Java SE 7 Fundamentals

Student Guide - Volume II

D67234GC20

Edition 2.0

November 2011

D74823

ORACLE

Oracle Internal & Oracle Academy Use Only

Author

Jill Moritz

Kenneth Somerville

Cindy Church

Technical Contributors and Reviewers

Mike Williams

Tom McGinn

Matt Heimer

Joe Darcy

Brian Goetz

Alex Buckley

Adam Messenger

Steve Watts

Editors

Smita Kommini

Aju Kumar

Richard Wallis

Graphic Designers

Seema M. Bopaiah

Rajiv Chandrabhanu

Publishers

Giri Venugopal

Jayanthy Keshavamurthy

Copyright © 2011, 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.

Oracle Internal & Oracle Academy
Use Only

Contents

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

12 Using Advanced Object-Oriented Concepts

Oracle Internal & Oracle Academy
Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

8

Creating and Using Arrays

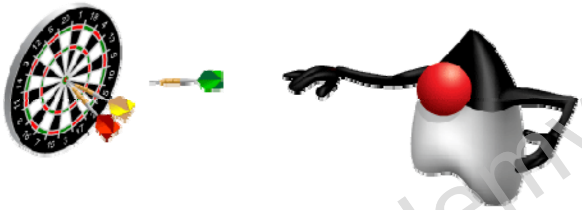
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Declare, instantiate, and initialize a one-dimensional array
- Declare, instantiate, and initialize a two-dimensional array
- Access a value within an array
- Describe how arrays are stored in memory
- Declare and initialize an ArrayList
- Use an `args` array



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

- Overview of arrays
- Declaring, instantiating, and initializing arrays
- Accessing command-line arguments
- Working with two-dimensional arrays
- Working with ArrayList

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Introduction to Arrays

- An array is a container object that holds a group of values of a single type.
- A value in the array can be a primitive or an object 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*.
- Each element is accessed by a numerical index.
- The index of the first element is 0 (zero).

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

One-Dimensional Arrays

Example:

```
int ageOne = 27;
int ageTwo = 12;
int ageThree = 82;
int ageFour = 70;
int ageFive = 54;
int ageSix = 6;
int ageSeven = 1;
int ageEight = 30;
int ageNine = 34;
int ageTen = 42;
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Consider a program where you store the ages of 10 people. You could create individual variables to hold each of the 10 values. You could do this using the code shown in the slide, but there are problems with this approach. What if you had to store 1,000 ages or 10,000 ages? As the number of values increases, your program becomes increasingly unmanageable. Or, what if you had to find the average age, or sort the ages into ascending order? You would have to refer to each variable individually in your code.

As you will see, arrays in Java (and related constructs such as lists) give you a much more convenient way to work with sets of data. In this lesson, you learn about arrays. In the lesson titled “Using Loop Constructs,” you learn how to use loops to programmatically work through all the values in an array.

Oracle Internal & Oracle Academy Use Only

Creating One-Dimensional Arrays

Array of `int` types

27	12	82	70	54	1	30	34

Array of Shirt types

Array of String types

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

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Java programming language allows you to group multiple values of the same type (lists) using arrays. Arrays are useful when you have related pieces of data (such as the ages of several people), but you do not want to create separate variables to hold each piece of data.

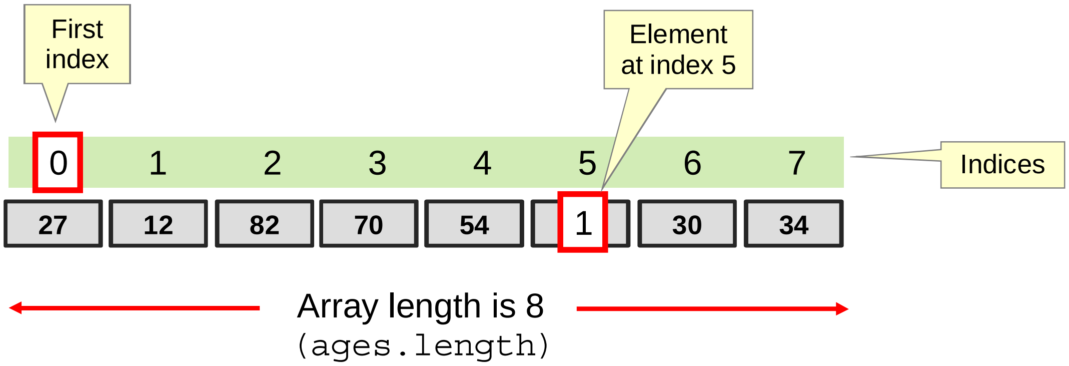
You can create an array of primitive types, such as `int`, or an array of references to object types, such as Shirt or String. Each part of the array is an element. If you declare an array of 100 `int` types, there are 100 elements. You can access each specific element within the array by using its location or index in the array.

The diagram in the slide shows examples of arrays for `int` types, Shirt types, and String types.

Oracle Internal & Oracle Academy Use Only

Array Indices and Length

Array `ages` of eight elements



Copyright © 2011, 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 in the slide, 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 the array in the diagram is called `ages`, you can use: `int agesLength = ages.length;`

This assigns a value of `8` to `int agesLength`.

Oracle Internal & Oracle Academy Use Only

Topics

- Overview of arrays
- Declaring, instantiating, and initializing arrays
- Accessing command-line arguments
- Working with two-dimensional arrays
- Working with ArrayList

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Declaring a One-Dimensional Array

- Syntax:

```
    type [] array_identifier;
```

- Declare arrays of types `char` and `int`:

```
    char [] status;
    int  [] ages;
```

- Declare arrays of object references of types Shirt and String:

```
    Shirt [] shirts;
    String [] names;
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Arrays are handled by an implicit Array object (which is not available in the Java API). Just like with any object, you must declare an object reference to the array, instantiate an Array object, and then initialize the Array object before you can use it.

The syntax used to declare a one-dimensional array is:

```
        type [] array_identifier;
```

where:

- `type` represents the primitive data type or object type for the values stored in the array
- `[]` informs the compiler that you are declaring an array
- `array_identifier` is the name that you are assigning to refer to the array

When you declare an array, the compiler and the Java Virtual Machine (JVM) have no idea how large the arrays will be because you have declared a reference variable that does not currently point to any objects.

Oracle Internal & Oracle Academy Use Only

Instantiating a One-Dimensional Array

- Syntax:

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

- Examples:

```
status = new char [20];
ages = new int [5];

names = new String [7];
shirts = new Shirt [3];
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Before you can initialize an array, you must instantiate an Array object large enough to hold all of the values in the array. Instantiate an array by defining the number of elements in the array.

The syntax used to instantiate an Array object is:

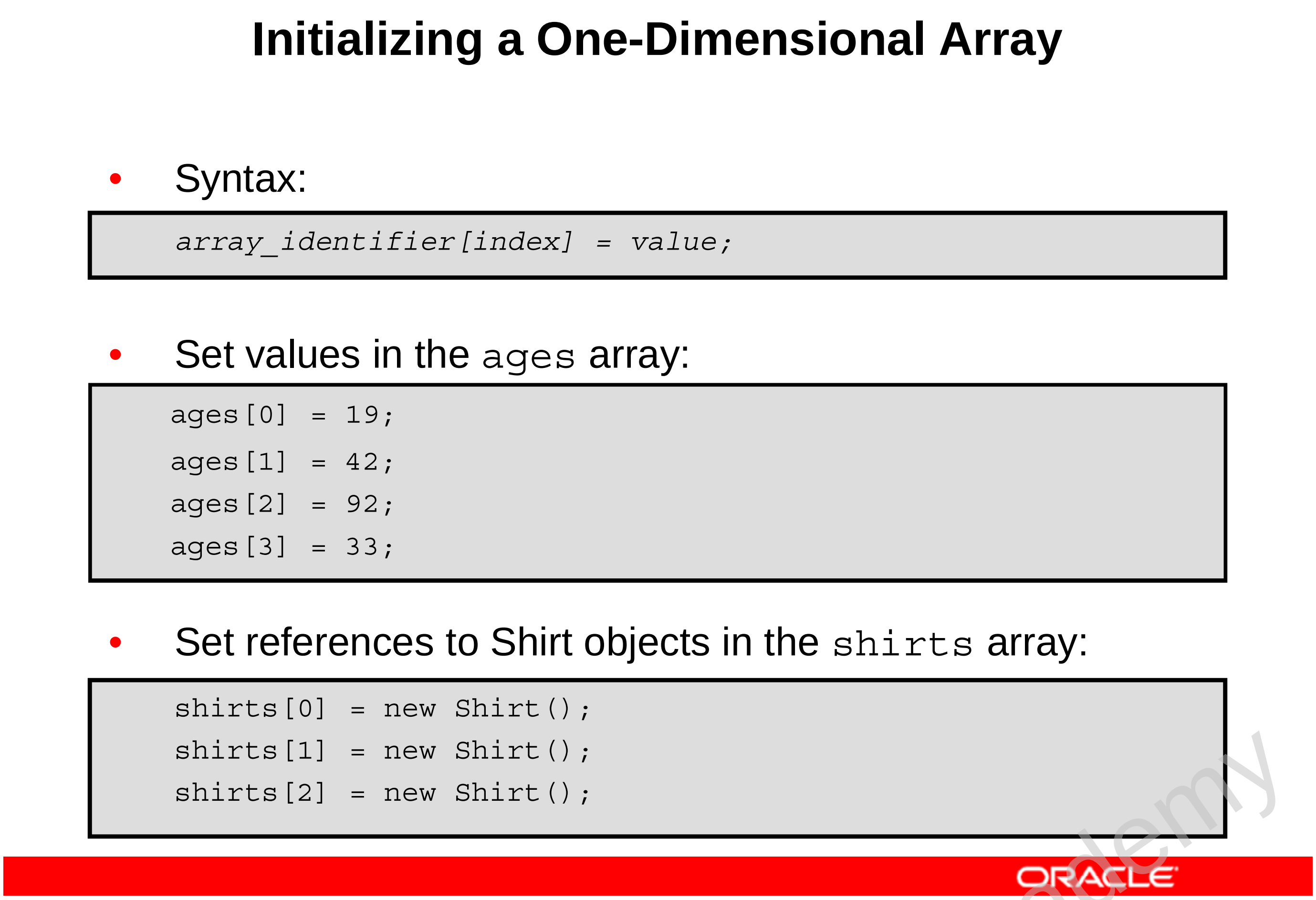
```
array_identifier = new type [length];
```

where:

- `array_identifier` is the name you are assigning to reference the array
- `type` represents the primitive data type or object type for the values stored in the array
- `length` represents the size (in number of elements) of the array

When you instantiate an Array object, every primitive element is initialized to the zero value for the type you specified. In the case of the `char` array called `status`, each value is initialized to `\u0000` (the null character of the Unicode character set). For the `int` array called `ages`, the initial value is the integer value `0`. For the `names` and `shirt` arrays, the object references are initialized to null.

Oracle Internal & Oracle Academy Use Only



Initializing a One-Dimensional Array

- Syntax:

```
array_identifier[index] = value;
```

- Set values in the `ages` array:

```
ages[0] = 19;
ages[1] = 42;
ages[2] = 92;
ages[3] = 33;
```

- Set references to Shirt objects in the `shirts` array:

```
shirts[0] = new Shirt();
shirts[1] = new Shirt();
shirts[2] = new Shirt();
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You can fill the contents of an array after you have created it. The syntax for setting the values in an array is:

```
array_identifier[index] = value;
```

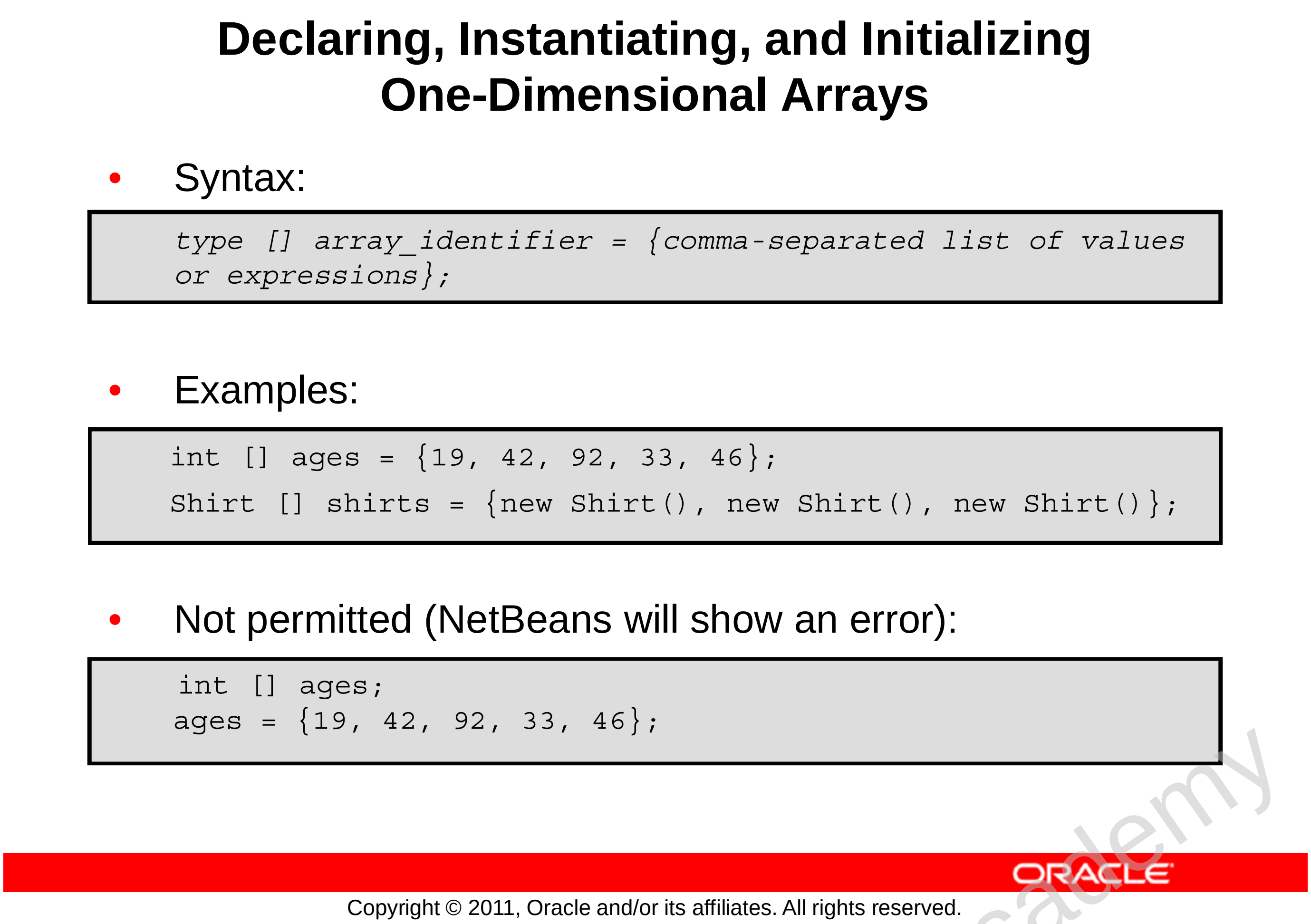
where:

- `array_identifier` is the name you are assigning to the array
- `index` represents the location in the array where the value will be placed

Use the `new` keyword to create the Shirt objects and to place the references to the Shirt objects into each position in the array.

Note: The index to the first element of an array is 0 and the index to the last element of the array is the length of the array minus 1. For example, the last element of a six-element array is index 5.

Oracle Internal & Oracle Academy Use Only



Declaring, Instantiating, and Initializing One-Dimensional Arrays

- Syntax:

```
type [] array_identifier = {comma-separated list of values
or expressions};
```

- Examples:

```
int [] ages = {19, 42, 92, 33, 46};
Shirt [] shirts = {new Shirt(), new Shirt(), new Shirt()};
```

- Not permitted (NetBeans will show an error):

```
 int [] ages;
ages = {19, 42, 92, 33, 46};
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

If you know the values you want in your array at the time that you declare it, you can declare, instantiate, and set the values for an Array object in the same line of code. The syntax for this is:

```
type [] array_identifier =
                              {comma-separated_list_of_values_or_expressions};
```

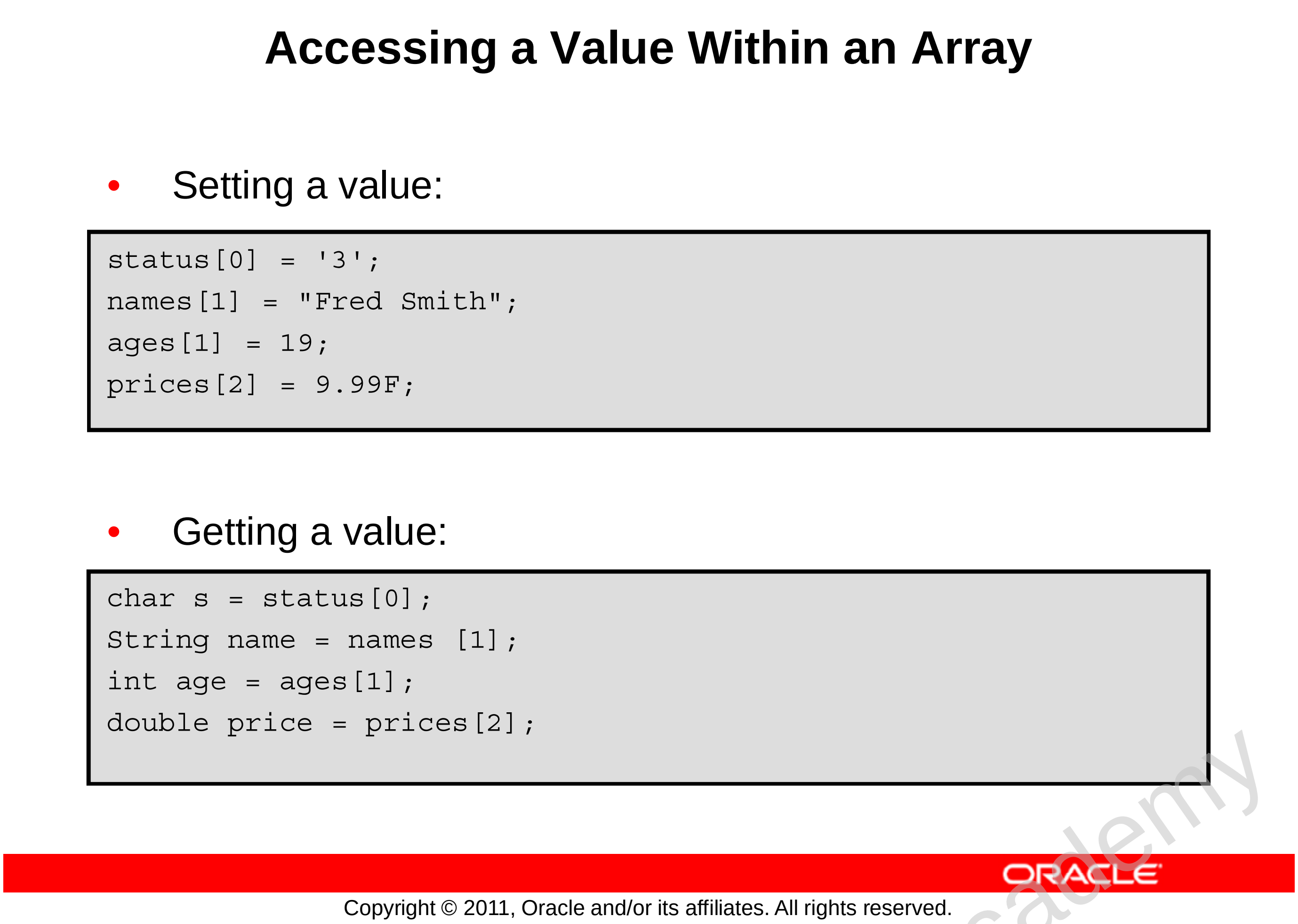
where:

- `type` represents the primitive data type or object type for the values to be stored
- `[]` informs the compiler that you are declaring an array
- `array_identifier` is the name you are assigning to the array
- `{comma-separated_list_of_values_or_expressions}` represents a list of values that you want to store in the array

The examples in the slide show statements that combine the declaration, instantiation, and initialization. Notice how the `new` keyword is used to instantiate the Shirt object so that a reference to that object can be placed in the array.

The final example in the slide returns an error. You cannot declare and initialize an array in separate lines by using the comma-separated list technique.

Oracle Internal & Oracle Academy Use Only



Accessing a Value Within an Array

- Setting a value:

```
status[0] = '3';
names[1] = "Fred Smith";
ages[1] = 19;
prices[2] = 9.99F;
```

- Getting a value:

```
char s = status[0];
String name = names [1];
int age = ages[1];
double price = prices[2];
```

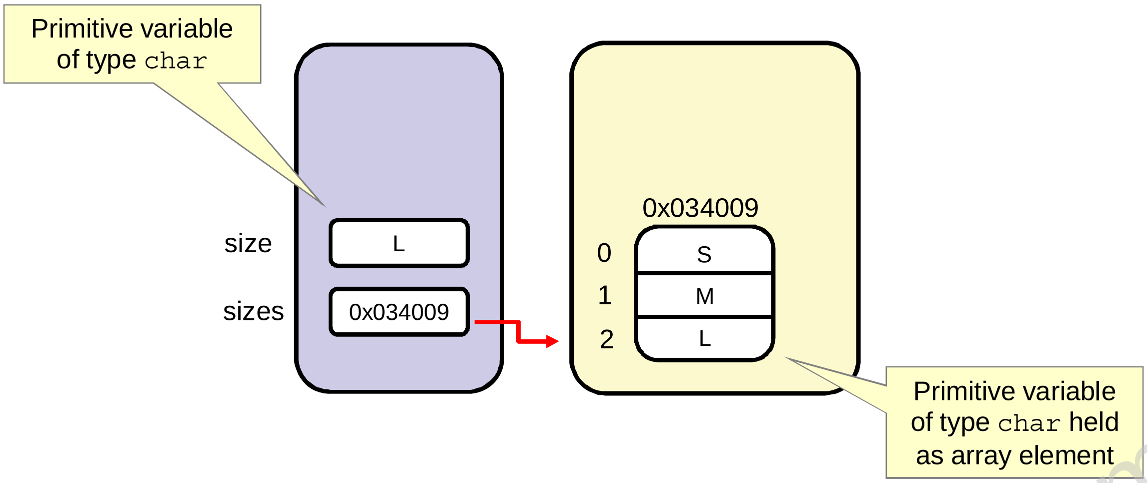
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Each element of an array is accessed using its index. To access a value from the array, state the array name and the index number for the element (in square brackets `[]`) on the right side of an assignment operator.

Oracle Internal & Oracle Academy Use Only

Storing Arrays in Memory

```
char size = 'L'
char[] sizes = {'S', 'M', 'L' };
```



Primitive variable of type `char`

Primitive variable of type `char` held as array element

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

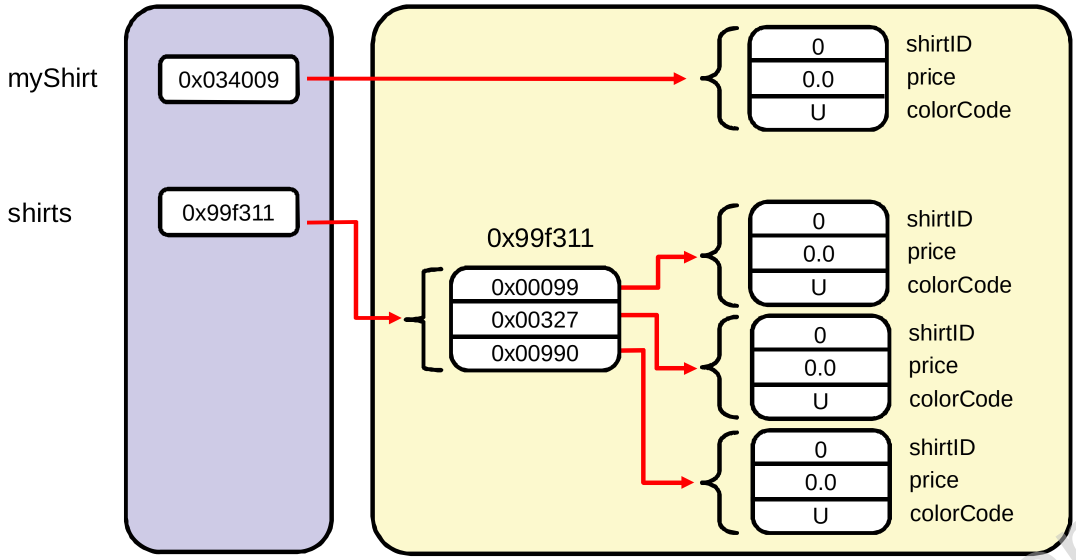
Arrays are objects referred to by an object reference variable. The diagram in the slide illustrates how a primitive array is stored in memory in comparison to how a primitive data type is stored in memory.

The value of the size variable (a `char` primitive) is `L`. The value of `sizes[]` is `0x334009`, and it points to an object of type `array` (of `char` types) with three values. The value of `sizes[0]` is `char S`, the value of `sizes[1]` is `char M`, and the value of `sizes[2]` is `char L`.

Oracle Internal & Oracle Academy Use Only

Storing Arrays of References in Memory

```
Shirt myShirt = new Shirt();
Shirt[] shirts = { new Shirt(), new Shirt(), new Shirt() };
```



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide illustrates how an object reference array is stored in memory. The value of the `myShirt` object reference is x034009, which is an address to an object of type Shirt with the values 0, 0.0, and U. The value of the `shirts[ ]` object reference is x99f311, which is an address to an object of type `array` (of Shirt object references) containing three object references:

- The value of the `shirts[0]` index is 0x00099, which is an object reference pointing to an object of type Shirt.
- The value of the `shirts[1]` index is 0x00327, which is an object reference pointing to another object of type Shirt.
- The value of the `shirts[2]` index is 0x00990, which is an object reference pointing to another object of type Shirt.

Oracle Internal & Oracle Academy Use Only

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 © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Quiz

Given the following array declaration, determine which of the three statements below it are true.

```
int [ ] autoMobile = new int [13];
```

a. `autoMobile[0]` is the reference to the first element in the array.

b. `autoMobile[13]` is the reference to the last element in the array.

c. There are 13 integers in the `autoMobile` array.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

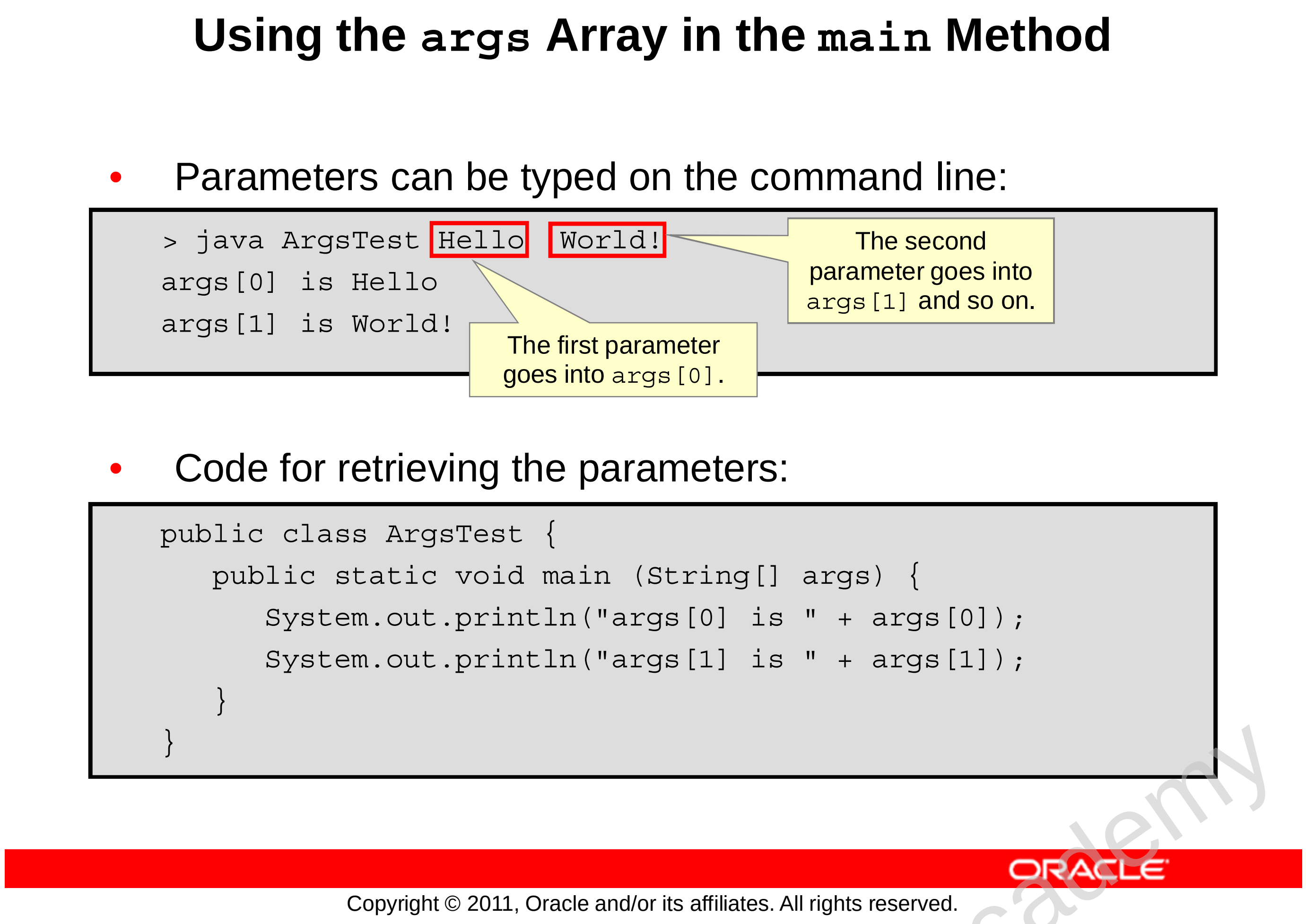
Oracle Internal & Oracle Academy Use Only

Topics

- Overview of arrays
- Declaring, instantiating, and initializing arrays
- Accessing command-line arguments
- Working with two-dimensional arrays
- Working with ArrayList

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Using the `args` Array in the `main` Method

- Parameters can be typed on the command line:

```
> java ArgsTest Hello World!
args[0] is Hello
args[1] is World!
```

The second parameter goes into `args[1]` and so on.

The first parameter goes into `args[0]`.

- Code for retrieving the parameters:

```
public class ArgsTest {
   public static void main (String[] args) {
      System.out.println("args[0] is " + args[0]);
      System.out.println("args[1] is " + args[1]);
   }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

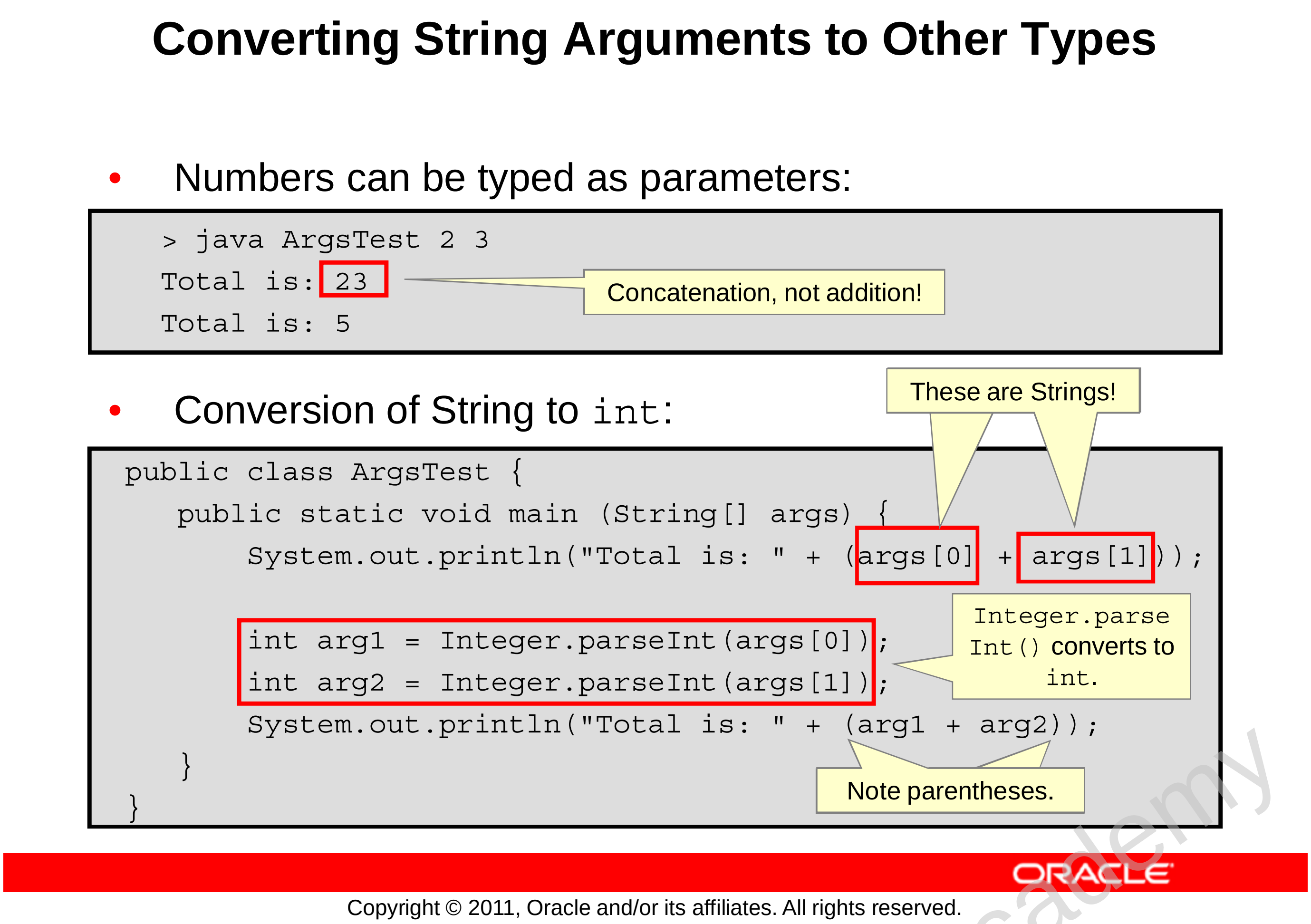
When you pass strings to your program on the command line, the strings are put in the `args` array. To use these strings, you must extract them from the `args` array and, optionally, convert them to their proper type (because the `args` array is of type String).

The `ArgsTest` class shown in the slide extracts two String arguments passed on the command line and displays them.

To add parameters on the command line, you must leave one or more spaces after the class name (in this case, `ArgsTest`) and one or more spaces between each parameter added.

NetBeans does not allow you a way to run a Java class from the command line, but you can set command-line arguments as a property of the project your code is in. You use this technique in the practice for this lesson.

Oracle Internal & Oracle Academy Use Only



Converting String Arguments to Other Types

- Numbers can be typed as parameters:

```
> java ArgsTest 2 3
Total is: 23
Total is: 5
```

Concatenation, not addition!

- Conversion of String to `int`:

```
public class ArgsTest {
   public static void main (String[] args) {
       System.out.println("Total is: " + (args[0] + args[1]));

       int arg1 = Integer.parseInt(args[0]);
       int arg2 = Integer.parseInt(args[1]);
       System.out.println("Total is: " + (arg1 + arg2));
   }
}
```

These are Strings!

`Integer.parseInt()` converts to `int`.

Note parentheses.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The `main` method treats everything you type as a literal string. If you want to use the string representation of a number in an expression, you must convert the string to its numerical equivalent. Every data type has an associated class containing static utility methods for converting strings to that data type (`Integer` class for `int`, `Byte` class for `byte`, `Long` class for `long`, and so on). For example, to convert the first argument passed to the `main` method to an `int` type, use `Integer.parseInt(args[0])`.

Note that the parentheses around `arg1 + arg2` are required so that the `+` sign indicates addition rather than concatenation.

Oracle Internal & Oracle Academy Use Only

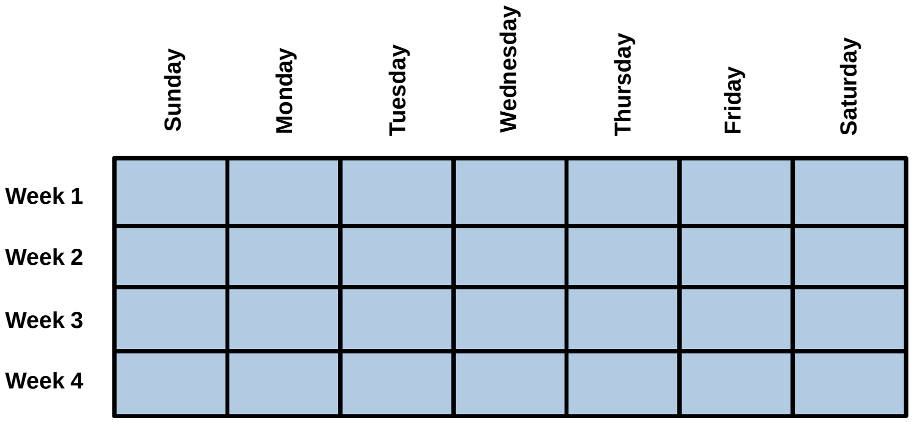
Topics

- Overview of arrays
- Declaring, instantiating, and initializing arrays
- Accessing command-line arguments
- Working with two-dimensional arrays
- Working with ArrayList

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Describing Two-Dimensional Arrays



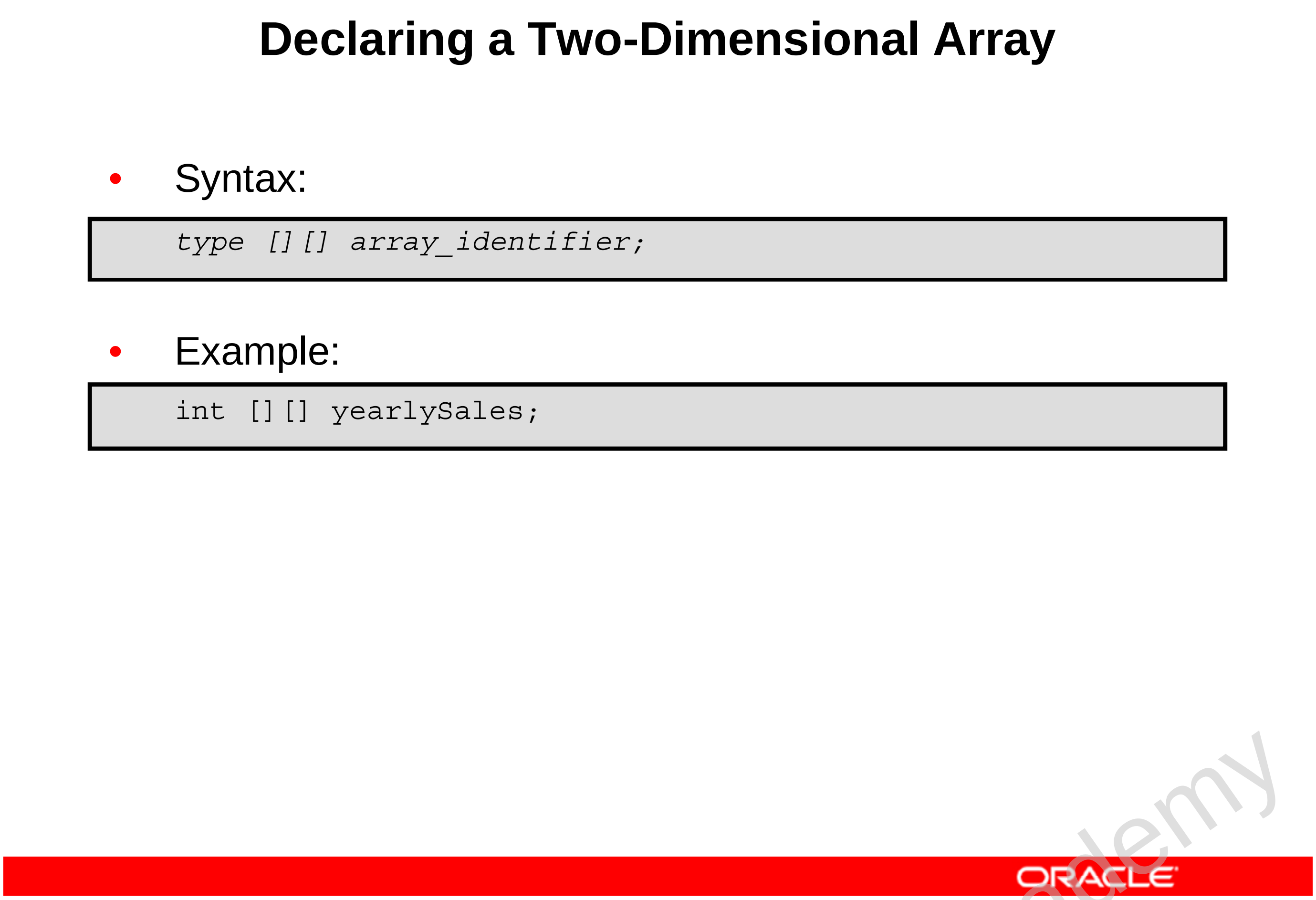
	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Week 1							
Week 2							
Week 3							
Week 4							

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You can also store matrices of data by using multidimensional arrays. Multidimensional arrays have two or more dimensions. A two-dimensional (2D) array is an array of arrays, a 3D array is an array of 2D arrays, a 4D array is an array of 3D arrays, and so on. A two-dimensional array is similar to a spreadsheet with multiple columns (each column represents one array or list of items) and multiple rows.

The diagram in the slide shows a two-dimensional array. Note that the descriptive names Week 1, Week 2, Monday, Tuesday, and so on would not be used to access the elements of the array. Instead, Week 1 would be index 0 and Week 4 would be index 3 along that dimension, while Sunday would be index 0 and Saturday would be index 6 along the other dimension.

Oracle Internal & Oracle Academy Use Only



Declaring a Two-Dimensional Array

- Syntax:

```
type [][] array_identifier;
```

- Example:

```
int [][] yearlySales;
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Two-dimensional arrays require an additional set of square brackets. The process of creating and using two-dimensional arrays is otherwise the same as with one-dimensional arrays. The syntax for declaring a two-dimensional array is:

```
type [][] array_identifier;
```

where:

- `type` represents the primitive data type or object type for the values stored in the array
- `[][]` inform the compiler that you are declaring a two-dimensional array
- `array_identifier` is the name you have assigned to the array during declaration

The example shown declares a two-dimensional array (an array of arrays) called `yearlySales`.

Oracle Internal & Oracle Academy Use Only

Instantiating a Two-Dimensional Array

- Syntax:

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

- Example:

```
// Instantiates a 2D array: 5 arrays of 4 elements each
yearlySales = new int[5][4];
```

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1				
Year 2				
Year 3				
Year 4				
Year 5				

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The syntax for instantiating a two-dimensional array is:

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

where:

- `array_identifier` is the name you have assigned the array during declaration
- `number_of_arrays` is the number of arrays within the array
- `length` is the length of each array within the array

The example shown in the slide instantiates an array of arrays for quarterly sales amounts over five years. The `yearlySales` array contains five elements of the type `int` array (five subarrays). Each subarray is four elements in size and tracks the sales for one year over four quarters.

Oracle Internal & Oracle Academy Use Only

Initializing a Two-Dimensional Array

Example:

```
yearlySales[0][0] = 1000;
yearlySales[0][1] = 1500;
yearlySales[0][2] = 1800;
yearlySales[1][0] = 1000;
yearlySales[3][3] = 2000;
```

	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Year 1	1000	1500	1800	
Year 2	1000			
Year 3				
Year 4				2000
Year 5				

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

When setting (or getting) values in a two-dimensional array, indicate the index number in the array by using a number to represent the row, followed by a number to represent the column. The example in the slide shows five assignments of values to elements of the `yearlySales` array.

Oracle Internal & Oracle Academy Use Only

Topics

- Overview of arrays
- Declaring, instantiating, and initializing arrays
- Accessing command-line arguments
- Working with two-dimensional arrays
- Working with ArrayList

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

ArrayList Class

Arrays are not the only way to store lists of related data:

- ArrayList is one of a number of list classes.
- It has a set of useful methods for managing its elements:
  - `add()`, `get()`, `remove()`, `indexOf()`, and many others
- You do not need to specify a size when you instantiate an ArrayList:
  - As you add more elements, the ArrayList grows as necessary.
  - You can specify an initial capacity, but it is not mandatory to do so.
- An ArrayList can store only objects, not primitives.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

For lists that are very dynamic, it may be easier to work with a specialized List type object. This can free you from having to write code to:

- Keep track of the index of the last piece of data added
- Keep track of how full the array is and determine if it needs to be resized
- Increase the size of the array by creating a new one and copying the elements from the current one into it

Oracle Internal & Oracle Academy Use Only

Class Names and the Import Statement

- ArrayList is in the package `java.util`.
- To refer to the ArrayList in your code, you can fully qualify

  `java.util.ArrayList myList;`

  or you can add the import statement at the top of the class.

```
import java.util.ArrayList;
public class ArrayListExample {
   public static void main (String[] args) {
       ArrayList myList;
   }
}
```

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Classes in the Java programming language are grouped into packages depending on their functionality. For example, all classes related to the core Java programming language are in the `java.lang` package, which contains classes that are fundamental to the Java programming language, such as String, Math, and Integer. Classes in the `java.lang` package can be referred to in code by just their class names. They do not require full qualification or the use of an import statement.

All classes in other packages (for example, ArrayList) require that you fully qualify them in the code, or that you use an import statement so that they can be referred to directly in the code.

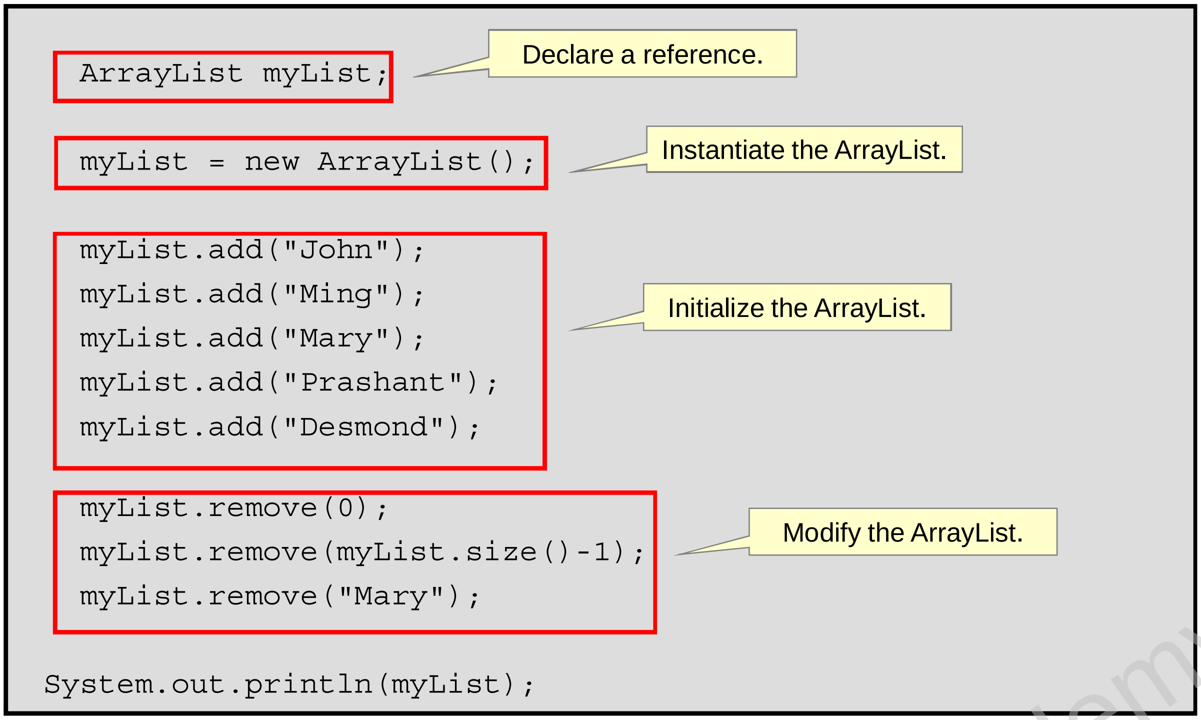
The import statement can be:

- For just the class in question

  `java.util.ArrayList;`
- For all classes in the package

  `java.util.*;`

Working with an ArrayList



```
ArrayList myList;

myList = new ArrayList();

myList.add("John");
myList.add("Ming");
myList.add("Mary");
myList.add("Prashant");
myList.add("Desmond");

myList.remove(0);
myList.remove(myList.size()-1);
myList.remove("Mary");

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

Declare a reference.

