



Integrated Cloud Applications & Platform Services

Java SE: Programming I

Student Guide - Volume I

D102470GC10

Edition 1.0 | August 2018 | D104107



Learn more from Oracle University at education.oracle.com

ORACLE®

Author

Anjana Shenoy

**Technical Contributors
and Reviewers**

Kenny Somerville

Nick Ristuccia

Joni Gordon

Geetha Nazare

Joe Boulenouar

Graphic Designer

Prakash Dharmalingam

Editor

Moushmi Mukherjee

Publishers

Veena Narasimhan

Asief Baig

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introduction

- Audience 1-2
- Introductions 1-3
- Course Objectives 1-4
- Schedule 1-5
- Lesson Format 1-8
- Course Environment 1-9
- How Do You Learn More After the Course? 1-10
- Additional Resources 1-11
- Summary 1-13

2 What Is a Java Program?

- Objectives 2-2
- Topics 2-3
 - Purpose of a Computer Program 2-4
 - Translating High-Level Code to Machine Code 2-5
 - Linked to Platform-Specific Libraries 2-6
 - Platform-Dependent Programs 2-7
 - Topics 2-8
 - Key Features of the Java Language 2-9
 - Java Is Platform-Independent 2-10
 - Java Programs Run In a Java Virtual Machine 2-11
 - Procedural Programming Languages 2-12
 - Java Is an Object-Oriented Language 2-13
 - Topics 2-14
 - Verifying the Java Development Environment 2-15
 - Examining the Installed JDK: The Tools 2-16
 - Examining the Installed JDK: The Libraries 2-17
 - Topics 2-18
 - Compiling and Running a Java Program 2-19
 - Compiling a Program 2-20
 - Executing (Testing) a Program 2-21
 - Output for a Java Program 2-22
 - Exercise 2-1 2-23

Quiz 2-24

Summary 2-25

3 Creating a Java Main Class

Objectives 3-2

Topics 3-3

Java Classes 3-4

Program Structure 3-5

Java Packages 3-6

Java IDEs 3-7

The NetBeans IDE 3-8

Creating a Java Project 3-9

Creating a Java Class 3-10

Exercise 3-1:Creating a New Project and Java Class 3-11

Opening an Existing Java Project 3-12

Creating a New Java Package 3-13

Topics 3-14

The main Method 3-15

A main Class Example 3-16

Output to the Console 3-17

Avoiding Syntax Errors 3-18

Compiling and Running a Program by Using NetBeans 3-19

Exercise 3-2: Creating a main Method 3-20

Quiz 3-21

Summary 3-22

4 Data in a Cart

Objectives 4-2

Topics 4-3

Variables 4-4

Variable Types 4-5

Naming a Variable 4-6

Java SE 9: The Underscore Character Is Not a Legal Name 4-7

Uses of Variables 4-8

Topics 4-9

Examples: Variable Declaration and Initialization 4-10

String Concatenation 4-11

Exercise 4-1: Using String Variables 4-13

Quiz 4-14

Topics 4-15

int and double Values 4-16

Initializing and Assigning Numeric Values	4-17
Topics	4-18
Standard Mathematical Operators	4-19
Increment and Decrement Operators (++ and --)	4-20
Operator Precedence	4-21
Using Parentheses	4-23
Exercise 4-2: Using and Manipulating Numbers	4-24
Quiz	4-25
Summary	4-27

5 Managing Multiple Items

Objectives	5-2
Topics	5-3
Making Decisions	5-4
The if/else Statement	5-5
Boolean Expressions	5-6
Relational Operators	5-7
Examples	5-8
Exercise 5-1: Using if Statements	5-9
Quiz	5-10
Topics	5-11
What If There Are Multiple Items in the Shopping Cart?	5-12
Introduction to Arrays	5-13
Array Examples	5-14
Array Indices and Length	5-15
Declaring and Initializing an Array	5-16
Accessing Array Elements	5-18
Exercise 5-2: Using an Array	5-19
Quiz	5-20
Topics	5-22
Loops	5-23
Processing a String Array	5-24
Using break with Loops	5-25
Exercise 5-3: Using a Loop to Process an Array	5-26
Quiz	5-27
Summary	5-28

6 Describing Objects and Classes

Interactive Quizzes	6-2
Objectives	6-3
Topics	6-4

Object-Oriented Programming	6-5
Duke's Choice Order Process	6-6
Characteristics of Objects	6-7
Classes and Instances	6-8
Quiz	6-9
Topics	6-10
The Customer Properties and Behaviors	6-11
The Components of a Class	6-12
Modeling Properties and Behaviors	6-13
Exercise 6-1: Creating the Item Class	6-14
Topics	6-15
Customer Instances	6-16
Object Instances and Instantiation Syntax	6-17
The Dot (.) Operator	6-18
Objects with Another Object as a Property	6-19
Quiz	6-20
Topics	6-21
Accessing Objects by Using a Reference	6-22
Working with Object References	6-23
References to Different Objects	6-26
References and Objects in Memory	6-28
Assigning a Reference to Another Reference	6-29
Two References, One Object	6-30
Exercise 6-2: Modifying the ShoppingCart to Use Item Fields	6-31
Topics	6-32
Arrays Are Objects	6-33
Declaring, Instantiating, and Initializing Arrays	6-34
Storing Arrays in Memory	6-35
Storing Arrays of Object References in Memory	6-36
Quiz	6-37
Topics	6-39
Soccer Application	6-40
Creating the Soccer Application	6-41
Soccer Web Application	6-42
Summary	6-43
Practices Overview	6-44

7 Manipulating and Formatting the Data in Your Program

Objectives	7-2
Topics	7-3
String Class	7-4

Concatenating Strings	7-5
String Method Calls with Primitive Return Values	7-8
String Method Calls with Object Return Values	7-9
Topics	7-10
Java API Documentation	7-11
JDK 10 API Documentation	7-12
Java Platform SE and JDK Version 10 API Specification	7-13
Java Platform SE 10: Method Summary	7-14
Java Platform SE 10: Method Detail	7-15
indexOf Method Example	7-16
Exercise 7-1: Use indexOf and substring Methods	7-17
Topics	7-18
StringBuilder Class	7-19
StringBuilder Advantages over String for Concatenation (or Appending)	7-20
StringBuilder: Declare and Instantiate	7-21
StringBuilder Append	7-22
Quiz	7-23
Exercise 7-2: Instantiate the StringBuilder object	7-24
Topics	7-25
Primitive Data Types	7-26
Some New Integral Primitive Types	7-27
Floating Point Primitive Types	7-28
Textual Primitive Type	7-29
Java Language Trivia: Unicode	7-30
Constants	7-31
Quiz	7-32
Topics	7-33
Modulus Operator	7-34
Combining Operators to Make Assignments	7-35
More on Increment and Decrement Operators	7-36
Increment and Decrement Operators (++ and —)	7-37
Topics	7-38
Promotion	7-39
Caution with Promotion	7-40
Type Casting	7-42
Caution with Type Casting	7-43
Using Promotion and Casting	7-45
Compiler Assumptions for Integral and Floating Point Data Types	7-46
Automatic Promotion	7-47
Using a long	7-48
Using Floating Points	7-49

Floating Point Data Types and Assignment	7-50
Quiz	7-51
Exercise 7-3: Declare a Long, Float, and Char	7-52
Summary	7-53
Practices Overview	7-54

8 Creating and Using Methods

Objectives	8-2
Topics	8-3
Basic Form of a Method	8-4
Calling a Method from a Different Class	8-5
Caller and Worker Methods	8-6
A Constructor Method	8-7
Writing and Calling a Constructor	8-8
Calling a Method in the Same Class	8-9
Topics	8-10
Method Arguments and Parameters	8-11
Method Parameter Examples	8-12
Method Return Types	8-13
Method Return Types Examples	8-14
Method Return Animation	8-15
Passing Arguments and Returning Values	8-16
More Examples	8-17
Code Without Methods	8-18
Better Code with Methods	8-19
Even Better Code with Methods	8-20
Variable Scope	8-21
Advantages of Using Methods	8-22
Exercise 8-1: Declare a setColor Method	8-23
Topics	8-24
Static Methods and Variables	8-25
Example: Setting the Size for a New Item	8-26
Creating and Accessing Static Members	8-27
When to Use Static Methods or Fields	8-28
Some Rules About Static Fields and Methods	8-29
Static Fields and Methods vs. Instance Fields and Methods	8-30
Static Methods and Variables in the Java API	8-31
Examining Static Variables in the JDK Libraries	8-32
Using Static Variables and Methods: System.out.println	8-33
More Static Fields and Methods in the Java API	8-34
Converting Data Values	8-35

Topics	8-36
Passing an Object Reference	8-37
What If There Is a New Object?	8-38
A Shopping Cart Code Example	8-39
Passing by Value	8-40
Reassigning the Reference	8-41
Passing by Value	8-42
Topics	8-43
Method Overloading	8-44
Using Method Overloading	8-45
Method Overloading and the Java API	8-47
Exercise 8-2: Overload a setItemFields Method, Part 1	8-48
Exercise 8-2: Overload a setItemFields Method, Part 2	8-49
Quiz	8-50
Summary	8-51
Practices Overview	8-52

9 Using Encapsulation

Interactive Quizzes	9-2
Objectives	9-3
Topics	9-4
What Is Access Control?	9-5
Access Modifiers	9-6
Access from Another Class	9-7
Another Example	9-8
Using Access Control on Methods	9-9
Topics	9-10
Encapsulation	9-11
Get and Set Methods	9-12
Why Use Setter and Getter Methods?	9-13
Setter Method with Checking	9-14
Using Setter and Getter Methods	9-15
Exercise 9-1: Encapsulate a Class	9-16
Topics	9-17
Initializing a Shirt Object	9-18
Constructors	9-19
Shirt Constructor with Arguments	9-20
Default Constructor and Constructor with Args	9-21
Overloading Constructors	9-22
Quiz	9-23
Exercise 9-2: Create an Overloaded Constructor	9-24

Summary 9-25
Practices Overview 9-26

10 More on Conditionals

Objectives 10-2
Topics 10-3
Review: Relational Operators 10-4
Testing Equality Between String variables 10-5
Common Conditional Operators 10-9
Ternary Conditional Operator 10-10
Using the Ternary Operator 10-11
Exercise 10-1: Using the Ternary Operator 10-12
Topics 10-13
Handling Complex Conditions with a Chained if Construct 10-14
Determining the Number of Days in a Month 10-15
Chaining if/else Constructs 10-16
Exercise 10-2: Chaining if Statements 10-17
Topics 10-18
Handling Complex Conditions with a switch Statement 10-19
Coding Complex Conditions: switch 10-20
switch Statement Syntax 10-21
When to Use switch Constructs 10-22
Exercise 10-3: Using switch Construct 10-23
Quiz 10-24
Topics 10-25
Working with an IDE Debugger 10-26
Debugger Basics 10-27
Setting Breakpoints 10-28
The Debug Toolbar 10-29
Viewing Variables 10-30
Summary 10-31
Practices Overview 10-32

11 Working with Arrays, Loops, and Dates

Objectives 11-2
Topics 11-3
Displaying a Date 11-4
Class Names and the Import Statement 11-5
Working with Dates 11-6
Working with Different Calendars 11-7
Some Methods of LocalDate 11-8

Formatting Dates	11-9
Exercise 11-1: Declare a LocalDateTime Object	11-10
Topics	11-11
Using the args Array in the main Method	11-12
Converting String Arguments to Other Types	11-13
Exercise 11-2: Parsing the args Array	11-14
Topics	11-15
Describing Two-Dimensional Arrays	11-16
Declaring a Two-Dimensional Array	11-17
Instantiating a Two-Dimensional Array	11-18
Initializing a Two-Dimensional Array	11-19
Quiz	11-20
Topics	11-21
Some New Types of Loops	11-22
Repeating Behavior	11-23
Coding a while Loop	11-24
A while Loop Example	11-25
while Loop with Counter	11-26
Coding a Standard for Loop	11-27
Standard for Loop Compared to a while loop	11-28
Standard for Loop Compared to an Enhanced for Loop	11-29
do/while Loop to Find the Factorial Value of a Number	11-30
Coding a do/while Loop	11-31
Comparing Loop Constructs	11-32
The continue Keyword	11-33
Exercise 11-3: Processing an Array of Items	11-34
Topics	11-35
Nesting Loops	11-36
Nested for Loop	11-37
Nested while Loop	11-38
Processing a Two-Dimensional Array	11-39
Output from Previous Example	11-40
Quiz	11-41
Topics	11-43
ArrayList Class	11-44
Benefits of the ArrayList Class	11-45
Importing and Declaring an ArrayList	11-46
Working with an ArrayList	11-47
Exercise 11-4: Working with an ArrayList	11-48
Summary	11-49
Practices Overview	11-50

12 Using Inheritance

Interactive Quizzes	12-2
Objectives	12-3
Duke's Choice Classes: Common Behaviors	12-4
Code Duplication	12-5
Inheritance	12-6
Inheritance Terminology	12-7
Topics	12-8
Implementing Inheritance	12-9
More Inheritance Facts	12-10
Clothing Class: Part 1	12-11
Shirt Class: Part 1	12-12
Constructor Calls with Inheritance	12-13
Inheritance and Overloaded Constructors	12-14
Exercise 12-1: Creating a Subclass, Part 1	12-15
Exercise 12-1: Creating a Subclass, Part 2	12-16
Topics	12-17
More on Access Control	12-18
Overriding Methods	12-19
Review: Duke's Choice Class Hierarchy	12-20
Clothing Class: Part 2	12-21
Shirt Class: Part 2	12-22
Overriding a Method: What Happens at Run Time?	12-23
Exercise 12-2: Overriding a Method in the Superclass	12-24
Topics	12-25
Polymorphism	12-26
Superclass and Subclass Relationships	12-27
Using the Superclass as a Reference	12-28
Polymorphism Applied	12-29
Accessing Methods Using a Superclass Reference	12-30
Casting the Reference Type	12-31
instanceof Operator	12-32
Exercise 12-3: Using the instanceof Operator, Part 1	12-33
Exercise 12-3: Using the instanceof Operator, Part 2	12-34
Topics	12-35
Abstract Classes	12-36
Extending Abstract Classes	12-38
Summary	12-39
Practice Overview	12-40

13 Using Interfaces

Objectives	13-2
Topics	13-3
The Object Class	13-4
Calling the <code>toString</code> Method	13-5
Overriding <code>toString</code> in Your Classes	13-6
Topics	13-7
The Multiple Inheritance Dilemma	13-8
The Java Interface	13-9
No Multiple Inheritance of State	13-10
Multiple Hierarchies with Overlapping Requirements	13-11
Using Interfaces in Your Application	13-12
Implementing the Returnable Interface	13-13
Access to Object Methods from Interface	13-14
Casting an Interface Reference	13-15
Quiz	13-16
Topics	13-18
What is This Feature?	13-19
Benefits	13-20
Where Can it be Used?	13-21
Why is The Scope So Narrow?	13-22
Exercise 13-1: Local Variable Type Inference	13-23
Topics	13-24
The Collections Framework	13-25
<code>ArrayList</code> Example	13-26
List Interface	13-27
Example: <code>Arrays.asList</code>	13-28
Exercise 13-2: Converting an Array to an <code>ArrayList</code> , Part 1	13-30
Exercise 13-2: Converting an Array to an <code>ArrayList</code> , Part 2	13-31
Topics	13-32
Example: Modifying a List of Names	13-33
Using a Lambda Expression with <code>replaceAll</code>	13-34
Lambda Expressions	13-35
The Enhanced APIs That Use Lambda	13-36
Lambda Types	13-37
The UnaryOperator Lambda Type	13-38
The Predicate Lambda Type	13-39
Using <code>var</code> With Lambda Expressions	13-40
Exercise 13-3: Using a Predicate Lambda Expression	13-41
Summary	13-42
Practice Overview	13-43

14 Handling Exceptions

- Objectives 14-2
- Topics 14-3
- What Are Exceptions? 14-4
- Examples of Exceptions 14-5
- Code Example 14-6
- Another Example 14-7
- Types of Throwable classes 14-8
- Error Example: OutOfMemoryError 14-9
- Quiz 14-10
- Topics 14-11
- Normal Program Execution: The Call Stack 14-12
- How Exceptions Are Thrown 14-13
- Topics 14-14
- Working with Exceptions in NetBeans 14-15
- The try/catch Block 14-16
- Program Flow When an Exception Is Caught 14-17
- When an Exception Is Thrown 14-18
- Throwing Throwable Objects 14-19
- Uncaught Exception 14-20
- Exception Printed to Console 14-21
- Summary of Exception Types 14-22
- Exercise 14-1: Catching an Exception 14-23
- Quiz 14-24
- Exceptions in the Java API Documentation 14-25
- Calling a Method That Throws an Exception 14-26
- Working with a Checked Exception 14-27
- Best Practices 14-28
- Bad Practices 14-29
- Somewhat Better Practice 14-30
- Topics 14-31
- Multiple Exceptions 14-32
- Catching IOException 14-33
- Catching IllegalArgumentException 14-34
- Catching Remaining Exceptions 14-35
- Summary 14-36
- Practices Overview 14-37

15 Deploying and Maintaining the Soccer Application

- Interactive Quizzes 15-2
- Objectives 15-3
- Topics 15-4
- Packages 15-5
 - Packages Directory Structure 15-6
 - Packages in NetBeans 15-7
 - Packages in Source Code 15-8
- Topics 15-9
- SoccerEnhanced.jar 15-10
- Set Main Class of Project 15-11
- Creating the JAR File with NetBeans 15-12
- Topics 15-14
 - Client/Server Two-Tier Architecture 15-15
 - Client/Server Three-Tier Architecture 15-16
- Topics 15-17
 - Client/Server Three-Tier Architecture 15-18
 - Different Outputs 15-20
 - The Soccer Application 15-21
 - IDisplayDataItem Interface 15-22
 - Running the JAR File from the Command Line 15-23
 - Text Presentation of the League 15-24
 - Web Presentation of the League 15-25
- Topics 15-26
 - Enhancing the Application 15-27
 - Adding a New GameEvent Kickoff 15-28
 - Game Record Including Kickoff 15-29
- Summary 15-30

16 Understanding Modules

- Objectives 16-2
- Topics 16-3
- Reusing Code in Java: Classes 16-4
- Reusing Code in Java: Packages 16-5
- Reusing Code in Java: Programming in the Large 16-6
- What Is a Module? 16-7
- Example: java.base Module 16-8
- Module System 16-9
- Modular Development in JDK 9 16-10
- Topics 16-11
- JARs 16-12

JAR Files and Distribution Issues	16-13
Class Path Problems	16-14
Example JAR Duplicate Class Problem 1	16-15
Example JAR Duplicate Class Problem 2	16-16
Module System: Advantages	16-17
Accessibility Between Classes	16-18
Access Across Non-Modular JARs	16-19
Access Across Modules	16-20
Topics	16-21
module-info.java	16-22
Example: module-info.java	16-23
Creating a Modular Project	16-24
exports Module Directive	16-25
exports...to Module Directive	16-26
requires Module Directive	16-27
requires transitive Module Directive	16-28
Implementing a Requires Transitively Relationship	16-29
Summary of Keywords	16-30
Compiling Modules	16-31
Running a Modular Application	16-32
Topics	16-33
The JDK	16-34
The Modular JDK	16-35
Listing the Modules in JDK 9	16-36
Java SE Modules	16-37
The Base Module	16-38
Summary	16-39
Practice Overview	16-40

17 JShell

Objectives	17-2
Topics	17-3
A Million Test Classes and Main Methods	17-4
JShell Provides a Solution	17-5
Topics	17-6
Comparing Normal Execution with REPL	17-7
Getting Started with JShell and REPL	17-8
Scratch Variables	17-9
Declaring Traditional Variables	17-10
Code Snippets	17-11
Completing a Code Snippet	17-12

Tab Completion and Tab Tips	17-13
Semicolons	17-14
JShell Commands	17-15
Importing Packages	17-16
Quiz 17-1	17-17
Topics	17-18
Why Incorporate JShell in an IDE?	17-19
Use Cases	17-20
Two Ways to Open JShell in NetBeans	17-21
Summary	17-22
Practices	17-23

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

Introduction

1



ORACLE®



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Moisés Ocampo Sámano (aos_moyer@hotmail.com) has a
non-transferable license to use this Student Guide.

Audience

- Beginners to programming who have basic mathematical, logical, and analytical problem-solving skills and who want to begin learning the Java programming language
- Novice programmers and those programmers who prefer to start learning the Java programming language at an introductory level
- Students who beginning their study to become an Oracle Certified Professional (OCP)
 - Java SE Programmer I Exam (this course)
 - Java SE Programmer II Exam (the next course)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Introductions

Meet your classmates and briefly introduce yourself:

- Name
- Title or position
- Company
- Experience with Java programming and Java applications
- Reasons for attending



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Course Objectives

After completing this course, you should be able to:

- Demonstrate knowledge of basic programming language concepts
- Demonstrate knowledge of the Java programming language
- Implement intermediate Java programming and object-oriented (OO) concepts



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Demonstrate knowledge of basic programming language concepts.

- Source code versus machine code
- Platform dependence and platform independence
- The use of APIs and libraries

Demonstrate knowledge of the Java programming language

- Compile and run a Java program from both the command line and from NetBeans.
- Create a Java class with fields and methods.
- Declare and use arrays.
- Use methods of the `StringBuilder`, `String`, and `ArrayList` classes.
- Display and manipulate dates using one or two classes from the new `java.time` package.
- Write conditional statements.
- Write loop statements (enhanced for, for, while, do/while), as well as nested loops.
- Implement a `try` block to handle exceptions.

Implement intermediate Java programming and object-oriented (OO) concepts

- Instantiate an object and invoke its methods
- Explain how objects vs. primitive types or references are stored in memory
- Create an inheritance hierarchy of Java classes by creating a subclass or implementing a Java Interface
- Overload a method and a constructor
- Encapsulate the fields of a class and use modifiers to control access to a field or a method
- Create superclasses, abstract classes, and Interfaces and use them as reference types
- Use a Predicate lambda expression as the argument to a method

Schedule

Day One

- Getting Started
 - Lesson 1: Introduction
 - Lesson 2: What Is a Java Program?
- The Basic Shopping Cart
 - Lesson 3: Creating a Java Main Class
 - Lesson 4: Data in a Cart
 - Lesson 5: Managing Multiple Items



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Schedule

Day Two

- Filling the Cart
 - Lesson 6: Describing Objects and Classes
 - Lesson 7: Manipulating and Formatting the Data in Your Program
- Improving Cart Efficiency
 - Lesson 8: Creating and Using Methods

Day Three

- Lesson 9: Using Encapsulation
- Expanding the Business
 - Lesson 10: More on Conditionals
 - Lesson 11: Working with Arrays, Loops, and Dates



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Schedule

Day Four

- Lesson 12: Using Inheritance
- Lesson 13: Using Interfaces
- Lesson 14: Handling Exceptions

Day Five

- Lesson 15: Deploying and Maintaining the Soccer Application
- Lesson 16: Understanding Modularity
- Lesson 17: JShell

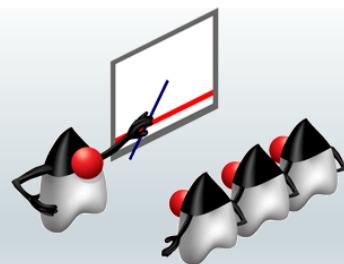


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Lesson Format

Lecture / Student Guide (50%)

- Traditional slides
- Sample code
- Exercises
- Quizzes & interactive quizzes



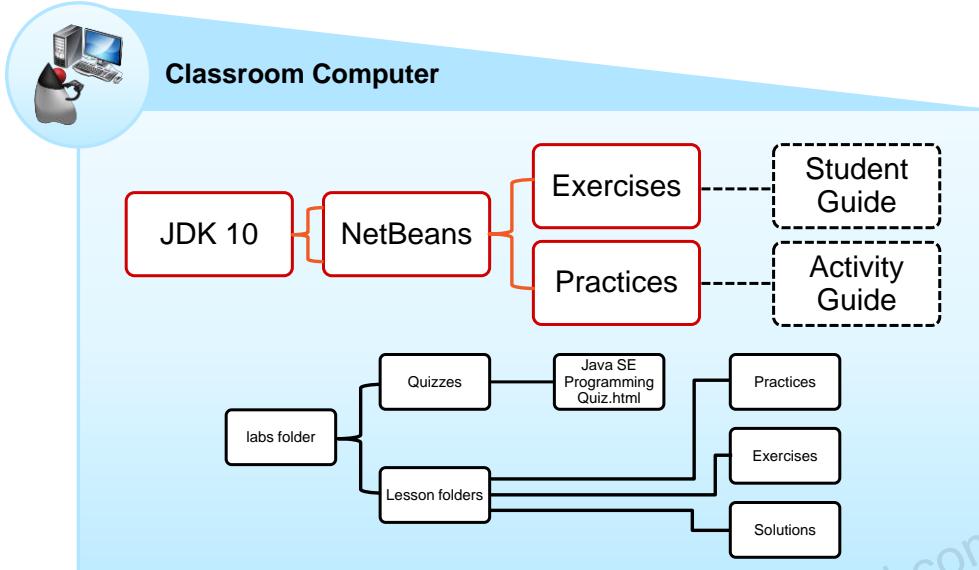
Practices / Activity Guide (50%)

- Hands-on learning
- Work with Java code
- Larger-scale labs
- Intended for the OU Practice Environment



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Course Environment



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

How Do You Learn More After the Course?

- In the Oracle Learning Library, there is a list of resources that you can use to learn more about Java programming. Look for the collection on the oracle.com/oll/java page.
- *Oracle Learning Library:*
 - <http://www.oracle.com/goto/oll>



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Additional Resources

Resource	Website
Education and Training	http://education.oracle.com
Product Documentation	http://www.oracle.com/technology/documentation
Product Downloads	http://www.oracle.com/technology/software
Product Articles	http://www.oracle.com/technology/pub/articles
Product Support	http://www.oracle.com/support
Product Forums	http://forums.oracle.com
Product Tutorials	http://www.oracle.com/technology/obe
Sample Code	http://www.oracle.com/technology/sample_code



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The table in the slide lists web resources where you can obtain additional information about Java.

Additional Resources

Resource	Website
Java Documentation	https://docs.oracle.com/javase
API Documentation	https://docs.oracle.com/javase/10/docs/api/index.html



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The table in the slide lists web resources where you can obtain additional information about Java.

Summary

In this lesson, you reviewed the course objectives and the tentative class schedule. You met your fellow students, and you saw an overview of the computer environment that you will use during the course.

Enjoy the next five days of *Java SE Programming I*



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

What Is a Java Program?

2



ORACLE®



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Moisés Ocampo Sámano (aos_moyra@hotmail.com) has a
non-transferable license to use this Student Guide.

