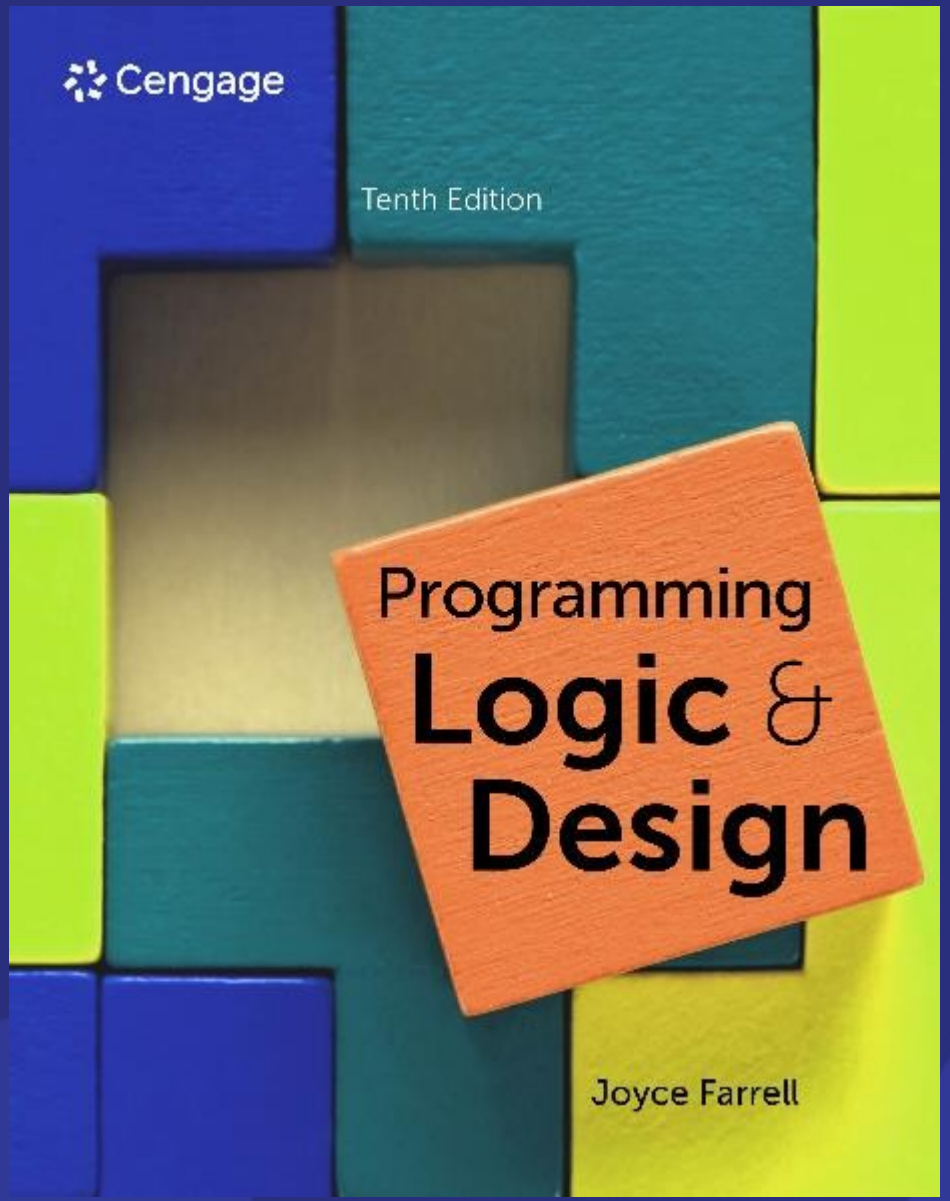


Programming Logic and Design, 10e

Chapter 10: Object-Oriented Programming

Chapter Objectives

When you complete this chapter, you will be able to:

- Describe the principles of object-oriented programming
- Create classes and class diagrams
- Use public and private access
- Organize classes
- Use instance methods
- Use static methods
- Use objects

Principles of Object-Oriented Programming (1 of 2)

- Object-oriented programming (OOP): Programming model that focuses on an application's components and data and the methods the components use
  - Uses all of the familiar concepts from modular procedural programming
    - Variables, methods, and passing arguments
    - Sequence, selection, looping structures, and arrays
  - Adds several new concepts to programming and requires to learn new vocabulary to describe those concepts

Principles of Object-Oriented Programming (2 of 2)

- Important features of object-oriented languages
  - Classes
  - Objects
  - Polymorphism
  - Inheritance
  - Encapsulation