Instantiate the ArrayList.

Initialize the ArrayList.

Modify the ArrayList.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Declaring an ArrayList is exactly the same as declaring any other reference type. Likewise, instantiating an ArrayList is the same as instantiating any other object. (You can check the documentation for other possibilities for instantiating.)

There are a number of methods to add data to the ArrayList. The example in the slide uses the simplest, `add()`, to add a string. Each call to `add` adds a new element to the end of the ArrayList.

Finally, a big advantage of ArrayList over an array is that there are many methods available for manipulating the data. The example here shows just one method, but it is very powerful.

- `remove(0)`: This removes the first element (in this case, `"John"`).
- `remove(myList.size() - 1)`: This removes the last element because `myList.size()` gives the number of elements of the array, so the last one is the size minus 1 (this removes `"Desmond"`).
- `remove("Mary")`: This removes a specific element. In this case, you have the convenience of referring not to where the element is in the ArrayList, but rather to what it is.

You can pass an ArrayList to `System.out.println()` and the resulting output will be:

```
[Ming, Prashant]
```

Oracle Internal & Oracle Academy Use Only

Quiz

A two-dimensional array is similar to a _______.

a. Shopping list
b. List of chores
c. Matrix
d. Bar chart containing the dimensions for several boxes

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned the following:

- An array in Java is a data type that is composed of a set of other data types:
  - The data types can be objects or primitives.
  - Each data value is an element of the array.
- Arrays are created with a specific size (number of elements).
- Each element in an array can be accessed using its index:
  - The first index is 0 (zero).
- The data type of an array can be another array:
  - This creates a two-dimensional array.
- Another option is to use a specialized List class, such as ArrayList.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 8-1 Overview: Creating a Class with a One-Dimensional Array of Primitive Types

In this practice, you create an array containing the number of vacation days that an employee at the Duke's Choice company receives.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 8-2 Overview: Creating and Working with an `ArrayList`

In this practice, you experiment with populating and manipulating ArrayLists. During the practice, you:

- Create two classes, `NamesList` and `NamesListTest`
- Add a method to the `NamesList` class to populate the list and display its contents
- Add a method to manipulate the values in the list

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 8-3 Overview: Using Runtime Arguments and Parsing the `args` Array

In this practice, you write a guessing game that accepts an argument and displays an associated message. During the practice, you:

- Create a class that accepts a runtime argument
- Generate a random number
- Compare the random number with an argument value

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

9

Using Loop Constructs

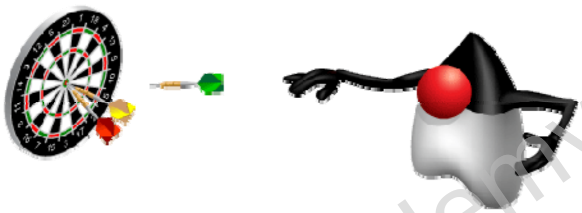
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Create a `while` loop
- Nest a `while` loop
- Develop and nest a `for` loop
- Code and nest a `do/while` loop
- Use an ArrayList in a `for` loop
- Compare loop constructs



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Loops

Loops are frequently used in programs to repeat blocks of statements until an expression is false.

There are three main types of loops:

- `while` loop: Repeats while an expression is true
- `do/while` loop: Executes once and then continues to repeat while true
- `for` loop: Repeats a set number of times

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Repeating Behavior

```
while (!areWeThereYet) {

   read book;
   argue with sibling;
   ask, "Are we there yet?";

}

Woohoo!;
Get out of car;
```

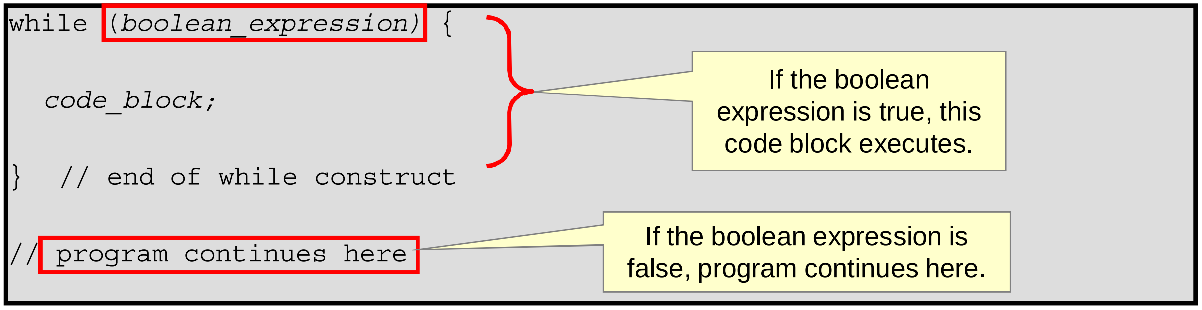
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In computer programming, it is common to need to repeat a number of statements. Typically, the code continues to repeat the statements until something changes. Then the code breaks out of the loop and continues with the next statement.

Oracle Internal & Oracle Academy Use Only

Creating `while` Loops

Syntax:



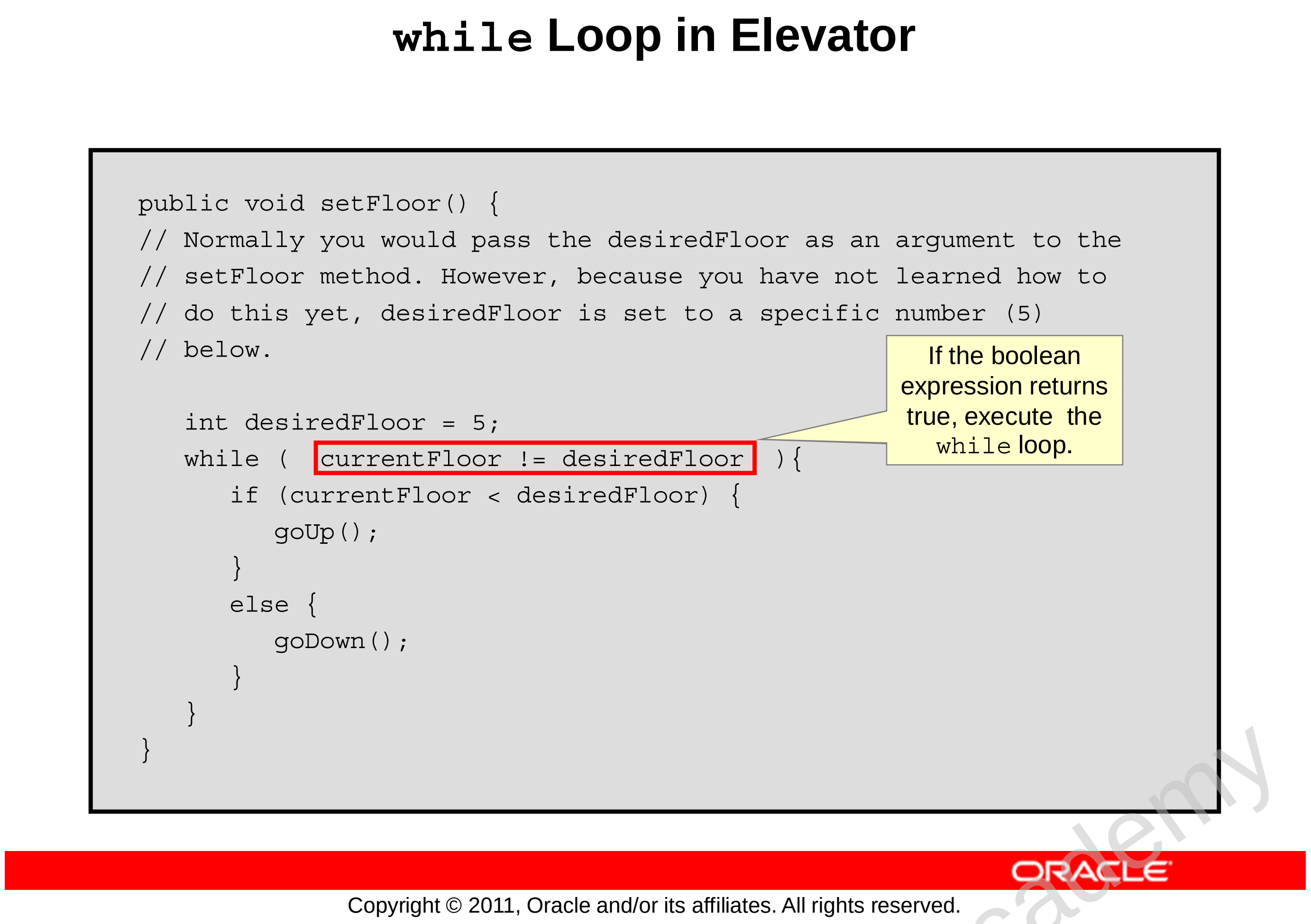
```
while (boolean_expression) {

   code_block;

}  // end of while construct

// program continues here
```

If the boolean expression is true, this code block executes.

If the boolean expression is false, program continues here.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



while Loop in Elevator

```
public void setFloor() {
// Normally you would pass the desiredFloor as an argument to the
// setFloor method. However, because you have not learned how to
// do this yet, desiredFloor is set to a specific number (5)
// below.

   int desiredFloor = 5;
   while ( currentFloor != desiredFloor ){
      if (currentFloor < desiredFloor) {
         goUp();
      }
      else {
         goDown();
      }
   }
}
```

If the boolean expression returns true, execute the while loop.

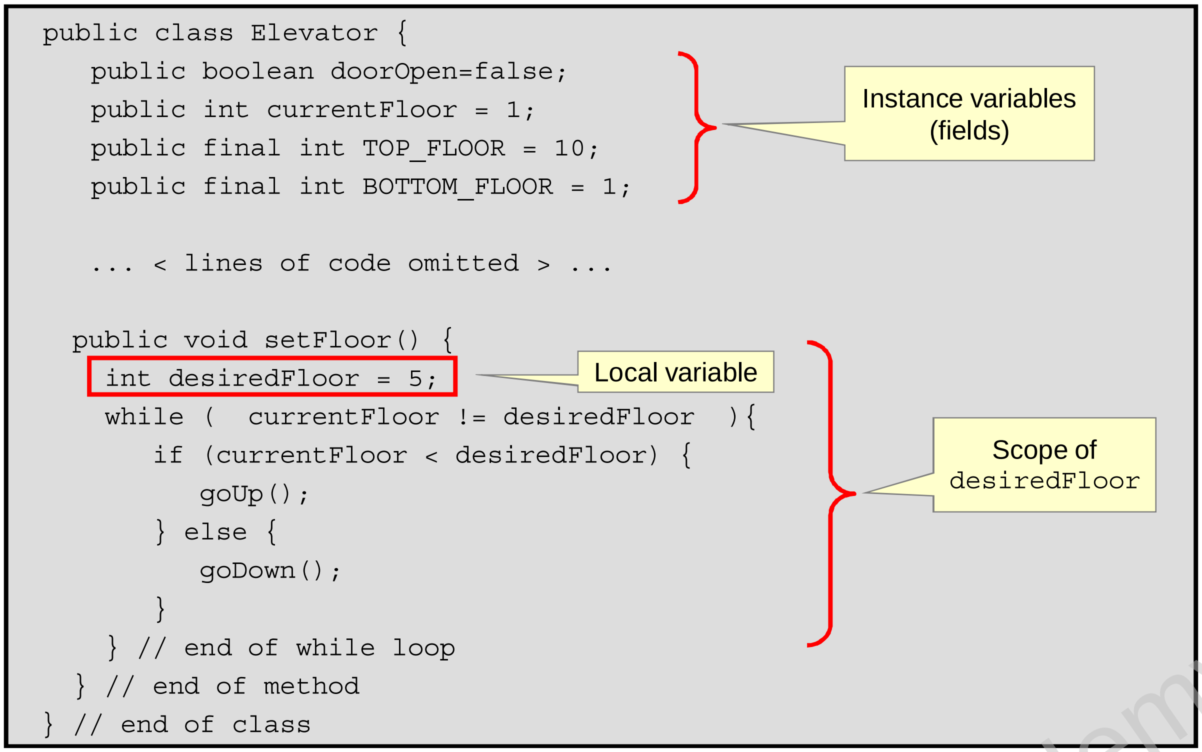
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows a very simple `while` loop in the `Elevator` class. Remember that this particular elevator accepts commands for going up or down only one floor at a time. So to move a number of floors, the `goUp()` or `goDown()` method needs to be called a number of times.

Notice how the boolean expression is written. The expression returns true if `currentFloor` is not equal to `desiredFloor`. So, when these two variables are equal, this expression returns false (because the elevator is now at the desired floor), and the `while` loop is not executed.

Oracle Internal & Oracle Academy Use Only

Types of Variables



```
public class Elevator {
   public boolean doorOpen=false;
   public int currentFloor = 1;
   public final int TOP_FLOOR = 10;
   public final int BOTTOM_FLOOR = 1;

   ... < lines of code omitted > ...

  public void setFloor() {
    int desiredFloor = 5;
    while (  currentFloor != desiredFloor  ){
       if (currentFloor < desiredFloor) {
          goUp();
       } else {
          goDown();
       }
    } // end of while loop
  } // end of method
} // end of class
```

Instance variables (fields)

Local variable

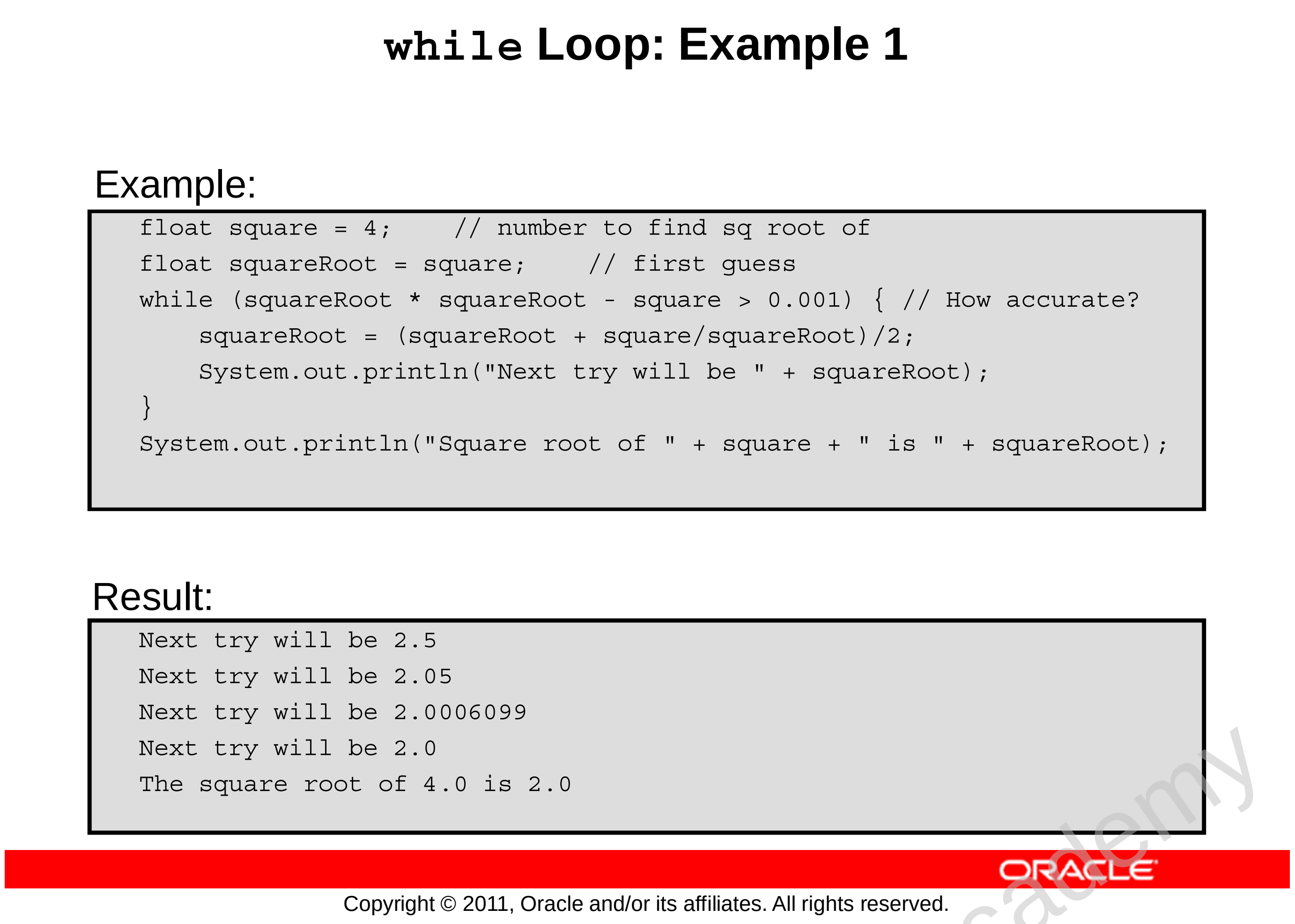
Scope of `desiredFloor`

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

This `setFloor` method uses two different types of variables. The `currentFloor` variable is an instance variable, usually called a *field*. It is a member of the `Elevator` class. In an earlier lesson, you saw how fields of an object could be accessed by using the dot notation. Fields are declared outside of method code, usually just after the class declaration.

The `desiredFloor` variable is a local variable, declared within the `setFloor` method and accessible only within that method. Another way to say this is that its scope is the `setFloor` method. As you will see later, local variables can also be declared within loops or `if` statements. Regardless of whether a local variable is declared within a method, a loop, or an `if` statement, its scope is always the block within which it is declared.

Oracle Internal & Oracle Academy Use Only



`while` Loop: Example 1

Example:

```
float square = 4;     // number to find sq root of
float squareRoot = square;    // first guess
while (squareRoot * squareRoot - square > 0.001) { // How accurate?
    squareRoot = (squareRoot + square/squareRoot)/2;
    System.out.println("Next try will be " + squareRoot);
}
System.out.println("Square root of " + square + " is " + squareRoot);
```

Result:

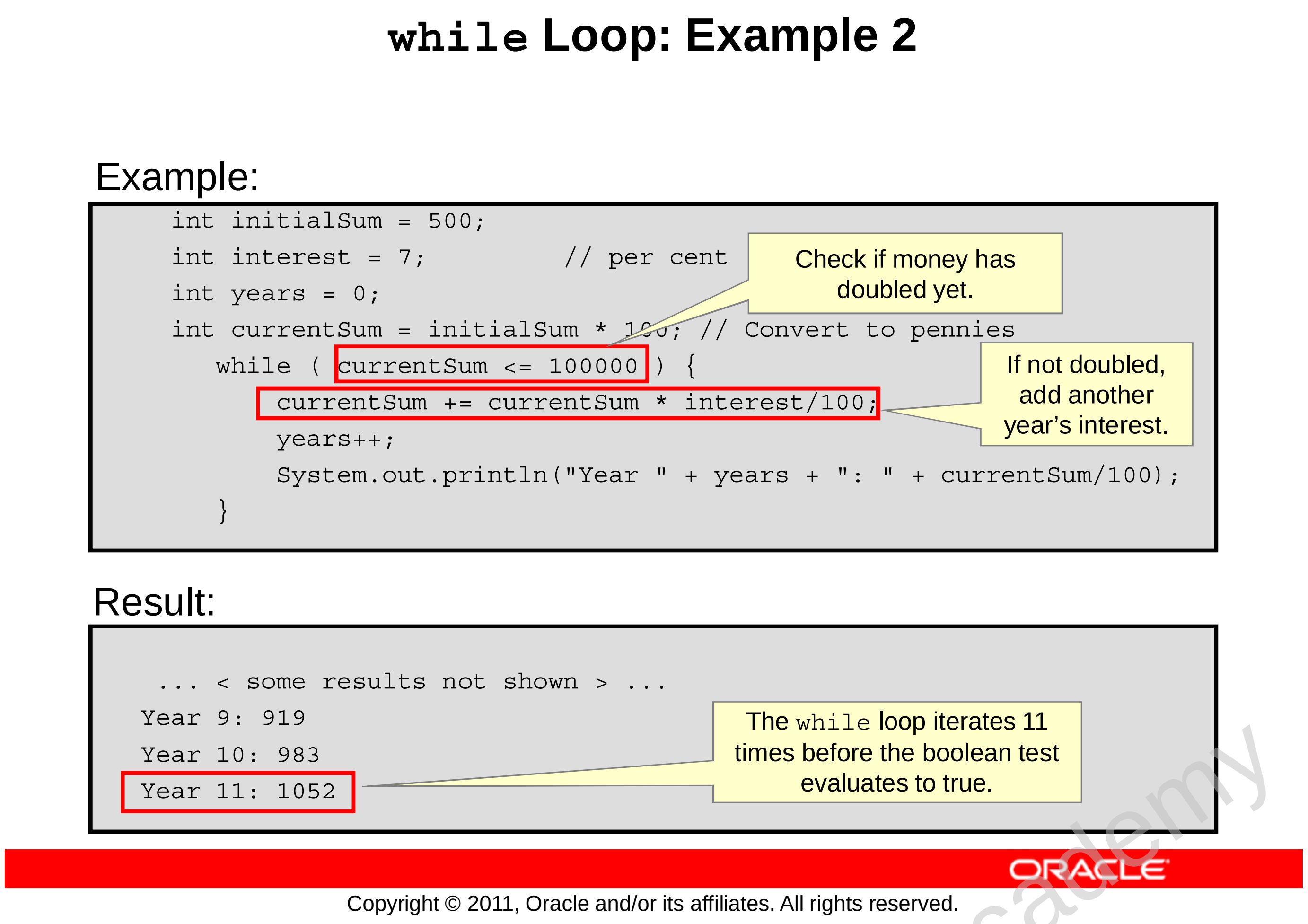
```
Next try will be 2.5
Next try will be 2.05
Next try will be 2.0006099
Next try will be 2.0
The square root of 4.0 is 2.0
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example shows some code for generating the square root of number. The boolean expression squares the current value of the square root and checks to see whether it is close to the number of which you are trying to find the square root. If it is close enough (the expression returns true), the program execution skips the statements in the `while` block and continues with the `System.out.println()` statement that outputs the square root. If the value is not yet close enough, the code within the block runs and does two things:

- Adjusts the value of `squareRoot` so that it will be closer the next time it is checked
- Prints the current “guessed” value of `squareRoot`

Oracle Internal & Oracle Academy Use Only



`while` Loop: Example 2

Example:

```
int initialSum = 500;
int interest = 7;           // per cent
int years = 0;
int currentSum = initialSum * 100; // Convert to pennies
   while ( currentSum <= 100000 ) {
      currentSum += currentSum * interest/100;
      years++;
      System.out.println("Year " + years + ": " + currentSum/100);
   }
```

Check if money has doubled yet.

If not doubled, add another year's interest.

Result:

```
 ... < some results not shown > ...
Year 9: 919
Year 10: 983
Year 11: 1052
```

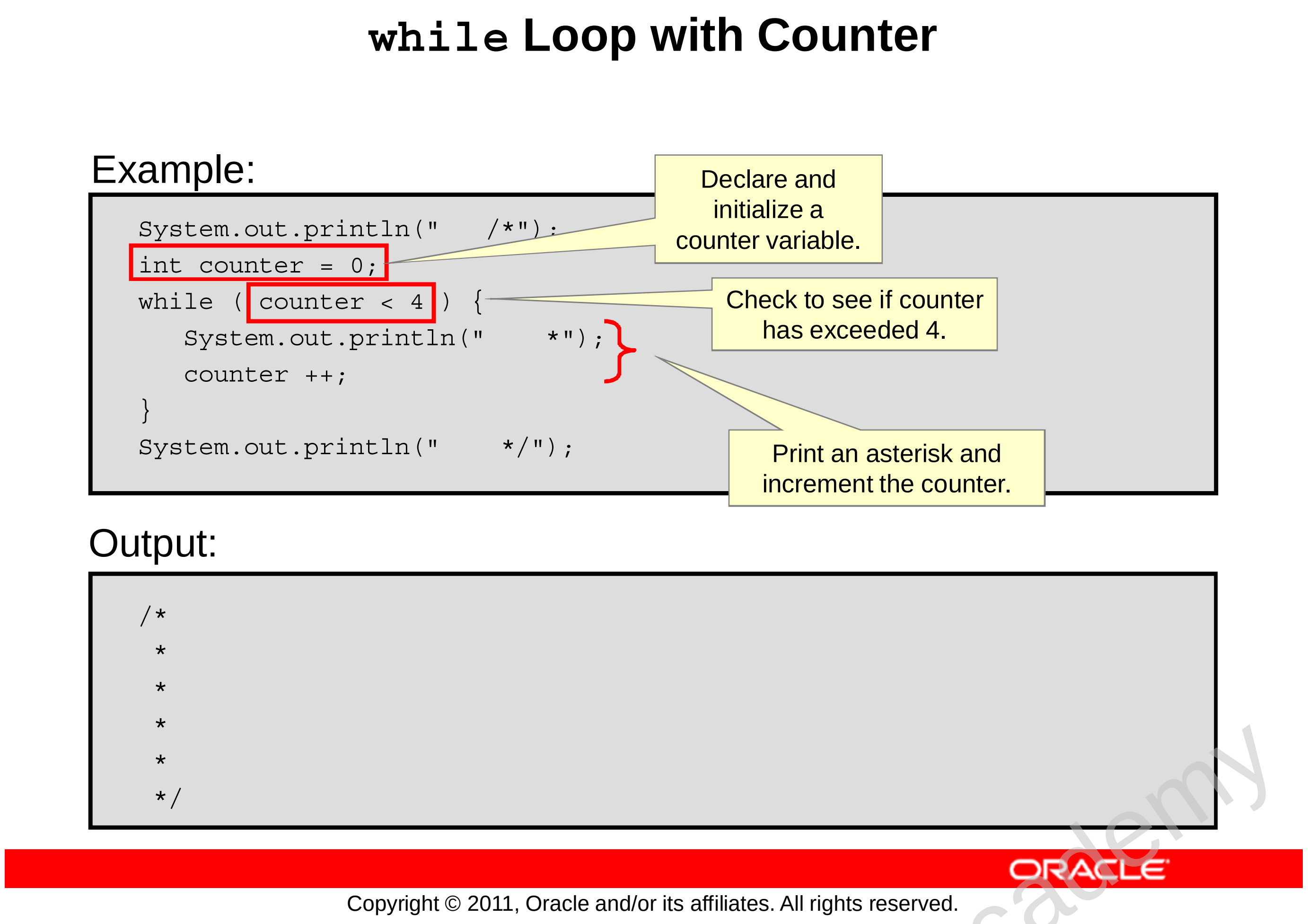
The `while` loop iterates 11 times before the boolean test evaluates to true.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows how long it would take to double your money at a particular interest rate. The `while` loop's boolean expression checks to see whether your money (converted to pennies) has doubled. If it has not, the block of the loop adds the interest of another year to the current total, and the loop repeats the boolean expression check.

Note: Converting to pennies is done to simplify the example so that the `int` type can be used.

Oracle Internal & Oracle Academy Use Only



while Loop with Counter

Example:

```
System.out.println("   /*");
int counter = 0;
while ( counter < 4 ) {
   System.out.println("    *");
   counter ++;
}
System.out.println("    */");
```

Declare and initialize a counter variable.

Check to see if counter has exceeded 4.

Print an asterisk and increment the counter.

Output:

```
/*
 *
 *
 *
 *
 */
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Loops are often used to repeat a set of commands a specific number of times. You can easily do this by declaring and initializing a counter (usually of type `int`), incrementing that variable inside the loop, and checking if the counter has reached a specific value in the `while` boolean expression.

Although this works, Java has a special counter loop (a `for` loop), which is covered in the following slides.

Oracle Internal & Oracle Academy Use Only

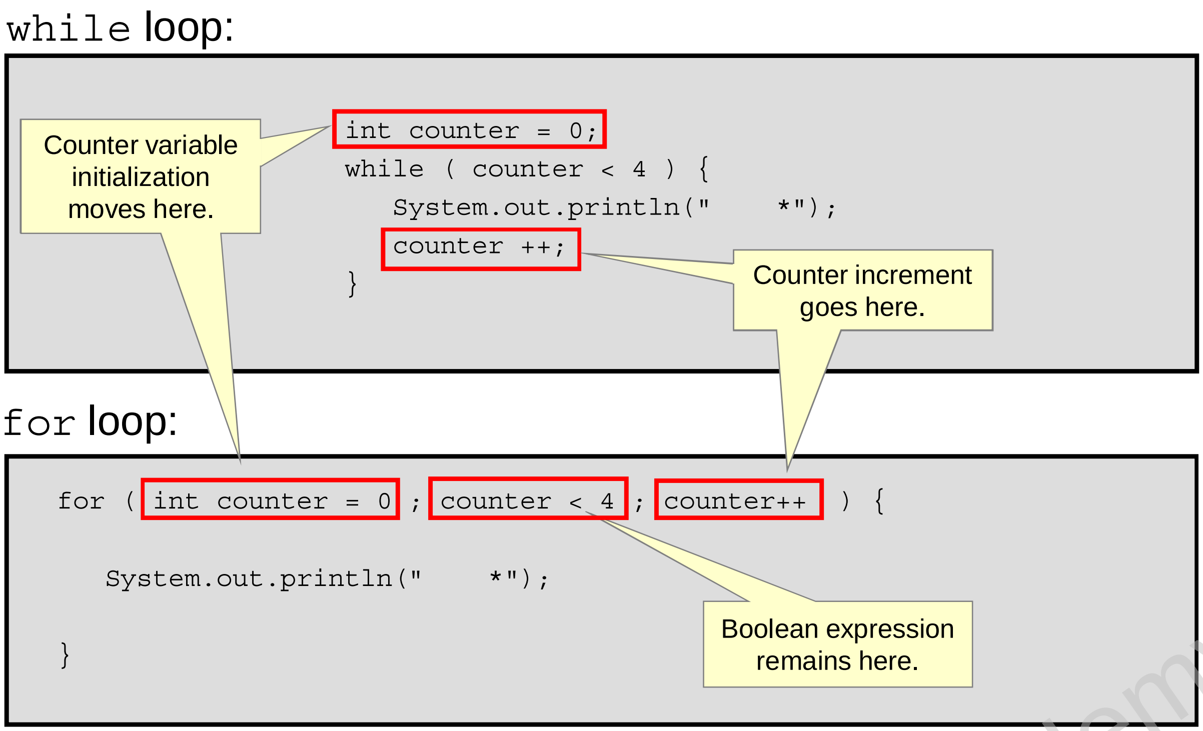
Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

for Loop



while loop:

```
int counter = 0;
while ( counter < 4 ) {
   System.out.println("    *");
   counter ++;
}
```

Counter variable initialization moves here.
Counter increment goes here.
Boolean expression remains here.

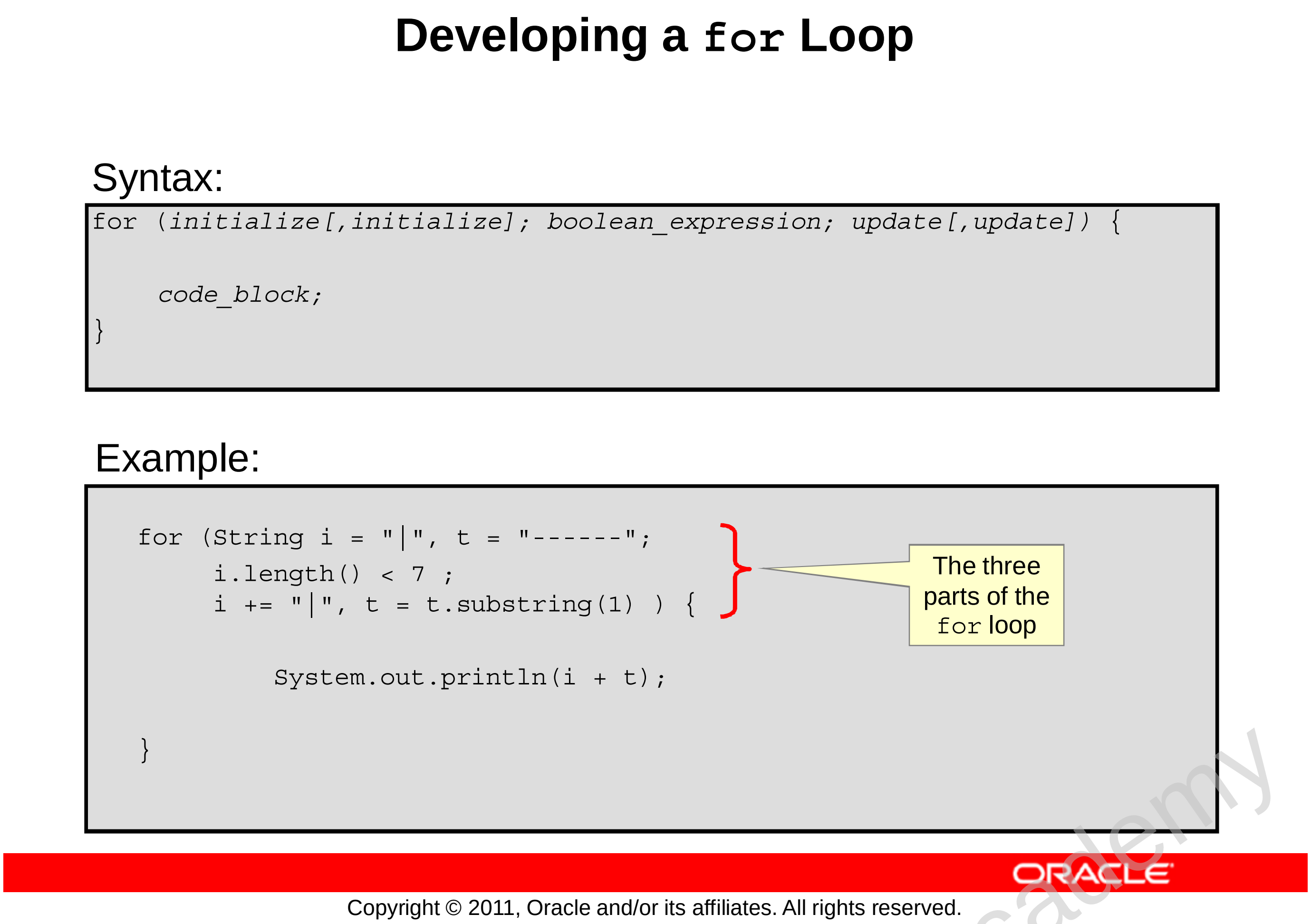
for loop:

```
for ( int counter = 0 ; counter < 4 ; counter++ ) {

   System.out.println("    *");

}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the for loop, the three expressions needed for a loop that runs a set number of times are all moved into the parentheses after the for keyword. This makes the for loop more compact and readable.

Oracle Internal & Oracle Academy Use Only



Developing a `for` Loop

Syntax:

```
for (initialize[,initialize]; boolean_expression; update[,update]) {

    code_block;
}
```

Example:

```
for (String i = "|", t = "------";
     i.length() < 7 ;
     i += "|", t = t.substring(1) ) {

         System.out.println(i + t);

}
```

The three parts of the `for` loop

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Notice that `for` loops are very versatile; you can initialize more than one variable in the first part and modify more than one variable in the third part of the `for` statement. Also, the type need not be an `int`.

The code in the slide declares two Strings and, as it loops, appends to one String while removing from the other String. These changes are in the third part of the `for` statement. This part is for updates and, although often used for incrementing the String, can be used for any kind of update (as shown here).

The output of the loop is:

```
|------
||-----
|||----
||||---
|||||--
||||||-
```

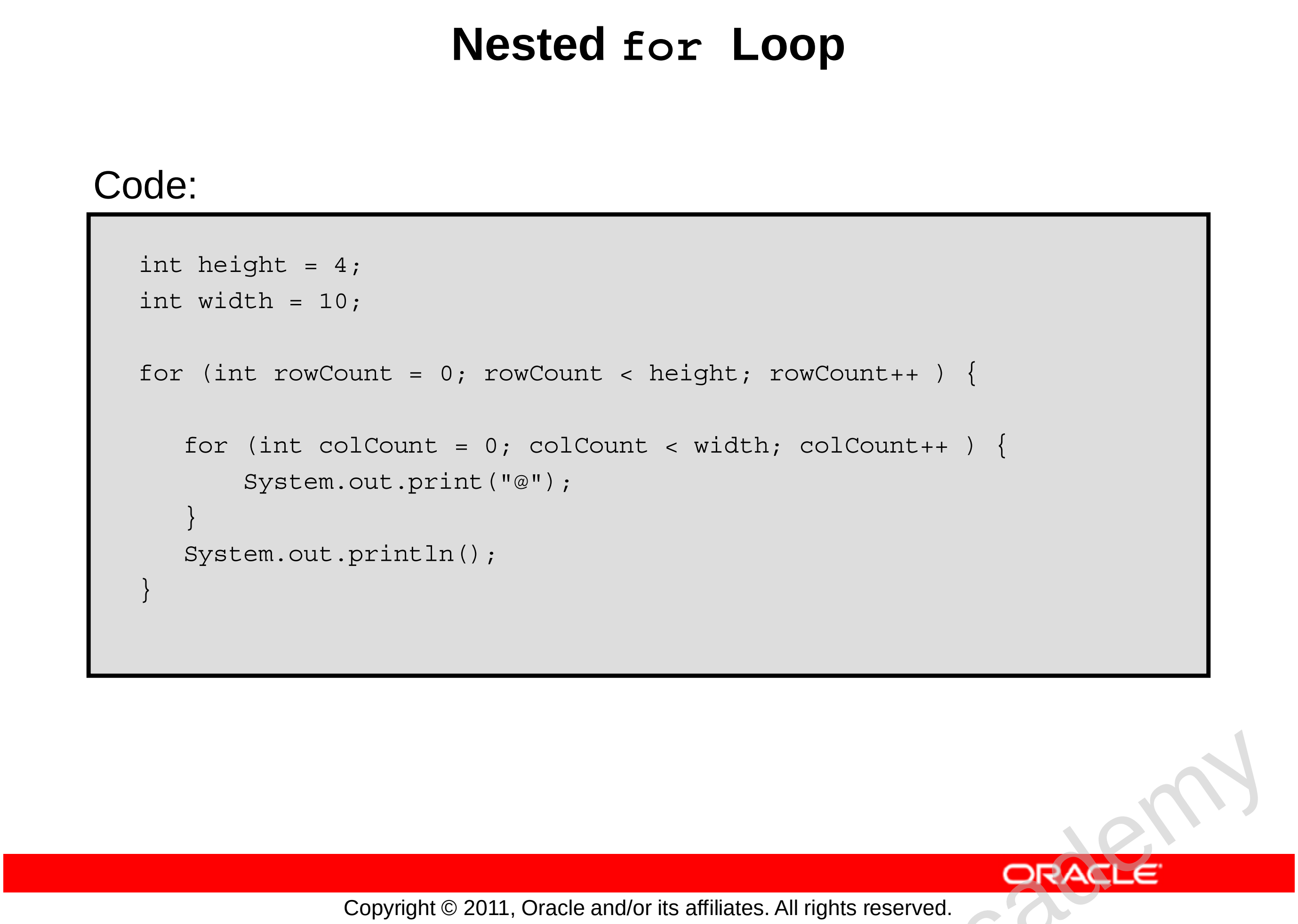
Oracle Internal & Oracle Academy Use Only

Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Nested `for` Loop

Code:

```
int height = 4;
int width = 10;

for (int rowCount = 0; rowCount < height; rowCount++ ) {

   for (int colCount = 0; colCount < width; colCount++ ) {
       System.out.print("@");
   }
   System.out.println();
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows a simple nested loop to output a block of `@` symbols with height and width given in the initial local variables. Notice how the outer code prints a new line to start a new row, while the inner loop uses the `print()` method of `System.out` to print an `@` symbol for every column.

Oracle Internal & Oracle Academy Use Only

Nested `while` Loop

Code:

```
String name = "Lenny";
String guess = "";
int numTries = 0;

while (!guess.equals(name.toLowerCase())) {
    guess = "";
    while (guess.length() < name.length()) {
       char asciiChar = (char)(Math.random() * 26 + 97);
       guess = guess + asciiChar;
    }
    numTries++;
}
System.out.println(name + " found after " + numTries + " tries!");
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Here's a nested `while` loop that is a little more complex than the previous `for` example. The nested loop tries to guess a name by building a String of the same length completely at random.

Looking at the inner loop first, the code initializes `char asciiChar` to a lowercase letter randomly. These `chars` are then added to `String guess`, until that String is as long as the String that it is being matched against. Notice the convenience of the concatenation operator here, allowing concatenation of a String and a `char`.

The outer loop tests to see if the guess is the same as a lowercase version of the original name. If it is not, `guess` is reset to an empty String and the inner loop runs again, usually millions of times for a five-letter name. (Note that names longer than five letters will take a very long time.)

Oracle Internal & Oracle Academy Use Only

Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Loops and Arrays

One of the most common uses of loops is when working with sets of data.

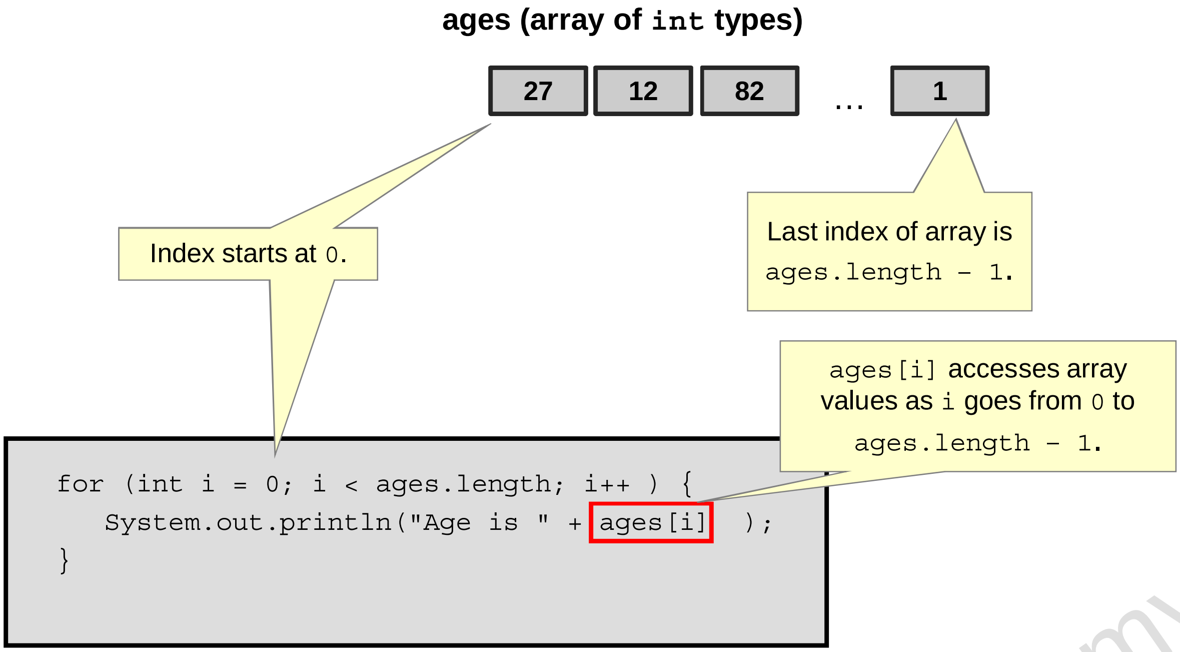
All types of loops are useful:

- `while` loops (to check for a particular value
- `for` loops (to go through the entire array)
- Enhanced `for` loops

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

for Loop with Arrays



Index starts at 0.

Last index of array is ages.length – 1.

ages[i] accesses array values as i goes from 0 to ages.length – 1.

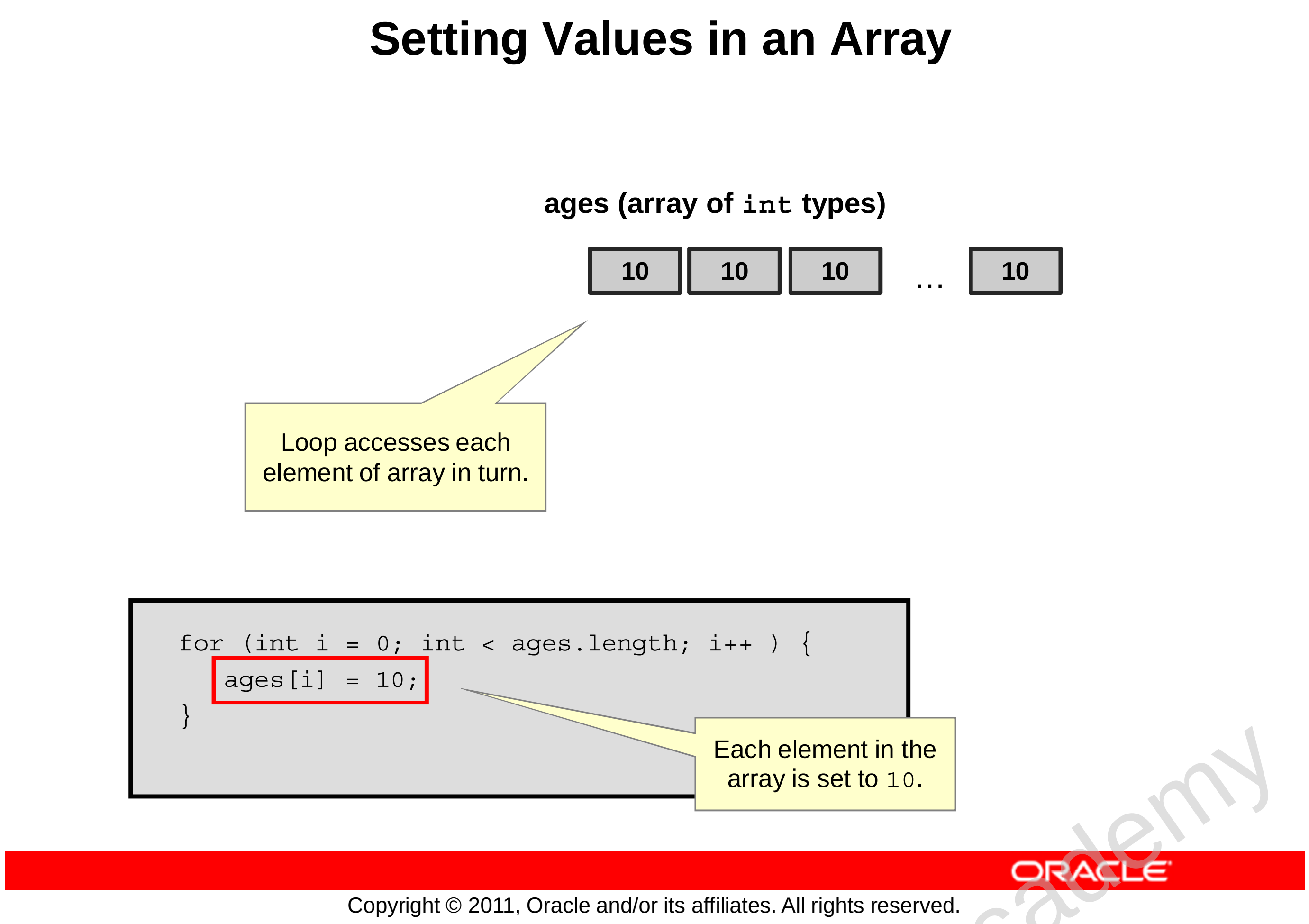
```
for (int i = 0; i < ages.length; i++ ) {
   System.out.println("Age is " + ages[i]  );
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Output:

```
Age is 27
Age is 12
Age is 82
  …
Age is 1
```

Oracle Internal & Oracle Academy Use Only



Setting Values in an Array

ages (array of `int` types)

Loop accesses each element of array in turn.

```
for (int i = 0; int < ages.length; i++ ) {
   ages[i] = 10;
}
```

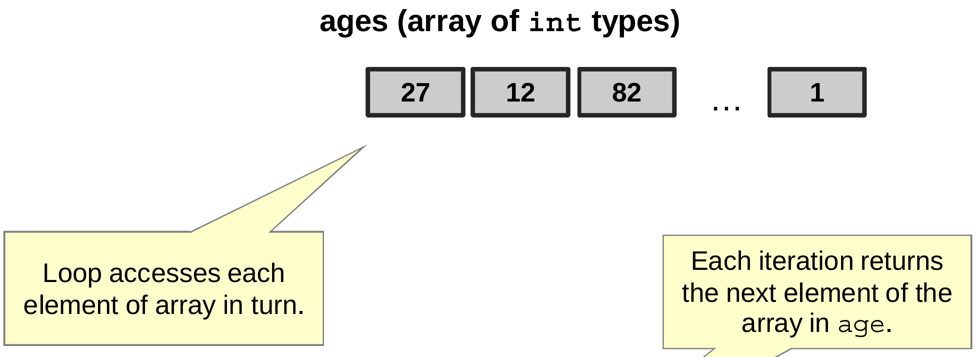
Each element in the array is set to `10`.

