要, 是因为方法返回的数据类型必须与方法指定的返回类型兼容。因此, 如果想让一个方法返回 `double` 类型的数据, 那么该方法的返回类型也必须是 `double` 类型。

尽管前面的程序是正确的, 但效率不尽如人意。具体来说, `range1` 或 `range2` 变量是没用的。对 `range()` 的调用可直接在语句 `println()` 中进行, 如下所示:

```

System.out.println("Minivan can carry " + minivan.passengers +
    " with range of " + minivan.range() + " Miles");

```

本例中, 执行 `println()` 时会自动调用 `minivan.range()`, 而它的值会传递给 `println()`。不仅如此, 还可在需要 `Vehicle` 对象的行驶里程时调用 `range()`。例如, 下面的语句就比较了两辆汽车的行驶里程:

```
if(v1.range() > v2.range()) System.out.println("v1 has greater range");
```


4.7 使用形参


调用方法时, 可向方法传递一个或多个值。如前所述, 向方法传递的值称为实参(argument)。在方法中接收实参的变量称为形参(parameter)。形参在方法名后的圆括号中声明。形参的声明语法与用于变量的语法是一样的。形参位于自己方法的作用域中, 执行接收实参的特殊任务, 工作方式与局部变量十分相似。

下面是一个使用形参的简单示例。在 `ChkNum` 类中, 如果传递给 `isEven()` 的值是偶数, 就返回 `true`, 否则返

回 false。因此, isEven() 有 boolean 返回类型。

```
// A simple example that uses a parameter.

class ChkNum {
    // return true if x is even
    boolean isEven(int x) {  这里, x 是 isEven() 的整数形参
        if((x%2) == 0) return true;
        else return false;
    }
}

class ParmDemo {
    public static void main(String args[]) {
        ChkNum e = new ChkNum();
         将实参传递给 isEven()
        if(e.isEven(10)) System.out.println("10 is even.");

        if(e.isEven(9)) System.out.println("9 is even.");


        if(e.isEven(8)) System.out.println("8 is even.");
    }
}
```


程序的输出如下所示:

```
10 is even.
8 is even.
```

在程序中, isEven() 被调用了三次, 每一次都有一个不同的值传递给它。下面详细介绍这一过程。首先, 注意如何调用 isEven()。实参在圆括号之间指定。当第一次调用 isEven() 时, 传递给它的值是 10。因此, 当 isEven() 开始执行时, 形参 x 接收值 10。在第二次调用中, 9 是实参, 所以 x 得到值 9。在第三次调用中, 实参为 8, x 接收该值。在调用 isEven() 时, 作为实参传递的值正是形参 x 接收的值。

一个方法可以有多个形参。声明形参时, 只需要用逗号将形参分隔开即可。例如, Factor 类定义了一个名为 isFactor() 的方法, 该方法用于确定第一个形参是不是第二个形参的因数。

```
class Factor {
    boolean isFactor(int a, int b) {  该方法有两个形参
        if( (b % a) == 0) return true;
        else return false;
    }
}

class IsFact {
    public static void main(String args[]) {
        Factor x = new Factor();
         向 isFactor() 传递两个实参
        if(x.isFactor(2, 20)) System.out.println("2 is factor");
        if(x.isFactor(3, 20)) System.out.println("this won't be displayed");
    }
}
```

注意, 在调用 isFactor() 时, 实参也是用逗号分隔的。

在使用多个形参时, 每个形参都指定了自己的类型, 这些类型可以相互不同。例如, 下面一行代码就是完全

有效的:

```
int myMeth(int a, double b, float c) {
    // ...
}
```

向 Vehicle 类添加带形参的方法

可使用带形参的方法向 `Vehicle` 类添加新功能: 计算给定距离所需的耗油总量。新方法名为 `fuelneeded()`。该方法获取要行驶的英里数, 返回所需汽油的加仑数。`fuelneeded()` 方法的定义如下:

```
double fuelneeded(int miles) {
    return (double) miles / mpg;
}
```

