Object-Oriented Programming

Introduction

DMSI

Objectives

- After studying Chapter 13, you should be able to:
- Understand the principles of object-oriented programming
- Define classes
- Instantiate and use objects
- Understand polymorphism

Objectives (continued)

- **Understand constructor and destructor methods**
- **Use predefined classes to create GUI objects**
- **Understand the advantages of object-oriented programming**

An Overview of Object-Oriented Programming

- Object-oriented programming:
  - focuses on an application's data and the methods you need to manipulate that data
  - uses all of the concepts you are familiar with from modular procedural programming, such as
    - variables, modules, and passing values to modules

An Overview of Object-Oriented Programming (continued)

- With object-oriented programming:
  - You analyze the objects you are working with and the tasks that need to be performed with, and on, those objects
  - You pass messages to objects, requesting the objects to take action
  - The same message works differently (and appropriately) when applied to different objects

An Overview of Object-Oriented Programming (continued)

- A module or procedure can work appropriately with different types of data it receives, without the need to write separate modules
- Objects can share or inherit traits of objects that have already been created, reducing the time it takes to create new objects
- Encapsulation and information hiding are more complete than with the modules used in procedural programs

An Overview of Object-Oriented Programming (continued)

- focus on the objects that will be manipulated by the program
  - for example, a customer invoice, a loan application, or a menu from which the user will select an option
- can create multiple methods with the same name,
  - will act differently and appropriately when used with different types of objects

An Overview of Object-Oriented Programming (continued)

- Inheritance:
  - process of acquiring the traits of one’s predecessors
- Four concepts that are integral components of all object-oriented programming language are:
  - Classes
  - Objects
  - Inheritance
  - Polymorphism

Defining Classes

- Class:
  - category of things
- Object:
  - specific item that belongs to a class
  - is an instance of a class
- A class defines the characteristics of its objects and the methods that can be applied to its objects

Defining Classes (continued)

- A class 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
- You have worked with very similar constructs throughout this book
  - the name and data of a class constitute what procedural programming languages call a record

Defining Classes (continued)

- When working with classes, you call the data fields attributes
- Programmers often use a class diagram to illustrate class features
- A class diagram consists of a rectangle divided into three sections, as shown in Figure 13-1
- Figure 13-2 shows the class diagram for the `Employee` class

Defining Classes (continued)

