7.11 Cohesion

We introduced the idea of cohesion in Section 7.3: a unit of code should always be responsible for one and only one task. We shall now investigate the cohesion principle in more depth and analyze some examples.

The principle of cohesion can be applied to classes and methods: classes should display a high degree of cohesion, and so should methods.

7.11.1 Cohesion of methods

Concept:

Method cohesion. A cohesive method is responsible for one and only one well-defined task.

When we talk about cohesion of methods, we seek to express the ideal that any one method should be responsible for one and only one well-defined task.

We can see an example of a cohesive method in the `Game` class. This class has a private method named `printWelcome` to show the opening text, and this method is called when the game starts in the `play` method (Code 7.8).

Code 7.8
Two methods with a good degree of cohesion

```
/**
 * Main play routine. Loops until end of play.
 */
public void play()
{
    printWelcome();

    // Enter the main command loop. Here we repeatedly read
    // commands and execute them until the game is over.

    boolean finished = false;
    while (! finished) {
        Command command = parser.getCommand();
        finished = processCommand(command);
    }
    System.out.println("Thank you for playing.  Good bye.");
}

/**
 * Print out the opening message for the player.
 */
private void printWelcome()
{
    System.out.println();
    System.out.println("Welcome to The World of Zuul!");
```

Code 7.8 continued
Two methods with a good degree of cohesion

```
        System.out.println(
                "Zuul is a new, incredibly boring adventure game.");
        System.out.println("Type 'help' if you need help.");
        System.out.println();
        System.out.println(currentRoom.getLongDescription());
    }
```

From a functional point of view we could have just entered the statements from the `printWelcome` method directly in the `play` method, and achieved the same result without defining an extra method and making a method call. The same can, by the way, be said for the `processCommand` method that is also invoked in the `play` method: this code, too, could have been written directly into the `play` method.

It is, however, much easier to understand what a segment of code does, and to make modifications, if short, cohesive methods are used. In the chosen method structure, all methods are reasonably short and easy to understand, and their names indicate their purposes quite clearly. These characteristics represent valuable help for a maintenance programmer.

7.11.2 Cohesion of classes

Concept:

Class cohesion. A cohesive class represents one well-defined entity.

The rule of cohesion of classes states that each class should represent one single, well-defined entity in the problem domain.

As an example of class cohesion, we now discuss another extension to the *zuul* project. We now want to add *items* to the game. Each room may hold an item, and each item has a description and a weight. An item's weight can be used later to determine whether it can be picked up or not.

A naïve approach would be to add two fields to the `Room` class: `itemDescription` and `itemWeight`. This could work. We could now specify the item details for each room, and we could print out the details whenever we enter a room.

This approach, however, does not display a good degree of cohesion: the `Room` class now describes both a room and an item. It also suggests that an item is bound to a particular room, which we might not wish to be the case.

A better design would create a separate class for items, probably called `Item`. This class would have fields for a description and weight, and a room would simply hold a reference to an item object.