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Enhanced `for` Loop with Arrays



Loop accesses each element of array in turn.

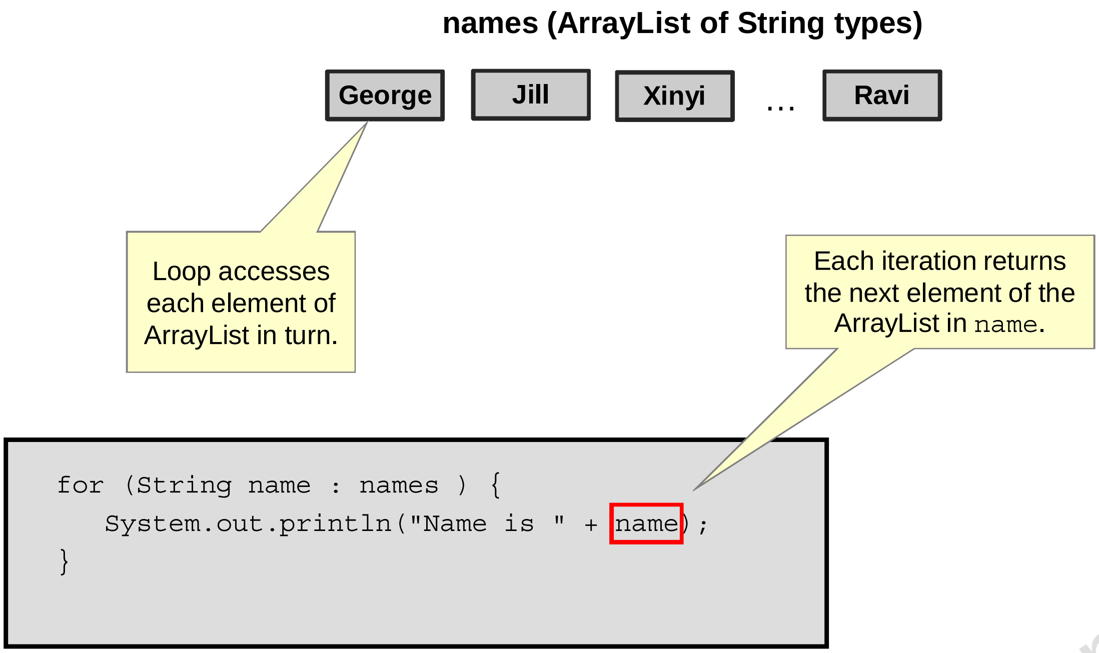
Each iteration returns the next element of the array in `age`.

```
for (int age : ages ) {
    System.out.println("Age is " + age  );
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Enhanced `for` Loop with ArrayLists


names (ArrayList of String types)
George
Jill
Xinyi
…
Ravi
Loop accesses each element of ArrayList in turn.
Each iteration returns the next element of the ArrayList in name.

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

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

ArrayLists can be iterated through in exactly the same way as arrays.

Oracle Internal & Oracle Academy Use Only

Using `break` with Loops

`break` example:

```
int passmark = 12;
boolean passed = false;
int[] score = { 4, 6, 2, 8, 12, 34, 9 };
for (int unitScore : score ) {
    if ( unitScore > passmark ) {
      passed = true;
      break;
    }
}
System.out.println("One or more units passed? " + passed);
```

There is no need to go through the loop again, so use break.

Output:

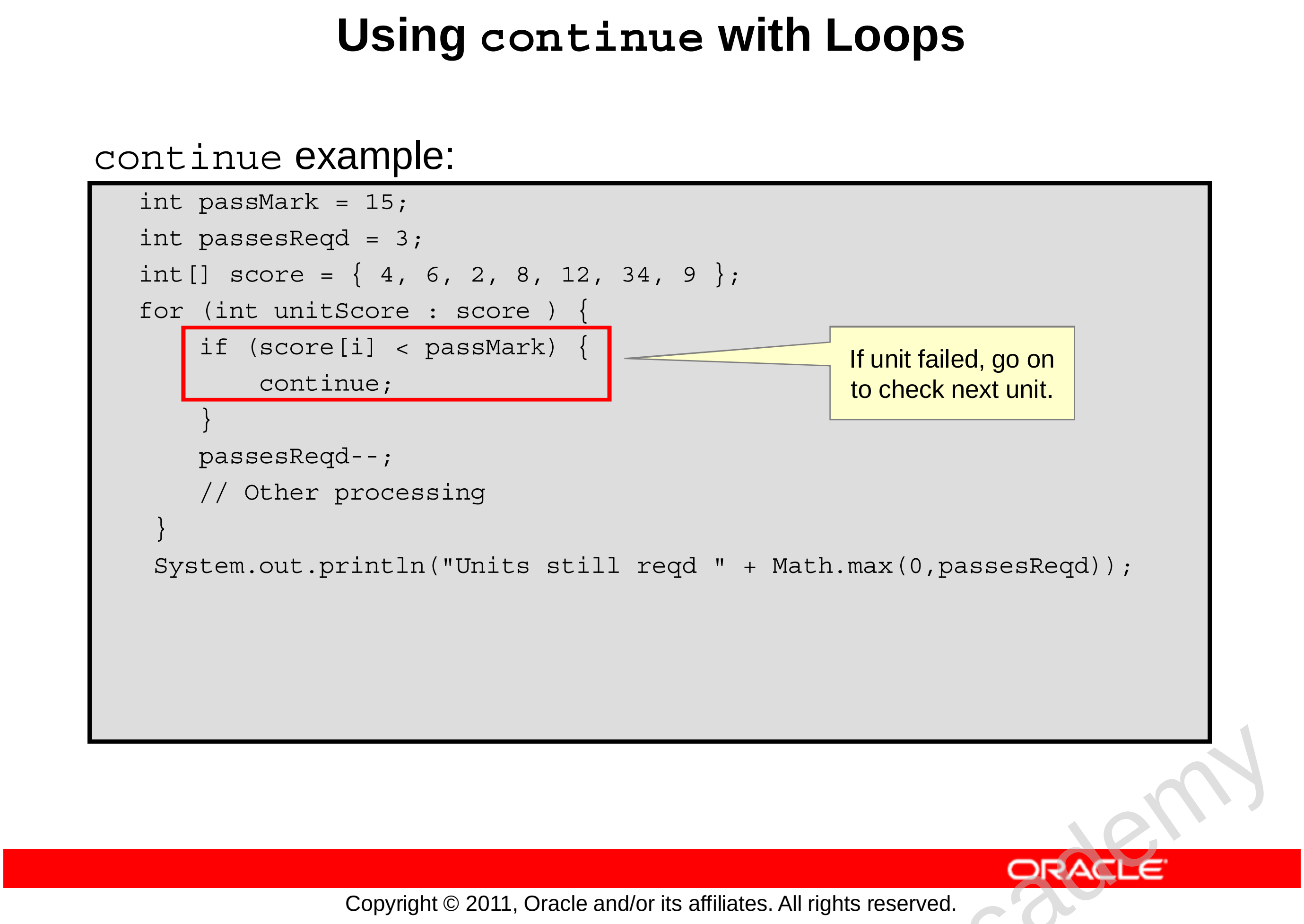
```
One or more units passed? true
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

There are two useful keywords that can be used when you work with loops: `break` and `continue`. `break` enables you to jump out of a loop, while `continue` sends you back to the start of the loop.

The example in the slide shows the use of `break`. Assuming that the code is to find out if any of the scores in the array are above `passmark`, you can set `passed` to `true` and jump out of the loop as soon as the first such score is found.

Oracle Internal & Oracle Academy Use Only



Using continue with Loops

continue example:

```
int passMark = 15;
int passesReqd = 3;
int[] score = { 4, 6, 2, 8, 12, 34, 9 };
for (int unitScore : score ) {
    if (score[i] < passMark) {
        continue;
    }
    passesReqd--;
    // Other processing
 }
 System.out.println("Units still reqd " + Math.max(0,passesReqd));
```

If unit failed, go on to check next unit.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in this slide shows the use of continue on a similar example. In this case, assume that you want to know if a certain number of passes has been achieved. So the approach is to check first to see whether the unit's score is not enough. If this is the case, the continue command goes to the start of the loop again. If the score is sufficient, the number of passesReqd is decremented and further processing possibly takes place.

This example and the previous one are intended only to show what the functions of break and continue are, and not to show particular programming techniques. Both have a similar function: They ensure that parts of the loop are not processed unnecessarily. Sometimes this can also be achieved by the design of if blocks, but it is useful to have these two options in complex algorithms.

Oracle Internal & Oracle Academy Use Only

Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Coding a `do/while` Loop

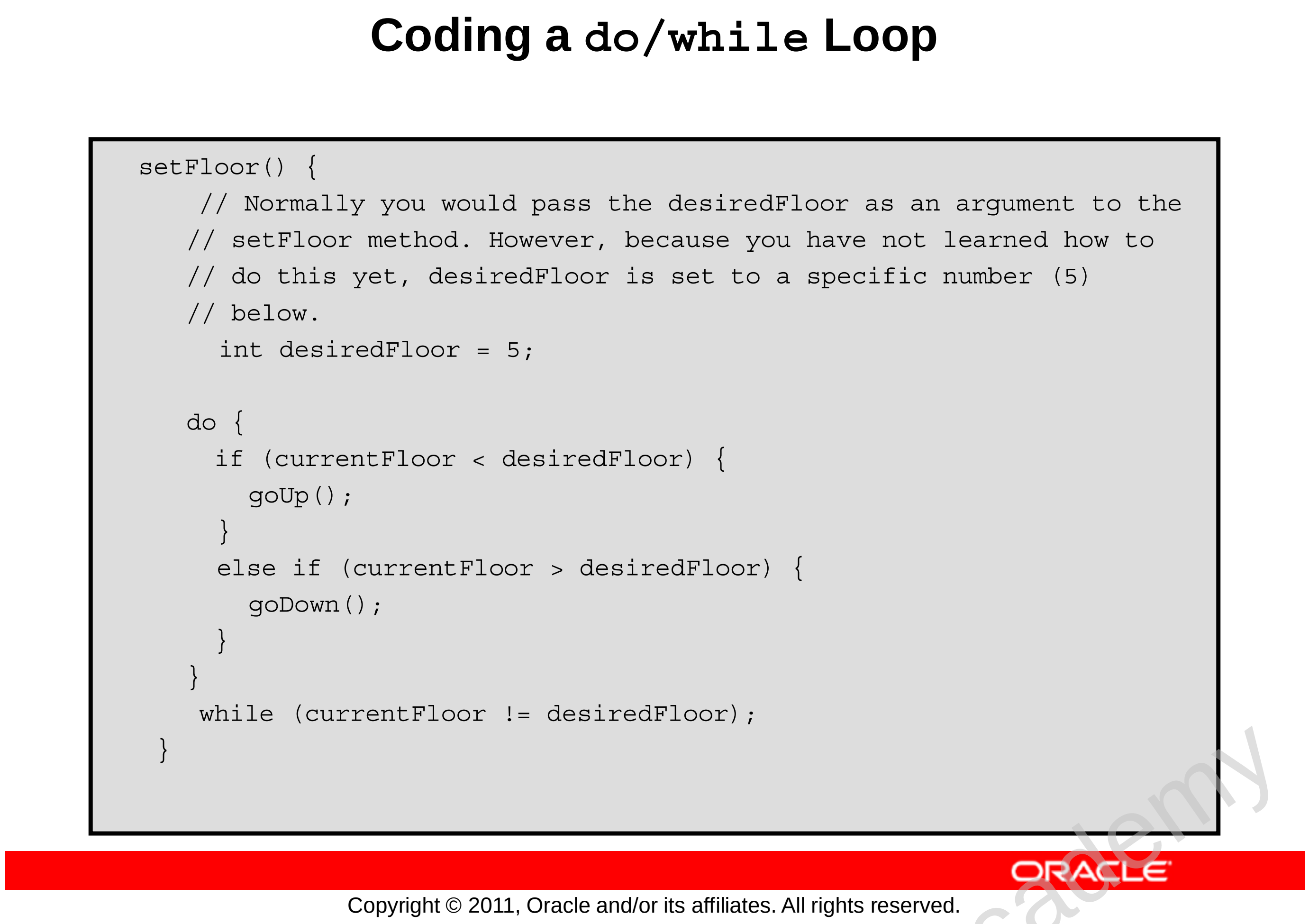
Syntax:

```
do {

    code_block;
}
while (boolean_expression); // Semicolon is mandatory.
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The `do/while` loop is a one-to-many iterative loop: The condition is at the bottom of the loop and is processed *after the body.* The *body of the loop* is, therefore, processed at least once. If you want the statement or statements in the body to be processed at least once, use a `do/while` loop instead of a `while` or `for` loop. The syntax for the `do/while` loop is shown in the slide.

Oracle Internal & Oracle Academy Use Only



Coding a do/while Loop

```
setFloor() {
    // Normally you would pass the desiredFloor as an argument to the
   // setFloor method. However, because you have not learned how to
   // do this yet, desiredFloor is set to a specific number (5)
   // below.
     int desiredFloor = 5;

   do {
     if (currentFloor < desiredFloor) {
       goUp();
     }
     else if (currentFloor > desiredFloor) {
       goDown();
     }
   }
    while (currentFloor != desiredFloor);
 }
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The `setFloor` method of the `Elevator` class uses a `do/while` loop to determine whether the elevator is at the chosen floor. If the value of the `currentFloor` variable is not equal to the value of the `desiredFloor` variable, the elevator continues moving either up or down.

Oracle Internal & Oracle Academy Use Only

Topics

- Create a `while` loop
- Develop a `for` loop
- Nest a `for` loop and a `while` loop
- Use an array in a `for` loop
- Code and nest a `do/while` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Comparing Loop Constructs

- Use the `while` loop to iterate indefinitely through statements and to perform the statements zero or more times.
- Use the `do/while` loop to iterate indefinitely through statements and to perform the statements *one* or more times.
- Use the `for` loop to step through statements a predefined number of times.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Quiz

________________ enable you to check and recheck a decision to execute and re-execute a block of code.

a. Classes
b. Objects
c. Loops
d. Methods

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Quiz

Which of the following loops always executes at least once?

a. The `while` loop

b. The nested `while` loop

c. The `do/while` loop

d. The `for` loop

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Create a `while` loop
- Nest a `while` loop
- Develop and nest a `for` loop
- Code and nest a `do/while` loop
- Use an ArrayList in a `for` loop
- Compare loop constructs

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 9-1 Overview: Writing a Class That Uses a `for` Loop

In this practice, you create the `Counter` class that uses a simple `for` loop to print a sequence of numbers.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 9-2 Overview:
Writing a Class That Uses a while Loop

In this practice, you write a class named Sequence that displays a sequence starting with the numbers 0 and 1. Successive numbers in the sequence are the sum of the previous two numbers (for example, 0 1 1 2 3 5 8 13 21…). This sequence is also called the Fibonacci series.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Challenge Practice 9-3 Overview: Converting a `while` Loop to a `for` Loop

In this practice, you convert an existing `while` loop to a `for` loop. During this practice, you:

- Create a new class, `ChallengeSequence`, based on the Sequence class you created in the last practice
- Modify the `displaySequence` method to use a `for` loop instead of a `while` loop

Note: This practice (9-3) is an optional Challenge practice.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 9-4 Overview:
Using `for` Loops to Process an ArrayList

In this practice, you create two new methods in two different classes. This practice contains two sections:

- Using a `for` loop with the `VacationScaleTwo` class
- Using an enhanced `for` loop with the `NamesListTwo` class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 9-5 Overview: Writing a Class That Uses a Nested `for` Loop to Process a Two-Dimensional Array

In this practice, you create and process a two-dimensional array using a nested `for` loop.

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Challenge Practice 9-6 Overview: Adding a Search Method to `ClassMap`

In this practice, you add another method to `ClassMap`. This method searches through `deskArray` to find a certain name.

Note: This practice (9-6) is an optional Challenge practice.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

10

Working with Methods and Method Overloading

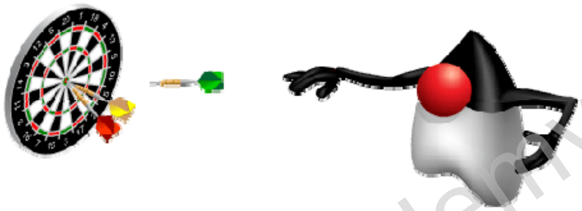
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Declare methods with arguments and return values
- Declare static methods and variables
- Create an overloaded method



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

- Creating and invoking methods
- Static methods and variables
- Method overloading

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Creating and Invoking Methods

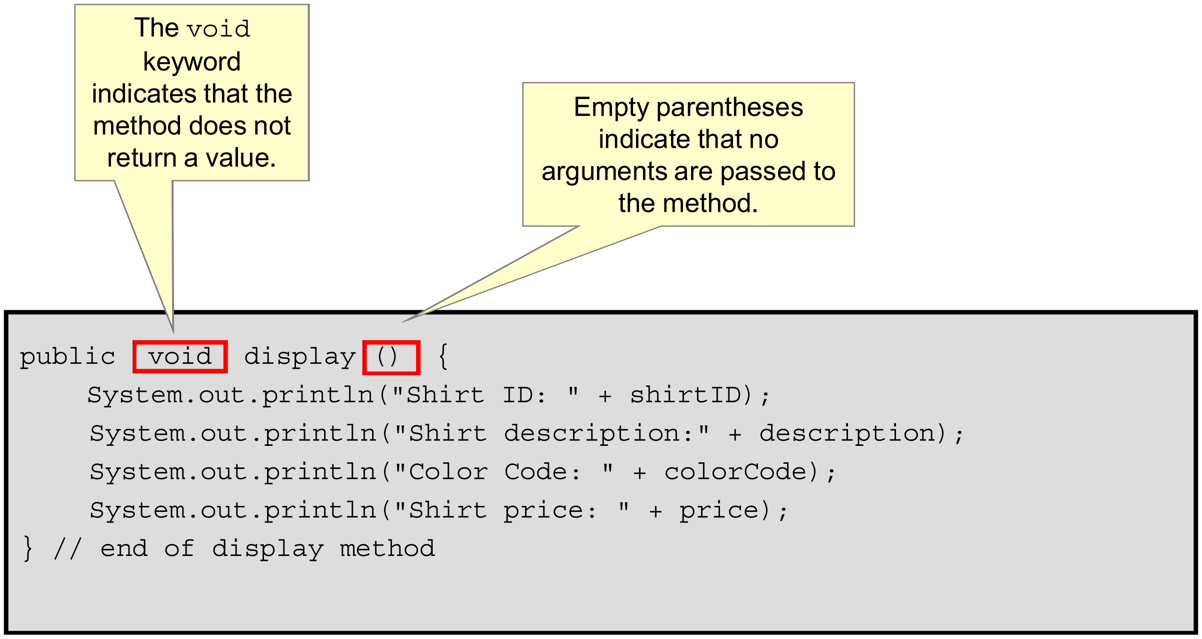
Syntax:

```
[modifiers] return_type method_identifier ([arguments]) {
  method_code_block
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Basic Form of a Method



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

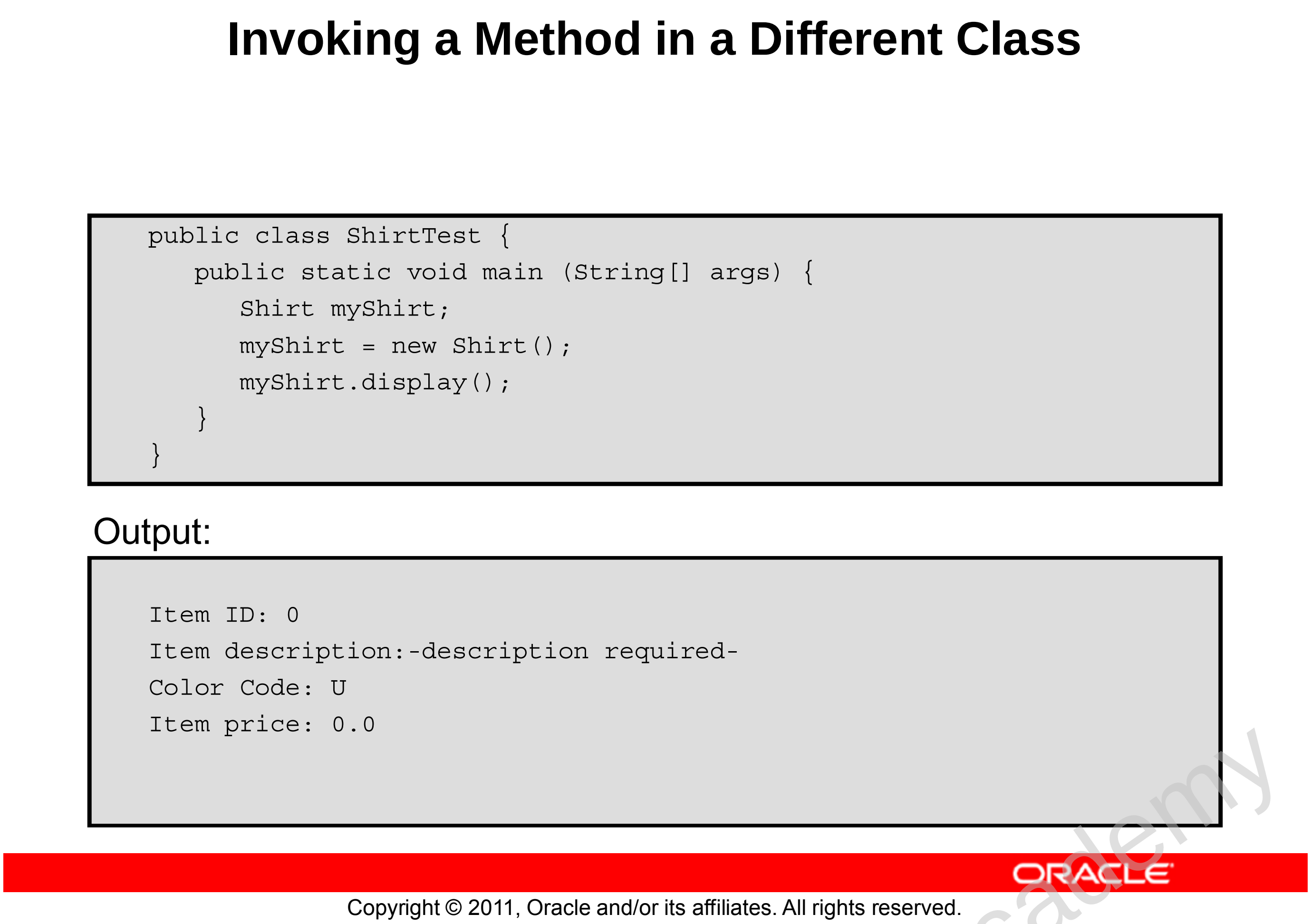
Empty parentheses indicate that no arguments are passed to the method.

```
public  void  display () {
     System.out.println("Shirt ID: " + shirtID);
     System.out.println("Shirt description:" + description);
     System.out.println("Color Code: " + colorCode);
     System.out.println("Shirt price: " + price);
} // end of display method
```

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only



Invoking a Method in a Different Class

```
public class ShirtTest {
   public static void main (String[] args) {
      Shirt myShirt;
      myShirt = new Shirt();
      myShirt.display();
   }
}
```

Output:

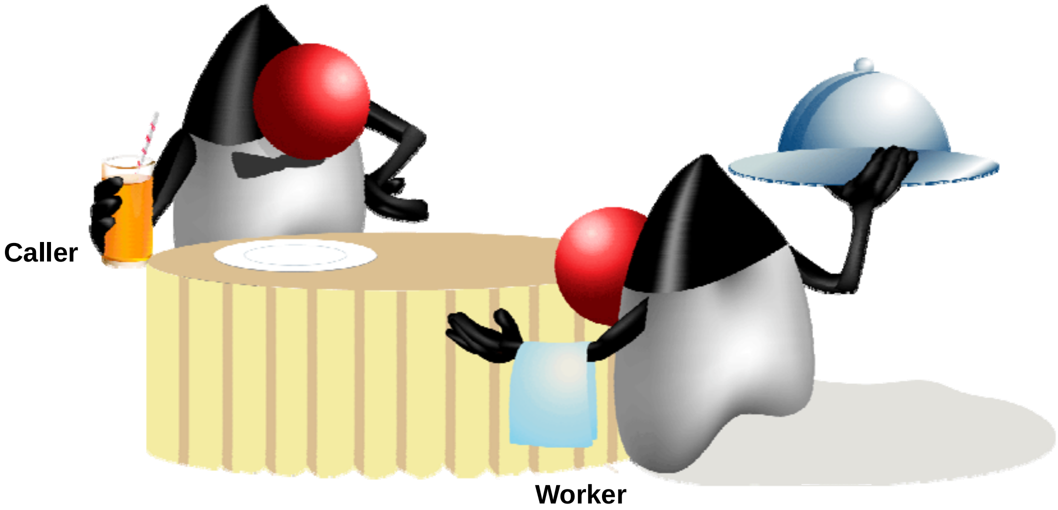
```
Item ID: 0
Item description:-description required-
Color Code: U
Item price: 0.0
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the example in this slide, `display()` is called. But because the Shirt object has not had any of its fields set, the default values for those fields are displayed.

Oracle Internal & Oracle Academy Use Only

Caller and Worker Methods



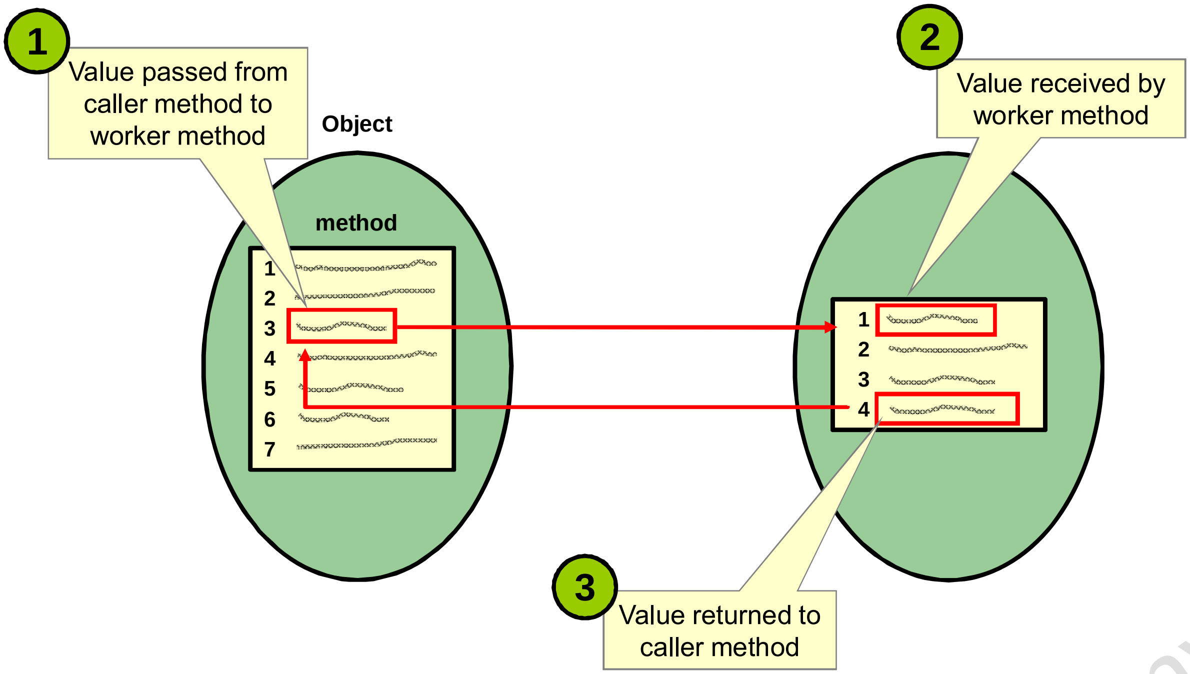
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the previous example, the `ShirtTest` class calls the `display()` method from within another method (the `main` method). Therefore, 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.

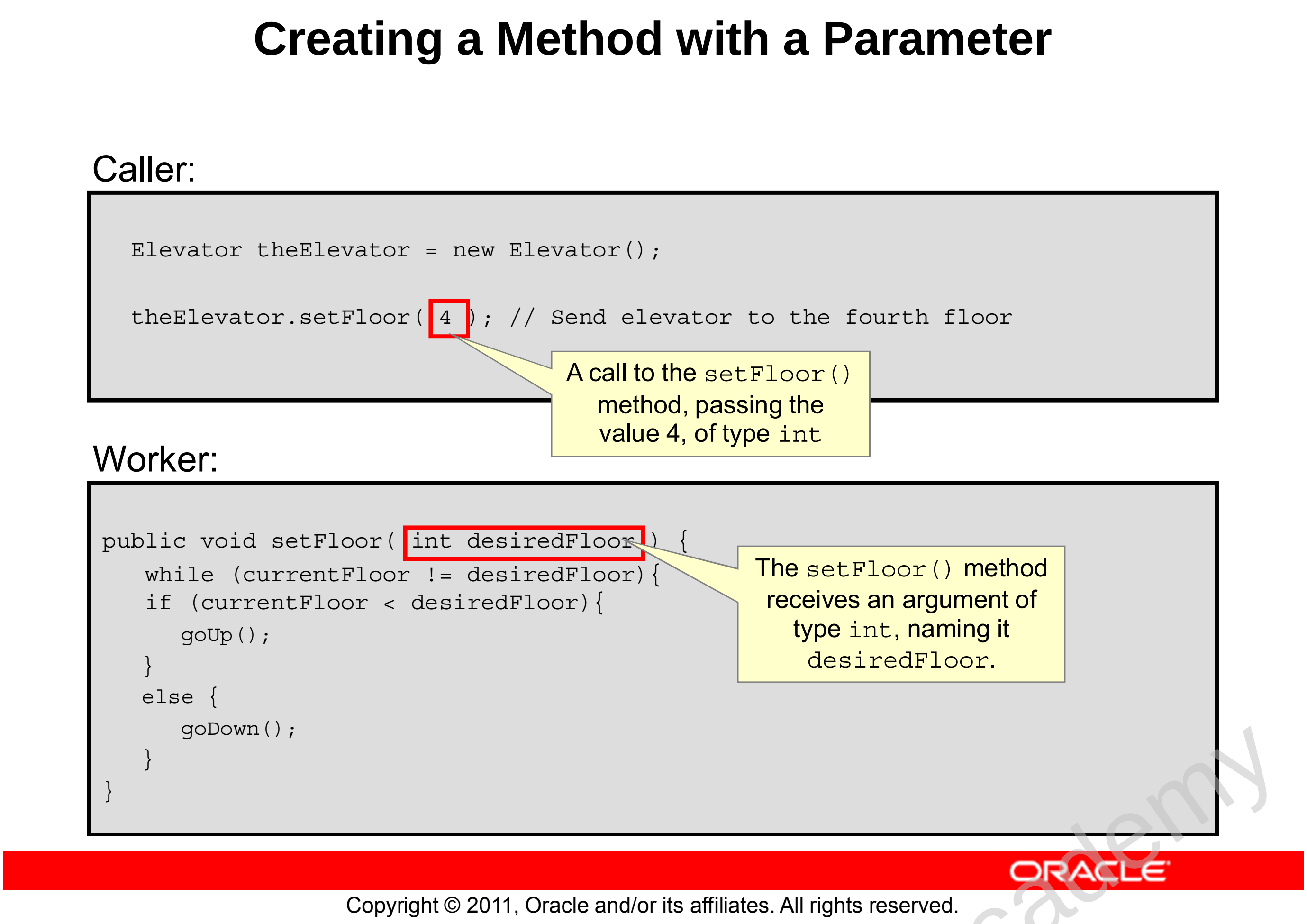
Oracle Internal & Oracle Academy Use Only

Passing Arguments and Returning Values


1
Value passed from caller method to worker method
Object
method
1
2
3
4
5
6
7
2
Value received by worker method
1
2
3
4
3
Value returned to caller method

ORACLE
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Creating a Method with a Parameter

Caller:

```
Elevator theElevator = new Elevator();

theElevator.setFloor( 4 ); // Send elevator to the fourth floor
```

A call to the `setFloor()` method, passing the value 4, of type `int`

Worker:

```
public void setFloor( int desiredFloor ) {
   while (currentFloor != desiredFloor){
   if (currentFloor < desiredFloor){
      goUp();
   }
   else {
      goDown();
   }
}
```

The `setFloor()` method receives an argument of type `int`, naming it `desiredFloor`.

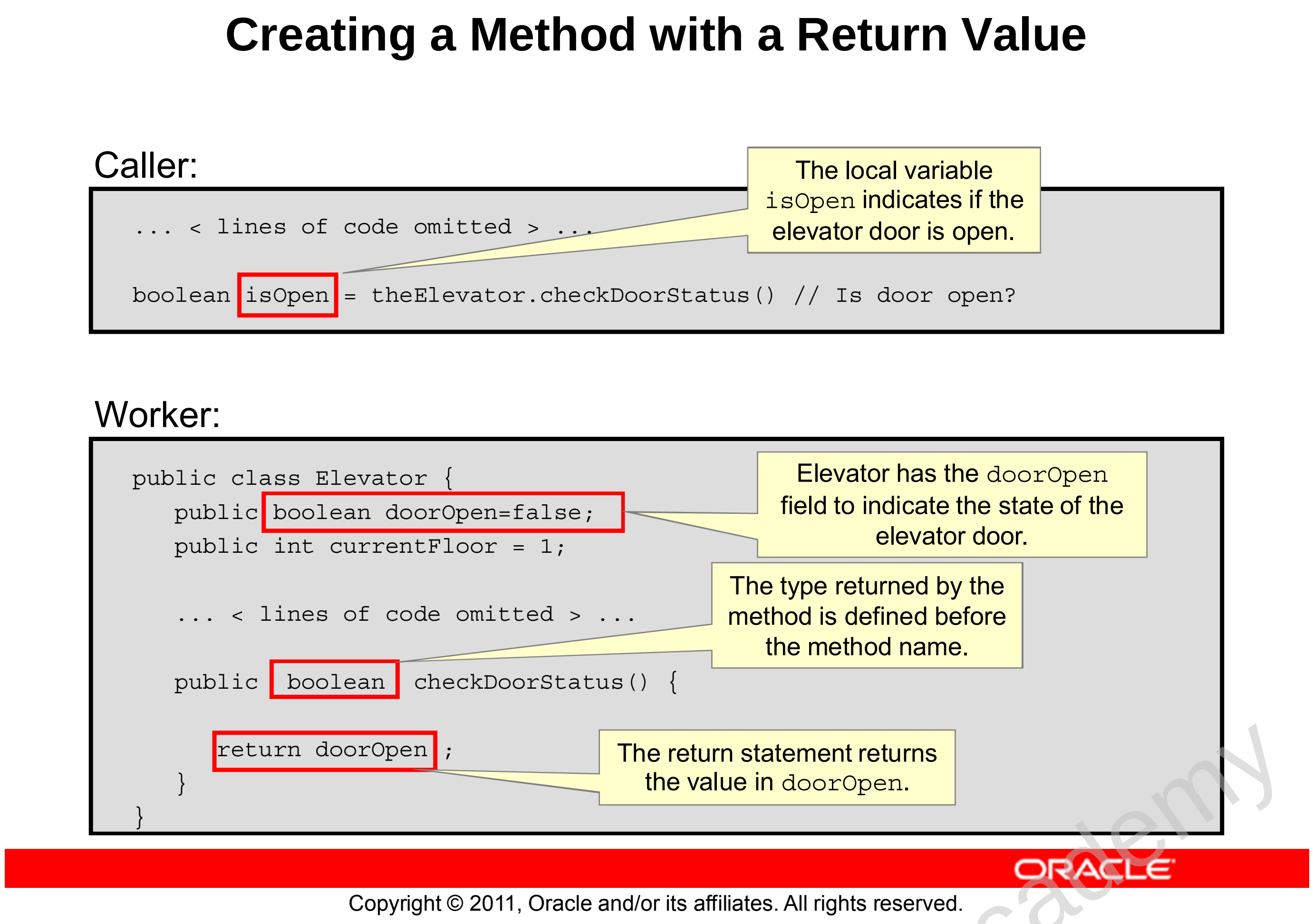
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows the `setFloor()` method (introduced in the lesson titled "Using Loop Constructs"). The method receives a value of `int` type and gives it the name `desiredFloor`. `desiredFloor` is now a local variable whose scope is the method.

It is called (in this case, from a calling method in another class) by using the dot notation and including the argument.

Note: A variable defined in the method declaration is called a *method parameter*, whereas a value passed into the method call is called an *argument*.

Oracle Internal & Oracle Academy Use Only



Creating a Method with a Return Value

Caller:

```
... < lines of code omitted > ...

boolean isOpen = theElevator.checkDoorStatus() // Is door open?
```

The local variable isOpen indicates if the elevator door is open.

Worker:

```
public class Elevator {
   public boolean doorOpen=false;
   public int currentFloor = 1;

   ... < lines of code omitted > ...

   public  boolean  checkDoorStatus() {

     return doorOpen ;
   }
}
```

Elevator has the doorOpen field to indicate the state of the elevator door.

The type returned by the method is defined before the method name.

The return statement returns the value in doorOpen.

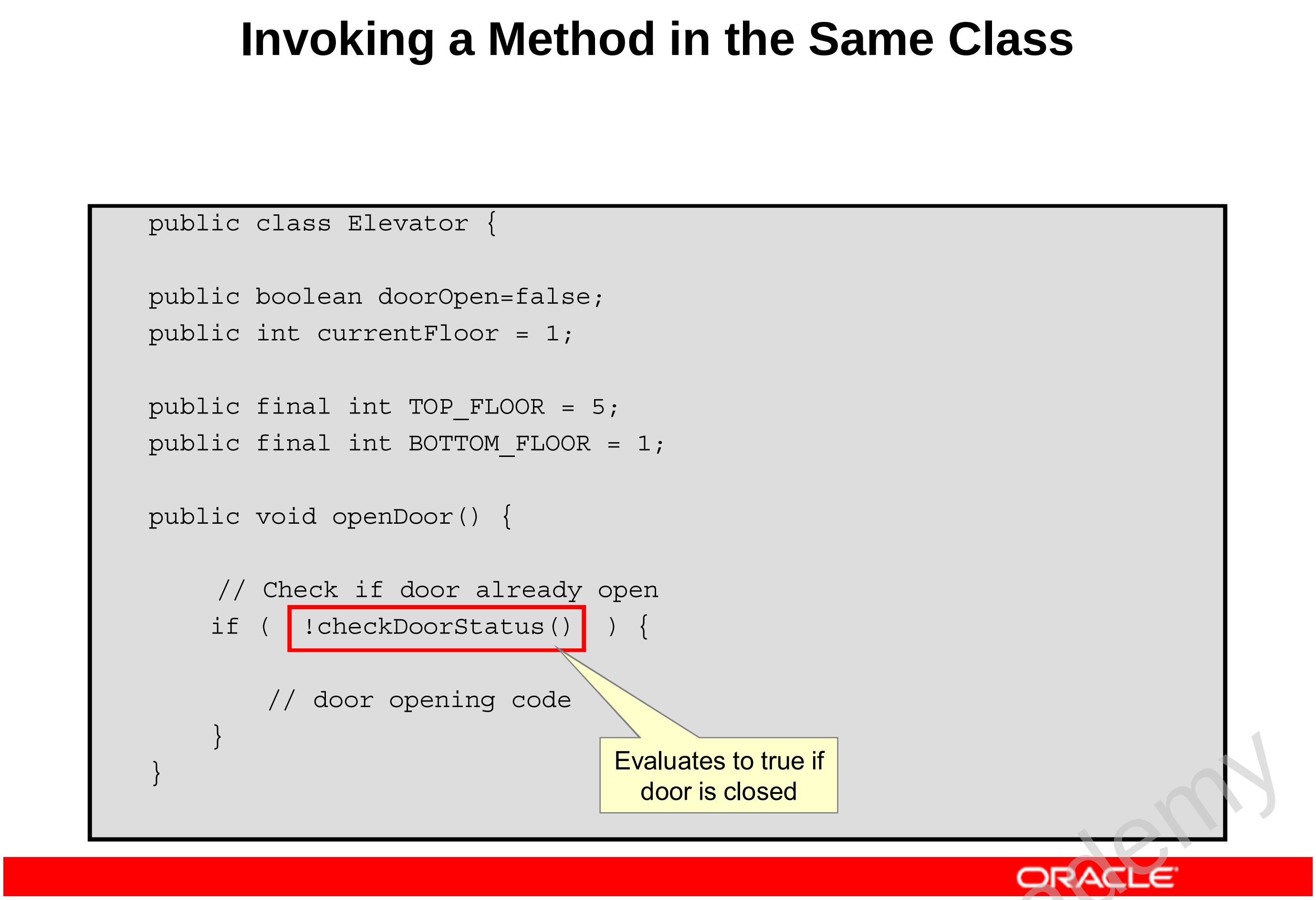
ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows the checkDoorStatus() method being called by the caller method. Note how checkDoorStatus() defines that it will return a boolean. Any single type can be defined here, or the keyword void is used if the method does not return a value.

The value is returned to the calling statement by the return statement. Note that because the method has been declared with a return type of boolean, NetBeans indicates an error if there is no return or if the return is of an incorrect type.

Oracle Internal & Oracle Academy Use Only



Invoking a Method in the Same Class

```
public class Elevator {

public boolean doorOpen=false;
public int currentFloor = 1;

public final int TOP_FLOOR = 5;
public final int BOTTOM_FLOOR = 1;

public void openDoor() {

     // Check if door already open
    if ( !checkDoorStatus() ) {

        // door opening code
    }
}
```

Evaluates to true if door is closed

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

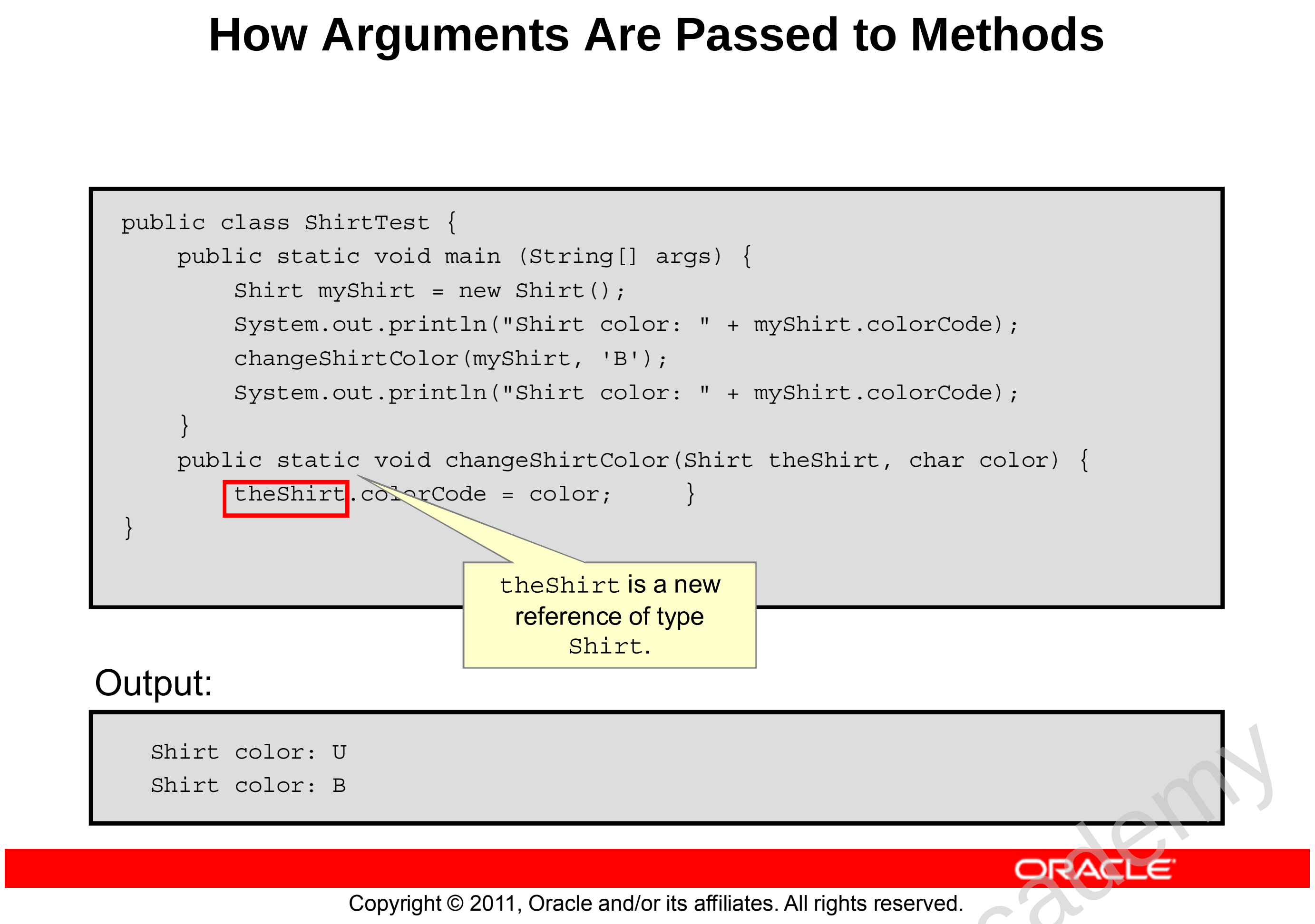
Calling a method in the same class is very straightforward. You can simply use the method name without a reference and dot notation. This is the same as when accessing a field; you can simply use the field name.

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.