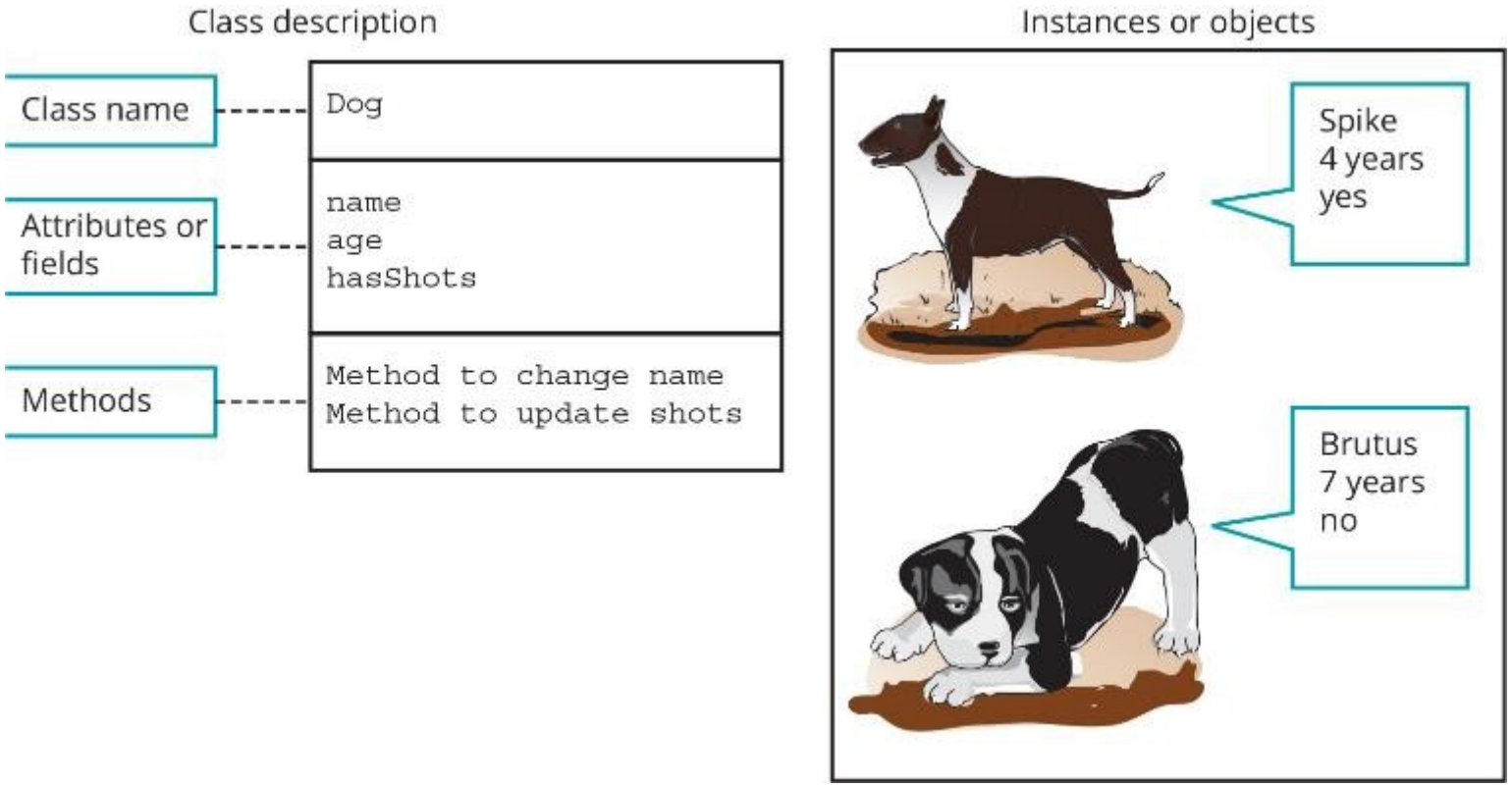
Classes and Objects (1 of 4)

- **Class**: Describes a group or collection of objects with common attributes
- **Object**: One **instance** or one **instantiation** of a class
  - When a program creates an object, it **instantiates** the object
- Example
  - Class name: dog
  - Attributes: name, age, hasShots
  - Methods: Methods to change name and update shots

Classes and Objects (2 of 4)

- **Attributes**: Characteristic of an object
  - Example: Automobile's attributes are its make, model, year, and purchase price
- Methods: Actions that can be taken on an object
  - Alter, use, or retrieve the attributes
  - Example
    - Methods for changing and viewing an automobile's speed

Figure 10-1: A Dog Class and Two Instances



Classes and Objects (3 of 4)

- Thinking in an object-oriented manner
  - Everything is an object
  - Every object is a member of a class
- **Is-a relationship**: “My oak desk with the scratch on top is a Desk”
- Concept of a class is useful because of its reusability
- Class’s **instance variables**: Data components of a class that belong to every instantiated object
  - Often called fields

