Introduction to Interfaces

by Sophia

WHAT'S COVERED

In this lesson, you will learn about relating groups of classes using interfaces, when using Java.

Table of Contents

1. Class Interfaces

In Java and other programming languages, the way that an object interacts with other objects is referred to as its **interface**. For instance, a class's public and protected methods and attributes define the class's programming interface.

There is also a specific Java programming construct called an interface. This construct is designed to allow the way a class interacts with other objects to be abstracted and separate from the specifics of the implementation. An interface can be thought of as a programming contract that specifies behaviors that a class will make available when it implements an interface.

TERM TO KNOW

Interface
The defined ways in which one object interacts with another.

---

2. Defining an Interface

A Java interface is similar to a Java class, but they lack attributes or instance variables. An interface consists of methods, but most of the time these methods lack bodies (implementations) and the interface just specifies the signature and return type of the methods. This allows an interface to specify what classes that implement the interface can do, but not how they do it.

Below is an example of an interface that specifies some common methods (behavior) that classes using it

must implement. This example also includes a constant DecimalFormat that all classes implementing the interface can use.

Just like a Java class, an interface needs to be declared in a .java file with a name matching the name of the interface (in this case, Beverage.java):

```
import java.text.DecimalFormat; // Needed for DecimalFormat for price

public interface Beverage {
  // Shared constant for price format. Available to all classes
  // that implement the Beverage interface.
  final DecimalFormat PriceFormat = new DecimalFormat("$0.00");

  // Common methods
  String getName();  // Coffee, Tea, etc...
  int getSize();     // Size in ounces
  double getPrice(); // Price in dollars
}
```

While the Beverage interface does not specify how the classes have to do it, any class that implements Beverage must have a method called getName() that returns a String, a method namedgetSize() that returns an integer, and a method called getPrice() that returns a double. Exactly how these methods do their work is up to the classes implementing the interface, but the interface is a kind of contract that stipulates that these methods are available and returns the data types specified (none of these methods have any parameters). The DecimalFormat constant named PriceFormat can also be used by any class that implements the Beverage interface.

The Beverage interface, like a Java class, needs to be compiled before it can be used (implemented) by other code:

↗ EXAMPLE

```
javac Beverage.java
```

3. Implementing an Interface

A Java interface by itself does not do anything. It engages when classes explicitly implement it. Here are a couple examples of classes that implement the Beverage interface. First, the Coffee class, which needs to be in a file called Coffee.java:

```
public class Coffee implements Beverage {
  private String roastType;
  private int size;
  private boolean decaf;
  private double price;
```

```
  public Coffee(String roastType, int size, boolean decaf, double price) {
      this.roastType = roastType;
      this.size = size;
      this.decaf = decaf;
      this.price = price;
  }

  // Required by Beverage interface
  public String getName() {
   return "coffee";
  }

  // Required by Beverage interface
  public int getSize() {
   return size;
  }

  // Required by Beverage interface
  public double getPrice() {
   return price;
  }

  public boolean isDecaf() {
   return decaf;
  }

  @Override // toString inherited from Java Object base class
  public String toString() {
   // PriceFormat is constant provided by Beverage interface
   String item = roastType + " coffee (" + size + " oz.) " + PriceFormat.format(price);
   if(decaf) {
    return "decaf " + item;
   }
   else {
    return item;
   }
  }
}
```

The `Coffee` class has the attributes and parameterized constructor that you might expect for such a class. Note, too, how the `Coffee` class has specific **implementations** for the `getName()`, `getSize()`, and `getPrice()` methods, along with the accessor methods for the attributes that are specific to coffee. The Beverage interface provides the `PriceFormat` that is used as part of the `toString()` method.

The `Tea` class also implements the Beverage interface:

```
public class Tea implements Beverage {
  private String teaType;
  private int size;
  private boolean iced;
  private double price;

  public Tea(String teaType, int size, boolean iced, double price) {
    this.teaType = teaType;
    this.size = size;
    this.iced = iced;
    this.price = price;
  }

  public String getName() {
    return "tea";
  }

  public int getSize() {
    return size;
  }

  public double getPrice() {
    return price;
  }

  public boolean isIced() {
    return iced;
  }

  public String toString() {
    String item = teaType + " tea (" + size + " oz.) " + PriceFormat.format(price);
    if(iced) {
      return "iced " + item;
    }
    else {
      return "hot " + item;
    }
  }
}
```

Implementation

The specific code written to produce a method that carries out the action described or specified in an interface.

---

4. Using an Interface as a Type

Classes that implement the same interface can be treated as having the same data type. This means that you can write methods that take classes that implement an interface as the same type for parameters or return values.

Here is a sample class called DrinkOrder that shows how Beverage can be used as a data type with a collection or with methods:

TRY IT

Directions: Enter this code in Replit:

```
import java.util.ArrayList;

public class DrinkOrder {
  // ArrayList can hold both Coffee & Tea objects since both implement the
  // Beverage interface
  private ArrayList<Beverage> order = new ArrayList<>();

  // add() method accepts an object any class that implements Beverage interface
  public void add(Beverage beverage) {
    order.add(beverage);
  }

  // Add up total for order
  public double getTotalPrice() {
    double total = 0;
    for(Beverage beverage : order) {
      total += beverage.getPrice();
    }
    return total;
  }

  // Return ArrayList of Beverages in order
  public ArrayList<Beverage> getOrder() {
    return order;
  }
}
```

Here is a fairly simple program that uses this class (we'll leave out the user input and menu prompts that you may remember from an earlier drink order program).

