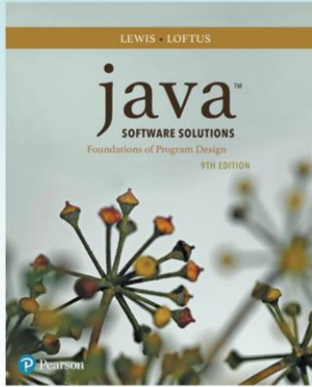


Chapter 7

Object-Oriented Design



Java Software Solutions
Foundations of Program Design
9th Edition

John Lewis
William Loftus

Copyright © 2017 Pearson Education, Inc.

Object-Oriented Design

- Now we can extend our discussion of the design of classes and objects
- Chapter 7 focuses on:
 - software development activities
 - determining the classes and objects that are needed for a program
 - the relationships that can exist among classes
 - the static modifier
 - writing interfaces
 - the design of enumerated type classes
 - method design and method overloading

Copyright © 2017 Pearson Education, Inc.

Outline

Software Development Activities

Identifying Classes and Objects

Static Variables and Methods

Class Relationships



Interfaces

Enumerated Types Revisited

Method Design

Testing

Interfaces

- A Java *interface* is a collection of abstract methods and constants
- An *abstract method* is a method header without a method body
- An abstract method can be declared using the modifier `abstract`, but because all methods in an interface are abstract, usually it is left off
- An interface is used to establish a set of methods that a class will implement

Copyright © 2017 Pearson Education, Inc.

-Note an abstract method contains the header (name, parameters) **without** any code (implementation)

```
abstract public void function( int );
```

-Note there are no braces containing statements, only the method header (name, parameters)

-An interface is used to describe functionality that classes might want to implement

-Classes that use an interface must define what this functionality is for their purpose

-They accomplish this by “filling in” the empty abstract methods with their own definitions

-In this way, we say that classes “implement” an interface

Interfaces

interface is a reserved word

None of the methods in
an interface are given
a definition (body)

```
public interface Doable
{
    public void doThis();
    public int doThat();
    public void doThis2 (double value, char ch);
    public boolean doTheOther (int num);
}
```

A semicolon immediately
follows each method header

Copyright © 2017 Pearson Education, Inc.

- Note how a Java interface is similar to a Java class
- The difference is that an interface can **only** contain constants and **abstract** methods
- It is not necessary to specify the abstract modifier on the methods (they are abstract by default)

Interfaces

- An interface cannot be instantiated
- Methods in an interface have public visibility by default
- A class formally implements an interface by:
 - stating so in the class header
 - providing implementations for every abstract method in the interface
- If a class declares that it implements an interface, it must define all methods in the interface

Copyright © 2017 Pearson Education, Inc.

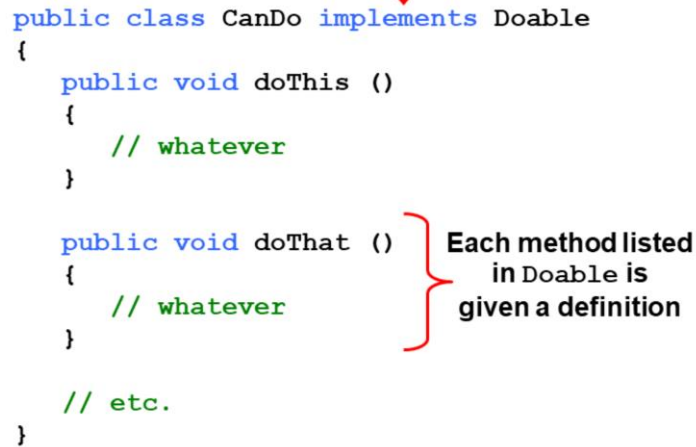
-Unlike classes, objects **cannot** be created from interfaces; in other words, we **cannot** specify:

```
Doable do = new Doable();
```

-Instead interfaces are **used by classes** to implement their designated functionality (methods)

Interfaces

implements is a
reserved word



```
public class CanDo implements Doable
{
    public void doThis ()
    {
        // whatever
    }

    public void doThat ()
    {
        // whatever
    }

    // etc.
}
```

Copyright © 2017 Pearson Education, Inc.