Objectives

After completing this lesson, you should be able to:

- Contrast the terms “platform-dependent” and “platform-independent”
- Describe the purpose of the JVM
- Explain the difference between a procedural program and an object-oriented program
- Describe the purpose of `javac` and `java` executables
- Verify the Java version on your system
- Run a Java program from the command line



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environments
- Running and testing a Java program

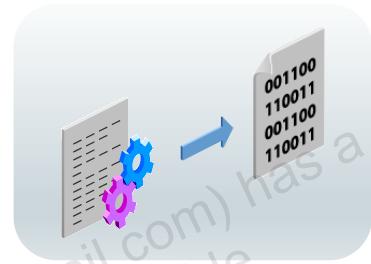


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Purpose of a Computer Program

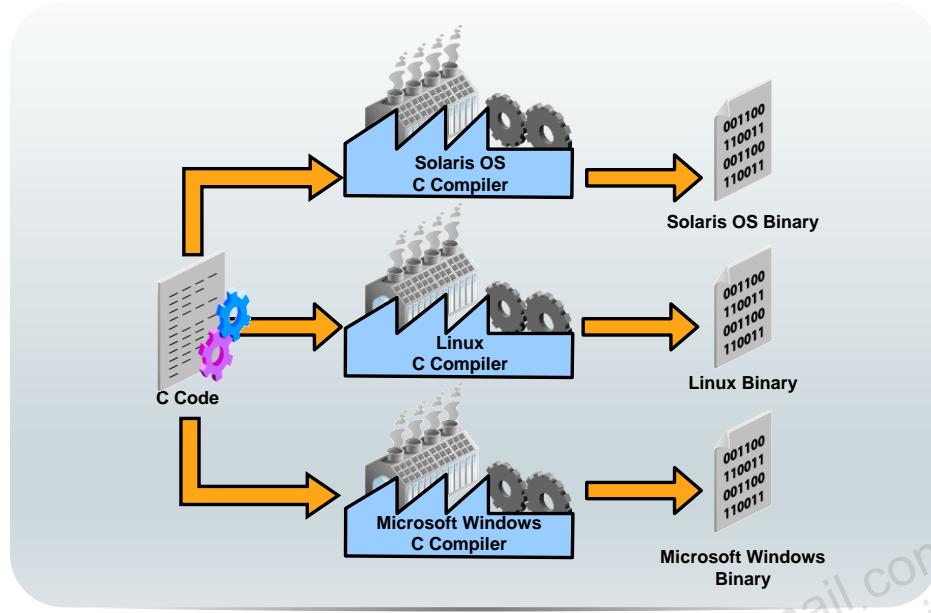
A computer program is a set of instructions that run on a computer or other digital device.

- At the machine level, the program consists of binary instructions (1s and 0s).
 - Machine code
- Most programs are written in *high-level* code (readable).
 - Must be translated to machine code



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Translating High-Level Code to Machine Code

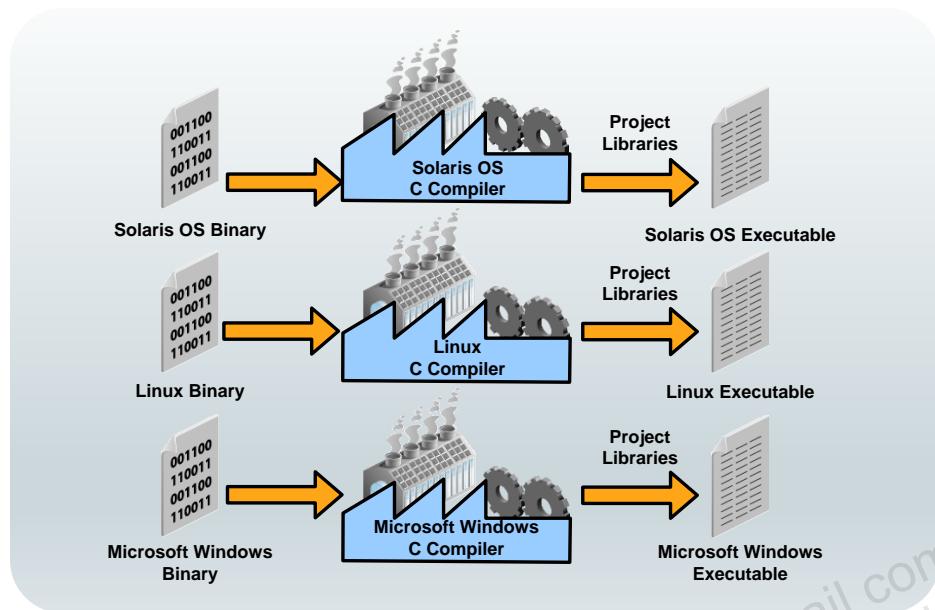


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Programs written in most languages usually require numerous modifications to run on more than one type of computing platform, (a combination of a CPU and operating system). This platform-dependence is because most languages require you to write code specific to the underlying platform. Popular programming languages, such as C and C++, require programmers to compile and link their programs, resulting in an executable program unique to a platform. A compiler is an application that converts a program that you write into a CPU-specific code called *machine code*. These platform-specific files (binary files) are often combined with other files, such as libraries of prewritten code, using a linker to create a platform-dependent program, called an *executable*, which can be executed by an end user. Unlike C and C++, the Java programming language is platform-independent.

The image illustrates how a compiler creates a binary file.

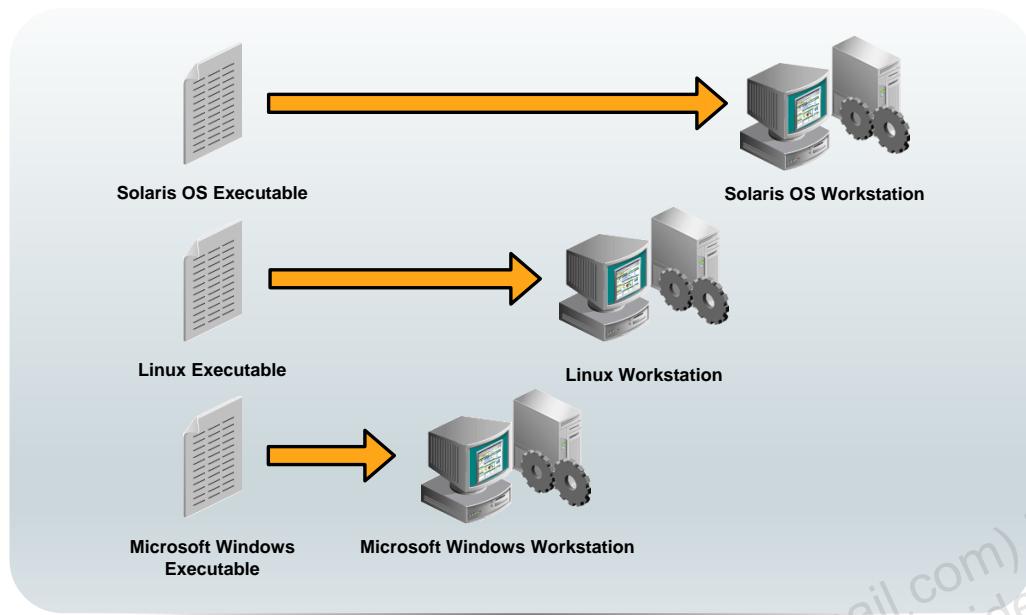
Linked to Platform-Specific Libraries



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The image illustrates how a binary file is linked with libraries to create a platform-dependent executable.

Platform-Dependent Programs



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The image illustrates how platform-dependent executables can execute only on one platform.

Topics

- Introduction to computer programs
- **Introduction to the Java language**
- Verifying the Java development environment
- Running and testing a Java program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Key Features of the Java Language

Some of the features that set Java apart from most other languages are that:

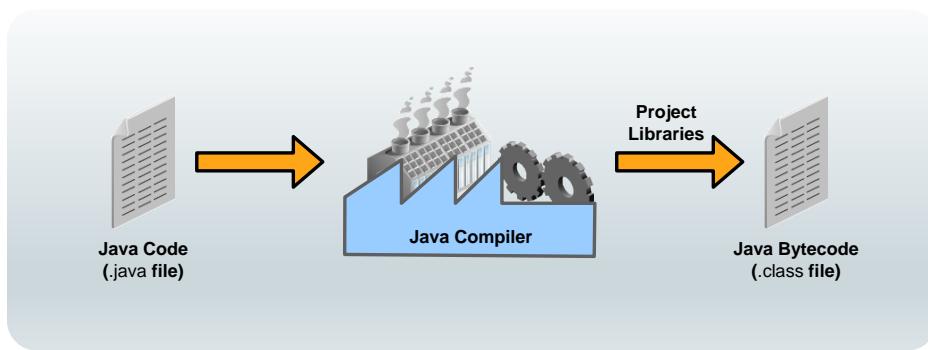
- It is platform-independent
- It is object-oriented



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

There are several other key features of the Java language, but in this course, only the two mentioned above will be discussed.

Java Is Platform-Independent



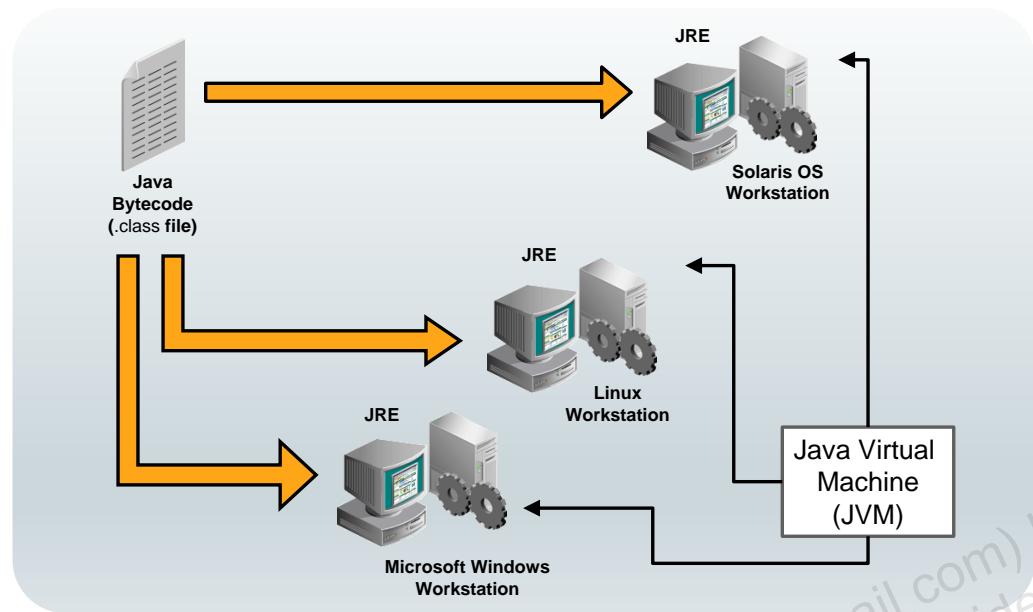
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A Java program can run on several different CPUs and operating system combinations, such as the Solaris OS on a SPARC chip, Mac OS X on an Intel chip, and Microsoft Windows on an Intel chip, usually with few or no modifications.

As illustrated above, Java programs are compiled using a Java compiler. The resulting format of a compiled Java program is platform-independent Java bytecode instead of CPU-specific machine code.

After the bytecode is created, it is interpreted by a bytecode interpreter called the Java Virtual Machine or JVM. A virtual machine is a platform-specific program that understands platform-independent bytecode and can execute it on a particular platform. For this reason, the Java programming language is often referred to as an interpreted language, and Java technology programs are said to be portable or executable on any platform. Other interpreted languages include Perl.

Java Programs Run In a Java Virtual Machine



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The image illustrates a Java bytecode file executing on several platforms where a Java runtime environment exists.

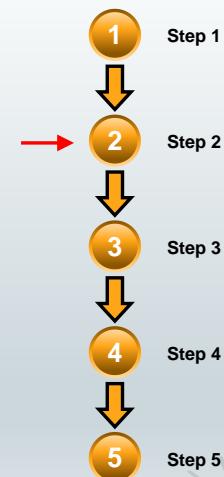
A virtual machine gets its name because it is a piece of software that runs code, a task usually accomplished by the CPU or hardware machine. For Java programs to be platform-independent, a virtual machine called the JVM is required on every platform where your program will run. The JVM is responsible for interpreting Java code, loading Java classes, and executing Java programs.

However, a Java program needs more than just a JVM to execute. A Java program also needs a set of standard Java class libraries for the platform. Java class libraries are libraries of prewritten code that can be combined with the code that you write to create robust applications.

Combined, the JVM software and Java class libraries are referred to as the Java Runtime Environment (JRE). Java Runtime Environments are available from Oracle for many common platforms.

Procedural Programming Languages

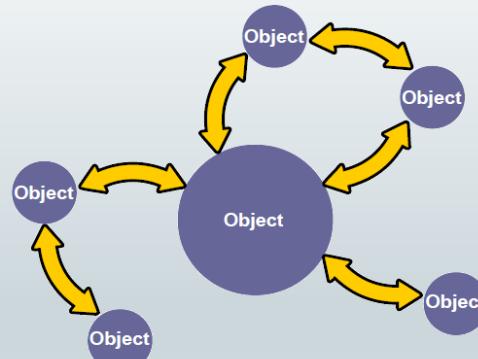
- Many early programming languages followed a paradigm called *Procedural Programming*.
- These languages use a sequential pattern of program execution.
- Drawbacks to procedural programming:
 - Difficult to translate real-world use cases to a sequential pattern
 - Difficult to maintain programs
 - Difficult to enhance as needed



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java Is an Object-Oriented Language

- Interaction of objects
- No prescribed sequence
- Benefits:
 - Modularity
 - Information hiding
 - Code reuse
 - Maintainability



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Object-oriented programming differs from procedural programming, because procedural programming stresses the sequence of coding steps required to solve a problem, whereas object-oriented programming stresses the interaction of objects. Java is an object-oriented programming (OO) language. One of the main goals of an OO language is to create objects—pieces of autonomous code—that can interact with other objects to solve a problem. OO programming languages began in 1967 and have led to popular programming languages such as C++, upon which Java is loosely based.

This provides many benefits:

- **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. After it is created, an object can be easily passed around inside the system.
- **Information hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- **Code reuse:** If an object already exists (perhaps written by another software developer), you can use that object in your program.
- **Maintainability:** If a particular object is found to be problematic, you can create another, slightly modified one and simply replace the original one in your application. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace the bolt, not the entire machine.

The diagram illustrates an object-oriented program's focus on objects and object interactions.

Topics

- Introduction to computer programs
- Introduction to the Java language
- **Verifying the Java development environment**
- Running and testing a Java program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Verifying the Java Development Environment

1. Download and install the Java Development Kit (JDK) from oracle.com/java.
2. Explore the Java Help menu.
3. Compile and run a Java application by using the command line.

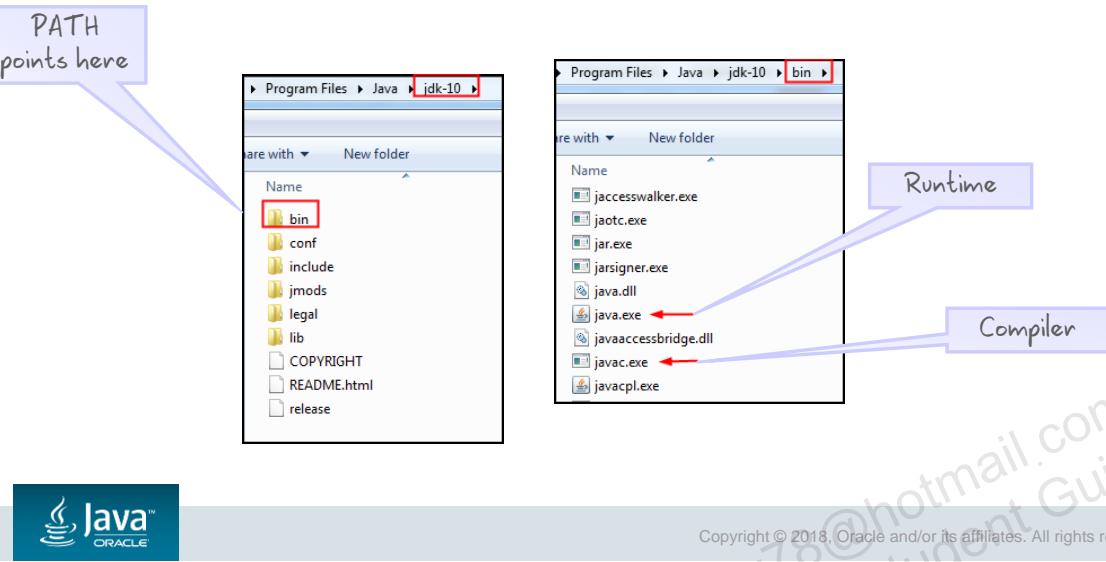


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Setting up your Java development environment is a simple task. The JDK is available for free from the Oracle Java website.

- After you have installed the JDK, you can explore the Java environment by typing some commands at the command line. For example, open a terminal window and enter `java`.
- Review the command options displayed.
- Enter `java -version` to see what Java version is installed on your system.
- Compile and run a Java application using the command line.

Examining the Installed JDK: The Tools



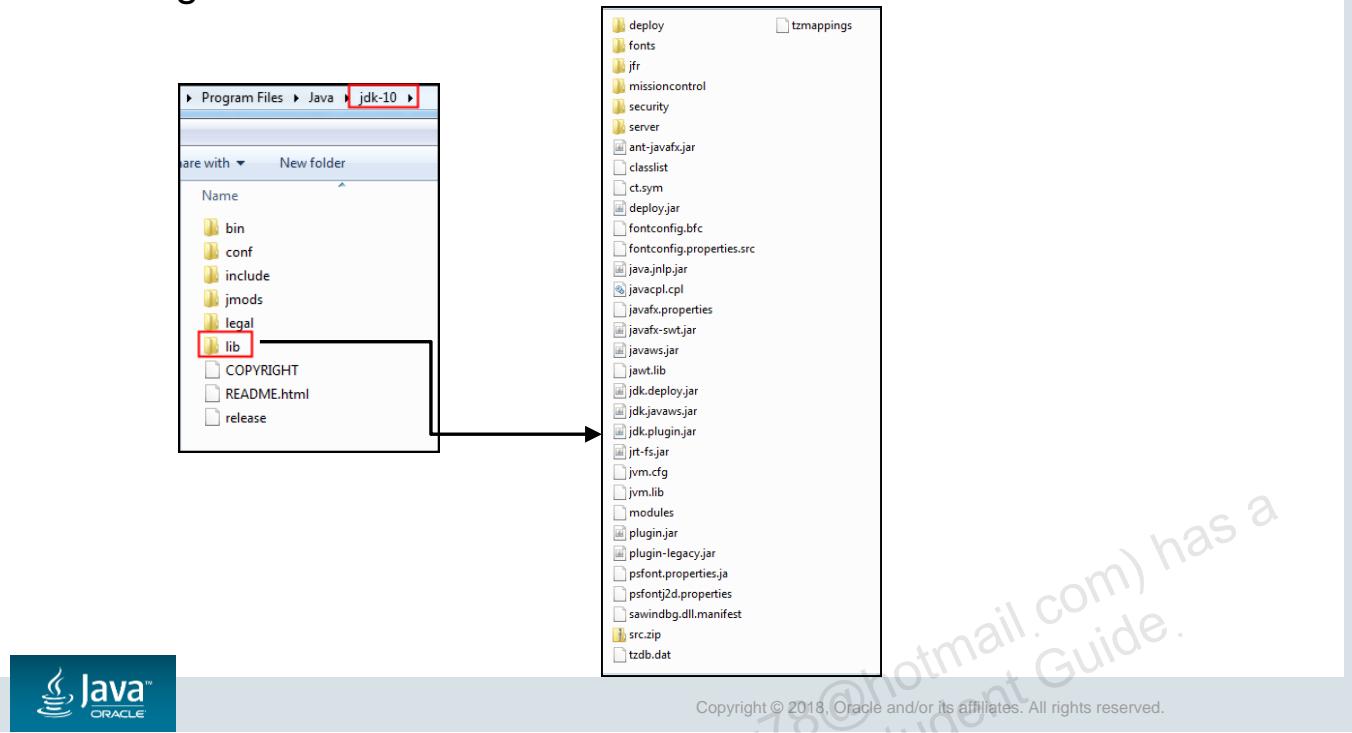
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java SE (Standard Edition) Development Kit

The Java SE Development Kit includes both the tools and classes that you will use to develop a Java program. The tools and utilities are stored in the `bin` directory. These are shown in the screenshot on the right. They include:

- A Java Virtual Machine (JVM) for the platform you choose. Here you see a Windows example. The runtime engine is started by running the `java` program.
- A Java compiler, started by running the `javac` program
- Additional utilities, such as utilities for creating Java archive files (`JAR` files) and for debugging Java programs
- The `bin` directory, which must be on the system PATH in order to run or compile a Java program. The Java installer automatically adds the `bin` to your system PATH.
- **Note:** The Java Runtime Environment used in *production* (commonly called the JRE) is also included with Java SE Development Kit. This is found in the `jre` directory.

Examining the Installed JDK: The Libraries



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java SE Development Kit

In addition to the executable files found in the `bin` directory of the JDK, various class libraries are installed that conform to the particular platform that you chose. Here you see a Windows example. The core libraries are found in the `lib` directory as shown above.

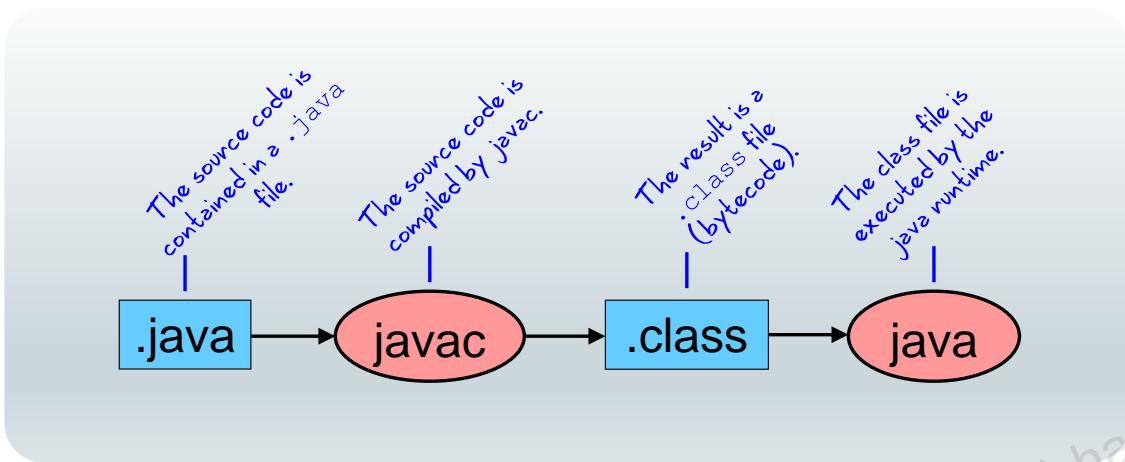
Topics

- Introduction to computer programs
- Introduction to the Java language
- Verifying the Java development environment
- Running and testing a Java program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Compiling and Running a Java Program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Compiling a Program

1. Go to the directory where the source code files are stored.
2. Enter the following command for each `.java` file you want to compile.

- Syntax:

```
javac <filename>
```

- Example:

```
javac SayHello.java
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Compiling converts the source files that you write into bytecode that can be executed by a Java Virtual Machine. The source file has a `.java` extension. It also defines a public class of the same name. For example, the class, `SayHello`, must be saved in a file called `SayHello.java`. (You learn more about classes later in this course.)

To compile the `SayHello` source code, perform the following steps:

1. Go to the directory where the source code files are stored.
2. Enter the following command for each `.java` file that you want to compile (Note that the `.java` extension is required.):

Example: `javac SayHello.java`

After the compilation has finished, and assuming no compilation errors have occurred, you should have a new file called `<classname>.class` in your directory for each source code file that you compiled.

Example: `SayHello.class`

Executing (Testing) a Program

1. Go to the directory where the class files are stored.
2. Enter the following for the class file that contains the main method:

- Syntax:

```
java <classname>
```

- Example: *Do not specify .class.*

```
java SayHello
```

- Output:

```
Hello World!
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When you have successfully compiled your source code files, you can execute and test them using the Java Virtual Machine.

To execute and test your program:

1. Go to the directory where the class files are stored.
2. Enter the following command for the class file that contains the `main` method. Note that here the file extension (`.class`) should *not* be included.

Example: `java SayHello`

This command runs the `SayHello` class. The `SayHello` class contains the `main` method. This is the entry point to a Java application. The `java` executable only works with a class containing a `main` method. In the above example, the `main` method contains code that prints the string "Hello World!".

Output for a Java Program

A Java program can output data in many ways. Here are some examples:

- To a file or database
- To the console
- To a webpage or other user interface



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this course, we will be outputting data only to the console. You can learn more about writing to other destinations, such as a file, database, or webpage, by taking the *Java SE Programming II* course.

Exercise 2-1

- From a Terminal window, enter `java -version` to see the system's Java version.
- Look for `SayHello.java` in:
`/labs/02-GettingStarted/Exercises/Exercise1`
- Compile it: `javac SayHello.java`
- Run the resulting class file: `java SayHello`
 - Did you see the output?



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you look at the Java version installed on your system, and then you run a simple Java program from the command line.

- Open a terminal window by double-clicking the Terminal shortcut on your desktop. It will open at your home directory, which is `/home/oracle`.
 - **Note:** A handy shortcut to navigate to your home directory from anywhere is `~`. Example: `cd ~` to go to `/home/oracle`.
- Enter `java` to see the available command options.
- Enter `java -version` to verify the version of Java installed on your system.
- Navigate to the folder containing the Java source file for this exercise:
`cd labs/02-GettingStarted/Exercises/Exercise1`
- Enter `javac SayHello.java` to compile it.
- Enter `java SayHello` to run it. You should see a "Hello World!" message as output.

Quiz



Which of the following is correct? (Choose all that apply.)

- a. javac OrderClass
- b. java OrderClass
- c. javac OrderClass.java
- d. java OrderClass.java



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

The .java extension is needed only when you compile a class (using javac).

Summary

In this lesson, you should have learned how to:

- Describe the distinction between high-level language and machine code
- Describe what platform-independence means
- Describe how a Java program is compiled and to what format
- Explain what it means to say that Java is an object-oriented language
- Determine the version number of a Java install
- Use the `javac` tool to compile Java source code and the `java` tool to run or test your program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

Creating a Java Main Class

3



ORACLE®



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Moisés Ocampo Sámano (aos_moyra@hotmail.com) has a
non-transferable license to use this Student Guide.

Objectives

After completing this lesson, you should be able to:

- Use the NetBeans IDE to create and test Java classes
- Write a `main` method
- Use `System.out.println` to write a String literal to system output



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Java classes and packages
- The `main` method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Java Classes

A Java class is the building block of a Java application.