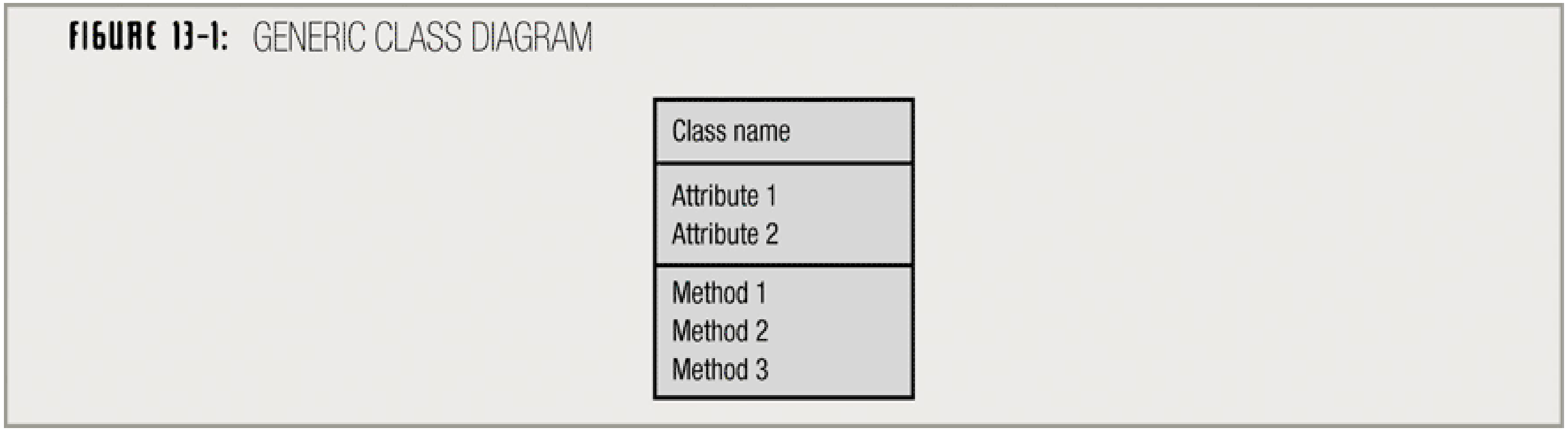


FIGURE 13-1: GENERIC CLASS DIAGRAM

Class name

Attribute 1
Attribute 2

Method 1
Method 2
Method 3

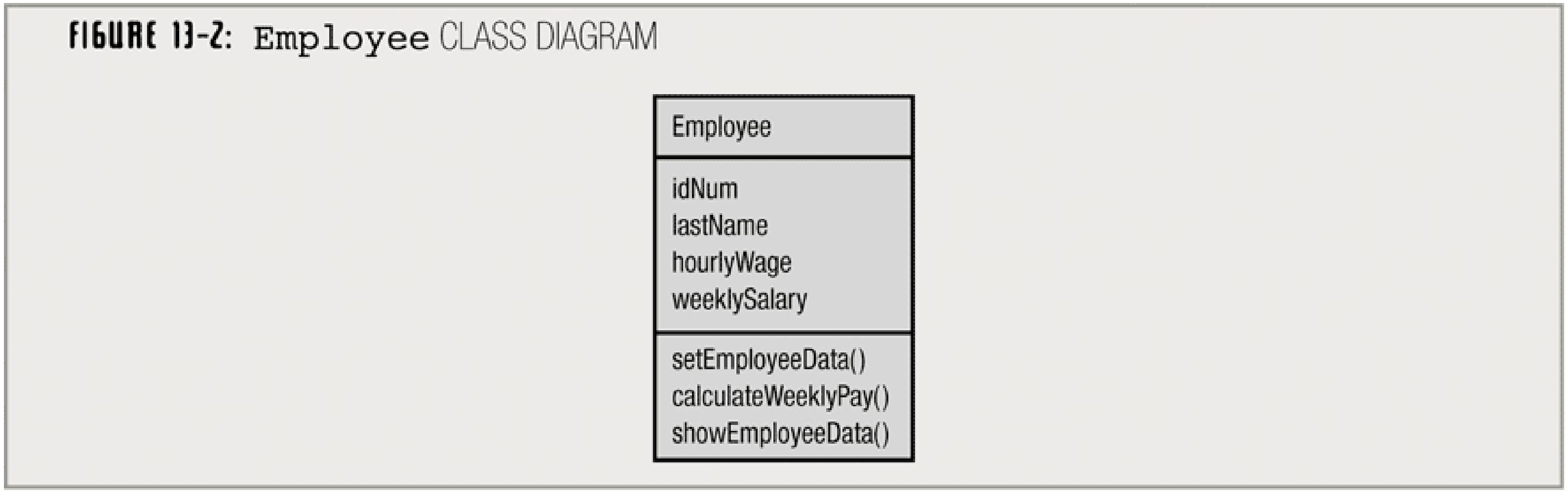


FIGURE 13-2: Employee CLASS DIAGRAM

Employee

idNum
lastName
hourlyWage
weeklySalary

setEmployeeData()
calculateWeeklyPay()
showEmployeeData()

Defining Classes (continued)

- Class diagram is an overview of class attributes and methods
- Object-oriented programmers usually specify that their data fields will have private access:
  - data cannot be accessed by any method that is not part of the class
- Methods themselves, like `setEmployeeData()`, support public access
  - other programs and methods may use the methods that control access to the private data

Defining Classes (continued)

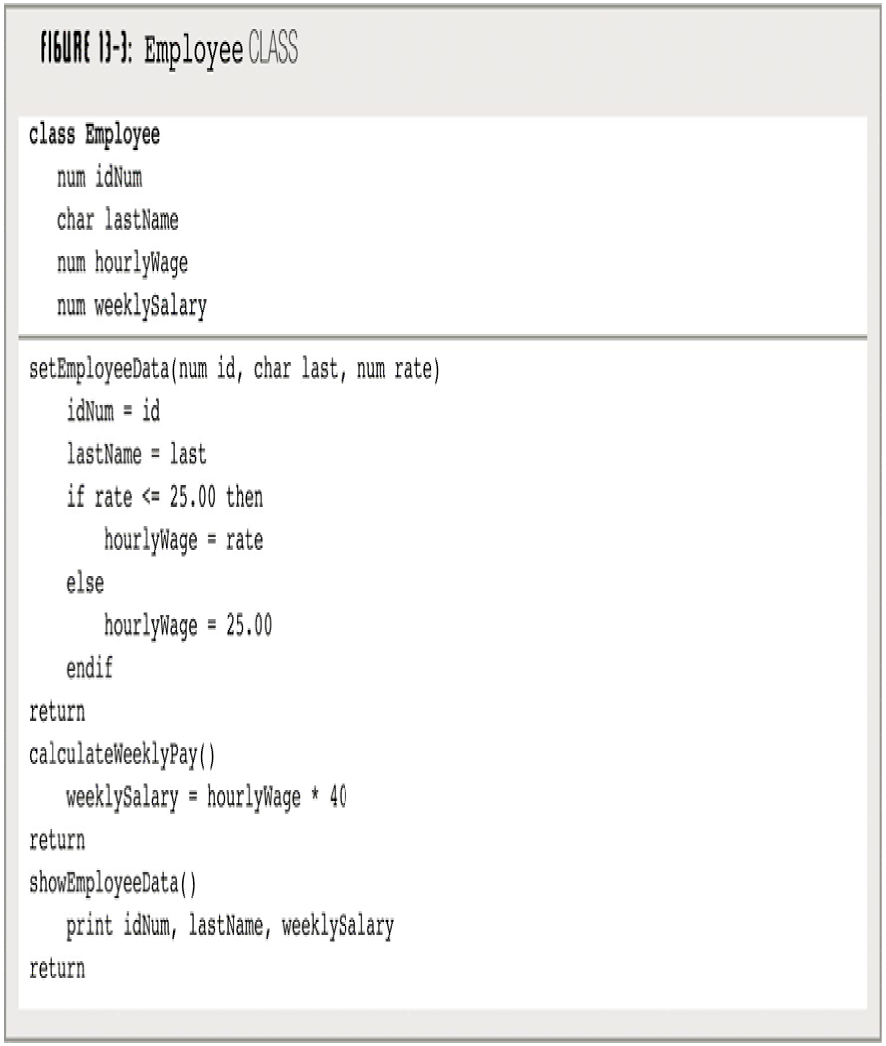


FIGURE 13-3: Employee CLASS

```
class Employee
   num idNum
   char lastName
   num hourlyWage
   num weeklySalary

setEmployeeData(num id, char last, num rate)
    idNum = id
    lastName = last
    if rate <= 25.00 then
        hourlyWage = rate
    else
        hourlyWage = 25.00
    endif
return
calculateWeeklyPay()
    weeklySalary = hourlyWage * 40
return
showEmployeeData()
    print idNum, lastName, weeklySalary
return
```

