Programming Logic and Design
Ninth Edition

Chapter 9

Advanced Modularization Techniques

Objectives

In this lecture, you will learn about:

- The parts of a method
- Methods with no parameters
- Methods that require parameters
- Methods that return a value

The Parts of a Method

- **Method**
  - A program module that contains a series of statements that carry out a task
  - Invoke or call a method from another program or method
  - Any program can contain an unlimited number of methods
  - Each method can be called an unlimited number of times
  - Calling program or method is called the method's **client**

The Parts of a Method (continued)

- Method must include
  - **Method header** (also called the declaration or definition)
  - **Method body**
    - Contains the **implementation** (Statements that carry out the tasks)
  - **Method `return` statement** (Returns control to the calling method)
- Variables and constants
  - **Local**: declared in a method
  - **Global**: known to all program modules

Using Methods with No Parameters

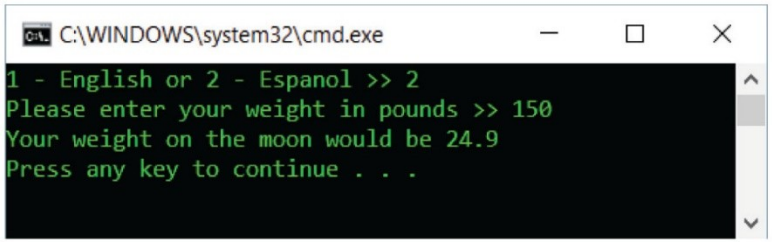


Figure 9-2 Output of moon weight calculator program in Figure 9-1

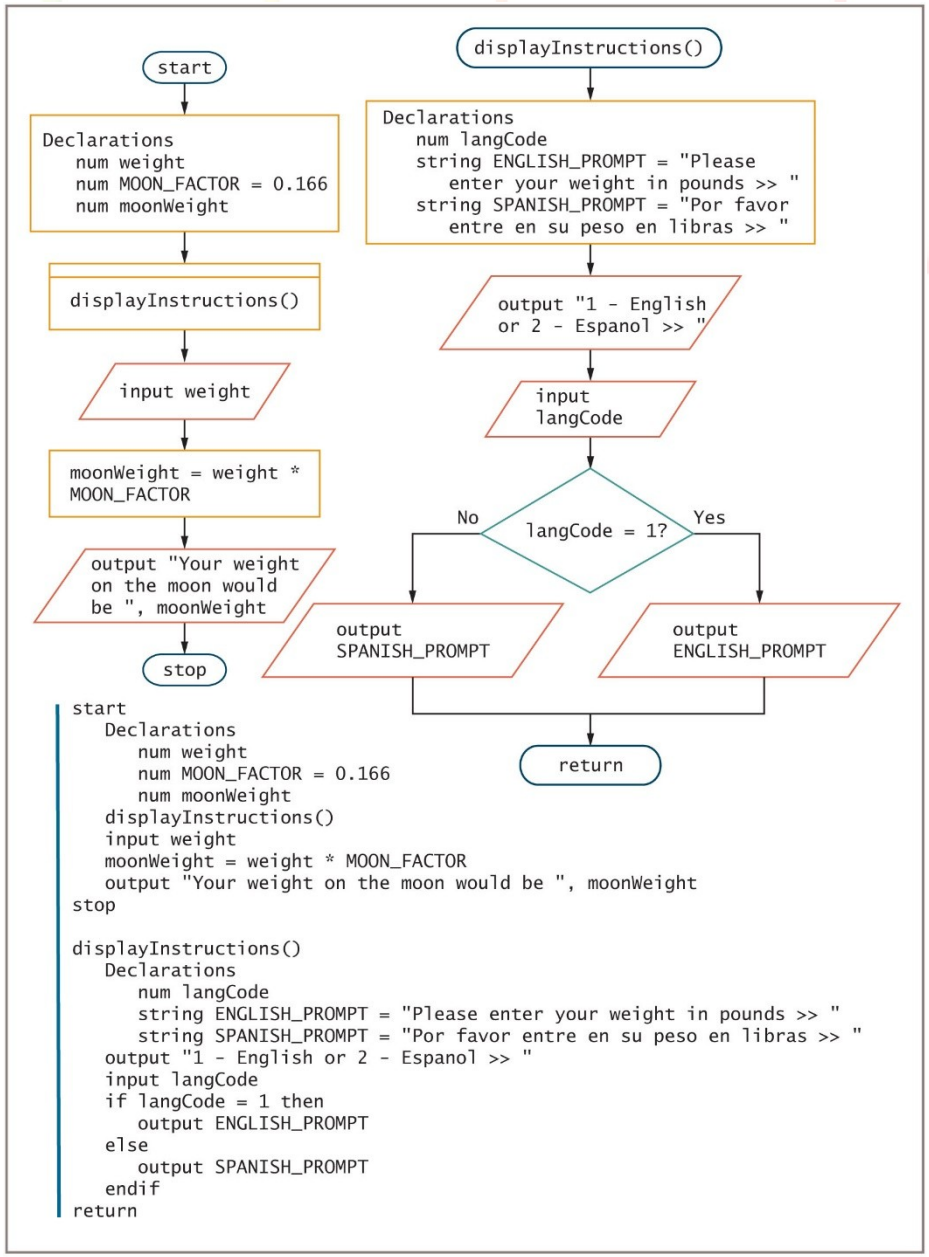


Figure 9-1 A program that calculates the user's weight on the moon

Using Methods with No Parameters (continued -1)

- When methods must share data
  - Pass the data into and return the data out of methods
- When you call a method from a program, you must know four things:
  - What the method does
  - Name of the called method
  - Type of information to send to the method, if any
  - Type of return data to expect from the method, if any

Creating Methods that Require Parameters

- **Argument to the method**
  - Pass a data item into a method from a calling program
- **Parameter to the method**
  - Method receives the data item
- When a method receives a parameter, you must provide a **parameter list** that includes:
  - The type of the parameter
  - The local name for the parameter
- **Signature**
  - Method's name and parameter list

Creating Methods that Require Parameters (continued -1)

- Improve the moon weight program by making the final output more user-friendly
- Several approaches
  - Rewrite the program without including any methods
  - Retain the `displayInstructions()` method, but make the `langCode` variable global
  - Retain the `displayInstructions()` method as is, but add a section to the main program that also asks the user for a preferred language
  - Store the variable that holds the language

Creating Methods that Require Parameters (continued -2)

- **Passed by value**
  - A copy of a value is sent to the method and stored in a new memory location accessible to the method
- Each time a method executes, parameter variables listed in the method header are redeclared

Creating Methods that Require Parameters (continued -3)