ShoppingCart.java

Includes code that:

- Allows a customer to add items to the shopping cart
- Provides visual confirmation to the customer



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Program Structure

- A class consists of:
 - The class name. Class names begin with a capital letter.
 - The body of the class surrounded with braces { }
 - Data (called fields)
 - Operations (called methods)
- Example:

Java is case-sensitive!

```
public class Hello {  
    // fields of the class  
    // methods  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- A class is declared using the keyword, `class`, followed by the class name.
- Convention dictates that the class name start with a capital letter. If there are two words in the class name (SayHello), each word should begin with a capital letter. In the example above, the class name is `Hello`.
- The keyword `public` is called a *modifier*. You learn about these in the lesson titled “Using Encapsulation.”
- **Java is case-sensitive.** It does not recognize the following two words as being the same thing: `class` and `Class`.
- A class would typically contain data (called fields) and operations (called methods). You learn about this a little later.
- Notice that the body of the `Hello` class is enclosed in braces ({}).

Java Packages

- A package provides a namespace for the class.
 - This is a folder in which the class will be saved.
 - The folder name (the package) is used to uniquely identify the class.
 - Package names begin with a lowercase letter.
- Example:

```
package greeting;  
  
public class Hello {  
    // fields and methods here  
}
```

Package name

The class's unique name is:
greeting.Hello

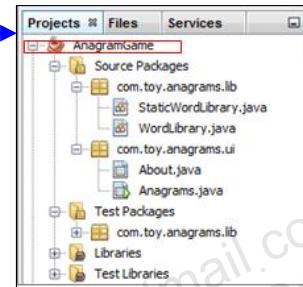


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java IDEs

A Java Integrated Development Environment (IDE) is a type of software that makes it easier to develop Java applications.

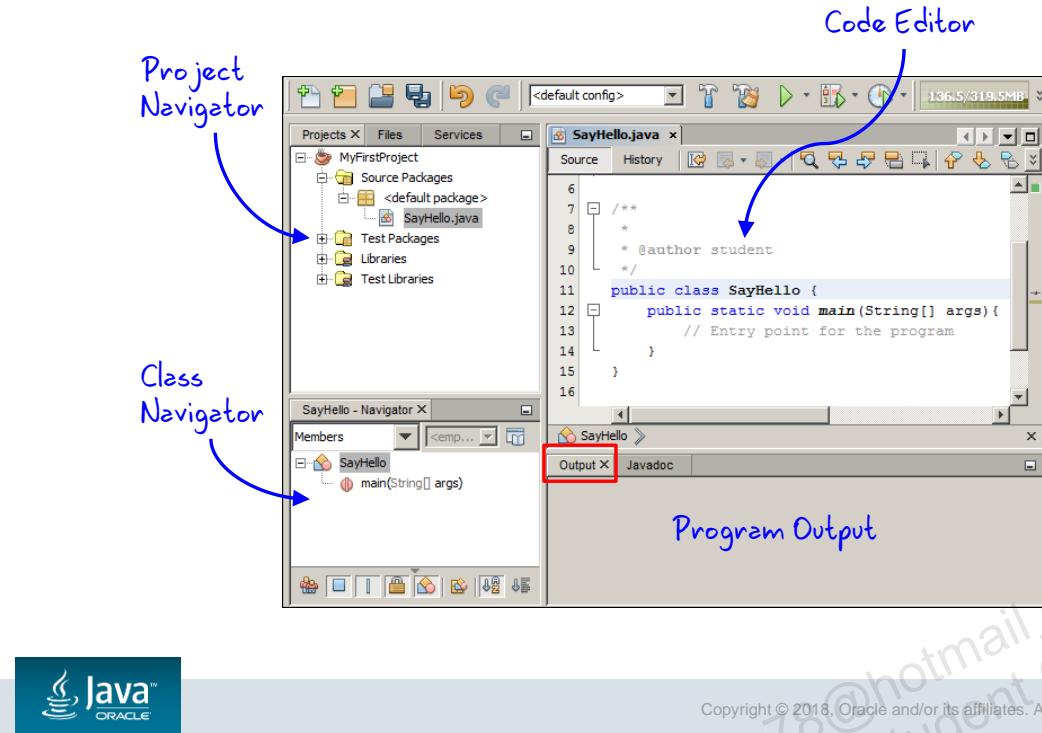
- An IDE provides:
 - Syntax checking
 - Various automation features
 - Runtime environment for testing
- It enables you to organize all your Java resources and environment settings into a *Project*.
- Projects contain packages.
- Packages contain files, such as .java.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Some well-known Java IDEs are NetBeans (used in this class to perform the practices and exercises), Eclipse, and JDeveloper.

The NetBeans IDE



The Java project provides a mechanism by which you can organize all of the source and class files and other resources (connection profiles, configuration information, and so on) required by the Java application.

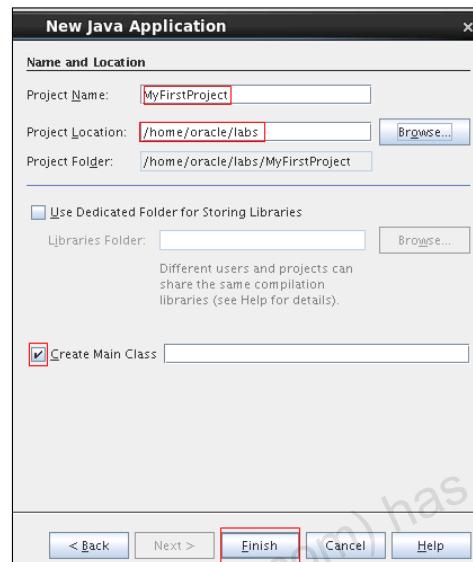
- When you begin working in NetBeans, you either create a project or open an existing one.
- The Project Navigator gives you a visual representation of the project contents.
- You can open files from your project in the code editor by double-clicking the file or using the context menu.

When you select a class within the project, the structure of that class is displayed in the Class Navigator, shown in the lower left part of the NetBeans window.

When you run a file or the entire Java program, any program output appears in the Output panel in the lower right part of the window.

Creating a Java Project

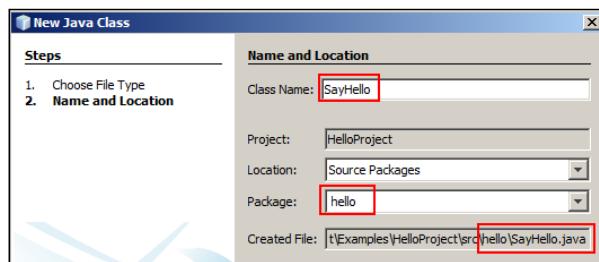
1. Select **File > New Project**.
2. Select Java Application.
3. Name and set the location for the project.
4. Select “Create Main Class” if you want it done for you automatically.
5. Click **Finish**.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Creating a Java Class

1. Select **File > New File**.
2. Select your project and choose **Java Class**.
3. Name the class.
4. Assign a package.
5. Click **Finish**.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

To create a class within your new project, perform the following steps:

1. Select **File > New File** from the menu.
2. On the first page of the New File Wizard, select your project, and then accept the default file type of Java Class. Click **Next**.
3. On the next page of the wizard, enter a name for the Java class. By convention, Java classes should start with an uppercase letter and each subsequent word in the class name should be capitalized (for example, `SayHello`). This is illustrated in the screenshot above.
4. Assign a package for the class.
5. Click **Finish**.

Note: If the package for this new class already exists, you can create the class by right-clicking the package in the Project Navigator panel in NetBeans and selecting **New > Java class** from the context menu instead of starting from the File menu.

Exercise 3-1:Creating a New Project and Java Class

In this exercise, you use NetBeans to create a new Java Class.

1. Create a new project called **Exercise_03-1**.

- Deselect the box to create the `main` method. You will write the `main` method yourself in the next exercise.

2. Create a new **Java Class** file in this project.

- Class name = `ShoppingCart`
- Package name =`exercise`



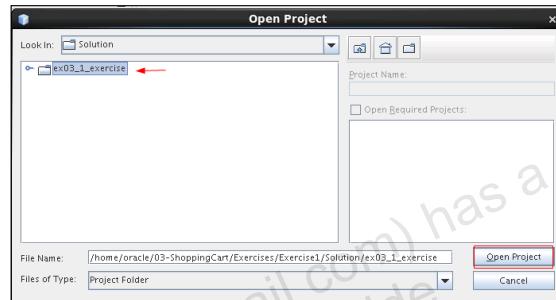
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The fully-qualified class name should be `exercise.ShoppingCart`. Note: You won't be able to run and test your code until create the main method in the next exercise.

Opening an Existing Java Project

If you need to open an existing project in NetBeans, perform the following steps:

1. Select **File > Open Project**.
2. Navigate to the directory that contains your projects.
3. Select the project file you want. (This file must be unzipped.)
4. Click **Open Project**.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

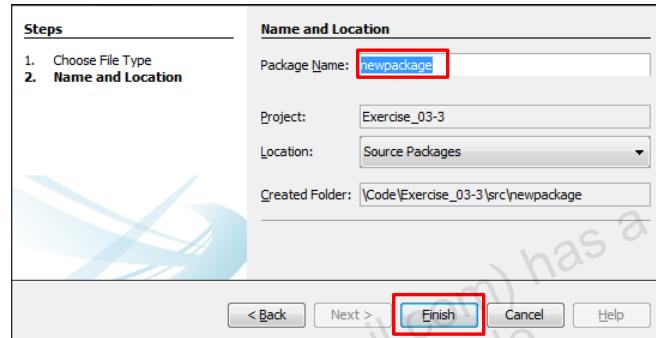
To open an existing project, perform the following steps:

1. Select **File > Open Project**.
2. Navigate to the directory that contains your projects.
3. Select the project file you want. (This file must be unzipped.)
4. Click **Open Project**.

Creating a New Java Package

If you ever need to create a new package, perform the following steps in NetBeans:

1. Right-click your project.
2. Select **New > Java Package**.
3. Name the package.
4. Click **Finish**.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

To create a new package within your new project, perform the following steps:

1. Right-click your project.
2. Select **New > Java Package**.
3. Name the package.
4. Click **Finish**.

Topics

- Java classes and packages
- The `main` method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



The main Method

- It is a special method that the JVM recognizes as the starting point for every Java program.
- The syntax is always the same:

```
public static void main (String[] args) {  
    // code goes here in the code block  
}
```

- It surrounds entire method body with braces { } .



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- The main method is a special method that the Java Virtual Machine recognizes as the starting point for a Java program.
- Any program that you want to run must have a public `main` method.
- A class containing a main method is referred to as a “main class.”

Note: Brackets ([]) can be placed to the right of String or to the right of args, but the former is recommended:

```
(String[] args)  
(String args[])
```

A main Class Example

```
public class Hello {  
  
    public static void main (String[] args) {  
        // Entry point to the program.  
        // Write code here:  
        System.out.println ("Hello World!");  
    }  
}
```

main
method

Class name

Comments

Program output



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Here you see a simple example of a class (Hello) that includes a `main` method. The `main` method writes a message to the console ("Hello World!"). This is called *program output*.

You can include comments that the compiler will ignore, by preceding the comment line with two forward slashes: //

Output to the Console

- Syntax:

```
System.out.println (<some string value>);
```

- Example:

```
System.out.println ("This is my message.");
```

String literal

Be sure to include the
semicolon

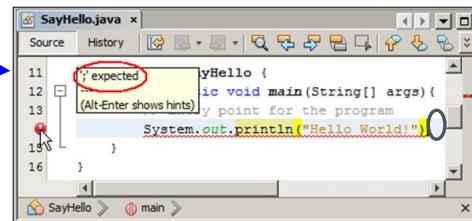


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Use the `System.out.println` method to print a message to the console. Use double quotation marks to enclose the text of the message (called a String literal).

Avoiding Syntax Errors

- NetBeans will tell you if you have done something wrong.
- Common errors include:
 - Unrecognized word (check for case-sensitivity error)
 - Missing close quotation mark
 - Unmatched brace
 - Missing semicolon



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

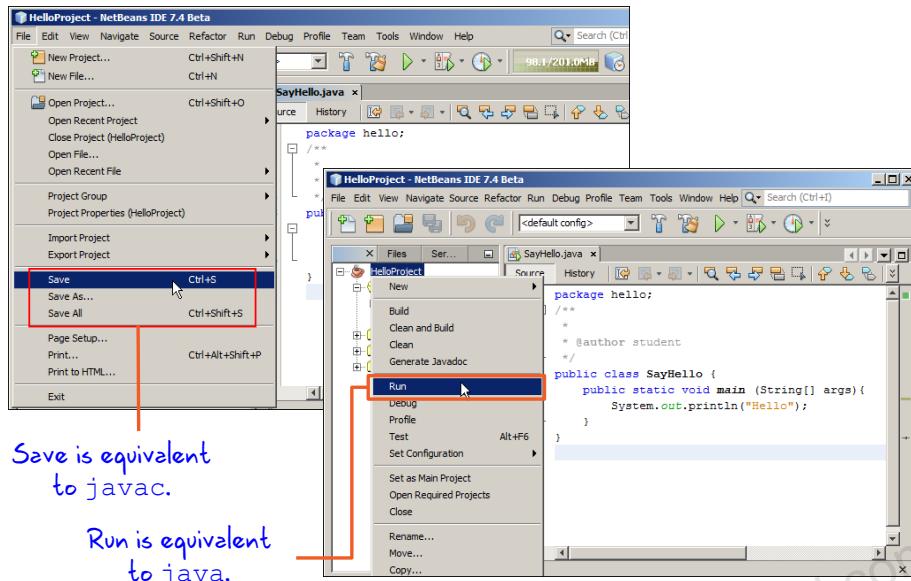
Most Java editors check the code syntax and show alerts by using icons and red underlines where there are errors in the code.

To avoid syntax problems, be sure to do the following:

- Observe any red bubble indicators in the code editor to locate syntax errors.
- Have a semicolon at the end of every line where one is required.
- Have an even number of symbols such as braces, brackets, and quotation marks.

The screenshot shows an error in Line 13, in which there is a missing semicolon. If you place your cursor over the red bubble, the editor offers a suggestion for fixing the error.

Compiling and Running a Program by Using NetBeans



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Exercise 3-2: Creating a `main` Method

In this exercise, you manually enter a `main` method that prints a message to the console.

1. Continue editing `Exercise_03-1` or open `Exercise_03-2`.
2. In the code editor, add the `main` method structure to the `ShoppingCart` class.
3. In the code block of the `main` method, use a `System.out.println` method to print "Welcome to the Shopping Cart!"
4. Save your program.
5. Click the **Run** button to test program. 

- Select `exercise.ShoppingCart` as the main class.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you manually enter a `main` method that prints a message to the console.

Quiz



Which main method syntax is correct?

- a. Public static void main (String[] args){ }
- b. public Static void Main (String[] args){ }
- c. public static void main (String () args)[]
- d. public static void main (String[] args){ }



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: d

- a is incorrect. It should be “public”, not “Public”.
- b is incorrect. Both “Static” and “Main” should begin with a lowercase letter.
- c is incorrect because there should be brackets following “String” and braces defining the method scope.
- d is correct.

Summary

In this lesson, you should have learned how to:

- Use the NetBeans IDE to create and test Java classes
- Write a `main` method
- Use `System.out.println` to write a String literal to system output



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Data in a Cart



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Objectives

After completing this lesson, you should be able to:

- Describe the purpose of a variable in the Java language
- List and describe four data types
- Declare and initialize `String` variables
- Concatenate `String` variables with the '+' operator
- Make variable assignments
- Declare and initialize `int` and `double` variables
- Modify variable values by using numeric operators
- Override default operator precedence using ()



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Topics

- Introducing variables
- Working with String variables
- Working with numbers
- Manipulating numeric data

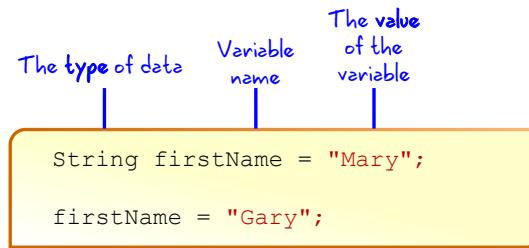


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Variables

- A variable refers to something that can change.
 - Variables can be initiated with a value.
 - The value can be changed.
 - A variable holds a specific type of data.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A variable is simply a storage location in memory that holds a specific value. That value can be changed by copying (or “assigning”) a different value to that variable.

Variable Types

- Some of the types of values a variable can hold:
 - `String` (example: "Hello")
 - `int` (examples: -10, 0, 2, 10000)
 - `double` (examples: 2.00, 99.99, -2042.00009)
 - `boolean` (true or false)
- If uninitialized, variables have a default value:
 - `String`: `null`
 - `int`: `0`
 - `double`: `0.0`
 - `boolean`: `false`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Variables are declared to hold a specific type of data. Some of the more common types are:

- `String`: This is text data, such as "Hello".
- `int`: This is integer data—positive or negative whole numbers.
- `double`: These are positive or negative *real* numbers containing a decimal portion.
- `boolean`: This data type has a value of either true or false.

Most variables that have not been initialized are given a default value. The default values for `String`, `int`, `double`, and `boolean` are shown above. (Local variables are the exception. You will learn about local variables in the lesson titled "Creating and Using Methods.")

Notice that `String` begins with an uppercase letter, but the other types do not. You will learn the reason for this later, when you also learn about some other data types.

Naming a Variable

Guidelines:

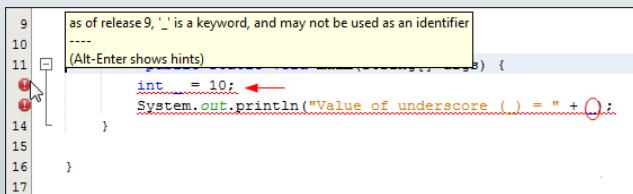
- Begin each variable with a lowercase letter. Subsequent words should be capitalized:
 - myVariable
- Names are case-sensitive.
- Names cannot include white space.
- Choose names that are mnemonic and that indicate to the casual observer the intent of the variable.
 - outOfStock (a boolean)
 - itemDescription (a String)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java SE 9: The Underscore Character Is Not a Legal Name

- If you use the underscore character ("_") as a one-character identifier in source code, then your code won't compile in Java SE 9.
- For example:



A screenshot of an IDE showing a warning message: "as of release 9, '_' is a keyword, and may not be used as an identifier". The code snippet shows a variable named '_'. A tooltip says "(Alt-Enter shows hints)". The code is as follows:

```
9
10
11     int _ = 10;
12     System.out.println("Value of underscore (_)= " + _);
13
14 }
15
16 }
17 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Uses of Variables

- Holding data used within a method:

```
String name = "Sam" ;  
double price = 12.35;  
boolean outOfStock = true;
```

- Assigning the value of one variable to another:

```
String name = name1;
```

- Representing values within a mathematical expression:

```
total = quantity * price ;
```

- Printing the values to the screen:

```
System.out.println(name);
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Variables are used extensively in the Java programming language for tasks such as:

- Holding data used within a method, such as the `main` method
- Assigning the value of one variable to another. In the first example above, the `name` variable is initialized with the value, “Sam”, and in the second example, its value is changed to the value of `name1` (unknown here).
- Representing values within a mathematical expression (* is the symbol for multiplication)
- Printing the values to the screen. For example, the same `System.out.println` method that you used in the last exercise to print out the text literal, “Welcome to the Shopping Cart”, can also be used to print out the value stored in the `name` variable.

Topics

- Introducing variables
- Working with String variables
- Working with numbers
- Manipulating numeric data



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Examples: Variable Declaration and Initialization

- Basic Example :

```
String address = "123 Oak St";           //one variable declared
                                         // and initialized
    type   identifier      value
```

- Other Examples:

```
String customer;           //One variable declared
String name, city;        //Two variables declared
String country ="USA", state="CO" //Two variables declared
                               //and initialized
city=" USA";               //One variable initialized after
                           //being declared earlier
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The syntax for declaring and initializing a variable is:

type identifier [= value];

where:

- type** represents the type of information or data held by the variable. In the examples in the slide, you see only `String` variable types declared.
- identifier** is the variable name. In the first example in the slide, the variable name is `customer`.

The second example shows how you can declare any number of variables of the same type on a single line without initializing them. Notice that when declaring multiple variables in a single line, they are separated by a comma.

You can either declare a variable without assigning an initial or you can initialize the variable at the same time you declare it.

String Concatenation

- String variables can be combined using the '+' operator.
 - stringVariable1 + stringVariable2
 - stringVariable1 + "String literal"
 - stringVariable1 + "String literal" + stringVariable2

- Example:

```
String greet1 = "Hello";
String greet2 = "World";
String message = greet1 + " " + greet2 + "!";
String message = greet1 + " " + greet2 + " " + 2014 +"!";
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Combining multiple Strings is called “concatenation.” You can concatenate a String variable to another String variable. You can also concatenate a String literal to a String variable.

As you can see in the example above, you can concatenate any number of String variables and String literals to achieve your goal.

You may find the last example surprising. You can also concatenate a number into a String variable. The compiler converts the numeric value to its equivalent String value. If we were to print the message variable after the last example, the output would be “Hello World 2014!”

String Concatenation

You can concatenate String variables outside or inside a method call:

```
String greet1 = "Hello";
String greet2 = "World";
String message = greet1 + " " + greet2 + "!";
System.out.println(message);
System.out.println(greet1 + " " + greet2 + "!");
```

Output:

```
Hello World!
Hello World!
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the examples in the slide, you see two variations of printing out String data by using the `System.out.println` method.

- In the first example, the `message` variable will be printed.
- In the second example, the expression containing the concatenation of variables plus String literals can be used within the method parentheses. The concatenation will be completed by the runtime engine before the `println` method is executed.
- As you can see, the output of both method invocations is the same.

Exercise 4-1: Using String Variables

1. In NetBeans, open the project **Exercise_04-1**.
2. Declare and initialize two String variables: `custName` and `itemDesc`.
3. Declare a String variable called `message`. Do not initialize it.
4. Assign the `message` variable with a concatenation of the `custName` and `itemDesc`. Include a String literal that results in a complete sentence.
 - Example: "Mary Smith wants to purchase a Shirt"
5. Print `message` to the System output.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you declare, initialize, and concatenate String variables and literals.



Quiz

Which of the following variable declarations and/or initializations are correct?

- a. int count = 5; quantity = 2;
- b. string name, label;
- c. boolean complete = "false";
- d. boolean complete = true;



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Introducing variables
- Working with `String` variables
- **Working with numbers**
- Manipulating numeric data



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

int and double Values

- int variables hold whole number values between:
 - -2,147,483,648
 - 2,147,483,647
 - Examples: 2, 1343387, 1_343_387
- double variables hold larger values containing decimal portions.
 - Use when greater accuracy is needed.
 - Examples: 987640059602230.7645 , -1111, 2.1E12



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- The int data type stores 32 bits of data. This means that you can store whole numbers within the range: -2,145,483,648 and 2,147,483,647. You cannot use commas to make the number more readable when you assign a value to an int variable. However, you can use underscores (_) to make your code more readable, as shown in one of above int examples. The compiler ignores these underscores. If you print the number to system output, the underscores will not appear. The only benefit of this is readability in your code.
- The double data type stores 64 bits of data. This means that you can store extremely large values—either negative or positive. The examples above show:
 - An extremely large number with four decimal points of precision
 - A negative whole number
 - A decimal number using exponential notation

Initializing and Assigning Numeric Values

- **int variables:**

- int quantity = 10;
 - int quantity = 5.5;



Compilation fails!

- **double variables:**

- double price = 25.99;
 - double price = 75;



Run time will
interpret as 75.0.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Introducing variables
- Working with `String` variables
- Working with numbers
- Manipulating numeric data



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Standard Mathematical Operators

Purpose	Operator	Example	Comments
Addition	+	sum = num1 + num2;	If num1 is 10 and num2 is 2, sum is 12.
Subtraction	-	diff = num1 - num2;	If num1 is 10 and num2 is 2, diff is 8.
Multiplication	*	prod = num1 * num2;	If num1 is 10 and num2 is 2, prod is 20.
Division	/	quot = num1 / num2;	If num1 is 31 and num2 is 6, quot is 5. The remainder portion is discarded. Division by 0 returns an error.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The table above assumes that all operands and result variables are integers (int). Mixing double and int types can alter the results. For instance, in the division example, if the quotient and dividend (or if all three) are double values, the quotient would show the decimal portion:

```
double quot, num1;  
num1 = 31;  
int num2 = 5;  
quot = num1 / num2;  
Answer: quot = 6.2
```

Increment and Decrement Operators (++ and --)

The long way:

```
age = age + 1;
```

or

```
count = count - 1;
```

The short way:

```
age++;
```

or

```
count--;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A common requirement in programs is to add or subtract 1 from the value of a variable. You can do this by using the + operator as follows:

```
age = age + 1;
```

Operator Precedence

Here's an example of the need for rules of precedence.

Is the answer to the following problem 34 or 9?

```
int c = 25 - 5 * 4 / 2 - 10 + 4;
```



Operator Precedence

Rules of precedence:

1. Operators within a pair of parentheses
2. Increment and decrement operators (++ or --)
3. Multiplication and division operators, evaluated from left to right
4. Addition and subtraction operators, evaluated from left to right



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In a complex mathematical statement with multiple operators on the same line, how does the computer pick which operator it should use first? To make mathematical operations consistent, the Java programming language follows the standard mathematical rules for operator precedence. Operators are processed in the following order:

1. Operators within a pair of parentheses
2. Increment and decrement operators
3. Multiplication and division operators, evaluated from left to right
4. Addition and subtraction operators, evaluated from left to right

If standard mathematical operators of the same precedence appear successively in a statement, the operators are evaluated from left to right.

Using Parentheses

Examples:

```
int c = (((25 - 5) * 4) / (2 - 10)) + 4;  
int c = ((20 * 4) / (2 - 10)) + 4;  
int c = (80 / (2 - 10)) + 4;  
int c = (80 / -8) + 4;  
int c = -10 + 4;  
int c = -6;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Your expression will be automatically evaluated with the rules of precedence. However, you should use parentheses to provide the structure you intend:

```
int c = (((25 - 5) * 4) / (2 - 10)) + 4;  
int c = ((20 * 4) / (2 - 10)) + 4;  
int c = (80 / (2 - 10)) + 4;  
int c = (80 / -8) + 4;  
int c = -10 + 4;  
int c = -6;
```

Exercise 4-2: Using and Manipulating Numbers

1. Continue editing **Exercise_04-1** or open **Exercise_04-2**.
2. Declare and initialize numeric fields: `price` (`double`) `tax` (`double`), and `quantity` (`int`). Also declare a double called `total`, but do not initialize it.
3. Change the message variable to include `quantity`
 - Example: "Mary Smith wants to purchase 1 Shirt."
4. Calculate total by multiplying `price * quantity * tax`.
5. Print a message showing the total cost (example: "Total cost with tax is: 25.78.").



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you declare and initialize numeric variables, and concatenate Strings with numbers.



Quiz

Which of the following statements are correct Java code?

- a. int count = 11.4;
- b. double amount = 11.05;
- c. int cost = 133_452_667;
- d. double total = 1.05 * amount;



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Quiz



Given:

```
String name = "Bob";  
String msg;  
int num = 3;
```

Which of the following statements correctly assigns the value “Bob wrote 3 Java programs.” to the msg variable?

- a. msg = name + " wrote " + num " Java programs.;"
- b. msg = name + " wrote " + 3 + " Java programs.;"
- c. msg = "Bob wrote "+ (2+1) + " Java programs.;"
- d. msg = name + " wrote " + 2+1 + " Java programs.;"



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

- a is incorrect because it is missing a + sign between the num variable and the final String literal.
- b is correct because the compiler converts the int of value 3 to a String.
- c is correct because, due to the use of parentheses, the addition operation is performed first, before the concatenation.
- d is incorrect because it would result in “Bob wrote 21 Java programs.” The compiler converts each number to a String separately and concatenates them together.

Summary

In this lesson, you should have learned how to:

- Describe the purpose of a variable in the Java language
- List and describe four data types
- Declare and initialize `String` variables
- Concatenate `String` variables with the '+' operator
- Make variable assignments
- Declare and initialize `int` and `double` variables
- Modify numeric values by using operators
- Override default operator precedence using ()



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

Managing Multiple Items

5



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



ORACLE®

Objectives

After completing this lesson, you should be able to:

- Explain what a boolean expression is
- Create a simple `if/else` statement
- Describe the purpose of an array
- Declare and initialize a `String` or `int` array
- Access the elements of an array
- Explain the purpose of a `for` loop
- Iterate through a `String` array using a `for` loop



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Working with conditions
- Working with an array of items
- Processing an array of items



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



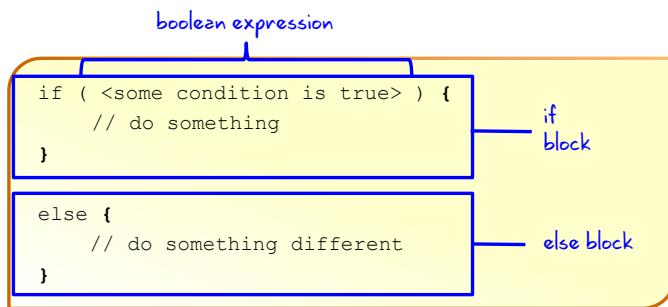
Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

Making Decisions



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The if/else Statement



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The **if/else** statement is one way of branching your code depending on some condition. It uses the two Java keywords, **if** and **else**.