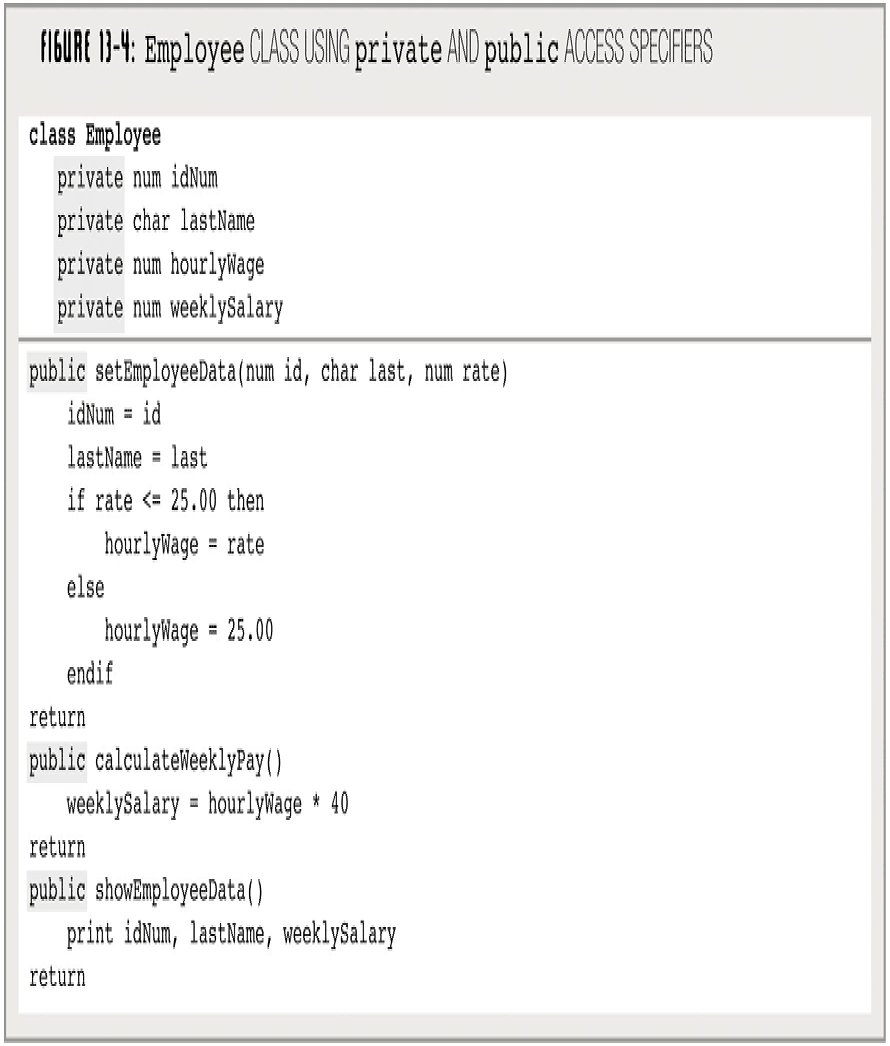


FIGURE 13-4: Employee CLASS USING private AND public ACCESS SPECIFIERS

```
class Employee
   private num idNum
   private char lastName
   private num hourlyWage
   private num weeklySalary

public setEmployeeData(num id, char last, num rate)
    idNum = id
    lastName = last
    if rate <= 25.00 then
        hourlyWage = rate
    else
        hourlyWage = 25.00
    endif
return
public calculateWeeklyPay()
    weeklySalary = hourlyWage * 40
return
public showEmployeeData()
    print idNum, lastName, weeklySalary
return
```

Instantiating and Using Objects

- When you write an object-oriented program,
  - you create objects that are members of a class, in the same way you create variables in procedural programs
- Instead of declaring a numeric variable named `money` with a statement that includes the type and identifying name such as `num money`, you
  - instantiate, or create, a class object with a statement that includes the type of object and an identifying name, such as Employee myAssistant

Instantiating and Using Objects (continued)

- For example, you can write a program such as the one shown in pseudocode in Figure 13-5

- A program that uses a class object is a client of the class

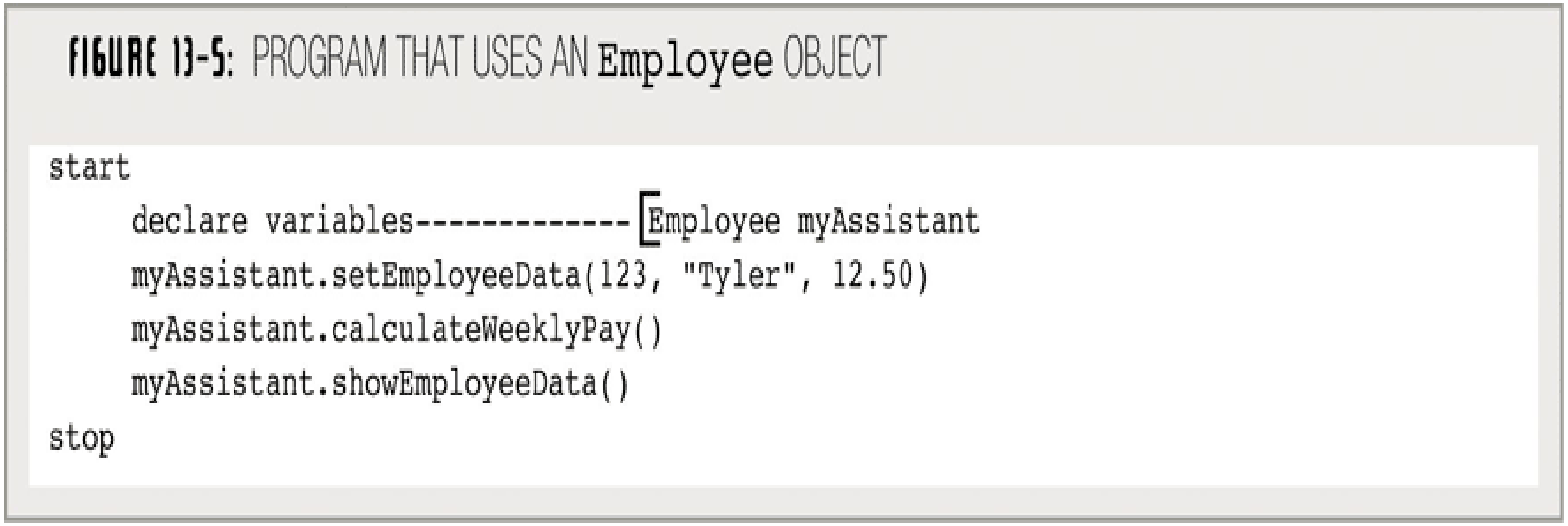


FIGURE 13-5: PROGRAM THAT USES AN Employee OBJECT

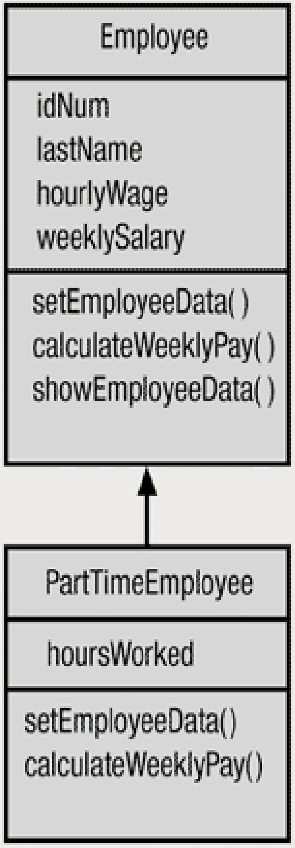
```
start
     declare variables-------------[Employee myAssistant
     myAssistant.setEmployeeData(123, "Tyler", 12.50)
     myAssistant.calculateWeeklyPay()
     myAssistant.showEmployeeData()
stop
```

