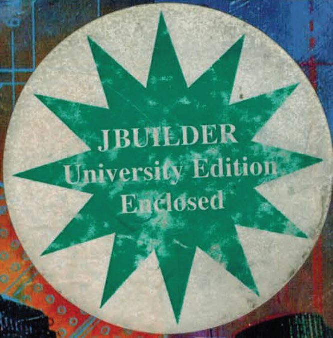


JAVA™

SOFTWARE SOLUTIONS

Foundations of Program Design

JOHN LEWIS
WILLIAM LOFTUS

SYMANTEC Exhibit 1025 Page 1

Editor-in-Chief	Lynne Doran Cote
Associate Editor	Deborah Lafferty
Production Manager	Karen Wernholm
Production Editor	Amy Willcutt
Marketing Manager	Tom Ziolkowski
Compositor	Michael and Sigrid Wile
Technical Artist	George Nichols
Copyeditor	Roberta Lewis
Text Design	Ron Kosciak
Indexer	Nancy Fulton
Proofreading	Phyllis Coyne et al.
Cover Designer	Diana Coe

Library of Congress Cataloging-in-Publication Data

Lewis, John, Ph.D.
Java software solutions : foundations of program design / John Lewis, William Loftus.
p. cm.
Includes index.
ISBN 0-201-57164-1
1. Java (Computer program language) 2. Object-oriented programming (Computer science) I. Loftus, William. II. Title.
QA76.73.J38L49 1998
005.13'3--dc21 97-19400
CIP

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Cover image © Jerry Blank/SIS

Access the latest information about Addison-Wesley titles from our World Wide Web site: http://www.awl.com/cseng

Reprinted with corrections, January 1998.

Copyright © 1998 by Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

6 7 8 9 10-MA-01009998

Pref

We have design
language. It ser
for pursuing ad
with object-orie
ment high-quali
This text w
gramming and
ground up, with
emerged in mid
Web effects. Ov
object-oriented
discovered that
ing concepts wh
In response
nary version. Si
ing topics to p
students better g
embrace Java 1.
earlier version, i

Object-Orient

We introduce ob
out. We have fo
they are present

4.4 Defining Methods

We've already used many methods in various programs and we know that methods are part of a class. Let's now examine method definitions closely in preparation for defining our own classes.

A *method* is a group of programming language statements that are given a name. A method is associated with a particular class. Each method has a *method definition* that specifies the code that gets executed when the method is invoked. We've invoked the `println` method many times, but because we didn't write `println`, we have not been concerned about its definition. We call it assuming that it will do its job reliably.

When a method is called, the flow of control transfers to that method. One by one, the statements of that method are executed. When that method is done, control returns to the location where the call was made and execution continues. This process is pictured in Fig. 4.4.

We've defined the `main` method of a program many times. Its definition follows the same syntax as all methods:

```
return-type  method-name  ( parameter-list ) {
   statement-list
}
```

The header of a method includes the type of the return value, the method name, and a list of parameters that the method accepts. The list of statements that makes up the

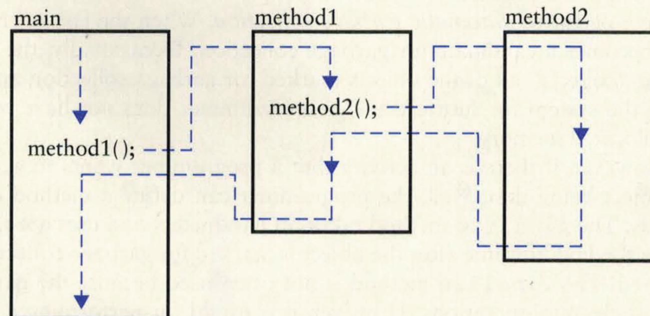


Figure 4.4 The flow of control following method invocations

body of the method are
method called third_p

```
int third_power (int
   int cube;
   cube = number * n
   return cube;
} // method third_p
```

A method may dec
that method. The vari
method. Local variable
other methods of the s
the `main` method. The
do not exist except wh
local variable is lost fr
value to be maintained
the class level. The dec
list, but it must be decl

Key Concept A var
used outside of it.

The `return` stat

Methods can return a
method header. The re
When a method does no
type, as is always done
in the method header. T

```
return;
```

or

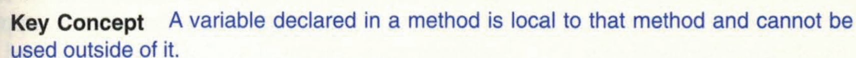
```
return expressio
```

SYMANTEC Exhibit 1025 Page 3

body of the method are defined in a block. The following code is the definition of a method called `third_power`:

```
int third_power (int number) {
   int cube;
   cube = number * number * number;
   return cube;
}  // method third_power
```

A method may declare *local variables* in the body of the method for use only in that method. The variable `cube` in the `third_power` method is local to that method. Local variables cannot be accessed from outside of the method, even from other methods of the same class. In previous examples we've declared variables in the `main` method. These variables were local to the `main` method. Local variables do not exist except when the method is executing; therefore the value stored in a local variable is lost from one invocation of the method to the next. If you want a value to be maintained from one call to the next, you should define the variable at the class level. The declaration of a local variable can be mixed into the statement list, but it must be declared before it is used.



Key Concept A variable declared in a method is local to that method and cannot be used outside of it.