Example:

```
this.checkDoorStatus()
```

Oracle Internal & Oracle Academy Use Only



How Arguments Are Passed to Methods

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt myShirt = new Shirt();
        System.out.println("Shirt color: " + myShirt.colorCode);
        changeShirtColor(myShirt, 'B');
        System.out.println("Shirt color: " + myShirt.colorCode);
    }
    public static void changeShirtColor(Shirt theShirt, char color) {
        theShirt.colorCode = color;     }
}
```

theShirt is a new reference of type Shirt.

Output:

```
Shirt color: U
Shirt color: B
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

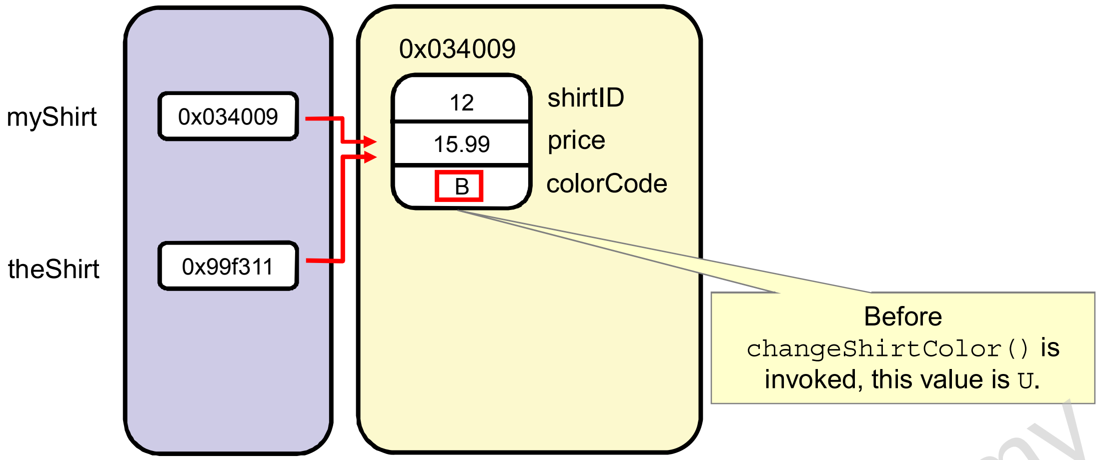
When the method is invoked, the values of the arguments are used to initialize newly created parameter variables, each of the declared type, before execution of the body of the method or constructor. This is true for both primitive types and reference types. (Objects are not passed to methods.)

This means that in the example in the slide, the reference `myShirt` is passed by value into the `changeShirtColor()` method. The reference `theShirt` inside the method is a different reference than `myShirt`. However, they both point to the same object, so the change to the color made using `theShirt` is printed out by accessing `myShirt.color`.

Oracle Internal & Oracle Academy Use Only

Passing by Value

```
Shirt myShirt = new Shirt();
changeShirtColor(myShirt, 'B');
```

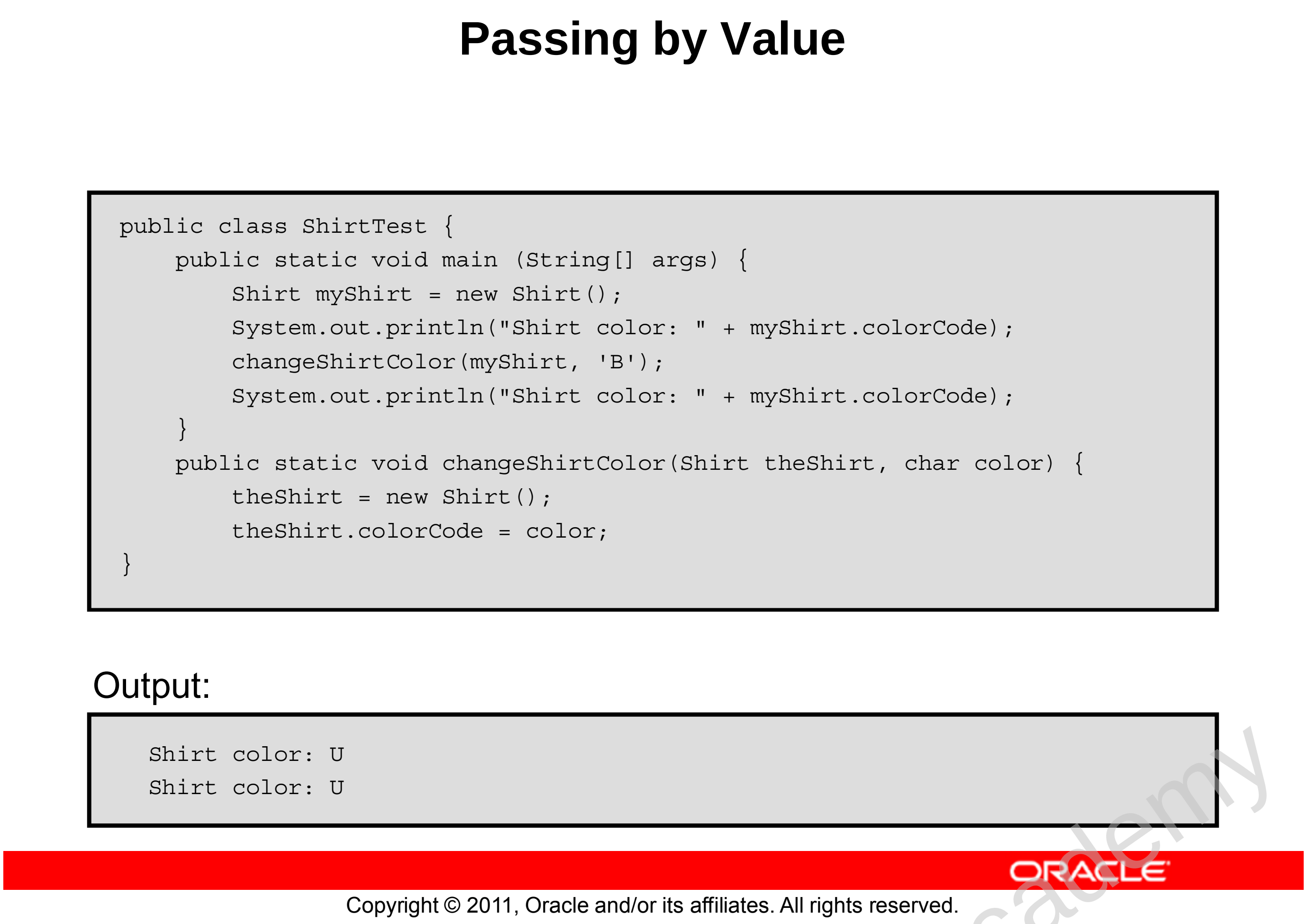


Before `changeShirtColor()` is invoked, this value is `U`.

Copyright © 2011, 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`).

Oracle Internal & Oracle Academy Use Only



Passing by Value

```
public class ShirtTest {
    public static void main (String[] args) {
        Shirt myShirt = new Shirt();
        System.out.println("Shirt color: " + myShirt.colorCode);
        changeShirtColor(myShirt, 'B');
        System.out.println("Shirt color: " + myShirt.colorCode);
    }
    public static void changeShirtColor(Shirt theShirt, char color) {
        theShirt = new Shirt();
        theShirt.colorCode = color;
}
```

Output:

```
Shirt color: U
Shirt color: U
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

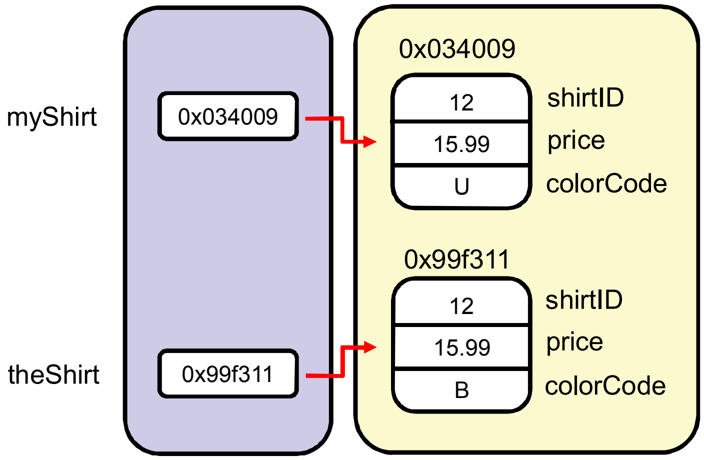
Here is another example with a small change in the code of the `changeShirtColor()` method. In this example, the reference value passed into the method is assigned to a new shirt. Then, as before, the color of the `Shirt` object is changed to `'B'`. But in this case, the line printed after the method call shows the color to still be `'U'` (Unset).

This illustrates that the reference `myShirt` is indeed passed by value. Changes made to references passed into worker methods do not affect the references in the calling method. (Note that this discussion is about changes made to references passed into the method, and not to the objects they point to.)

Oracle Internal & Oracle Academy Use Only

Passing by Value

```
Shirt myShirt = new Shirt();
changeShirtColor(myShirt, 'B');
```



Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only

Advantages of Using Methods

Methods:

- Make programs more readable and easier to maintain
- Make development and maintenance quicker
- Are central to reusable software
- Allow separate objects to communicate and to distribute the work performed by the program

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Quiz

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

a. A class can contain only one method declaration.

b. A method must always specify a return type.

c. The same method can be both a worker method and a calling method.

d. Arguments need not be listed in the same order in the method invocation as in the method signature.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

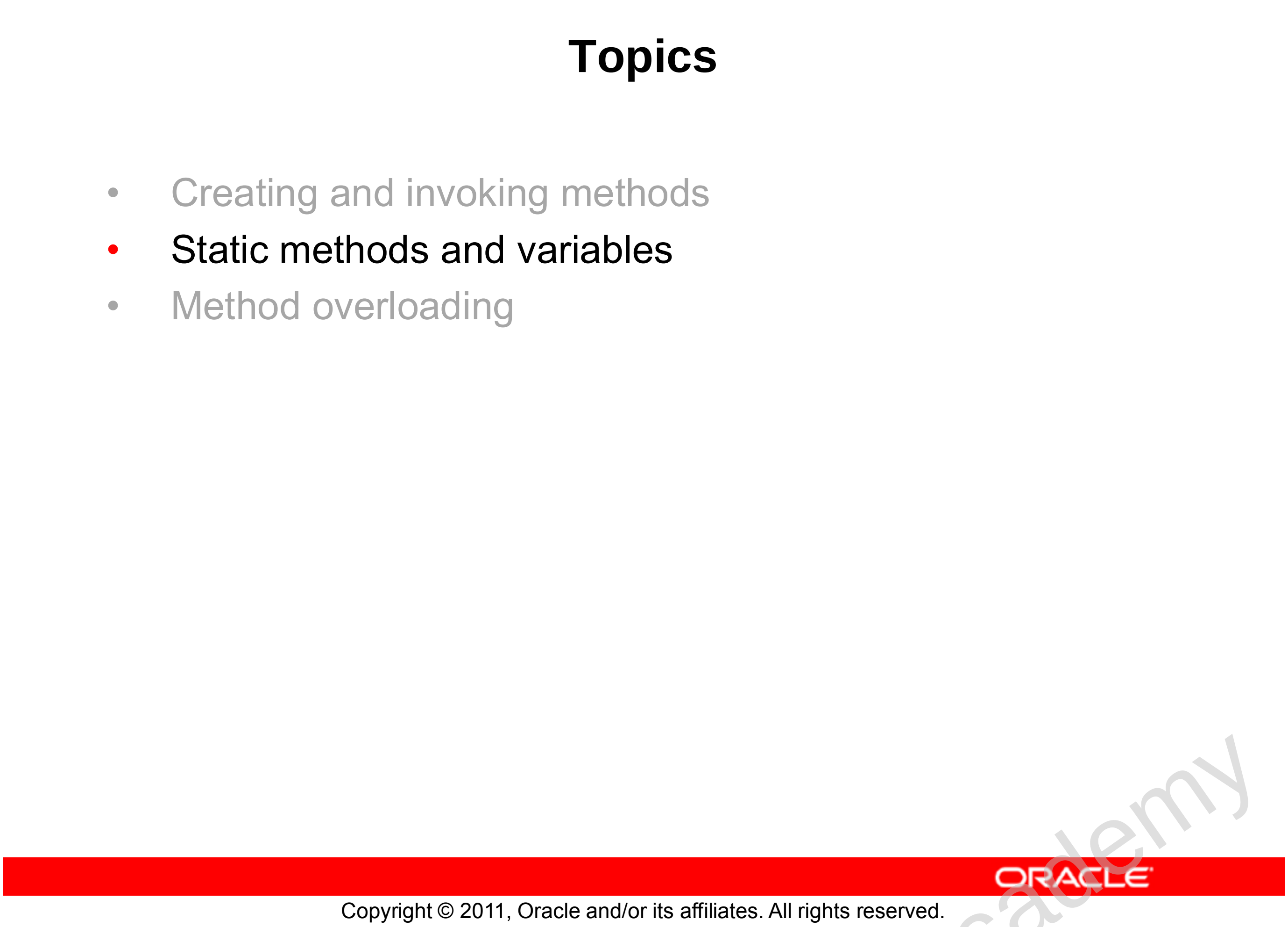
Oracle Internal & Oracle Academy Use Only

Invoking Methods: Summary

- There is no limit to the number of method calls that a calling method can make.
- The calling method and the worker method can be in the same class or in different classes.
- The way you invoke the worker method is different depending on whether it is in the same class or in a different class from the calling method.
- You can invoke methods in any order.
  - Methods do not need to be completed in the order in which they are listed in the class where they are declared (the class containing the worker methods).
- All arguments passed into a method are passed by value.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

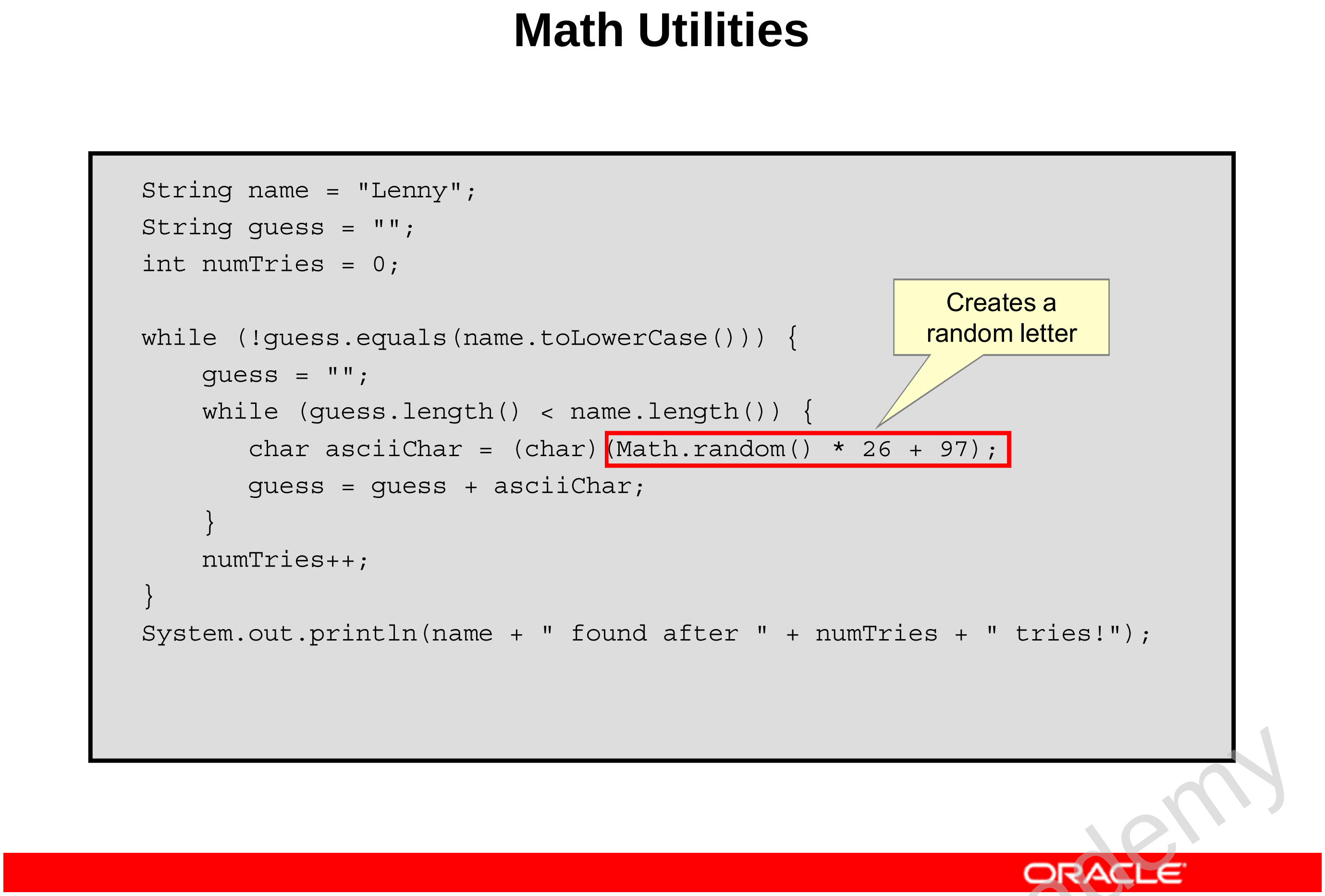


Topics

- Creating and invoking methods
- Static methods and variables
- Method overloading

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Math Utilities

```
String name = "Lenny";
String guess = "";
int numTries = 0;

while (!guess.equals(name.toLowerCase())) {
    guess = "";
    while (guess.length() < name.length()) {
       char asciiChar = (char)(Math.random() * 26 + 97);
       guess = guess + asciiChar;
    }
    numTries++;
}
System.out.println(name + " found after " + numTries + " tries!");
```

Creates a random letter

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

This slide revisits the code used in the lesson titled “Using Loop Constructs,” but one part of it—the part where a random letter is generated—was not explained in that lesson.

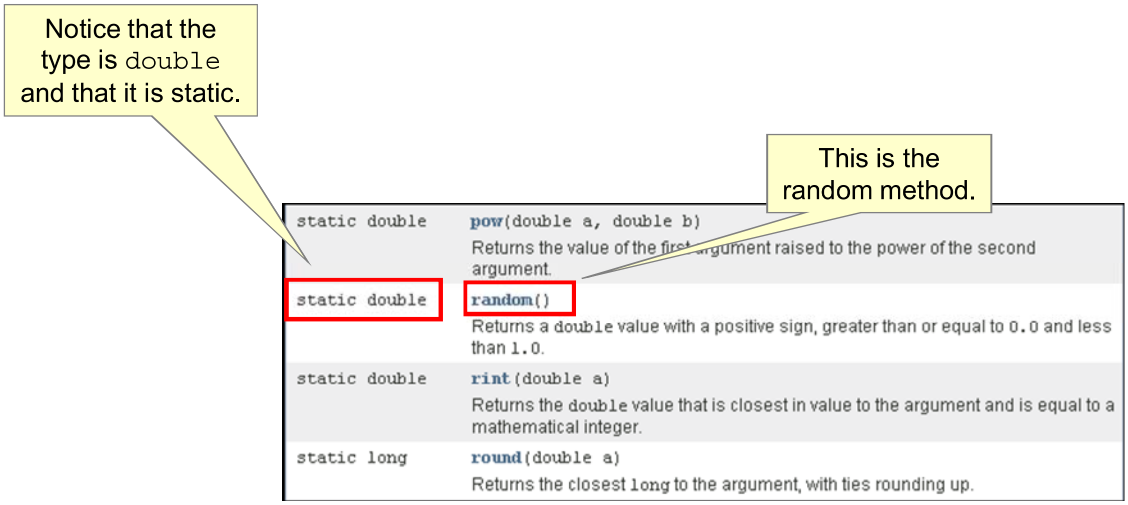
ASCII character values encode lowercase letters *a* to *z* from 97 to 122. By generating a number in that range and putting it into a `char`, you can use the concatenation operator to build a String as shown here.

Note: Java actually uses Unicode, not ASCII, but the first 128 characters in Unicode and ASCII are the same.

In the next slide, you look a little closer at the `Math.random()` method and what kind of method it is.

Oracle Internal & Oracle Academy Use Only

Static Methods in `Math`



Notice that the type is `double` and that it is static.

This is the random method.

static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	**random**() Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
static double	**rint**(double a) Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.
static long	**round**(double a) Returns the closest `long` to the argument, with ties rounding up.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The screenshot shows a small selection of methods from the `Math` class. The method in focus here is `random()`. It returns a double between 0 and 1. So to generate a double between 0 and 10, simply multiply by 10:

```
Math.random * 10
```

Or, to generate a double between 1 and 10, multiply by 9 and add 1.

Often you will want an integer rather than a double, so all you need to do is cast to `int` or, in the case of the example on the previous page, to `char`.

Notice that the method is static, as indeed are all the methods in `Math`. This means that `Math` does not need to be instantiated to call any of its methods (in fact, `Math` cannot be instantiated).

You can call the static methods of a class with the following syntax:

```
<classname>.<method_name>
```

Oracle Internal & Oracle Academy Use Only

Creating `static` Methods and Variables

Methods and nonlocal variables can be static.

- They belong to the class and not to the object.
- They are declared using the `static` keyword:

  ```
  static Properties getProperties()
  ```

- To invoke `static` methods:

  ```
  Classname.method();
  ```

- To access `static` variables in another class:

  ```
  Classname.attribute_name;
  ```

- To access `static` variables in the same class:

  ```
  attribute_name;
  ```

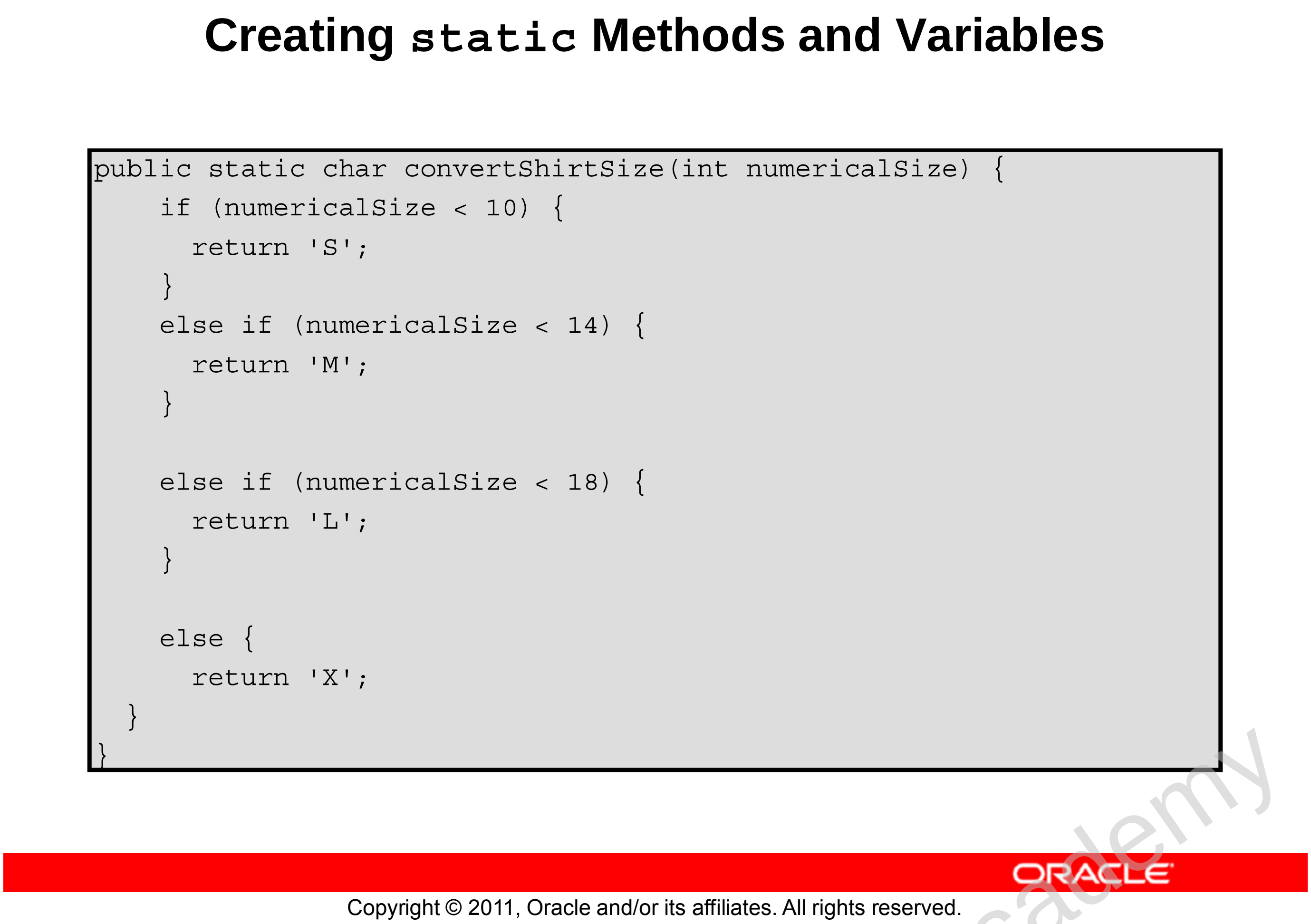
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

So far, you have learned how to access methods and variables by creating an object of the class that the method or variable belongs to, and invoking the method or accessing the variable (if it is a public variable). Methods and variables that are unique to an instance are called *instance methods* and *instance variables*.

You have also been using methods that do not require object instantiation, such as the `main` method. These are called *class methods* or *static methods*; you can invoke them without creating an object first.

Similarly, the Java programming language allows you to create static variables or class variables, which you can use without creating an object.

Oracle Internal & Oracle Academy Use Only



Creating `static` Methods and Variables

```
public static char convertShirtSize(int numericalSize) {
    if (numericalSize < 10) {
      return 'S';
    }
    else if (numericalSize < 14) {
      return 'M';
    }

    else if (numericalSize < 18) {
      return 'L';
    }

    else {
      return 'X';
  }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of a method that could be added to the `Shirt` class to convert numerical shirt sizes to sizes such as small, medium, and large. This method is a static method because:

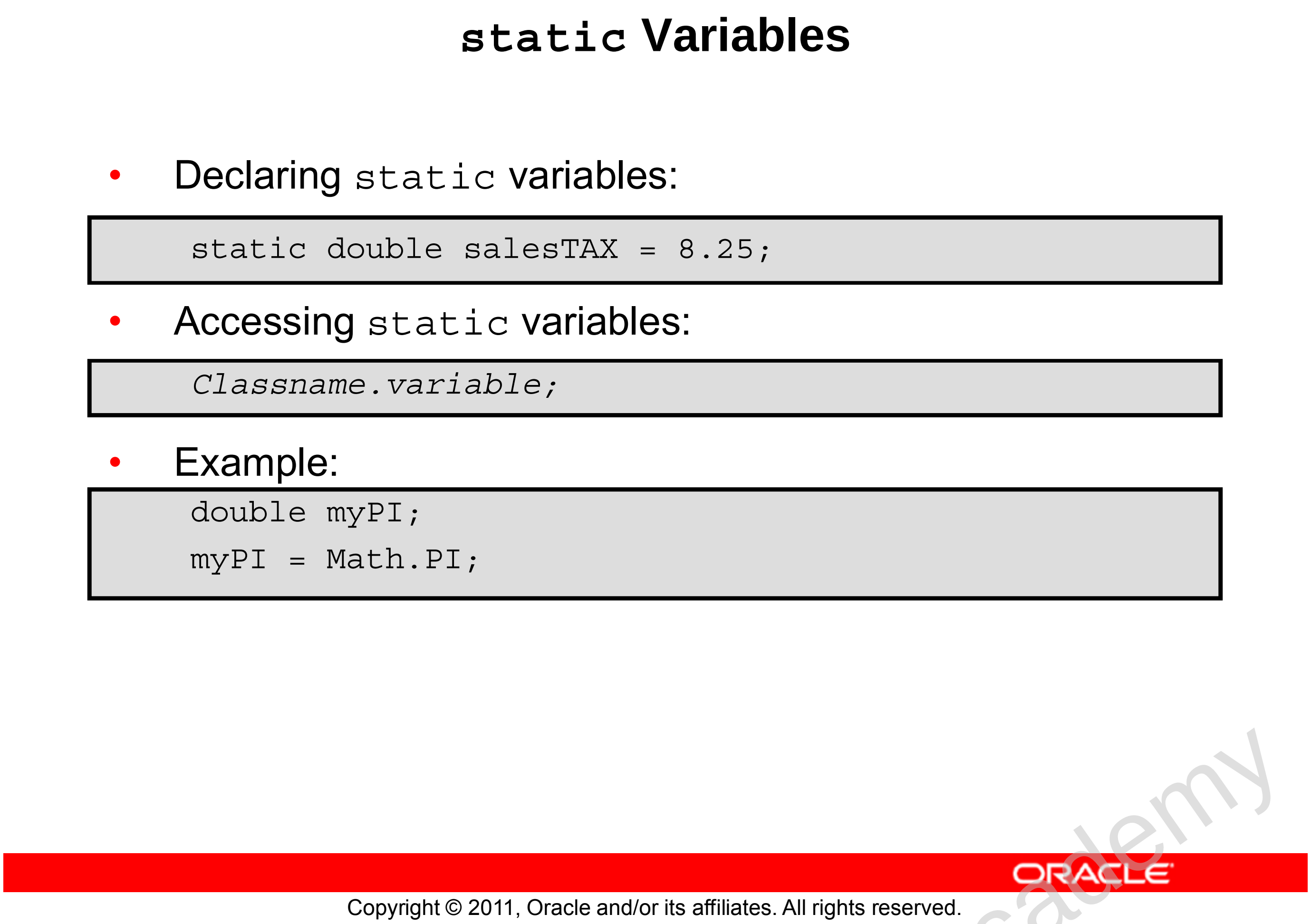
- It does not directly use any attributes of the `Shirt` class
- You might want to invoke the method even if you do not have a Shirt object

The `convertShirtSize` method accepts a numerical size, determines the corresponding character size (S, M, L, or X), and returns the character size.

For example, to access the `convertShirtSize()` static method of the `Shirt` class:

```
char size = Shirt.convertShirtSize(16);
```

Oracle Internal & Oracle Academy Use Only



`static` Variables

- Declaring `static` variables:

```
static double salesTAX = 8.25;
```

- Accessing `static` variables:

```
Classname.variable;
```

- Example:

```
double myPI;
myPI = Math.PI;
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You can also use the `static` keyword to declare a class variable. This means that there can be only one copy of the variable in memory associated with a class, rather than a copy for each object instance.

In the example in the slide, `salesTAX` is a static variable. You can access it from any method in any class by using the class name of its class. Assume that it is in a class called `TaxUtilities`. Then you could access it by using the code:

```
TaxUtilities.salesTAX
```

Or, if `TaxUtilities` has methods, those methods (static or instance) can access the variable by name without the class name:

```
salesTAX
```

Note that variables can have both the static and final modifier to indicate that there is only one copy of the variable and that the contents of the variable cannot be changed. The `PI` variable in the `Math` class is a static final variable.

Oracle Internal & Oracle Academy Use Only

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()`)

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only

Static Methods and Variables in the Java API

When to declare a `static` method or variable:

- 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.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

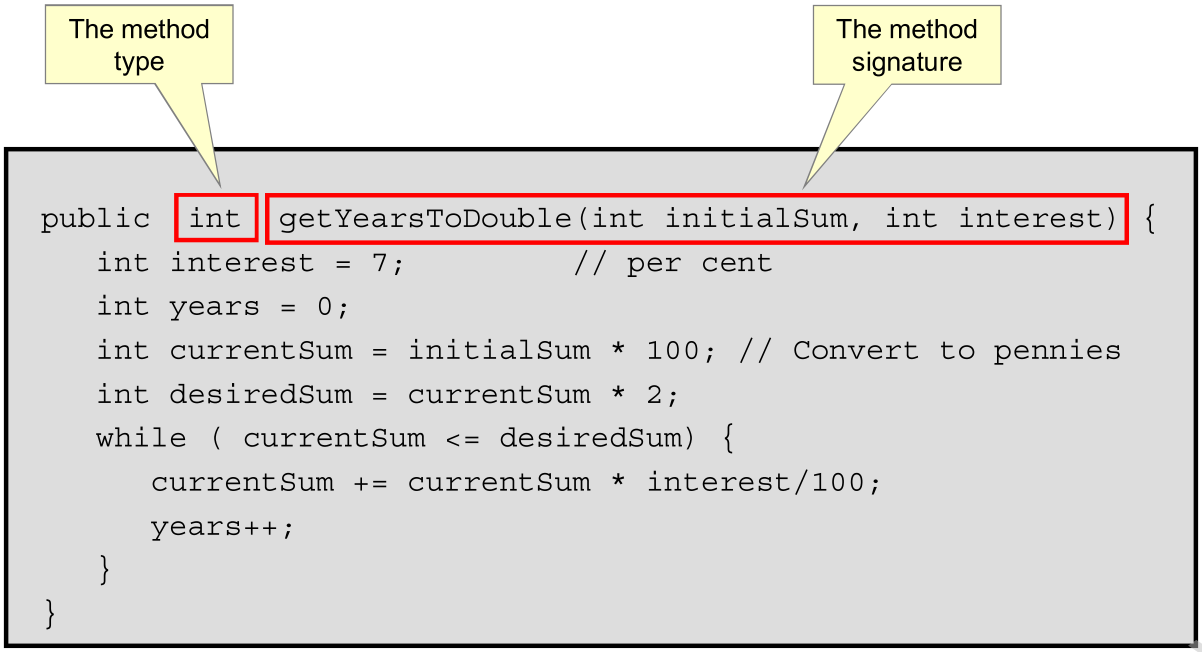
- Creating and invoking methods
- Static methods and variables
- Method overloading

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Method Signature



The method type

The method signature

```
public int getYearsToDouble(int initialSum, int interest) {
   int interest = 7;         // per cent
   int years = 0;
   int currentSum = initialSum * 100; // Convert to pennies
   int desiredSum = currentSum * 2;
   while ( currentSum <= desiredSum) {
      currentSum += currentSum * interest/100;
      years++;
   }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example shows some code from the lesson on loops, rewritten as a method that has two parameters (the initial sum of money and the interest rate) and returns the number of years required to double that initial sum.

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.

Oracle Internal & Oracle Academy Use Only

Method Overloading

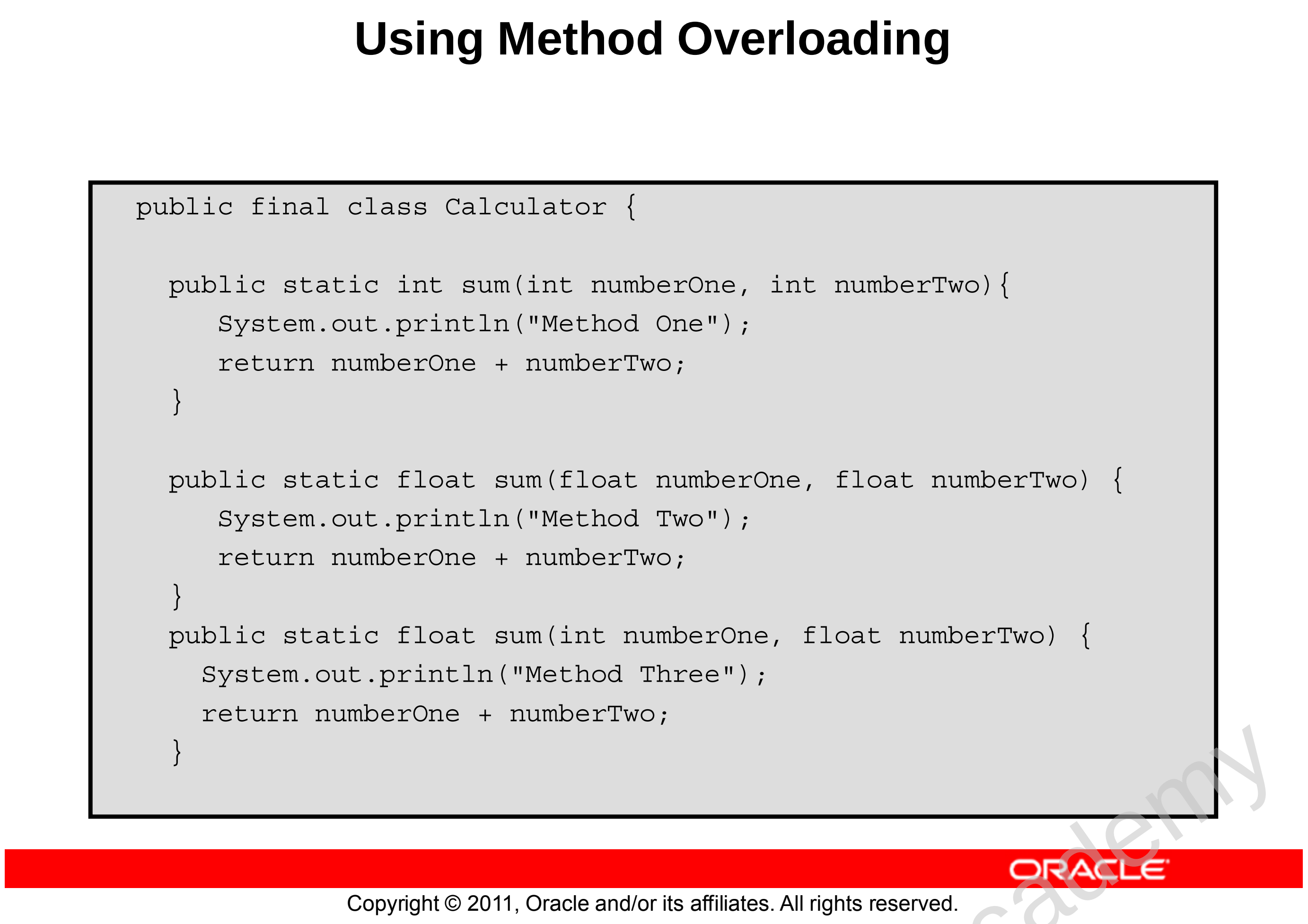
Overloaded methods:

- Have the same name
- Have different signatures
  - Different number and/or different type and/or different order of parameters
- May have different functionality or similar functionality
- Are widely used in the foundation classes

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only



Using Method Overloading

```
public final class Calculator {

  public static int sum(int numberOne, int numberTwo){
     System.out.println("Method One");
     return numberOne + numberTwo;
  }

  public static float sum(float numberOne, float numberTwo) {
     System.out.println("Method Two");
     return numberOne + numberTwo;
  }
  public static float sum(int numberOne, float numberTwo) {
    System.out.println("Method Three");
    return numberOne + numberTwo;
  }
```

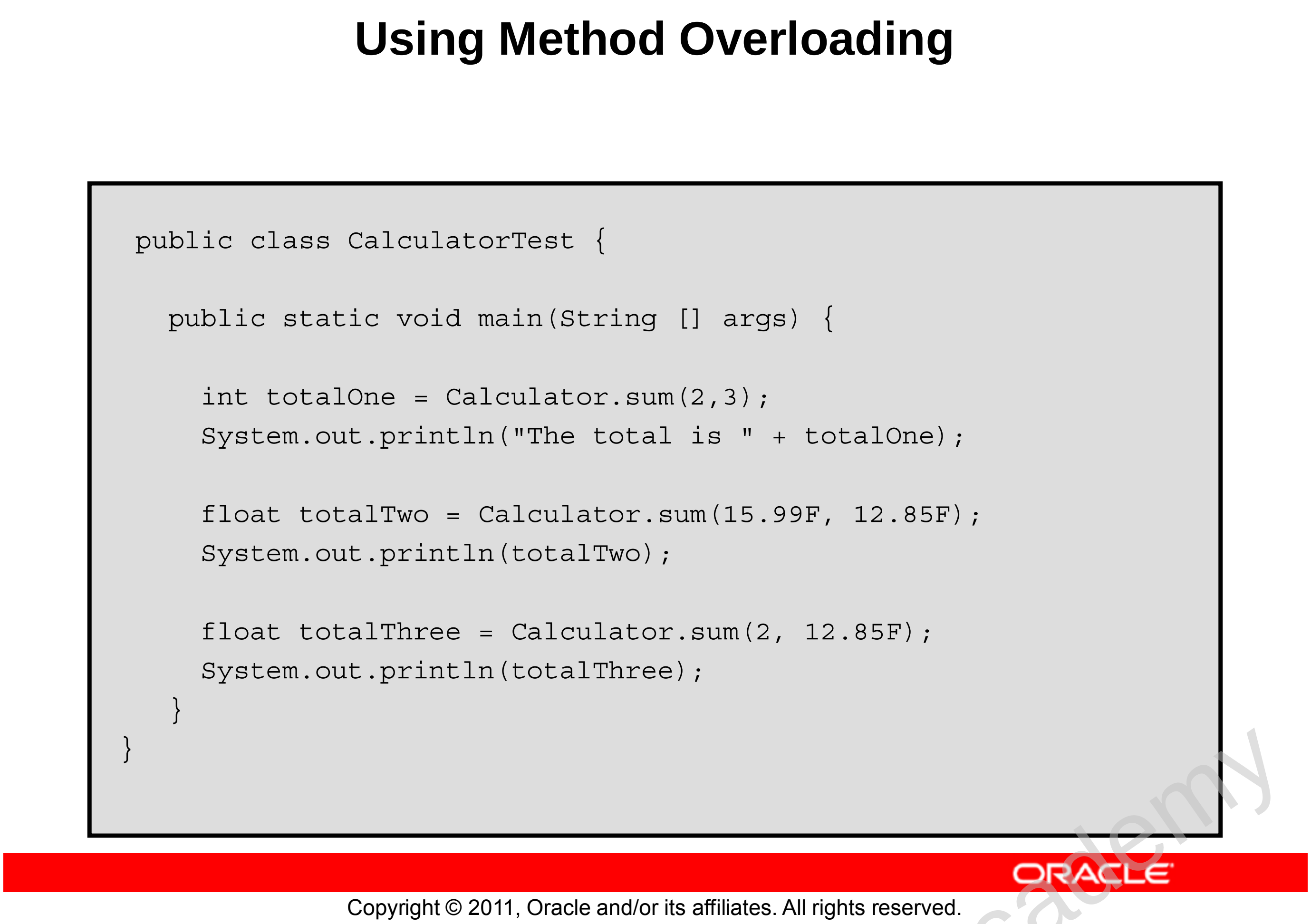
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

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`.

To invoke any of the `sum` methods, the compiler compares the method signature in your method invocation against the method signatures in a class.

Oracle Internal & Oracle Academy Use Only



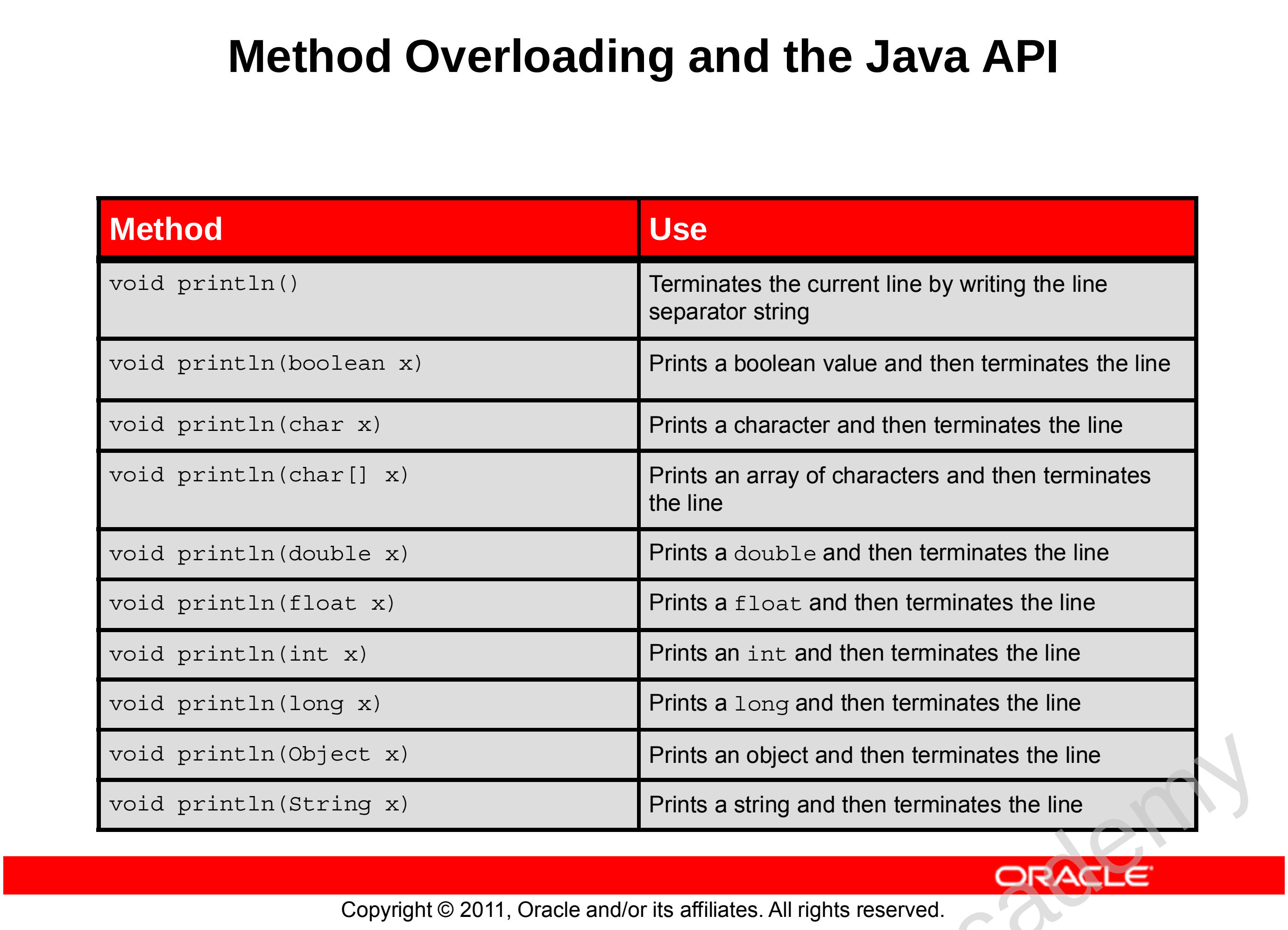
Using Method Overloading

```
 public class CalculatorTest {

   public static void main(String [] args) {

     int totalOne = Calculator.sum(2,3);
     System.out.println("The total is " + totalOne);

     float totalTwo = Calculator.sum(15.99F, 12.85F);
     System.out.println(totalTwo);

     float totalThree = Calculator.sum(2, 12.85F);
     System.out.println(totalThree);
   }
}
```

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only



Method Overloading and the Java API

Method	Use
`void println()`	Terminates the current line by writing the line separator string
`void println(boolean x)`	Prints a boolean value and then terminates the line
`void println(char x)`	Prints a character and then terminates the line
`void println(char[] x)`	Prints an array of characters and then terminates the line
`void println(double x)`	Prints a `double` and then terminates the line
`void println(float x)`	Prints a `float` and then terminates the line
`void println(int x)`	Prints an `int` and then terminates the line
`void println(long x)`	Prints a `long` and then terminates the line
`void println(Object x)`	Prints an object and then terminates the line
`void println(String x)`	Prints a string and then terminates the line

Copyright © 2011, 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 all the variations of the `println` method.

Oracle Internal & Oracle Academy Use Only

Quiz

Which method corresponds to the following method call?

```
myPerson.printValues(100, 147.7F, "lavender");
```

a. `public void printValues (int pantSize, float ageInYears)`

b. `public void printValues (pantSize, float ageInYears, favoriteColor)`

c. `public void printValues (int pantSize, float ageInYears, String favoriteColor)`

d. `public void printValues (float ageInYears, String favoriteColor, int pantSize)`

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Declare methods with arguments and return values
- Declare static methods and variables
- Create an overloaded method

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 10-1 Overview: Writing a Method with Arguments and Return Values

In this practice, you create a class to order more than one shirt, and then display the total order value of the shirts.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Challenge Practice 10-2 Overview: Writing a Class That Contains an Overloaded Method

In this practice, you write a Customer class with an overloaded method called `setCustomerInfo()`.