- This example shows how a class **implements** (or uses) an interface
- Note the use of the reserved word **implements** followed by the name of the interface
- The class must then provide definitions for **each** of the abstract methods declared in the interface

Interfaces

- In addition to (or instead of) abstract methods, an interface can contain constants
- When a class implements an interface, it gains access to all its constants
- A class that implements an interface can implement other methods as well
- **See** `Complexity.java`
- **See** `Question.java`
- **See** `MiniQuiz.java`

Copyright © 2017 Pearson Education, Inc.


```

//*****
// Complexity.java      Author: Lewis/Loftus
//
// Represents the interface for an object that can be assigned an
// explicit complexity.
//*****

public interface Complexity
{
    public void setComplexity (int complexity);
    public int getComplexity ();
}

```

```

//*****
// Question.java      Author: Lewis/Loftus
//
// Represents a question (and its answer).
//*****

public class Question implements Complexity
{
    private String question, answer;
    private int complexityLevel;

    //-----
    // Constructor: Sets up the question with a default complexity.
    //-----
    public Question (String query, String result)
    {
        question = query;
        answer = result;
        complexityLevel = 1;
    }
}

continue

```

continue

```
//-----  
// Sets the complexity level for this question.  
//-----  
public void setComplexity (int level)  
{  
    complexityLevel = level;  
}  
  
//-----  
// Returns the complexity level for this question.  
//-----  
public int getComplexity()  
{  
    return complexityLevel;  
}  
  
//-----  
// Returns the question.  
//-----  
public String getQuestion()  
{  
    return question;  
}
```

continue

Copyright © 2017 Pearson Education, Inc.

continue

```
//-----  
// Returns the answer to this question.  
//-----  
public String getAnswer()  
{  
    return answer;  
}  
  
//-----  
// Returns true if the candidate answer matches the answer.  
//-----  
public boolean answerCorrect (String candidateAnswer)  
{  
    return answer.equals(candidateAnswer);  
}  
  
//-----  
// Returns this question (and its answer) as a string.  
//-----  
public String toString()  
{  
    return question + "\n" + answer;  
}  
}
```

Copyright © 2017 Pearson Education, Inc.

```

//*****
// MiniQuiz.java      Author: Lewis/Loftus
//
// Demonstrates the use of a class that implements an interface.
//*****

import java.util.Scanner;

public class MiniQuiz
{
    //-----
    // Presents a short quiz.
    //-----
    public static void main (String[] args)
    {
        Question q1, q2;
        String possible;

        Scanner scan = new Scanner (System.in);

        q1 = new Question ("What is the capital of Jamaica?",
                           "Kingston");
        q1.setComplexity (4);

        q2 = new Question ("Which is worse, ignorance or apathy?",
                           "I don't know and I don't care");
        q2.setComplexity (10);

        continue

```

Inc.

continue

```
System.out.print (q1.getQuestion());  
System.out.println (" (Level: " + q1.getComplexity() + ")");  
possible = scan.nextLine();  
if (q1.answerCorrect(possible))  
    System.out.println ("Correct");  
else  
    System.out.println ("No, the answer is " + q1.getAnswer());  
  
System.out.println();  
System.out.print (q2.getQuestion());  
System.out.println (" (Level: " + q2.getComplexity() + ")");  
possible = scan.nextLine();  
if (q2.answerCorrect(possible))  
    System.out.println ("Correct");  
else  
    System.out.println ("No, the answer is " + q2.getAnswer());  
}  
}
```

Copyright © 2017 Pearson Education, Inc.