Understanding Inheritance

- Descendent classes (or child classes):
  - can inherit all of the attributes of the original class (or parent class) OR
  - can override those attributes that are inappropriate
- When you create a child class, you can show its relationship to the parent with a class diagram like the one for `PartTimeEmployee` in Figure 13-6

PartTimeEmployee Class Diagram

FIGURE 13-6: PartTimeEmployee CLASS DIAGRAM



Employee

idNum
lastName
hourlyWage
weeklySalary

setEmployeeData( )
calculateWeeklyPay( )
showEmployeeData( )

PartTimeEmployee

hoursWorked

setEmployeeData()
calculateWeeklyPay()

Understanding Inheritance (continued)

- The complete `PartTimeEmployee` class appears in Figure 13-7

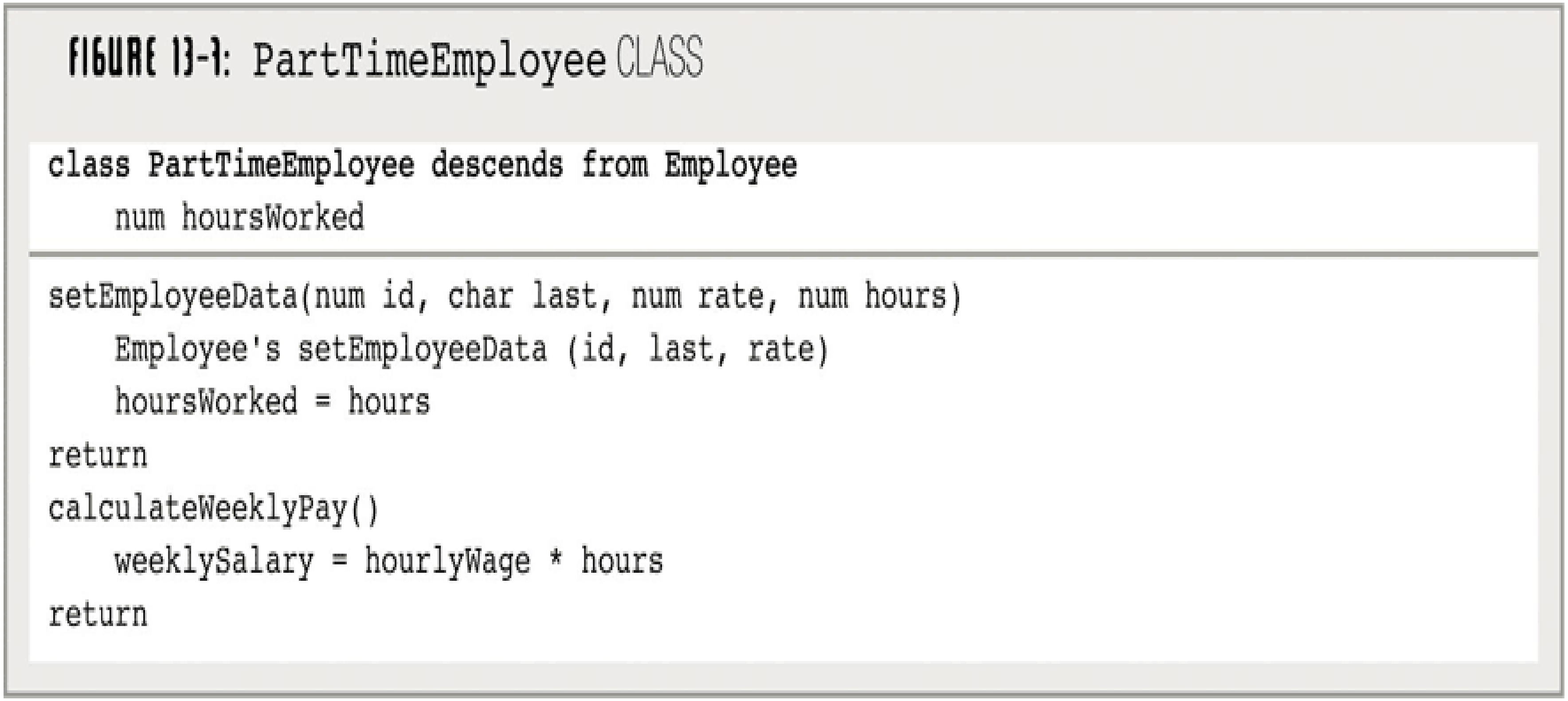


FIGURE 13-7: PartTimeEmployee CLASS

```
class PartTimeEmployee descends from Employee
    num hoursWorked
setEmployeeData(num id, char last, num rate, num hours)
    Employee's setEmployeeData (id, last, rate)
    hoursWorked = hours
return
calculateWeeklyPay()
    weeklySalary = hourlyWage * hours
return
```

Understanding Polymorphism (continued)

- Methods or functions need to operate differently, depending on context
- Object-oriented programs use polymorphism:
  - Allow the same request—that is, the same method call—to be carried out differently, depending on the context
  - Never allowed in non-object-oriented languages

Understanding Polymorphism (continued)

- Polymorphic method in object-oriented programming language can entail a lot of work
  - you must write each version of the method
- Benefit of polymorphism
  - can use methods in all sorts of applications
- Method overloading, closely related to polymorphism, occurs when different methods have the same name but different argument lists

Understanding Polymorphism (continued)

- Figure 13-9 shows an `Inventory` class that contains several versions of a `changeData()` method

- When you write a client program that uses this `Inventory` class to declare an `Inventory` item, and you use the `changeData()` method with it,
  - the computer determines which of the three available `changeData()` methods to call based on the arguments used with the method call