注意, 该方法返回一个 `double` 类型的值。这是非常有用的, 因为行驶给定里程所需的耗油总量可能不是一个整数。包含 `fuelneeded()` 的整个 `Vehicle` 类如下:

```
/*
   Add a parameterized method that computes the
   fuel required for a given distance.
*/

class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon

    // Return the range.
    int range() {
        return mpg * fuelcap;
    }

    // Compute fuel needed for a given distance.
    double fuelneeded(int miles) {
        return (double) miles / mpg;
    }
}

class CompFuel {
    public static void main(String args[]) {
        Vehicle minivan = new Vehicle();
        Vehicle sportscar = new Vehicle();
        double gallons;
        int dist = 252;

        // assign values to fields in minivan
        minivan.passengers = 7;
        minivan.fuelcap = 16;
        minivan.mpg = 21;

        // assign values to fields in sportscar
        sportscar.passengers = 2;
        sportscar.fuelcap = 14;
        sportscar.mpg = 12;

        gallons = minivan.fuelneeded(dist);
    }
}
```

```

System.out.println("To go " + dist + " miles minivan needs " +
    gallons + " gallons of fuel.");

gallons = sportscar.fuelneeded(dist);

System.out.println("To go " + dist + " miles sportscar needs " +
    gallons + " gallons of fuel.");
}
}

```

程序的输出如下所示:

```

To go 252 miles minivan needs 12.0 gallons of fuel.
To go 252 miles sportscar needs 21.0 gallons of fuel.

```

练习 4-1(HelpClassDemo.java)

创建 Help 类

如果想要用一句话总结类的本质,那就是:类封装了功能。当然,有时关键是要弄清楚在什么地方一个功能结束,另一个功能开始。作为一条基本原则,类通常作为构建大型应用程序的代码块。为此,每个类必须表示一个执行描述清晰的动作的功能单元。因此,类应尽可能小,但不能太小。也就是说,包含与类无关的功能会混淆、破坏代码,但类包含的功能太少又显得支离破碎。平衡点在哪里?这就是程序设计科学成为一门艺术的地方。幸运的是,多数程序员通过经验都可以轻松地找到这个平衡点。

为获得这种经验,把上一章中的练习 3-3 改写为 Help 类。这样做的益处有以下几点。首先,帮助系统定义了一个逻辑单元,只用来显示 Java 控制语句的语法,因此功能简洁,定义良好。其次,把帮助放在类中从美学上讲也是一种令人愉悦的方式。无论何时想给用户提供帮助系统,只需要实例化帮助系统对象即可。最后,因为封装了帮助,所以更新或修改帮助系统都不会对使用它的程序产生意想不到的副作用。

步骤:

(1) 创建一个名为 HelpClassDemo.java 的新文件。如果想省去输入的麻烦,那么可以把练习 3-3 中的文件 Help3.java 复制到 HelpClassDemo.java 中。

(2) 为将帮助系统转换为类,首先必须清楚组成帮助系统的各个部分。例如,在 Help3.java 中有用于显示菜单、输出用户选择、检查响应有效性以及信息显示的各种代码。程序循环执行,直到输入字母 q 才停止。如果考虑一下,就会发现显示菜单、检查响应有效性以及信息显示是构成帮助系统不可缺少的部分。然而,如何获得用户输入,以及重复请求是否应该被处理则不是必需的。因此,创建的类要显示帮助信息和帮助菜单以及检查选择是否有效。这些方法分别称为 helpOn()、showMenu() 和 isValid()。

(3) 创建 helpOn() 方法,如下所示:

```

void helpOn(int what) {
    switch(what) {
        case '1':
            System.out.println("The if:\n");
            System.out.println("if(condition) statement;");
            System.out.println("else statement;");
            break;
        case '2':
            System.out.println("The switch:\n");
            System.out.println("switch(expression) {");