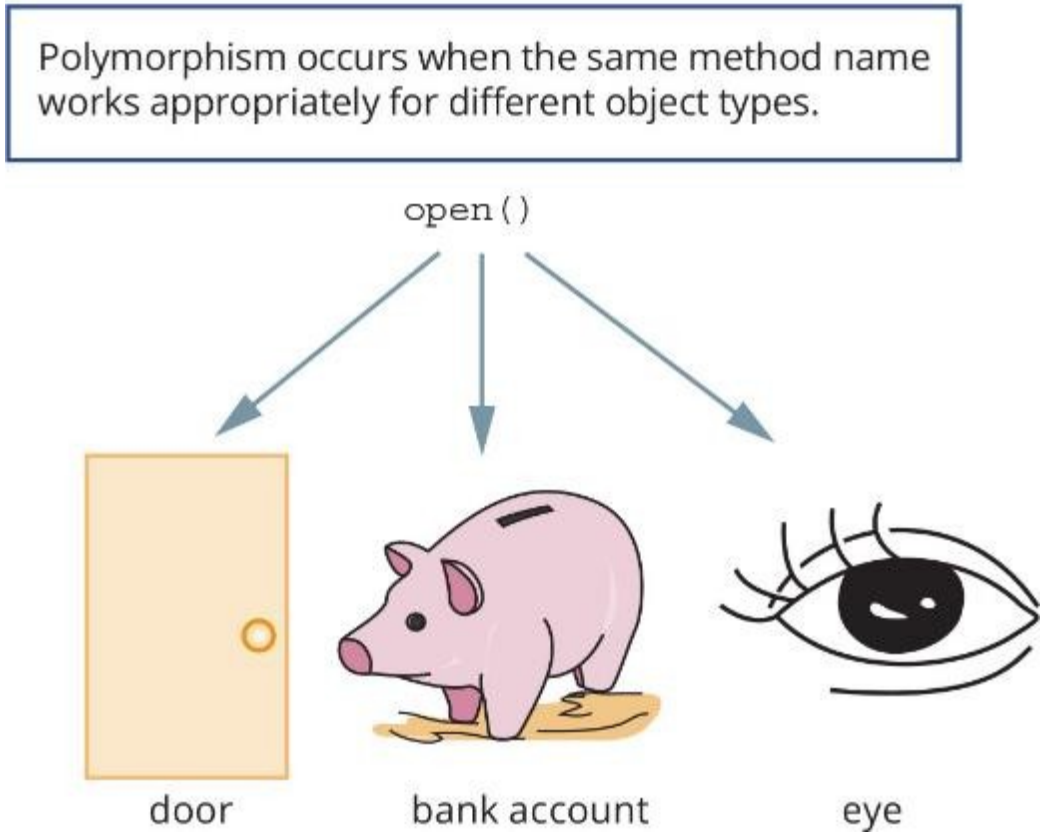
Classes and Objects (4 of 4)

- **State**: Set of all the values or contents of a class object's instance variables
- Every object instantiated from a given class possesses the same methods
- Classes can be created from which objects will be instantiated
- **Class client** or **class user**: Program or class that instantiates objects of another prewritten class

Polymorphism

- The real world is full of objects
  - Door is an object that needs to be opened and closed
  - But an open procedure works differently on different objects
    - Open a door, a drawer, a bank account, a computer file, or one’s eyes
- The term refers to the ability to use a single name to communicate multiple meanings

Figure 10-2: Examples of Polymorphism



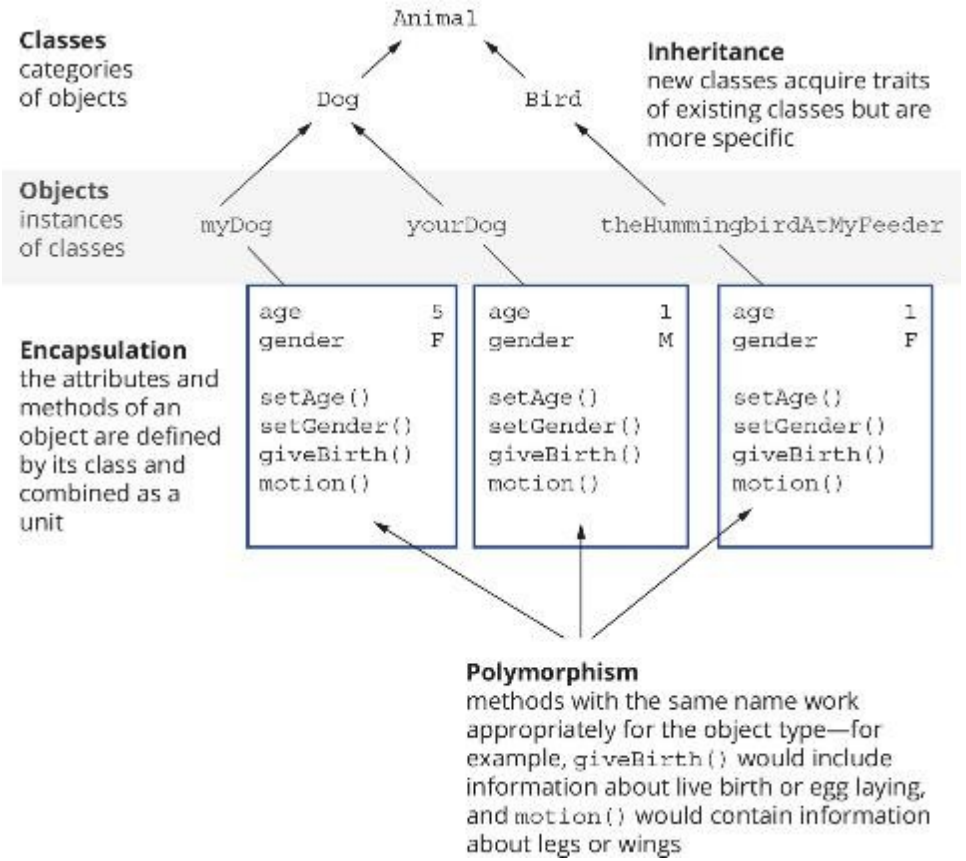
Inheritance

- **Inheritance**: Process of acquiring the traits of one's predecessors
  - Example: A new door with a stained glass window inherits its attributes (doorknob and hinges) and methods (opening and closing) of a door
- Once programmers create a class, they can develop:
  - New classes whose objects possess all the traits of objects of the original class
  - Any new traits the new class needs