Note: This practice (10-2) is an optional Challenge practice.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

11

Using Encapsulation and Constructors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

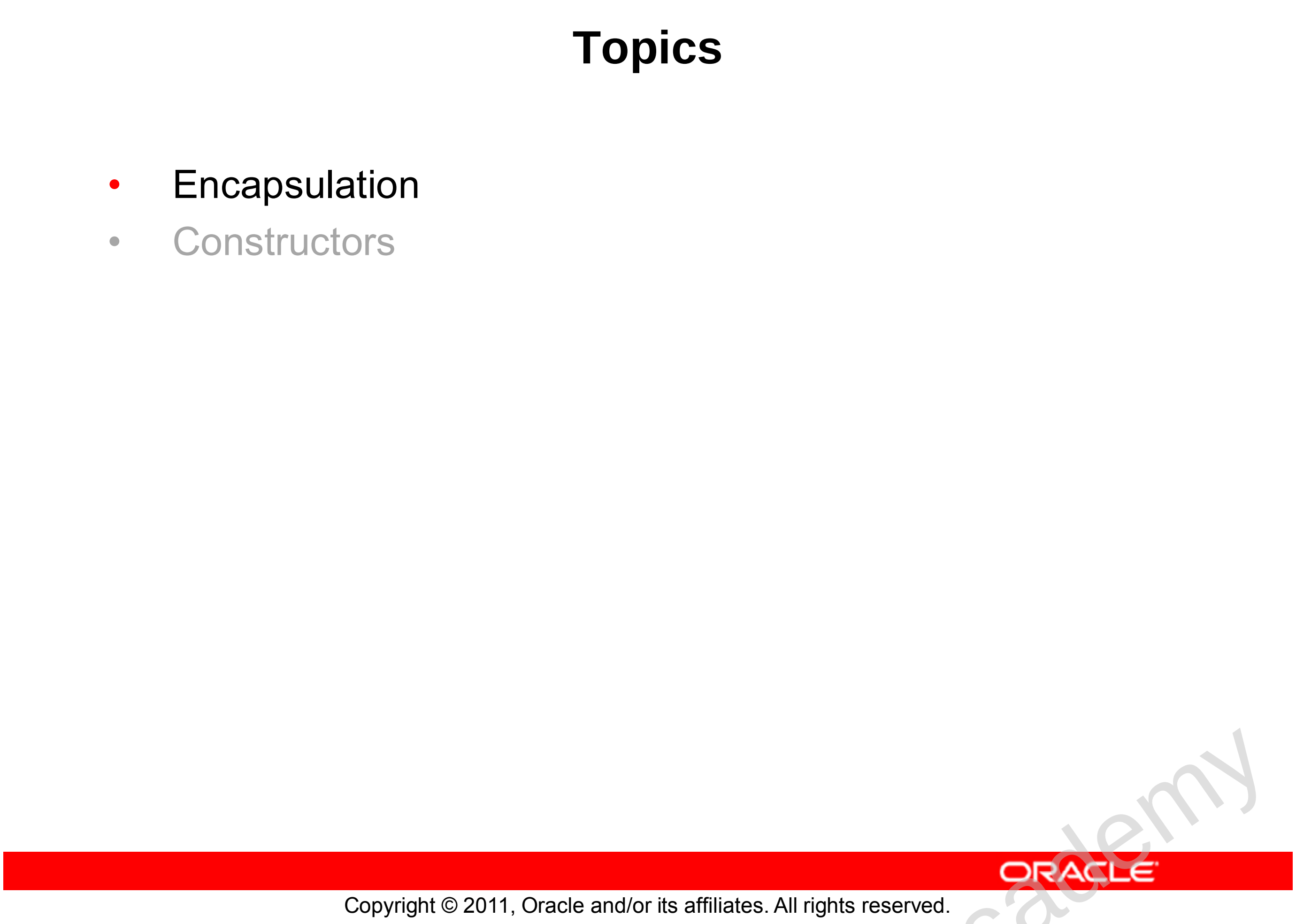
Objectives

After completing this lesson, you should be able to:

- Use access modifiers
- Describe the purpose of encapsulation
- Implement encapsulation in a class
- Create a constructor

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

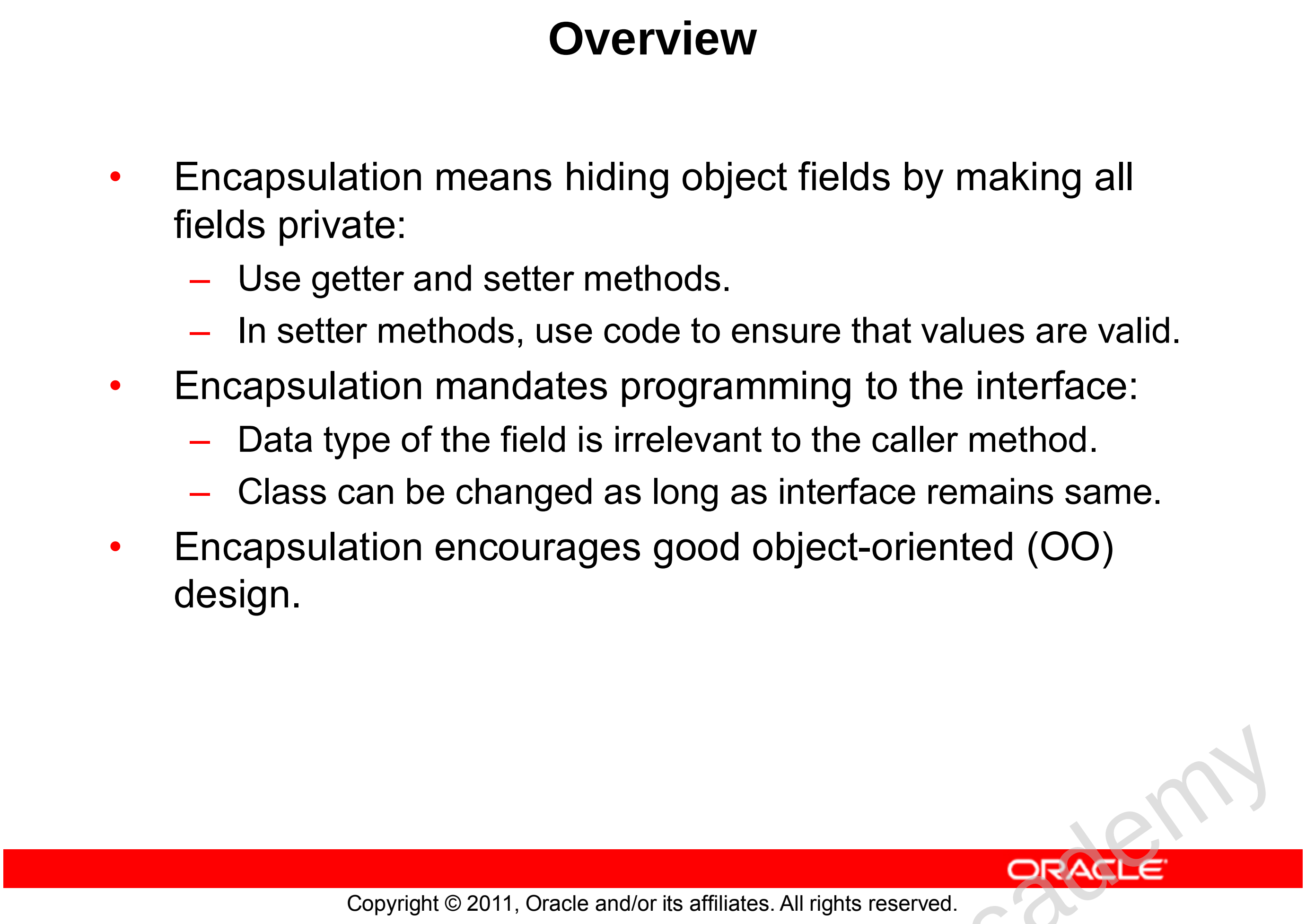


Topics

- Encapsulation
- Constructors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

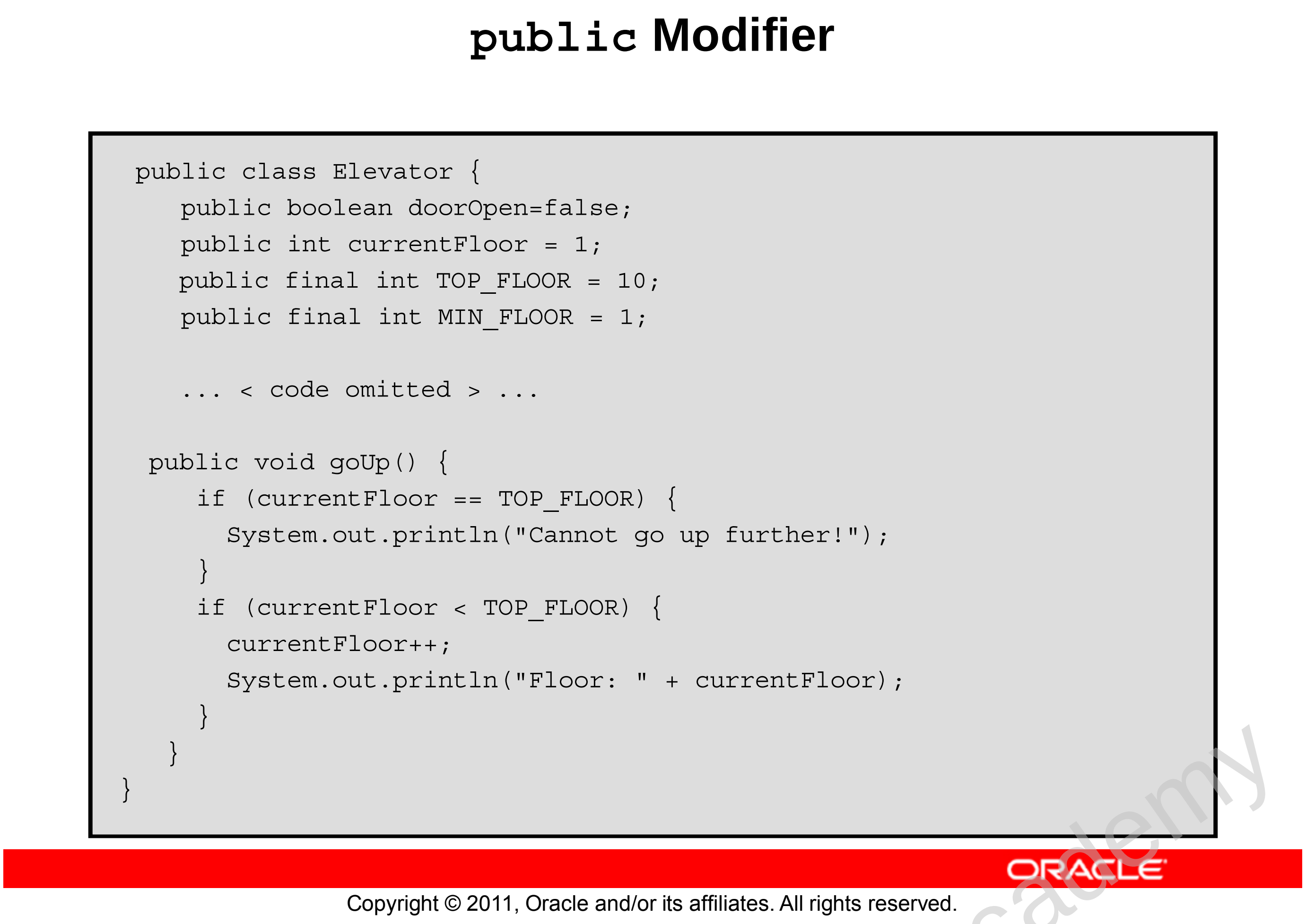


Overview

- Encapsulation means hiding object fields by making all fields private:
  - Use getter and setter methods.
  - In setter methods, use code to ensure that values are valid.
- Encapsulation mandates programming to the interface:
  - Data type of the field is irrelevant to the caller method.
  - Class can be changed as long as interface remains same.
- Encapsulation encourages good object-oriented (OO) design.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



public Modifier

```
public class Elevator {
   public boolean doorOpen=false;
   public int currentFloor = 1;
   public final int TOP_FLOOR = 10;
   public final int MIN_FLOOR = 1;

   ... < code omitted > ...

 public void goUp() {
    if (currentFloor == TOP_FLOOR) {
      System.out.println("Cannot go up further!");
    }
    if (currentFloor < TOP_FLOOR) {
      currentFloor++;
      System.out.println("Floor: " + currentFloor);
    }
  }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows the goUp() method and the currentFloor field. It is the corresponding method to the goDown() method previously discussed, and prevents the elevator from trying to go above the top floor.

But the code shown here has a problem. The goUp() method can be circumvented; there is nothing to stop the currentFloor field from being modified directly.

Oracle Internal & Oracle Academy Use Only

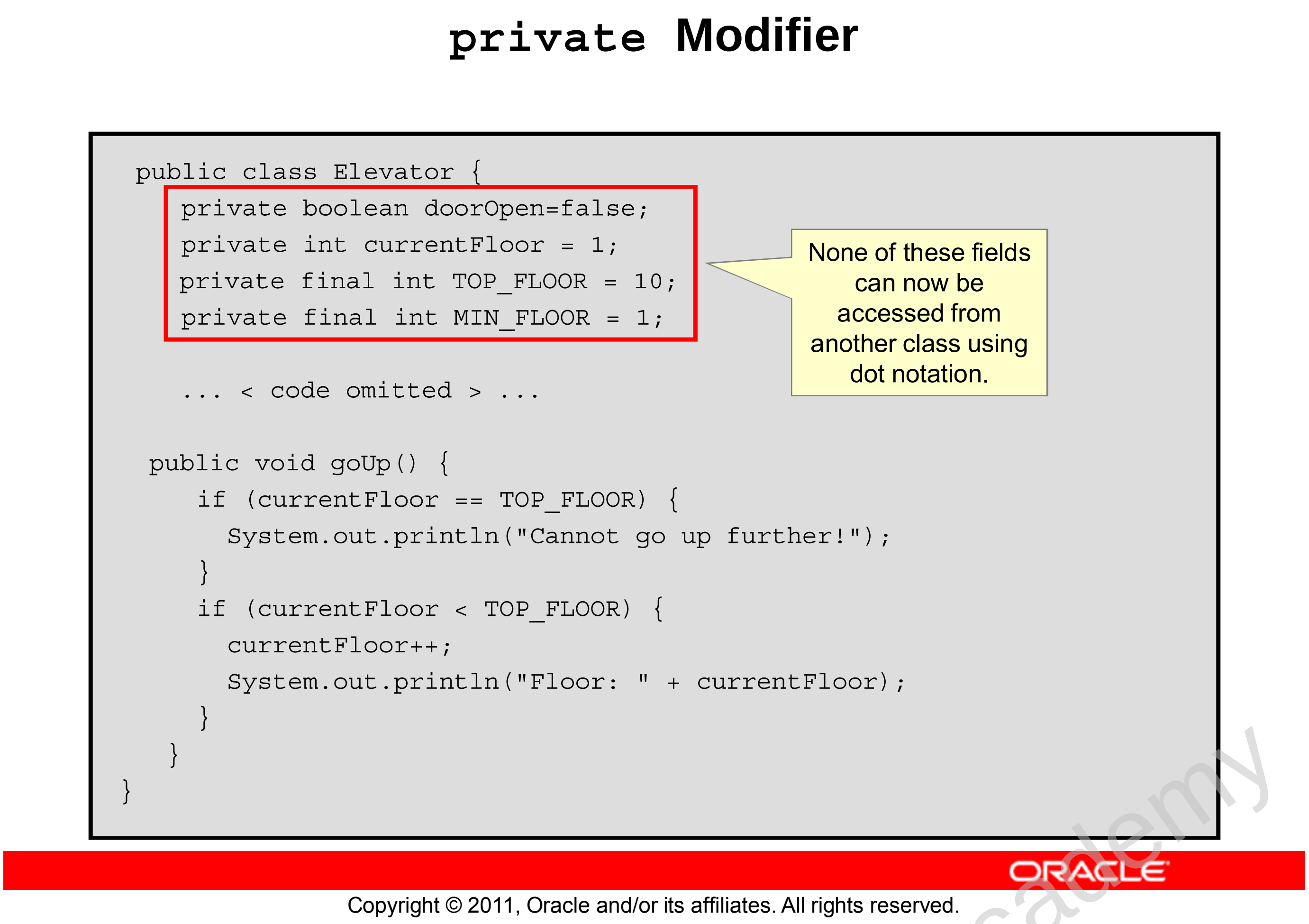
Dangers of Accessing a `public` Field

```
Elevator theElevator = new Elevator();

theElevator.currentFloor = 15;
```

Could cause a problem!

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



`private` Modifier

```
public class Elevator {
   private boolean doorOpen=false;
   private int currentFloor = 1;
   private final int TOP_FLOOR = 10;
   private final int MIN_FLOOR = 1;

   ... < code omitted > ...

  public void goUp() {
     if (currentFloor == TOP_FLOOR) {
       System.out.println("Cannot go up further!");
     }
     if (currentFloor < TOP_FLOOR) {
       currentFloor++;
       System.out.println("Floor: " + currentFloor);
     }
   }
}
```

None of these fields can now be accessed from another class using dot notation.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the example shown, the fields have all been made private. Now they cannot be accessed from a caller method that is outside this class. So any calling method that wants to control the floor that the elevator will go to must do so through its public methods.

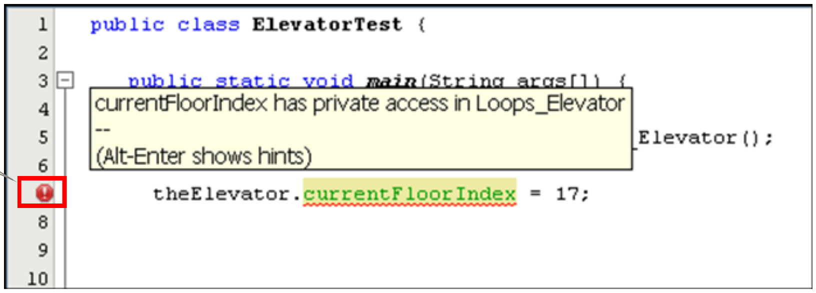
Oracle Internal & Oracle Academy Use Only

Trying to Access a `private` Field

```
Elevator theElevator = new Elevator();

theElevator.currentFloor = 15;
```

← not permitted

NetBeans will show an error. You can get an explanation if you place your cursor here.

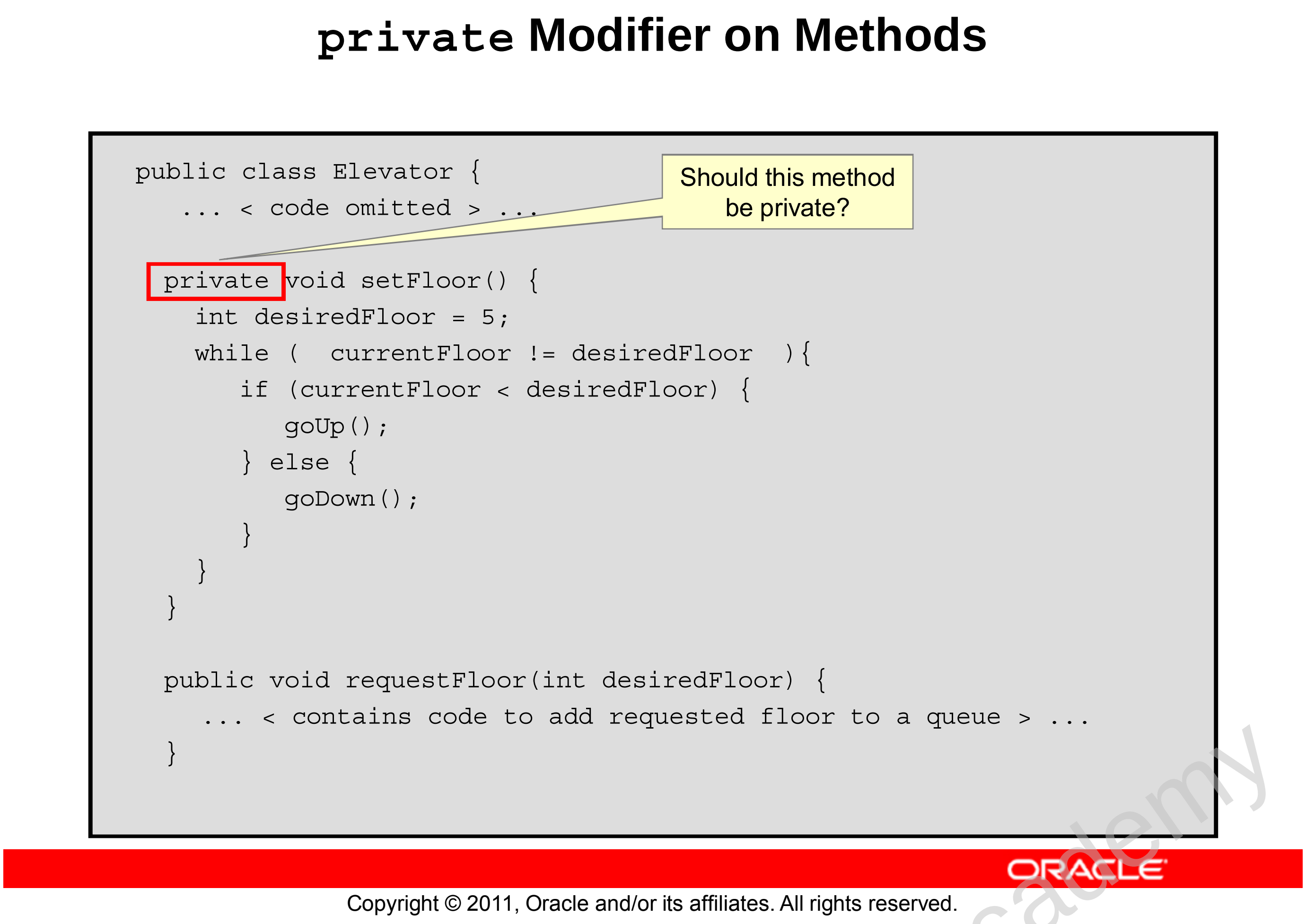


```
public class ElevatorTest {

    public static void main(String args[]) {
                                                    Elevator();

      theElevator.currentFloorIndex = 17;
```

currentFloorIndex has private access in Loops_Elevator
--
(Alt-Enter shows hints)

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



private Modifier on Methods

```
public class Elevator {
   ... < code omitted > ...

  private void setFloor() {
    int desiredFloor = 5;
    while (  currentFloor != desiredFloor  ){
       if (currentFloor < desiredFloor) {
          goUp();
       } else {
          goDown();
       }
    }
  }

  public void requestFloor(int desiredFloor) {
     ... < contains code to add requested floor to a queue > ...
  }
```

Should this method be private?

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Remember the `setFloor()` method? Just like fields, methods are declared with a modifier. Can you think of a reason why this method might be best declared with a private modifier?

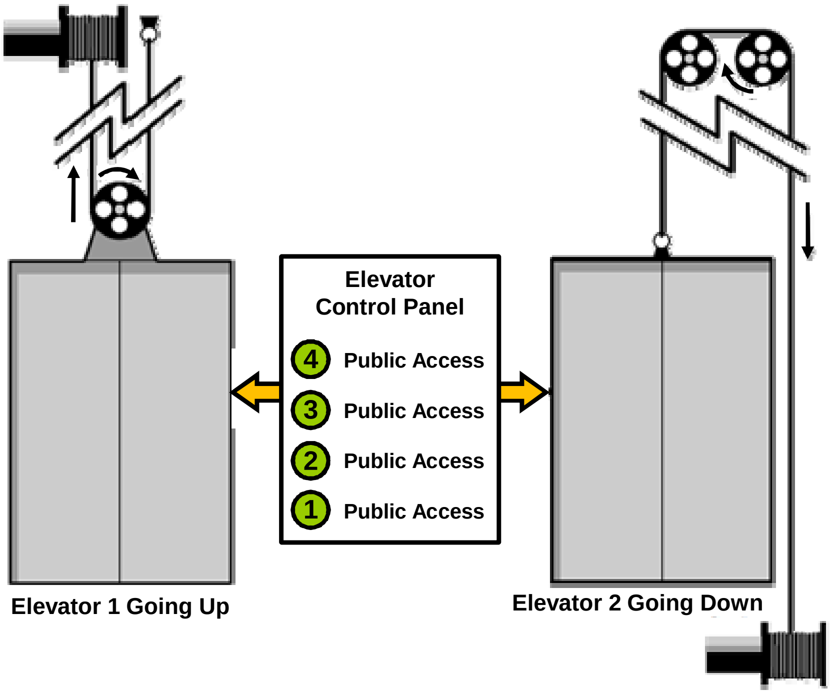
Well, if the elevator works like most elevators do, the controls operated by the general public (either the button to call an elevator, or the button to request a floor) do not directly affect the elevator.

Instead, a user presses a button—for example, a request for an elevator to go to the fifth floor. The elevator does not respond immediately to the request, but puts the request in a queue and then eventually, perhaps after bringing users already on the elevator down to the first floor, goes to the fifth floor.

It may be that the only public method needed is `requestFloor()`, at least for the software that controls the buttons used by the general public.

Oracle Internal & Oracle Academy Use Only

Interface and Implementation



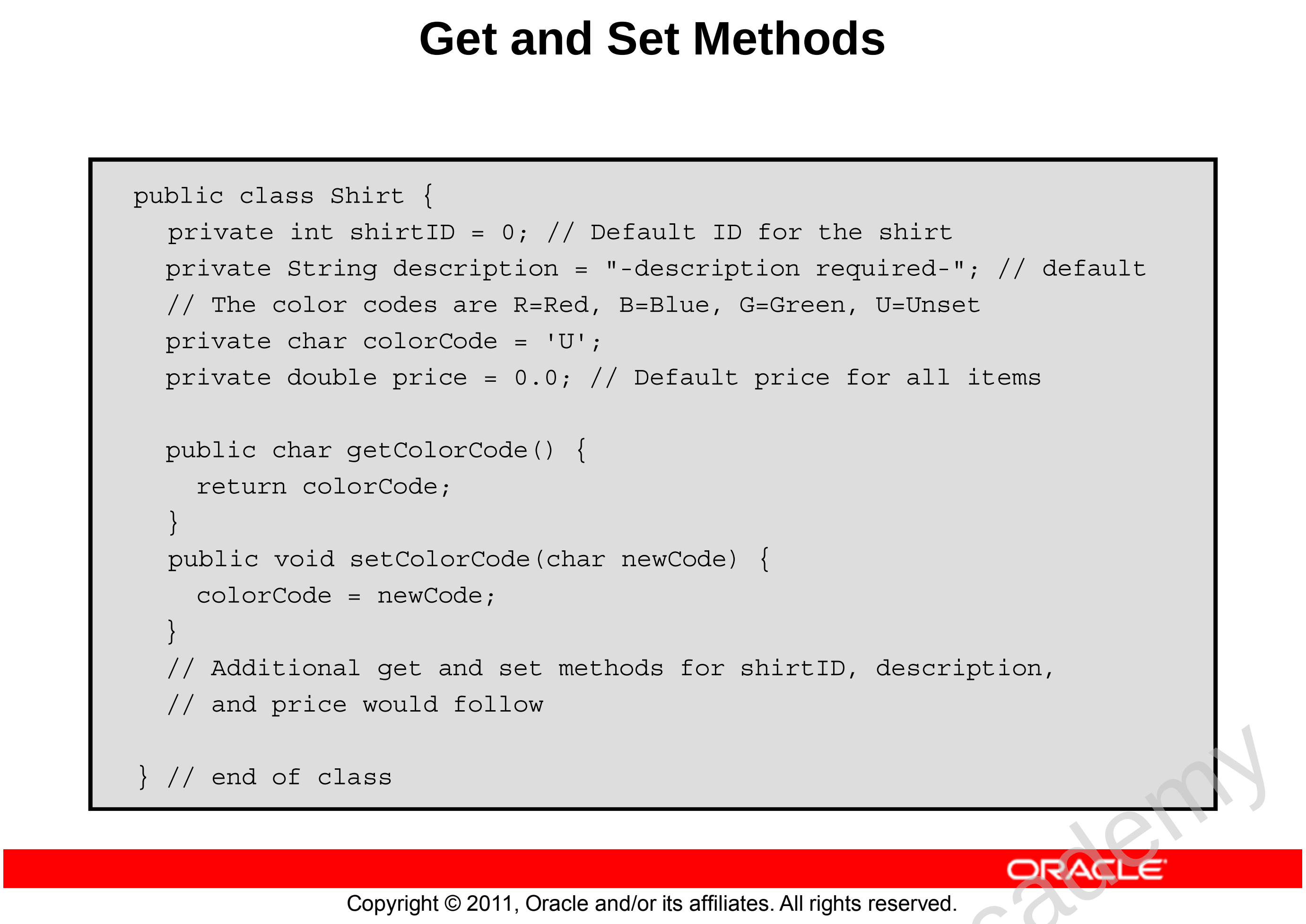
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

When classes are encapsulated, other objects interact with only a few parts (methods) of every other class.

In the example of the elevator, the control program that is triggered by the buttons can only call the `requestFloor()` method of `Elevator`. And, as long as `Elevator` implements this method, it does not matter exactly how it is implemented. The method could store requests in a binary array where setting an element to true indicates that there is a request on the floor with that index. Or an ArrayList could be used to store the numbers of the floors requested.

There might also be a `moveElevator()` method that is triggered by something, perhaps by the doors closing. Again, as long as this method `moveElevator()` is implemented, its implementation can be changed to change the way in which the elevator responds to requests coming in at the same time from different floors.

Oracle Internal & Oracle Academy Use Only



Get and Set Methods

```
public class Shirt {
  private int shirtID = 0; // Default ID for the shirt
  private String description = "-description required-"; // default
  // The color codes are R=Red, B=Blue, G=Green, U=Unset
  private char colorCode = 'U';
  private double price = 0.0; // Default price for all items

  public char getColorCode() {
    return colorCode;
  }
  public void setColorCode(char newCode) {
    colorCode = newCode;
  }
  // Additional get and set methods for shirtID, description,
  // and price would follow

} // end of class
```

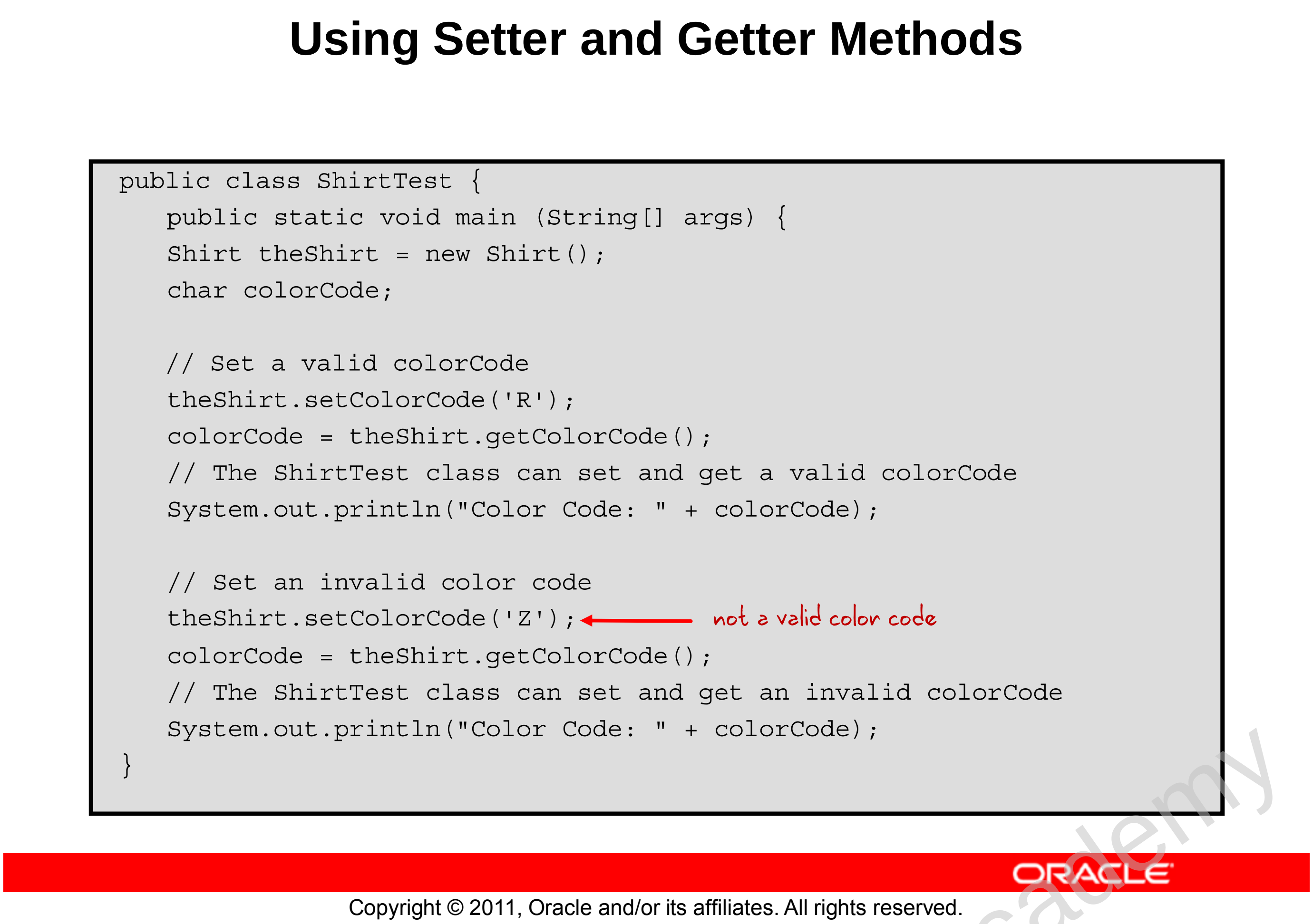
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

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, in the same way you wrote code to ensure that the `currentFloor` field of the elevator could not be set to an invalid value.

Oracle Internal & Oracle Academy Use Only



Using Setter and Getter Methods

```
public class ShirtTest {
   public static void main (String[] args) {
   Shirt theShirt = new Shirt();
   char colorCode;

   // Set a valid colorCode
   theShirt.setColorCode('R');
   colorCode = theShirt.getColorCode();
   // The ShirtTest class can set and get a valid colorCode
   System.out.println("Color Code: " + colorCode);

   // Set an invalid color code
   theShirt.setColorCode('Z');
   colorCode = theShirt.getColorCode();
   // The ShirtTest class can set and get an invalid colorCode
   System.out.println("Color Code: " + colorCode);
}
```

not a valid color code

Copyright © 2011, 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` via a setter method, `Shirt` can now be recoded without modifying any of the classes that depend on it.

Oracle Internal & Oracle Academy Use Only

Setter Method with Checking

```
 public void setColorCode(char newCode) {
    switch (newCode) {
       case 'R':
       case 'G':
       case 'B':
       colorCode = newCode;
       break;
       default:
       System.out.println("Invalid colorCode. Use R, G, or B");
    }
}
```

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only

Using Setter and Getter Methods

```
public class ShirtTest {
   public static void main (String[] args) {
   Shirt theShirt = new Shirt();
   System.out.println("Color Code: " + theShirt.getColorCode());

   // Try to set an invalid color code
   Shirt1.setColorCode('Z'); ←—— not a valid color code
   System.out.println("Color Code: " + theShirt.getColorCode());
}
```

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 © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Encapsulation: Summary

Encapsulation protects data:

- By making all fields private
  - Use getter and setter methods.
  - In setter methods, use code to check whether values are valid.
- By mandating programming to the interface
  - Data type of the field is irrelevant to the caller method.
  - Class can be changed as long as interface remains same.
- By encouraging good OO design

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

- Encapsulation
- Constructors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Initializing a `Shirt` Object

```
public class ShirtTest {
   public static void main (String[] args) {
   Shirt theShirt = new Shirt();

   // Set values for the Shirt
   theShirt.setColorCode('R');
   theShirt.setDescription("Outdoors shirt");
   theShirt.price(39.99);

}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Assuming 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*.

Oracle Internal & Oracle Academy Use Only

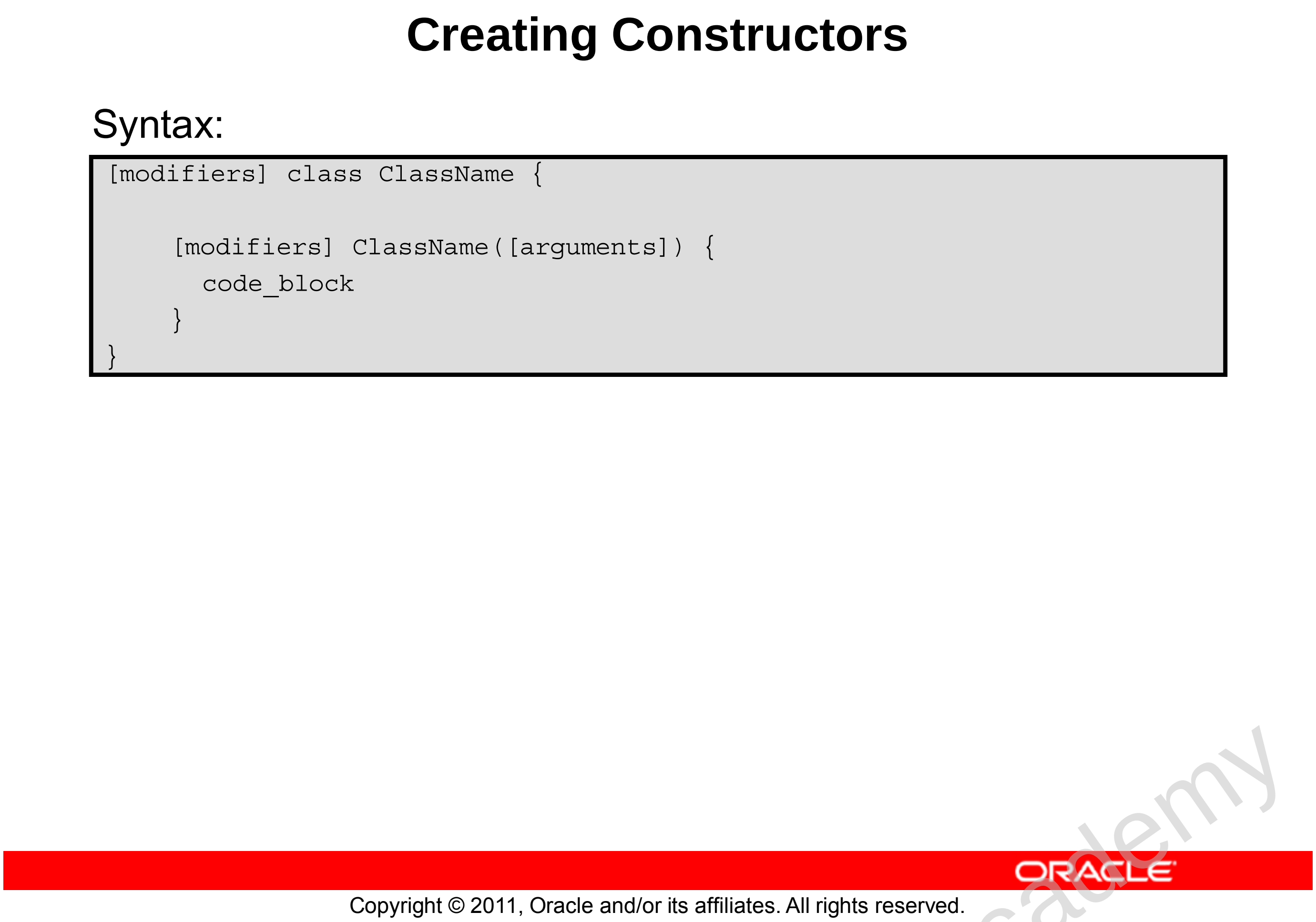
Constructors

- Constructors are method-like structures in a class:
  - They have the same name as the class.
  - They are usually used to initialize fields in an object.
  - They can receive arguments.
  - They can be overloaded.
- All classes have at least one constructor:
  - If there are no explicit constructors, the Java compiler supplies a default no-argument constructor.
  - If there are one or more explicit constructors, no default constructor will be supplied.

Copyright © 2011, 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.

Oracle Internal & Oracle Academy Use Only



Creating Constructors

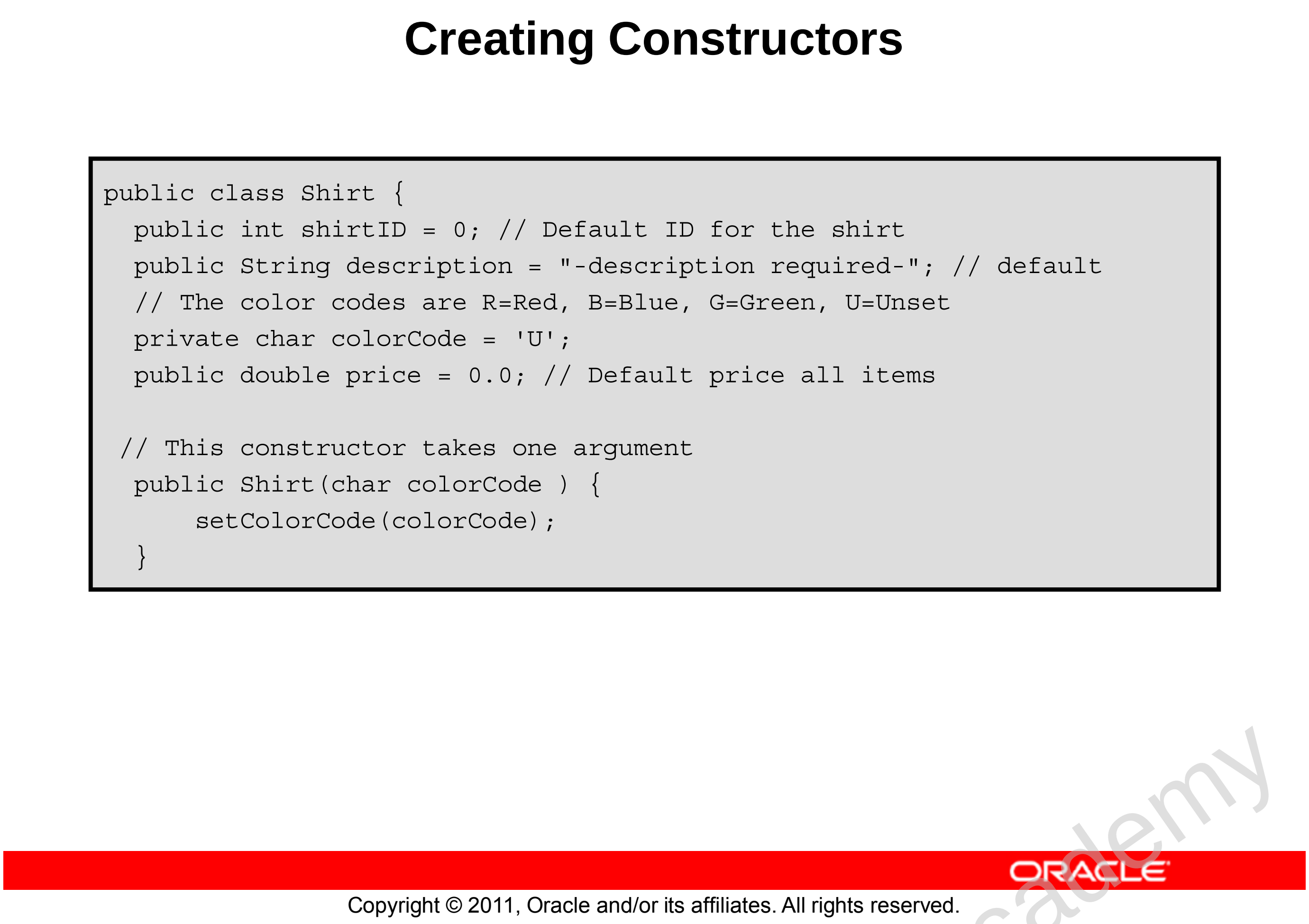
Syntax:

```
[modifiers] class ClassName {

    [modifiers] ClassName([arguments]) {
      code_block
    }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

- [modifiers] represent several unique Java technology keywords that can modify the way constructors are accessed. Modifiers are optional (indicated by the square brackets).
- ClassName is the name of the class and the name of the constructor method. The name of the constructor must be the same as the ClassName in the class declaration.
- [arguments] represents one or more optional arguments passed to the constructor.
- code_block represents one or more optional lines of code for the constructor.

Oracle Internal & Oracle Academy Use Only



Creating Constructors

```
public class Shirt {
  public int shirtID = 0; // Default ID for the shirt
  public String description = "-description required-"; // default
  // The color codes are R=Red, B=Blue, G=Green, U=Unset
  private char colorCode = 'U';
  public double price = 0.0; // Default price all items

 // This constructor takes one argument
  public Shirt(char colorCode ) {
      setColorCode(colorCode);
  }
```

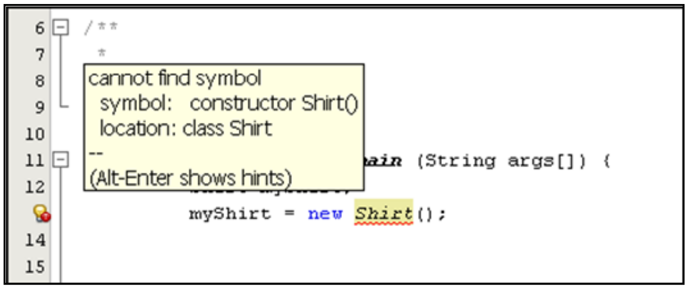
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Shirt example shown in the slide has a constructor that accepts a `char` value to initialize the color code for this object. Because `setColorCode()` ensures that an invalid code cannot be set, the constructor can just call this method.

Oracle Internal & Oracle Academy Use Only

Initializing a `Shirt` Object by Using a Constructor

```
public class ShirtTest {
    public static void main (String[] args) {
    Shirt theShirt = new Shirt('G');

    theShirt.display();
}
```



```
/**
 *
                   ain (String args[]) {
            myShirt = new Shirt();
```

cannot find symbol
  symbol: constructor Shirt()
  location: class Shirt
--
(Alt-Enter shows hints)

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

As you would expect, passing a valid color code to the Shirt constructor creates a new Shirt object, and calling `display()` results in the following output:

```
Item ID: 0
Item description:-description required-
Color Code: G
Item price: 0.0
```

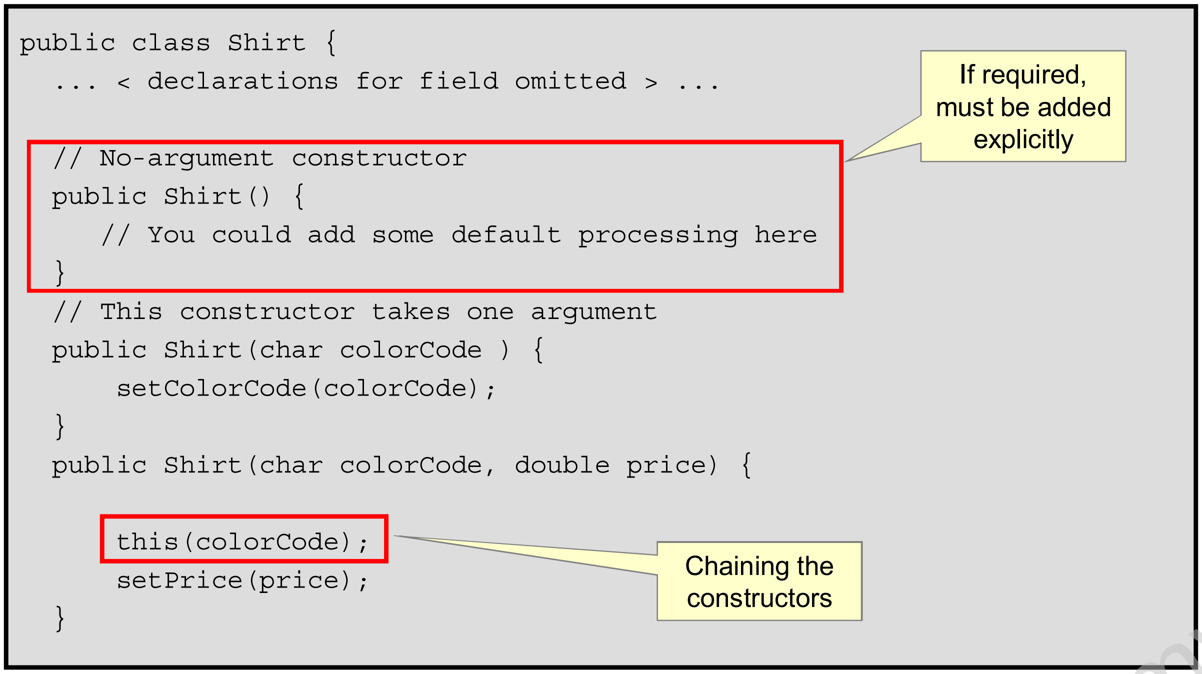
However, look at the message you get in NetBeans if you try to call the Shirt constructor with no arguments (as you have been doing earlier in the course).

The reason for the problem is that if there is no explicit constructor in a class, Java assumes that you want to be able to instantiate the class and gives you a default no-argument constructor. Otherwise, how could you instantiate the class?

But if you have one explicit constructor, Java assumes that you might want that to be the only constructor, and no longer provides a default no-argument implementation.

Oracle Internal & Oracle Academy Use Only

Multiple Constructors



```
public class Shirt {
  ... < declarations for field omitted > ...