The `return` statement

Methods can return a value, whose type must correspond to the *return type* in the method header. The return type can be a primitive type or a reference to an object. When a method does not return any value, the reserved word `void` is used as the return type, as is always done with the `main` method. A return type must always be specified in the method header. The `return` statement in a method can take one of two forms:

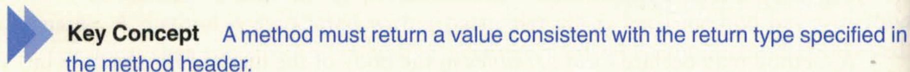
```
return;
```

or

```
return expression;
```


SYMANTEC Exhibit 1025 Page 4

The first form causes the processing flow to return to the calling location without returning a value. The second form returns to the calling method and specifies the value that is to be returned. If a return type other than `void` is specified in the method header, then the Java compiler insists that a `return` statement exist in the program and that a value of the proper type is returned.



Key Concept A method must return a value consistent with the return type specified in the method header.

The following code is another way to define the `third_power` method, performing a calculation in the expression of the `return` statement. This modification eliminates the need for the local variable.

```
int third_power (int number) {
   return (number * number * number);
}  // method third_power
```

If there is no `return` statement in a method, processing continues until the end of the method is reached. If there is a `return` statement, then processing is stopped for that method when the `return` statement is executed, and control is returned to the statement that invoked the method.

It is usually not good practice to use more than one `return` statement in a method even though it is possible to do so. In general, a method should have one `return` statement as the last line of the method body unless it makes the method overly complex.

Parameters

A *parameter* is a value that is passed into a method when it is invoked. The *parameter* list in the header of a method specifies the types of the values that are passed and the names by which the called method will refer to the parameters in the method definition. In the method definition, the names of the parameters accepted are called *formal parameters*. In the invocations, the values passed into a method are called *actual parameters*. A method invocation and definition always specify the parameter list in parentheses after the method name. If there are no parameters, an empty set of parentheses are used.

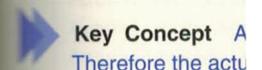
The formal parameters are identifiers that essentially act as local variables for the method and whose initial value comes from the calling method. Actual parame-

ters can be literals,
passed as the paran

```
public void acid_
   String title
   generate_repo
}  // method aci
```

This example is a
invoking another r
acid_test are s
Note that substa
method invocation
in acid_test,
generate_repor

When primitiv
meter. When objects
to the actual param
parameter becomes
Java, primitive data



Key Concept A
Therefore the actu

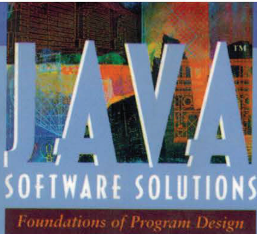
Let's look at a
we have a class Num
gram demonstrates

```
// Demonstrates
class Parameter_

   public static
      (int forma
       Num forma

      System.out
      System.out
      System.out
      System.out
```

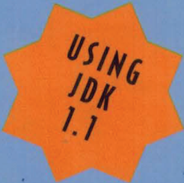
SYMANTEC Exhibit 1025 Page 5



"The book is a good one. Examples have been kept simple, focused, and useful. This is important to hold a student's attention. The book is based on JDK 1.1, which is a more mature API than the earlier version. It does a good job of introducing programming principles in Java."

—Vijay Srinivasan, JavaSoft, Sun Microsystems, Inc.

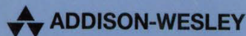
John Lewis, Villanova University
William Loftus, WPL Laboratories, Inc.



Using Java 1.1, *Java Software Solutions* teaches beginning programmers how to design and implement high-quality object-oriented software. The authors emphasize problem solving through understanding requirements, exploring options, and designing conceptually clean solutions. John Lewis and Bill Loftus wrote this book from the ground up, taking full advantage of Java to teach introductory programming. Throughout the book, the authors intertwine the use of applets and applications demonstrate computing concepts. Applets are introduced early, building on the excitement of the web, while applications help readers gain a clear understanding of programming concepts.

Highlights

- Presents objects early and reinforces the design principles of object-oriented programming throughout
- Combines small, readily understandable examples with larger, practical ones to explore a variety of programs
- Provides a diverse wealth of self-review questions, exercises, programming projects and clear chapter objectives
- Contains Java reference material, including 15 appendices that contain style guidelines and more

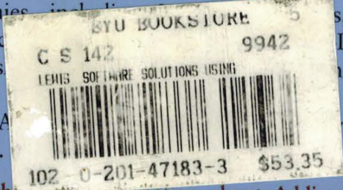


Addison-Wesley is an imprint of Addison Wesley Longman, Inc.

About the Authors

John Lewis is an Assistant Professor of Computer Science at Villanova University. Dr. Lewis received his Ph.D. from Virginia Tech in 1991. His area of specialization is software engineering, and he regularly teaches courses in introductory computing, software engineering, object-oriented design, operating systems, and algorithms & data structures. Dr. Lewis is a member of the ACM, the IEEE Computer Society, and Sigma Xi, the scientific research society. He is also the Conference Chair for the 1998 SIGCSE Technical Symposium.

William Loftus is the Technical Director of WPL Laboratories, Inc. where he directs many software development and research projects. He is a highly respected expert in the areas of object-oriented systems development and software engineering. He has personally consulted to many Fortune 500 companies, s, Computer Sc Packard. Previous Unisys. He has Award from DA Action Awards.



Access th information about Addison-Wesley books at our World Wide Web site: http://www.a