```

```

        System.out.println(" case constant:");
        System.out.println(" statement sequence");
        System.out.println(" break;");
        System.out.println(" // ...");
        System.out.println("}");
        break;
    case '3':
        System.out.println("The for:\n");
        System.out.print("for(init; condition; iteration)");
        System.out.println(" statement;");
        break;
    case '4':
        System.out.println("The while:\n");
        System.out.println("while(condition) statement;");
        break;
    case '5':
        System.out.println("The do-while:\n");
        System.out.println("do {");
        System.out.println(" statement;");
        System.out.println("} while (condition);");
        break;
    case '6':
        System.out.println("The break:\n");
        System.out.println("break; or break label;");
        break;
    case '7':
        System.out.println("The continue:\n");
        System.out.println("continue; or continue label;");
        break;
    }
    System.out.println();
}

```

(4) 创建 showMenu() 方法:

```

void showMenu() {
    System.out.println("Help on:");
    System.out.println(" 1. if");
    System.out.println(" 2. switch");
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while");
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
}

```

(5) 创建 isValid() 方法, 如下所示:

```

boolean isValid(int ch) {
    if(ch < '1' | ch > '7' & ch != 'q') return false;
    else return true;
}

```

(6) 把前面创建的方法加入 Help 类中, 如下所示:

```

class Help {

```

```

void helpOn(int what) {
    switch(what) {
        case '1':
            System.out.println("The if:\n");
            System.out.println("if(condition) statement;");
            System.out.println("else statement;");
            break;
        case '2':
            System.out.println("The switch:\n");
            System.out.println("switch(expression) {");
            System.out.println("    case constant:");
            System.out.println("        statement sequence");
            System.out.println("    break;");
            System.out.println("    // ...");
            System.out.println("}");
            break;
        case '3':
            System.out.println("The for:\n");
            System.out.println("for(init; condition; iteration)");
            System.out.println("    statement;");
            break;
        case '4':
            System.out.println("The while:\n");
            System.out.println("while(condition) statement;");
            break;
        case '5':
            System.out.println("The do-while:\n");
            System.out.println("do {");
            System.out.println("    statement;");
            System.out.println("} while (condition);");
            break;
        case '6':
            System.out.println("The break:\n");
            System.out.println("break; or break label;");
            break;
        case '7':
            System.out.println("The continue:\n");
            System.out.println("continue; or continue label;");
            break;
    }
    System.out.println();
}

void showMenu() {
    System.out.println("Help on:");
    System.out.println(" 1. if");
    System.out.println(" 2. switch");
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while");
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
}

boolean isValid(int ch) {

```



```

        if(ch < '1' | ch > '7' & ch != 'q') return false;
        else return true;
    }

}

```

(7) 最后重写练习 3-3 中的 `main()` 方法, 以使其使用新的 `Help` 类。将这个类命名为 `HelpClassDemo.java`。`HelpClassDemo.java` 的完整清单如下所示:

```

/*
    Try This 4-1

    Convert the help system from Try This 3-3 into
    a Help class.
*/

class Help {
    void helpOn(int what) {
        switch(what) {
            case '1':
                System.out.println("The if:\n");
                System.out.println("if(condition) statement;");
                System.out.println("else statement;");
                break;
            case '2':
                System.out.println("The switch:\n");
                System.out.println("switch(expression) {");
                System.out.println("    case constant:");
                System.out.println("        statement sequence");
                System.out.println("    break;");
                System.out.println("    // ...");
                System.out.println("}");
                break;
            case '3':
                System.out.println("The for:\n");
                System.out.println("for(init; condition; iteration)");
                System.out.println("    statement;");
                break;
            case '4':
                System.out.println("The while:\n");
                System.out.println("while(condition) statement;");
                break;
            case '5':
                System.out.println("The do-while:\n");
                System.out.println("do {");
                System.out.println("    statement;");
                System.out.println("} while (condition);");
                break;
            case '6':
                System.out.println("The break:\n");
                System.out.println("break; or break label;");
                break;
            case '7':
                System.out.println("The continue:\n");
                System.out.println("continue; or continue label;");
                break;
        }
    }
}

