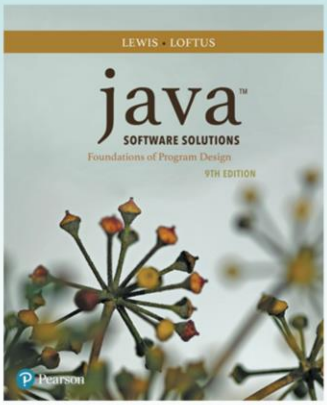
Chapter 9
Inheritance



Java Software Solutions
Foundations of Program Design
9th Edition

John Lewis
William Loftus

Inheritance

- Inheritance is a fundamental object-oriented design technique used to create and organize reusable classes
- Chapter 9 focuses on:
  - deriving new classes from existing classes
  - the `protected` modifier
  - creating class hierarchies
  - abstract classes
  - indirect visibility of inherited members
  - designing for inheritance

Outline

- Creating Subclasses
- Overriding Methods
- Class Hierarchies
- Visibility
- Designing for Inheritance

Overriding Methods

- A child class can *override* the definition of an inherited method in favor of its own
- The new method must have the same signature as the parent's method, but can have a different body
- The type of the object executing the method determines which version of the method is invoked
- See `Messages.java`
- See `Thought.java`
- See `Advice.java`

-A child class can "re-define", or **override** a method of a parent class

-It does so by implementing a method with the same **name** and **signature** as the parent

-Since the method signature is the same in both classes, there could be confusion as to which one is called

-Consider an object of the child class calling this method; which one should be executed (the parent's or the child's)?

-Note the answer in the 3rd bullet above, the **type** of the object determines which version is executed

-When an object of the **parent** calls the method, the method from the **parent** class is invoked

-When an object of the **child** calls the method, the method from the **child** class is invoked

-Overriding allows classes to implement different meanings (definitions) to the same method name (signature)

-In other words, different classes can respond differently to the same behavior!

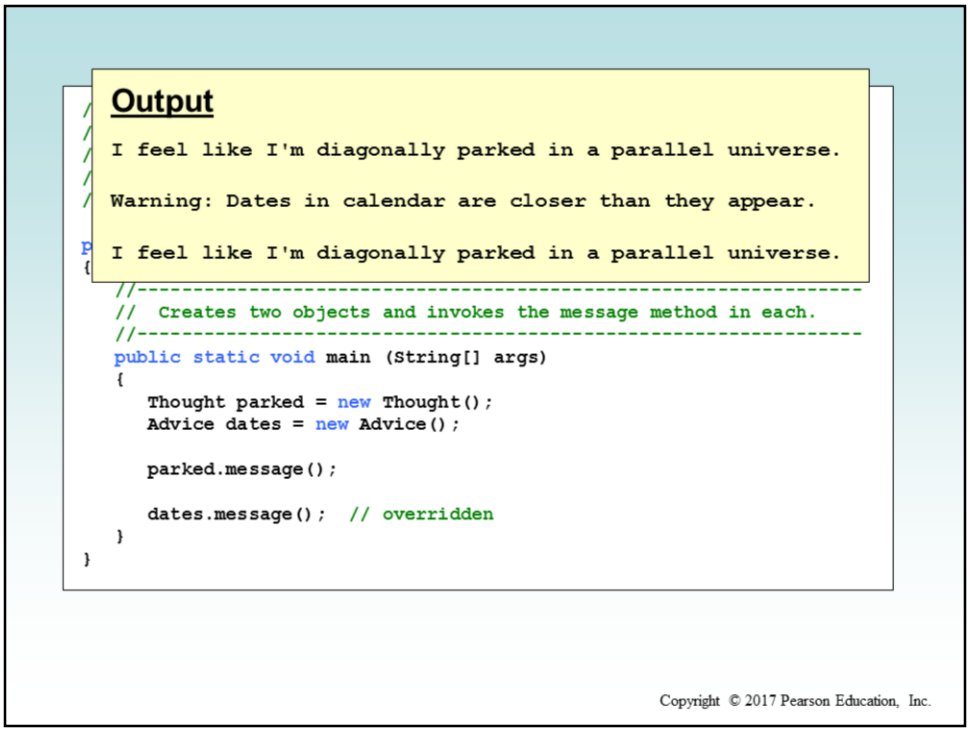
-This is a fundamental concept in object-oriented design called **polymorphism**

```
//********************************************************************
//  Messages.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an overridden method.
//********************************************************************

public class Messages
{
   //-----------------------------------------------------------------
   //  Creates two objects and invokes the message method in each.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Thought parked = new Thought();
      Advice dates = new Advice();

      parked.message();

      dates.message();  // overridden
   }
}
```



-Note that the message method from the Thought class is invoked when called from a Thought object (parked):

parked.message();

-Note that the message method from the Advice class is invoked when called from an Advice object (dates):

dates.message();

-Even though Advice is derived from Thought, the method from Advice executes when called from an Advice object!

```
//********************************************************************
//  Thought.java       Author: Lewis/Loftus
//
//  Represents a stray thought. Used as the parent of a derived
//  class to demonstrate the use of an overridden method.
//********************************************************************

public class Thought
{
   //-----------------------------------------------------------------
   //  Prints a message.
   //-----------------------------------------------------------------
   public void message()
   {
      System.out.println ("I feel like I'm diagonally parked in a " +
                          "parallel universe.");

      System.out.println();
   }
}
```

```
//********************************************************************
//  Advice.java       Author: Lewis/Loftus
//
//  Represents some thoughtful advice. Used to demonstrate the use
//  of an overridden method.
//********************************************************************

public class Advice extends Thought
{
   //-----------------------------------------------------------------
   //  Prints a message. This method overrides the parent's version.
   //-----------------------------------------------------------------
   public void message()
   {
      System.out.println ("Warning: Dates in calendar are closer " +
                          "than they appear.");

      System.out.println();

      super.message();  // explicitly invokes the parent's version
   }
}
```


-Note another use of the **super** keyword to explicitly call a method from the parent class

Overriding

- A method in the parent class can be invoked explicitly using the `super` reference
- If a method is declared with the `final` modifier, it cannot be overridden
- The concept of overriding can be applied to data and is called *shadowing variables*
- Shadowing variables should be avoided because it tends to cause unnecessarily confusing code

-A “shadow variable” is when a child class re-declares a variable from the parent class

-In other words, a variable with the same name and type in the parent is declared again in the child class

-Due to the confusion this could cause, this is to be avoided

Overloading vs. Overriding

- Overloading deals with multiple methods with the same name in the same class, but with different signatures
- Overriding deals with two methods, one in a parent class and one in a child class, that have the same signature
- Overloading lets you define a similar operation in different ways for different parameters
- Overriding lets you define a similar operation in different ways for different object types

-Note the difference between overloading and overriding

-Overloading defines methods with the **same** name, **different** signatures in a **single** class

-Overriding defines methods with the **same** name, **same** signatures in a **parent and child** classes

Quick Check

True or False?

A child class may define a method with the same name as a method in the parent.

A child class can override the constructor of the parent class.

A child class cannot override a `final` method of the parent class.

It is considered poor design when a child class overrides a method from the parent.

A child class may define a variable with the same name as a variable in the parent.

Quick Check

True or False?

A child class may define a method with the same name as a method in the parent.	True
A child class can override the constructor of the parent class.	False
A child class cannot override a `final` method of the parent class.	True
It is considered poor design when a child class overrides a method from the parent.	False
A child class may define a variable with the same name as a variable in the parent.	True, but shouldn't