Static Methods

Chapter 4
Chapter Contents

Objectives
4.1 Introductory Example: Old MacDonald Had a Farm …
4.2 Getting Started with Methods
4.3 Example: The Volume of a Sphere
4.4 Methods: A Summary
4.5 Graphic/Internet Java: Old MacDonald … Applet
Chapter Objectives

- Look at how to build static (class) methods
- Study use of methods calling, parameters, returning values
- Contrast reference and primitive parameter passing
- Compare design process for methods to program design
- Give an example of Swing class JApplet used as a container in building applets.
Motivation

We seek to write reusable code for the purpose of avoiding having to do the same task repeatedly.

This can be done by putting the code in a method.

Various objects in a program can invoke the same method.

This chapter gives us the means to write Java methods.
4.1 Introductory Example: Old MacDonald Had a Farm …

Consider the children's song, Old MacDonald

Programming Problem: Write a program that displays the lyrics

Approaches:

- Simply display a very long `String`
- Use repetitiveness of lyrics, noting the only difference in each verse is the sound and the name of the creature.
Eliminating Redundant Code

Parameters for each verse
- Creature
- Sound made by creature
- Both will be `String` parameter variables

Form of method:
```java
private static String buildVerse(String creature, String sound)
{
    // statements to build verse
}
```

Note source code Figure 4.1
buildVerse Method

Tasks performed by buildVerse
- Receives two string parameters
- Uses these values to build string with lyrics
- Lyric string returned (separator of the buildVerse message)

Call (invocation) of buildVerse in main
- String variable lyrics created
- Initialized with concatenated calls to buildVerse
4.2 Getting Started With Methods

Formulas that compute values need not be limited to a single program.

Can be made available to other programs.

Perform the calculation in a reusable method.
Java Method Definition

Syntax:

modifiers returnType methodName (paramDecls)
{
    statements
}

- **modifiers**: describers (public, private, etc.)
- **returnType**: type of value returned by method, or void if it does not return a value
- **methodName**: identifier that names the method
- **paramDecls**: comma separated list of parameters
- **statements**: define the behavior of the method
Methods

- Heading of the method includes:
  - modifiers
  - return type
  - name parentheses with parameters

- Return statement – syntax:
  ```java
  return expression;
  ```
  - expression is evaluated
  - method terminates
  - execution transferred to caller
  - value of expression returned as value computed by the method
Methods That Return Nothing

- Type `void` is specified

- Example
  ```java
  public static void main (String [] args) {
    statements
  }
  ```

- No `return` statement is required

- These methods can be thought of as “doing a task”, instead of returning a value
Designing and Defining Methods

- Note the usefulness of object-centered design
- Similar to design of programs
- They have objects, behavior, and use an algorithm
- Consider a method for mass-to-energy conversion from Section 3.1
# Objects for Mass-to-Energy

<table>
<thead>
<tr>
<th>Object Descriptions</th>
<th>Type</th>
<th>Kind</th>
<th>Movement</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>mass</td>
<td>double</td>
<td>variable</td>
<td>received</td>
<td>m</td>
</tr>
<tr>
<td>c</td>
<td>double</td>
<td>constant</td>
<td>none</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>integer</td>
<td>constant</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>energy</td>
<td>double</td>
<td>variable</td>
<td>returned</td>
<td>none</td>
</tr>
</tbody>
</table>
Method Specifications

Give a description of what the method is supposed to do:

1. What values does the method receive?
2. What values are input to the method?
3. What are the restrictions or limitations the preconditions?
4. What values does the method return?
5. What values does the method output?
6. What effects are produced, the postconditions?
Method Specifications for Mass-to-Energy

For our Mass-to-Energy method:

- Receive: mass, a double
- Precondition: mass > 0
- Return: the amount of energy when mass is converted to energy
Method Stubs

- Stub includes:
  - Heading
  - Empty braces `{ }` for body

- Heading includes:
  - Name
  - Parameter for each argument received must be specified
  - Return type (or `void`)

```java
public static double massToEnergy (double m) {
}
```
Local Variables

- Other constant values or temporary variables
- Named values not received from the caller
- They exist only while the method is executing
  - Another method may use same identifier
  - The local variable/constant can be accessed only from within the method where declared
- Compiler will catch this error
Method Algorithm

- After stub is defined
- return to design phase
- specify operations
- establish algorithm

Algorithm for Mass-to-Energy

- Receive mass $m$ from caller
- Return $m \times C^2$
Method for Mass–To–Energy

```java
public static double
    massToEnergy(double m)
{
    final double C = 2.997925e8;
    // meters per sec
    return m * Math.pow(C, 2);
}
```
Method Documentation

Include a /* comment */ at the top to give the method's specification

what it does

Parameters
Receive:

Description of value return
Return:
Flow of Execution

- Statements in `main()` executed
- Method call encountered
- Control passes to method
  - values passed to parameters
  - statements in method executed
  - return encountered
- Control passes back to statement following method call
Method Testing: Verification and Validation