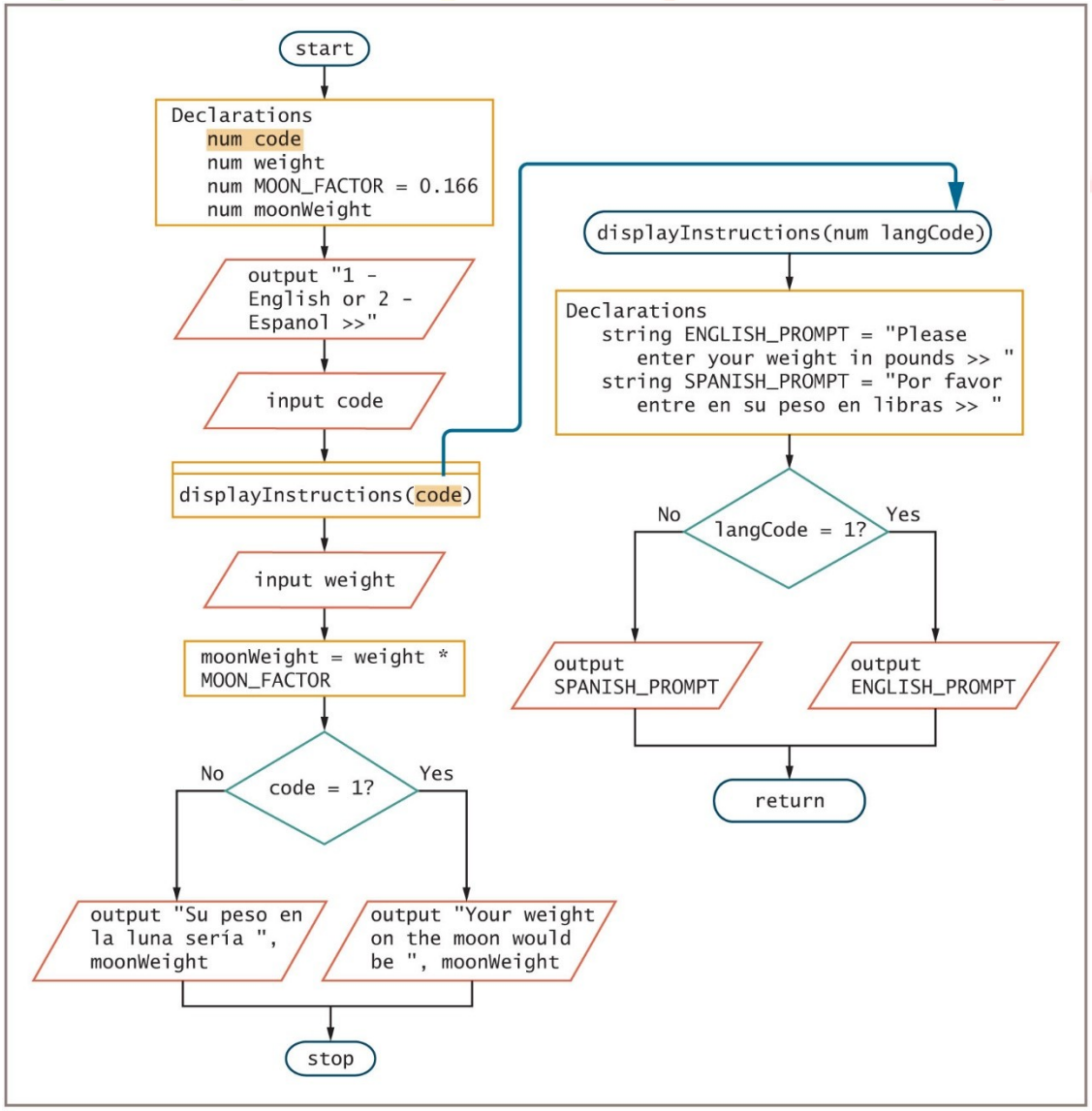


Figure 9-3 Moon weight program that passes an argument to a method *(continues)*

Creating Methods that Require Parameters (continued -4)

(continued)

```
start
   Declarations
      num code
      num weight
      num MOON_FACTOR = 0.166
      num moonWeight
   output "1 - English or 2 - Espanol >>"
   input code
   displayInstructions(code)
   input weight
   moonWeight = weight * MOON_FACTOR
   if code = 1 then
      output "Your weight on the moon would be ", moonWeight
   else
      output "Su peso en la luna sería ", moonWeight
   endif
stop

displayInstructions(num langCode)
   Declarations
      string ENGLISH_PROMPT = "Please enter your weight in pounds >> "
      string SPANISH_PROMPT = "Por favor entre en su peso en libras >> "
   if langCode = 1 then
      output ENGLISH_PROMPT
   else
      output SPANISH_PROMPT
   endif
return
```

Figure 9-3 Moon weight program that passes an argument to a method

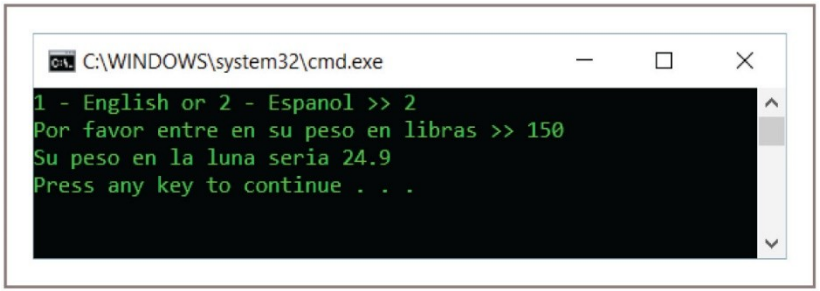


Figure 9-4 Typical execution of moon weight program in Figure 9-3

Creating Methods that Require Parameters (continued -5)

