Interfaces

Recall that an abstract method in Java is a method with no body. This is used in situations where we would like a number of derived classes of a common base class to all share a common behavior, but we cannot specify at the base class level exactly how the behavior works, only that it exists.

Examples:

- Base class Pet, derived classes Cat, Dog, etc. We would like all Pets to be able to "speak," so we need to define a speak() method at the Pet-class level, so all the derived classes can inherit this method. However, it is unclear what the speak() method would actually do at the base-class level. What code would go in this method? So we mark it as abstract, and the Pet class becomes abstract.
- Base class Shape, derived classes Circle, Square, etc. We would like each class to be able to calculate its area; that behavior should be common to all shapes. Therefore, that behavior belongs at the base class level so all the derived classes can inherit it. But what would a getArea() method do in the Shape class? At that level, we don't know how to calculate the area of a generic shape, because the mathematical formula depends on what type of shape it is. So we mark it abstract.

Abstract classes usually contain a combination of abstract methods (methods with no bodies/code) and non-abstract (concrete) methods (methods with bodies).

An **interface** is like an abstract class, but contains only abstract methods; none of the methods have any code. Furthermore, no instance variables may be specified in an interface.

Interfaces are often used in situations where we want to specify common behaviors for a group of classes, but all the classes might not be related through a common is-a inheritance hierarchy.

```
public interface NameOfInterface {
   public returnType methodName1(type1 param1, type2 param2, ...);
   public returnType methodName2(type1 param1, type2 param2, ...);
}
```

The keyword to use when a class wants to use the interface is called **`implements`**, rather than extends:

```
public class MyClass implements NameOfInterface { . . . }
```

or

```
public class MyClass extends BaseClass implements NameOfInterface { ... }
```

You can even have a class implement multiple interfaces:

```
public class MyClass extends BaseClass implements Interface1, Interface2 { ... }
```