Inventory Class Containing Three Overloaded changeData() Methods

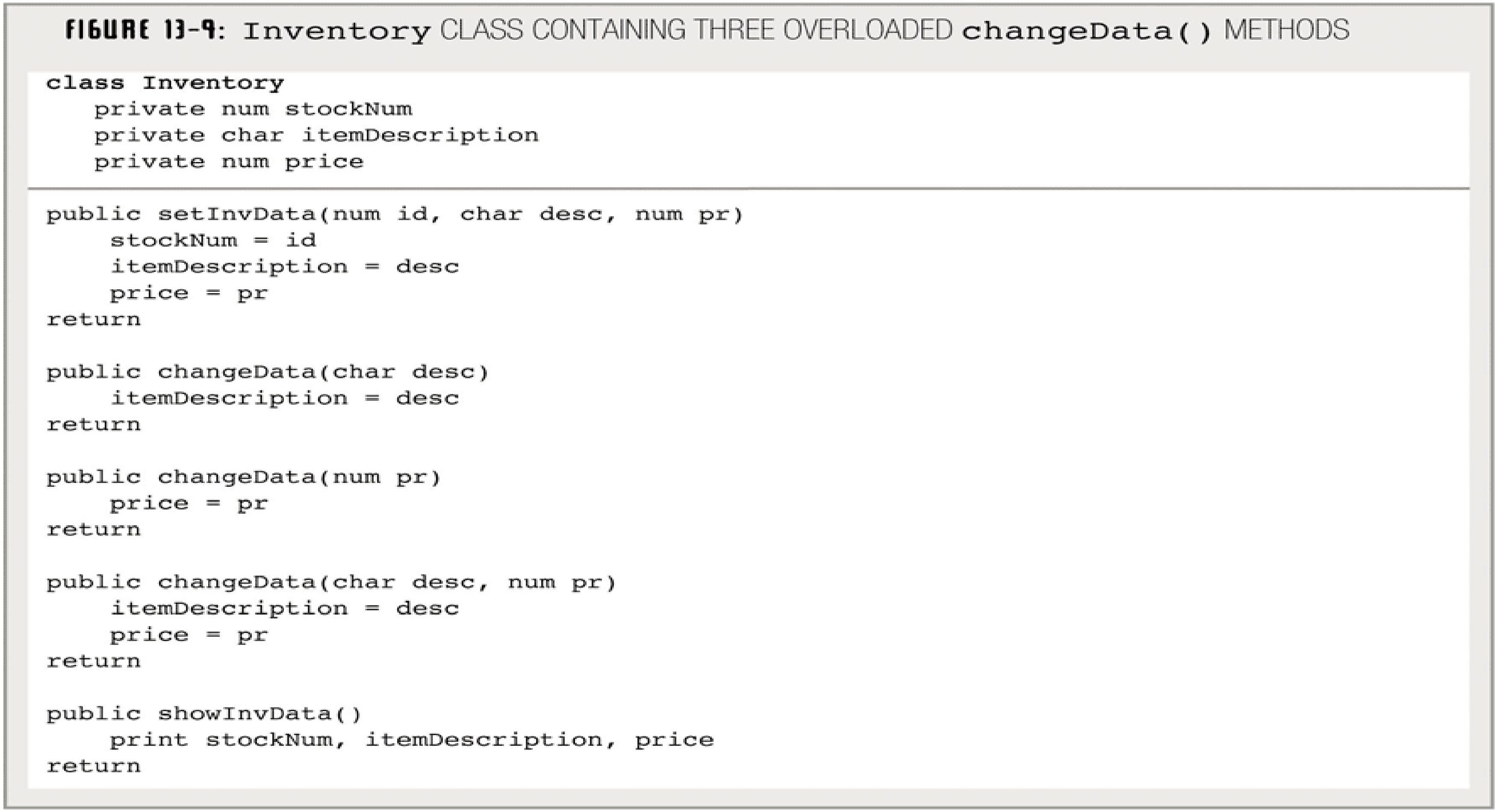


FIGURE 13-9: Inventory CLASS CONTAINING THREE OVERLOADED changeData() METHODS

```
class Inventory
   private num stockNum
   private char itemDescription
   private num price

public setInvData(num id, char desc, num pr)
    stockNum = id
    itemDescription = desc
    price = pr
return

public changeData(char desc)
    itemDescription = desc
return

public changeData(num pr)
    price = pr
return

public changeData(char desc, num pr)
    itemDescription = desc
    price = pr
return

public showInvData()
    print stockNum, itemDescription, price
return
```

Understanding Polymorphism (continued)

- When you execute the client program shown in Figure 13-10, declaring an `Inventory` object,
  - each of the three `changeData()` methods will be called one time, depending on the argument used
- When you read the program, it should seem clear in each instance whether the programmer intends to change the price, descriptions, or both

Understanding Polymorphism (continued)