  // No-argument constructor
  public Shirt() {
     // You could add some default processing here
  }
  // This constructor takes one argument
  public Shirt(char colorCode ) {
       setColorCode(colorCode);
  }
  public Shirt(char colorCode, double price) {

       this(colorCode);
       setPrice(price);
  }
```

If required, must be added explicitly

Chaining the constructors

Copyright © 2011, 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 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.

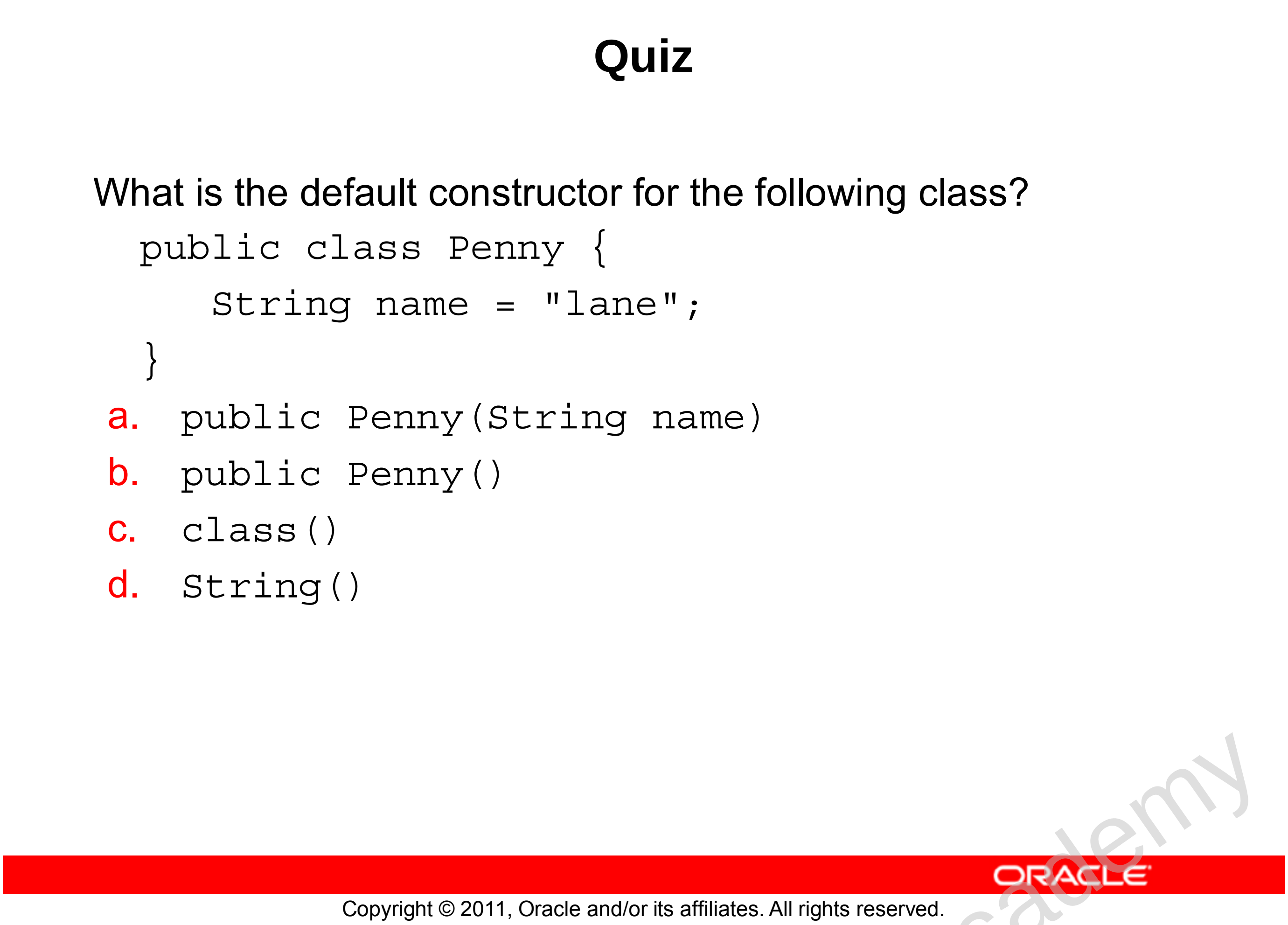
You can chain the constructors by calling the second constructor in the first line of the third constructor using the following syntax:

```
this(argument);
```

`this` is a special keyword that is a reference to the current object.

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.

Oracle Internal & Oracle Academy Use Only



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()`

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: b

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Use access modifiers
- Describe the purpose of encapsulation
- Implement encapsulation in a class
- Create a constructor

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 11-1 Overview: Implementing Encapsulation in a Class

In this practice, you create a class containing private attributes and try to access them in another class. During this practice, you:

- Implement encapsulation in a class
- Access encapsulated attributes of a class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Challenge Practice 11-2 Overview: Adding Validation to the `DateThree` Class

In this practice, you add a `setDate()` method to the `DateThree` class that performs validation on the date part values that are passed into the method.

Note: This practice (11-2) is an optional Challenge practice.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 11-3 Overview: Creating Constructors to Initialize Objects

In this practice, you create a class and use constructors to initialize objects.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

12

Using Advanced Object-Oriented Concepts

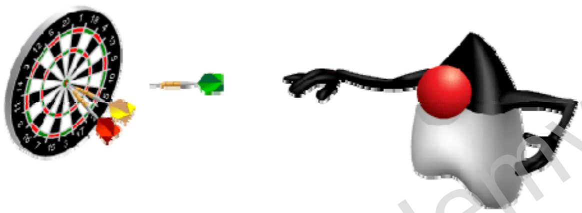
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Describe inheritance
- Test superclass and subclass relations
- Describe polymorphism
- Create a subclass
- Use abstract classes and interfaces



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

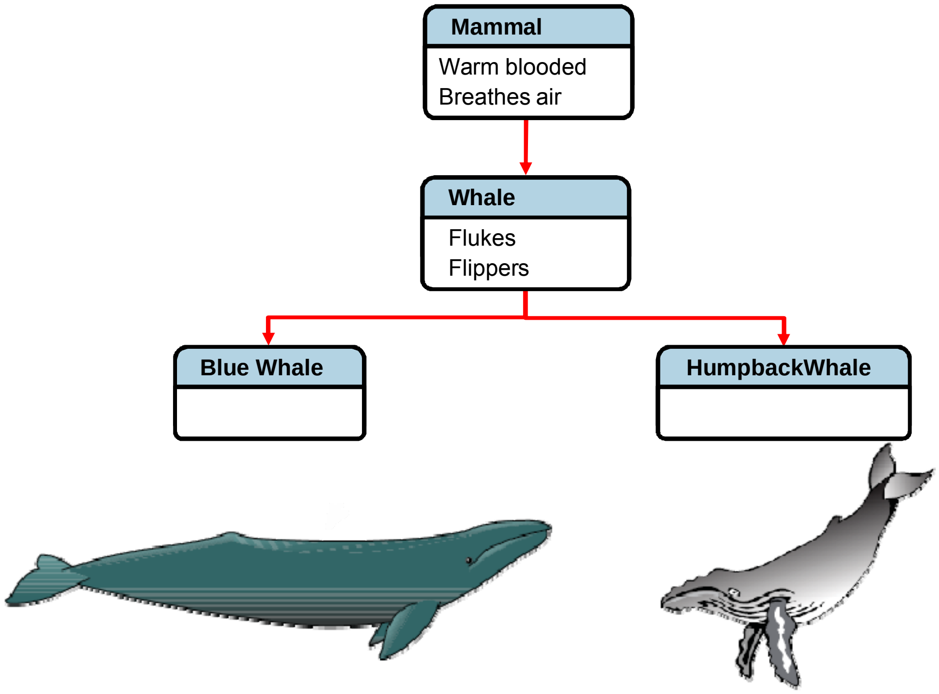
Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Polymorphism and overriding methods
- Interfaces
- The Object class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Class Hierarchies



Mammal

Warm blooded
Breathes air

Whale

Flukes
Flippers

Blue Whale

HumpbackWhale

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Inheritance results in a class hierarchy of Java technology classes similar to the taxonomies found in biology, such as “Blue Whale is a subclass of Whale.”

The diagram in the slide illustrates a hierarchy for whales. “Warm blooded” is an attribute of the Mammal superclass. The phrase “breathes air” represents some operation that is also a part of the Mammal superclass. Flukes and flippers are attributes that are specific to the Whale class, which is a subclass of the Mammal class.

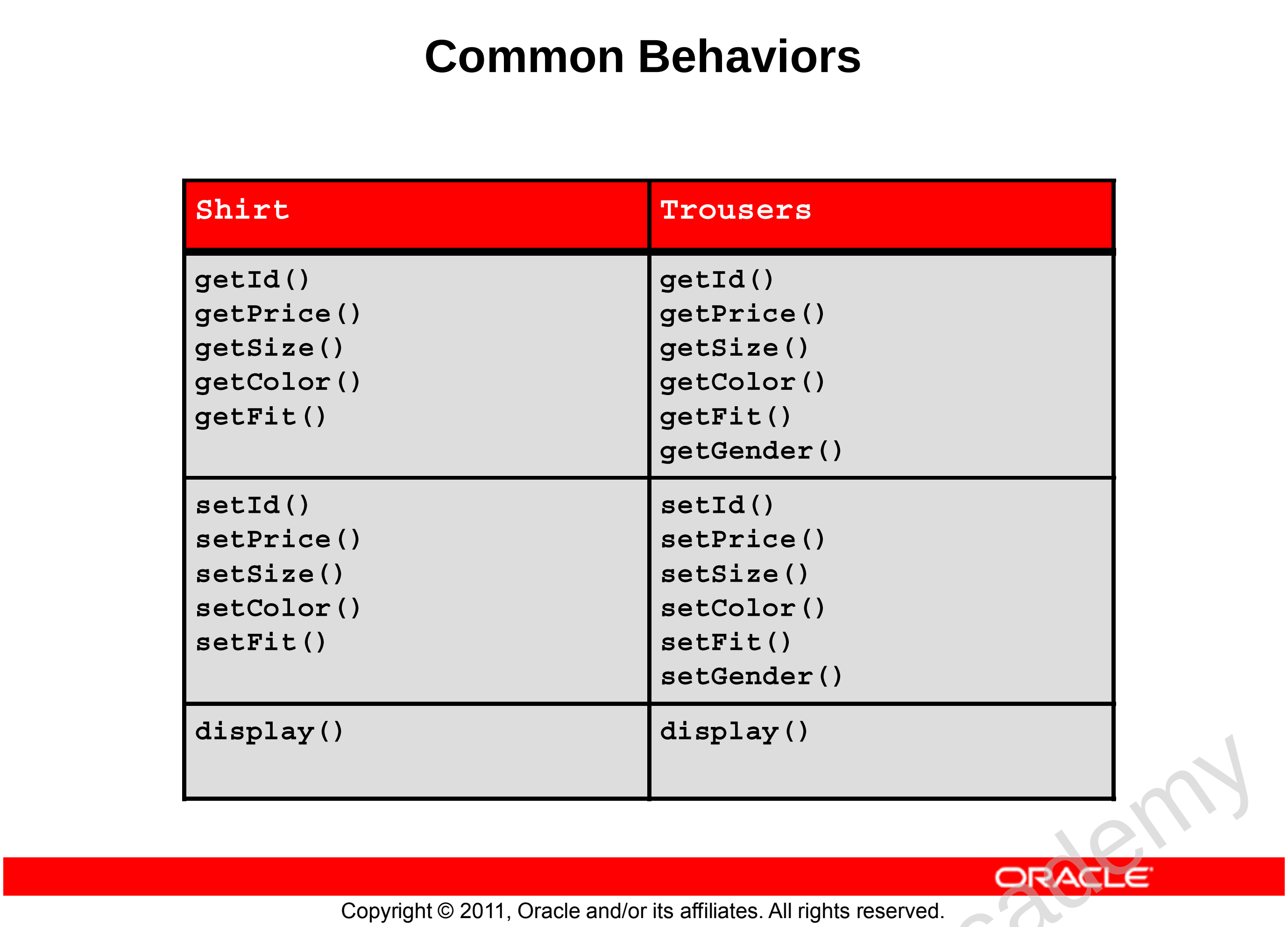
Oracle Internal & Oracle Academy Use Only

Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Polymorphism and overriding methods
- Interfaces
- The Object class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Common Behaviors

Shirt	Trousers

getId()
getPrice()
getSize()
getColor()
getFit()
	getId()
	getPrice()
	getSize()
	getColor()
	getFit()
	getGender()

setId()
setPrice()
setSize()
setColor()
setFit()
	setId()
	setPrice()
	setSize()
	setColor()
	setFit()
	setGender()

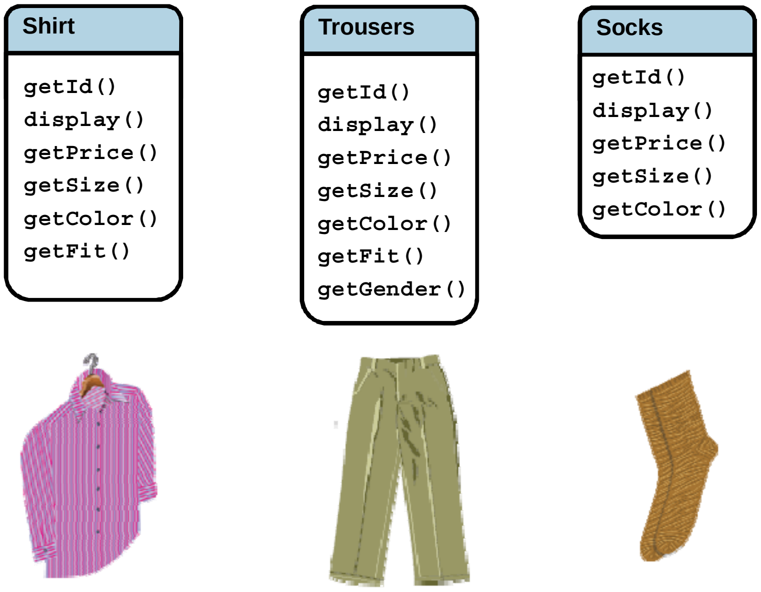
display()	display()

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The table in the slide shows a set of behaviors for the Shirt class and for a new class: Trousers. The classes are shown fully encapsulated so that all field values are accessible only through setter and getter methods. Notice how both classes use many of the same methods; this may result in code duplication, making maintenance and further expansion more difficult and error prone.

Oracle Internal & Oracle Academy Use Only

Code Duplication



Shirt

```
getId()
display()
getPrice()
getSize()
getColor()
getFit()
```

Trousers

```
getId()
display()
getPrice()
getSize()
getColor()
getFit()
getGender()
```

Socks

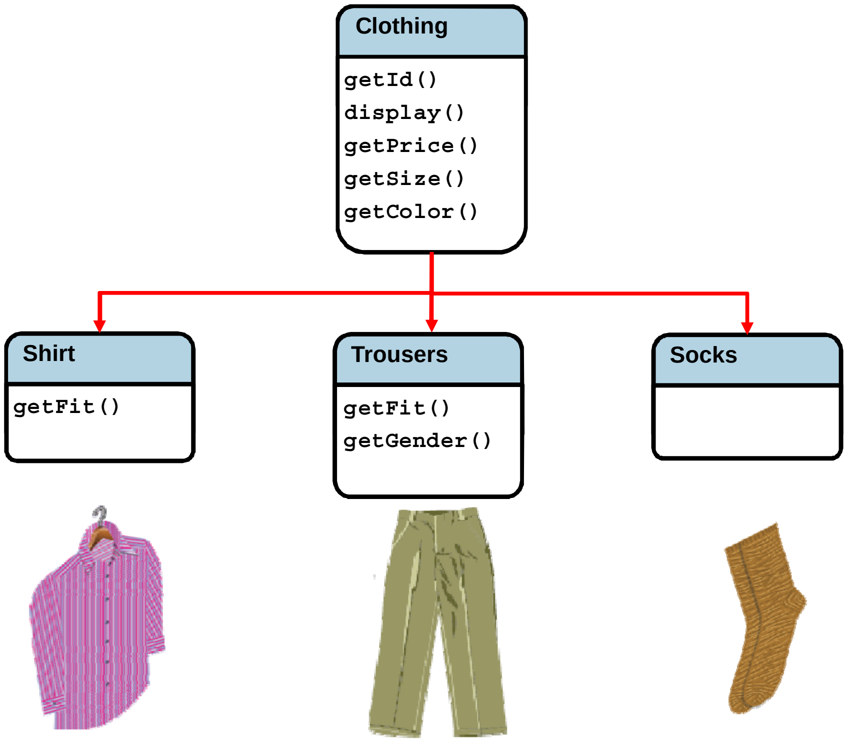
```
getId()
display()
getPrice()
getSize()
getColor()
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

If Duke's Choice decides to add a third item, socks, as well as trousers and shirts, you may find even greater code duplication. The diagram in the slide shows only the getter methods for accessing the properties of the new objects.

Oracle Internal & Oracle Academy Use Only

Inheritance



Clothing

getId()
display()
getPrice()
getSize()
getColor()

Shirt

getFit()

Trousers

getFit()
getGender()

Socks

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You can eliminate code duplication in the classes by implementing inheritance. Inheritance enables programmers to put common members (fields and methods) in one class (the superclass) and have other classes (the subclasses) inherit these common members from this new class.

An object instantiated from a subclass behaves as if the fields and methods of the subclass were in the object. For example, the Trousers class can be instantiated and have the `display()` method called even though the Trousers class does not contain a `display()` method; it is inherited from the Clothing class.

Oracle Internal & Oracle Academy Use Only

Overriding Superclass Methods

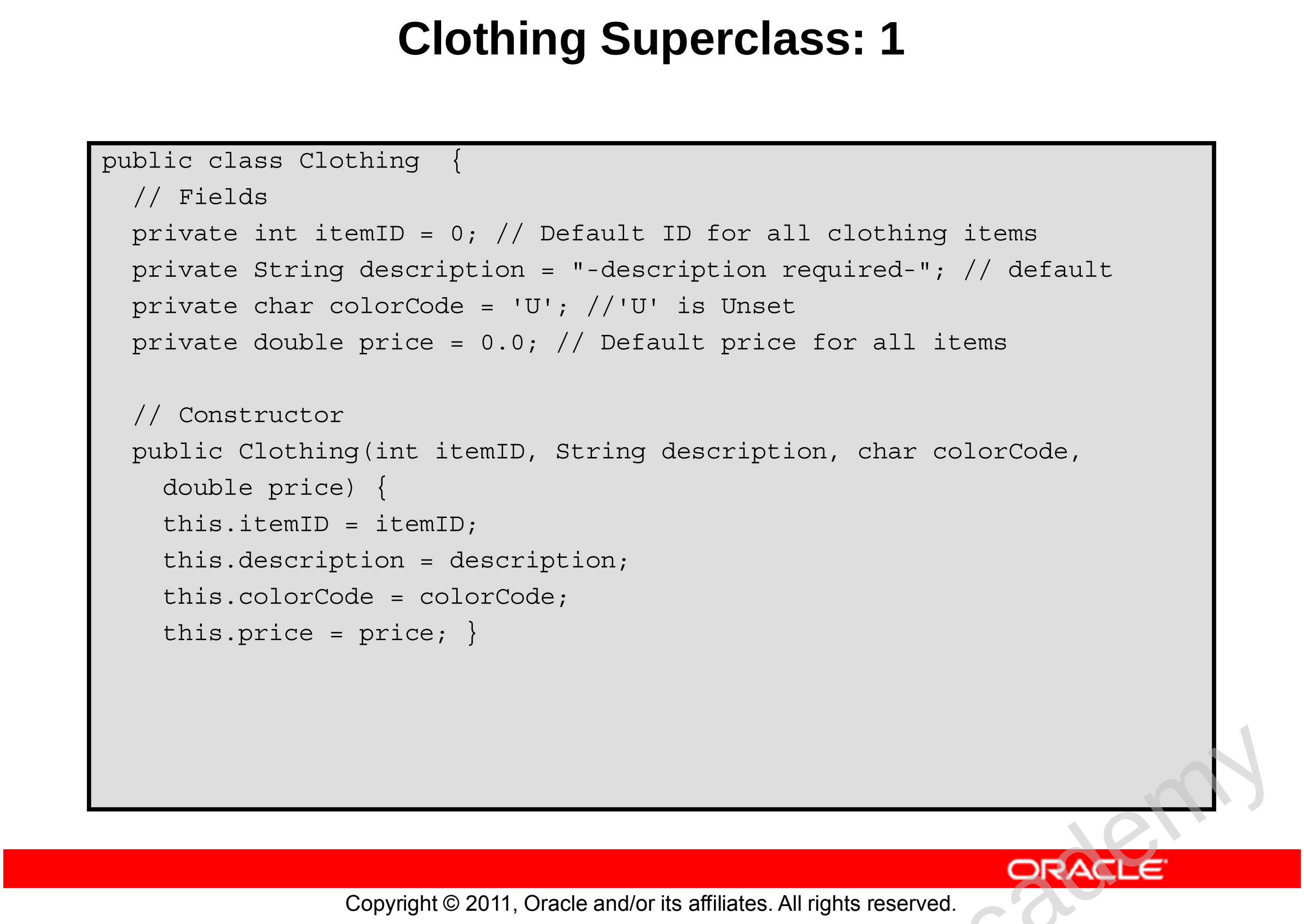
Methods that exist in the superclass can be:

- Not implemented in the subclass
  - The method declared in the superclass is used at runtime.
- Implemented in the subclass
  - The method declared in the subclass is used at runtime.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Subclasses may implement methods that already have implementations in the superclass. In this case, the method implementations in the subclass are said to override the method implementation from the superclass. For example, although the `colorCode` field (and its accessor methods) is in the superclass, the color choices may be different in each subclass. So, it may be necessary to override the get and set methods for this field in the individual subclasses.

Oracle Internal & Oracle Academy Use Only



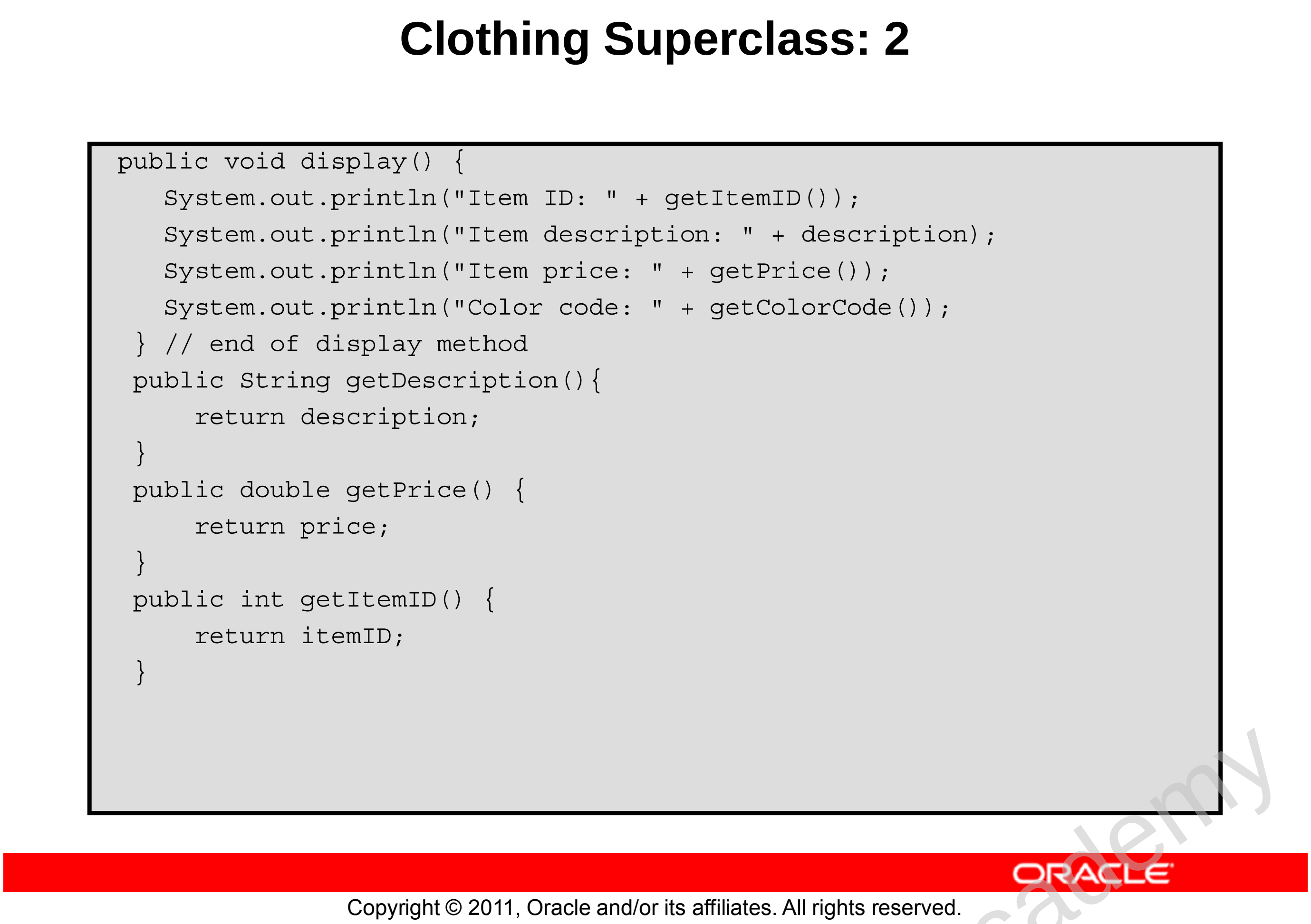
Clothing Superclass: 1

```
public class Clothing  {
  // Fields
  private int itemID = 0; // Default ID for all clothing items
  private String description = "-description required-"; // default
  private char colorCode = 'U'; //'U' is Unset
  private double price = 0.0; // Default price for all items

  // Constructor
  public Clothing(int itemID, String description, char colorCode,
    double price) {
    this.itemID = itemID;
    this.description = description;
    this.colorCode = colorCode;
    this.price = price; }
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows the fields and the constructor for the Clothing superclass.

Oracle Internal & Oracle Academy Use Only



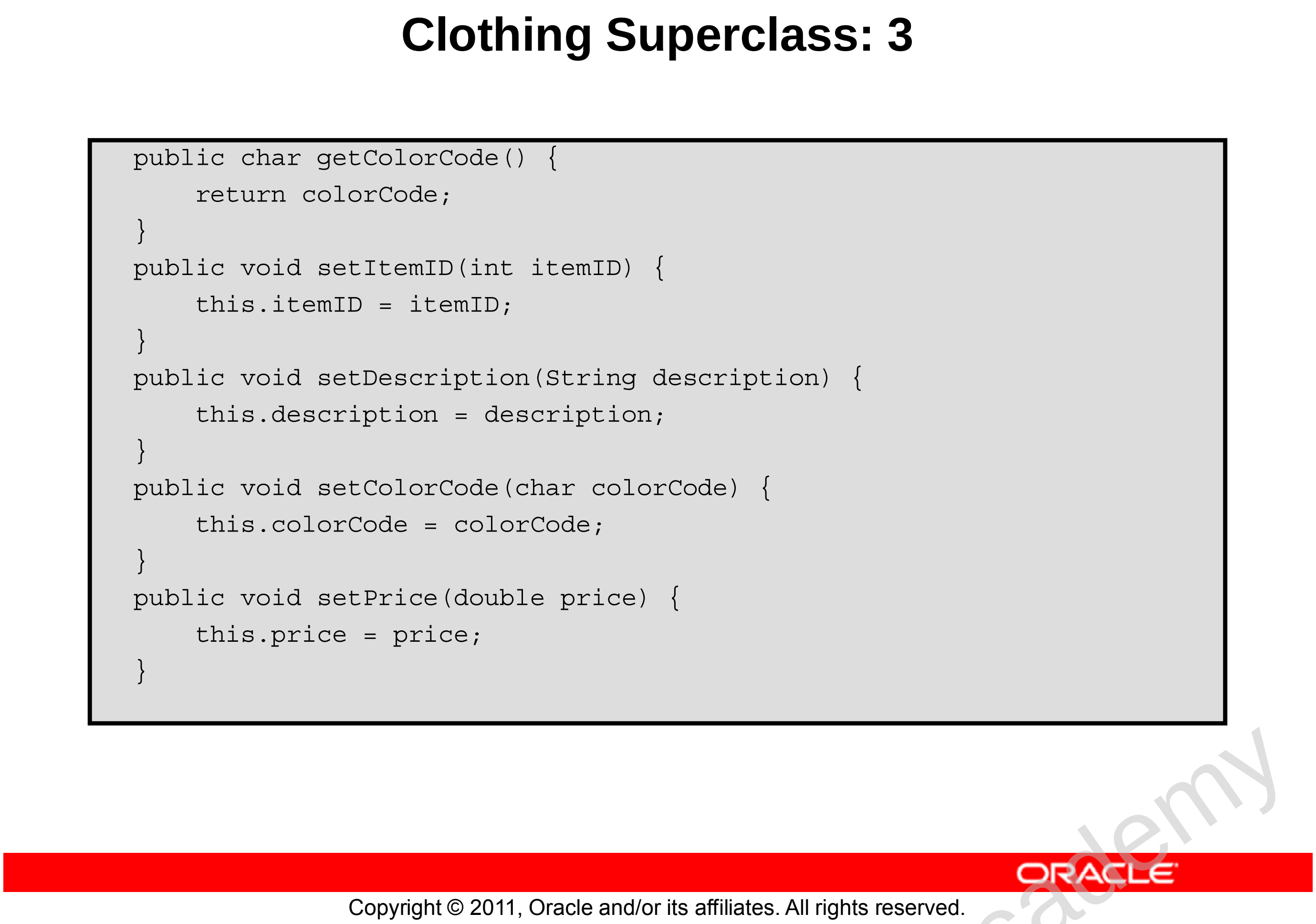
Clothing Superclass: 2

```
public void display() {
   System.out.println("Item ID: " + getItemID());
   System.out.println("Item description: " + description);
   System.out.println("Item price: " + getPrice());
   System.out.println("Color code: " + getColorCode());
 } // end of display method
 public String getDescription(){
     return description;
 }
 public double getPrice() {
     return price;
 }
 public int getItemID() {
     return itemID;
 }
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows methods for the Clothing superclass.

Oracle Internal & Oracle Academy Use Only



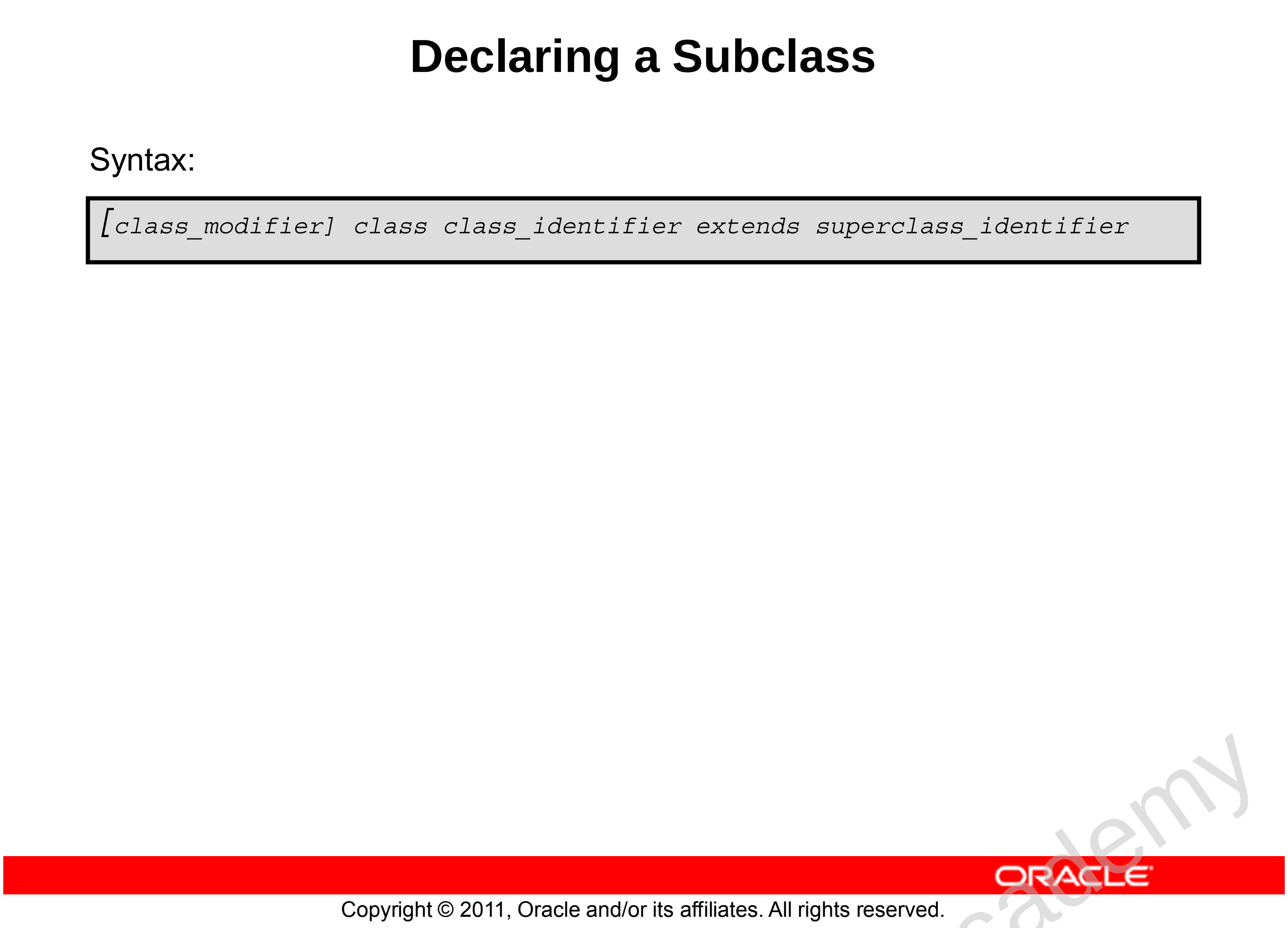
Clothing Superclass: 3

```
public char getColorCode() {
    return colorCode;
}
public void setItemID(int itemID) {
    this.itemID = itemID;
}
public void setDescription(String description) {
    this.description = description;
}
public void setColorCode(char colorCode) {
    this.colorCode = colorCode;
}
public void setPrice(double price) {
    this.price = price;
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows the remaining methods of the Clothing superclass.

Oracle Internal & Oracle Academy Use Only



Declaring a Subclass

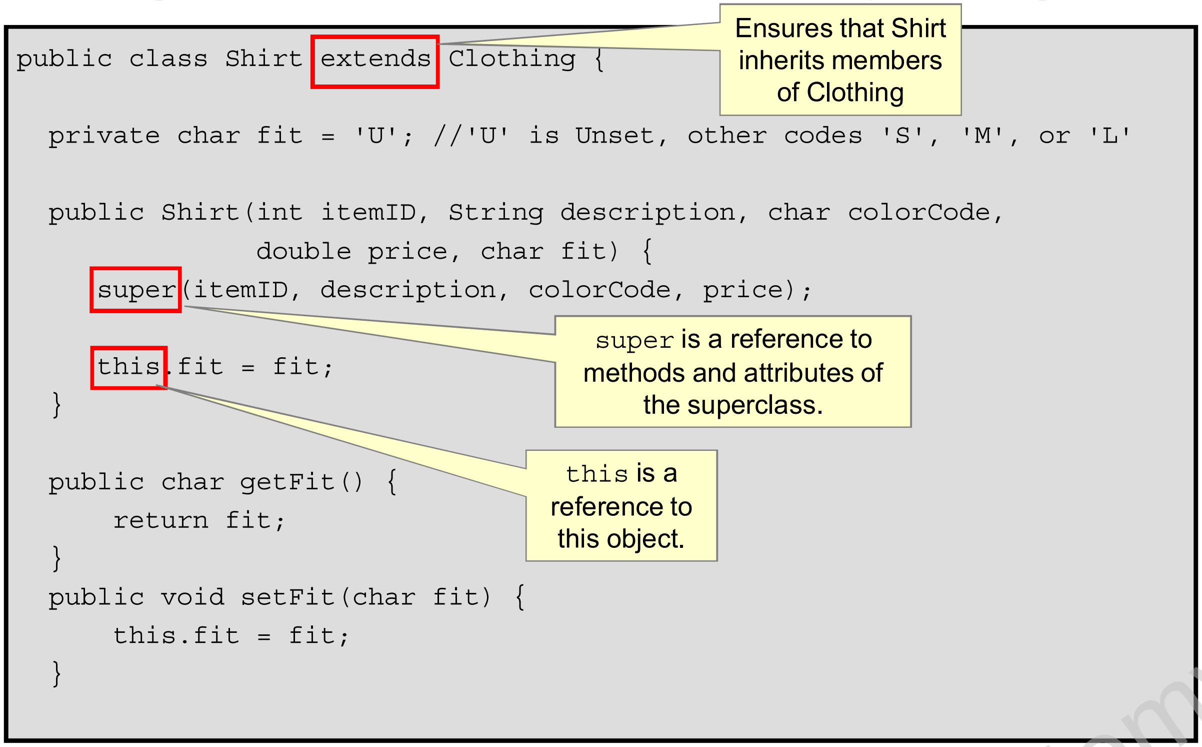
Syntax:

```
[class_modifier] class class_identifier extends superclass_identifier
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Declaring a Subclass (`extends`, `super`, and `this` keywords)



```
public class Shirt extends Clothing {

  private char fit = 'U'; //'U' is Unset, other codes 'S', 'M', or 'L'

  public Shirt(int itemID, String description, char colorCode,
               double price, char fit) {
     super(itemID, description, colorCode, price);

     this.fit = fit;
  }

  public char getFit() {
      return fit;
  }
  public void setFit(char fit) {
      this.fit = fit;
  }
```

Ensures that Shirt inherits members of Clothing

`super` is a reference to methods and attributes of the superclass.

`this` is a reference to this object.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

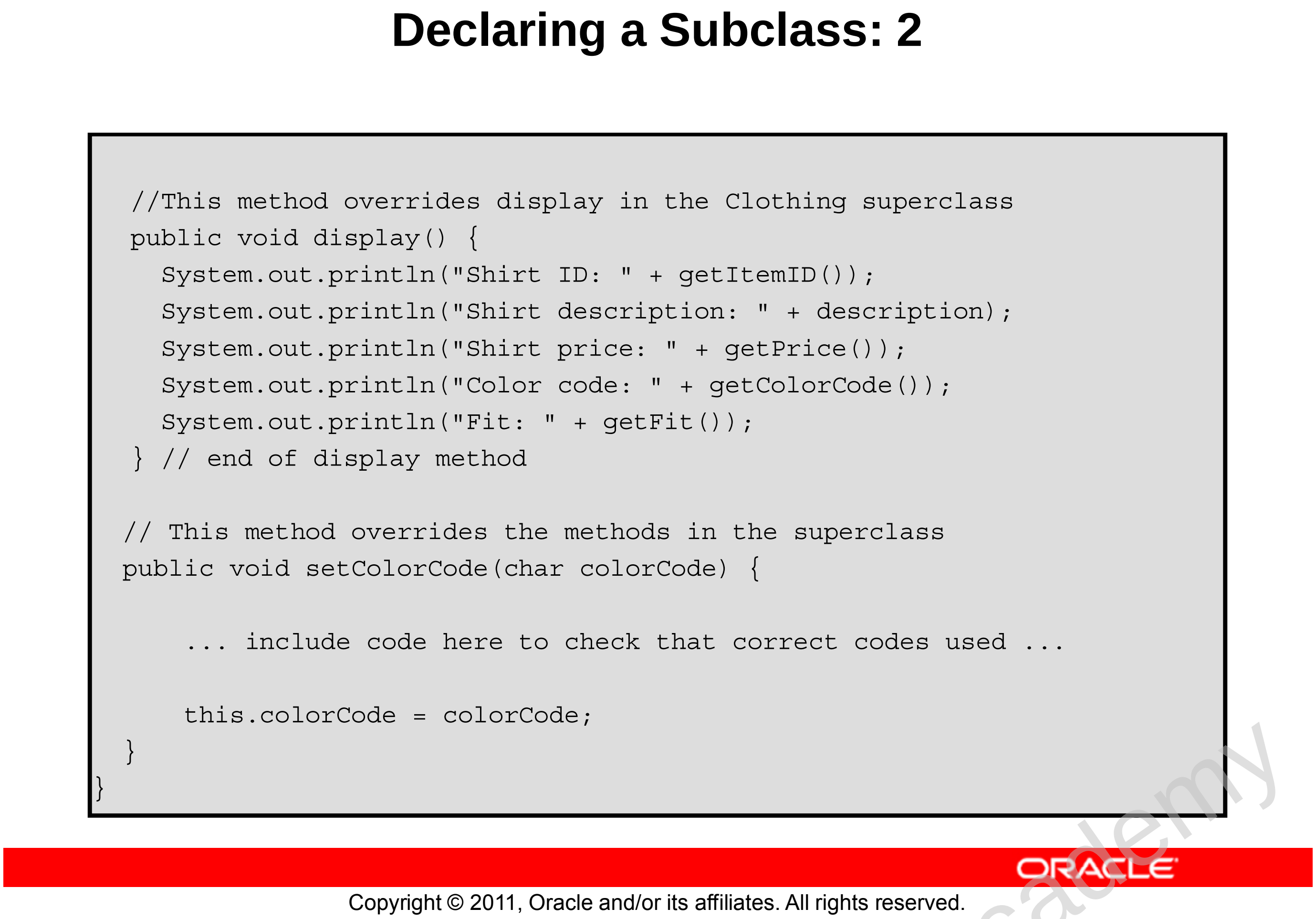
The slide shows the code of the `Shirt` subclass. The code declares attributes and methods that are unique to this class. Attributes and methods that are common with the `Clothing` class are inherited and do not need to be declared.

It also includes two useful keywords and shows a common way of implementing constructors in a subclass.

`super` refers to the superclass. Even if a method of the superclass has been overridden in the subclass, using the `super` keyword allows you to invoke the method of the superclass. In the example in the slide, it is used to invoke the constructor on the superclass. By using this technique, the constructor on the superclass can be invoked to set all the common attributes of the object being constructed. Then, as in the example here, additional attributes can be set in following statements.

The only additional attribute that `Shirt` has is the `fit` attribute, and it is set after the invocation of the superclass constructor. Note the use of the `this` keyword. In contrast to the `super` keyword, `this` is a reference to the object of this class. It is not necessary to use it in the example in the slide, but it is common to do so in constructors to help make the code more readable.

