Supplement: Enumerated Types

For Introduction to Java Programming
By Y. Daniel Liang

1 Simple Enumerated Types

An enumerated type defines a list of enumerated values. Each value is an identifier. For example, the following statement declares a type, named **MyFavoriteColor**, with values **RED**, **BLUE**, **GREEN**, and **YELLOW** in this order.

```
enum MyFavoriteColor {RED, BLUE, GREEN, YELLOW};
```

A value of an enumerated type is like a constant and so, by convention, is spelled with all uppercase letters. So, the preceding declaration uses RED, not red. By convention, an enumerated type is named like a class with first letter of each word capitalized.

Once a type is defined, you can declare a variable of that type:

```
MyFavoriteColor color;
```

The variable **color** can hold one of the values defined in the enumerated type **MyFavoriteColor** or **null**, but nothing else. Java enumerated type is *type-safe*, meaning that an attempt to assign a value other than one of the enumerated values or **null** will result in a compilation error.

The enumerated values can be accessed using the syntax

```
enumeratedTypeName.valueName
```

For example, the following statement assigns enumerated value **BLUE** to variable **color**:

```
color = MyFavoriteColor.BLUE;
```

Note that you have to use the enumerated type name as a qualifier to reference a value such as BLUE.

As with any other type, you can declare and initialize a variable in one statement:

```
MyFavoriteColor color = MyFavoriteColor.BLUE;
```

An enumerated type is treated as a special class. An enumerated type variable is therefore a reference variable.

An enumerated type is a subtype of the **Object** class and the **Comparable** interface. Therefore, an enumerated type inherits all the methods in the **Object** class and the **compraeTo** method in the **Comparable** interface. Additionally, you can use the following methods on an enumerated object:

- **public String name();**
  Returns a name of the value for the object.

- **public int ordinal();**
  Returns the ordinal value associated with the enumerated value. The first value in an enumerated type has an ordinal value of 0, the second has an ordinal value of 1, the third one 3, and so on.

Listing 1 gives a program that demonstrates the use of enumerated types. Figure 1 shows a sample run of the program.

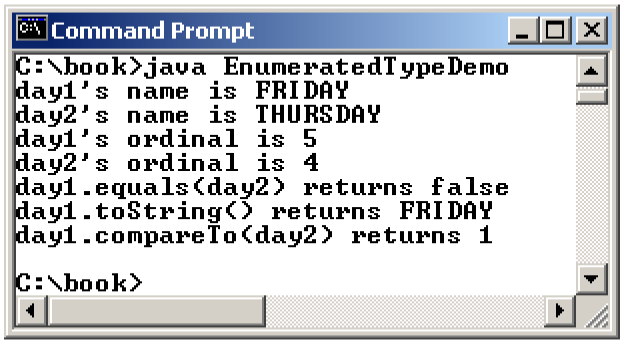
Listing 1 EnumeratedTypeDemo.java (Declaring enumerated type inside a class)

```
public class EnumeratedTypeDemo {
  static enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
THURSDAY,
    FRIDAY, SATURDAY};

  public static void main(String[] args) {
    Day day1 = Day.FRIDAY;
    Day day2 = Day.THURSDAY;

    System.out.println("day1's name is " + day1.name());
    System.out.println("day2's name is " + day2.name());
    System.out.println("day1's ordinal is " + day1.ordinal());
    System.out.println("day2's ordinal is " + day2.ordinal());

    System.out.println("day1.equals(day2) returns " +
      day1.equals(day2));
    System.out.println("day1.toString() returns " +
      day1.toString());
    System.out.println("day1.compareTo(day2) returns " +
      day1.compareTo(day2));
  }
}
```



```
C:\book>java EnumeratedTypeDemo
day1's name is FRIDAY
day2's name is THURSDAY
day1's ordinal is 5
day2's ordinal is 4
day1.equals(day2) returns false
day1.toString() returns FRIDAY
day1.compareTo(day2) returns 1

C:\book>
```

Figure 1

An enumerated type is a class type.

An enumerated type is defined in Lines 2-3. Variables **day1** and **day2** are declared as the **Day** type and assigned enumerated values in Lines 6-7. Since **day1**'s value is FRIDAY, its ordinal value is 5 (Line 11). Since **day2**'s value is THURSDAY, its ordinal value is 4 (Line 12).

An enumerated type is a subclass of the **Object** class and the **Comparable** interface, so you can invoke the methods **equals**, **toString**, and **comareTo** from an enumerated object reference variable (Lines 14-19). **day1.equals(day2)** returns true if **day1** and **day2** have the same ordinal value. **day1.compareTo(day2)** returns the difference between **day1**'s ordinal value to **day2**'s.

Alternatively, you can rewrite the code in Listing 1 into Listing 2.