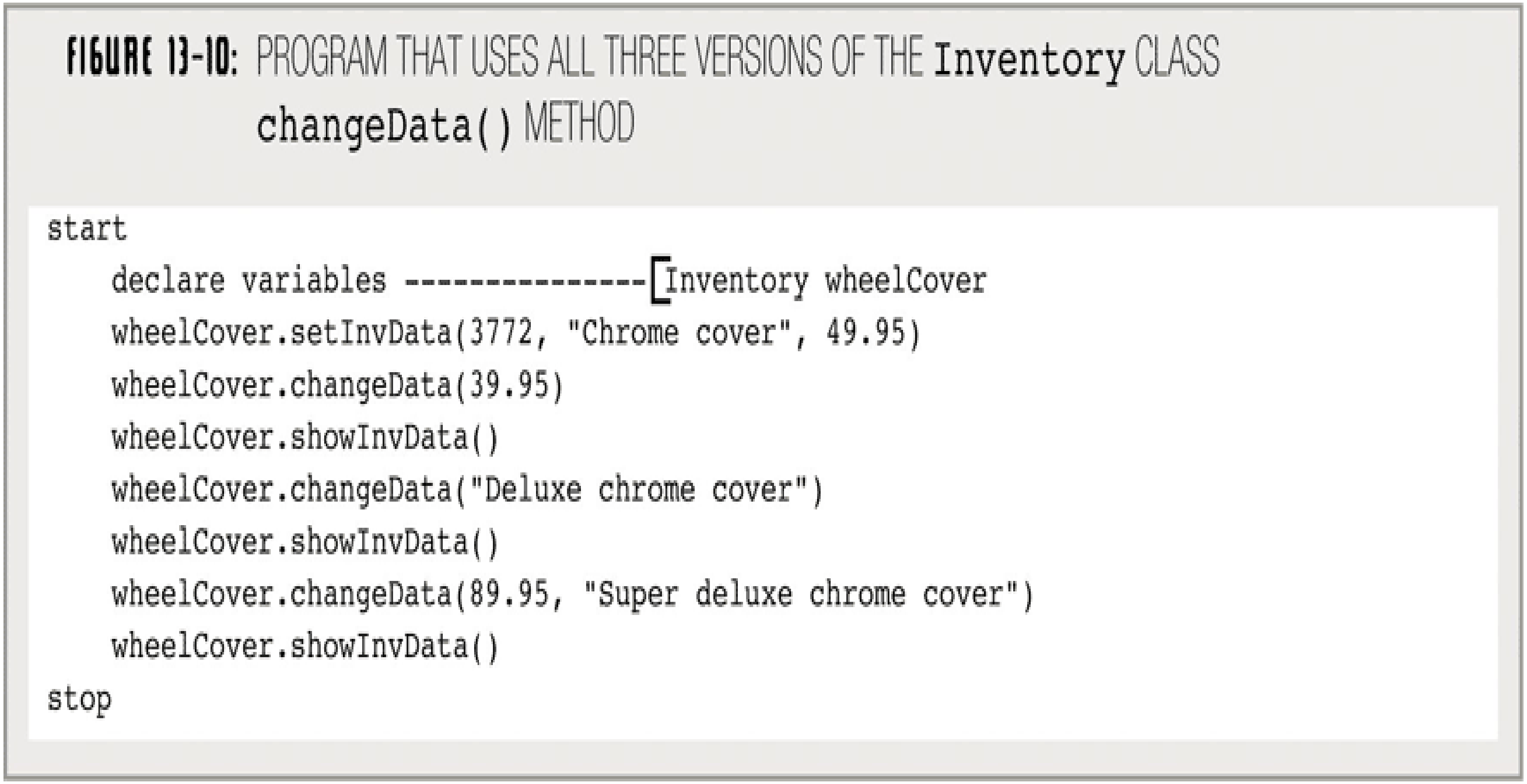


FIGURE 13-10: PROGRAM THAT USES ALL THREE VERSIONS OF THE `Inventory` CLASS `changeData()` METHOD

```
start
    declare variables ----------------[Inventory wheelCover
    wheelCover.setInvData(3772, "Chrome cover", 49.95)
    wheelCover.changeData(39.95)
    wheelCover.showInvData()
    wheelCover.changeData("Deluxe chrome cover")
    wheelCover.showInvData()
    wheelCover.changeData(89.95, "Super deluxe chrome cover")
    wheelCover.showInvData()
stop
```

Understanding Constructor and Destructor Methods

- When using an object-oriented programming language to instantiate an object with a statement like `Employee myAssistant`,
  - Actually calling a prewritten method with the name `Employee()`

Understanding Constructor and Destructor Methods (continued)

- A method with the same name as its class is a constructor method, or more simply, a constructor
  - Called automatically every time you instantiate an object that is a member of the class
  - Constructs, or creates, the object at a specific memory location
  - Provides initial values for the attributes contained within the object—usually 0 for numeric fields and an empty string containing no characters (also called a null string) for the character fields

Understanding Constructor and Destructor Methods (continued)

- When a programmer uses the `Inventory` class (figure 13-10) to create an `Inventory` object using a statement such as Inventory `someItem`,
  - the `someItem` object automatically has a `stockNum` of 999,
  - an `itemDescription` of “XXX”,
  - and a `price` of 0.00
- If programmers eventually construct thousands of items from the `Inventory` class, then each begins its existence with the same initial values