Encapsulation

- **Encapsulation**: Process of combining all of an object's attributes and methods into a single package
- **Information hiding**, or **data hiding**
  - Other classes should not alter an object's attributes
    - Outside classes should only be allowed to make a request that an attribute be altered
    - It is up to the class's methods to determine whether the request is appropriate

Figure 10-4: Components of Object-oriented Programming



Creating Classes and Class Diagrams (1 of 2)

- **Class definition**
  - Set of program statements that lists the characteristics of the class's objects and the methods that can be applied to its objects
  - Contains three parts
    - Every class has a name
    - Most classes contain data, although this is not required
    - Most classes contain methods, although this is not required

Creating Classes and Class Diagrams (2 of 2)

- Declaring a class does not create any actual objects
- After an object is instantiated, its methods can be accessed using an identifier, a dot, and a method call
  - myAssistant . setHourlyWage (26.75)
- Employee myAssistant
  - When declaring the myAssistant object, it contains all the data fields and has access to all methods contained within the class

Figure 10-5: Application That Declares and Uses an Employee Object

```
start
   Declarations
      Employee myAssistant
   myAssistant.setLastName("Reynolds")
   myAssistant.setHourlyWage(26.75)
   output "My assistant makes ",
      myAssistant.getHourlyWage(), " per hour"
stop
```

Defining Classes and Creating Class Diagrams

- Programmers call the classes they write **user-defined types**
  - More accurate term is **programmer-defined types**
  - OOP programmers call them **abstract data types** (**ADTs**)
- In older object-oriented programming languages, simple numbers and characters are said to be **primitive data types**
- Ability to use methods as a “black box” without knowing their contents is a feature of encapsulation

Creating Class Diagrams (1 of 3)

- **Class diagram**: Consists of a rectangle divided into three sections
  - Top contains the name of the class
  - Middle contains the names and data types of the attributes
  - Bottom contains the methods

Creating Class Diagrams (2 of 3)

Figure 10-6: Generic class diagram

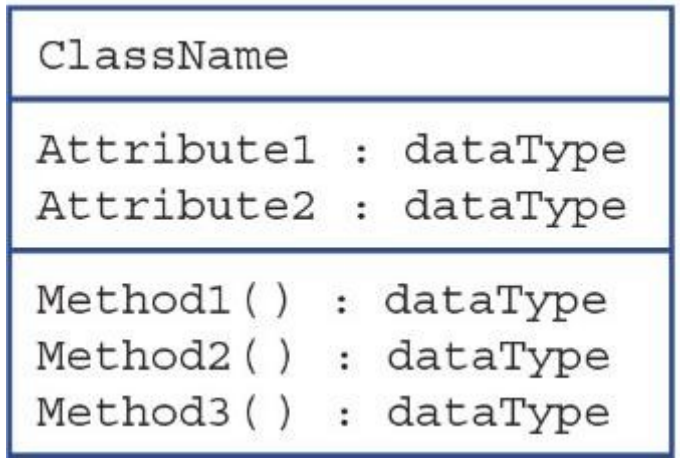


Figure 10-7: Employee class diagram

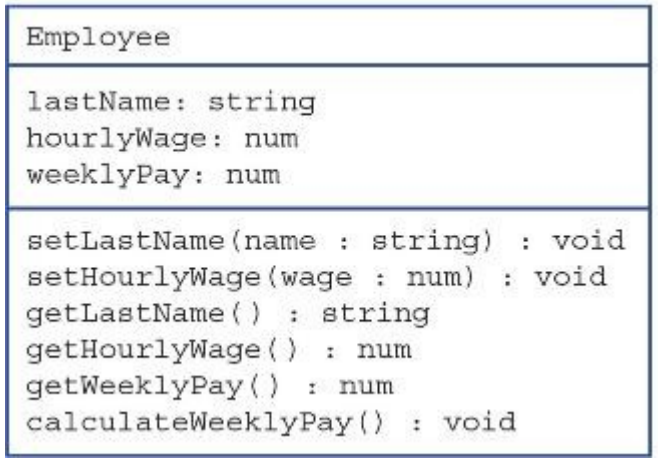


Figure 10-8: Pseudocode for Employee Class Described in Class Diagram in Figure 10-7

```
class Employee
   Declarations
      string lastName
      num hourlyWage
      num weeklyPay

   void setLastName(string name)
      lastName = name
   return

   void setHourlyWage(num wage)
      hourlyWage = wage
      calculateWeeklyPay()
   return

   string getLastName()
   return lastName

   num getHourlyWage()
   return hourlyWage

   num getWeeklyPay()
   return weeklyPay

   void calculateWeeklyPay()
      Declarations
         num WORK_WEEK_HOURS = 40
      weeklyPay = hourlyWage * WORK_WEEK_HOURS
   return
endClass
```

Creating Class Diagrams (3 of 3)

- Purpose of Employee class methods
  - Two of the methods accept values from a client and assign them to data fields
  - Three of the methods send data to a client
  - One method performs work within the class

Think, Pair, Share

- Suppose you have to create a class diagram for student details in a university. Take a few minutes to **think** about the name, attributes, and methods to use to create a solution.
- **Pair** up with a partner and discuss your thought process and approach to the problem. Ask each other questions and provide feedback on each other's solutions.
- **Share** your approach and solution with the rest of the class.

