8.9 enum Types

In Fig. 6.8, we introduced the basic `enum` type, which defines a set of constants represented as unique identifiers. In that program the `enum` constants represented the game's status. In this section we discuss the relationship between `enum` types and classes. Like classes, all `enum` types are *reference* types. An `enum` type is declared with an `enum` **declaration**, which is a comma-separated list of `enum` *constants*—the declaration may optionally include other components of traditional classes, such as constructors, fields and methods (as you'll see momentarily). Each `enum` declaration declares an `enum` class with the following restrictions:

1. `enum` constants are *implicitly* `final`.
2. `enum` constants are *implicitly* `static`.
3. Any attempt to create an object of an `enum` type with operator `new` results in a compilation error.

The `enum` constants can be used anywhere constants can be used, such as in the `case` labels of `switch` statements and to control enhanced `for` statements.

Declaring Instance Variables, a Constructor and

Methods in an `enum` Type

Figure 8.10 demonstrates instance variables, a constructor and methods in an `enum` type. The `enum` declaration contains two parts—the `enum` constants and the other members of the `enum` type.

```
// Fig. 8.10: Book.java
// Declaring an enum type with a constructor and
// and accessors for these fields

public enum Book {
   // declare constants of enum type
   JHTP(“Java How to Program”, “2018”),
   CHTP(“C How to Program”, “2016”),
   IW3HTP(“Internet & World Wide Web How to Prog
   CPPHTP(“C++ How to Program”, “2017”),
   VBHTP(“Visual Basic How to Program”, “2014”),
   CSHARPHTP(“Visual C# How to Program”, “2017”)

   // instance fields
   private final String title; // book title
   private final String copyrightYear; // copyri

   // enum constructor
   Book(String title, String copyrightYear) {
      this.title = title;
      this.copyrightYear = copyrightYear;
   }

   // accessor for field title
   public String getTitle() {
      return title;
   }

   // accessor for field copyrightYear
   public String getCopyrightYear() {
```

```
31          return copyrightYear;
32       }
33    }
```

Fig. 8.10

Declaring an `enum` type with a constructor and explicit instance fields and accessors for these fields.

The first part (lines 7–12) declares six constants. Each is optionally followed by arguments that are passed to the `enum` **constructor** (lines 19–22). Like the constructors in classes, an `enum` constructor can specify any number of parameters and can be overloaded. In this example, the `enum` constructor requires two `String` parameters—one that specifies the book's title and one that specifies its copyright year. To properly initialize each `enum` constant, we follow it with parentheses containing two `String` arguments.

The second part (lines 15–32) declares the `enum` type's other members—instance variables `title` and `copyrightYear` (lines 15–16), a constructor (lines 19–22) and two methods (lines 25–27 and 30–32) that return the book title and copyright year, respectively. Each `enum` constant in `enum` type `Book` is an object of `enum` type `Book` that has its own copy of instance variables.

Using enum type Book

Figure 8.11 tests the `Book enum` and illustrates how to iterate through a range of its constants. For every `enum`, the compiler generates the `static` method `values` (called in line 10), which returns an array of the `enum`’s constants in the order they were declared. Lines 10–13 display the constants. Line 12 invokes the `enum Book`’s `getTitle` and `getCopyrightYear` methods to get the title and copyright year associated with the constant. When an `enum` constant is converted to a `String` (e.g., `book` in line 11), the constant’s identifier is used as the `String` representation (e.g., `JHTP` for the first `enum` constant).

```
1  // Fig. 8.11: EnumTest.java
2  // Testing enum type Book.
3  import java.util.EnumSet;
4
5  public class EnumTest {
6     public static void main(String[] args) {
7        System.out.println(“All books:”);
8
9        // print all books in enum Book
10       for (Book book : Book.values()) {
11          System.out.printf(“%-10s%-45s%s%n”, boo
12             book.getTitle(), book.getCopyrightYe
13       }
14
15       System.out.printf(“%nDisplay a range of en
16
17       // print first four books
18       for (Book book : EnumSet.range(Book.JHTP,
19          System.out.printf(“%-10s%-45s%s%n”, boo
20             book.getTitle(), book.getCopyrightYe
```

```
         }
      }
   }
```

```
All books:
```

JHTP	Java How to Program	2018
CHTP	C How to Program	2016
IW3HTP	Internet & World Wide Web How to Program	2012
CPPHTP	C++ How to Program	2017
VBHTP	Visual Basic How to Program	2014
CSHARPHTP	Visual C# How to Program	2017

```
Display a range of enum constants:
```

JHTP	Java How to Program	2018
CHTP	C How to Program	2016
IW3HTP	Internet & World Wide Web How to Program	2012
CPPHTP	C++ How to Program	2017

Fig. 8.11

Testing `enum` type `Book`.

Lines 18–21 use the `static` method `range` of class `EnumSet` (package `java.util`) to display a range of the `enum Book`'s constants. Method `range` takes two parameters—the first and the last `enum` constants in the range—and returns an `EnumSet` that contains all the constants between these two constants, inclusive. For example, the expression `EnumSet.range(Book.JHTP, Book.CPPHTP)` returns an `EnumSet` containing `Book.JHTP`, `Book.CHTP`, `Book.IW3HTP` and `Book.CPPHTP`. The enhanced `for` statement can be used with an `EnumSet` just as it can with an array, so lines 18–21 use it to display the title and copyright year of every book in the `EnumSet`. Class `EnumSet` provides several other `static` methods for creating sets of `enum` constants from the same `enum` type.

Common Programming Error 8.4

In an `enum` *declaration, it's a syntax error to declare* `enum` *constants after the* `enum` *type's constructors, fields and methods.*