```

```

    }
    System.out.println();
}

void showMenu() {
    System.out.println("Help on:");
    System.out.println(" 1. if");
    System.out.println(" 2. switch");
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while");
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
}

boolean isValid(int ch) {
    if(ch < '1' | ch > '7' & ch != 'q') return false;
    else return true;
}

}

class HelpClassDemo {
    public static void main(String args[])
        throws java.io.IOException {
        char choice, ignore;
        Help hlpobj = new Help();

        for(;;) {
            do {
                hlpobj.showMenu();

                choice = (char) System.in.read();

                do {
                    ignore = (char) System.in.read();
                } while(ignore != '\n');

            } while( !hlpobj.isValid(choice) );

            if(choice == 'q') break;

            System.out.println("\n");

            hlpobj.helpOn(choice);
        }
    }
}

```

当测试该程序时，会发现它的功能与原来一样。这种方法的优势就是现在你有了可在需要时随意重用的帮助系统。

4.8 构造函数

在前面的示例中，每个 `Vehicle` 对象的实例变量都需要使用一组语句来手动设置，例如：

```
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
```

这种方法不会在专业编写的 Java 代码中出现。除了容易出错(可能会忘记设置某个域)，还有一种更简单、更好的方法来完成这项任务——使用构造函数。

构造函数(constructor)在创建对象时初始化对象。它与类同名，在语法上与方法相似。然而，构造函数没有显式的返回类型。通常，构造函数用来初始化类定义的实例变量，或执行其他创建完整对象所需的启动过程。

无论是否定义，所有的类都有构造函数，因为 Java 自动提供了一个默认的构造函数，将所有成员变量初始化为它们的默认值，即 `0`、`null` 和 `false`，分别用于数值类型、引用类型和布尔类型。当然，一旦定义自己的构造函数，就不会再使用默认构造函数了。

下面是使用构造函数的一个简单示例：

```
// A simple constructor.

class MyClass {
    int x;

    MyClass() { ← 这是 MyClass 的构造函数
        x = 10;
    }
}

class ConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass();

        System.out.println(t1.x + " " + t2.x);
    }
}
```

本例中，`MyClass` 的构造函数如下：

```
MyClass() {
    x = 10;
}
```

该构造函数把数值 10 赋给 `MyClass` 的实例变量 `x`。该构造函数在创建对象时由 `new` 调用。例如，在下面这行代码中：

```
MyClass t1 = new MyClass();
```

`t1` 对象调用了构造函数 `MyClass()`，并把 10 赋给了 `t1.x`。对于 `t2` 也是这样。在构造完成后，`t2.x` 的值为 10。因此，该程序的输出为：

```
10 10
```

4.9 带形参的构造函数

在前面的示例中，使用了无形参的构造函数。尽管对于某些情况这已经足够用了，但大多数情况下，还需要一个可以接收一个或多个形参的构造函数。向构造函数添加形参的方式与向方法添加形参的方式一样：只需要在构造函数名称后的圆括号内声明形参即可。例如，下面的 `MyClass` 就有一个带形参的构造函数：

```
// A parameterized constructor.

class MyClass {
    int x;

    MyClass(int i) { ← 该构造函数有一个形参
        x = i;
    }
}

class ParmConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass(10);
        MyClass t2 = new MyClass(88);

        System.out.println(t1.x + " " + t2.x);
    }
}
```

程序的输出如下所示：

```
10 88
```

在该程序中，构造函数 `MyClass()` 定义了一个名为 `i` 的形参，它用于初始化实例变量 `x`。因此，当执行以下代码时：

```
MyClass t1 = new MyClass(10);
```