Set Methods, or Mutator Methods

- Sets or changes the values of data fields within the class

```
void setLastName (string name)
    lastName = name
return
myAssistant.setLastName("Johnson“)
```

- No requirement that such methods start with the set prefix
- Some languages allow to create a **property** to set field values instead of creating a set method

Figure 10-9: A Version of the setHourlyWage() Method Including Validation

```
void setHourlyWage(num wage)
   Declarations
      num MINWAGE = 20.00
      num MAXWAGE = 70.00
   if wage < MINWAGE then
      hourlyWage = MINWAGE
   else
      if wage > MAXWAGE then
         hourlyWage = MAXWAGE
      else
         hourlyWage = wage
      endif
   endif
   calculateWeeklyPay()
return
```

Get Methods, or Accessor Methods

- Purpose is to return a value from the class to a client

  string getLastName ()

  return lastName

- Value returned from a get method can be used as any other variable of its type would be used

Work Methods, Help Methods, or Facilitators (1 of 2)

- Performs tasks within a class

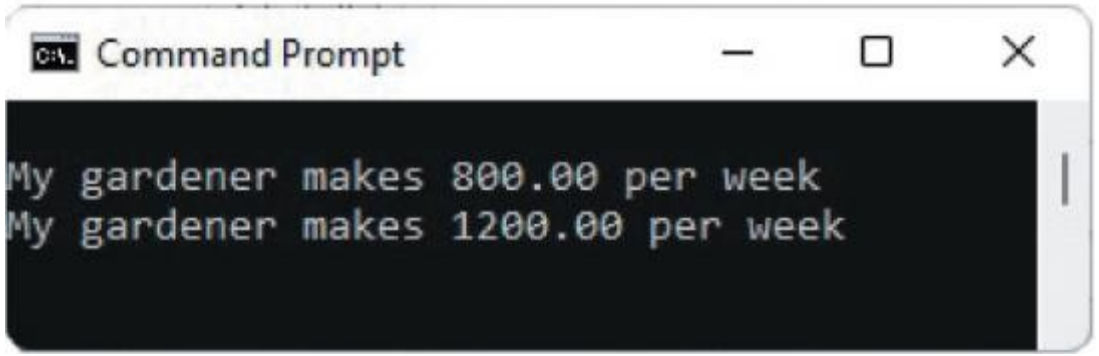
```
void calculateWeeklyPay ()
    Declarations
        num WORK_WEEK_HOURS = 40
    weeklyPay = hourlyWage * WORK_WEEK_HOURS
return
```

Work Methods, Help Methods, or Facilitators (2 of 2)

Figure 10-10: Program that sets and displays Employee data two times

```
start
   Declarations
      num LOW = 20.00
      num HIGH = 30.00
      Employee myGardener
   myGardener.setLastName("Greene")
   myGardener.setHourlyWage(LOW)
   output "My gardener makes ",
      myGardener.getWeeklyPay(), " per week"
   myGardener.setHourlyWage(HIGH)
   output "My gardener makes ",
      myGardener.getWeeklyPay(), " per week"
stop
```

Figure 10-11: Execution of program in Figure 10-10



Understanding Public and Private Access (1 of 3)

- Programmers do not want any outside programs or methods to alter their class's data fields unless they have control over the process
- To prevent unauthorized field modifications, programmers:
  - Force other programs and methods to use a method to alter data by using a method that is part of their class
  - Specify that data fields have **private access**
    - Data cannot be accessed by any method that is not part of the class

Understanding Public and Private Access (2 of 3)

- **Public access**
  - Other programs and methods can use the methods to get access to the private data
- **Access specifier**
  - Adjective defining the type of access (public or private) outside classes will have to the attribute or method

Figure 10-12: Employee class including public and private access specifiers

```
class Employee
   Declarations
      private string lastName
      private num hourlyWage
      private num weeklyPay

   public void setLastName(string name)
      lastName = name
   return

   public void setHourlyWage(num wage)
      hourlyWage = wage
      calculateWeeklyPay()
   return

   public string getLastName()
   return lastName

   public num getHourlyWage()
   return hourlyWage

   public num getWeeklyPay()
   return weeklyPay

   private void calculateWeeklyPay()
      Declarations
         num WORK_WEEK_HOURS = 40
      weeklyPay = hourlyWage * WORK_WEEK_HOURS
   return
endClass
```

Understanding Public and Private Access (3 of 3)

- Incorrect statement:

  myAssistant.hourlyWage = 25.00

- Correct statement:

