10.10 Java SE 8 Interface Enhancements

8

This section introduces interface features that were added in Java SE 8. We discuss these in more detail in later chapters.

10.10.1 `default` Interface Methods

Prior to Java SE 8, interface methods could be *only* `public abstract` methods. This meant that an interface specified *what* operations an implementing class must perform but not *how* the class should perform them.

As of Java SE 8, interfaces also may contain `public default` **methods** with *concrete* default implementations that specify *how* operations are performed when an implementing class does not override the methods. If a class implements such an interface, the class also receives the interface's `default` implementations (if any). To declare a `default` method, place the keyword `default` before the method's return type and provide a concrete method implementation.

Adding Methods to Existing Interfaces

Prior to Java SE 8, adding methods to an interface would break any implementing classes that did not implement the new methods. Recall that if you didn't implement each of an interface's methods, you had to declare your class `abstract`.

Any class that implements the original interface will *not* break when a `default` method is added—the class simply receives the new `default` method. When a class implements a Java SE 8 interface, the class "signs a contract" with the compiler that says, "I will declare all the `abstract` methods specified by the interface or I will declare my class `abstract`"—the implementing class is not required to override the interface's `default` methods, but it can if necessary.

Software Engineering Observation 10.13

Java SE 8 `default` *methods enable you to evolve existing interfaces by adding new methods to those interfaces without breaking code that uses them.*

Interfaces vs. `abstract`

Classes

Prior to Java SE 8, an interface was typically used (rather than an `abstract` class) when there were no implementation details to inherit—no fields and no method implementations. With `default` methods, you can instead declare common method implementations in interfaces. This gives you more flexibility in designing your classes, because a class can implement many interfaces, but can extend only one superclass.

10.10.2 `static` Interface Methods

Prior to Java SE 8, it was common to associate with an interface a class containing `static` helper methods for working with objects that implemented the interface. In Chapter 16, you'll learn about class `Collections` which contains many `static` helper methods for working with objects that implement interfaces `Collection`, `List`, `Set` and more. For example, `Collections` method `sort` can sort objects of *any* class that implements interface `List`. With `static` interface methods, such helper methods can now be declared directly in interfaces rather than in separate classes.

10.10.3 Functional

Interfaces

As of Java SE 8, any interface containing only one `abstract` method is known as a **functional interface**—these are also called SAM (single `abstract` method) interfaces. There are many such interfaces throughout the Java APIs. Some functional interfaces that you'll use in this book include:

- `ChangeListener` (Chapter 12)—You'll implement this interface to define a method that's called when the interacts with a slider graphical user interface control.
- `Comparator` (Chapter 16)—You'll implement this interface to define a method that can compare two objects of a given type to determine whether the first object is less than, equal to or greater than the second.
- `Runnable` (Chapter 23)—You'll implement this interface to define a task that may be run in parallel with other parts of your program.

Functional interfaces are used extensively with Java's lambda capabilities that we introduce in Chapter 17. Lambdas provide a shorthand notation for implementing functional interfaces.