Oracle Internal & Oracle Academy Use Only



Declaring a Subclass: 2

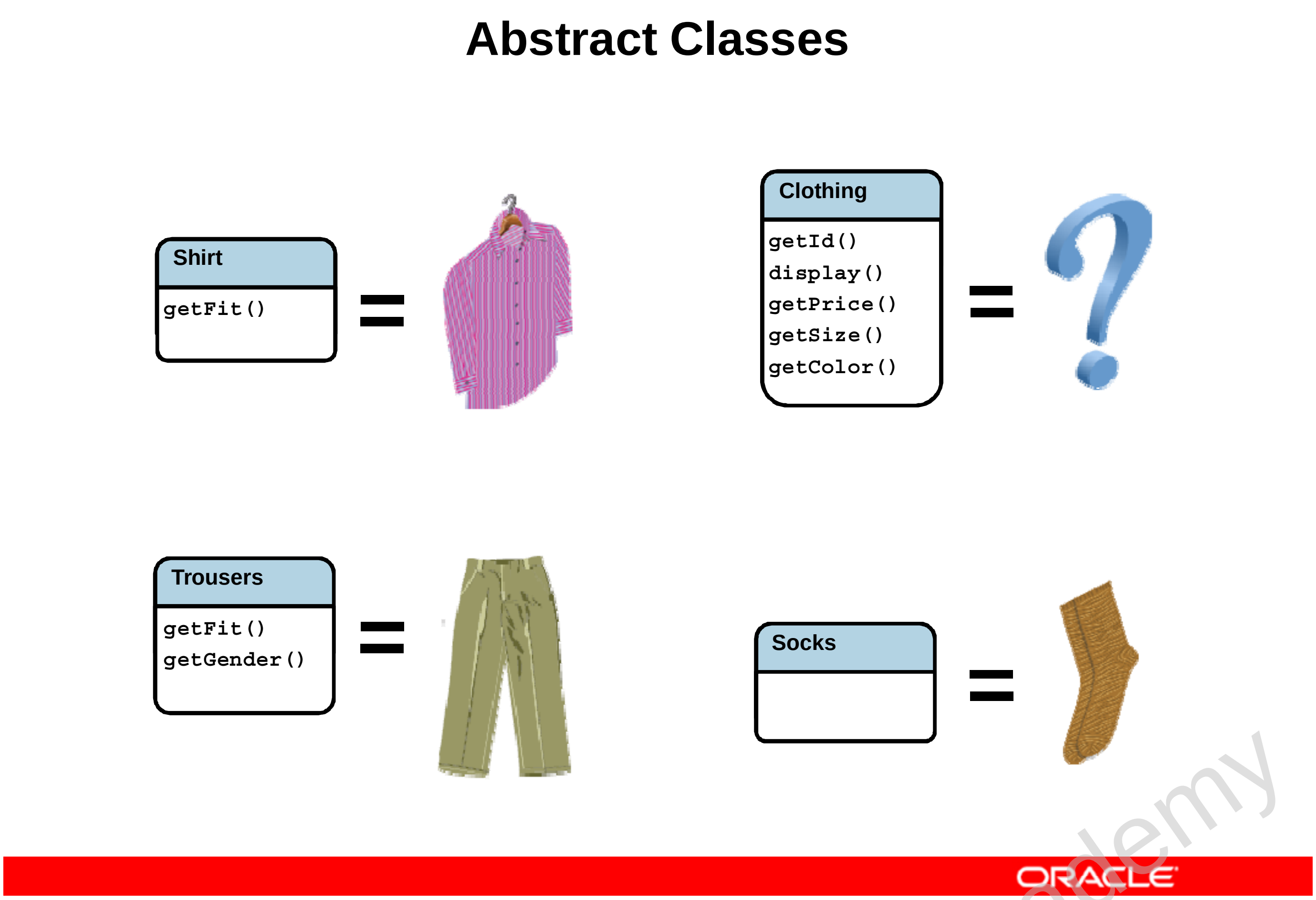
```
  //This method overrides display in the Clothing superclass
  public void display() {
    System.out.println("Shirt ID: " + getItemID());
    System.out.println("Shirt description: " + description);
    System.out.println("Shirt price: " + getPrice());
    System.out.println("Color code: " + getColorCode());
    System.out.println("Fit: " + getFit());
  } // end of display method

  // This method overrides the methods in the superclass
  public void setColorCode(char colorCode) {

      ... include code here to check that correct codes used ...

      this.colorCode = colorCode;
  }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Notice that the `display()` method overrides the `display()` method of the superclass and is more specific to the Shirt class.

Likewise, the `setColorCode()` method overrides the `setColorCode()` method of the superclass to check whether a valid value is being used for this class. (The code is not shown here, but remember that this is one of the advantages of encapsulating fields, as discussed in the lesson titled “Using Encapsulation and Constructors.”)

Oracle Internal & Oracle Academy Use Only

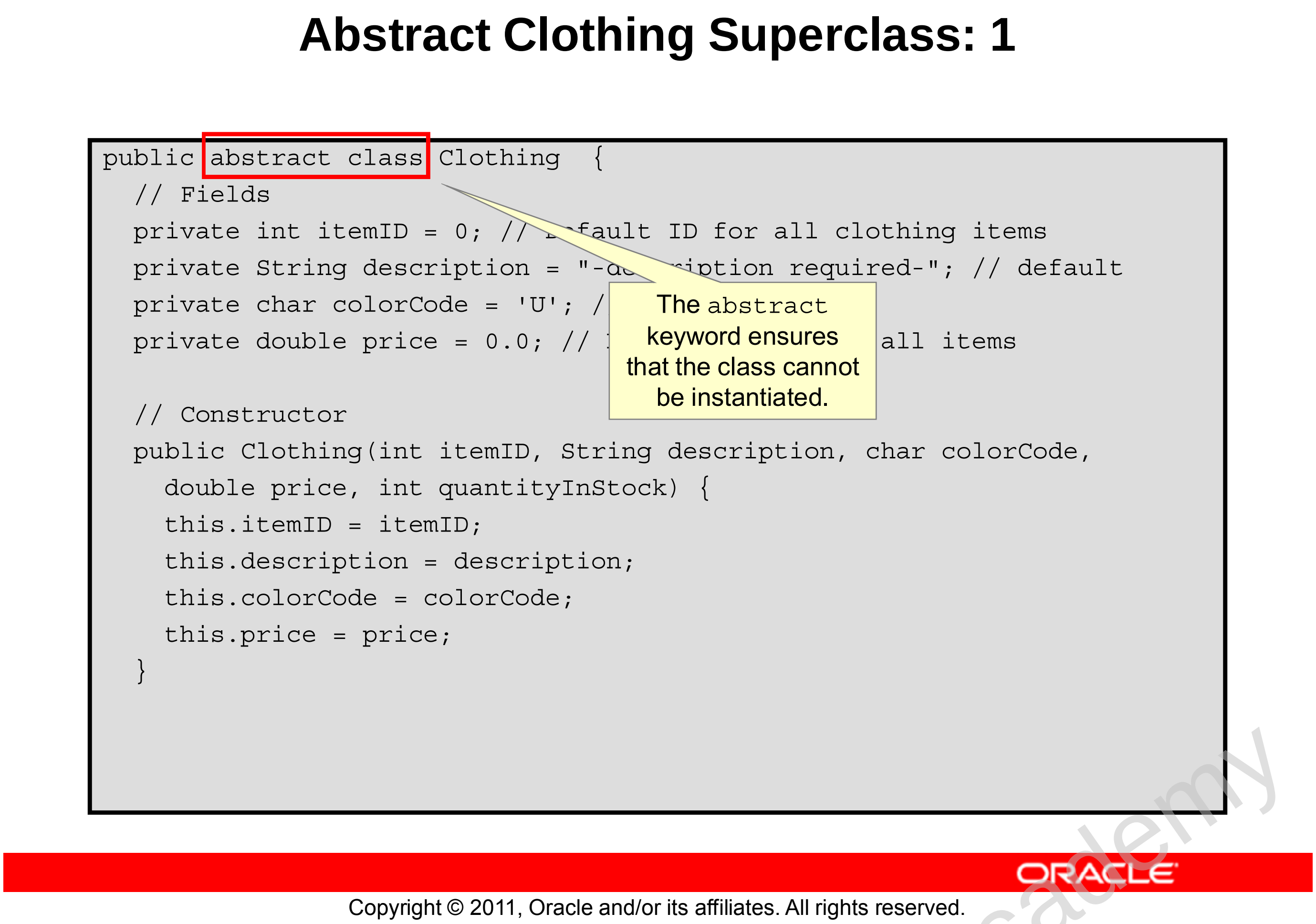


Abstract Classes

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Sometimes a superclass makes sense as an object, and sometimes it does not. Duke's Choice carries shirts, socks, and trousers, but it does not have an individual item called a “clothing.” Also, in the application, the superclass Clothing may declare some methods that may be required in each subclass (and thus can be in the superclass), but cannot really be implemented in the superclass.

Oracle Internal & Oracle Academy Use Only



Abstract Clothing Superclass: 1

```
public abstract class Clothing  {
  // Fields
  private int itemID = 0; // Default ID for all clothing items
  private String description = "-description required-"; // default
  private char colorCode = 'U'; /
  private double price = 0.0; // I                    all items

  // Constructor
  public Clothing(int itemID, String description, char colorCode,
    double price, int quantityInStock) {
    this.itemID = itemID;
    this.description = description;
    this.colorCode = colorCode;
    this.price = price;
  }
```

The abstract keyword ensures that the class cannot be instantiated.

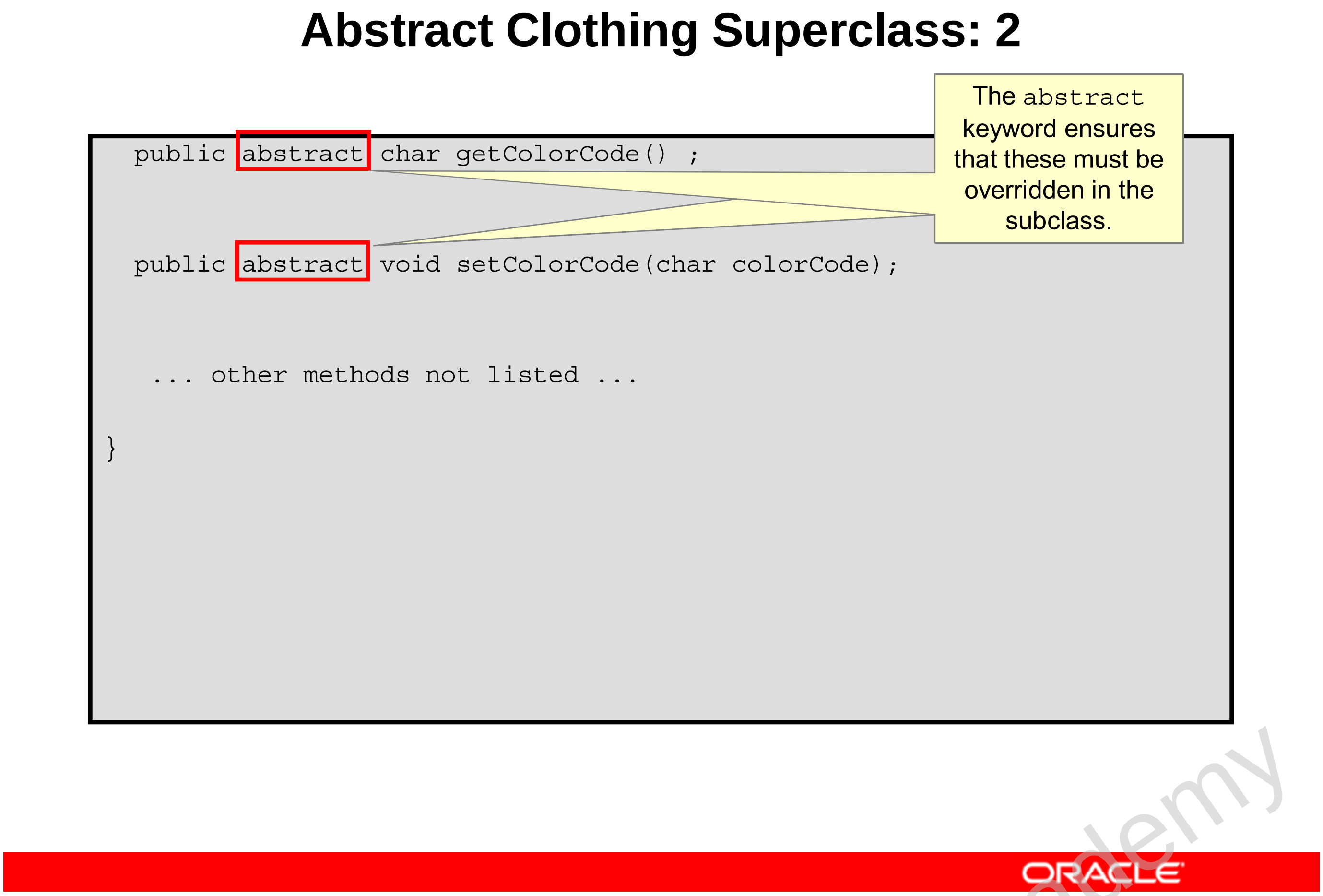
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Abstraction refers to creating classes that are general and may contain methods without particular implementation or method body code.

An example of an abstract class is the Clothing class as coded in this slide and the following slides. Clothing is an abstract concept that can refer to anything. (You usually do not go to a store and say, “I want to buy a clothing item.”)

However, all clothing items have some similar characteristics in the context of an order entry system, such as an ID or a method to display information about the item. Classes that are generic and cannot be fully defined, such as an Item class, are referred to as *abstract* classes. Classes that extend an abstract class must implement the empty methods of the abstract class with code specific to the subclass. You should spend time on your analysis and design to make sure that your solution has enough abstraction to ensure flexibility.

Oracle Internal & Oracle Academy Use Only



Abstract Clothing Superclass: 2

```
  public abstract char getColorCode() ;


  public abstract void setColorCode(char colorCode);



   ... other methods not listed ...

}
```

The `abstract` keyword ensures that these must be overridden in the subclass.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The get and set methods for the `colorCode` field are abstract to ensure that they are implemented appropriately in each subclass.

Note that the `Shirt` subclass shown previously will compile correctly as a subclass of this abstract class because it already has implementations of these two methods. But if the implementations of `getColorCode()` and `setColorCode()` are removed from the `Shirt` subclass, the compile will fail because abstract methods in the superclass must be implemented in the subclass.

Oracle Internal & Oracle Academy Use Only

Superclass and Subclass Relationships

It is very important to consider the best use of inheritance:

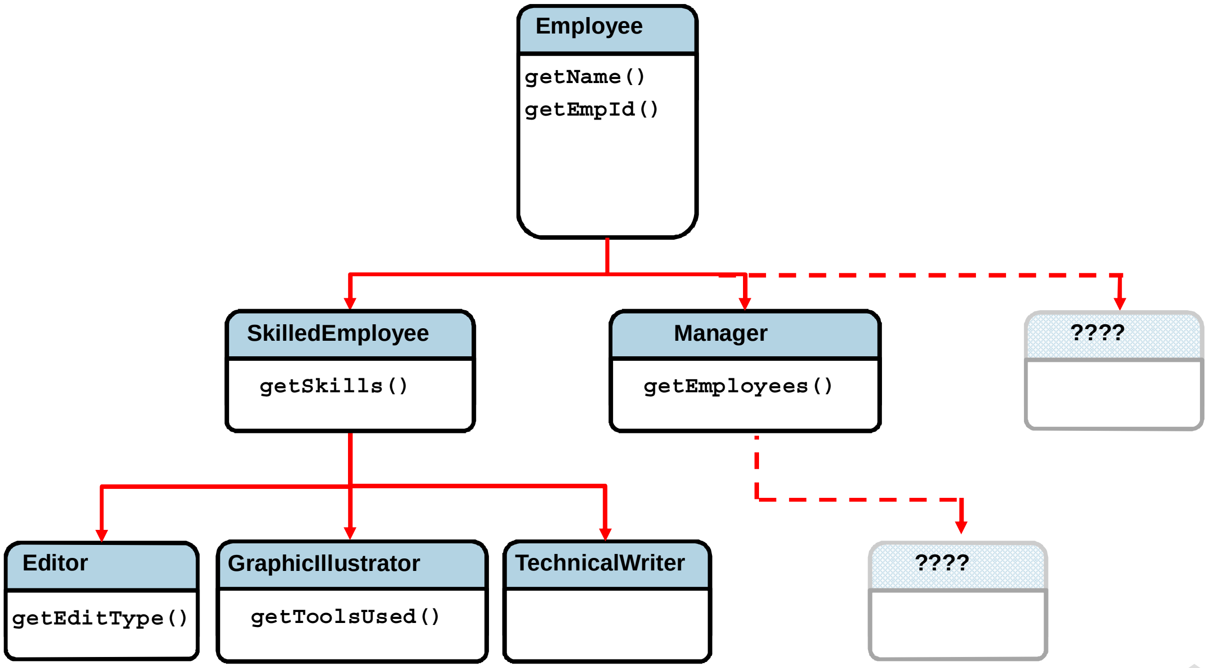
- Use inheritance only when it is completely valid or unavoidable.
- Check appropriateness with the “*is a”* phrase:
  - The phrase “a Shirt is a piece of Clothing” expresses a valid inheritance link.
  - The phrase “a Hat is a Sock” expresses an invalid inheritance link.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the examples in this course, shirts, trousers, hats, and socks are all types of clothing. So Clothing is a good name for the superclass to these subclasses (types) of clothing.

Oracle Internal & Oracle Academy Use Only

Another Inheritance Example



ORACLE
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of another set of superclasses and subclasses. In this case, there are more than two levels. The base superclass is Employee, and Employee currently has two subclasses. One of the big advantages of inheritance is that it is easy at any future time to create a new class that extends Employee, and that class inherits all the functionality that Employee has.

One of the Employee subclasses is SkilledEmployee, and the diagram shows that it has three subclasses of its own: Editor, GraphicIllustrator, and TechnicalWriter.

None of these classes are abstract. There is such a thing as an employee and some processes in an application using these classes may work with the Employee class.

Oracle Internal & Oracle Academy Use Only

Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Polymorphism and overriding methods
- Interfaces
- The Object class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Superclass Reference Types

So far you have seen the class used as the reference type for the created object:

- To use the Shirt class as the reference type for the Shirt object:

```
Shirt myShirt = new Shirt();
```

- But you can also use the superclass as the reference:

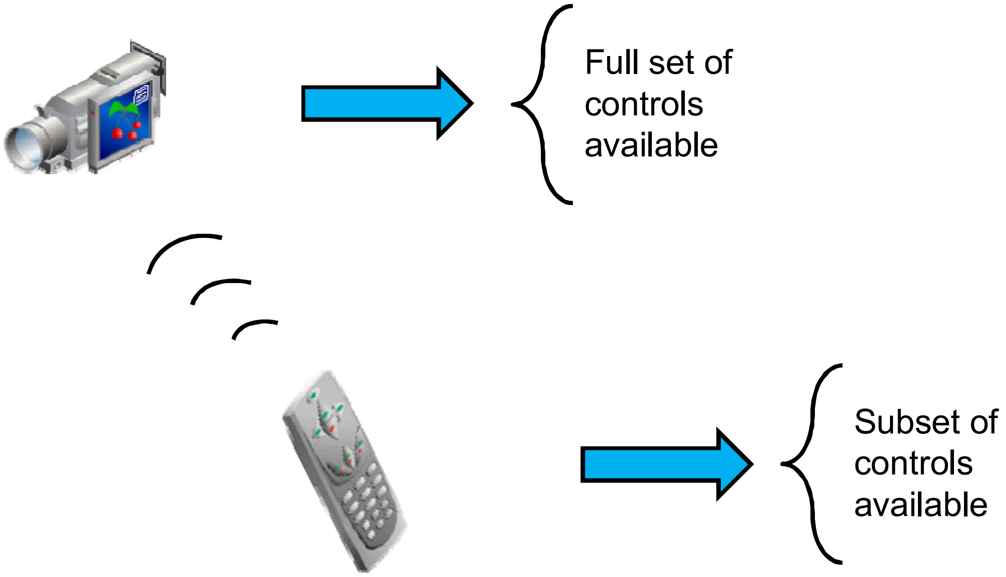
```
Clothing clothingItem1 = new Shirt();
Clothing clothingItem2 = new Trousers();
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

A very important feature of Java is this ability to use not only the class itself but any superclass of the class as its reference type. In the example shown in the slide, notice that you can refer to both a Shirt object and a Trousers object with a Clothing reference. This means that a reference to a Shirt or Trousers object can be passed into a method that requires a Clothing reference. Or a Clothing array can contain references to Shirt, Trousers, or Socks objects.

Oracle Internal & Oracle Academy Use Only

Access to Object Functionality



Full set of controls available

Subset of controls available

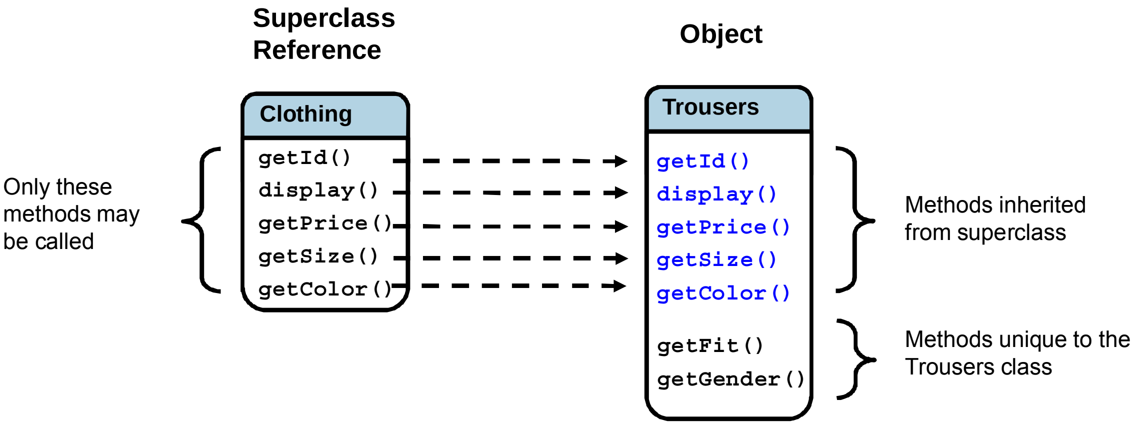
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Accessing the methods of a class using a superclass reference is a little like accessing the controls of an electronic device using a remote control instead of the controls on the device itself. Often a device such as a video camera has a comprehensive set of controls for recording, playing, editing, and otherwise accessing every available function of the camera. This is a lot like using the class of the object as the reference type.

For some combinations of video camera and remote, the remote may give you exactly the same controls, and this can also be the case when using a superclass as reference for an object (the superclass gives you access to all the methods of the object; the object's class does not add any new methods). But it is often the case that the remote control does not have the full set of controls available on the camera itself. Again, this is common when using the superclass as reference. The superclass has access only to the methods of the object that are declared on the superclass even if the object has a number of other methods.

Oracle Internal & Oracle Academy Use Only

Accessing Class Methods from Superclass

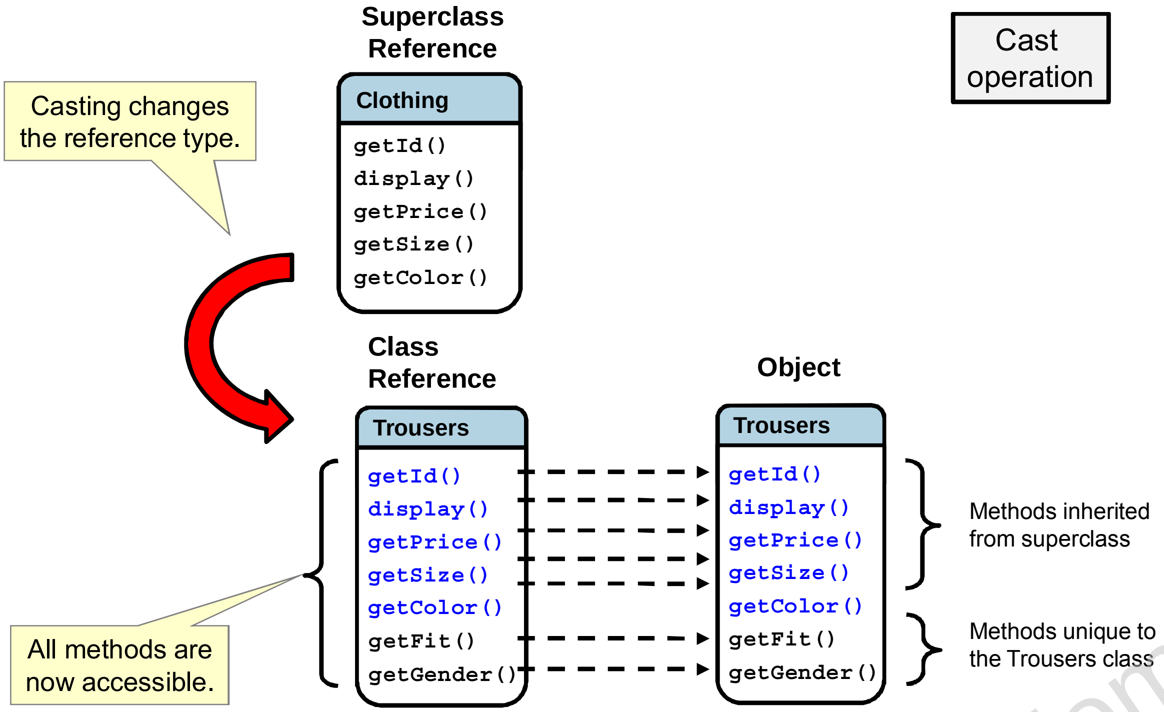


Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Using a reference type `Clothing` does not allow access to the `getFit()` or `getGender()` method of the `Trouser` object. Usually, this is not a problem because you are most likely to be passing `Clothing` references to methods that do not require access to these methods. For example, a `purchase()` method could receive a `Clothing` argument because it needs access only to the `getPrice()` method.

Oracle Internal & Oracle Academy Use Only

Casting the Reference Type



ORACLE

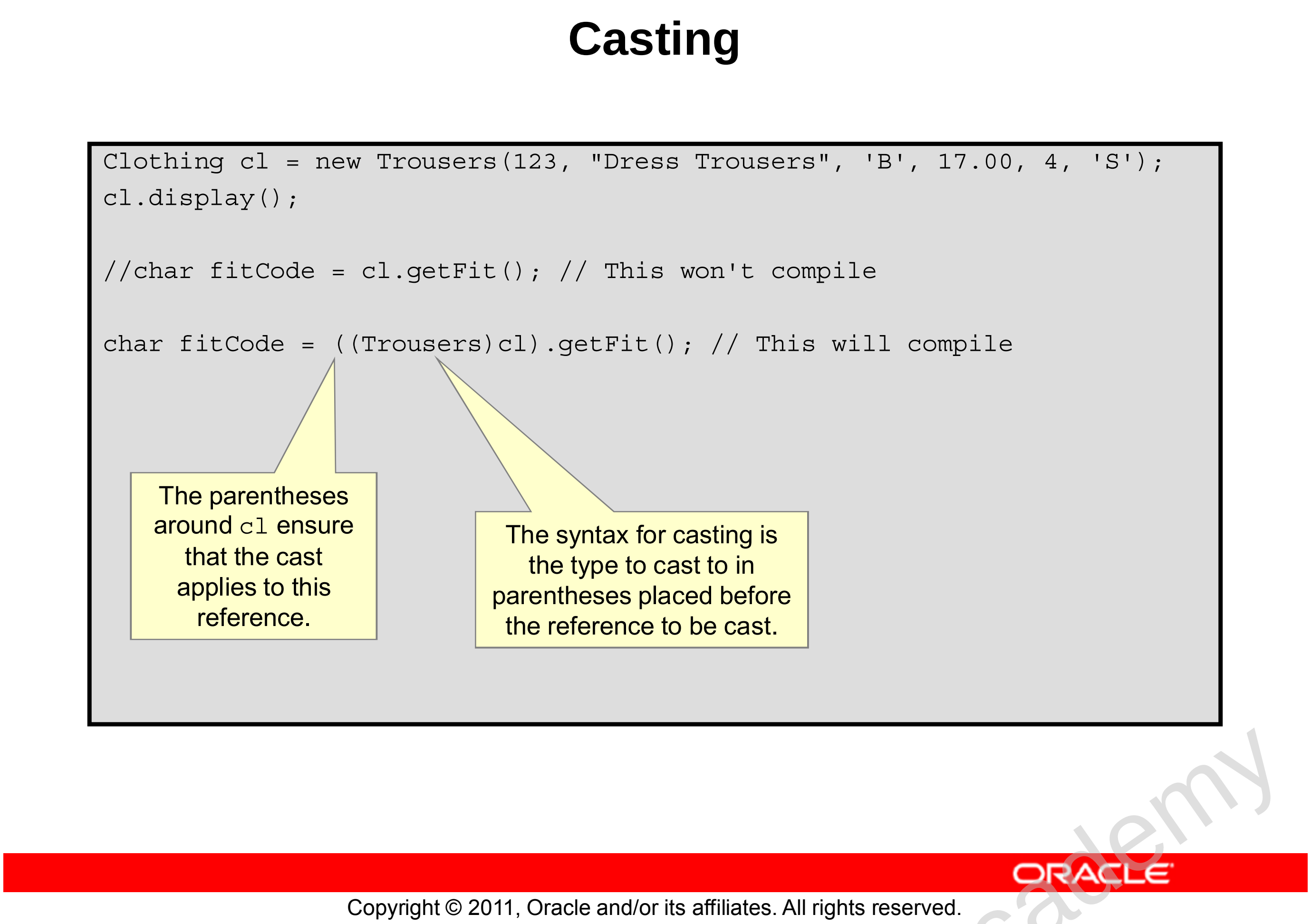
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Given that a superclass may not have access to all the methods of the object it is referring to, how can you access those methods? The answer is that you can do so by replacing the superclass reference by:

- A reference that is the same type as the object
- An interface that declares the methods in question and is implemented by the class of the object

(Interfaces are covered in the next topic of this lesson.)

Oracle Internal & Oracle Academy Use Only



Casting

```
Clothing cl = new Trousers(123, "Dress Trousers", 'B', 17.00, 4, 'S');
cl.display();

//char fitCode = cl.getFit(); // This won't compile

char fitCode = ((Trousers)cl).getFit(); // This will compile
```

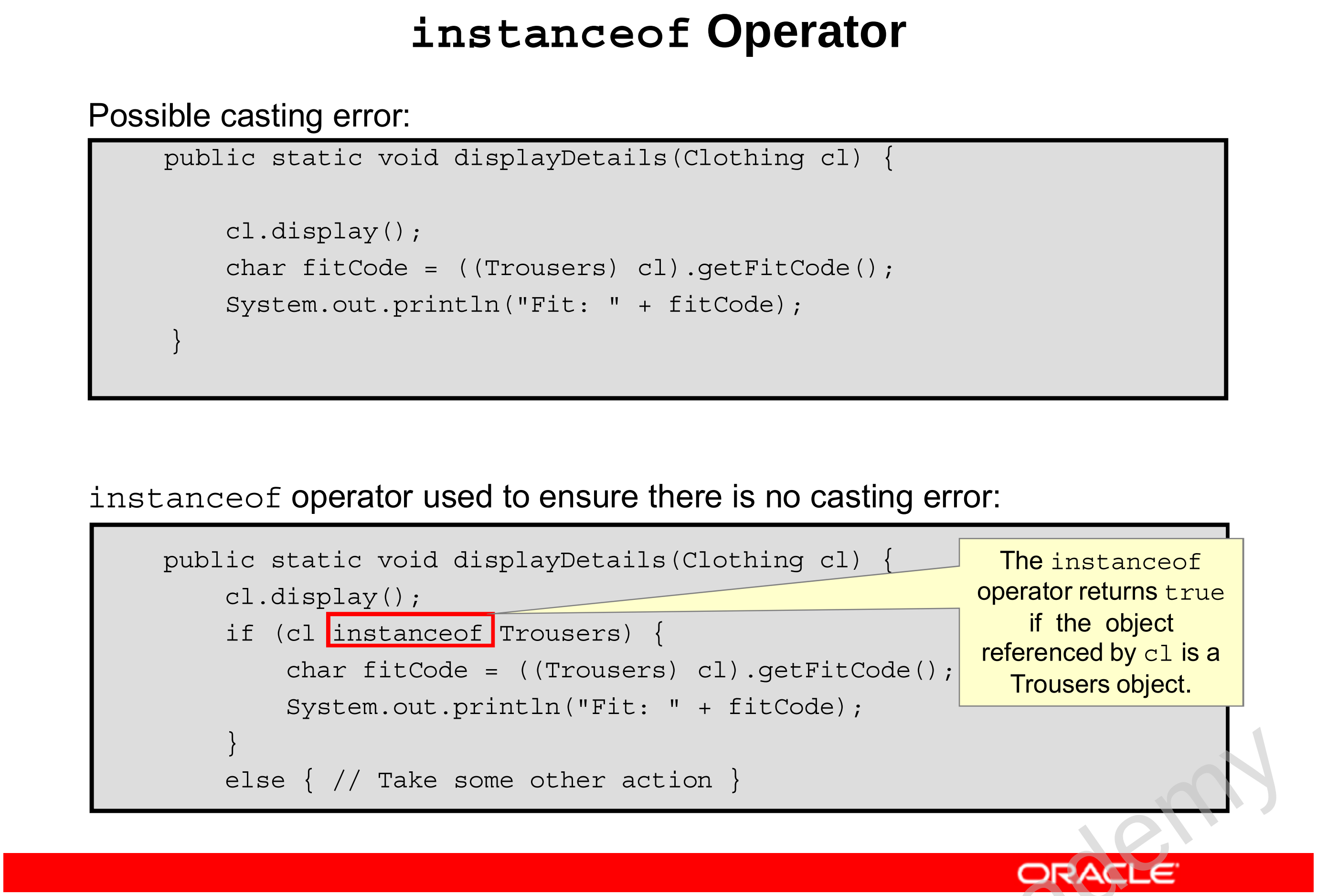
The parentheses around `cl` ensure that the cast applies to this reference.

The syntax for casting is the type to cast to in parentheses placed before the reference to be cast.

ORACLE
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in this example shows a `Clothing` reference being cast to a `Trousers` reference to access the `getFit()` method, which is not accessible via the `Clothing` reference. Note that the inner parentheses around `Trousers` are part of the cast syntax, and the outer parentheses around `(Trousers)cl` are there to apply the cast to the `Clothing` type.

Oracle Internal & Oracle Academy Use Only



`instanceof` Operator

Possible casting error:

```
    public static void displayDetails(Clothing cl) {

        cl.display();
        char fitCode = ((Trousers) cl).getFitCode();
        System.out.println("Fit: " + fitCode);
    }
```

`instanceof` operator used to ensure there is no casting error:

```
    public static void displayDetails(Clothing cl) {
        cl.display();
        if (cl instanceof Trousers) {
            char fitCode = ((Trousers) cl).getFitCode();
            System.out.println("Fit: " + fitCode);
        }
        else { // Take some other action }
```

The `instanceof` operator returns `true` if the object referenced by `cl` is a Trousers object.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

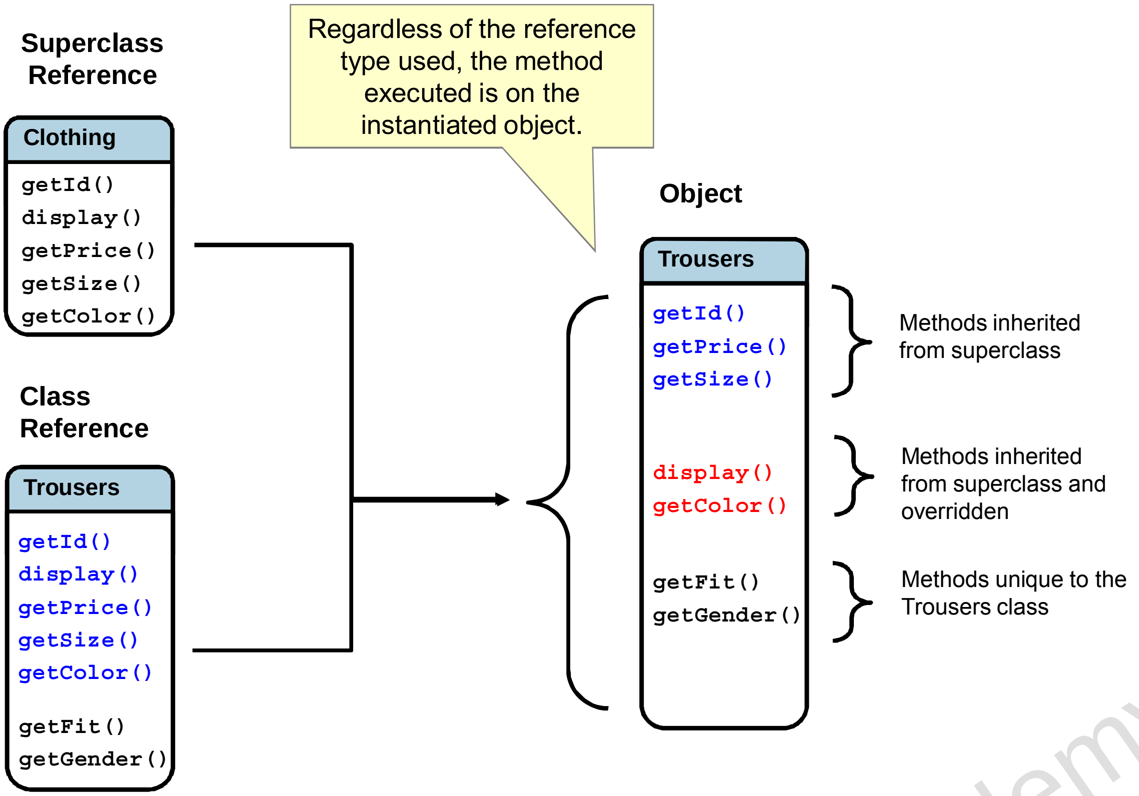
The first code example in the slide shows a method that is designed to receive an argument of type `Clothing`, and then cast it to `Trousers` to invoke a method that exists only on a `Trousers` object. But it is not possible to know what object type the reference `cl` points to. And if it is, say, a `Shirt`, the attempt to cast it will cause a problem. (It will throw a `CastClassException`. Throwing exceptions is covered in the lesson titled “Handling Errors.”)

You can code around this potential problem with the code shown in the second code example in the slide. Here the `instanceof` operator is used to ensure that `cl` is referencing an object of type `Trousers` before the cast is attempted.

If you think your code requires casting, be aware that there are often ways to design code so that casting is not necessary, and this is usually preferable. But if you do need to cast, you should use `instanceof` to ensure that the cast does not throw a `CastClassException`.

Oracle Internal & Oracle Academy Use Only

Polymorphic Method Calls



Regardless of the reference type used, the method executed is on the instantiated object.

Superclass Reference

Clothing

```
getId()
display()
getPrice()
getSize()
getColor()
```

Class Reference

Trousers

```
getId()
display()
getPrice()
getSize()
getColor()

getFit()
getGender()
```

Object

Trousers

```
getId()
getPrice()
getSize()
```

Methods inherited from superclass

```
display()
getColor()
```

Methods inherited from superclass and overridden

```
getFit()
getGender()
```

Methods unique to the Trousers class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Polymorphic behavior displayed by a statement may invoke one of the methods of Clothing. This is a polymorphic method call because the invocation does not know or need to know the type of the object (sometimes called the *runtime* type), but the correct method—that is, the method of the actual object—will be invoked. In the example in the slide, the object is Trousers, but it could be any subclass of Clothing.

Oracle Internal & Oracle Academy Use Only

Quiz

How can you change the reference type of an object?

a. By calling `getReference()`

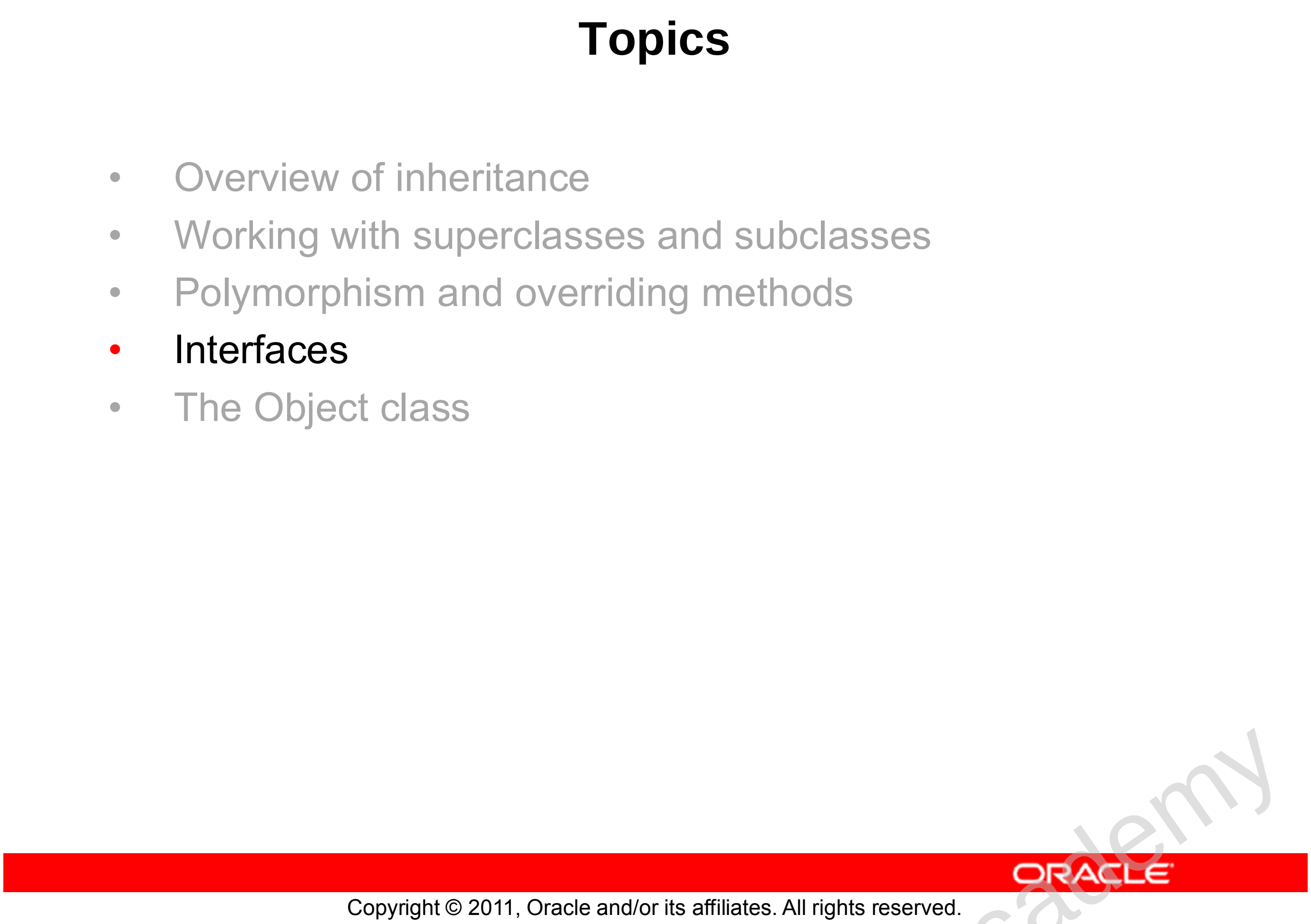
b. By casting

c. By declaring a new reference and assigning the object

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: b

Oracle Internal & Oracle Academy Use Only



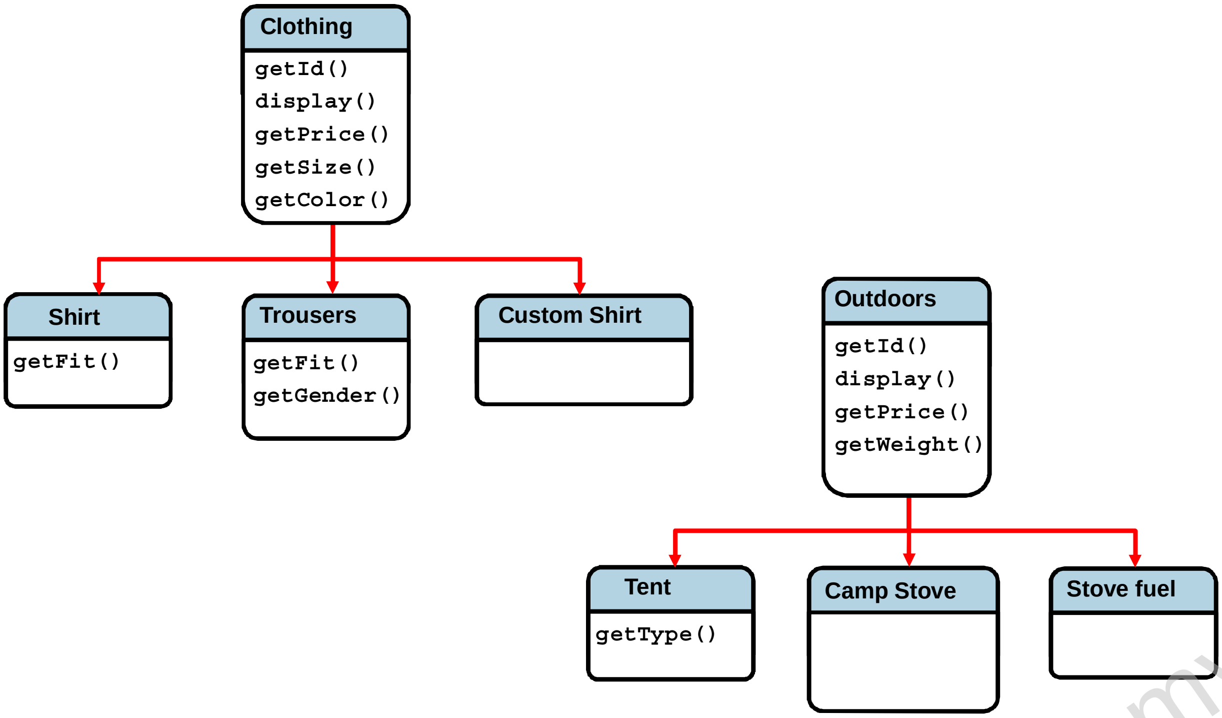
Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Polymorphism and overriding methods
- Interfaces
- The Object class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Multiple Hierarchies



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

A more complex set of classes may have items in two different hierarchies. If Duke's Choice starts selling outdoors gear, it may have a completely different superclass called Outdoors, with its own set of subclasses (for example, `getWeight()` as an Outdoors method).

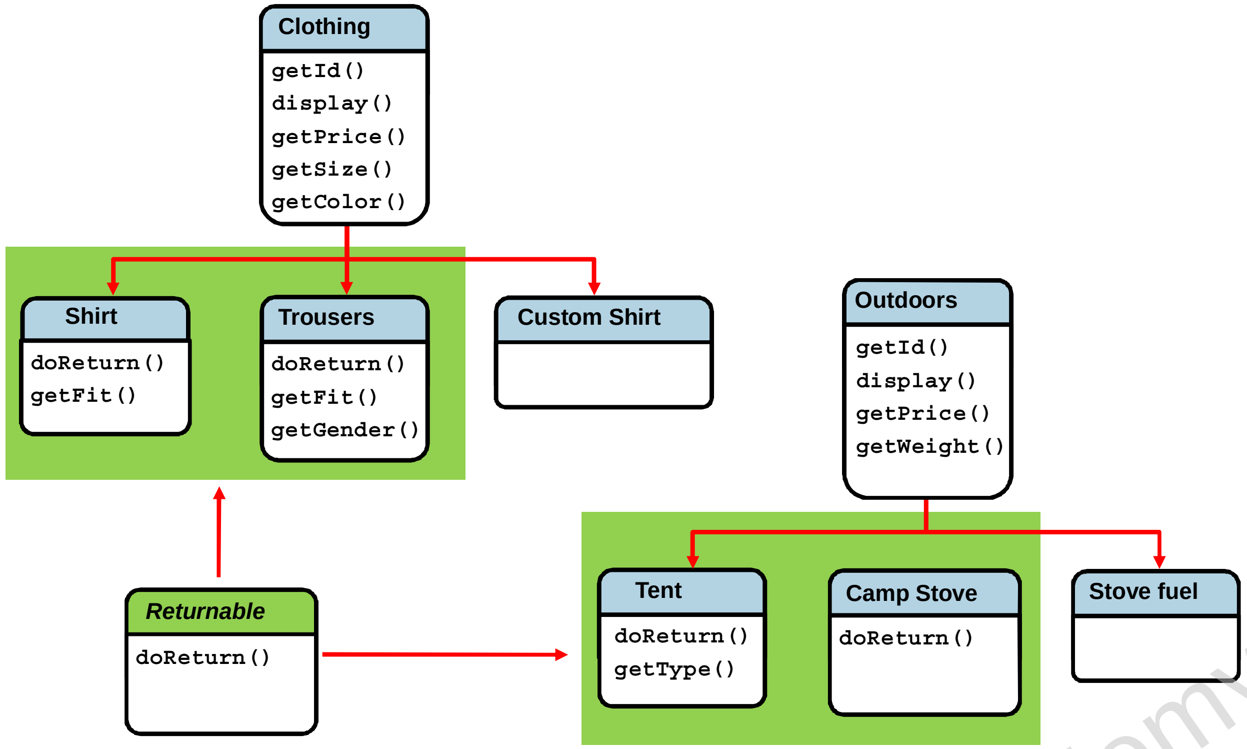
In this scenario, there may be some classes from each hierarchy that have something in common. For example, the custom shirt item in Clothing is not returnable (because it is made by hand for a particular person), and neither is the Stove fuel item in the Outdoors hierarchy. All other items are returnable.

How can this be modeled? Here are some things to consider:

- A new superclass will not work because a class can extend only one superclass, and all items are currently extending either Outdoors or Clothing.
- A new field named returnable, added to every class, could be used to determine if an item can be returned. This is certainly possible, but then there is no single reference type to pass to a method that initiates or processes a return.
- You can use a special type called an *Interface* that can be implemented by any class. This Interface type can then be used to pass a reference of any class that implements it.