Inventory Class Containing a Constructor Method

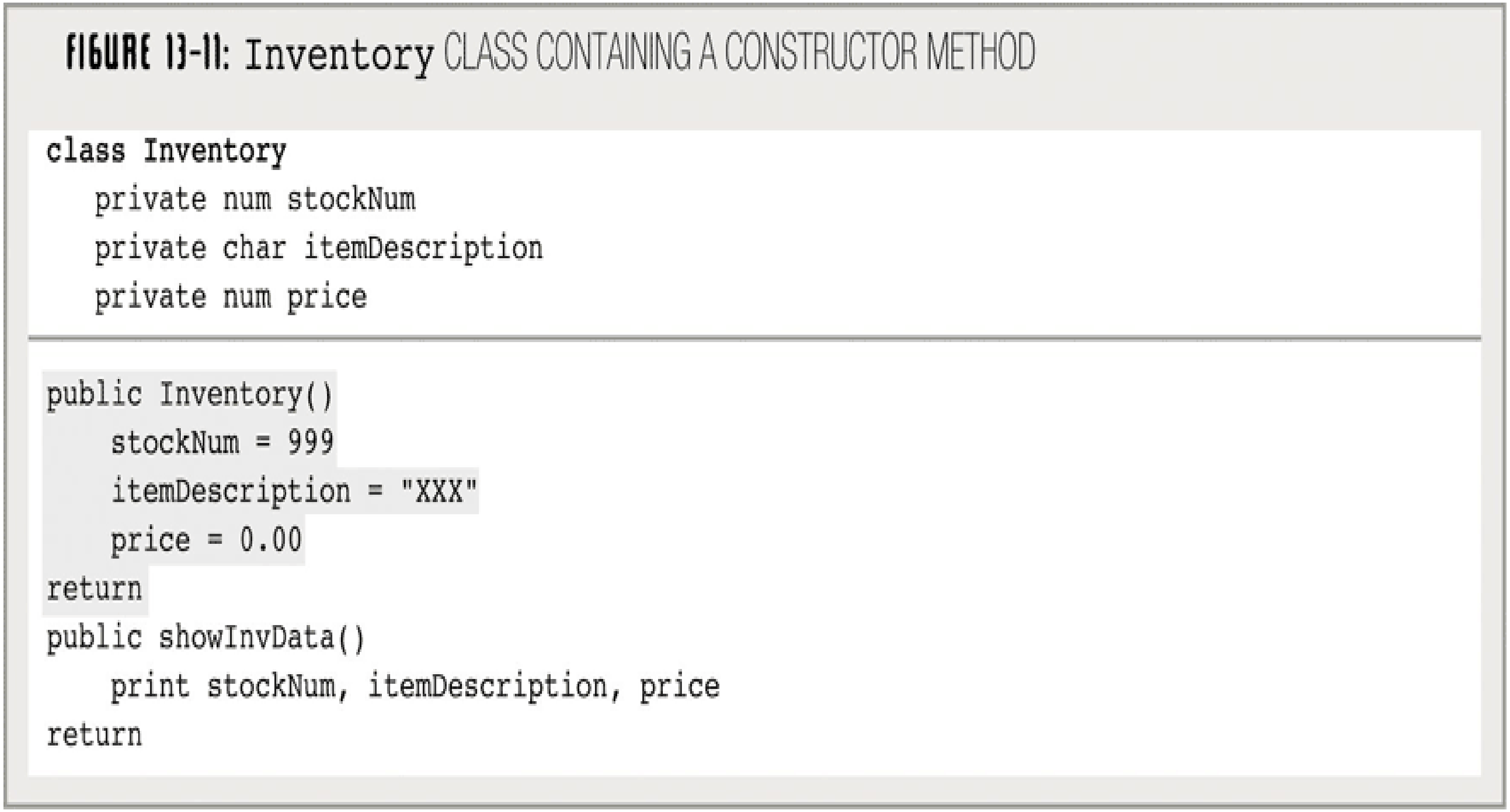


FIGURE 13-11: Inventory CLASS CONTAINING A CONSTRUCTOR METHOD

```
class Inventory
   private num stockNum
   private char itemDescription
   private num price

public Inventory()
    stockNum = 999
    itemDescription = "XXX"
    price = 0.00
return
public showInvData()
    print stockNum, itemDescription, price
return
```

Understanding Constructor and Destructor Methods (continued)

- Just as you can overload other class methods, you also can overload constructors
- For example, Figure 13-12 shows the `Inventory` class with two constructors
- One version, which takes no arguments, and is called the default constructor, sets an `Inventory` object's fields to 999, "XXX", and 0.00

Understanding Constructor and Destructor Methods (continued)

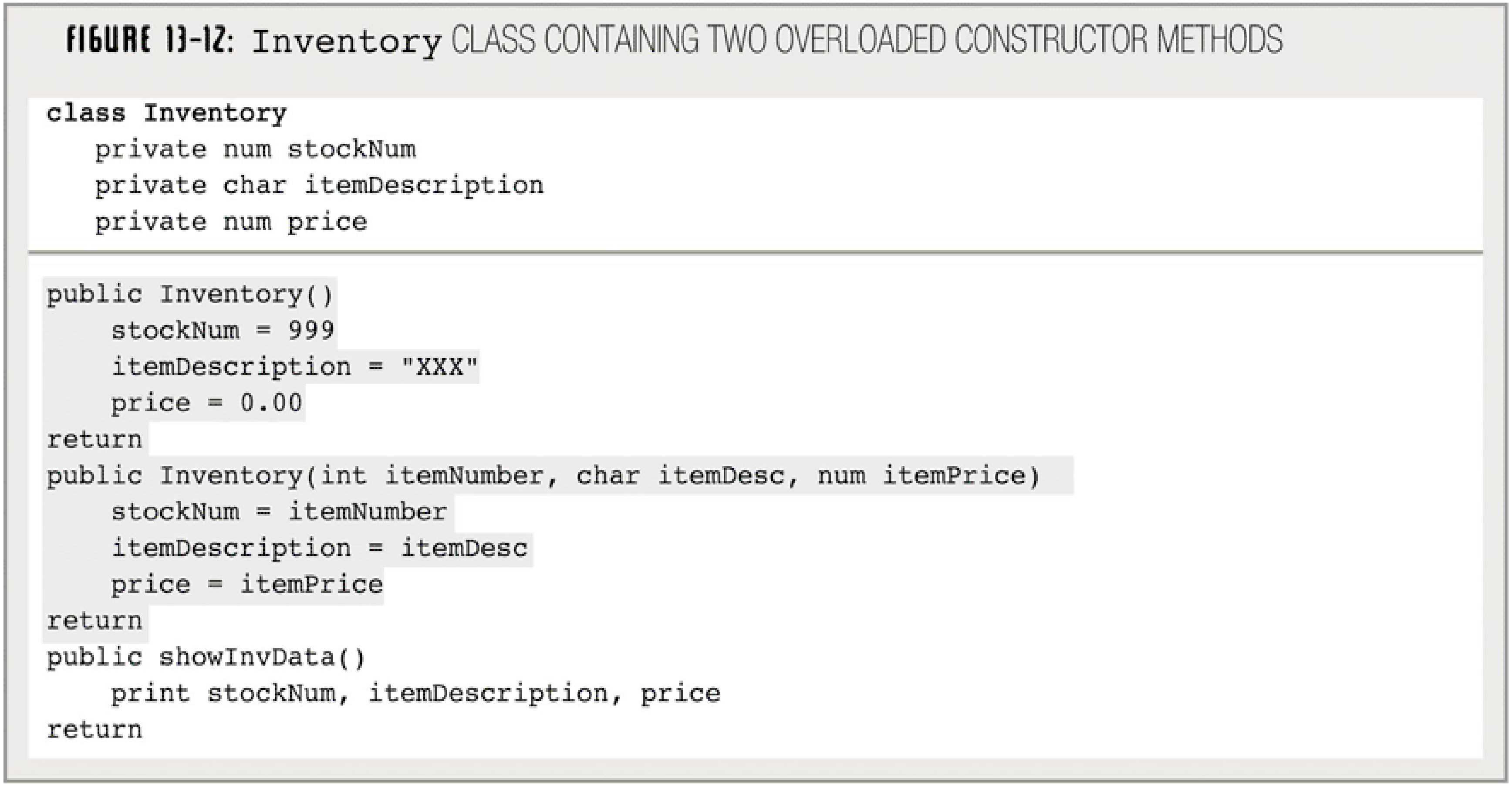


FIGURE 13-12: Inventory CLASS CONTAINING TWO OVERLOADED CONSTRUCTOR METHODS

```
class Inventory
   private num stockNum
   private char itemDescription
   private num price

public Inventory()
    stockNum = 999
    itemDescription = "XXX"
    price = 0.00
return
public Inventory(int itemNumber, char itemDesc, num itemPrice)
    stockNum = itemNumber
    itemDescription = itemDesc
    price = itemPrice
return
public showInvData()
    print stockNum, itemDescription, price
return
```