TRY IT

Directions: Enter this code in Replit:

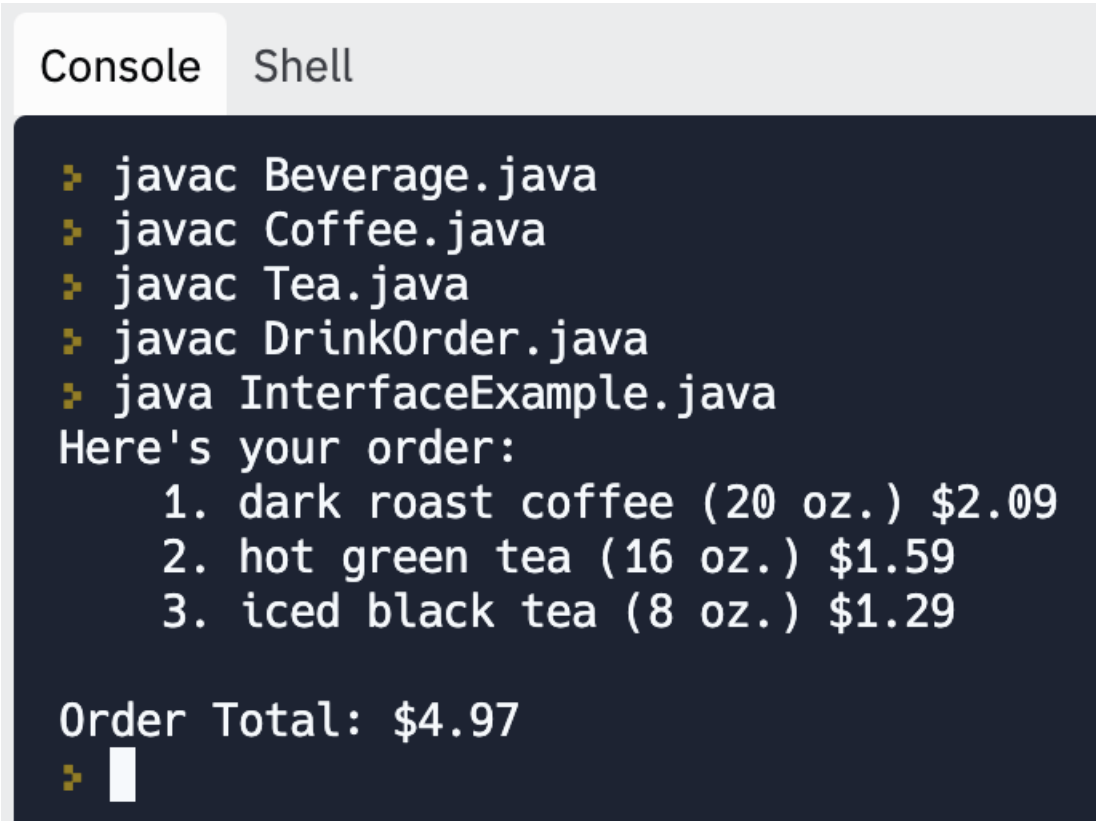
```
class InterfaceExample {
  public static void main(String[] args) {
    // Create order and add drinks
```

```
    DrinkOrder toGoOrder = new DrinkOrder();
    Coffee coffee = new Coffee("dark roast", 20, false, 2.09);
    toGoOrder.add(coffee);
    Tea greenTea = new Tea("green", 16, false, 1.59);
    toGoOrder.add(greenTea);
    Tea blackTea = new Tea("black", 8, true, 1.29);
    toGoOrder.add(blackTea);

    // Number to use for numbered list of drinks in order
    int itemNumber = 1;
    System.out.println("Here's your order: ");
    // getOrder() returns ArrayList<Beverage>
    for(Beverage bev : toGoOrder.getOrder()) {
      System.out.println("\t" + itemNumber++ + ". " + bev);
    }

    System.out.println("\nOrder Total: " +
        // Note how Beverage.PriceFormat can be used here, too
        Beverage.PriceFormat.format(toGoOrder.getTotalPrice()));
    }
}
```

The output from running this program (after compiling the interface and needed classes, if not already done) should look like this:



SUMMARY

In this lesson, you have learned how a Java **interface can be defined** as a kind of contract that

specifies what behaviors (methods) and sometimes attributes a class will include when it implements a **class interface**. You learned that an interface specifies how classes can interact without specifying the internal details. You also learned that interfaces can serve as a common type for the classes that **implement the interface**. Finally, you learned that an **interface can be used with collections, as a method parameter type**, and as a return type.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source **cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf**

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source **py4e.com/html3/**

TERMS TO KNOW

Implementation

The specific code written to produce a method that carries out the action described or specified in an interface.

Interface

The defined ways in which one object interacts with another.