- If some condition is true, execute the code within the **if** block.
- Else, if that condition is false, execute the code in the **else** block.

The condition to be evaluated is surrounded by parentheses. It is referred to as a Boolean expression because it must evaluate to either **true** or **false**.

Boolean Expressions

Review:

- boolean data type has only two possible values:
 - true
 - false

A boolean expression is a combination of variables, values, and operators that evaluate to true or false.

- `length > 10;`
 - `size <= maxSize;`
 - `total == (cost * price);`
- Relational operators*



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Relational Operators

Condition	Operator	Example
Is equal to	==	int i=1; (i == 1)
Is not equal to	!=	int i=2; (i != 1)
Is less than	<	int i=0; (i < 1)
Is less than or equal to	<=	int i=1; (i <= 1)
Is greater than	>	int i=2; (i > 1)
Is greater than or equal to	>=	int i=1; (i >= 1)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Examples

Sometimes there is a quicker way to meet your objective. Boolean expressions can be used in many ways.

```
24      int attendees = 4;
25      boolean largeVenue;
26
27      // if statement example
28      if (attendees >= 5){
29          largeVenue = true;
30      }
31      else {
32          largeVenue = false;
33      }
34
35      // same outcome with less code
36      largeVenue = (attendees >= 5);
```

Assign a boolean by using an if statement.

Assign the boolean directly from the boolean expression.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the slide above, you see examples of two different ways to set the `largeVenue` boolean value:

- In lines 28–33, an `if` statement tests the value of the `attendees` variable. If it is greater than 5, `largeVenue` is set to `true`; otherwise it is set to `false`.
- In line 36, the same outcome is achieved with one line of code. The result of the same boolean expression that was evaluated in the `if` statement (`attendees >= 5`) is directly assigned to the `largeVenue` boolean.

Exercise 5-1: Using if Statements

1. Open the project **Exercise_05-1**.
2. Use an `if` statement to test the quantity of the item:
 - if it is > 1, concatenate an 's' to message so that it indicates multiple items.
3. Declare a boolean, `outOfStock`.
4. Use an `if|else` statement to test if the item is out of stock:
 - if item is out of stock, inform the user that the item is unavailable.
 - else, print the message and total cost
5. Run the program with `outOfStock = true`.
6. Run it again with `outOfStock = false`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you use an `if` and an `if|else` statement to check if an item is out of stock.

Quiz



What is the purpose of the `else` block in an `if/else` statement?

- a. To contain the remainder of the code for a method
- b. To contain code that is executed when the expression in an `if` statement is false
- c. To test if an expression is false



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Topics

- Working with conditions
- Working with an array of items
- Processing an array of items



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



What If There Are Multiple Items in the Shopping Cart?

```
01 // Without an array  
02 String itemDesc1 = "Shirt"; Not realistic if  
100s of items!  
03 String itemDesc2 = "Trousers";  
04 String itemDesc3 = "Scarf";  
05 // Using an array  
06 String[] items = {"Shirt", "Trousers", "Scarf"};  
07
```

Much better!



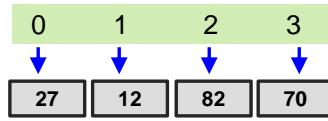
Think about how your code would look if there were multiple items in the shopping cart. You would have to initialize each item description separately. Imagine if you had a thousand items!

As you continued to build out this shopping cart application, the amount of code needed to handle each item individually would not only be time-consuming, but would make your code hard to read and difficult to maintain.

The code example above shows a better alternative that we will explore now: the array.

Introduction to Arrays

- An array is an indexed container that holds a set of values of a single type.
- Each item in an array is called an *element*.
- Each element is accessed by its numerical index.
- The index of the first element is 0 (zero).
 - A four-element array has indices: 0, 1, 2, 3.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The array is a container that holds a set of `String` values, or a set of `int` values, or a set of `double` values, and so on.

The elements (items) of the array are accessed through a numeric index. Using this index, you can set or get a value from a specific element.

Array Examples

Array of int types

```
27 12 82 70 54 1 30 34
```

Array of String types

```
Hugh Mongus
Aaron Datiress
Stan Ding
Albert Kerkie
Carrie DeKeys
Walter Mellon
Hugh Morris
Moe DeLawn
```

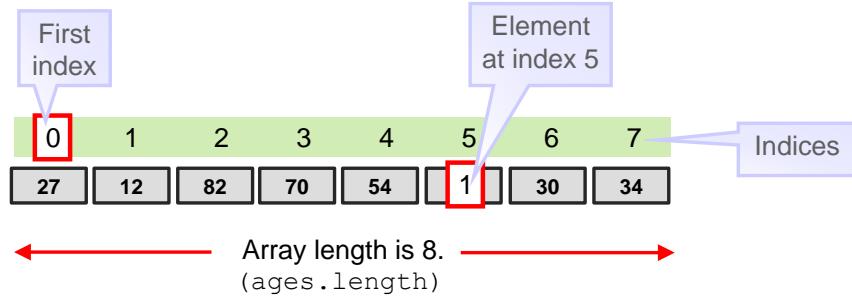


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Arrays can be of any data type, but all elements have to share the same type.

Array Indices and Length

The `ages` array has eight elements.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, the length of an array cannot be changed.
- Each item in an array is called an *element*, and each element is accessed by its numerical index. As shown in the diagram above, index numbering begins with 0. For example, the eighth element would be accessed at index 7.
- The length of an array can be accessed using dot notation to access the `length` field. Assuming that the array in the diagram is called `ages`, you can determine how many elements are in the array by using:

```
int agesLength = ages.length;
```

Declaring and Initializing an Array

- Syntax:

```
type[] arrayIdentifier = {comma-separated list of values};
```

- Declare arrays of types String and int:

```
String[] names = {"Mary", "Bob", "Carlos"};  
int[] ages = {25, 27, 48};
```

All in one line



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this slide, you see the syntax and an example of how to declare the array and initialize the values. (This assumes that you know at this time what the values will be).

- Syntax for declaring an array:

```
type [] arrayIdentifier = {comma-separated list of values};
```

- **Note:** Another acceptable syntax is: type arrayIdentifier[] = {comma-separated list of values};

where:

- type represents the data type for each of the values stored in the array
- [] informs the compiler that you are declaring an array
- arrayIdentifier is the variable name that you use when you refer to the array
- You can list as many values as you need. Separate the values with a comma.

Declaring and Initializing an Array

Examples:

```
1 int[] ages = new int[3];
2 ages[0] = 19;
3 ages[1] = 42;
4 ages[2] = 92;
5
6 String[] names = new String[3];
7 names[0] = "Mary";
8 names[1] = "Bob";
9 names[2] = "Carlos";
```

Multistep approach

Multistep approach



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this example, the `int` array, `ages`, is instantiated with a size of 3 on line 1. The creation of the array uses the `new` keyword. You will learn much more about the purpose of this keyword in the lesson titled “Describing Objects and Classes.”

On lines 2 through 4, the elements of the `ages` array are initialized.

Likewise, on line 6, the `String` array, `names`, is instantiated with a size of 3, and its elements are initialized on lines 7 through 9.

Accessing Array Elements

- Get values from the `ages` array:

```
int[] ages = {25, 27, 48};  
int myAge = ages[0];  
int yourAge = ages[1];  
System.out.println("My age is " + ages[0]);
```

- Set values from the `names` array:

```
String[] names = {"Mary", "Bob", "Carlos"};  
names[0] = "Gary";  
names[1] = "Rob";
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Elements of the array are accessed by referencing the index of that element. For example:

- To get the value from the first element of the `ages` array, use `ages[0]`.
- To get the value from the second element of the `ages` array, use `ages[1]`.
- You can directly use the value of an array element in an expression by using the same syntax. In the third example, you see `ages[0]` referenced directly when calling `System.out.println`.
- To set a value in the first element of the `names` array, use `names[0] = "some value"`.

Exercise 5-2: Using an Array

1. Open the project **Exercise_05-2** in NetBeans.
2. Declare a `String` array and initialize it with four elements.
 - Each element represents a different item description ("Shirt", for instance).
3. Change `message` to show how many items the customer wants to purchase.
 - Hint: Use the `.length` property of your array.
4. Print just one element in the array.
 - What happens if you use index number 4?



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Quiz



Why does the following code not compile? Select all that apply.

```
int[] lengths = {2, 4, 3.5, 0, 40.04};
```

- a. lengths cannot be used as an array identifier.
- b. All of the element values should have the same format (all using double values, or all using int values).
- c. The array was declared to hold int values. double values are not allowed.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: c

a is incorrect because lengths is a perfectly valid array identifier.

b is incorrect because it implies that this array could contain elements of type double.

c is correct.

Quiz



Given the following array declaration, which of the following statements are true?

- ```
int[] classSize = {5, 8, 0, 14, 194};
```
- a. `classSize[0]` is the reference to the first element in the array.
  - b. `classSize[5]` is the reference to the last element in the array.
  - c. There are 5 integers in the `classSize` array.
  - d. `classSize.length = 5`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

### Answer: a, c, d

a is correct.

b is incorrect because the array index begins with 0. Thus, the index for the last element is one less than the total number of elements.

c is correct.

d is correct.

## Topics

- Working with conditions
- Working with an array of items
- Processing an array of items



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Loops

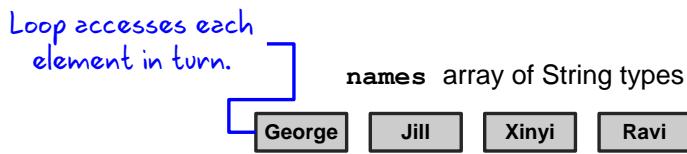
Loops are used in programs to repeat blocks of statements

- Until an expression is false
  - or
- For a specific number of times:
  - I want to print each element of an array.
  - I want to print each element of an ArrayList. (The ArrayList class is covered in the lesson titled “Working with Arrays, Loops, and Dates.”)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# Processing a String Array



```
for (String name : names) {
 System.out.println("Name is " + name);
}
```

Each iteration returns the next element of the array.

Output:

```
Name is George
Name is Jill
Name is Xinyi
Name is Ravi
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The for loop syntax is:

```
for (<type> <variable> : <array name>) {
 <code_block to be performed for each array element>
}
```

where:

- `for` indicates that a loop is being defined
- `<type>` is the data type of each of the elements within the array
- `<variable>` is a placeholder used to store each element of an array
- `:` indicates that the object reference that follows is an array
- `<array name>` is the array, whose length determines the number of iterations to perform
- `code_block` is the code that will be executed in each iteration of the loop

In the example above, there are four elements in the `names` array. Therefore, the code block will be executed four times. Each time, the `name` variable holds a different array element.

## Using break with Loops

break example:

```
01 int passmark = 12;
02 boolean passed = false;
03 int[] scores = {4,6,2,8,12,35,9};
04 for (int unitScore : scores){
05 if (unitScore >= 12){
06 passed = true;
07 break;
08 }
09 }
10 System.out.println("At least one passed? " +passed);
```

No need to go through the loop again, so use break.

Output:

```
At least one passed? true
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Occasionally, some condition occurs that makes it unnecessary to continue the loop. The `break` keyword enables you to do this. When `break` is encountered, the program execution moves to the first line of code outside the `for` block.

- The example in the slide shows the use of `break`. You will notice that it uses an `if` statement within the `for` block. This `if` statement is executed on each iteration of the loop.
- Assuming that the purpose of the code is to find out whether any of the scores in the array are equal or above the `passmark`, you can set `passed` to `true` and jump out of the loop as soon as the first such score is found.
- When `break` is called on line 7, execution of the program skips to line 10.

## Exercise 5-3: Using a Loop to Process an Array

1. In NetBeans, continue editing **Exercise\_05-2** or open **Exercise\_05-3**.
2. Create a `for` loop that iterates through the array of item descriptions, displaying each element.
3. Precede the list of elements with the message: "Items purchased:".



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Quiz



Given the following code,

```
int[] sizes = {4, 18, 5, 20};
for (int size : sizes){
 if (size > 16){break;}
 System.out.println("Size: "+size + ", ");
}
```

which option below shows the correct output?

- a. Size: 4,
- b. Size: 4
- c. Size: 4,  
 Size: 5,
- d. There is no output.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

### Answer: a

- a is correct.  
b is incorrect because the comma appears within each `println` method.  
c is incorrect because when the first size greater than 16 is found, the loop breaks and does not return.  
d is incorrect because the first iteration of the loop would print.

## Summary

In this lesson, you should have learned how to:

- Use a boolean expression
- Create a simple if/else block
- Describe the purpose of an array
- Declare and initialize a String or int array
- Access the elements of an array
- Explain the purpose of a for loop
- Iterate through a String Array using a for loop



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



# Describing Objects and Classes

6



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



ORACLE®

Moisés Ocampo Sámano (aos\_moyer@hotmail.com) has a  
non-transferable license to use this Student Guide.

## Interactive Quizzes



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Objectives

After completing this lesson, you should be able to:

- List the characteristics of an object
- Define an object as an instance of a class
- Instantiate an object and access its fields and methods
- Describe how objects are stored in memory
- Instantiate an array of objects
- Describe how an array of objects is stored in memory
- Declare and instantiate an object as a field



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case

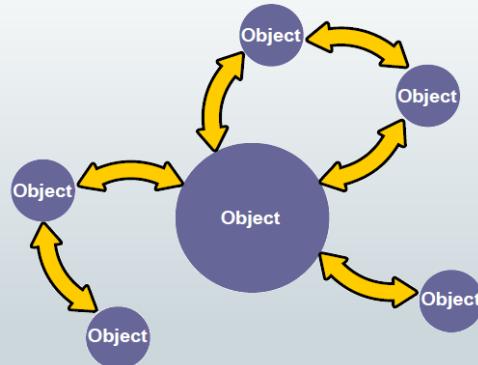


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



# Object-Oriented Programming

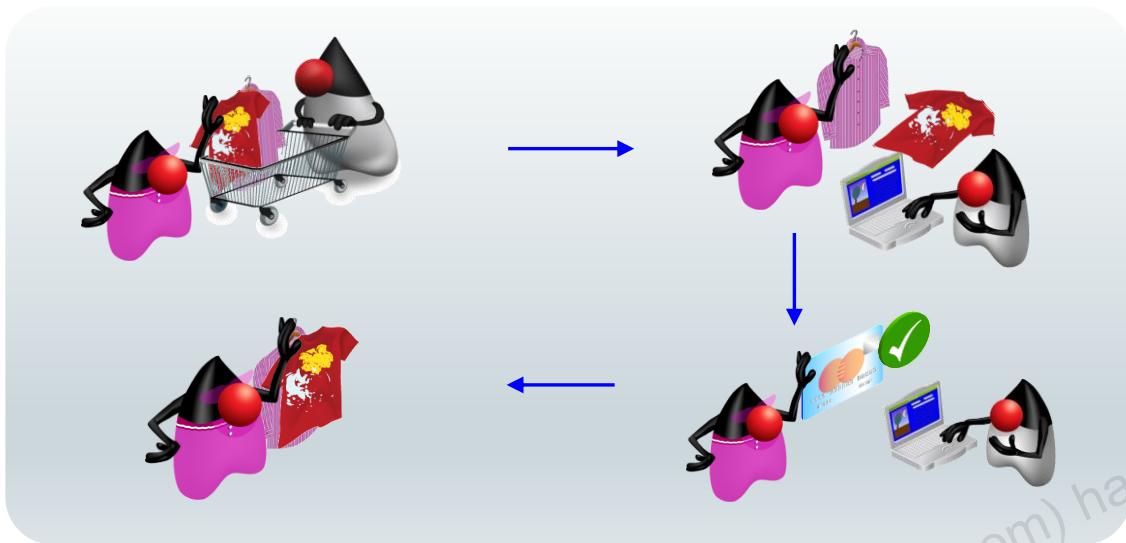
- Interaction of objects
- No prescribed sequence



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



## Duke's Choice Order Process



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the first five lessons, the exercises mention a shopping cart class that contains items. Take another look at the shopping cart scenario.

Imagine an online store called Duke's Choice. His number one shopper is his mother, Mrs. Duke. As Mrs. Duke shops, she places items in a shopping cart. Mrs. Duke likes shirts, so she places shirts in her cart. After she fills the cart, she checks out. The checkout process applies the purchase to a credit card, which is verified, and then Mrs. Duke receives an order number so that she can track her order or return it.

As a software developer, when you are presented with a scenario such as Duke's Choice for an application that you need to develop, you can analyze the scenario by breaking it into steps and defining the objects of the scenario.



## Characteristics of Objects

Objects are physical or conceptual.

- Objects have **properties**:
  - Size
  - Shape
  - Name
  - Color
- Objects have **behaviors**:
  - Shop
  - Put item in cart
  - Pay

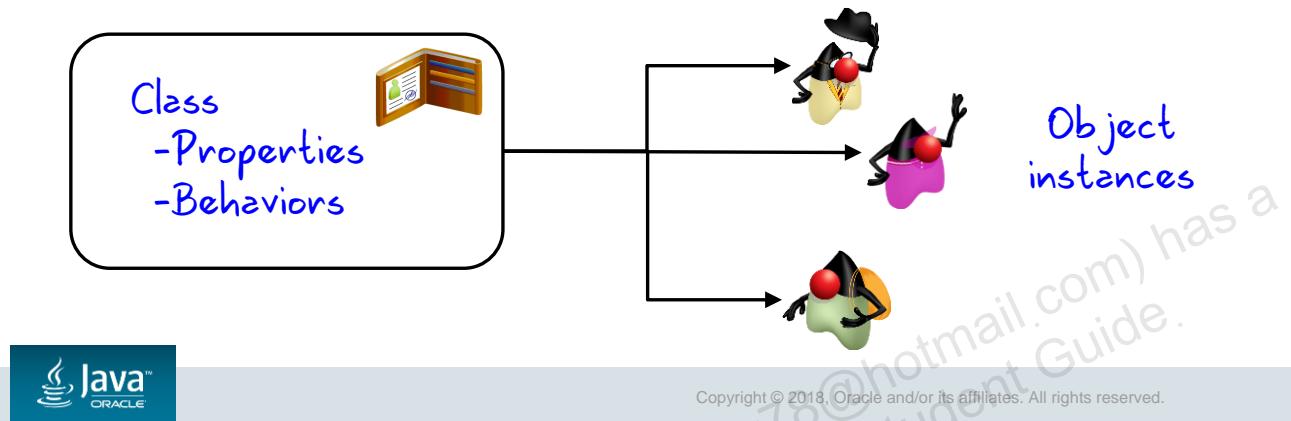


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



## Classes and Instances

- A class:
  - Is a blueprint or recipe for an object
  - Describes an object's properties and behaviors
  - Is used to create object instances



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You just learned about some of the objects, characteristics, and behaviors in the Duke's Choice scenario. Here is an example of one of Duke's Choice objects, the `Customer`, and its function in the store. `Customer` is the class, and a class is a blueprint or recipe for an object. The class describes an object's properties and behaviors.

Classes are used to create object instances, such as the three `Customer` object instances, as illustrated by the three images.



## Quiz

Which of the following statements is true?

- a. An object is a blueprint for a class.
- b. An object and a class are exactly the same.
- c. An object is an instance of a class.
- d. A class is an instance of an object.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



You have just learned about objects, classes, and their characteristics (properties and behaviors). Now it is time to look at fields and methods.

# The Customer Properties and Behaviors

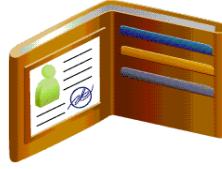


## Properties:

- Name
- Address
- Age
- Order number
- Customer number

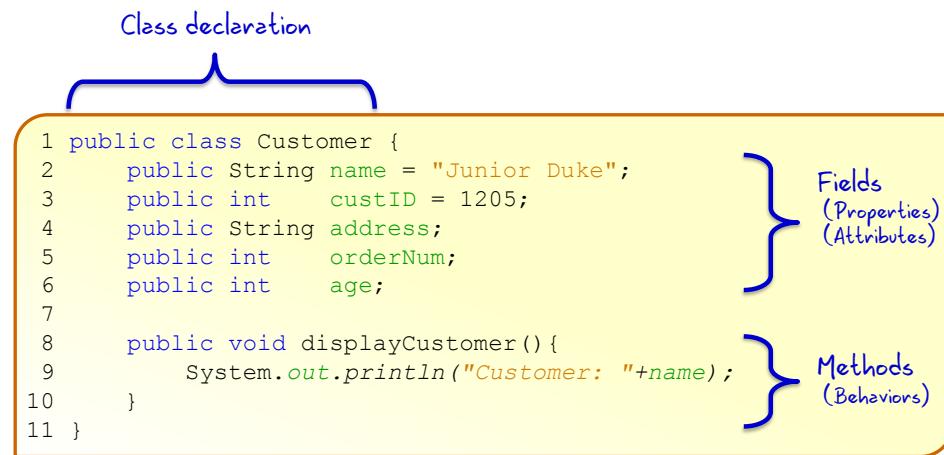
## Behaviors:

- Shop
- Set Address
- Add item to cart
- Ask for a discount
- Display customer details



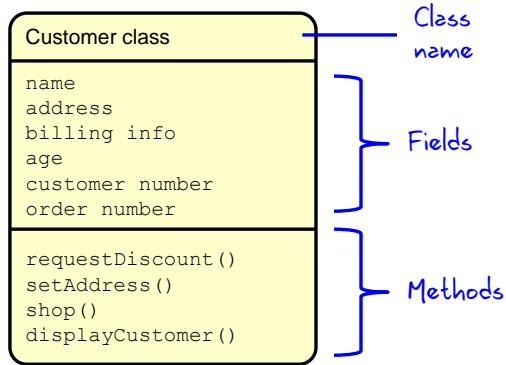
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# The Components of a Class



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# Modeling Properties and Behaviors



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Exercise 6-1: Creating the Item Class

1. Open the project **Exercise\_06-1** in NetBeans
2. Create the Item class as a plain **Java class**.
3. Declare public fields for `ID` (`int`), `descr` (`String`), `price` (`double`), and `quantity` (`int`).
  - You will not be able to test the `Item` class until Exercise 6-2.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Topics

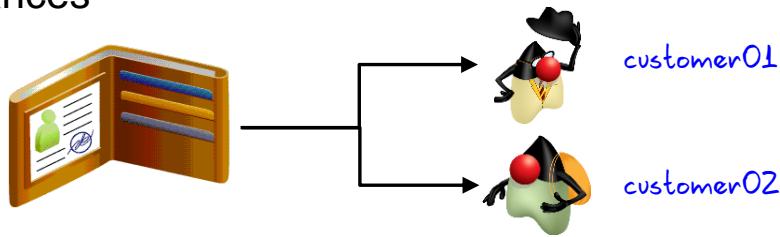
- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



## Customer Instances



```
public static void main(String[] args){

 Customer customer01 = new Customer();
 Customer customer02 = new Customer();

 customer01.age = 40;
 customer02.name = "Duke";

 customer01.displayCustomer();
 customer02.displayCustomer();
}
}
```

- } Create new instances (instantiate).
- } Fields are accessed.
- } Methods are called.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



# Object Instances and Instantiation Syntax

The syntax is:  
`<class name> variable = new <class name>()`

variable becomes a reference to that object.  
The new keyword creates (instantiates) a new instance.

```
public static void main(String[] args) {

 Customer customer01 = new Customer(); //Declare and instantiate

 Customer customer02; //Declare the reference
 customer02 = new Customer(); //Then instantiate

 new Customer(); //Instantiation without a reference
 //We can't use this object later
 //without knowing how to reference it.
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

By using the `new` keyword, a new instance of the class is now available to be accessed through the variable, which stores a reference to that object. It can be referred to as a reference variable or an object reference.

Notice that, following the `new` keyword, you see the class name followed by parentheses. This looks similar to calling a method, doesn't it? You are calling a method—the `constructor` method of the `Customer` class. Every class has a `constructor` method that has the same name as the class. Constructors are covered in more detail in the lesson titled “Creating and Using Methods.”

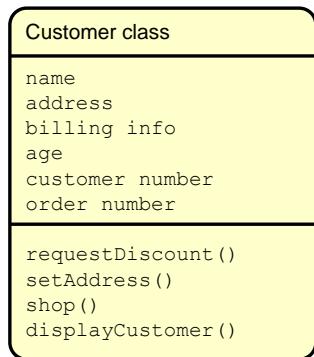
**To summarize, there are three steps to getting an object reference:**

1. Declare the reference.
2. Instantiate the object using the `new` keyword and the class `constructor` method.
3. Assign the object to the reference.

Note that the way that the assignment operator (an `=` symbol) works requires that the reference and the newly created object must be in the same statement. (Statements are ended with the semicolon symbol and are not the same as lines. The end of a line means nothing to the Java compiler; it only helps make the code more readable.)

## The Dot (.) Operator

Follow the reference variable with a dot operator (.) to access the fields and methods of an object.



```
public static void main(String[] args) {
 Customer customer01 = new Customer();

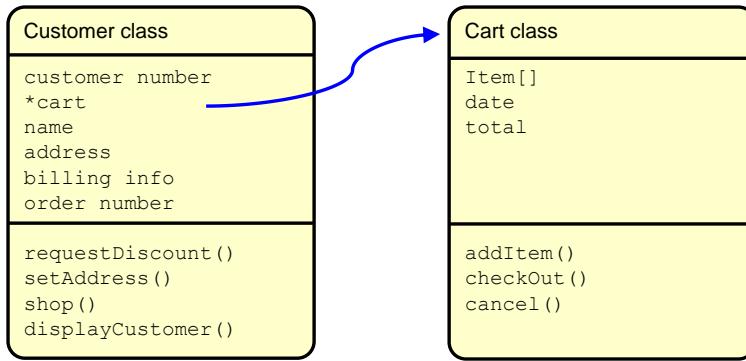
 //Accessing fields
 System.out.println(customer01.name);
 customer01.age = 40;

 //Calling methods
 customer01.requestDiscount();
 customer01.displayCustomer();
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Objects with Another Object as a Property



```
public static void main(String[] args){

 Customer customer01 = new Customer();
 customer01.cart.cancel();

}

//How to access methods of an
//object within another object
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

So far you have seen objects with properties such as boolean, int, double, and String. What if you wanted an object's property to be another object with its own set of properties and behaviors, such as a customer with a cart property? That way, an instance of a Customer would have access to the properties and behaviors found in a Cart. This would enable the customer to add items to the cart and then checkOut (purchase) the cart. Can this be done? The answer is yes.

You can access fields and methods of objects within another object by applying the dot operator multiple times.

**Note:** A best practice is to use attribute and operation names that clearly describe the attribute or operation. The asterisk (\*) denotes an attribute that is a reference to another object.

## Quiz



Which of the following lines of code instantiates a `Boat` object and assigns it to a `sailBoat` object reference?