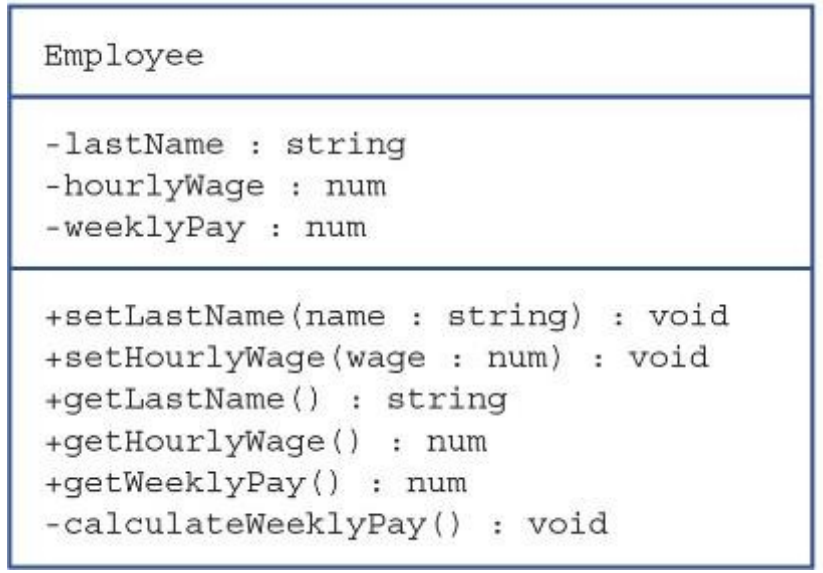
  myAssistant.setHourlyWage (25.00)

- Methods may be private; incorrect statement:

  myAssistant.calculateWeeklyPay ()

- Minus sign (–) precedes the private items; a plus sign (+) precedes public items

Figure 10-13: Employee class diagram with public and private access specifiers



Knowledge Check Activity

Identify a true statement about a class diagram.

a. An asterisk sign (*) precedes each item that is public.

b. A minus sign (−) precedes each item that is private.

c. A plus sign (+) precedes each item that is private.

d. An asterisk sign (*) precedes each item that is public.

Knowledge Check Activity: Answer

Identify a true statement about a class diagram.

Answer: b. A minus sign (−) precedes each item that is private.

In a class diagram, a minus sign (−) precedes each item that is private; a plus sign (+) precedes each item that is public.

Organizing Classes (1 of 2)

- Most programmers place data fields in some logical order at the beginning of a class
  - An ID number is most likely used as a unique identifier
  - Flexibility exists in positioning data fields within a class
- In some languages, data fields and methods can be organized in any order

Organizing Classes (2 of 2)

- Ways of ordering class methods
  - Storing in alphabetical order
  - Arranging in pairs of get and set methods
  - Arranging in the same order as the data fields are defined
  - Listing all accessor (get) methods together and all mutator (set) methods together
- If one's company distributes guidelines for organizing class components, one must follow those rules

Using Instance Methods (1 of 5)

Classes contain data and methods, and every instance of a class possesses the same data and has access to the same methods

Figure 10-14: Class diagram for Student class

Using Instance Methods (2 of 5)

Figure 10-15: Pseudocode for the Student class

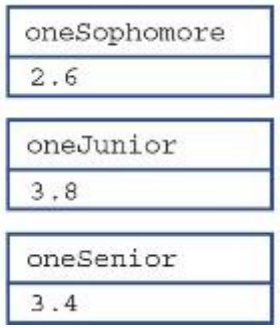
```
class Student
   Declarations
      private num gradePointAverage

   public void setGradePointAverage(num gpa)
      gradePointAverage = gpa
   return

   public num getGradePointAverage()
   return gradePointAverage
endClass
```

Figure 10-16: Program that creates three Student objects and picture of how they look in memory

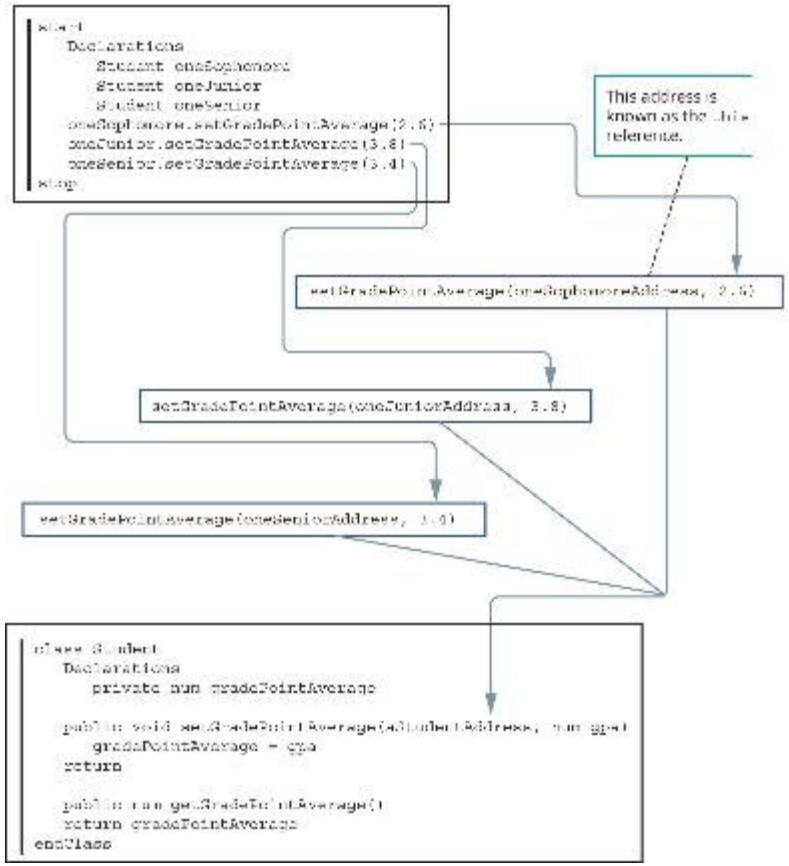
```
start
   Declarations
      Student oneSophomore
      Student oneJunior
      Student oneSenior
   oneSophomore.setGradePointAverage(2.6)
   oneJunior.setGradePointAverage(3.8)
   oneSenior.setGradePointAverage(3.4)
stop
```