10 就传递给了 `i`，然后由 `i` 赋给 `x`。

向 Vehicle 类添加构造函数

可通过添加一个在创建对象时自动初始化 `passengers`、`fuelcap` 和 `mpg` 域的构造函数来改善 `Vehicle` 类。特别要注意如何创建 `Vehicle` 对象。

```
// Add a constructor.

class Vehicle {
    int passengers; // number of passengers
    int fuelcap;    // fuel capacity in gallons
    int mpg;        // fuel consumption in miles per gallon

    // This is a constructor for Vehicle.
    Vehicle(int p, int f, int m) { ← Vehicle 类的构造函数
        passengers = p;
        fuelcap = f;
        mpg = m;
    }
}
```

```

// Return the range.
int range() {
    return mpg * fuelcap;
}

// Compute fuel needed for a given distance.
double fuelneeded(int miles) {
    return (double) miles / mpg;
}
}

class VehConsDemo {
    public static void main(String args[]) {

        // construct complete vehicles
        Vehicle minivan = new Vehicle(7, 16, 21);
        Vehicle sportscar = new Vehicle(2, 14, 12);
        double gallons;
        int dist = 252;

        gallons = minivan.fuelneeded(dist);

        System.out.println("To go " + dist + " miles minivan needs " +
            gallons + " gallons of fuel.");

        gallons = sportscar.fuelneeded(dist);

        System.out.println("To go " + dist + " miles sportscar needs " +
            gallons + " gallons of fuel.");

    }
}

```

minivan 和 sportscar 都是在创建时由构造函数 Vehicle() 初始化的。按照构造函数中形参指定的那样初始化每个对象。例如，下面这行代码：

```
Vehicle minivan = new Vehicle(7, 16, 21);
```

在使用 new 创建对象时，值 7、16 和 21 被传递给 Vehicle() 构造函数。因此，minivan 的 passengers、fuelcap 和 mpg 的副本会分别存储值 7、16 和 21。该程序的输出与前一个版本一致。

4.10 深入介绍 new 运算符

前面详细讨论了类及其构造函数，现在可以详细介绍一下 new 运算符了。new 运算符的基本形式如下：

```
class-var = new class-name(arg-list);
```

这里，class-var 是要创建的类类型的变量。class-name 是被初始化的类的类名。圆括号包含的实参列表(可以为空)前面的类名指定了类的构造函数。如果类不定义自己的构造函数，那么 new 将使用 Java 默认的构造函数。因此，new 可用于创建任何类类型的对象。new 运算符返回对新创建的对象引用，在本例中将其赋给 class-var。

内存是有限的，由于内存不足，new 可能无法为对象分配内存。如果出现这种情况，就会发生运行时异常(第 9 章将详细介绍异常)。对于本书的示例程序，不必担心内存不足的情况，但是在实际的程序中需要考虑这种可能性。

4.11 垃圾回收

使用 `new` 运算符可以把空闲内存空间分配给对象。如前所述,内存不是无限的,而空闲内存也是可能耗尽的。因此, `new` 可能会因为没有足够的空闲空间来创建对象而失败。动态分配内存方案的关键就是回收无用对象占用的内存,以使内存用于后面的分配。在许多程序设计语言中,释放已经分配的内存是手动处理的,而 Java 使用一种不同的、更方便的方法——垃圾回收(garbage collection)。

Java 的垃圾回收系统会自动回收对象,透明地在后台操作,不需要程序员干预。具体工作方式为:当不存在对某对象的任何引用时,该对象就被认定为没有存在的必要了,它所占用的内存就要被释放。被回收的内存可用于以后的分配。

专家解答

问: 为什么对于基本数据类型(如 `int` 或 `float`)的变量,不需要使用 `new`?