- a. `Boat sailBoat = new Boat();`
- b. `Boat sailBoat;`
- c. `Boat = new Boat()`
- d. `Boat sailBoat = Boat();`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

## Topics

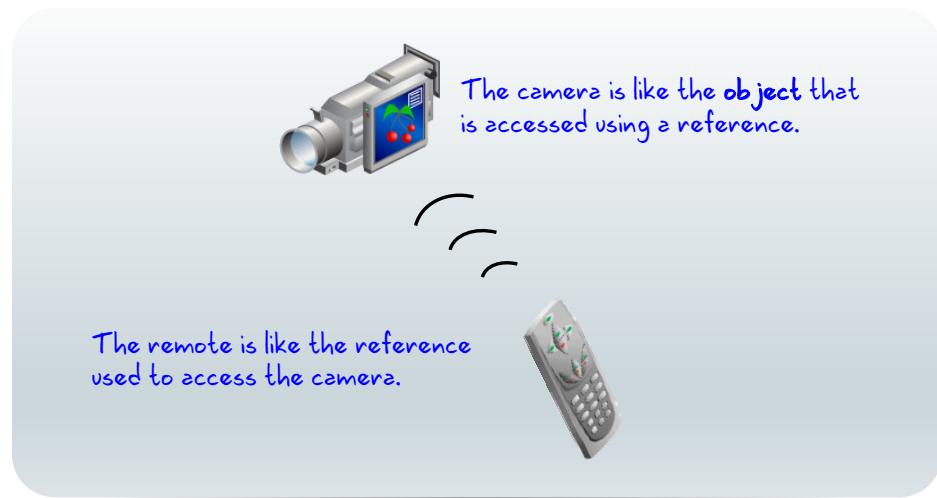
- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- **Working with object references**
- Doing more with arrays
- Introducing the soccer league use case



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



## Accessing Objects by Using a Reference



What you have learned up to this point is:

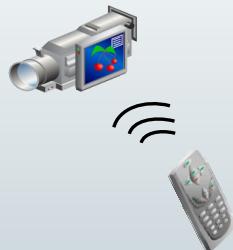
- Objects are accessed using references.
- Objects are instantiated objects of their class type.
- Objects consist of properties and operations, which in Java are fields and methods.

To work with an object, you need to access it using a reference. A good analogy is using a remote control to operate an electronic device. The buttons on the remote control can be used to modify the behavior of the device (in this case, a camera). For example, you can make the camera stop, play, or record by interacting with the remote.

# Working with Object References

1

Pick up the remote to gain access to the camera.



1

Create a Camera object and get a reference to it.

```
11 Camera remote1;
12
13 remote1 = new Camera();
14
15 remote1.play();
```

2

Press the remote's controls to have camera do something.

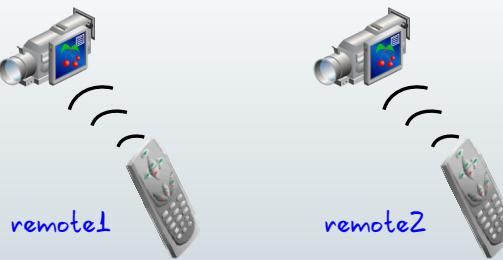
2

Call a method to have the Camera object do something.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# Working with Object References



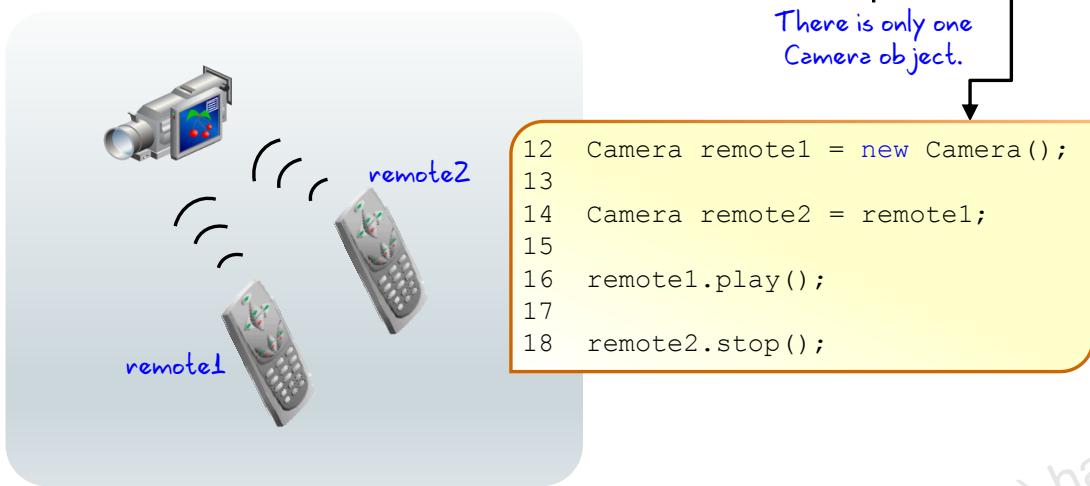
```
12 Camera remote1 = new Camera();
13
14 Camera remote2 = new Camera();
15
16 remote1.play();
17
18 remote2.play();
```

There are two Camera objects.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



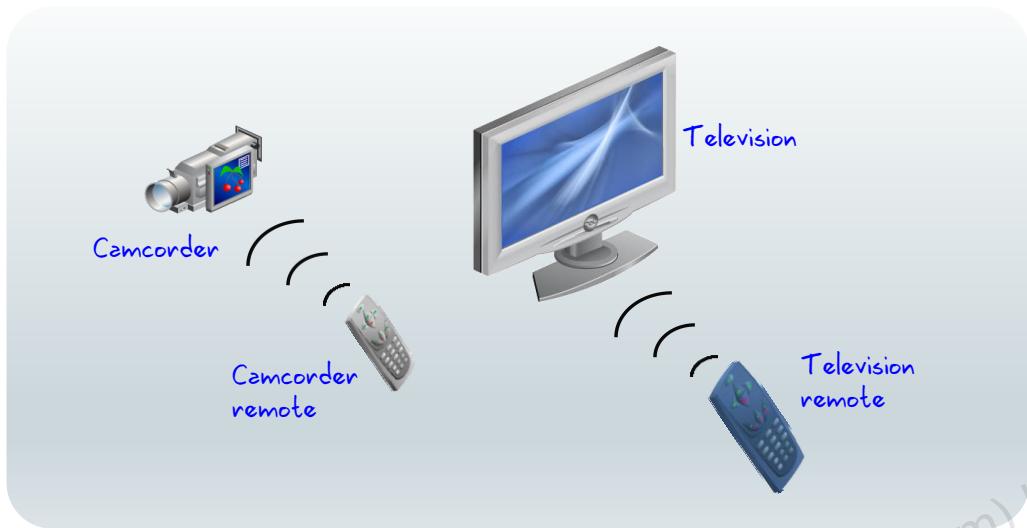
# Working with Object References



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The diagram shows another important aspect of how references work. In this example, a `Camera` object is created and its reference assigned to a `Camera` reference, `remote1`. This reference is then assigned to another `Camera` reference, `remote2`. Now both references are associated with the same `Camera` object, and methods called on either reference will affect the same `Camera` object. Calling `remote1.play` is no different than calling `remote2.play`. Both remotes operate the same camera.

## References to Different Objects



To extend the analogy just a little further, to work with a different type of object (for example, a television), you need a remote for that object. In the Java world, you need a reference of the correct type for the object that you are referencing.

You can ignore the fact that there is such a thing as a universal remote controller, although later in the course you will discover that Java also has the concept of references that are not limited to a single object type! For the moment, let's just say that a reference of the same type as the object is one of the reference types that can be used, and is a good place to start exploring the world of Java objects.

## References to Different Objects

```
6 Camera remote1 = new Camera();
7 remote1.menu();
8
9 TV remote2 = new TV();
10 remote2.menu();
11
12 Shirt myShirt = new Shirt();
13 myShirt.display();
14
15 Trousers myTrousers = new Trousers();
16 myTrousers.display();
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

remote1 references a Camera object.

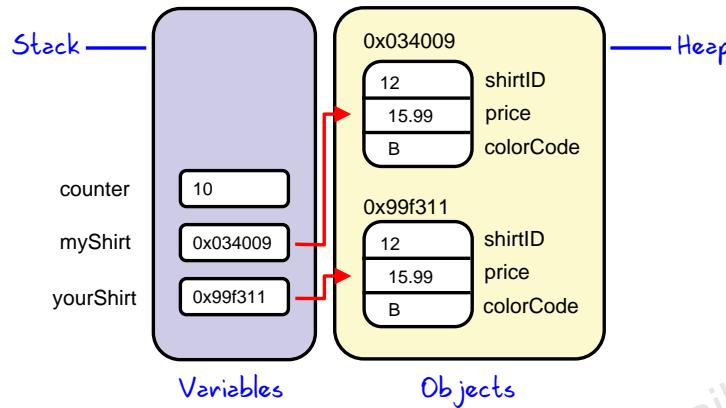
remote2 references a TV object.

myShirt references a Shirt object.

myTrousers references a Trousers object.

## References and Objects in Memory

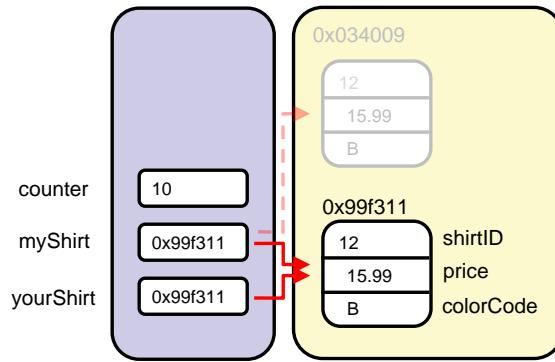
```
12 int counter = 10;
13 Shirt myShirt = new Shirt();
14 Shirt yourShirt = new Shirt();
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# Assigning a Reference to Another Reference

```
myShirt = yourShirt;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Two References, One Object

Code fragment:

```
12 Shirt myShirt = new Shirt();
13 Shirt yourShirt = new Shirt();
14
15 myShirt = yourShirt; //The old myShirt object is
16 //no longer referenced
17 myShirt.colorCode = 'R';
18 yourShirt.colorCode = 'G';
19
20 System.out.println("Shirt color: " + myShirt.colorCode);
```

Output from code fragment:

```
Shirt color: G
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This example now shows what happens if you use either reference to make a change or get a value from the object. References `yourShirt` and `myShirt` refer to the same object, so making a change or getting a field value by using one reference is exactly the same as doing it with the other reference. The old object that was previously referenced by `myShirt` goes away.

## Exercise 6-2: Modifying the ShoppingCart to Use Item Fields

1. Continue editing **Exercise\_06-1** or open **Exercise\_06-2** in NetBeans.
2. Create a new Java Main Class called `ShoppingCart`. This class contains a single main method. The rest of this exercise is spent modifying `ShoppingCart.java`.
3. Declare and instantiate two objects of type `Item`. Initialize only the `descry` field in each, using different values for each.
4. Print the description for each item and run the code.
5. (Optional) Above the code that prints the descriptions, assign `item2` to `item1`. Run it again.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you declare and instantiate two variables of type `Item` in the `ShoppingCart` class and experiment with accessing properties and calling methods on the object.

## Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- **Doing more with arrays**
- Introducing the soccer league use case



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



## Arrays Are Objects

Arrays are handled by an implicit Array *object*.

- The Array variable is an *object reference*, not a primitive data type.
- It must be instantiated, just like other objects.

- Example:

```
int[] ages = new int[4];
```

This array can hold  
four elements.

- Previously, you have been using a shortcut to instantiate your arrays.

- Example:

```
int[] ages = {8, 7, 4, 5};
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

An array is actually an object type and is handled implicitly through a class called `Array` (not available in the Java API documentation). Therefore, like other object types (`String` is an exception) it must be instantiated using the `new` keyword.

- In the top example, an `int` array called `ages` is declared and instantiated with a capacity to hold four elements.

## Declaring, Instantiating, and Initializing Arrays

- Examples:

```
1 String[] names = {"Mary", "Bob", "Carlos"};
2
3 int[] ages = new int[3];
4 ages[0] = 19;
5 ages[1] = 42;
6 ages[2] = 92;
```

- Not permitted (compiler will show an error):

```
int[] ages;
ages = {19, 42, 92};
```

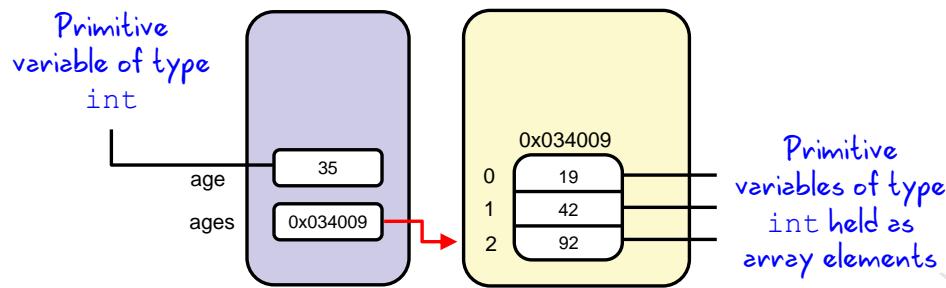


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

As introduced in the lesson titled “Managing Multiple Items,” there are two approaches for creating and initializing arrays. Using the **new** keyword allows you to declare and instantiate an array of a particular size and initialize it at a later time.

## Storing Arrays in Memory

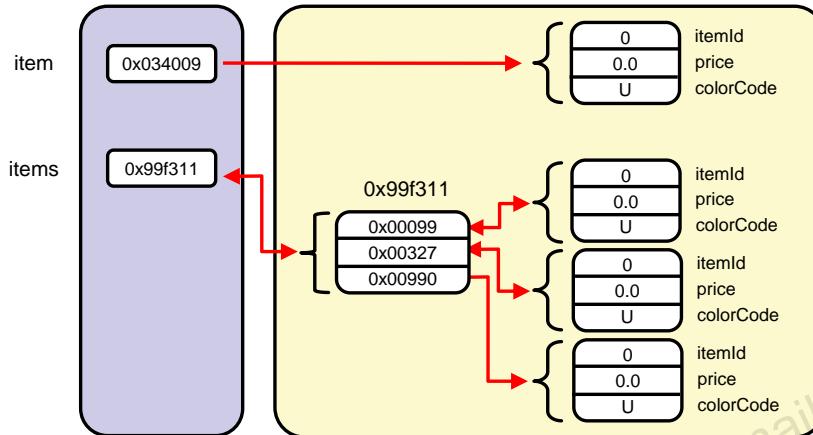
```
int age = 35;
int[] ages = {19, 42, 92};
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

# Storing Arrays of Object References in Memory

```
Item item = new Item();
Item[] items = { new Item(), new Item(), new Item() };
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

## Quiz



The following code is the correct syntax for \_\_\_\_\_ an array:

```
array_identifier = new type[length];
```

- a. Declaring
- b. Setting array values for
- c. Instantiating
- d. Declaring, instantiating, and setting array values for



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

### Answer: c

**a** is incorrect. Declaring the array would look like this, assuming an array of object types: Type [ ]  
array\_identifier;

**b** is incorrect. Setting array values would look like this, assuming an array of object types:

```
array_identifier[0]= new Type();
```

**c** is correct. The code example shows the array being initialized to a specific size.

**d** is incorrect. Declaring, instantiating, and setting array values would look like this, assuming an array of object types:

```
Type[] array_identifier = {new Type(), new Type(), new Type()};
```

## Quiz



Given the following array declaration, which of the following statements are true?

- ```
int[ ] ages = new int[13];
```
- a. ages [0] is the reference to the first element in the array.
 - b. ages [13] is the reference to the last element in the array.
 - c. There are 13 integers in the ages array.
 - d. ages [5] has a value of 0.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, c, d

Topics

- Describing objects and classes
- Defining fields and methods
- Declaring, instantiating, and using objects
- Working with object references
- Doing more with arrays
- Introducing the soccer league use case



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Soccer Application

Practices 6 through 14 build a soccer league application with the following features:

- Any number of soccer teams, each with up to 11 players
- Set up an all-play-all league.
- Use a random play game generator to create test games.
- Determine the rank order of teams at the end of the season.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the remaining practices in this course, you will build an application that manages a Soccer League. The application will keep details on teams and players, as well as the results of games.

You will also write code that will randomly generate game results so that you can then develop code to list the Teams in rank order.

Creating the Soccer Application

A separate project for each practice

Sample output showing events in a game

Sample output showing rank order of teams

The Greys vs. The Pinks (2014-03-08)
Kickoff by Agatha Christie of The Greys. (0.0 mins.)
Arthur Conan Doyle of The Pinks currently has possession. (6.0 mins.)
GOAL! Scored by W. B. Yeats of The Greys. (7.0 mins.)
Kickoff by Alan Patton of The Pinks. (8.0 mins.)
Alexander Solzhenitsyn of The Pinks currently has possession. (11.0 mins.)
GOAL! Scored by Arthur Conan Doyle of The Pinks. (14.0 mins.)
Kickoff by Agatha Christie of The Greys. (18.0 mins.)
Alan Patton of The Pinks currently has possession. (23.0 mins.)
Agatha Christie of The Greys currently has possession. (24.0 mins.)
GOAL! Scored by Agatha Christie of The Greys. (40.0 mins.)
Kickoff by Arthur Conan Doyle of The Pinks. (44.0 mins.)
Arthur Conan Doyle of The Pinks currently has possession. (49.0 mins.)
GOAL! Scored by Arthur Conan Doyle of The Pinks. (55.0 mins.)
Kickoff by Agatha Christie of The Greys. (59.0 mins.)
Alan Patton of The Pinks currently has possession. (73.0 mins.)
GOAL! Scored by W. B. Yeats of The Greys. (89.0 mins.)
The Pinks win! (3 - 2)

Team Points
The Reds:17:20
The Blues:17:17
The Pinks:12:17
The Greens:8:12
The Greys:6:13
BUILD SUCCESSFUL (total time: 0 seconds)

Java™ ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Soccer Web Application

Soccer League Games								
Replay games								
Home Teams	Away Teams							
	The Magpies	X	(0 - 1)	(4 - 2)	(1 - 0)	(3 - 0)	(1 - 0)	15 18
	The Crows	(2 - 1)	X	(1 - 0)	(0 - 1)	(0 - 0)	(0 - 0)	10 18
	The Reds	(0 - 1)	(0 - 1)	X	(1 - 1)	(1 - 0)	(1 - 0)	13 14
	The Blues	(4 - 1)	(0 - 2)	(0 - 1)	X	(3 - 4)	(1 - 0)	12 14
	The Rovers	(3 - 0)	(5 - 2)	(2 - 4)				18 15
	The Harriers	(1 - 3)	(1 - 1)	(3 - 3)				8 7
The Rovers vs. The Reds (2 - 4)								
Team Player Time								
The Reds	Jane Austin	7						
The Rovers	J. M. Synge	21						
The Reds	Jane Austin	41						
The Reds	Mark Twain	46						
The Reds	Brian Moore	76						
The Rovers	Charlotte Bronte	83						
Return to main page								

Teams listed in rank order

Click the score
of a game to
show game
details.

Points and goals scored
used for ordering



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The code that you write in the practices can be used by a simple web application to view the results of games in the league. You will see a demonstration of this.

Summary

In this lesson, you should have learned how to:

- Describe the characteristics of a class
- Define an object as an instance of a class
- Instantiate an object and access its fields and methods
- Describe how objects are stored in memory
- Instantiate an array of objects
- Describe how an array of objects is stored in memory
- Declare an object as a field



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Practices Overview

- 6-1: Creating Classes for the Soccer League
- 6-2: Creating a Soccer Game



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Manipulating and Formatting the Data in Your Program

7



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



ORACLE®

Moisés Ocampo Sámano (aos_moyer@hotmail.com) has a
non-transferable license to use this Student Guide.

Objectives

After completing this lesson, you should be able to:

- Describe the `String` class and use some of the methods of the `String` class
- Use the JDK API documentation to search for and learn how to use a class
- Describe the `StringBuilder` class
- Explain what a constant is and how to use it
- Explain the difference between promoting and casting of variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



String Class

```
String hisName = "Fred Smith"; — Standard syntax
```

The new keyword can be used,
but it is not best practice:

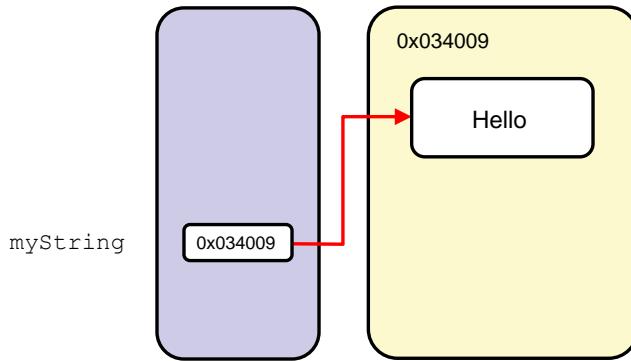
```
String herName = new String("Anne Smith");
```



- A String object is immutable; its value cannot be changed.
- A String object can be used with the string concatenation operator symbol (+) for concatenation.
- The String class is one of the many classes included in the Java class libraries. The String class provides you with the ability to store a sequence of characters. You will use the String class frequently throughout your programs. Therefore, it is important to understand some of the special characteristics of strings in the Java programming language. Because a String object is immutable, its value cannot be changed. (There are technical reasons, beyond the scope of this course, as to why this immutability is useful. One simple example is that this immutability ensures that a String can be used by several different classes safely because it cannot be changed.)
- Creating a String object using the new keyword creates two String objects in memory, whereas creating a String object by using a string literal creates only one object; therefore, the latter practice is more memory-efficient. To avoid the unnecessary duplication of String objects in memory, create String objects without using the new keyword.

Concatenating Strings

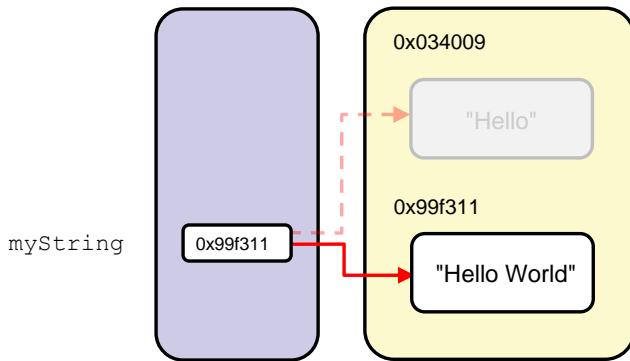
```
String myString = "Hello";
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Concatenating Strings

```
String myString = "Hello";
myString = myString.concat(" World");
```



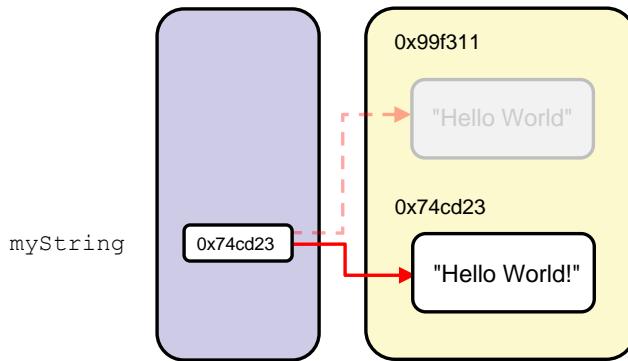
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Here is the string `"World"` being concatenated to the original string. The `concat` method is being used here, but whether you use that or the concatenation operator (`+`), a new `String` object is created and a new `String` reference is returned that points to this new object.

In the diagram, this is shown by the fact that the `String` reference `myString` is no longer `0x034009`, and because that object is no longer referred to, it is now inaccessible and will be garbage collected.

Concatenating Strings

```
String myString = "Hello";
myString = myString.concat(" World");
myString = myString + "!"
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

String Method Calls with Primitive Return Values

A method call can return a single value of any type.

- An example of a method of primitive type `int`:

```
String hello = "Hello World";
int stringLength = hello.length();
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Like most classes, the `String` class has a number of useful methods. Almost all of these methods do their useful work by returning a single value (Java allows only a single return from a method). The return type (essentially the type of the method) can be a primitive or a reference to an object.

To be able to use the return value in your code, you will typically use the assignment operator to assign the value (or reference) to a type that you have declared for this purpose.

The example in the slide shows the use of reference `hello` to call the method `length`. Because the object this reference refers to is the string `Hello World`, this method call will return the value `11` and place it in the variable `stringLength`. `int` is the type of the method `length`.

String Method Calls with Object Return Values

Method calls returning objects:

```
String greet = " HOW ".trim();
String lc = greet + "DY".toLowerCase();
```

Or

```
String lc = (greet + "DY").toLowerCase();
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This example shows several method calls that return object references.

First, the `String` object " HOW " is instantiated and has the method `trim` called on it. Because a string literal returns an object reference, this is exactly the same as calling the method `trim` on the reference. Notice that the string " HOW " has two spaces on either side of the word. The string returned will be just three characters long because these spaces will be removed. This new string will be referenced by `greet`.

The next example shows a method call not being assigned to a type, but simply used in an expression. The method `toLowerCase` is called on the string "DY", returning "dy". `lc` now references an object containing "HOWdy".

Finally, note how an alternative version with parentheses ensures that the two strings are concatenated (creating a new string) before `toLowerCase` is called. `lc` now references an object containing "howdy".

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Java API Documentation

Consists of a set of webpages;

- Lists all the classes in the API
 - Descriptions of what the class does
 - List of constructors, methods, and fields for the class
- Highly hyperlinked to show the interconnections between classes and to facilitate lookup
- Available on the Oracle website at:
<https://docs.oracle.com/javase/10/docs/api/overview-summary.html>



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

All of the Java technology JDKs contain a series of prewritten classes for you to use in your programs. These Java technology class libraries are documented in the Java API documentation for the version of the JDK that you are using. The class library specification is a series of HTML webpages that you can load in your web browser.

A Java class library specification is a very detailed document outlining the classes in the API. Every API includes documentation describing the use of classes and their fields and methods. When you are looking for a way to perform a certain set of tasks, this documentation is the best source for information about the classes in the Java class libraries.

You learn more about constructors in the “Using Encapsulation” lesson.

JDK 10 API Documentation

Select one of the Module.



The packages for the selected module are listed here.



The classes for the selected package are listed here.



Java SE 10 & JDK 10

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Modules

java.activation
java.base **java.base** ←
java.compiler
java.corba

Java SE 9 & JDK 9

ALL CLASSES ALL PACKAGES ALL MODULES

java.base Packages

java.io
java.lang **java.lang** ←
java.lang.annotation
java.lang.invoke
java.lang.module

StrictMath
String **String** ←
StringBuffer
StringBuilder
System
System.LoggerFinder
Thread

Class String

java.lang.Object
java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence

The String class represents character strings. All string literals in Java programs are constant; their values cannot be changed after they are created. String str = "abc";

Strings are constant; their values cannot be changed after they are created. String str = "abc";

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

Details about the class selected

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



In the screenshot in the slide, you can see the three main panels of the webpage.

Modules are a new programming construct introduced in JDK 9 and you will learn more about modules in Lesson 16 of this course.

The top-right panel allows you to select a module. The packages of a particular module are listed. You can select a class from a particular package. But if you do not know the package of a particular class, you can select All Classes.

The bottom-left panel gives the list of packages in a module, you can then select a particular package and then all the classes in that are listed.

In this panel, the class `String` has been selected, populating the main panel on the right with the details of the class `String`. The main panel on the right contains a lot of information about the class, so you need to scroll down to access the information you need.

Java Platform SE and JDK Version 10 API Specification

This document is divided into three sections:

- **Java SE:**
 - The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing.
 - These APIs are in modules whose names start with `java`.
- **JDK**
 - The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform.
 - These APIs are in modules whose names start with `jdk`.
- **JavaFX:**
 - The JavaFX APIs define a set of user-interface controls, graphics, media, and web packages for developing rich client applications.
 - These APIs are in modules whose names start with `javafx`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Java Platform SE 10: Method Summary

```
public int charAt(String str)
```

The return type
of the method

The name of
the method

The type of the
parameter that must be
passed into the method

Method Summary		Description		
All Methods	Static Methods	Instance Methods	Concrete Methods	Deprecated Methods
char	charAt(int index)	Returns the char value at the specified index.		
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.		
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.		
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.		
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.		



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If you keep scrolling through the details for the `String` class, you will come to the list of methods (only a small subset of this list is shown here).

This master list of methods gives the basic details for the method. In this case, you can see that the name of the method is `charAt`, its type is `char`, and it requires an index (of type `int`) to be passed in. There is also a brief description that this method returns the `char` value at a particular index in the string. For any of the methods, the method name and the parameter types are hyperlinked so that you can get more details.

Java Platform SE 10: Method Detail

Click here to get the detailed description of the method.