Using Instance Methods (3 of 5)

- **Instance method**: Method that works appropriately with different objects
- Only one copy of each instance method is stored in memory
  - Program needs a way to determine whose gradePointAverage is being set or retrieved when one of the methods is called

Figure 10-17: How Student Object Memory Addresses Are Passed from an Application to an Instance Method of the Student Class



Using Instance Methods (4 of 5)

- **this reference**
  - Automatically created variable
  - Holds the address of an object that is passed to an instance method whenever the method is called
  - Refers to “this particular object” that is using the method at the moment
  - Implicitly passed as a parameter to each instance method

Using Instance Methods (5 of 5)

- Within an instance method, the following two identifiers mean exactly the same thing:
  - Any field name defined in the class
  - **this**, followed by a dot, followed by the same field name

Figure 10-18: Explicitly Using this in the Student Class

```
class Student
   Declarations
      private num gradePointAverage

   public void setGradePointAverage(num gpa)
      this.gradePointAverage = gpa
   return

   public num getGradePointAverage()
   return this.gradePointAverage
endClass
```

You can write `this` as a reference in these locations.

Using Static Methods

- Some methods do not require a this reference
  - displayStudentMotto () is a static method instead of an instance method
- Two types of methods
  - **Static methods,** or **class methods**: Methods for which no object needs to exist
  - **Nonstatic methods**: Methods that exist to be used with an object

```
Student.displayStudentMotto ()
```

Figure 10-19: Student Class displayStudentMotto() Method

```
public static void displayStudentMotto()
   output "Every student is an individual"
   output "in the pursuit of knowledge."
   output "Every student strives to be"
   output "a literate, responsible citizen."
return
```

Discussion Activity

Compare and contrast instance, static, and nonstatic methods.

Discussion Activity: Answer

An instance method operates correctly yet differently for every object instantiated from a class. When an instance method is called, a **this** reference that holds the object's memory address is automatically and implicitly passed to the method.

Static method, or class method, does not receive a **this** reference as an implicit parameter. Nonstatic method, which is an instance method, receives a **this** reference implicitly.

Using Objects

- Class instances can be used as items of simpler data types
- Example:
  - Passing an object to a method
  - Returning an object from a method
  - Using an array of objects

Figure 10-20: InventoryItem Class

```
class InventoryItem
   Declarations
      private string inventoryNumber
      private string description
      private num price

   public void setInventoryNumber(string number)
      inventoryNumber = number
   return

   public void setDescription(string description)
      this.description = description
   return

   public void setPrice(num price)
      if(price < 0)
         this.price = 0
      else
         this.price = price
      endif
   return

   public string getInventoryNumber()
   return inventoryNumber

   public string getDescription()
   return description

   public num getPrice()
   return price

endClass
```

Notice the uses of the this reference to differentiate between the method parameter and the class field.

Passing an Object to a Method

Figure 10-21: Application that declares and uses an InventoryItem object

```
start
   Declarations
      InventoryItem oneItem
   oneItem.setInventoryNumber("1276")
   oneItem.setDescription("Mahogany chest")
   oneItem.setPrice(450.00)
   displayItem(oneItem)
stop

public static void displayItem(InventoryItem item)
   Declarations
      num TAX_RATE = 0.06
      num tax
      num pr
      num total
   output "Item #", item.getInventoryNumber()
   output item.getDescription()
   pr = item.getPrice()
   tax = pr * TAX_RATE
   total = pr + tax
   output "Price is $", pr, " plus $", tax, " tax"
   output "Total is $", total
return
```

