Java Object-Oriented Concepts Unit

Lesson 9: Interfaces

The Learning House

427 S 4th Street #300

Louisville KY 40202

Lesson 9: Interfaces

Overview

In this lesson, we're going to talk about another feature of the Java language that helps us organize our code and specify how other code is to interact with our components. We will cover declaration, implementation, and extension of interfaces. We will also look at the restrictions that Java places on interfaces and how interfaces work with polymorphism.

What is an Interface?

An interface is a contract. It defines a set of methods that must be implemented by any class that says it adheres to the contract. The key concept here is that interfaces only define the methods — they do not provide any implementations.

Declaring an Interface

Declaration of an interface is similar to the declaration of a class:

```
public interface Colorable {
      public void setColor(String color);
      public String getColor();
}
```

The key differences being:

- We use the **interface** keyword instead of the **class** keyword
- None of the defined methods have implementations
- We place a semicolon after each method definition

Implementing an Interface

Having a class implement an interface requires two things:

1. The class must declare that it implements a given interface.
2. The class must then either provide implementation for each of the methods defined in the interface or be declared abstract.

The following example shows a class called Ball that implements the Colorable interface, defined above:

```
public class Ball implements Colorable {
    // some methods...

    public void setColor(String color) {
        // implementation code here
    }
    public String getColor() {
    // implementation code here
```

```
    }
}
```

Interface Restrictions

Java imposes several restrictions on interfaces:

1. Interfaces cannot have member fields (but they may define constants).
2. None of the methods can have implementations (Java 8 does allow interfaces to have default implementations of methods that are useful in certain contexts).

Implementing Multiple Interfaces

A class may implement more than one interface. Such a class must simply list all of the interfaces that it implements and then provide implementations of all of the methods of each interface.

Extending an Interface

Interfaces can be extended in the same way that classes can be extended. For example, see DebugLogging extends Debuggable:

```
public interface Debuggable {
      public void displayStatus(String id);
      public void displayError(String error);
}
public interface DebugLogging extends Debuggable {
      public void logStatus(String id);
      public void logError(String error);
}
```

Any class that implements DebugLogging must provide implementations for the methods defined in DebugLogging **and** for all the methods defined in Debuggable.

Interfaces and Polymorphism

Interfaces can be treated polymorphically much like classes. For example, a class that implements the DebugLogging interface can be referred to with either a DebugLogging or a Debuggable reference. A class that implements one or more interfaces may also be referred to with a reference to its class type.

For example, let's look at the Manager class discussed in the Specialization and Inheritance lesson. In our example, the Manager class extends Employee. If Manager also implemented DebugLogging, we could refer to any given Manager object instance with any of the following references:

- Manager
- Employee
- DebugLogging
- Debuggable