```
int indexOf(String str)
    Returns the index within this string of the first occurrence of the specified substring.

int indexOf(String str, int fromIndex)
    Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
```

Detailed description for the indexOf() method

```
indexOf
public int indexOf(String str)
    Returns the index within this string of the first occurrence of the specified substring.
    The returned index is the smallest value k for which:
        this.startsWith(str, k)

    If no such value of k exists, then -1 is returned.

Parameters:
    str - the substring to search for.

Returns:
    the index of the first occurrence of the specified substring, or -1 if there is no such occurrence.
```

Further details about parameters and return value are shown in the method list.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

indexOf Method Example

```
1 String phoneNum = "404-543-2345";           The 1-arg version
2 int idx1 = phoneNum.indexOf('-');             |
3 System.out.println("index of first dash: "+ idx1);   |
4                                         The 2-arg version
5
6 int idx2 = phoneNum.indexOf('-', idx1+1);
7 System.out.println("second dash idx: "+idx2);
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This example shows how to get the location of the first '-' character by using the 1-arg version of `indexOf`, and then by using the 2-arg version to get the location of the second '-'.

If you wanted to convert the phone number to an `int`, you could do something like this:

1. Find the dashes by using the `indexOf` method (as shown above).
2. Build a new `String` without dashes by using the `substring` method and concatenation.
3. Convert this `String` to an `int` by using the `parseInt` method of `Integer`.

The `parseInt` method of the `Integer` class is covered in the lesson “Using Encapsulation.”

Exercise 7-1: Use indexOf and substring Methods

In this exercise, you get and display a customer's first name.

1. Open the project **Exercise_07-1** in NetBeans.
2. Use the `indexOf` method to get the index for the space character (" ") within `custName`. Assign it to `spaceIdx`.
3. Use the `substring` method and the `spaceIdx` to get the first name portion of `custName`.
 - Assign it to `firstName`.
 - Print `firstName`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Using the `String` class
- Using the Java API docs
- **Using the `StringBuilder` class**
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



StringBuilder Class

StringBuilder provides a mutable alternative to String. StringBuilder:

- Is instantiated using the new keyword
- Has many methods for manipulating its value
- Provides better performance because it is mutable
- Can be created with an initial capacity

String is still needed because:

- It may be safer to use an immutable object
- A method in the API may require a string
- It has many more methods not available on StringBuilder



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The StringBuilder class is “mutable.” This means that it can be changed in place. You will recall that when you modify the value of a String variable, a new String object is created for the new value. String objects are “immutable.” A String object’s value cannot be changed.

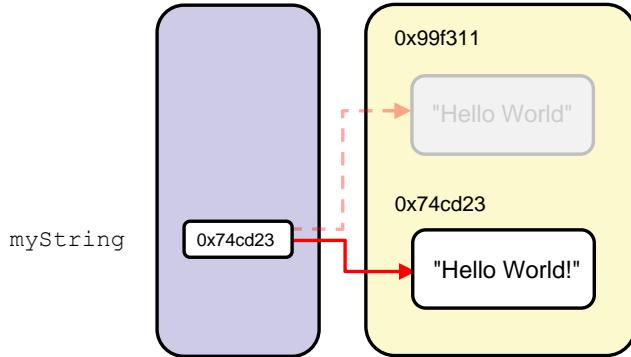
- Unlike String, there is no shortcut to instantiate a StringBuilder. It is simply instantiated like any other object by using the new keyword.
- A small sampling of the StringBuilder methods for manipulation of data values are: append, delete, insert, and replace.
- StringBuilder provides better performance because it does not create new objects in memory whenever a change is made. Performance is also benefited whenever you can set an initial capacity for the object, as opposed to letting it grow and allocate memory dynamically.
- StringBuilder is not a complete replacement for String, but it is more suitable if many modifications are likely to be made to its value.

StringBuilder Advantages over String for Concatenation (or Appending)

String Concatenation

- Costly in terms of creating new objects

```
String myString = "Hello";
myString = myString + " World";
```

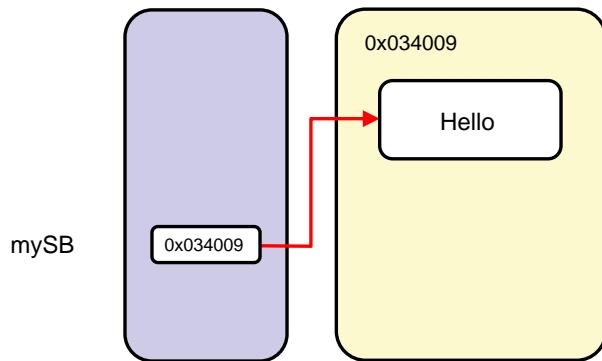


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This slide offers a reminder of what happens when the strings "Hello" and " World" are concatenated. A new String object is created, and the reference for that object is assigned to myString.

StringBuilder: Declare and Instantiate

```
StringBuilder mySB = new StringBuilder("Hello");
```

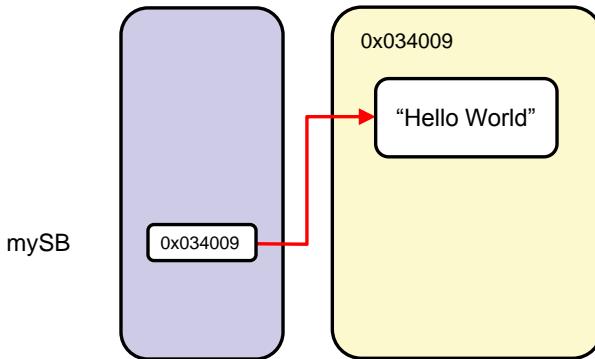


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This diagram shows the start of a sequence involving a `StringBuilder`. A `new StringBuilder` is instantiated, populated with the string "Hello", and the reference for this new object is assigned to `mySB`.

StringBuilder Append

```
StringBuilder mySB = new StringBuilder("Hello");
mySB.append(" World");
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

To append the string " World", all you need to do is call the `append` method and pass in "World". Note that no assignment (`=`) is necessary because there is already a reference to the `StringBuilder` object, and this `StringBuilder` object now contains a representation of the combined strings "Hello World".

Even if you did assign the return type of the `append` method (which is `StringBuilder`), there would still be no object creation cost; the `append` method modifies the current object and returns the reference to that object, the one already contained in `mySB`.

Quiz



Which of the following statements are true? (Choose all that apply.)

- a. The dot (.) operator creates a new object instance.
- b. The String class provides you with the ability to store a sequence of characters.
- c. The Java API specification contains documentation for all of the classes in a Java technology product.
- d. String objects cannot be modified.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b, c, d

Exercise 7-2: Instantiate the `StringBuilder` object

1. Open the project `Exercise_07-2` or continue editing the previous exercise.
2. Instantiate a `StringBuilder` object (`sb`), initializing it to `firstName`, using the `StringBuilder` constructor.
3. Use the `append` method of the `StringBuilder` to append the last name back onto the first name. You can just use a `String` literal for the last name. Print the `StringBuilder` object and test your code. It should show the full name.
4. (Optional) Can you append the last name without using a `String` literal?



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Primitive Data Types

- **Integral types** (`byte`, `short`, `int`, and `long`)
- **Floating point types** (`float` and `double`)
- **Textual type** (`char`)
- **Logical type** (`boolean`)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Some New Integral Primitive Types

Type	Length	Range
<code>byte</code>	8 bits	-2^7 to $2^7 - 1$ (-128 to 127, or 256 possible values)
<code>short</code>	16 bits	-2^{15} to $2^{15} - 1$ (-32,768 to 32,767, or 65,535 possible values)
<code>int</code>	32 bits	-2^{31} to $2^{31} - 1$ (-2,147,483,648 to 2,147,483,647, or 4,294,967,296 possible values)
<code>long</code>	64 bits	-2^{63} to $2^{63} - 1$ (-9,223,372,036854,775,808 to 9,223,372,036854,775,807, or 18,446,744,073,709,551,616 possible values)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

There are four integral primitive types in the Java programming language. You have already been using the `int` data type, so the focus here is on the other three. Integral types are used to store numbers that do not have decimal portions. They are shown here in order of size.

- **`byte`:** If you need to store people's ages, a variable of type `byte` would work because `byte` types can accept values in that range.
- **`short`:** A `short` will hold 16 bits of data.
- **`long`:** When you specify a literal value for a `long` type, put a capital `L` to the right of the value to explicitly state that it is a `long` type. Integer literals are assumed by the compiler to be of type `int` unless you specify otherwise by using an `L` indicating `long` type.
- You can express any of the integral types as `binary` (0s and 1s). For instance, a `binary` expression of the number 2 is shown as an allowed value of the `byte` integral type. The `binary` value is `0b10`. Notice that this value starts with `0b` (that is, zero followed by either a lowercase or uppercase letter `B`). This indicates to the compiler that a `binary` value follows.

Examples of allowed literal values:

- `byte = 2, -114, 0b10` (binary number)
- `short = 2, -32699`
- `int (default type for integral literals) = 2, 147334778, 123_456_678`
- `long = 2, -2036854775808L, 1`

Note: The only reason to use the `byte` and `short` types in programs is to save memory consumption. Because most modern desktop computers contain an abundance of memory, most desktop application programmers do not use `byte` and `short` types. This course uses primarily `int` and `long` types in the examples.

Floating Point Primitive Types

Type	Float Length
<code>float</code>	32 bits
<code>double</code> (default type for floating point literals)	64 bits

Example:

```
public float pi = 3.141592F;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

There are two types for floating point numbers: `float` and `double`. Again, the focus is on the new data type here, the `float`. Floating point types are used to store numbers with values to the right of the decimal point, such as 12.24 or 3.14159.

- `float` is used to store smaller floating point numbers. A float variable can hold 32 bits.
- Floating point values are assumed to be of type `double` unless you specify by putting a capital `F` (`float`) to the right of the value to explicitly state that it is a `float` type, not a `double` type.

Examples of allowed literal values:

```
float = 99F, -327456, 99.01F, 4.2E6F (engineering notation for 4.2 * 106)  
double = -1111, 2.1E12, 99970132745699.999
```

Note: Use the `double` type when a greater range or higher accuracy is needed.

Textual Primitive Type

- The only primitive textual data type is `char`.
- It is used for a single character (16 bits).
- Example:

— `public char colorCode = 'U';`

Single quotes must be used with `char` literal values.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Another data type that you use for storing and manipulating data is single-character information. The primitive type used for storing a single character (such as a 'y') is `char`, which is 16 bits in size. The `Shirt` class shows the use of one textual literal value to specify the default value for a `colorCode`:

```
public char colorCode = 'U';
```

When you assign a literal value to a `char` variable, you must use single quotation marks around the character as shown in the code example above.

Java Language Trivia: Unicode

- Unicode is a standard character encoding system.
 - It uses a 16-bit character set.
 - It can store all the necessary characters from most languages.
 - Programs can be written so they display the correct language for most countries.

Character	UTF-16	UTF-8	UCS-2
A	0041	41	0041
c	0063	63	0063
Ø	00F6	C3 B6	00F6
𩷶	4E9C	E4 BA 9C	4E9C
	D834 DD1E	F0 9D 84 9E	N/A



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Did You Know? Many older computer languages use American Standard Code for Information Interchange (ASCII), an 8-bit character set that has an entry for every English character, punctuation mark, number, and so on.

The Java programming language uses a 16-bit character set called Unicode that can store all the necessary displayable characters from the vast majority of languages used in the modern world. Therefore, your programs can be written so that they work correctly and display the correct language for most countries. Unicode contains a subset of ASCII (the first 128 characters).

Constants

- Variable (can change):
 - `double salesTax = 6.25;`
 - Constant (cannot change):
 - `final int NUMBER_OF_MONTHS = 12;`
- The `final` keyword causes a variable to be read only.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this lesson, you have learned about variables that have values that you can change. In this section, you learn how to use constants to represent values that cannot change.

Assume that you are writing part of a scheduling application, and you need to refer to the number of months in a year. Make the variable a constant by using the `final` keyword to inform the compiler that you do not want the value of the variable to be changed after it has been initialized. Example:

```
final int NUMBER_OF_MONTHS = 12;
```

Any values that do not need to change are good candidates for a constant variable (for example, `MAX_COUNT`, or `PI`).

If someone attempts to change the value of a constant after it has already been assigned a value, the compiler gives an error message. If you modify your code to provide a different value for the constant, you need to recompile your program.

Guidelines for Naming Constants

You should name constants so that they can be easily identified. Generally, constants should be capitalized, with words separated by an underscore (`_`).

Quiz



The variable declaration `public int myInteger=10;` adheres to the variable declaration and initialization syntax.

- a. True
- b. False



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- **Using the remaining numeric operators**
- Promoting and casting variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Modulus Operator

Purpose	Operator	Example	Comments
Remainder	$\%$ modulus	num1 = 31; num2 = 6; mod = num1 % num2; mod is 1	Remainder finds the remainder of the first number divided by the second number. $\begin{array}{r} 5 \text{ R } 1 \\ 6 \overline{)31} \\ 30 \\ \hline 1 \end{array}$ Remainder always gives an answer with the same sign as the first operand.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Combining Operators to Make Assignments

Purpose	Operator	Examples <code>int a = 6, b = 2;</code>	Result
Add to and assign	<code>+=</code>	<code>a += b</code>	<code>a = 8</code>
Subtract from and assign	<code>-=</code>	<code>a -= b</code>	<code>a = 4</code>
Multiply by and assign	<code>*=</code>	<code>a *= b</code>	<code>a = 12</code>
Divide by and assign	<code>/=</code>	<code>a /= b</code>	<code>a = 3</code>
Get remainder and assign	<code>%=</code>	<code>a %= b</code>	<code>a = 0</code>



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

More on Increment and Decrement Operators

Operator	Purpose	Example
++	Preincrement (<code>++variable</code>)	<code>int id = 6; int newId = ++id; id is 7, newId is 7</code>
	Postincrement (<code>variable++</code>)	<code>int id = 6; int newId = id++; id is 7, newId is 6</code>
--	Predecrement (<code>--variable</code>)	<i>(same principle applies)</i>
	Postdecrement (<code>variable--</code>)	



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You have used increment and decrement operators before, placing them *after* the variable that you wish to affect. But did you know that these operators can come *before* (preincrement and predecrement) or *after* (postincrement and postdecrement) a variable.

When you put the `++` or `--` operator *before* a variable, the value is changed immediately. When you put the operator *after* the variable, it is not changed until after that expression is evaluated.

- In the first code example above, `id` is initialized to 6. In the next line, you see `newId = ++id`. Because the operator precedes `id`, this increment is immediately evaluated and, therefore, the value assigned to `newId` is 7.
- In the second code example, the `++` operator follows `id`, rather than precedes it. `id` was incremented after the assignment occurred. Therefore, `newId` is 6.
- These same behaviors apply to a decrement (`--`) operator, in regard to its placement before or after the variable.

Increment and Decrement Operators (++ and --)

Examples:

```
1 int count=15;
2 int a, b, c, d;
3 a = count++;
4 b = count;
5 c = ++count;
6 d = count;
7 System.out.println(a + ", " + b + ", " + c + ", " + d);
```

Output:

15, 16, 17, 17



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows basic use of the increment and decrement operators:

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```

The result of this code fragment is:

15, 16, 17, 17

Discussion: What is the result of the following code?

```
int i = 16;
System.out.println(++i + " " + i++ + " " + i);
```

Topics

- Using the `String` class
- Using the Java API docs
- Using the `StringBuilder` class
- Doing more with primitive data types
- Using the remaining numeric operators
- Promoting and casting variables

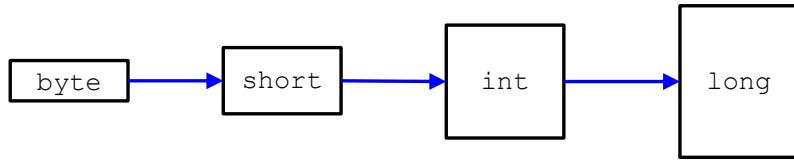


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

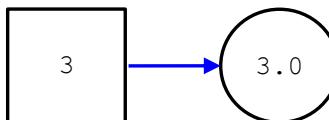


Promotion

- Automatic promotions:
 - If you assign a smaller type to a larger type



- If you assign an integral type to a floating point type



- Examples of automatic promotions:
 - `long intToLong = 6;`
 - `double int.ToDouble = 3;`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In some circumstances, the compiler changes the type of a variable to a type that supports a larger size value. This action is referred to as a *promotion*. Some promotions are done automatically by the compiler. These promotions include:

- If you assign a smaller type (on the right of the =) to a larger type (on the left of the =)
- If you assign an integral type to a floating point type (However, in some cases, such as an assignment of long to float, this could lead to loss of data.)

Caution with Promotion

Equation:

$$55555 * 66666 = 3703629630$$

Example of potential issue:

```
1 int num1 = 55555;  
2 int num2 = 66666;  
3 long num3;  
4 num3 = num1 * num2;           //num3 is -591337666
```

Example of potential solution:

```
1 int num1 = 55555;  
2 long num2 = 66666; ————— Changed from int to long  
3 long num3;  
4 num3 = num1 * num2;           //num3 is 3703629630
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Before being assigned to a variable, the result of an equation is placed in a temporary location in memory. The location's size is always equal to the size of an `int` type or the size of the largest data type used in the expression or statement. For example, if your equation multiplies two `int` types, the container size will be an `int` type in size, or 32 bits.

If the two values that you multiply yield a value that is beyond the scope of an `int` type, (such as $55555 * 66666 = 3,703,629,630$, which is too big to fit in an `int` type), the `int` value must be truncated to fit the result into the temporary location in memory. This calculation ultimately yields an incorrect answer because the variable for your answer receives a truncated value (regardless of the type used for your answer). To solve this problem, set at least one of the variables in your equation to the `long` type to ensure the largest possible temporary container size.

Caution with Promotion

Equation:

$$7 / 2 = 3.5$$

Example of potential issue:

```
1 int num1 = 7;  
2 int num2 = 2;  
3 double num3;  
4 num3 = num1 / num2; //num3 is 3.0
```

Example of potential solution:

```
1 int num1 = 7;  
2 double num2 = 2; ————— Changed from int to double  
3 double num3;  
4 num3 = num1 / num2; //num3 is 3.5
```



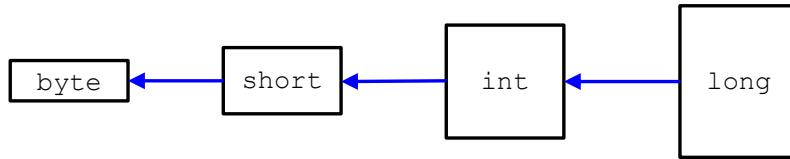
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The same issue occurs with other data types. Before being assigned to a variable, the result of an equation is placed in a temporary location in memory. The location's size is always equal to the size of the largest data type used in the expression or statement. For example, if your equation divides two `int` types, the container size will be an `int` type in size, or 32 bits.

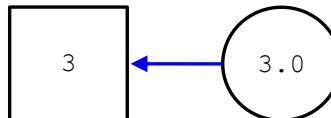
If the two values that you use yield a value that is beyond the scope of an `int` type, (such as $7 / 2 = 3.5$), the value must be truncated to fit the result into the temporary location in memory. This calculation ultimately yields an incorrect answer because the variable for your answer receives a truncated value (regardless of the type used for your answer). To solve this problem, set at least one of the variables in your equation to the `double` type to ensure the largest possible temporary container size.

Type Casting

- When to cast:
 - If you assign a larger type to a smaller type



- If you assign a floating point type to an integral type



- Examples of casting:
 - `int longToInt = (int)20L;`
 - `short doubleToShort = (short)3.0;`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Caution with Type Casting

Example of potential issue:

```
1 int myInt;  
2 long myLong = 123987654321L;  
3 myInt = (int) (myLong); // Number is "chopped"  
4 // myInt is -566397263
```

Safer example of casting:

```
1 int myInt;  
2 long myLong = 99L;  
3 myInt = (int) (myLong); // No data loss, only zeroes.  
4 // myInt is 99
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The loss of precision with casting can sometimes lead to situations where numbers are truncated, leading to errors in calculations.

Caution with Type Casting

- Be aware of the possibility of lost precision.

Example of potential issue:

```
1 int myInt;  
2 double myPercent = 51.9;  
3 myInt = (int) (myPercent); // Number is "chopped"  
4 // myInt is 51
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If you type cast a `float` or `double` value with a fractional part to an integral type such as an `int`, all decimal values are lost. However, this method of type casting is sometimes useful if you want to truncate the number down to the whole number (for example, 51.9 becomes 51).

Using Promotion and Casting

Example of potential issue:

```
1 int num1 = 53; // 32 bits of memory to hold the value
2 int num2 = 47; // 32 bits of memory to hold the value
3 byte num3;     // 8 bits of memory reserved
4 num3 = (num1 + num2); // causes compiler error
```

Solution using a larger type for num3:

```
1 int num1 = 53;
2 int num2 = 47;
3 int num3;           ————— Changed from byte to int
4 num3 = (num1 + num2);
```

Solution using casting:

```
1 int num1 = 53; // 32 bits of memory to hold the value
2 int num2 = 47; // 32 bits of memory to hold the value
3 byte num3;     // 8 bits of memory reserved
4 num3 = (byte) (num1 + num2); // no data loss
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Assigning a variable or an expression to another variable can lead to a mismatch between the data types of the calculation and the storage location that you are using to save the result. Specifically, the compiler will either recognize that precision will be lost and not allow you to compile the program, or the result will be incorrect. To fix this problem, variable types have to be either promoted to a larger size type, or type cast to a smaller size type. In the above example, the compiler assumes that because you are adding `int` values, the result will overflow the space allocated for a `byte`.

A `byte`, though smaller than an `int`, is large enough to store a value of 100. However, the compiler will not make this assignment and, instead, issues a “possible loss of precision” error because a `byte` value is smaller than an `int` value. To fix this problem, you can either type cast the right-side data type down to match the left-side data type, or declare the variable on the left side (`num3`) to be a larger data type, such as an `int`.

Compiler Assumptions for Integral and Floating Point Data Types

- Most operations result in an `int` or `long`:
 - `byte`, `char`, and `short` values are automatically promoted to `int` prior to an operation.
 - If an expression contains a `long`, the entire expression is promoted to `long`.
- If an expression contains a floating point, the entire expression is promoted to a floating point.
- All literal floating point values are viewed as `double`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The Java technology compiler makes certain assumptions when it evaluates expressions. You must understand these assumptions to make the appropriate type casts or other accommodations. The next few slides give examples.

Automatic Promotion

Example of potential problem:

```
short a, b, c;  
a = 1 ; } a and b are automatically promoted to integers.  
b = 2 ;  
c = a + b ; //compiler error
```

Example of potential solutions:

- Declare `c` as an `int` type in the original declaration:
`int c;`
- Type cast the `(a+b)` result in the assignment line:
`c = (short) (a+b);`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the following example, an error occurs because two of the three operands (`a` and `b`) are automatically promoted from a `short` type to an `int` type before they are added. In the last line, the values of `a` and `b` are converted to `int` types and the converted values are added to give an `int` result. Then the assignment operator (`=`) attempts to assign the `int` result to the `short` variable (`c`). However, this assignment is illegal and causes a compiler error.

The code works if you do either of the following:

- Declare `c` as an `int` in the original declaration:
`int c;`
- Type cast the `(a+b)` result in the assignment line:
`c = (short) (a+b);`

Using a long

```
1 public class Person {  
2  
3     public int ageYears = 32;  
4  
5     public void calculateAge() {  
6         int ageDays = ageYears * 365;  
7         long ageSeconds = ageYears * 365 * 24L * 60 * 60;  
8  
9         System.out.println("You are " + ageDays + " days old.");  
10        System.out.println("You are " + ageSeconds + " seconds old.");  
11    } // end of calculateAge method  
12 } // end of class
```

Using the L to indicate a long will result in the compiler recognizing the total result as a long.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The code example uses principles from this section to calculate a person's age in days and seconds. Because the `ageSeconds` variable is declared as a `long`, one of the literal values used as operands in the assigned expression must be initialized as a `long` value ('L') so that the compiler will allow the assignment.

Using Floating Points

Example of potential problem:

Expressions are automatically promoted to floating points.

```
int num1 = 1 + 2 + 3 + 4.0;           //compiler error  
int num2 = (1 + 2 + 3 + 4) * 1.0;     //compiler error
```

Example of potential solutions:

- Declare num1 and num2 as double types:

```
double num1 = 1 + 2 + 3 + 4.0;          //10.0  
double num2 = (1 + 2 + 3 + 4) * 1.0;    //10.0
```

- Type cast num1 and num2 as int types in the assignment line:

```
int num1 = (int)(1 + 2 + 3 + 4.0);      //10  
int num2 = (int)((1 + 2 + 3 + 4) * 1.0); //10
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If an expression contains a floating point, the entire expression is promoted to a floating point.

Floating Point Data Types and Assignment

- Example of potential problem:

```
float float1 = 27.9; //compiler error
```

- Example of potential solutions:

- The F notifies the compiler that 27.9 is a float value:

```
float float1 = 27.9F;
```

- 27.9 is cast to a float type:

```
float float1 = (float) 27.9;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Just as integral types default to int under some circumstances, values assigned to floating point types always default to a double type, unless you specifically state that the value is a float type.

For example, the following line causes a compiler error. Because 27.9 is assumed to be a double type, a compiler error occurs because a double type value cannot fit into a float variable.

```
float float1 = 27.9; //compiler error
```

Both of the following work correctly:

- The F notifies the compiler that 27.9 is a float value:

```
float float1 = 27.9F;
```

- 27.9 is cast to a float type:

```
float float1 = (float) 27.9;
```

Quiz



Which statements are true?

