10 Object-Oriented Programming: Polymorphism, Solutions

One Ring to rule them all,
One Ring to find them,
One Ring to bring them all
and in the darkness bind them.
—John Ronald Reuel Tolkien

General propositions do not decide concrete cases.
—Oliver Wendell Holmes

A philosopher of imposing stature doesn't think in a vacuum. Even his most abstract ideas are, to some extent, conditioned by what is or is not known in the time when he lives.
—Alfred North Whitehead

Why art thou cast down, O my soul?
—Psalms 42:5

Objectives

In this chapter you'll learn:

- The concept of polymorphism.
- To use overridden methods to effect polymorphism.
- To distinguish between abstract and concrete classes.
- To declare abstract methods to create abstract classes.
- How polymorphism makes systems extensible and maintainable.
- To determine an object's type at execution time.
- To declare and implement interfaces.

Self-Review Exercises

10.1 Fill in the blanks in each of the following statements:

a) If a class contains at least one abstract method, it's a(n) ________ class.

ANS: abstract.

b) Classes from which objects can be instantiated are called ________ classes.

ANS: concrete.

c) ________ involves using a superclass variable to invoke methods on superclass and subclass objects, enabling you to "program in the general."

ANS: Polymorphism.

d) Methods that are not interface methods and that do not provide implementations must be declared using keyword ________.

ANS: `abstract`.

e) Casting a reference stored in a superclass variable to a subclass type is called ________.

ANS: downcasting.

10.2 State whether each of the statements that follows is *true* or *false*. If *false*, explain why.

a) All methods in an `abstract` class must be declared as `abstract` methods.

ANS: False. An abstract class can include methods with implementations and `abstract` methods.

b) Invoking a subclass-only method through a subclass variable is not allowed.

ANS: False. Trying to invoke a subclass-only method with a superclass variable is no allowed.

c) If a superclass declares an `abstract` method, a subclass must implement that method.

ANS: False. Only a concrete subclass must implement the method.

d) An object of a class that implements an interface may be thought of as an object of that interface type.

ANS: True.

Exercises

NOTE: Solutions to the programming exercises are located in the ch10solutions folder. Each exercise has its own folder named `ex10_##` where ## is a two-digit number representing the exercise number. For example, exercise 10.8's solution is located in the folder `ex10_08`.

10.3 How does polymorphism enable you to program "in the general" rather than "in the specific"? Discuss the key advantages of programming "in the general."

ANS: Polymorphism enables you to concentrate on the common operations that are applied to objects of all the classes in a hierarchy. The general processing capabilities can be separated from any code that is specific to each class. Those general portions of the code can accommodate new classes without modification. In some polymorphic applications, only the code that creates the objects needs to be modified to extend the system with new classes.

10.4 What are abstract methods? Describe the circumstances in which an abstract method would be appropriate.

ANS: An abstract method is one with keyword `abstract` in its declaration. Abstract methods do not provide implementations. Each concrete subclass of an abstract superclass must provide concrete implementations of the superclass's abstract methods. An abstract method is appropriate when it does not make sense to provide an implementation for a method in a superclass (i.e., some additional subclass-specific data is required to implement the method in a meaningful manner).

10.5 How does polymorphism promote extensibility?

ANS: Software that invokes polymorphic behavior is independent of the object types to which messages are sent as long as those types are in the same inheritance hierarchy. New object types that can respond to existing method calls can be incorporated into a system without requiring modification of the base system, except that client code that instantiates new objects must be modified to accommodate new types.

10.6 Discuss four ways in which you can assign superclass and subclass references to variables of superclass and subclass types.

ANS: 1) Assigning a superclass reference to a superclass variable. 2) Assigning a subclass reference to a subclass variable. 3) Assigning a subclass object's reference to a superclass variable is safe, because the subclass object *is an* object of its superclass. However, this reference can be used to refer only to superclass members. If this code refers to subclass-only members through the superclass variable, the compiler reports errors. 4) Attempting to assign a superclass object's reference to a subclass variable is a compilation error. To avoid this error, the superclass reference must be downcast to a subclass type explicitly. At execution time, if the object to which the reference refers is not a subclass object, an exception will occur. The `instanceof` operator can be used to ensure that such a cast is performed only if the object is a subclass object.

10.7 Compare and contrast abstract classes and interfaces. Why would you use an abstract class? Why would you use an interface?

ANS: An abstract class describes the general notion of what it means to be an object of that class. Abstract classes are incomplete—they normally contain data and one or more methods that are declared `abstract` because the methods cannot be implemented in a general sense. Abstract classes can contain implemented methods. Objects of an abstract class cannot be instantiated. Subclasses must declare the "missing pieces"—implementations of the abstract methods that are appropriate for each specific subclass. Abstract class references can refer to objects of any of the classes below the abstract class in an inheritance hierarchy and therefore can be used to process any such objects polymorphically. An interface also describes abstract functionality that can be implemented by objects of any number of classes. Classes that implement an interface can be unrelated, whereas concrete subclasses of the same abstract superclass are all related to one other by way of a shared superclass. An interface is often used when disparate (i.e., unrelated) classes need to provide common functionality (i.e., methods) or use common constants. An interface can also be used in place of an abstract class when there are no default implementation details (i.e., method implementations and instance variables) to inherit. When a class implements an interface, it establishes an *is-a* relationship with the interface type, just as a subclass participates in an *is-a* relationship with its superclass. Therefore, interface references can be used to evoke polymorphic behavior, just as an abstract superclass reference can.