CHAPTER

2 Understanding Class Definitions

Main concepts discussed in this chapter:

- fields
- constructors
- parameters
- methods (accessor, mutator)
- assignment and conditional statement

Java constructs discussed in this chapter:

field, constructor, comment, parameter, assignment (=), block, return statement, `void`, compound assignment operators (`+=`, `-=`), if-statement

In this chapter, we take our first proper look at the source code of a class. We will discuss the basic elements of class definitions: *fields*, *constructors*, and *methods*. Methods contain statements, and initially we look at methods containing only simple arithmetic and printing statements. Later, we introduce *conditional statements* that allow choices between different actions to be made within methods.

We shall start by examining a new project in a fair amount of detail. This project represents a naíve implementation of an automated ticket machine. As we start by introducing the most basic features of classes, we shall quickly find that this implementation is deficient in a number of ways. So we shall then proceed to describe a more sophisticated version of the ticket machine that represents a significant improvement. Finally, in order to reinforce the concepts introduced in this chapter, we take a look at the internals of the *lab-classes* example encountered in Chapter 1.

2.1 Ticket machines

Train stations often provide ticket machines that print a ticket when a customer inserts the correct money for their fare. In this chapter, we shall define a class that models something like these ticket machines. As we shall be looking inside our first Java example classes, we shall keep our simulation fairly simple to start with. That will give us the

opportunity to ask some questions about how these models differ from the real-world versions, and how we might change our classes to make the objects they create more like the real thing.

Our ticket machines work by customers "inserting" money into them and then requesting a ticket to be printed. Each machine keeps a running total of the amount of money it has collected throughout its operation. In real life, it is often the case that a ticket machine offers a selection of different types of ticket, from which customers choose the one they want. Our simplified machines print tickets of only a single price. It turns out to be significantly more complicated to program a class to be able to issue tickets of different values, than it does to offer a single price. On the other hand, with object-oriented programming it is very easy to create multiple instances of the class, each with its own price setting, to fulfill a need for different types of tickets.

2.1.1 Exploring the behavior of a naïve ticket machine

Concept

Object creation: Some objects cannot be constructed unless extra information is provided.

Open the *naíve-ticket-machine* project in BlueJ. This project contains only one class—**`TicketMachine`**—which you will be able to explore in a similar way to the examples we discussed in Chapter 1. When you create a **`TicketMachine`** instance, you will be asked to supply a number that corresponds to the price of tickets that will be issued by that particular machine. The price is taken to be a number of cents, so a positive whole number such as 500 would be appropriate as a value to work with.

Exercise 2.1 Create a **`TicketMachine`** object on the object bench and take a look at its methods. You should see the following: **`getBalance`**, **`getPrice`**, **`insertMoney`**, and **`printTicket`**. Try out the **`getPrice`** method. You should see a return value containing the price of the tickets that was set when this object was created. Use the **`insertMoney`** method to simulate inserting an amount of money into the machine. The machine stores as a balance the amount of money inserted. Use **`getBalance`** to check that the machine has kept an accurate record of the amount just inserted. You can insert several separate amounts of money into the machine, just like you might insert multiple coins or bills into a real machine. Try inserting the exact amount required for a ticket, and use **`getBalance`** to ensure that the balance is increased correctly. As this is a simple machine, a ticket will not be issued automatically, so once you have inserted enough money, call the **`printTicket`** method. A facsimile ticket should be printed in the BlueJ terminal window.

Exercise 2.2 What value is returned if you get the machine's balance after it has printed a ticket?

Exercise 2.3 Experiment with inserting different amounts of money before printing tickets. Do you notice anything strange about the machine's behavior? What happens if you insert too much money into the machine—do you receive any refund? What happens if you do not insert enough and then try to print a ticket?