Figure 10-22: Output of the program in Figure 10-21

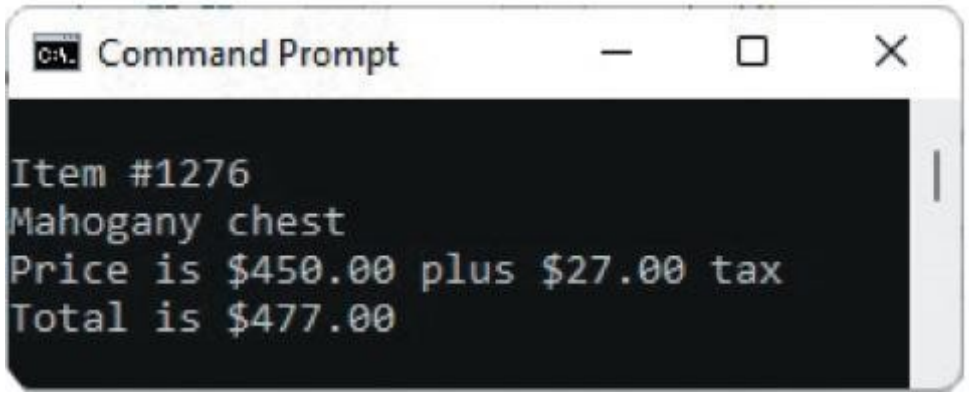


Figure 10-23: Application That Uses InventoryItem Objects

```
start
   Declarations
      InventoryItem oneItem
      string itemNum
      string QUIT = "0"
   output "Enter item number or ", QUIT, " to quit... "
   input itemNum
   while itemNum <> QUIT
      oneItem = getItemValues(itemNum)
      displayItem(oneItem)
      output "Enter next item number or ", QUIT, " to quit... "
      input itemNum
   endwhile
stop

public static InventoryItem getItemValues(string number)
   Declarations
      InventoryItem inItem
      string desc
      num price
   output "Enter description... "
   input desc
   output "Enter price... "
   input price
   inItem.setInventoryNumber(number)
   inItem.setDescription(desc)
   inItem.setPrice(price)
return inItem

public static void displayItem(InventoryItem item)
   Declarations
      num TAX_RATE = 0.06
      num tax
      num pr
      num total
   output "Item #", item.getInventoryNumber()
   output item.getDescription()
   pr = item.getPrice()
   tax = pr * TAX_RATE
   total = pr + tax
   output "Price is $", pr, " plus $", tax, " tax"
   output "Total is $", total
return
```

Figure 10-24: Typical Execution of Program in Figure 10-23

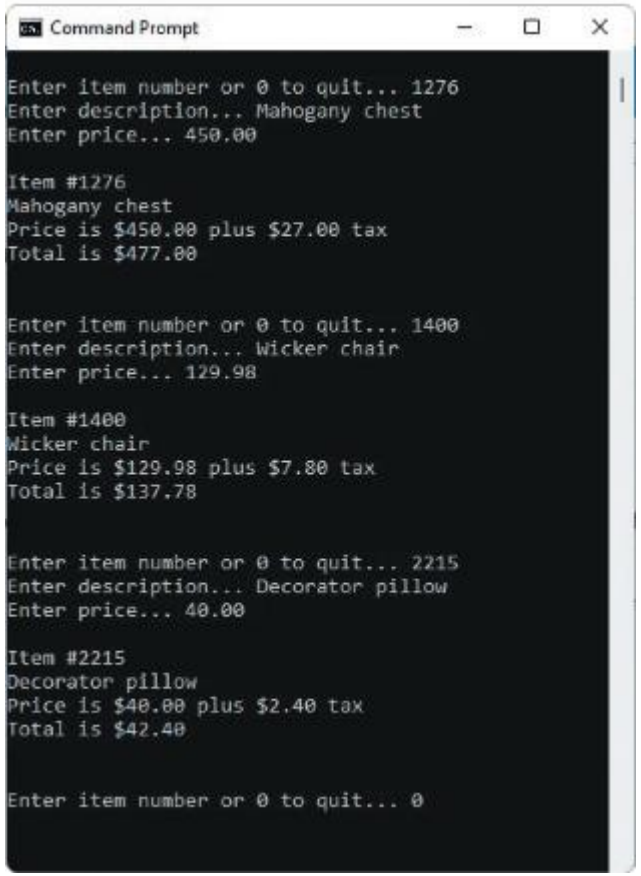
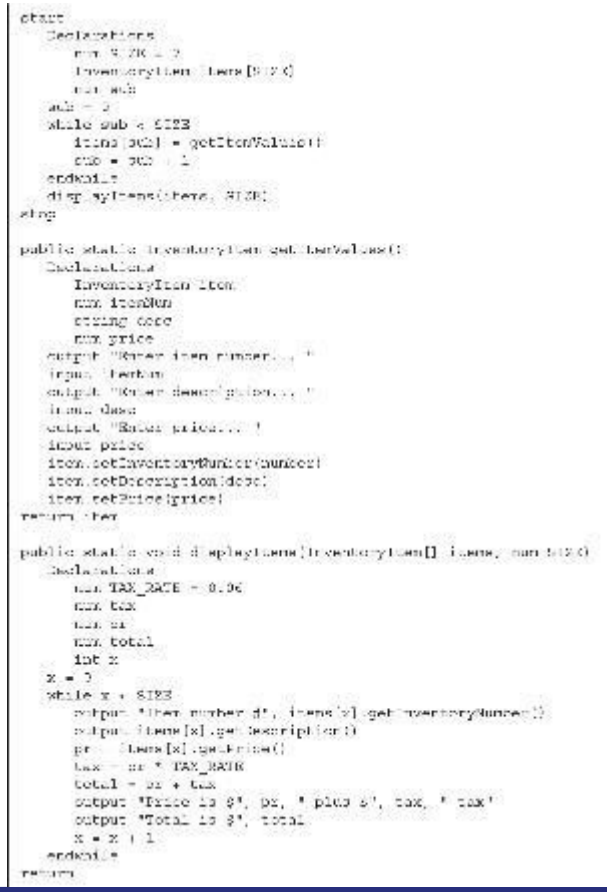


Figure 10-25: Application That Uses an Array of InventoryItem Objects



Self-Assessment

- What are the five features of object-oriented programming?
- How can one create a class and a class diagram in a program? Give an example.
- What is the difference between public and private access in object-oriented programming?
- How can one organize classes in a program?
- What is an instance method, and how is it different from a static method?
- What are the ways to use objects in a program?

Summary

Click the link to review the objectives for this presentation.

Link to Objectives