Listing 2 EnumeratedTypeDemo1.java (Declaring enumerated type standalone)

```
public class EnumeratedTypeDemo1 {
  public static void main(String[] args) {
    Day day1 = Day.FRIDAY;
    Day day2 = Day.THURSDAY;

    System.out.println("day1's name is " + day1.name());
    System.out.println("day2's name is " + day2.name());
    System.out.println("day1's ordinal is " + day1.ordinal());
    System.out.println("day2's ordinal is " + day2.ordinal());

    System.out.println("day1.equals(day2) returns " +
      day1.equals(day2));
    System.out.println("day1.toString() returns " +
      day1.toString());
    System.out.println("day1.compareTo(day2) returns " +
      day1.compareTo(day2));
  }
}

enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
  FRIDAY, SATURDAY}
```

An enumerated type can be defined inside a class, as shown in Lines 2-3 in Listing 1, or standalone as shown in Lines 20-21 Listing 2. In the former case, the type is treated as an inner class. After the program is compiled, a class named EnumeratedTypeDemo$Day.class is created. In the latter case, the type is treated as a standalone class. After the program is compiled, a class named Day.class is created.

NOTE

Since an enumerated type is like a class, the type should be named in the same way as a class with the first letter capitalized. Since the enumerated values are constants, they should be named as regular constants.

NOTE

When an enumerated type is declared inside a class, the type must be declared as a member of the class and cannot be declared inside a method. Furthermore, the type is always **static**. For this reason, the **static** keyword in Line 2 in Listing 1 may be omitted. The visibility modifiers on inner class can be also be applied to enumerated types defined inside a class.

TIP

Using enumerated values (e.g., **Day.MONDAY**, **Day.TUESDAY**, and so on) rather than literal integer values (e.g., 0, 1, and so on) can make program easier to read and maintain.

2 Using if or switch Statements with an Enumerated Variable

An enumerated variable holds a value. Often your program needs to perform a specific action depending on the value. For example, if the value is **Day.MONDAY**, play soccer; if the value is **Day.TUESDAY**, take piano lesson, and so on. You can use an **if** statement or a **switch** statement to test the value in the variable, as shown in (a) and (b)

```
if (day.equals(Day.MONDAY)) {
  // process Monday
}
else if (day.equals(Day.TUESDAY)) {
  // process Tuesday
}
else
  ...
```

(a)

Equivalent

```
switch (day) {
  case MONDAY:
    // process Monday
    break;
  case TUESDAY:
    // process Tuesday
    break;
  ...
}
```

(b)

In the **switch** statement in (b), the case label is an unqualified enumerated value (e.g., **MONDAY**, but not **Day.MONDAY**).

3 Processing Enumerated Values Using Enhanced **for** Loop

Each enumerated type has a static method **values()** that returns all enumerated values for the type in an array. For example,

```
Day[] days = Day.values();
```

You can use a regular for loop in (a) or an enhanced for loop in (b) to process all the values in the array.

```
for (int i = 0; i < days.length; i++)
  System.out.println(days[i]);
```

(a)

Equivalent

```
for (Day day: days)
  System.out.println(day);
```

(b)

4 Enumerated Types with Attributes and Methods

The simple enumerated types introduced in the preceding section define a type with a list of enumerated values. You can also define an enumerate type with attributes and methods, as shown in Listing 3.

Listing 3 TrafficLight.java (Enumerated type with attributes and methods)

```
public enum TrafficLight {
  red ("Please stop"), GREEN ("Please go"), YELLOW ("Please
caution");

  private String description;

  private TrafficLight(String description) {
    this.description = description;
  }
```

```
    public String getDescription() {
      return description;
    }
  }
```

The enumerated values are defined in Line 2. The value declaration must be the first statement in the type declaration. A data field named **description** is declared in Line 4 to describe an enumerated value. The constructor **TrafficLight** is declared in Lines 6-8. The constructor is invoked whenever an enumerated value is accessed. The enumerated value’s argument is passed to the constructor, which is then assigned to **description**.

Listing 4 gives a test program to use **TrafficLight**.

Listing 4 TestTrafficLight.java

```
public class TestTrafficLight {
  public static void main(String[] args) {
    TrafficLight light = TrafficLight.red;
    System.out.println(light.getDescription());
  }
}
```

An enumerated value **TrafficLight.red** is assigned to variable **light** (Line 3) Accessing **TrafficLight.red** causes the JVM to invoke the constructor with argument “please stop”. The methods in enumerated type are invoked in the same was as the methods in a class. **light.getDescription()** returns the description for the enumerated value (Line 4).

NOTE

The Java syntax requires that the constructor for enumerated types be private to prevent it from being invoked directly. The private modifier may be omitted. In this case, it is considered private by default.