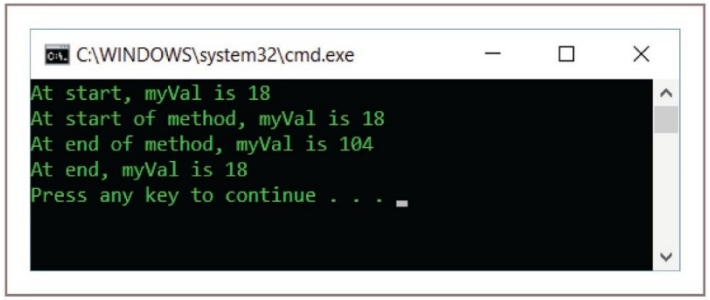


Figure 9-6 Execution of the program in Figure 9-5

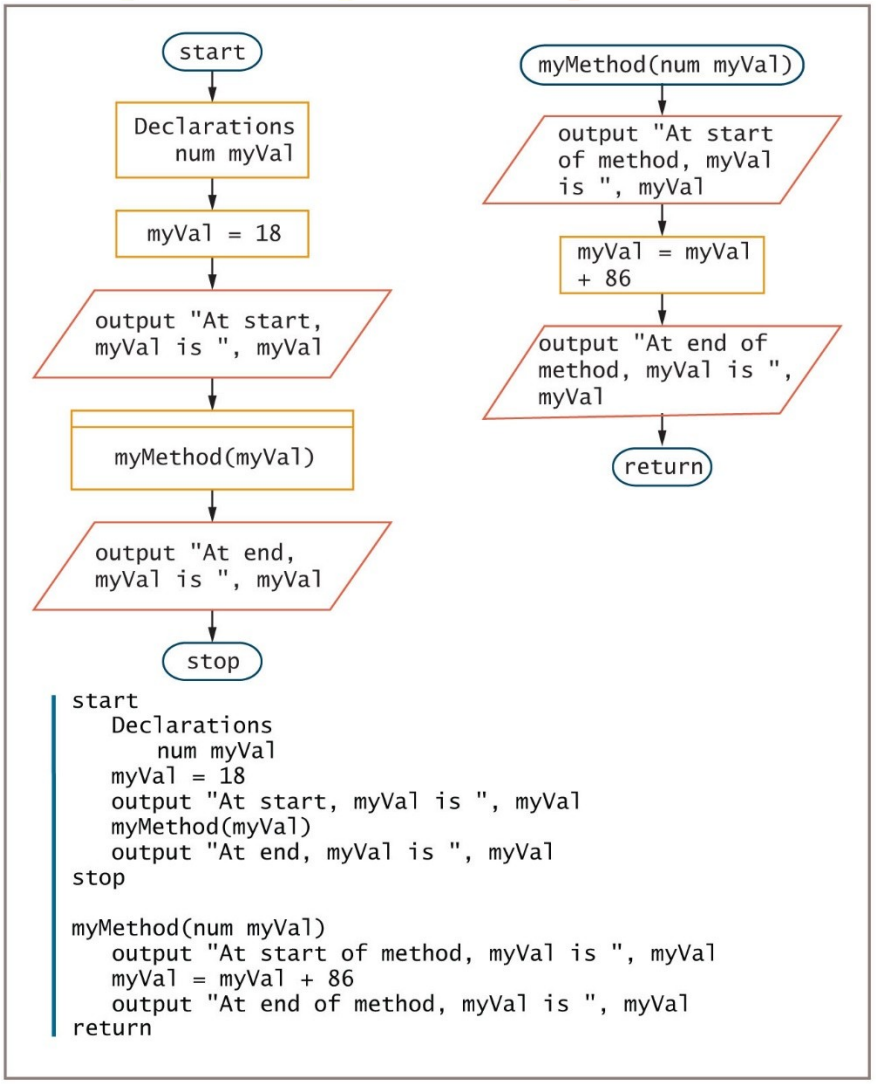


Figure 9-5 A program that calls a method in which the argument and parameter have the same identifier