contin

Sample Run

What is the capital of Jamaica? (Level: 4)

Kingston

Correct

Which is worse, ignorance or apathy? (Level: 10)

apathy

No, the answer is I don't know and I don't care

```
System.out.println();  
System.out.print (q2.getQuestion());  
System.out.println (" (Level: " + q2.getComplexity() + ")");  
possible = scan.nextLine();  
if (q2.answerCorrect(possible))  
    System.out.println ("Correct");  
else  
    System.out.println ("No, the answer is " + q2.getAnswer());  
}  
}
```

Interfaces

- A class can implement multiple interfaces
- The interfaces are listed in the `implements` clause
- The class must implement all methods in all interfaces listed in the header

```
class ManyThings implements interface1, interface2
{
    // all methods of both interfaces
}
```

Copyright © 2017 Pearson Education, Inc.

Interfaces

- The Java API contains many helpful interfaces
- The `Comparable` interface contains one abstract method called `compareTo`, which is used to compare two objects
- We discussed the `compareTo` method of the `String` class in Chapter 5
- The `String` class implements `Comparable`, giving us the ability to put strings in lexicographic order

Copyright © 2017 Pearson Education, Inc.

-The `Comparable` interface defines functionality to compare two objects (of the same type)

-It is meant to determine whether they are less than, equal to, or greater than one another

-What does that mean for an object of a specific type of class?

-One class might define this meaning to be different than another

-This is **exactly** why the interface doesn't define any methods for it, instead each class does!

-When a class implements this interface, it defines what it means to compare two objects of its class type

-Different classes provide different **meanings** (method definitions) for the same abstract method name!

-The object-oriented term for this important, fundamental concept is called **polymorphism**

The Comparable Interface

- Any class can implement `Comparable` to provide a mechanism for comparing objects of that type

```
if (obj1.compareTo(obj2) < 0)
    System.out.println ("obj1 is less than obj2");
```

- The value returned from `compareTo` should be negative if `obj1` is less than `obj2`, 0 if they are equal, and positive if `obj1` is greater than `obj2`
- It's up to the programmer to determine what makes one object less than another

Copyright © 2017 Pearson Education, Inc.

The Iterator Interface

- As we discussed in Chapter 5, an iterator is an object that provides a means of processing a collection of objects one at a time
- An iterator is created formally by implementing the `Iterator` interface, which contains three methods
 - The `hasNext` method returns a boolean result – true if there are items left to process
 - The `next` method returns the next object in the iteration
 - The `remove` method removes the object most recently returned by the `next` method

Copyright © 2017 Pearson Education, Inc.

The Iterable Interface

- Another interface, `Iterable`, establishes that an object provides an iterator
- The `Iterable` interface has one method, `iterator`, that returns an `Iterator` object
- Any `Iterable` object can be processed using the for-each version of the `for` loop
- Note the difference: an `Iterator` has methods that perform an iteration; an `Iterable` object provides an iterator on request

Copyright © 2017 Pearson Education, Inc.

- The `Iterable` interface is primarily used with an advanced topic known as Java collections
- The `Iterable` interface also uses the advanced concept of generics that we'll study later
- For now, just know that it is simply another type of interface in the Java API

Interfaces

- You could write a class that implements certain methods (such as `compareTo`) without formally implementing the interface (`Comparable`)
- However, formally establishing the relationship between a class and an interface allows Java to deal with an object in certain ways
- Interfaces are a key aspect of object-oriented design in Java
- We discuss this idea further in Chapter 10

Copyright © 2017 Pearson Education, Inc.

-By creating an interface, you define functionality that can be shared and defined differently among classes

-For example, consider designing an interface to operate a vehicle

-The abstract methods might be named “forward, reverse, start, stop”

-Now consider classes that might want to implement such functionality (e.g. car, boat, plane, bike)

-Each class implementing such an interface would perform “forward, reverse, start, stop” differently!