答: Java 的基本数据类型不是作为对象来实现的,出于效率的原因,它们是作为“普通”变量来实现的。基本类型的变量包含赋给它的值。如前所述,对象变量引用了对象。这种间接层次(和其他对象特性)增加了对象的开销,而基本数据类型则没有这种开销。

垃圾回收只在程序执行的过程中偶尔发生,并不是只要有一个或多个不再使用的对象,就会发生垃圾回收。为提高效率,垃圾回收器通常只在满足两个条件时才运行:有对象要回收和需要回收这些对象。切记垃圾回收要占用时间,因此 Java 运行时系统只在需要时才使用此机制。因此,我们无法知道垃圾回收发生的准确时间。

4.12 this 关键字

在结束本章前,有必要介绍一下 `this` 关键字。当调用一个方法时,会向它自动传递一个隐式实参,它是对调用对象(即调用方法的对象)的一个引用。这个引用就叫作 `this`。为理解 `this`,首先考虑下面的程序,它创建了一个名为 `Pwr` 的类,该类计算数值的不同幂的结果。

```
class Pwr {
    double b;
    int e;
    double val;

    Pwr(double base, int exp) {
        b = base;
        e = exp;

        val = 1;
        if(exp==0) return;
        for( ; exp>0; exp--) val = val * base;
    }

    double get_pwr() {
        return val;
    }
}

class DemoPwr {
    public static void main(String args[]) {
```

```

Pwr x = new Pwr(4.0, 2);
Pwr y = new Pwr(2.5, 1);
Pwr z = new Pwr(5.7, 0);

System.out.println(x.b + " raised to the " + x.e +
    " power is " + x.get_pwr());
System.out.println(y.b + " raised to the " + y.e +
    " power is " + y.get_pwr());
System.out.println(z.b + " raised to the " + z.e +
    " power is " + z.get_pwr());
}
}

```

在一个方法中，不需要对象或类的限定就可以直接访问类中的其他方法。因此，在 `get_pwr()` 中，语句

```
return val;
```

意味着要返回与调用对象相关的 `val` 的副本。然而，同一条语句也可以这样写：

```
return this.val;
```

这里，`this` 引用了调用 `get_pwr()` 的对象。因此，`this.val` 引用了该对象的 `val` 副本。例如，如果已经调用了 `x` 的 `get_pwr()`，那么前面语句中的 `this` 就会引用 `x`。不使用 `this` 来编写语句仅仅是为了方便。

下面是使用 `this` 引用的完整 `Pwr` 类：

```

class Pwr {
    double b;
    int e;
    double val;

    Pwr(double base, int exp) {
        this.b = base;
        this.e = exp;

        this.val = 1;
        if(exp==0) return;
        for( ; exp>0; exp--) this.val = this.val * base;
    }

    double get_pwr() {
        return this.val;
    }
}

```

事实上，没有 Java 程序员会编写如上所示的 `Pwr` 类，因为这样做不会带来任何好处，而使用标准形式会更简单。然而，`this` 有一些重要用途。例如，Java 语法允许形参名或局部变量名与实例变量名一致。当发生这种情况时，局部变量名会隐藏实例变量。通过使用 `this` 引用它，可访问隐藏的实例变量。例如，在语法上，下面的 `Pwr()` 构造函数是有效的：

```

Pwr(double b, int e) {
    this.b = b;
    this.e = e;
}

val = 1;
if(e==0) return;

```

引用实例变量 `b` 而不是形参


```
    for( ; e>0; e--) val = val * b;  
}
```

在这个版本中，形参名与实例变量名一致，因此隐藏了它们。然而，`this`可用于找到实例变量。

4.13 自测题

1. 类与对象的不同之处是什么？
2. 如何定义类？
3. 每个对象都有自己的什么副本？
4. 使用两条单独的语句，写出如何声明 `MyCounter` 类的对象 `counter`。
5. 写出如何声明一个名为 `myMeth()` 的方法，该方法的返回类型是 `double`，有两个 `int` 类型形参 `a` 和 `b`。
6. 如果方法返回一个值，那么必须如何返回？
7. 构造函数的名称是什么？
8. `new` 的作用是什么？
9. 什么是垃圾回收，它是如何工作的？
10. `this` 的作用是什么？
11. 构造函数可以有一个或多个形参吗？
12. 如果方法不返回值，那么返回类型必须是什么？