Creating Methods that Require Multiple Parameters

- Methods can require more than one parameter
  - List the arguments within the method call, separated by commas
  - List a data type and local identifier for each parameter within the method header's parentheses
  - Separate each declaration with a comma
  - The data type must be repeated with each parameter
  - Arguments sent are called actual parameters
  - Variables that accept the parameters in the method are called formal parameters

Creating Methods that Require Multiple Parameters

(continued -1)

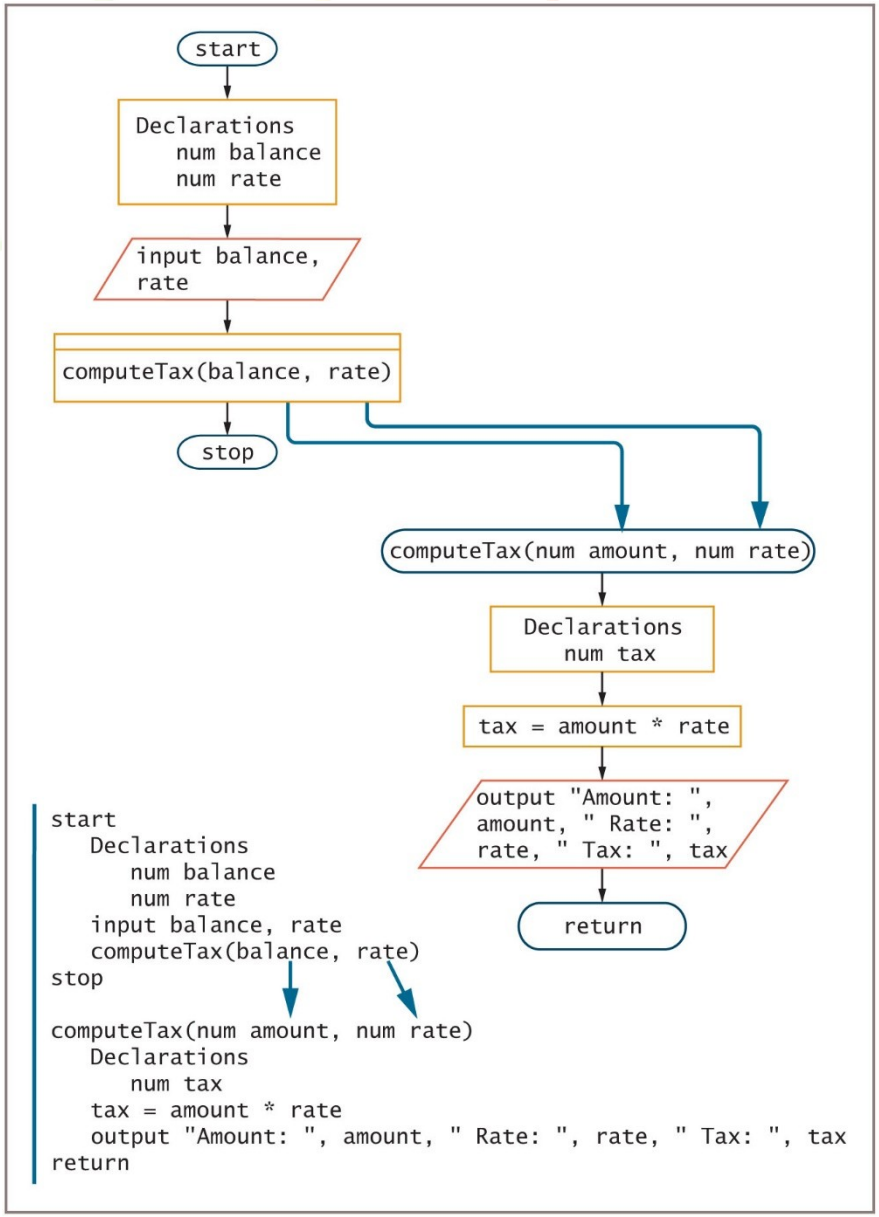


Figure 9-7 A program that calls a `computeTax()` method that requires two parameters