- a. There are eight primitive types built in to the Java programming language.
- b. byte, short, char, and long are the four integral primitive data types in the Java programming language.
- c. A boolean type variable holds true, false, and nil.
- d. short Long = 10; is a valid statement that adheres to the variable declaration and initialization syntax.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Exercise 7-3: Declare a Long, Float, and Char

1. Open the project **Practice_07-3** in NetBeans.
2. Declare a `long`, using the `L` to indicate a long value. Make it a very large number (in the billions).
3. Declare and initialize a `float` and a `char`
4. Print the `long` variable with a suitable label.
5. Assign the `long` to the `int` variable. Correct the syntax error by casting the `long` as an `int`.
6. Print the `int` variable. Note the change in value when you run it.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you experiment with the data types introduced in this lesson. You:

Declare and initialize variables

Cast one numeric type to another

Summary

In this lesson, you should have learned how to:

- Describe the `String` class and use some of the methods of the `String` class
- Use the JDK API documentation to search for and learn how to use a class
- Use the `StringBuilder` class to manipulate string data
- Create a constant by using the `final` keyword in the variable declaration
- Describe how the Java compiler can use promotion or casting to interpret expressions and avoid a compiler error



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Practices Overview

- 7-1: Manipulating Text



Creating and Using Methods

8



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



ORACLE®

Moisés Ocampo Sámano (aos_moyer@hotmail.com) has a
non-transferable license to use this Student Guide.

Objectives

After completing this lesson, you should be able to:

- Add an argument to a method
- Instantiate a class and call a method
- Overload a method
- Work with static methods and variables
- Convert data values using `Integer`, `Double`, and `Boolean` object types



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Using methods and constructors
- Method arguments and return values
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Basic Form of a Method

The void keyword indicates that the method does not return a value.

Empty parentheses indicate that no arguments are passed to the method.

```
1 public void display () {  
2     System.out.println("Shirt description: " + description);  
3     System.out.println("Color Code: " + colorCode);  
4     System.out.println("Shirt price: " + price);  
5 } // end of display method
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This is an example of a simple method that does not receive any arguments or return a value.

Calling a Method from a Different Class

```
1 public class ShoppingCart {  
2     public static void main (String[] args) {  
3         Shirt myShirt = new Shirt();  
4         myShirt.display();  
5     }  
6 }
```

Diagram annotations:

- A bracket under "myShirt" is labeled "Reference variable".
- A bracket under ".display()" is labeled "Dot operator".
- A bracket under "display()" is labeled "Method".

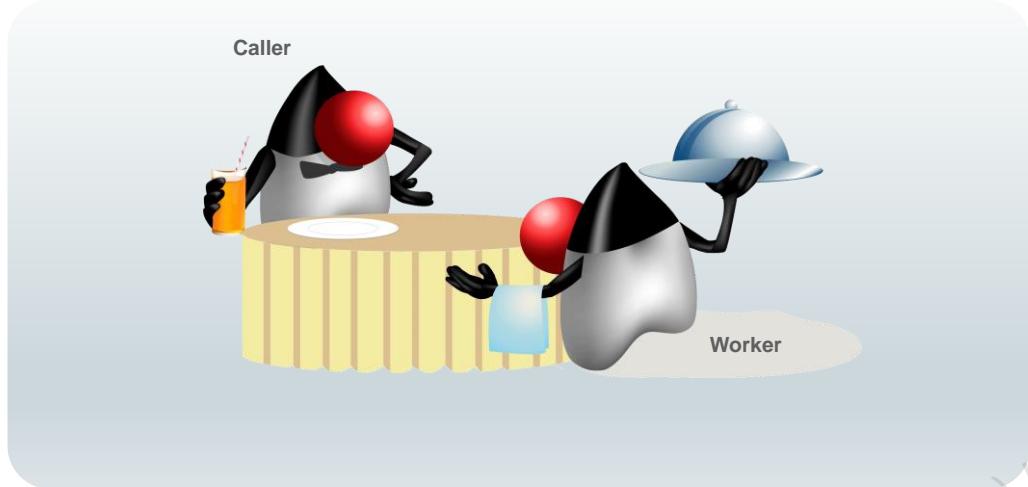
Output:

```
Item description:-description required-  
Color Code: U  
Item price: 0.0
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Caller and Worker Methods



In the previous example, the `ShoppingCart` class calls the `display` method on a `Shirt` object from within the `main` method. The `main` method is referred to as the *calling method* because it is invoking or “calling” another method to do some work. Conversely, the `display` method is referred to as the *worker method* because it does some work for the `main` method.

When a calling method calls a worker method, the calling method stops execution until the worker method is done. After the worker method has completed, program flow returns to the point after the method invocation in the calling method.

A Constructor Method

A constructor method is a special method that is invoked when you create an object instance.

- It is called by using the `new` keyword.
- Its purpose is to instantiate an object of the class and store the reference in the reference variable.

```
Shirt myShirt = new Shirt();
```

Constructor method is called.

- It has a unique method signature.

```
<modifier> ClassName()
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A constructor is invoked using the `new` keyword. Its job is to instantiate an object of the class and to provide a reference to the new object. If you do not write your own constructor in a class, Java will provide one for you. The constructor's name is the same as the class name.

In the `Shirt` example above, the reference returned by the `Shirt` constructor is assigned to the `myShirt` reference variable.

Writing and Calling a Constructor

```
1 public static void main(String[] args){  
2     Shirt myShirt = new Shirt()  
3 }  
  
1 public class Shirt {  
2     //Fields  
3     public String description;  
4     public char colorCode;  
5     public double price;  
6  
7     //Constructor  
8     public Shirt(){  
9         description = "--description required--";  
10        colorCode = 'U';  
11        price = 0.00;  
12    }  
13  
14    //Methods  
15    public void display(){  
16        System.out.println("Shirt description:" + description);  
17        System.out.println("Color Code: " + colorCode);  
18        System.out.println("Shirt price: " + price);  
19    }...  
20 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The constructor is the first method called when an object is instantiated. Its purpose is primarily to set default values.

Calling a Method in the Same Class

```
1 public class Shirt {  
2     public String description;  
3     public char colorCode;  
4     public double price;  
5  
6     public Shirt(){  
7         description = "--description required--";  
8         colorCode = 'U'  
9         price = 0.00;  
10    }  
11    display();           //Called normally  
12    this.display();     //Called using the 'this' keyword  
13}  
14  
15    public void display(){  
16        System.out.println("Shirt description:" + description);  
17        System.out.println("Color Code: " + colorCode);  
18        System.out.println("Shirt price: " + price);  
19    }  
20 ...
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Calling a method in the same class is very straightforward. You can simply use the method name without a reference. This is the same as when accessing a field; you can simply use the field name.

However, if you have local variables with similar names and you want to make it obvious that your code is accessing a field or method of the current object, you can use the `this` keyword with dot notation. `this` is a reference to the current object.

In this example, the `display` method is called twice from the constructor.

Topics

- Using constructors and methods
- **Method arguments and return values**
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Method Arguments and Parameters

- An **argument** is a value that is passed during a method call:

```
Calculator calc = new Calculator();  
double denominator = 2.0  
calc.calculate(3, denominator); //should print 1.5
```

Arguments

- A **parameter** is a variable defined in the method declaration:

```
public void calculate(int x, double y){  
    System.out.println(x/y);  
}
```

3 2.0
Parameters



Note: A value passed into the method when it is called is called an *argument*, whereas a variable that is defined in the method declaration is called a *method parameter*.

In this example, 3 and 2.0 are passed to be the values of x and y within the `calculate` method.

Method Parameter Examples

- Methods may have any number or type of parameters:

```
public void calculate0() {  
    System.out.println("No parameters");  
}
```

```
public void calculate1(int x) {  
    System.out.println(x/2.0);  
}
```

```
public void calculate2(int x, double y) {  
    System.out.println(x/y);  
}
```

```
public void calculate3(int x, double y, int z) {  
    System.out.println(x/y +z);  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Methods can take any number of parameters and use these values within the method code block.

Method Return Types

- Variables can have values of many different types:

short int double long char
String boolean int[] float byte
shirt

- Method calls can also return values of many different types:

short int double long char
String boolean int[] float byte
shirt

- How to make a method return a value:

- Declare the method to be a non-void return type.
- Use the keyword `return` within a method, followed by a value.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Variables may have values of many different types, including primitive data types, objects, and arrays.

Likewise, methods may return values of many different types, including primitive data types, objects, and arrays.

Note: Constructors are special. They cannot have a return type, not even void.

Method Return Types Examples

- Methods must `return` data that matches their return type:

```
public void printString(){  
    System.out.println("Hello");  
}
```

Void methods cannot return values in Java.

```
public String returnString(){  
    return("Hello");  
}
```

```
public int sum(int x, int y){  
    return(x + y);  
}
```

```
public boolean isGreater(int x, int y){  
    return(x > y);  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Void methods and constructors should not have a `return` statement. Void methods are incapable of returning a value in Java. The type of value a method returns must match the return type you declare. For instance, a `boolean` type method must return a `boolean`. A `String` type method must return a `String`.

Method Return Animation

- The following code examples produce equivalent results:

```
public static void main(String[] args) {  
    int num1 = 1, num2 = 2;  
    int result = num1 + num2;  
    System.out.println(result);  
}
```

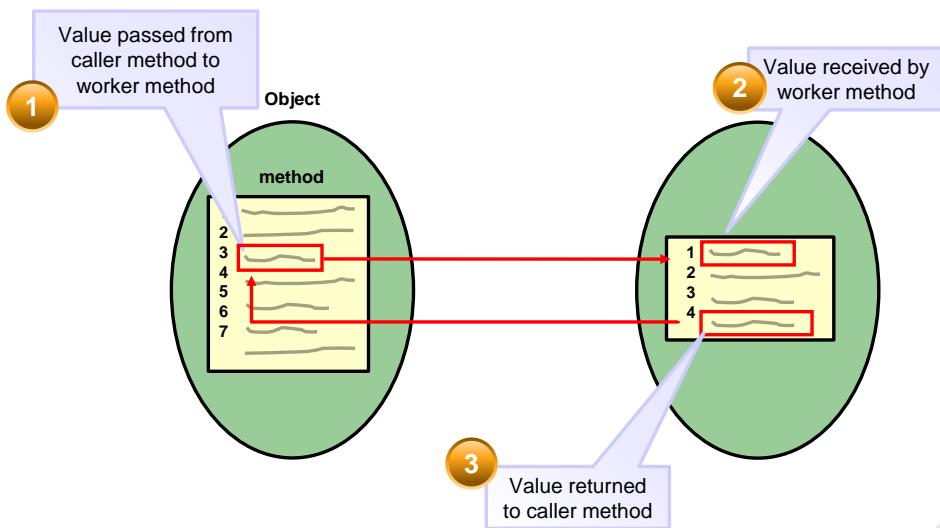
```
public static void main(String[] args) {  
    int num1 = 1, num2 = 2;  
    int result = sum(num1, num2);  
    System.out.println(result);  
}  
public int sum(int x, int y) {  
    return(x + y);  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the top example, `num1` and `num2` are added together. In the bottom example, this logic is put into the `sum` method. Values are passed to the `sum` method and added, with the resulting integer value being passed back and assigned to the `result` variable.

Passing Arguments and Returning Values



More Examples

```

1 public void setCustomerServices() {
2     String message = "Would you like to hear about "
3         +"special deals in your area?";
4     if (cust.isNewCustomer()) {
5
6         cust.sendEmail(message);
7     }
8 }
```

```

1 public class Customer{
2     public boolean isNew;
3
4     public boolean isNewCustomer(){
5         return isNew;      ————— Return a boolean
6     }
7     public void sendEmail(String message) {
8         // send email
9     }
10 }
```

String argument required



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Here you see a caller method, `setCustomerServices`, invoking worker methods in the `Customer` class.

- The example at the bottom of the slide shows the `Customer` class, which defines two methods:
 - `isNewCustomer` is defined with a return value of type `boolean`, but it does not define any input parameters.
 - `sendEmail` is defined with an input parameter of type `String`, called `message`. This method does not return a value.
- The example at top of the slide shows the `setCustomerServices` method in the `ShoppingCart` class invoking the methods of a `Customer` object by using dot notation (`object_reference.method`).
 - In line 4, `isNewCustomer` is called on the `cust` object reference. Because the method returns a `boolean`, the method invocation becomes a `boolean` expression evaluated by the `if` statement.
 - In line 6, `sendEmail` is called on the `cust` object reference, passing the `message` string as an argument.

Code Without Methods

```
1 public static void main(String[] args){  
2     Shirt shirt01 = new Shirt();  
3     Shirt shirt02 = new Shirt();  
4     Shirt shirt03 = new Shirt();  
5     Shirt shirt04 = new Shirt();  
6  
7     shirt01.description = "Sailor";  
8     shirt01.colorCode = 'B';  
9     shirt01.price = 30;  
10  
11    shirt02.description = "Sweatshirt";  
12    shirt02.colorCode = 'G';  
13    shirt02.price = 25;  
14  
15    shirt03.description = "Skull Tee";  
16    shirt03.colorCode = 'B';  
17    shirt03.price = 15;  
18  
19    shirt04.description = "Tropical";  
20    shirt04.colorCode = 'R';  
21    shirt04.price = 20;  
22 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Better Code with Methods

```
1 public static void main(String[] args){  
2     Shirt shirt01 = new Shirt();  
3     Shirt shirt02 = new Shirt();  
4     Shirt shirt03 = new Shirt();  
5     Shirt shirt04 = new Shirt();  
6  
7     shirt01.setFields("Sailor", 'B', 30);  
8     shirt02.setFields("Sweatshirt", 'G', 25);  
9     shirt03.setFields("Skull Tee", 'B', 15);  
10    shirt04.setFields("Tropical", 'R', 20);  
11 }
```

```
1 public class Shirt {  
2     public String description;  
3     public char colorCode;  
4     public double price;  
5  
6     public void setFields(String desc, char color, double price){  
7         this.description = desc;  
8         this.colorCode = color;  
9         this.price = price;  
10    }  
11 ...
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Even Better Code with Methods

```
1 public static void main(String[] args){  
2     Shirt shirt01 = new Shirt("Sailor", "Blue", 30);  
3     Shirt shirt02 = new Shirt("SweatShirt", "Green", 25);  
4     Shirt shirt03 = new Shirt("Skull Tee", "Blue", 15);  
5     Shirt shirt04 = new Shirt("Tropical", "Red", 20);  
6 }
```

```
1 public class Shirt {  
2     public String description;  
3     public char colorCode;  
4     public double price;  
5  
6     //Constructor  
7     public Shirt(String desc, String color, double price){  
8         setFields(desc, price);  
9         setColor(color);  
10    }  
11    public void setColor (String theColor){  
12        if (theColor.length() > 0)  
13            colorCode = theColor.charAt(0);  
14    }  
15 }  
16 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Taking advantage of a `Shirt` constructor can further reduce the amount of code needed in the `main` method.

Another issue is maintenance. Imagine if you wanted to change the constructor so that the color passed in is a `String`, but the instance variable, `colorCode`, remains a `char` type. You could create a method `setColor` that receives a `String` as an argument and then modifies it so that it sets `colorCode` correctly.

Remember, methods can call other methods (as shown by the call to `setColor`).

Variable Scope

```

1  public class Shirt {
2      public String description;
3      public char colorCode;
4      public double price;
5
6      public void setColor (String theColor) {
7          if (theColor.length() > 0)
8              colorCode = theColor.charAt(0);
9      }
10 }
11
12     public String getColor(){
13         return theColor; //Cannot find symbol
14     }
15
16 }
```

Annotations on the code:

- Instance variable (field)**: Points to the `colorCode` variable at line 3.
- Local variable**: Points to the `theColor` variable at line 6.
- Scope of theColor**: A callout box covers the `theColor` variable declaration and assignment at lines 6-8.
- Not scope of theColor**: A callout box covers the `theColor` variable reference at line 12, which is crossed out with a red circle.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This code illustrates the scope of two different types of variables. Variables live in the block where they are defined. This is called “scope.” The scope of a variable determines its accessibility and also how long you can count on its value to persist.

- The `colorCode` variable is an instance variable, usually called a field. It is a member of the `Shirt` class. It is accessible from any code within this class. The value of `fit` is stored only during the lifespan of an instance.
- `theColor` is a local variable. It is accessible only from within the `setColor` method. The value of `theColor` is deleted from memory when the method ends. Another way of saying this is that its scope is the `setColor` method.
- Regardless of whether a local variable is declared within a method, a loop (discussed later), or an `if` statement, its scope is always the block within which it is declared.
- In the example above, the `setColor` method uses the `charAt` method of the `String` object to extract the first character in the `theColor` String. It assigns it to the `fit` instance variable, which is a `char`.

Note: Local variables are stored in short-term memory, called “the stack,” whereas instance variables (fields) are stored in a longer-term area of memory called “the heap.”

Advantages of Using Methods

Methods:

- Are reusable
- Make programs shorter and more readable
- Make development and maintenance quicker
- Allow separate objects to communicate and to distribute the work performed by the program



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Exercise 8-1: Declare a `setColor` Method

1. Open the project `Exercise_08-1` in NetBeans.

In the `Item` class:

2. Declare a `setColor` method that takes a char as an argument (a color code) and returns a boolean. Return false if the `colorCode` is ' ' (a single space). Otherwise, assign the `colorCode` to the `color` field and return true.

In the `ShoppingCart` class:

3. Call the `setColor` method on `item1`. If it returns true, print `item1.color`. If it returns false, print an invalid color message.
4. Test the `setColor` method with both a valid color and an invalid one.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you declare a `setColor` method that takes a char as an argument, call the `setColor` method on `item1`, and test this method with both a valid color and invalid color.

Topics

- Using constructors and methods
- Method arguments and return values
- **Using static methods and variables**
- Understanding how arguments are passed to a method
- Overloading a method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Static Methods and Variables

The `static` modifier is applied to a method or variable.

It means the method/variable:

- Belongs to the *class* and is shared by all objects of that class
- Is *not unique* to an object instance
- Can be accessed without instantiating the class

Comparison:

- A **static variable** is shared by all objects in a class.
- An **instance variable** is unique to an individual object.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

So far you learned how to access variables and methods by creating an object instance of the class that the variable or method belongs to. The Java language allows you to declare a variable or method as `static`. This means that you can access it *without* creating an object instance of the class. Sometimes these are referred to as *class variables* or *class methods*.

Example: Setting the Size for a New Item

```
1 public class ItemSizes {  
2     static final String mSmall = "Men's Small";  
3     static final String mMed = "Men's Medium";  
4 }
```

Passing the static mMed variable
to the setSize method

```
Item item1 = new Item();  
item1.setSize(ItemSizes.mMed);
```

```
1 public class Item {  
2     public String size;  
3     public void setSize(String sizeArg) {  
4         this.size = sizeArg;  
5     }  
6 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the example above, the class `ItemSizes` contains two static variables of type `String`: `mSmall` and `mMed`. These are initialized to a description of a particular men's size. These values can be used without instantiating `ItemSizes`.

- The code snippet shown in the middle of the slide shows an `Item` object being instantiated and then the `setSize` method of the `Item` object is invoked, passing in `ItemSizes.mMed` as an argument.
- The code example at the bottom of the slide shows the `Item` class. It contains a `String` field, `size`. The `setSize` method requires a `String` parameter to set the `size` field.

Creating and Accessing Static Members

- To create a static variable or method:

```
static String mSmall;  
static void setMSmall(String desc);
```

- To access a static variable or method:

- From another class

```
ItemSizes.mSmall;  
ItemSizes.setMSmall("Men's Small");
```

- From within the class

```
mSmall;  
setMSmall("Men's Small");
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When to Use Static Methods or Fields

- Performing the operation on an individual object or associating the variable with a specific object type is not important.
- Accessing the variable or method before instantiating an object is important.
- The method or variable does not logically belong to an object, but possibly belongs to a utility class, such as the `Math` class, included in the Java API.
- Using constant values (such as `Math.PI`)



Some Rules About Static Fields and Methods

- Instance methods can access static methods or fields.
- Static methods cannot access instance methods or fields. Why?

```
1 public class Item{  
2     int itemID;  
3     public Item(){  
4         setId();  
5     }  
6     static int getID(){  
7         // whose itemID??  
8     }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Static Fields and Methods vs. Instance Fields and Methods

```
public class Item{  
    static int staticItemID;  
    int instanceItemID;  
    static main(){  
        Item item01 = new Item();  
  
        1 staticItemID = 6; ✓  
        2 instanceItemID = 3 ✗  
        3 showItemID(); ✗  
        4 item01.showItemID(); ✓  
  
    }  
    showItemID(){  
        ...println(staticItemID);  
        ...println(instanceItemID);  
    }  
}
```

Object (instance)
referenced by item01.

```
static int staticItemID;  
int instanceItemID;  
static main(){ ... }  
  
showItemID(){  
    5 ...println(staticItemID); ✓  
    6 ...println(instanceItemID); ✓  
}
```

Other instances
of Item



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The code example above shows a more complex example of an `Item` class that has an instance variable `instanceItemID` and a static variable `staticItemID`. In its `main` method, it instantiates an object referenced by `item01`. Look at the six lines of code and see the explanations below for why some work and some do not.

1. `staticItemID` is a static variable, and referenced from within a static method, `main`, so it does not need to access an instance.
2. `instanceItemID` is an instance variable, and referenced from within a static method, `main`, so it cannot be accessed unless a reference points to the particular object whose instance variable needs to be set.
3. `showItemID()` is a call to an instance method, and referenced from within a static method, `main`, so it cannot be accessed without a reference.
4. `item01.showItemID()` is a call to an instance method, but in this case the reference points to the particular object whose instance method needs to be called.
5. `...println(staticItemID)` refers to a static variable, but it is referred to from an instance. Instances can always access static variables.
6. `...println(instanceItemID)` refers to an instance variable, but it is referred to from an instance. No object reference is given, so it accesses the instance variable on the object itself.

Static Methods and Variables in the Java API

Examples:

- Some functionality of the `Math` class:
 - Exponential
 - Logarithmic
 - Trigonometric
 - Random
 - Access to common mathematical constants, such as the value PI (`Math.PI`)
- Some functionality of the `System` class:
 - Retrieving environment variables
 - Access to the standard input and output streams
 - Exiting the current program (`System.exit` method)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Certain Java class libraries, such as the `System` and the `Math` class, contain only static methods and variables. The `System` class contains utility methods for handling operating system-specific tasks. (They do not operate on an object instance.) For example, the `getProperties()` method of the `System` class gets information about the computer that you are using.

The `Math` class contains utility methods for math operations. Because these methods and variables are static, you do not need to create a new object every time you want your program to do some math.

Examining Static Variables in the JDK Libraries

The screenshot shows the Java API documentation for the `System` class. The class hierarchy tree on the left shows `java.lang` as the parent of `System`. The `System` class is highlighted with a red box. A blue bracket on the right points from the `out` field description to the `System` class in the hierarchy tree. Handwritten blue text next to the bracket states: "out is a static field of System and contains and is an object reference to a PrintStream object." Another handwritten note below the tree says: "System is a class in java.lang."

Field Detail

in

public static final InputStream in
The "standard" input stream. This stream is already open and ready to s

out

public static final PrintStream out
The "standard" output stream. This stream is already open and ready to For simple stand-alone Java applications, a typical way to write a line of System.out.println(data)

err

public static final PrintStream err
The "standard" error output stream. This stream is already open and rea Typically this stream corresponds to display output or another output de

See Also:

- `PrintStream.println()`, `PrintStream.println(boolean)`, `PrintStream.println(int)`, `PrintStream.println(long)`, `PrintStream.println(double)`



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The next few slides show how you might use the Java API documentation to find out more about `System.out.println()`. As you will see, this is a little unusual, because the class that has the methods that you need to investigate is not `System`. Rather, it is the class that is the type of the `out` field of the `System` object. Consider the following:

`System` is a class (in `java.lang`).

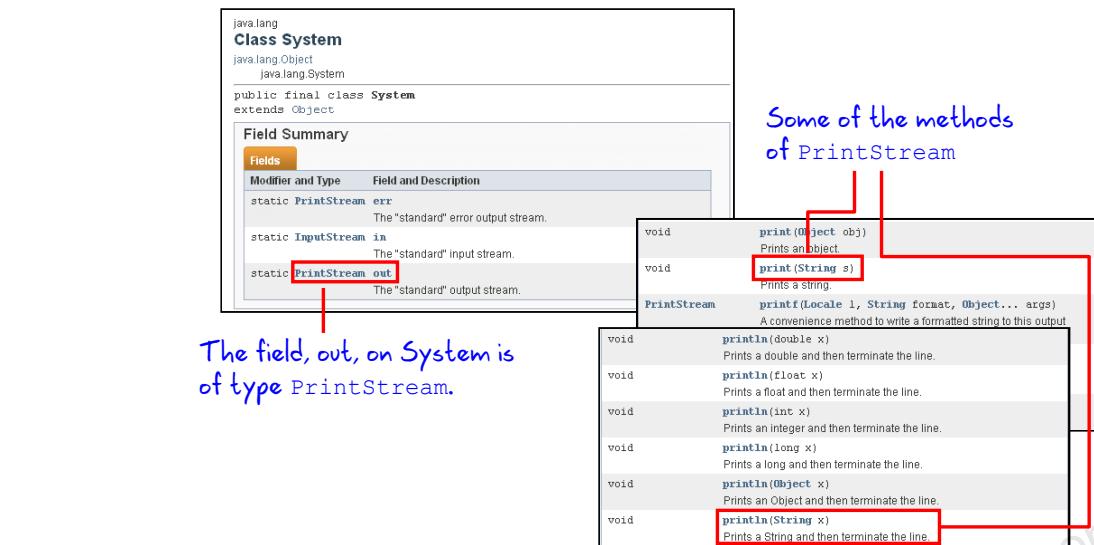
`out` is a static field of `System`. This is the reason that you reference it from the class name, not from an object instance: `System.out`

`out` is a reference type that allows calling `println()` on the object type it references.

To find the documentation:

1. Go to `System` class and find the type of the `out` field.
2. Go to the documentation for that field.
3. Review the methods available.

Using Static Variables and Methods: System.out.println



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

More Static Fields and Methods in the Java API

Java provides wrapper classes for each of the primitive data types.

- Boolean: Contains a single field of type boolean
- Double: Contains a single field of type double
- Integer: Contains a single field of type int

They also provide utility methods to work with the data.

Classes
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
ClassValue
Compiler
Double
Enum
Float
InheritableThreadLocal
Integer
Long



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A wrapper class is a class with the same name as one of the primitive data types. Wrapper classes are instantiated to contain a single value of the primitive type.

```
Integer myInt = new Integer(10);
```

These are very useful classes because they provide methods to help you work with the primitive values stored within.

Converting Data Values

- Methods often need to convert an argument to a different type.
- Most of the object classes in the JDK provide various conversion methods.