- Often test a method independently of program that uses it
- Write simple “driver” program
  - receive inputs
  - invoke method
  - print results
- Observe correctness of results
Parameter Passing

- When a method called
  - list of arguments in call matched (left to right) with parameters
  - must be same number of parameters
  - types must be compatible

- Values in call copied to parameters

- In examples shown so far, argument in call cannot be altered by action of method
Object–Centered Design with Methods

- **Behavior**
  - state precise behavior of program

- **Objects**
  - identify problem's objects
  - build a new class to represent types as necessary
Object-Centered Design with Methods

Operations
- identify required operations – if operation not predefined …
- build methods to implement the operation
- store method in class responsible for providing the operation

Algorithm
- arrange operations in an order that solves the problem
4.3 Example: Volume of a Sphere

Given the radius \( r \), what is the weight of a ball (sphere) of wound twine?

Object–Centered Design
- display prompt for radius
- read value for radius
- compute weight of sphere
- display results on screen

Note this is generalized for sphere of arbitrary size
## Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Type</th>
<th>Kind</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen</td>
<td>Screen</td>
<td>varying</td>
<td>theScreen</td>
</tr>
<tr>
<td>Prompt</td>
<td>String</td>
<td>constant</td>
<td></td>
</tr>
<tr>
<td>Radius</td>
<td>Double</td>
<td>varying</td>
<td>Radius</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Keyboard</td>
<td>varying</td>
<td>theKeyboard</td>
</tr>
<tr>
<td>Weight</td>
<td>Double</td>
<td>varying</td>
<td>Weight</td>
</tr>
<tr>
<td>Sphere</td>
<td></td>
<td>varying</td>
<td></td>
</tr>
</tbody>
</table>
Operations

- Display a String (prompt) on the screen
- Read a number from keyboard, store it in radius
- Compute weight using radius
- Display a number (weight) on screen
New Class Required

- Java has no predefined sphere object
- Also no predefined operation for volume or weight of a sphere

Solution:
- build a method to calculate weight
- create a sphere class to use the weight method
- We will need an additional variable object
  - density (weight = density * volume)
A Volume Method

Objects

\[ \text{Volume} = \frac{4}{3} \times \pi \times r^3 \]

Note

- \( r \) is the only variable
- 4, 3, and \( \pi \) are constants

These (along with the result, \( \text{volume} \)) are the objects of this method
Volume Method
Operations and Algorithm

- Receive real value (radius) from caller
- Cube the real value (radius^3)
- Multiply by 4.0 and by \(\pi\)
- Divide by 3.0
- Return result

\[4.0 \times \pi \times \text{radius}^3/3.0\]
class Sphere extends Object
{
    public static double volume(double radius)
    {
        return 4.0 * Math.PI * Math.pow(radius, 3) / 3.0;
    }
}
Mass Method

mass = density * volume(radius)

density and radius are the inputs to the method

volume is a call to the volume method

mass is the result to be returned

These are the objects of the method
Mass Algorithm

Receive inputs

radius

density

Multiply density times value returned by call to volume method

Return these results
class Sphere extends Object
{

    public static double volume
        (double radius)
    {
        . . . .
    }

    public static double density
        (double radius, double density)
    {
        return density * volume(radius);
    }


Defining the Density Method


Algorithm for Main Method

1. Construct theKeyboard, theScreen
2. theScreen displays prompt for radius
3. theKeyboard reads a double value into radius
4. theScreen displays prompt for density
5. theKeyboard reads a double into density
6. Compute weight, use mass() method from class Sphere
7. theScreen displays weight and descriptive text
Coding and Testing SphereWeigher Class

- Note source code Figure 4.5
  - import `Sphere` class
  - use of methods from `Sphere` class
- Note Sample Runs
4.4 Methods: A Summary

- Specify a parameter for each value received by the method.
- Value supplied to the parameter when method invoked is called an argument.
- Arguments matched with parameters from left to right.
  - Must be the same number of arguments.
  - Types must match (be compatible).
4.4 Methods: A Summary

- If argument is a reference type, address is copied to parameter.
- Both parameter and argument refer to the same object.
- Instance (object) methods defined without the `static` modifier.
- Messages invoking them are sent to an instance of the class.
- When `method1()` calls `method2()`, control returns to `method1()` when `method2()` finishes.
4.4 Methods: A Summary

- Local objects are defined only while method containing them is executing.

- `void` is used to specify the return type of a method which returns no values.

- Value is returned from a method to the call using the `return` statement.
4.5 Graphic/Internet Java
Old MacDonald ... Applet

- Convert previous application into an applet
- Include picture of Farmer MacDonald himself
- One basic difference is handling the output
  - text and picture are both painted in specified areas of the screen
Output In An Applet

- Window frame container
- Intermediate containers known as panes or panels
- Areas for panes include north, east, south, west (top, right, bottom, left), and center

Use the `.add()` method
```
g getContentPane().add(song, "West");
```
Old MacDonald Applet

Note features of source code, Figure 4.6

- re-use `buildVerse()` method
- `init()` instead of `main()`
- use of `.add()` to place lyrics on the left and picture (.gif file) on the right