Creating Methods that Return a Value

- A variable declared within a method ceases to exist when the method ends
  - Goes out of scope
- To retain a value that exists when a method ends, return the value from the method back to the calling method
- When a method returns a value, the method must have a **return type** that matches the data type of the value that is returned

Creating Methods that Return a Value (continued -1)

- Return type for a method
  - Indicates the data type of the value that the method will send back
  - Can be any type
  - Also called **method's type**
  - Listed in front of the method name when the method is defined
- Method can also return nothing
  - Return type `void`
  - **Void method**

Creating Methods that Return a Value (continued -2)

- Example: `num getHoursWorked()`
  - Returns a numeric value
- Usually, you want to use the returned value in the calling method
  - Not required
  - Store in variable or use directly without storing
    - `output "Hours worked is ", getHoursWorked()`

Creating Methods that Return a Value (continued -3)

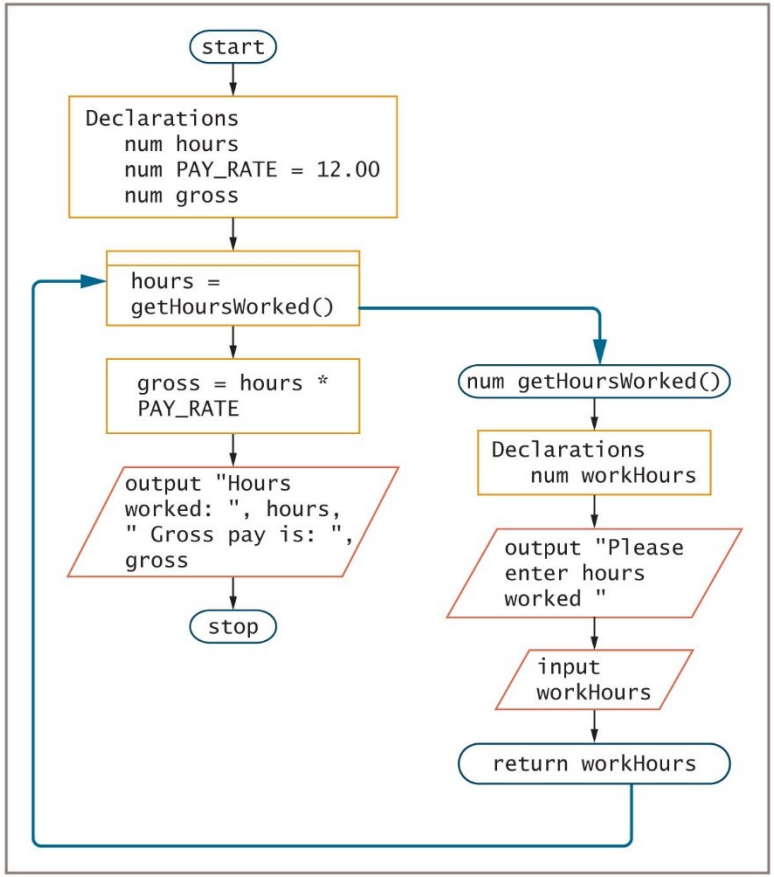


Figure 9-8 A payroll program that calls a method that returns a value

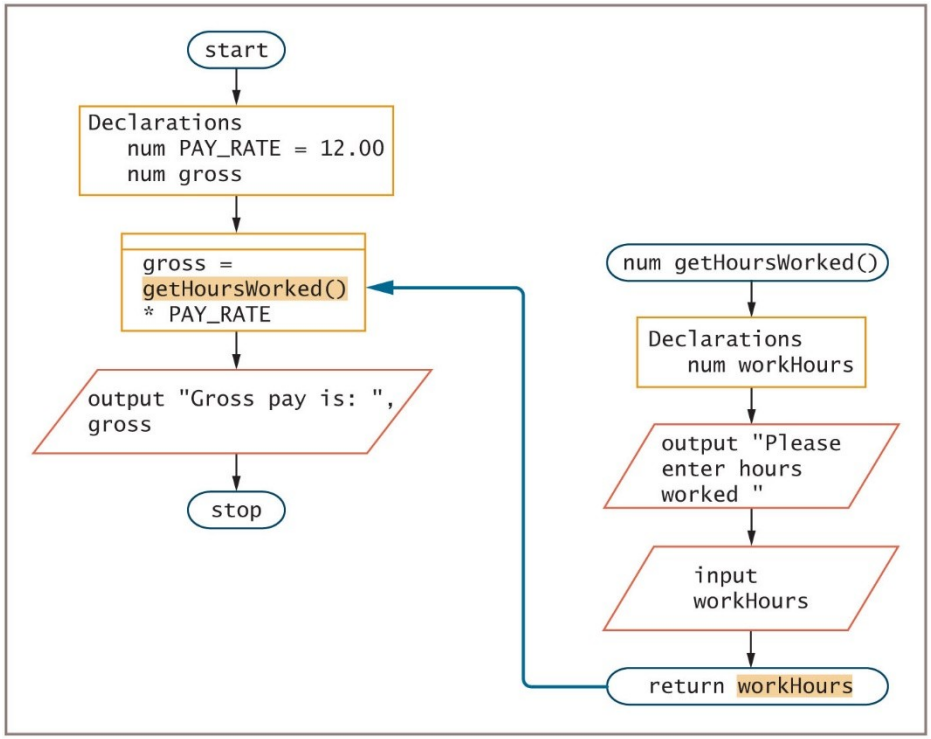


Figure 9-9 A program that uses a method's returned value without storing it

Creating Methods that Return a Value (continued -4)

- Technically, you are allowed to include multiple `return` statements in a method
  - It's not recommended
  - Violates structured logic

Creating Methods that Return a Value (continued -5)

