CS112 Lecture: Interfaces

Last revised 2/29/08

Objectives:

1. To introduce Java interfaces

Materials:

1. BlueJ project containing RobotRacer from ActiveObjects lecture plus javadoc
2. Source code for java.lang.Runnable - edited to omit prologue comment
3. Main class for Lab 3; Main class modified to use static import
4. Example of full and abbreviated forms of an interface declaration

I. Introduction

A. Thus far in the course, we have seen the notion of a class, which serves as a sort of template from which individual objects can be constructed.

B. A key feature of a class is that it specifies both the interface and the implementation of a family of objects.

1. The interface of an object is a description of the services it makes available to other objects - its public methods (behaviors) and constants - i.e. it describes what the an object provides.

2. The implementation of an object is how the object actually performs these things.

C. When we are creating a class, we are generally not conscious of this distinction. It does show up, though, when we look at the javadoc documentation for a class.

PROJECT side by side the code and Javadoc for RacerRobot

What features of the class are included in the Javadoc? What are not?

ASK

public features are included, private features are not

D. The public features of a class constitute its interface - what other classes can make use of. The private features of a class constitute its implemenation - how it actually performs the tasks specified by its interface.

II. Java Interfaces

A. We have become familiar with the notion of inheritance in both the robot world and objectdraw.

1. In the robot world, a special kind of robot typically extends Robot, and inherits all the capabilities of Robot (e.g. move() etc).

2. In the objectdraw world, we have sometimes extended objectdraw classes like WindowController or ActiveObject. Once again, the class we created inherits all the features of the base class (e.g. the canvas in the case of a WindowController).

3. On occassion, we have needed to have a subclass override an inherited feature of the base class.

   Example: Look at code for move() and turnLeft() in RacerRobot. Note how these override the inherited methods to inject a random pause.

4. That is to say, when a class inherits from another class, it inherits both the base class's interface and implementation, unless it explicitly overrides the base class's implementation (in which case it still inherits the interface in the sense that it still has a method with the given signature.)

B. Sometimes, though, we want to inherit an interface but not an implementation. Java makes this possible with an entity known as an interface. Let's look at an example from the Java library itself.

   PROJECT (edited) source code for java.lang.Runnable

1. An interface is declared the same way as a class is, except the declaration uses the word "interface" instead of the word "class".

2. An interface declaration can contain only a subset of the things that a class declaration may contain. These restrictions follow from the fact that we are declaring just an interface, not an implementation.

   a) An interface declaration may not contain anything that is not public.

   b) An interface declaration may not contain any variables.

   c) An interface declaration may not contain class methods (since they must necessarily have an implementation).

   d) An interface declaration may not contain constructors.

e) An interace declaration may contain instance method headings - but not their implementation. (Note the reserved word "abstract" and the semicolon after the method header instead of a method body).

3. An interface is used when we want to specify that a class includes a certain behavior (having a certain signature), without worrying about how the behavior is implemented.

   In this example, note the prologue comment to run()! This interface specifies the behavior expected of any "Runnable" object. Examples we have seen include:

   a) Main classes for Robot problems

      PROJECT Main for Lab 3

   b) Clock created in Lab 6

      PROJECT documentation for objectdraw ActiveObject class - note that it implements Runnable

   c) Robot class in the example we just did

      PROJECT code for RacerRobot

C. In Java, a class that has the behavior required by the interface "implements" the interface.

   SHOW in code for RacerRobot

   Note the Java wording - "implements" rather than "extends" - stressing that we are implementing one or more behaviors from scratch, rather than extending an inherited implementation of them.

D. In Java, it is also standard practice to use an interface if one wants to share symbolic constant definitions

   1. Example: Robot main program classes typically implement Directions.

      Show javadoc for Directions - note that all it does is define constants

      Show implements Directions in Main for Lab 3

   2. This is actually a bit kludgy. Starting with java 1.5, it has also been possible to import the contents of an interface.

      PROJECT MainModified.java- note use of word static in import statement

While this seems like a clearer way to accomplish the task, the earlier use of implements to inherit static constants has become something of a Java idiom.

3. Note that, in any case, we are creating an interface for the sole purpose of defining constants, not behavior - still a bit kludgy, but widely used.

III. Rules for Interfaces

A. We have previously noted that interfaces can contain only two kinds of things: instance methods, and class constants. Methods must be abstract - that is, they cannot contain a body.

B. We have also previously noted that everything in an interface must be public.

C. For these reasons, certain keywords may be omitted in declaring an interface - their presence is assumed.

1. The keyword "public" may be omitted from methods and constants - since everything must be public.

   a) But another word like "private" may not appear.

   b) The interface itself must still be declared public, since it is possible to have an interface that has a different visibility.

2. The keyword "abstract" may be omitted from methods - since methods are necessarily abstract. (The semicolon after the method header must still appear, though).

3. The keywords "static" and "final" may be omitted from constants - since the only fields that may appear are class constants.

   PROJECT - Two alternate definitions for an interface - full form, and all unnecessary keywords omitted

IV. Interfaces in Connection with GUI Components

A. One of the most common uses of interfaces is in connection with GUI components. In the Bruce book, the chapter on GUIs comes right after the chapter on Interfaces. Though we won't be looking at GUIs until after we've talked about loops (due to covering material needed for the next project), this is still an appropriate time to introduce this use of interfaces.

B. Several kinds of GUI components allow a user to request that some action be performed.

1. Buttons are an obvious example of this.

2. Menu items are another obvious example.

3. Text fields often allow the user to click return while typing in the text field to specify some action.

C. The package java.awt.event defines an interface known as ActionListener.

1. An object that implements this interface is one that carries out some action requested by the user performing one of these actions.

2. A GUI component that can be used to request actions is able to have one or more ActionListener objects registered with it. Since these actions can vary widely, implementation inheritance is out of the question. Rather, an interface is used to specify what behavior is needed.

3. This interface requires that the object have one method - actionPerformed. This method is called whenever the user requests an action, by clicking a button or menu item or pressing return in a text field, as the case may be.

   SHOW javadoc for java.awt.event.ActionListener