Understanding Constructor and Destructor Methods (continued)

- Besides constructors, most object-oriented languages contain automatically created methods called destructor methods, or simply, destructors
  - Execute when an object is destroyed
- Figure 13-14 shows a destructor for the `Inventory` class
  - Its only purpose is to notify the user that an object has been destroyed

Inventory Class Containing One Nondefault Constructor and a Destructor

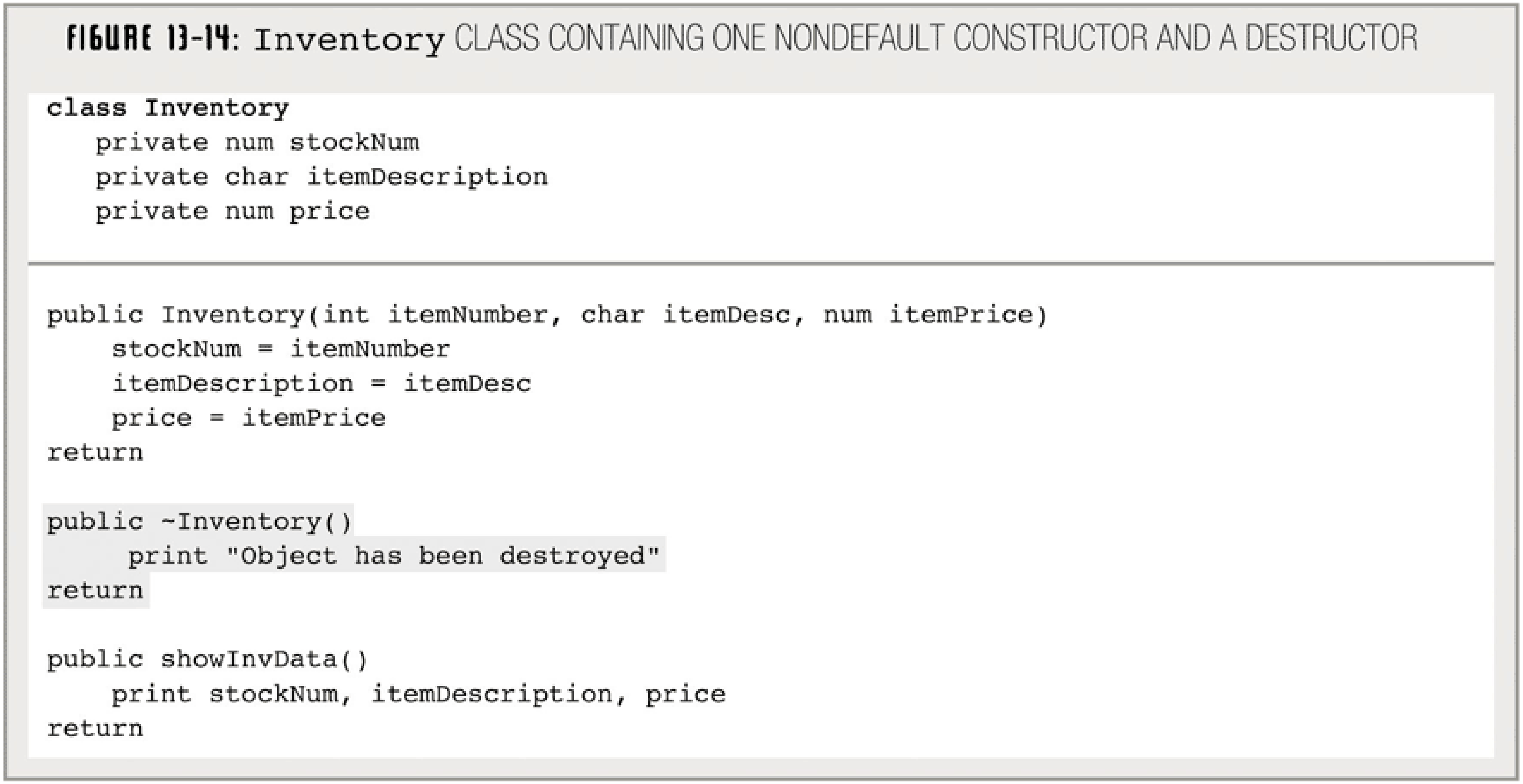


FIGURE 13-14: Inventory CLASS CONTAINING ONE NONDEFAULT CONSTRUCTOR AND A DESTRUCTOR

```
class Inventory
   private num stockNum
   private char itemDescription
   private num price

public Inventory(int itemNumber, char itemDesc, num itemPrice)
    stockNum = itemNumber
    itemDescription = itemDesc
    price = itemPrice
return

public ~Inventory()
     print "Object has been destroyed"
return

public showInvData()
    print stockNum, itemDescription, price
return
```

Using Predefined Classes to Create GUI Objects

- When you purchase or download an object-oriented programming language compiler, it comes packaged with a myriad of predefined, built-in classes stored in libraries:
  - collections of classes that serve related purposes
- Some of the most useful are the classes you can use to create graphical user interface (GUI) objects such as frames, buttons, labels, and text boxes

Using Predefined Classes to Create GUI Objects (continued)

- If no predefined GUI object classes existed, you could create your own
- However, there would be several disadvantages to doing this:
  - It would be a lot of work.
    - Requires a lot of code, and at least a modicum of artistic talent
  - It would be repetitious work
  - The components would look different in various applications

The Advantages of Object-Oriented Programming

- Whether you use classes you have created or use those created by others, when you instantiate objects in programs
  - you save development time because each object automatically includes appropriate, reliable methods and attributes
- When using inheritance, you can develop new classes more quickly
  - extend classes that already exist and work
  - concentrate only on new features the new class adds

Summary

- Object-oriented programming is a style of programming that focuses on an application’s data and the methods you need to manipulate that data
- A class is a category of items
- An object is a specific item that belongs to a class
- An object is an instance of a class
- You can create classes that are descendents of existing classes

Summary (continued)

- Object-oriented programs use polymorphism to allow the same operation to be carried out differently, depending on the context
- Constructors and destructors are methods that are automatically called when objects are created and destroyed
- You can use predefined classes to create GUI objects, saving development time and creating objects that work reliably and predictably
- When using objects in programs, you save development time