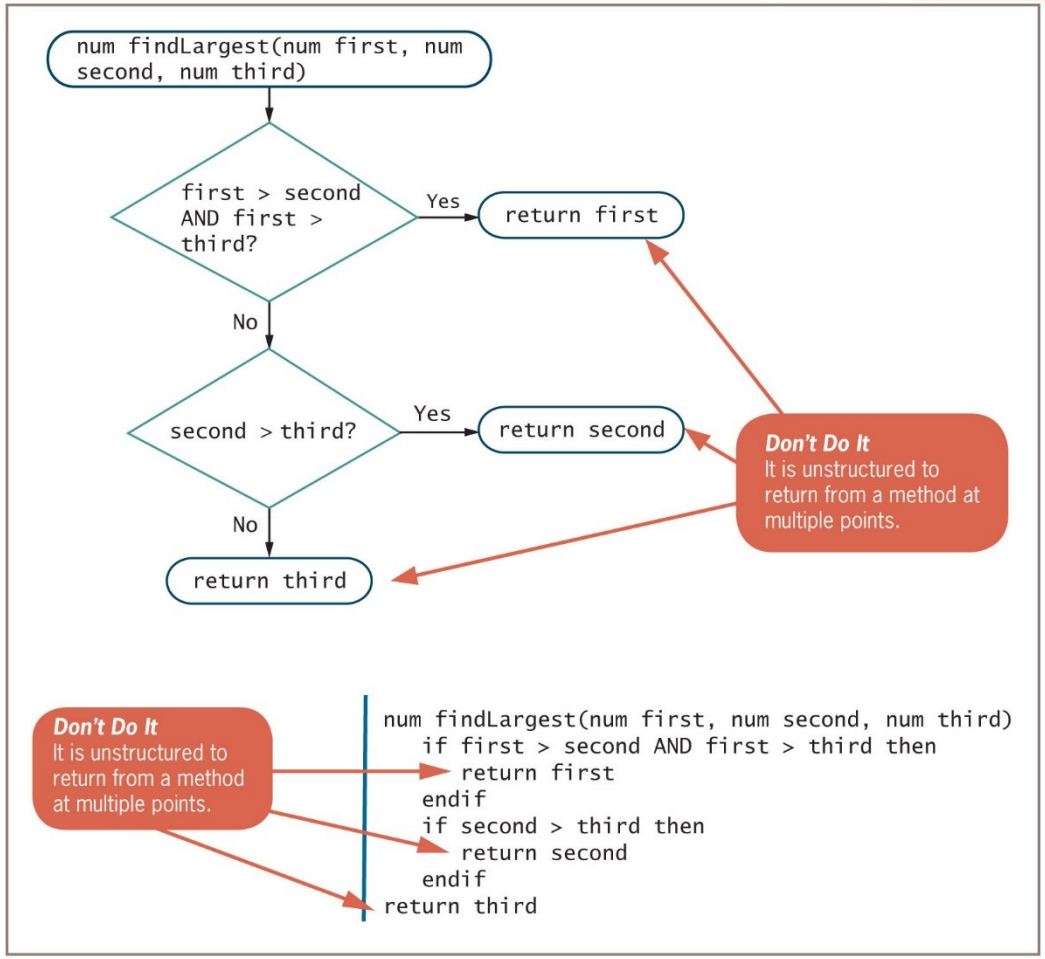
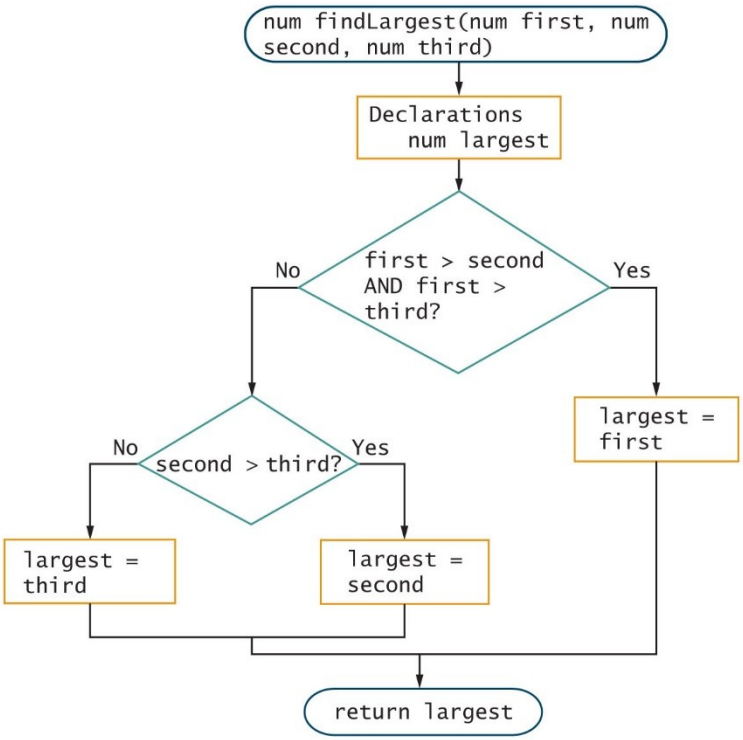


Figure 9-10 Unstructured approach to returning one of several values

Creating Methods that Return a Value

(continued -6)




```
num findLargest(num first, num second, num third)
   Declarations
      num largest
   if first > second AND first > third then
      largest = first
   else
      if second > third then
         largest = second
      else
         largest = third
      endif
   endif
return largest
```

Figure 9-11 Recommended, structured approach to returning one of several values

Creating Methods that Return a Value (continued -7)

- To use a method, you should know:
  - What the method does in general, but not necessarily how it carries out tasks internally
  - The method's name
  - The method's required parameters, if any
  - The method's return type, so that you can use any returned value appropriately
- **Overhead** refers to the extra resources and time required by an operation, such as calling a method

Summary

- A method is a program module that contains a series of statements that carry out a task
  - Must include a header body and `return` statement
  - A program can contain an unlimited number of methods
  - Methods can be called an unlimited number of times
- Data passed to a method is called an argument
- Data received by a method is called a