Examples:

- Converting a String to an int

```
int myInt1 = Integer.parseInt(s_Num);
```

- Converting a String to a double

```
double myDbl = Double.parseDouble(s_Num);
```

- Converting a String to boolean

```
boolean myBool = Boolean.valueOf(s_Bool);
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The examples show static conversion methods for Integer, Double, and Boolean.

There are also some conversion methods for the object classes (Integer, Double, and so on) that are not static. These methods are invoked on an object reference for one of these classes and convert the value of that specific object.

Topics

- Using constructors and methods
- Method arguments and return values
- Using static methods and variables
- **Understanding how arguments are passed to a method**
- Overloading a method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

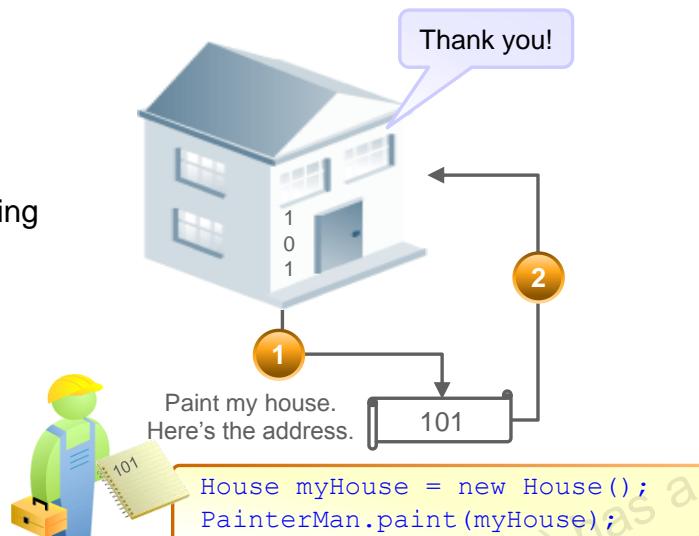


Moisés Ocampo Sámano (aos_moy78@hotmail.com) has a
non-transferable license to use this Student Guide.

Passing an Object Reference

An object reference is similar to a house address. When it is passed to a method:

- The object itself is not passed
- The method can access the object using the reference
- The method can act upon the object



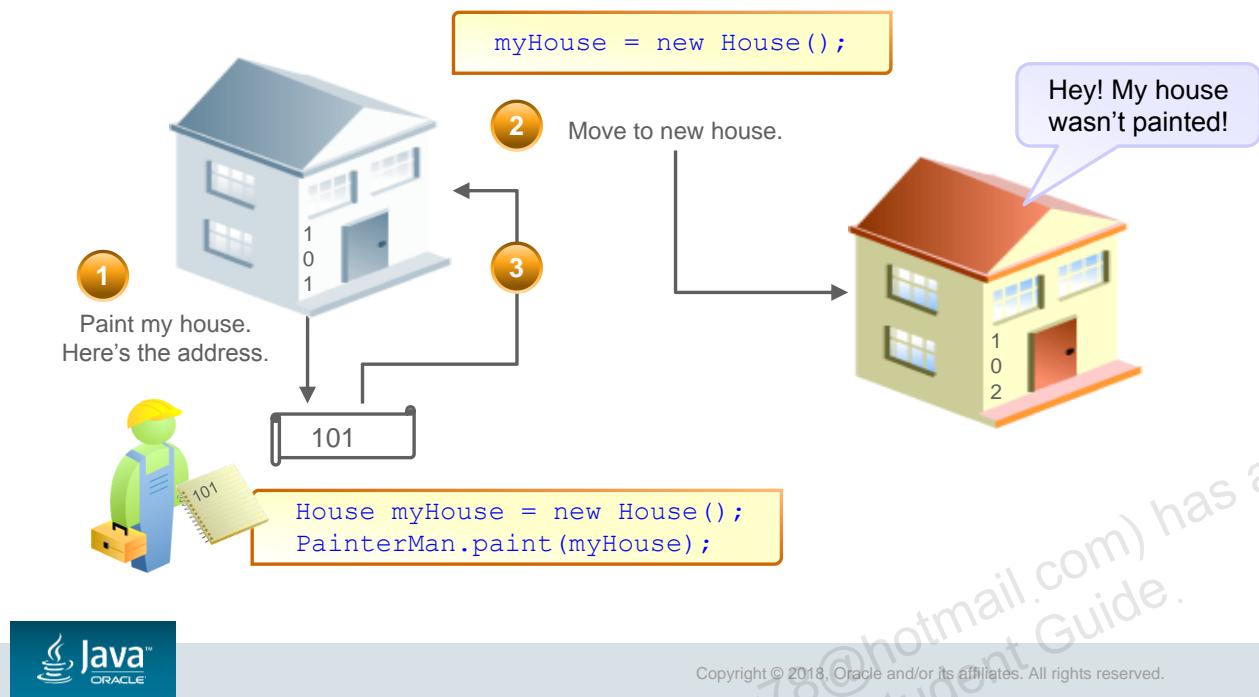
An object reference is not the same as the object. It simply provides a reference for access to that object. This is similar to the way a house address provides directions for finding a particular house.

In the graphic above, the house (call it `myHouse`) has an address (the `myHouse` reference) of 101. When the painter gets this address, he jots it down in his notebook (he makes a copy of it). This enables the house painter to find the house and paint it.

When you send an object reference as an argument to a method, you are sending a *copy* of the reference—not the object nor the actual reference.

The receiving method has the information it needs to act directly upon the object itself.

What If There Is a New Object?



Suppose that the owner of the house moves to another house before the job is finished. Will the painter be able to find the owner's new house in order to paint it? The object reference (`myHouse`) has changed to point to a new house, but the notation in the painter's notebook still refers to the old house. If the owner expects the new house to be painted, he or she will be disappointed.

A Shopping Cart Code Example

```
1 public class ShoppingCart {  
2     public static void main (String[] args) {  
3         Shirt myShirt = new Shirt();  
4         System.out.println("Shirt color: " + myShirt.colorCode);  
5         changeShirtColor(myShirt, 'B');  
6         System.out.println("Shirt color: " + myShirt.colorCode);  
7     }  
8     public static void changeShirtColor(Shirt theShirt, char color) {  
9         theShirt.colorCode = color;      }  
10 }
```

theShirt is a new reference of type Shirt.

Output:

```
Shirt color: U  
Shirt color: B
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

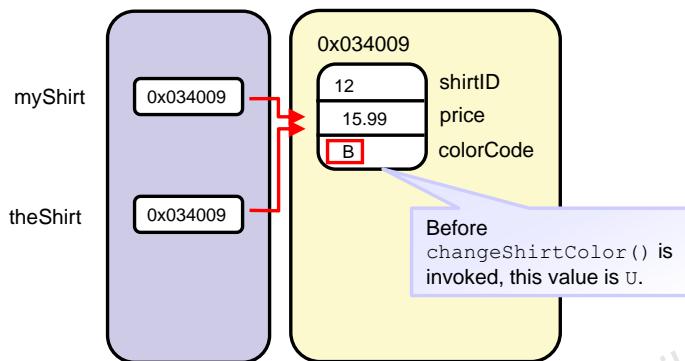
When a method is invoked, the values of the arguments are used to initialize the parameter variables before the body of the method is executed. This is true for both primitive types and reference types. (Objects are not passed to methods.)

In the example shown in the slide, the reference `myShirt` is passed by value into the `changeShirtColor` method. The reference, `theShirt` is assigned the value of the `myShirt` reference (the address). They now both point to the same object, so the change to the color made using `theShirt` is printed out by accessing `myShirt.color`.

Note: The call to the `changeShirtColor` method is made from the `main` method, which is static. Remember that a static method can only access other static methods. The `changeShirtColor` method is also static.

Passing by Value

```
Shirt myShirt = new Shirt();
changeShirtColor(myShirt, 'B');
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows how the value of the `myShirt` reference passed into the `changeShirtColor()` method is used to initialize a new `Shirt` reference (in this case, called `theShirt`). Remember that when a new `Shirt` is created, the `colorCode` is initialized to "U".

Reassigning the Reference

```
1 public class ShoppingCart {
2     public static void main (String[] args) {
3         Shirt myShirt = new Shirt();
4         System.out.println("Shirt color: " + myShirt.colorCode);
5         changeShirtColor(myShirt, 'B');
6         System.out.println("Shirt color: " + myShirt.colorCode);
7     }
8     public static void changeShirtColor(Shirt theShirt, char color) {
9         theShirt = new Shirt();
10        theShirt.colorCode = color;
11    }
12 }
```

Output:

```
Shirt color: U
Shirt color: U
```



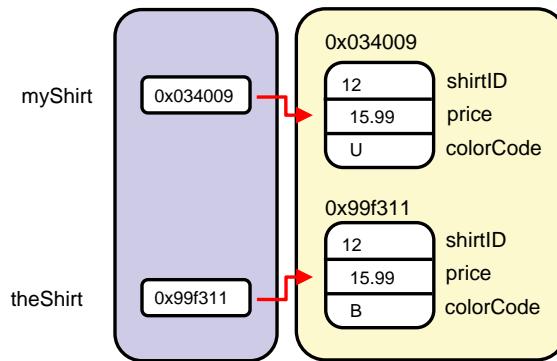
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Here is another example with a small change in the code of the `changeShirtColor()` method. In this example, the reference value passed into the method is assigned to a *new* shirt. The reference now points to a different `Shirt` object than the `myShirt` reference does. As before, the `Shirt.color` is changed to 'B'. The `println` method called on line 6 shows the color of the `myShirt` object still is 'U' (Unset). These references point to two different `Shirt` objects.

This illustrates that the reference `myShirt` is indeed passed by value. Changes made to a reference passed into a worker method (reassignment to a different object, for instance) do not affect the references in the calling method.

Passing by Value

```
Shirt myShirt = new Shirt();
changeShirtColor(myShirt, 'B');
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows the situation that results from the code in the previous slide.

When `myShirt` is passed into the `changeShirtColor()` method, a new reference variable, `theShirt`, is initialized with the value of `myShirt`. Initially, this reference points to the object that the `myShirt` reference points to. But after a new `Shirt` is assigned to `theShirt`, any changes made using `theShirt` affect only this new `Shirt` object.

Topics

- Using constructors and methods
- Method arguments and return values
- Using static methods and variables
- Understanding how arguments are passed to a method
- Overloading a method



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Method Overloading

Overloaded methods:

- Have the same name
- Have different signatures
 - The **number** of parameters
 - The **types** of parameters
 - The **order** of parameters
- May have different functionality or similar functionality
- Are widely used in the foundation classes

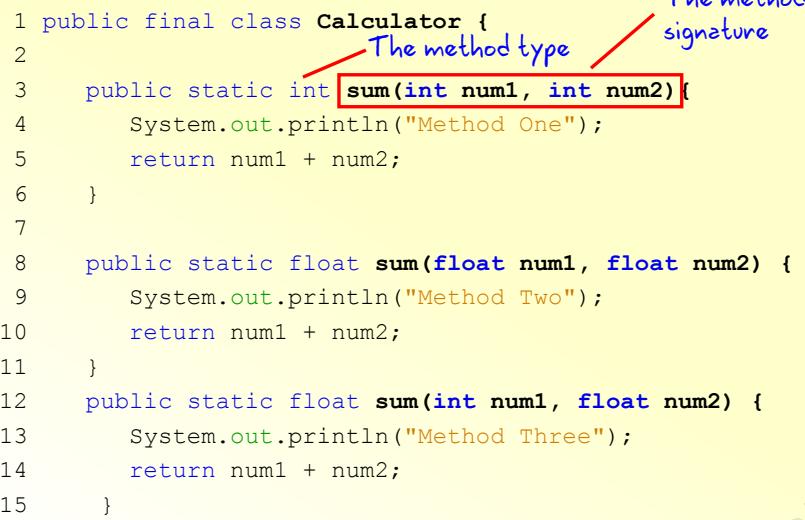


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the Java programming language, a class can contain several methods that have the same name but different arguments (so the method signature is different). This concept is called *method overloading*. Just as you can distinguish between two students named “Jim” in the same class by calling them “Jim in the green shirt” and “Jim with the beeper,” you can distinguish between two methods by their name and arguments.

Using Method Overloading

```
1 public final class Calculator {  
2       
3     public static int sum(int num1, int num2) {  
4         System.out.println("Method One");  
5         return num1 + num2;  
6     }  
7       
8     public static float sum(float num1, float num2) {  
9         System.out.println("Method Two");  
10        return num1 + num2;  
11    }  
12    public static float sum(int num1, float num2) {  
13        System.out.println("Method Three");  
14        return num1 + num2;  
15    }  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



The example in the slide shows three methods to add two numbers, such as two `int` types or two `float` types. With method overloading, you can create several methods with the same name and different signatures.

The first `sum` method accepts two `int` arguments and returns an `int` value. The second `sum` method accepts two `float` arguments and returns a `float` value. The third `sum` method accepts an `int` and a `float` as arguments and returns a `float`.

The callout shows the part of the method declaration that is called the *method signature*.

The method signature of a method is the unique combination of the method name and the number, types, and order of its parameters. The method signature does not include the return type. To invoke any of the `sum` methods, the compiler compares the method signature in your method invocation against the method signatures in a class.

Using Method Overloading

```
1 public class CalculatorTest {  
2  
3     public static void main(String[] args) {  
4  
5         int totalOne = Calculator.sum(2, 3);  
6         System.out.println("The total is " + totalOne);  
7  
8         float totalTwo = Calculator.sum(15.99F, 12.85F);  
9         System.out.println(totalTwo);  
10  
11        float totalThree = Calculator.sum(2, 12.85F);  
12        System.out.println(totalThree);  
13    }  
14 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The code example in the slide has a `main` method that invokes each of the previous `sum` methods of the `Calculator` class.

Method Overloading and the Java API

Method	Use
<code>void println()</code>	Terminates the current line by writing the line separator string
<code>void println(boolean x)</code>	Prints a boolean value and then terminates the line
<code>void println(char x)</code>	Prints a character and then terminates the line
<code>void println(char[] x)</code>	Prints an array of characters and then terminates the line



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Many methods in the Java API are overloaded, including the `System.out.println` method. The table in the slide shows four variations of the `println` method.

Exercise 8-2: Overload a `setItemFields` Method, Part 1

1. Open the project Practice_08-2 in NetBeans.

In the `Item` class:

2. Write a `setItemFields` method that takes three arguments and assigns them to the `desc`, `quantity`, and `price` fields. The method returns `void`.
3. Create an overloaded `setItemFields` method to take four arguments and return an `int`. The method assigns all four fields. A ' ' (a single space) is an invalid value for a `colorCode` argument.
 - If the `colorCode` argument is invalid, return `-1` without assigning the value.
 - If the `colorCode` is valid, assign the `colorCode` field and then assign the remaining fields by calling the three-argument method.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Exercise 8-2: Overload a `setItemFields` Method, Part 2

In the `ShoppingCart` class:

4. Call the 3-argument `setItemFields` method and then call `item1.displayItem()`.
5. Call the 4-argument `setItemFields` method. Check the return value.
 - If the return value < 0, print an invalid color code message.
 - Otherwise, call `displayItem()`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Quiz



Which method corresponds to the following method call?

```
myPerson.printValues(100, 147.7F, "lavender");
```

- a. public void printValues (int i, float f)
- b. public void printValues (i, float f, s)
- c. public void printValues (int i, float f, String s)
- d. public void printValues (float f, String s, int i)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: c

Summary

In this lesson, you should have learned how to:

- Add an argument to a method
- Instantiate a class and call a method
- Overload a method
- Work with static methods and variables
- Convert data values using `Integer`, `Double`, and `Boolean` object types



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Practices Overview

- 8-1: Using Methods
- 8-2: Creating Game Data Randomly
- 8-3: Creating Overloaded Methods



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Using Encapsulation



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Interactive Quizzes



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Before you start today's lessons, test your knowledge by answering some quiz questions that relate to yesterday's lessons. Open the Quiz files by clicking the quizzes.html shortcut from the desktop of your VM. In the welcome page, JavaSEProgrammingI.html, click the links for Lessons 6, 7, and 8.

Objectives

After completing this lesson, you should be able to:

- Use public and private access modifiers
- Restrict access to fields and methods using encapsulation
- Implement encapsulation in a class
- Overload a constructor by adding method parameters to a constructor



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Topics

- Access control
- Encapsulation
- Overloading constructors



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



What Is Access Control?

Access control allows you to:

- Hide fields and methods from other classes
- Determine how internal data gets changed
- Keep the implementation separate from the public interface
 - Public interface:

```
setPrice( Customer cust)
```

- Implementation:

```
public void setPrice(Customer cust) {  
    // set price discount relative to customer  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Access control allows you to hide internal data and functionality in a class. In this lesson, you distinguish between the public interface of a class and the actual implementation of that interface.

- The public interface is what you see when you look up a class in the JDK API documentation. You get just the information you need in order to use a class. That is, the signatures for public methods, and data types of any public fields.
- The implementation of a class is the code itself, and also any private methods or fields that are used by that class. These are the internal workings of a class and it is not necessary to expose them to another class.

Access Modifiers

- `public`: Accessible by anyone
- `private`: Accessible only within the class

```
1 public class Item {  
2     // Base price  
3     private double price = 15.50;  
4  
5     public void setPrice(Customer cust) {  
6         if (cust.hasLoyaltyDiscount()) {  
7             price = price*.85; }  
8     }  
9 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When a field is declared as public, any other class can access and potentially change the field's value. This is often problematic. It could be that the field represents sensitive data, such as a social security number, or that some type of logic or manipulation of the data may be required in order to safely modify the data. In the code example, the shirt price is declared in a private method. You would not want outside objects, such as a customer, to be able to freely manipulate the price of an item.

Access from Another Class

```
1 public class Item {  
2     private double price = 15.50;  
3  
4     public void setPrice(Customer cust) {  
5         if (cust.hasLoyaltyDiscount()) {  
6             price = price*.85; }  
7     }  
8 }
```

```
1 public class Order{  
2     public static void main(String args[]) {  
3         Customer cust = new Customer(int ID);  
4         Item item = new Item();  
5         item.price = 10.00;  
6         item.setPrice(cust);  
7     }  
8 }
```

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Won't compile
You don't need to know
how setPrice works in
order to use it.

Another Example

The data type of the field does not match the data type of the data used to set the field.

```
1 private int phone;
2 public void setPhoneNumber(String s_num) {
3     // parse out the dashes and parentheses from the
4     // String first
5     this.phone = Integer.parseInt(s_num);
6 }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

It may be that the data representing someone's phone number may be collected as a string, including spaces, dashes, and parentheses. If the phone number is represented internally as an `int`, then the setter method for the phone number will need to parse out spaces, dashes, and parentheses first, and then convert the `String` to an `int`. The `parseInt` method of `Integer` is covered in the "Using Encapsulation" lesson.

Using Access Control on Methods

```
1 public class Item {  
2     private int id;  
3     private String desc;  
4     private double price;  
5     private static int nextId = 1;  
6  
7     public Item() {  
8         setId(); Called from within a  
public method  
9         desc = "--description required--";  
10        price = 0.00;  
11    }  
12  
13    private void setId() { Private method  
14        id = Item.nextId++;  
15    }
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Here you see a private method that sets a new unique ID for an item. It is not necessary to expose this functionality to another class. The `setId` method is called from the public constructor method as part of its implementation.

Topics

- Access control
- **Encapsulation**
- Overloading constructors



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Encapsulation

- Encapsulation means hiding object fields. It uses access control to hide the fields.
 - Safe access is provided by getter and setter methods.
 - In setter methods, use code to ensure that values are valid.
- Encapsulation mandates programming to the interface:
 - A method can change the data type to match the field.
 - A class can be changed as long as interface remains same.
- Encapsulation encourages good object-oriented (OO) design.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Get and Set Methods

```
1 public class Shirt {
2     private int shirtID = 0;           // Default ID for the shirt
3     private String description = "-description required-"; // default
4     private char colorCode = 'U';    //R=Red, B=Blue, G=Green, U=Unset
5     private double price = 0.0;       // Default price for all items
6
7     public char getColorCode() {
8         return colorCode;
9     }
10    public void setColorCode(char newCode) {
11        colorCode = newCode;
12    }
13    // Additional get and set methods for shirtID, description,
14    // and price would follow
15
16 } // end of class
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If you make attributes private, how can another object access them? One object can access the private attributes of a second object if the second object provides public methods for each of the operations that are to be performed on the value of an attribute.

For example, it is recommended that all fields of a class should be private, and those that need to be accessed should have public methods for setting and getting their values.

This ensures that, at some future time, the actual field type itself could be changed, if that were advantageous. Or the getter or setter methods could be modified to control how the value could be changed, such as the value of the colorCode.

Why Use Setter and Getter Methods?

```
1 public class ShirtTest {
2     public static void main (String[] args) {
3         Shirt theShirt = new Shirt();
4         char colorCode;
5         // Set a valid colorCode
6         theShirt.setColorCode('R');
7         colorCode = theShirt.getColorCode();
8         System.out.println("Color Code: " + colorCode);
9         // Set an invalid color code
10        theShirt.setColorCode('Z'); Not a valid color code
11        colorCode = theShirt.getColorCode();
12        System.out.println("Color Code: " + colorCode);
13    }
14 ...
```

Output:

```
Color Code: R
Color Code: Z
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Though the code for the `Shirt` class is syntactically correct, the `setColorCode` method does not contain any logic to ensure that the correct values are set.

The code example in the slide successfully sets an invalid color code in the `Shirt` object.

However, because `ShirtTest` accesses a private field on `Shirt` using a setter method, `Shirt` can now be recoded without modifying any of the classes that depend on it.

In the code example above, starting with line 6, the `ShirtTest` class is setting and getting a valid `colorCode`. Starting with line 10, the `ShirtTest` class is setting an invalid `colorCode` and confirming that invalid setting.

Setter Method with Checking

```
15  public void setColorCode(char newCode) {  
16      if (newCode == 'R') {  
17          colorCode = newCode;  
18          return;  
19      }  
20      if (newCode == 'G') {  
21          colorCode = newCode;  
22          return;  
23      }  
24      if (newCode == 'B') {  
25          colorCode = newCode;  
26          return;  
27      }  
28      System.out.println("Invalid colorCode. Use R, G, or B");  
29  }  
30}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the slide is another version of the `Shirt` class. However, in this class, before setting the value, the setter method ensures that the value is valid. If it is not valid, the `colorCode` field remains unchanged and an error message is printed.

Note: Void type methods can have return statements. They just cannot return any values.

Using Setter and Getter Methods

```
1 public class ShirtTest {  
2     public static void main (String[] args) {  
3         Shirt theShirt = new Shirt();  
4         System.out.println("Color Code: " + theShirt.getColorCode());  
5  
6         // Try to set an invalid color code  
7         theShirt.setColorCode('Z')  
8         System.out.println("Color Code: " + theShirt.getColorCode());  
9     }  
}
```

Output:

Color Code: U ————— Before call to setColorCode() – shows default value
Invalid colorCode. Use R, G, or B ————— call to setColorCode prints error message
Color Code: U ————— colorCode not modified by invalid argument passed to setColorCode()



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Building on the previous slides, before the call to `setColorCode`, the default color value of U (unset) is printed. If you call `setColorCode` with an invalid code, the color code is not modified and the default value, U, is still the value. Additionally, you receive an error message that tells you to use the valid color codes, which are R, G, and B.

Exercise 9-1: Encapsulate a Class

In this exercise, you encapsulate the `Customer` class.

1. Open `Exercise_09-1` project in NetBeans.
2. Change access modifiers so that fields must be read or modified through public methods.
3. Allow the `name` field to be read and modified.
4. Allow the `ssn` field to be read but not modified (read only).



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In this exercise, you encapsulate the `Customer` class.

Topics

- Access control
- Encapsulation
- Overloading constructors



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



Initializing a Shirt Object

Explicitly:

```
1 public class ShirtTest {
2     public static void main (String[] args) {
3         Shirt theShirt = new Shirt();
4
5         // Set values for the Shirt
6         theShirt.setColorCode('R');
7         theShirt.setDescription("Outdoors shirt");
8         theShirt.price(39.99);
9     }
10 }
```

Using a constructor:

```
Shirt theShirt = new Shirt('R', "Outdoors shirt", 39.99);
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Assuming that you now have setters for all the private fields of `Shirt`, you could now instantiate and initialize a `Shirt` object by instantiating it and then setting the various fields through the setter methods.

However, Java provides a much more convenient way to instantiate and initialize an object by using a special method called a *constructor*.

Constructors

- Constructors are usually used to initialize fields in an object.
 - They can receive arguments.
 - When you create a constructor with arguments, it removes the default no-argument constructor.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

All classes have at least one constructor.

If the code does not include an explicit constructor, the Java compiler automatically supplies a no-argument constructor. This is called the default constructor.

Shirt Constructor with Arguments

```
1 public class Shirt {  
2     public int shirtID = 0;                      // Default ID for the shirt  
3     public String description = "-description required-"; // default  
4     private char colorCode = 'U';                  // R=Red, B=Blue, G=Green, U=Unset  
5     public double price = 0.0;                     // Default price all items  
6  
7     // This constructor takes three argument  
8     public Shirt(char colorCode, String desc, double price ) {  
9         setColorCode(colorCode);  
10        setDescription(desc);  
11        setPrice(price);  
12    }  
}
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The `Shirt` example shown in the slide has a constructor that accepts three values to initialize three of the object's fields. Because `setColorCode` ensures that an invalid code cannot be set, the constructor can just call this method.

Default Constructor and Constructor with Args

When you create a constructor with arguments, the default constructor is no longer created by the compiler.

```
// default constructor  
public Shirt()
```

This constructor is not in the source code. It only exists if no constructor is explicitly defined.

```
// Constructor with args
```

```
public Shirt (char color, String desc, double price)
```

```
6  /**  
7  *  
8  *cannot find symbol  
9  *symbol: constructor Shirt()  
10 *location: class Shirt  
11 --  
12 (Alt-Enter shows hints)  
13  
14 myShirt = new Shirt();  
15
```

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.



When you explicitly create an overloaded constructor, it replaces the default no-argument constructor.

You may be wondering why you have been able to instantiate a Shirt object with `Shirt myShirt = new Shirt()` even if you did not actually create that no-argument constructor. If there is no explicit constructor in a class, Java assumes that you want to be able to instantiate the class, and gives you an *implicit* default no-argument constructor. Otherwise, how could you instantiate the class?

The example above shows a new constructor that takes arguments. When you do this, Java removes the implicit default constructor. Therefore, if you try to use `Shirt myShirt = new Shirt()`, the compiler cannot find this constructor because it no longer exists.

Overloading Constructors



```
1 public class Shirt {  
2     ... //fields  
3  
4     // No-argument constructor  
5     public Shirt() {  
6         setColorCode('U');  
7     }  
8     // 1 argument constructor  
9     public Shirt(char colorCode) {  
10        setColorCode(colorCode);  
11    }  
12    // 2 argument constructor  
13    public Shirt(char colorCode, double price) {  
14        this(colorCode);  
15        setPrice(price);  
16    }  
}
```

If required, must be added explicitly

Calling the 1 argument constructor



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows three overloaded constructors:

- A default no-argument constructor
- A constructor with one parameter (a `char`)
- A constructor with two parameters (a `char` and a `double`)

This third constructor sets both the `colorCode` field and the `price` field. Notice, however, that the syntax where it sets the `colorCode` field is one that you have not seen yet. It would be possible to set `colorCode` with a simple call to `setColorCode()` just as the previous constructor does, but there is another option, as shown here.

You can chain the constructors by calling the second constructor in the first line of the third constructor using the following syntax:

```
this(argument);
```

The keyword `this` is a reference to the current object. In this case, it references the constructor method from this class whose signature matches.

This technique of chaining constructors is especially useful when one constructor has some (perhaps quite complex) code associated with setting fields. You would not want to duplicate this code in another constructor and so you would chain the constructors.

Quiz



What is the default constructor for the following class?

```
public class Penny {  
    String name = "lane";  
}  
  
a. public Penny(String name)  
b. public Penny()  
c. class()  
d. String()  
e. private Penny()
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Exercise 9-2: Create an Overloaded Constructor

1. Continue editing **Exercise_09-1** or open **Exercise_09-2** in NetBeans.

In the `Customer` class:

2. Add a custom constructor that initializes the fields.

In the `ShoppingCart` class:

3. Declare, instantiate, and initialize a new `Customer` object by calling the custom constructor.
4. Test it by printing the `Customer` object name (call the `getName` method).



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Use public and private access modifiers
- Restrict access to fields and methods using encapsulation
- Implement encapsulation in a class
- Overload a constructor by adding method parameters to a constructor



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Practices Overview

- 9-1: Encapsulating Fields
- 9-2: Creating Overloaded Constructors



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