Oracle Internal & Oracle Academy Use Only

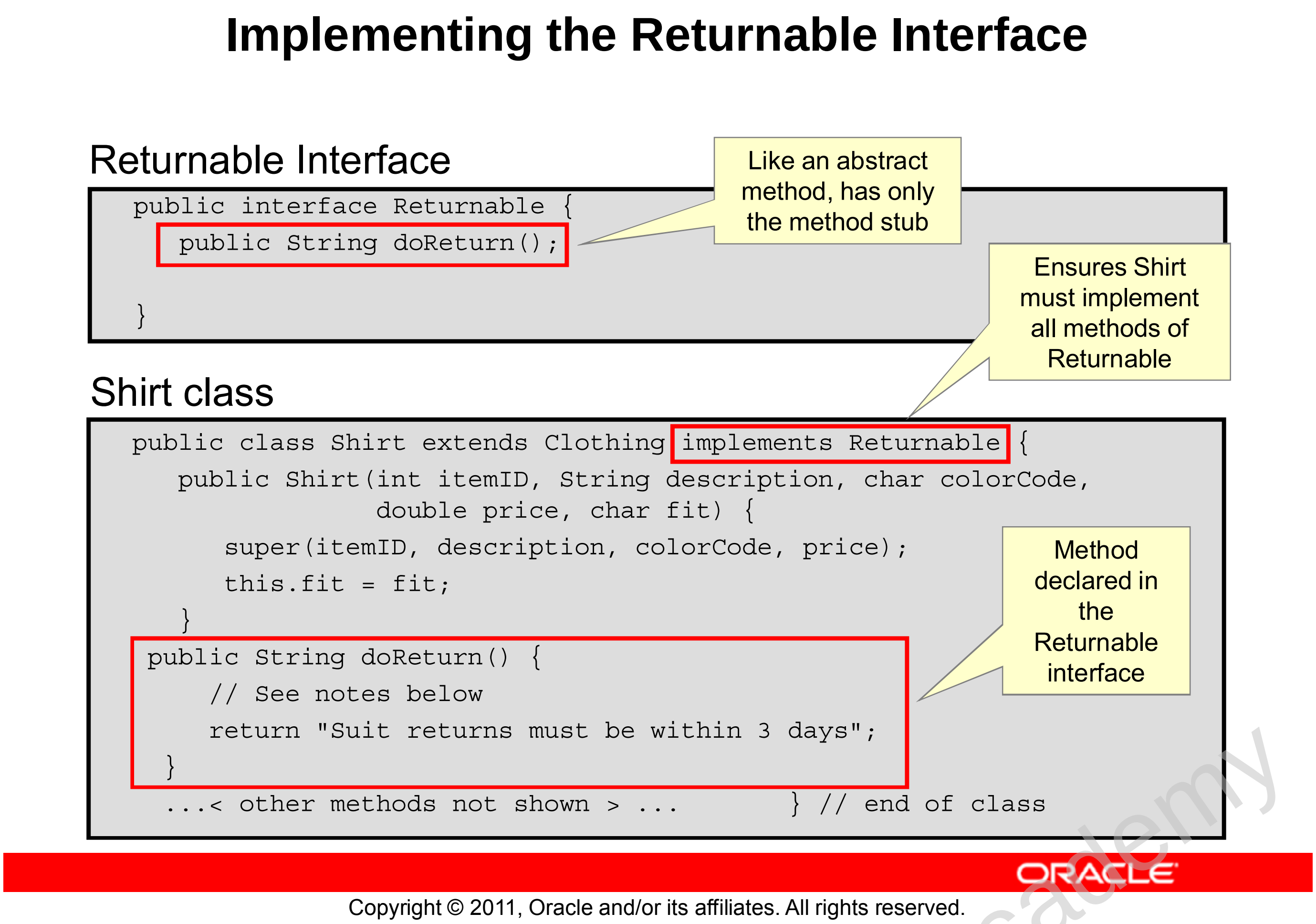
Interfaces



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows all returnable items implementing the Returnable interface with its single method, `doReturn()`. Methods can be declared in an interface, but they cannot be implemented in an interface. Therefore, each class that implements Returnable must implement `doReturn()` for itself. All returnable items could be passed to a `processReturns()` method of a Returns class, and then have their `doReturn()` method called.

Oracle Internal & Oracle Academy Use Only



Implementing the Returnable Interface

Returnable Interface

```
public interface Returnable {
   public String doReturn();

}
```

Like an abstract method, has only the method stub

Ensures Shirt must implement all methods of Returnable

Shirt class

```
public class Shirt extends Clothing implements Returnable {
   public Shirt(int itemID, String description, char colorCode,
                double price, char fit) {
      super(itemID, description, colorCode, price);
      this.fit = fit;
   }
 public String doReturn() {
     // See notes below
     return "Suit returns must be within 3 days";
 }
  ...< other methods not shown > ...        } // end of class
```

Method declared in the Returnable interface

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

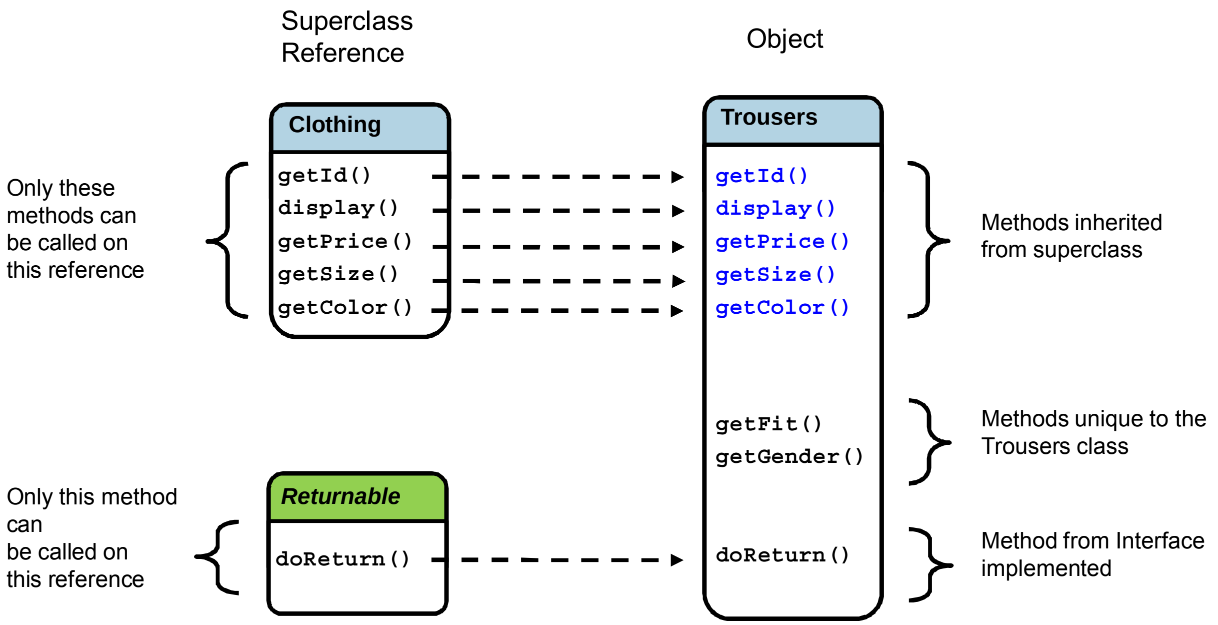
The code in this example shows the Returnable interface and the `Shirt` class. Only the constructor and the `doReturn()` method are shown.

In this implementation, `Returnable` provides a marker to indicate that the item can be returned, and ensures that the developer of Shirt must implement the `doReturn()` method.

The `doReturn()` method returns a String describing the conditions for returning the item.

Oracle Internal & Oracle Academy Use Only

Access to Object Methods from Interface



Superclass Reference

Object

Only these methods can be called on this reference

Clothing

```
getId()
display()
getPrice()
getSize()
getColor()
```

Trousers

```
getId()
display()
getPrice()
getSize()
getColor()
```

Methods inherited from superclass

```
getFit()
getGender()
```

Methods unique to the Trousers class

Only this method can be called on this reference

Returnable

```
doReturn()
```

```
doReturn()
```

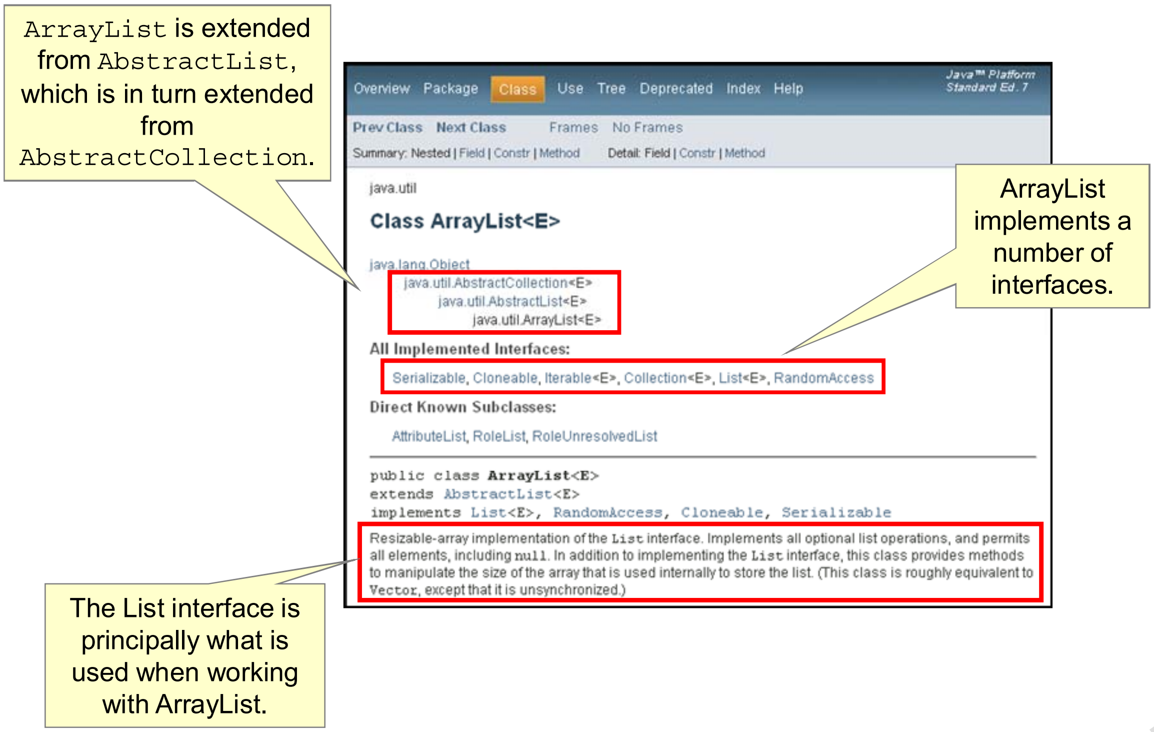
Method from Interface implemented

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

As shown in a previous slide, the reference used to access an object determines the methods that can be called on it. So in the case of the Interface reference shown in the slide, only the `getReturn()` method can be called. If a method receives a Returnable reference, however, and needs access to methods on Clothing or methods on Trousers, the reference can be cast to the appropriate reference type.

Oracle Internal & Oracle Academy Use Only

ArrayList


Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

`ArrayList` is extended from `AbstractList`, which is in turn extended from `AbstractCollection`.

java.util

Class ArrayList<E>

```
java.lang.Object
  java.util.AbstractCollection<E>
      java.util.AbstractList<E>
          java.util.ArrayList<E>
```

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)

ArrayList implements a number of interfaces.

The List interface is principally what is used when working with ArrayList.

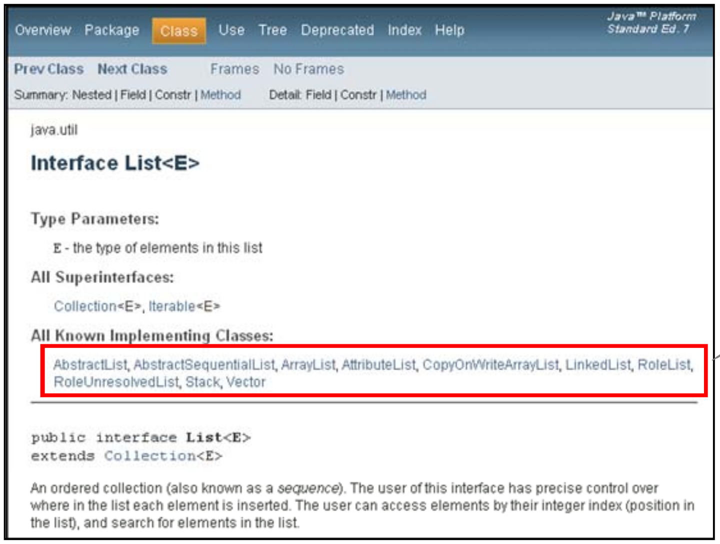
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Some of the best examples of inheritance and the utility of Interface and Abstract types can be found in the Java API. For example, the `ArrayList` class extends the `AbstractList` class, which itself extends `AbstractCollection`. `AbstractCollection` implements the List interface, which means that ArrayList also implements the List interface.

To use the ArrayList as a List, use the List interface as the reference type.

Oracle Internal & Oracle Academy Use Only

List Interface


Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Interface List<E>

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

```
public interface List<E>
extends Collection<E>
```

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Many classes implementing the List interface

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The List interface is implemented by many classes. This means that any method that requires a List may actually be passed a List reference to any objects of these types (but not the abstract classes, because they cannot be instantiated).

Oracle Internal & Oracle Academy Use Only

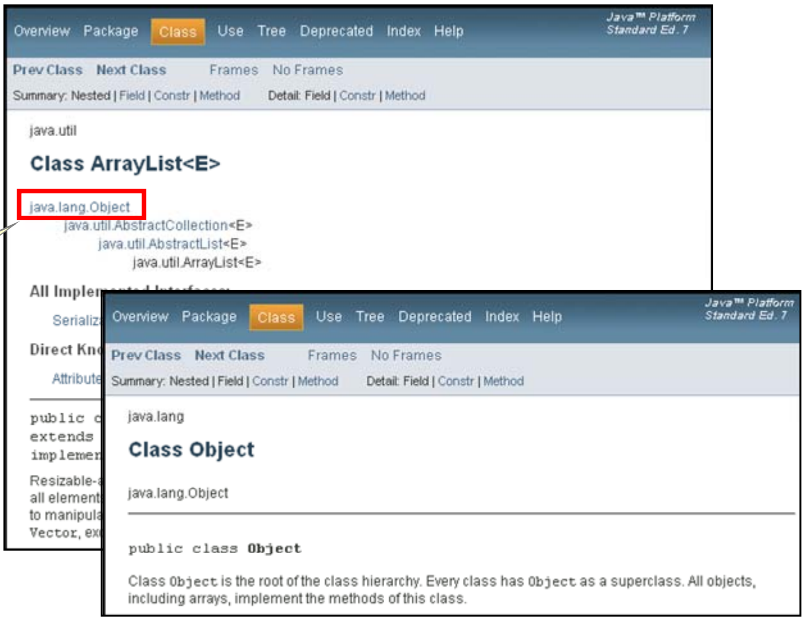
Topics

- Overview of inheritance
- Working with superclasses and subclasses
- Polymorphism and overriding methods
- Interfaces
- The Object class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Object Class


Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

```
java.util
Class ArrayList<E>
java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>
All Implem
    Serializ
Direct Kn
    Attribute
public c
extends
implemen
Resizable-
all element
to manipula
Vector, ex
```

Overview Package Class Use Tree Deprecated Index Help
Java™ Platform Standard Ed. 7
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class Object

java.lang.Object

`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

The `Object` class is the base class.

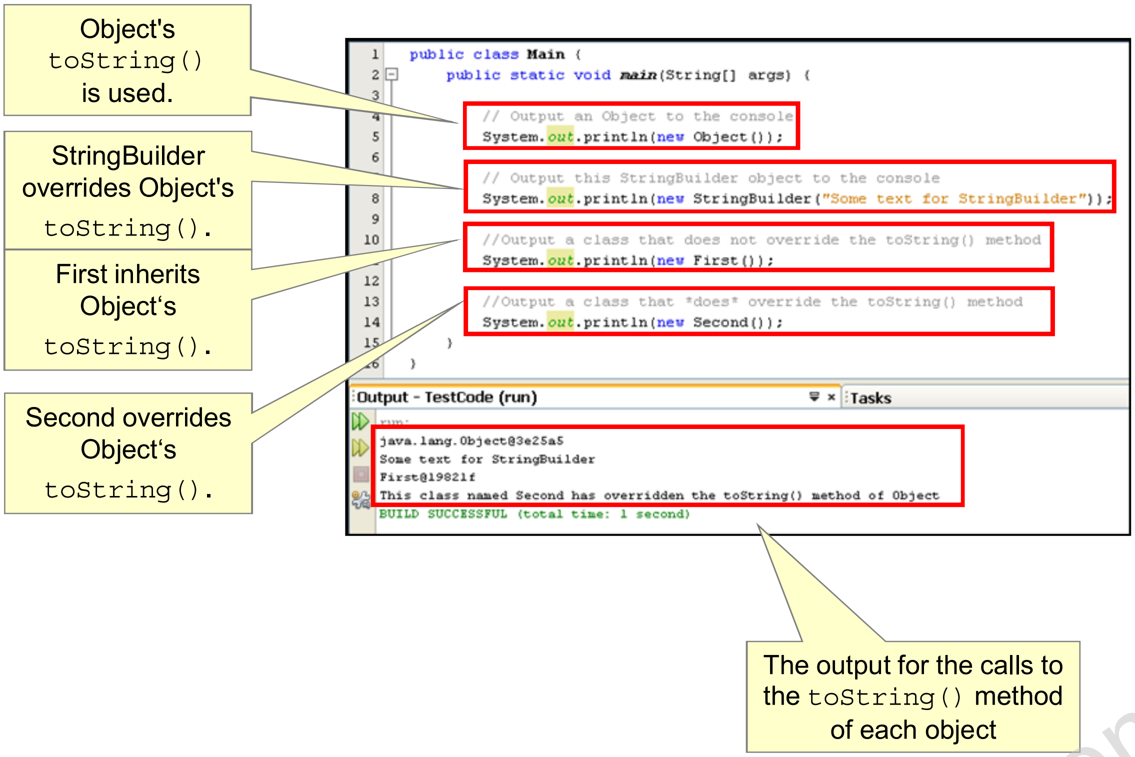
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

All classes have at the very top of their hierarchy the `Object` class. It is so central to how Java works that all classes that do not explicitly extend another class automatically extend `Object`. So all classes have `Object` at the root of their hierarchy. This means that all classes have access to the methods of `Object`. Being the root of the object hierarchy, `Object` does not have many methods—only very basic ones that all objects must have.

An interesting method is the `toString()` method. The `Object toString()` method gives very basic information about the object; generally classes will override the `toString()` method to provide more useful output. `System.out.println()` uses the `toString()` method on an object passed to it to output a string representation.

Oracle Internal & Oracle Academy Use Only

Calling the `toString()` Method



Object's `toString()` is used.

StringBuilder overrides Object's `toString()`.

First inherits Object‘s `toString()`.

Second overrides Object‘s `toString()`.

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

        // Output an Object to the console
        System.out.println(new Object());

        // Output this StringBuilder object to the console
        System.out.println(new StringBuilder("Some text for StringBuilder"));

        //Output a class that does not override the toString() method
        System.out.println(new First());

        //Output a class that *does* override the toString() method
        System.out.println(new Second());
    }
}
```

```
run:
java.lang.Object@3e25a5
Some text for StringBuilder
First@19821f
This class named Second has overridden the toString() method of Object
BUILD SUCCESSFUL (total time: 1 second)
```

The output for the calls to the `toString()` method of each object

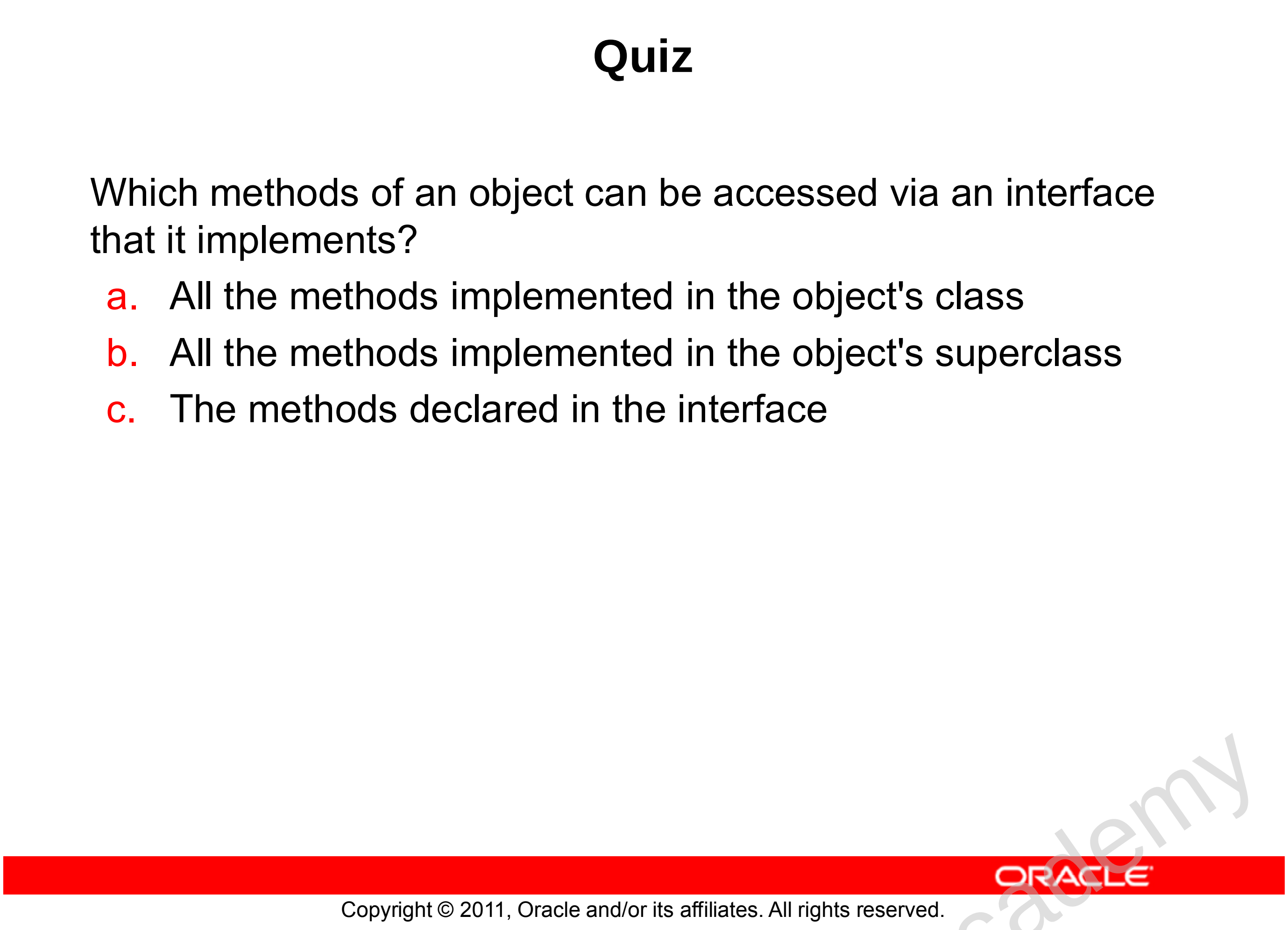
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

All objects have a `toString()` method because it exists in the `Object` class. But the `toString()` method may return different results depending on whether or not that method has been overridden. In the example in the slide, `toString()` is called (via the `println()` method of `System.out`) on four objects:

- **An Object object:** This calls the `toString()` method of the base class. It returns the name of the class (`java.lang.Object`), an `@` symbol, and a hash value of the object (a unique number associated with the object.
- **A StringBuilder object:** This calls the `toString()` method on the StringBuilder object. StringBuilder overrides the `toString()` method that it inherits from Object to return a String object of the set of characters it is representing.
- **An object of type First, a test class:** First is a class with no code, so the `toString()` method called is the one that is inherited from the Object class.
- **An object of type Second, a test class:** Second is a class with one method named `toString()`, so this overridden method will be the one that is called.

There is a case for reimplementing the `getDescription()` method used by the Clothing classes to instead use an overridden `toString()` method.



Quiz

Which methods of an object can be accessed via an interface that it implements?

a. All the methods implemented in the object's class
b. All the methods implemented in the object's superclass
c. The methods declared in the interface

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned the following:

- Creating class hierarchies with subclasses and superclasses helps to create extensible and maintainable code by:
  - Generalizing and abstracting code that may otherwise be duplicated
  - Using polymorphism
- Creating interfaces:
  - Allows you to link classes in different object hierarchies by their common behavior
  - Use an Interface reference type in your code so that the implementing class can be changed more easily.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Inheritance enables programmers to put common members (variables and methods) in one class and have other classes *inherit these common members* from this new class.

The class containing members common to several other classes is called the *superclass* or the *parent class*. The classes that inherit from, or extend, the superclass are called *subclasses* or *child classes*.

Inheritance also allows object methods and fields to be referred to by a reference that is the type of the object, the type of any of its superclasses, or an interface that it implements.

Finally, inheritance enables polymorphism.

Oracle Internal & Oracle Academy Use Only

Practice 12-1 Overview: Creating and Using Superclasses and Subclasses

In this practice, you design and then create a class hierarchy that will form the basis for an Employee Tracking System of the Marketing department in the Duke's Choice company.

During the practice, you:

- Create a simple design model for the class hierarchy
- Create the actual classes and test them

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 12-2 Overview: Using a Java Interface

In this practice, you create an interface. During the practice, you:

- Create an interface called Printable and implement it within the class hierarchy that you built in Practice 11-1
- Examine and run another small application that uses the same Printable interface

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

13

Handling Errors

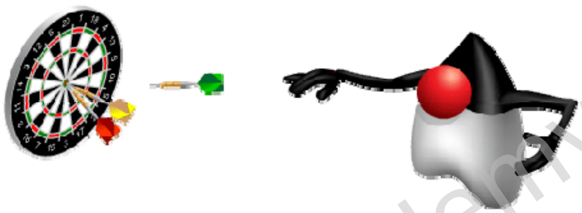
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Describe the different kinds of errors that can occur and how they are handled in Java
- Describe what exceptions are used for in Java
- Determine what exceptions are thrown, for any foundation class
- Write code to handle an exception thrown by the method of a foundation class



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Reporting Exceptions

Coding mistake:

```
int[] intArray = new int[5];
intArray[5] = 27;
```

Output in console:

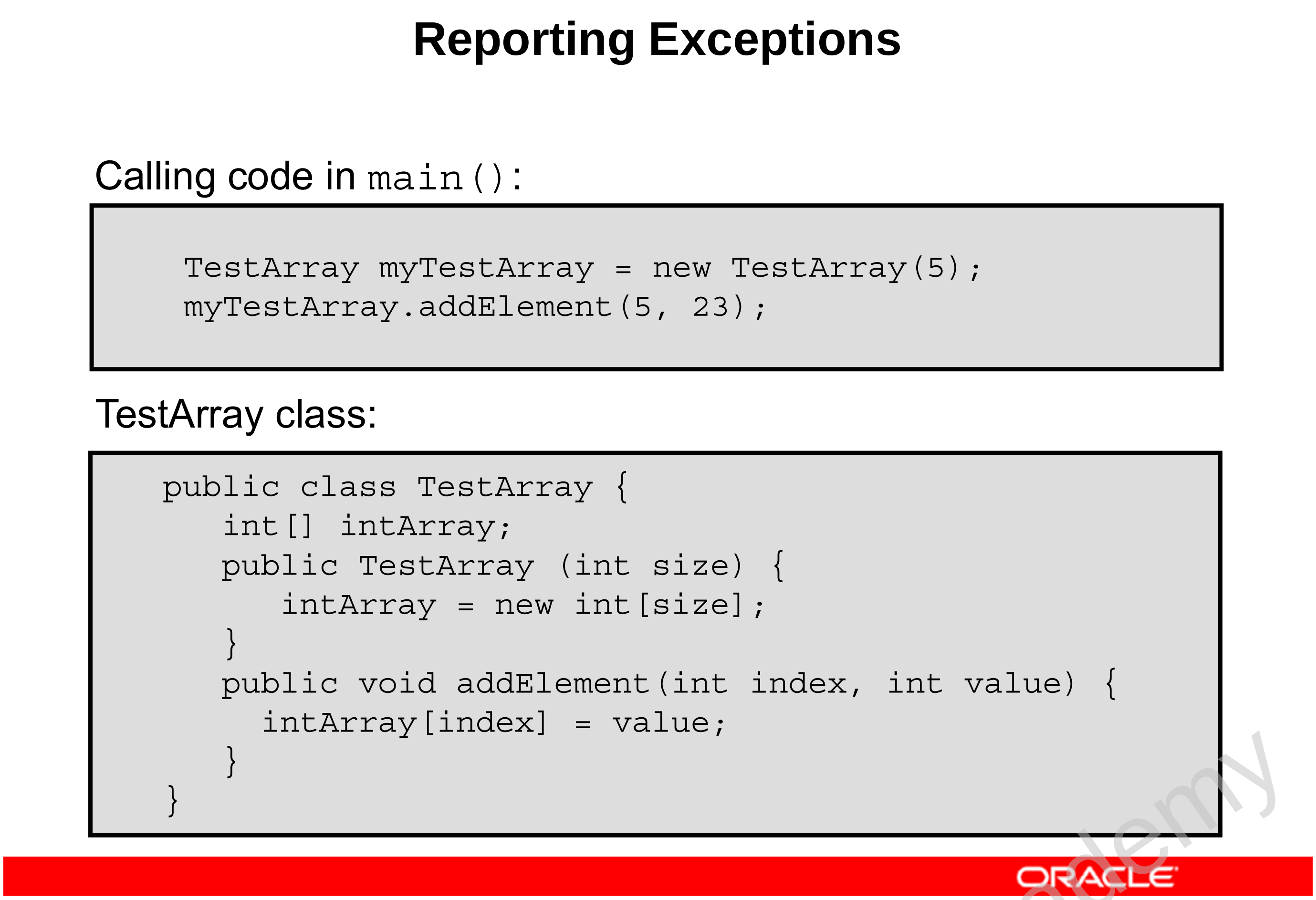
```
Exception in thread "main"
   java.lang.ArrayIndexOutOfBoundsException: 5
       at TestErrors.main(TestErrors.java:17)
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You may have come across the error shown in the slide while working on some of the previous practice activities. The code shows a common mistake made when accessing an array. Remember that arrays are zero based (the first element is accessed by a zero index), so in an array like the one in the slide that has five elements, the last element is actually intArray[4].

intArray[5] tries to access an element that does not exist, and Java responds to this programming mistake by printing the text shown in the console.

Oracle Internal & Oracle Academy Use Only



Reporting Exceptions

Calling code in `main()`:

```
TestArray myTestArray = new TestArray(5);
myTestArray.addElement(5, 23);
```

TestArray class:

```
public class TestArray {
   int[] intArray;
   public TestArray (int size) {
      intArray = new int[size];
   }
   public void addElement(int index, int value) {
     intArray[index] = value;
   }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Here is a very similar example, except this time the code that creates the array and tries to assign a value to a nonexistent element has been moved to a different class. Notice how the error message in the console is almost identical to the previous example, but this time the methods `main()` in TestException and `addElement()` in TestArray are listed.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at TestArray.addElement(TestArray.java:19)
    at TestException.main(TestException.java:20)
Java Result: 1
```

In this lesson, you learn why that message is printed to the console. You also learn how you can catch or trap the message so that it is not printed to the console, and what other kinds of errors are reported by Java.

Oracle Internal & Oracle Academy Use Only

How Exceptions Are Thrown

Normal program execution:

1. Caller method calls worker method.
2. Worker method does work.
3. Worker method completes work, and then execution returns to caller method.

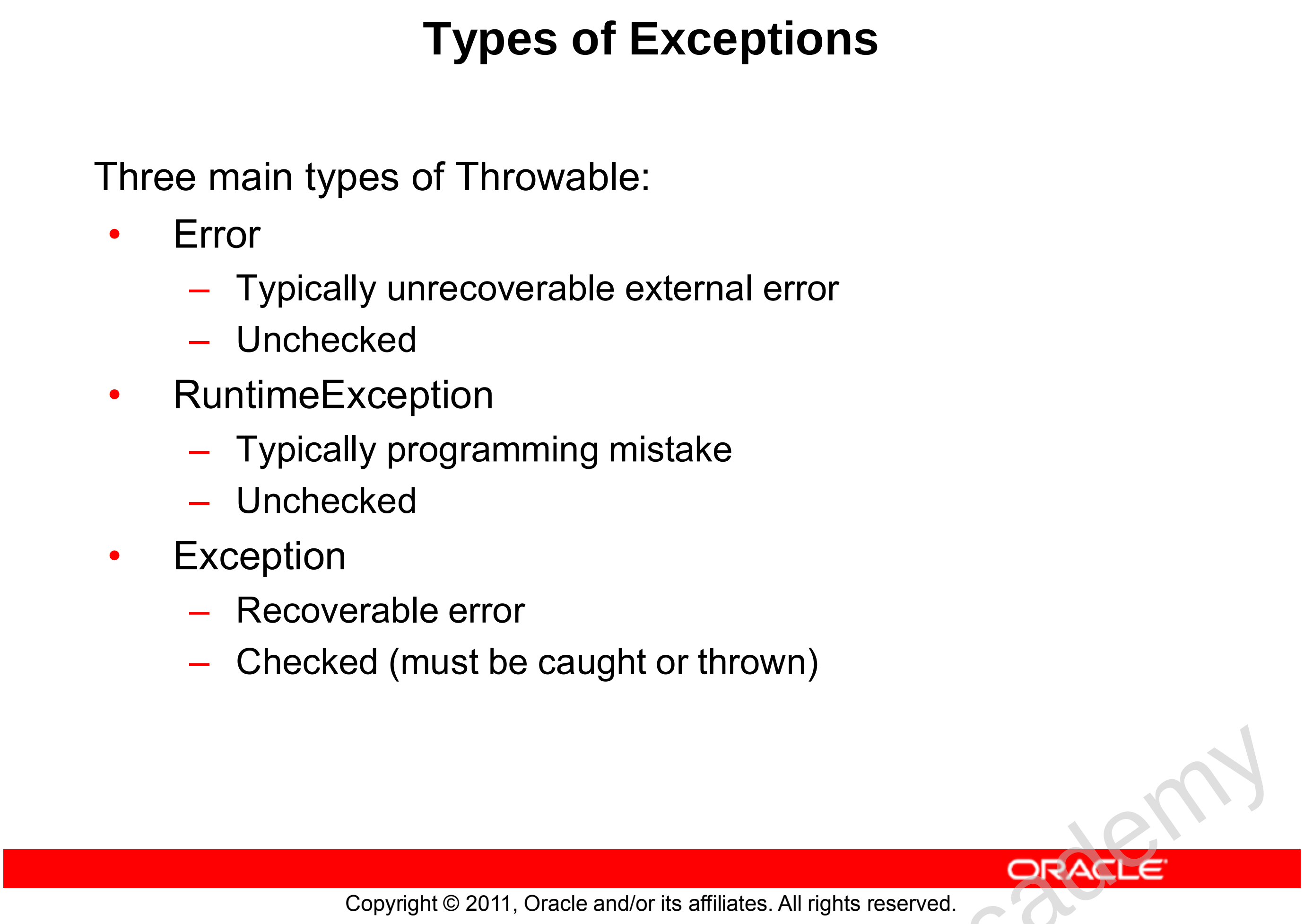
When an exception occurs, this sequence changes:

- Exception is thrown and either:
  - A special Exception object is passed to a special method-like catch block in the current method

    or
  - Execution returns to the caller method

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Types of Exceptions

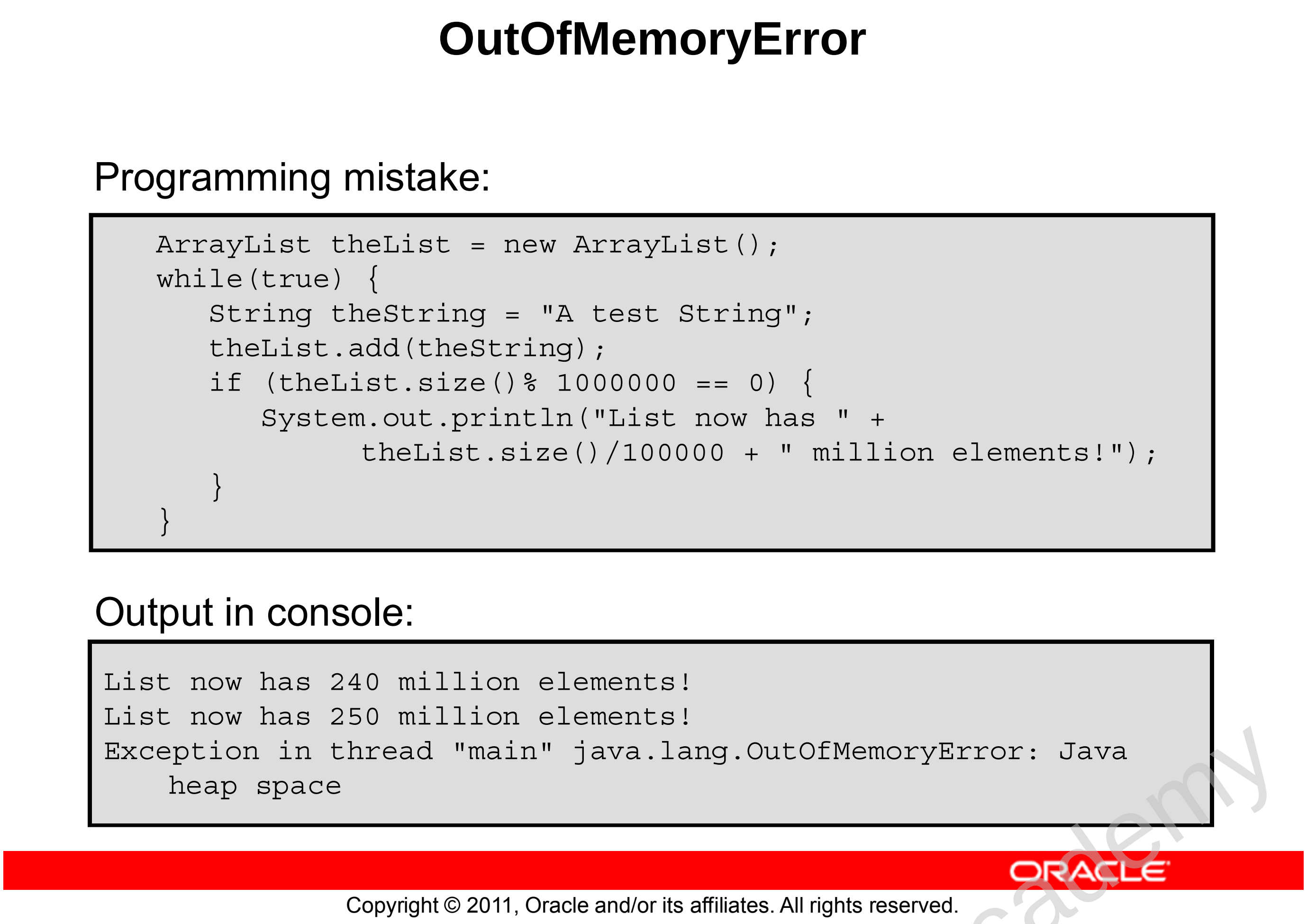
Three main types of Throwable:

- Error
  - Typically unrecoverable external error
  - Unchecked
- RuntimeException
  - Typically programming mistake
  - Unchecked
- Exception
  - Recoverable error
  - Checked (must be caught or thrown)

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

As mentioned in the previous slide, when an exception is thrown, that exception is an object that can be passed to a catch block. There are three main types of objects that can be thrown in this way, and all are derived from the class Throwable.

Oracle Internal & Oracle Academy Use Only



OutOfMemoryError

Programming mistake:

```
ArrayList theList = new ArrayList();
while(true) {
   String theString = "A test String";
   theList.add(theString);
   if (theList.size()% 1000000 == 0) {
      System.out.println("List now has " +
            theList.size()/100000 + " million elements!");
   }
}
```

Output in console:

```
List now has 240 million elements!
List now has 250 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java
    heap space
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

OutOfMemoryError is an error. Throwables of type Error are typically used for exceptional conditions that are external to the application and that the application usually cannot anticipate or recover from.

The example shown here has an infinite loop that continually adds an element to an ArrayList, guaranteeing that the JVM will run out of memory. The error is thrown up the call stack, and because it is not caught anywhere, it is displayed in the console as follows:

```
List now has 240 million elements!
List now has 250 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:2760)
        at java.util.Arrays.copyOf(Arrays.java:2734)
        at java.util.ArrayList.ensureCapacity(ArrayList.java:167)
        at java.util.ArrayList.add(ArrayList.java:351)
        at TestErrors.main(TestErrors.java:22)
```

Oracle Internal & Oracle Academy Use Only

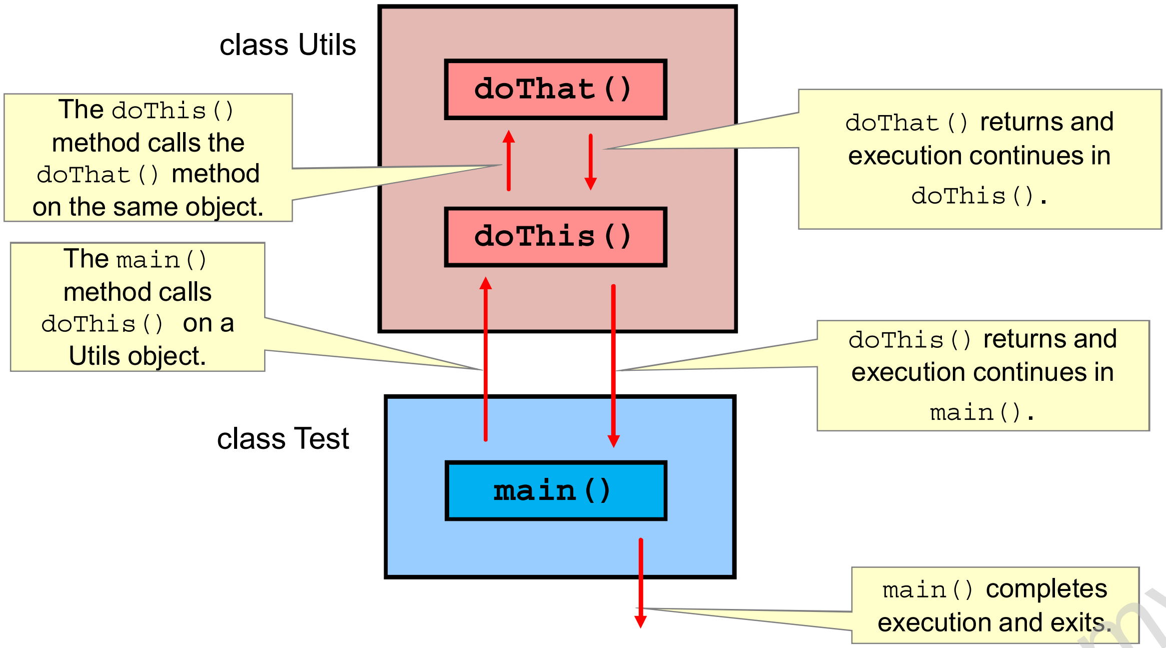
Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Method Stack


class Utils
doThat()
doThis()
The doThis() method calls the doThat() method on the same object.
doThat() returns and execution continues in doThis().
The main() method calls doThis() on a Utils object.
doThis() returns and execution continues in main().
class Test
main()
main() completes execution and exits.

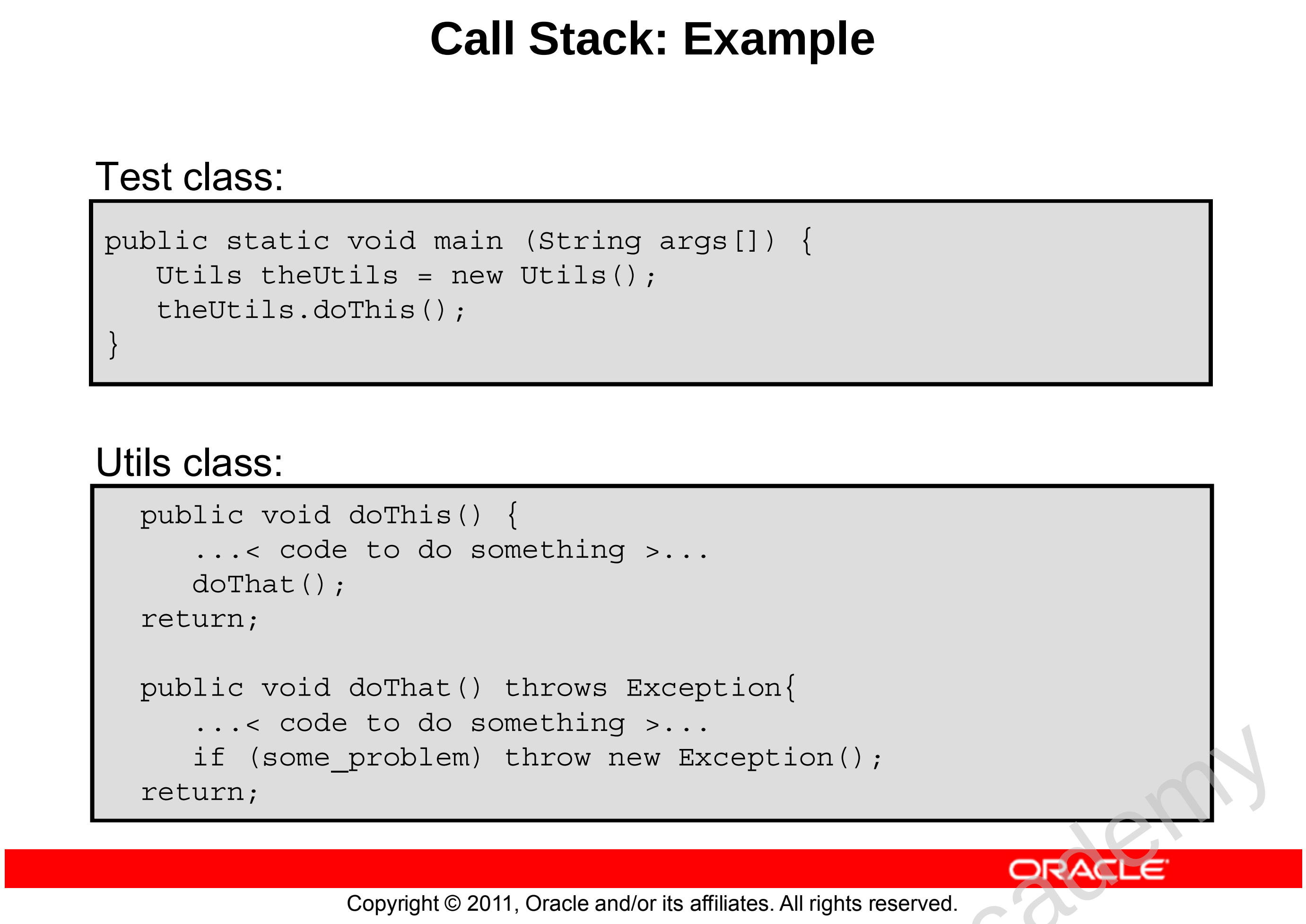
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

To understand exceptions, you need to think about how methods call other methods and how this can be nested deeply. The normal mode of operation is that a caller method calls a worker method, which in turn becomes a caller method and calls another worker method, and so on. This sequence of methods is called the *call stack*.

The example shown in the slide illustrates three methods in this relationship. The main method in the class Test (a static method) instantiates an object of type Utils and calls the method `doThis()` on that object. The `doThis()` method in turn calls a private method `doThat()` on the same object. When it comes to the end of its code or a return statement, each method returns execution to the method that called it.

Note that as far as how methods call and return and as far as how exceptions are thrown, the fact that there is one class method here and two instance methods on the same object is immaterial.

Oracle Internal & Oracle Academy Use Only



Call Stack: Example

Test class:

```
public static void main (String args[]) {
   Utils theUtils = new Utils();
   theUtils.doThis();
}
```

Utils class:

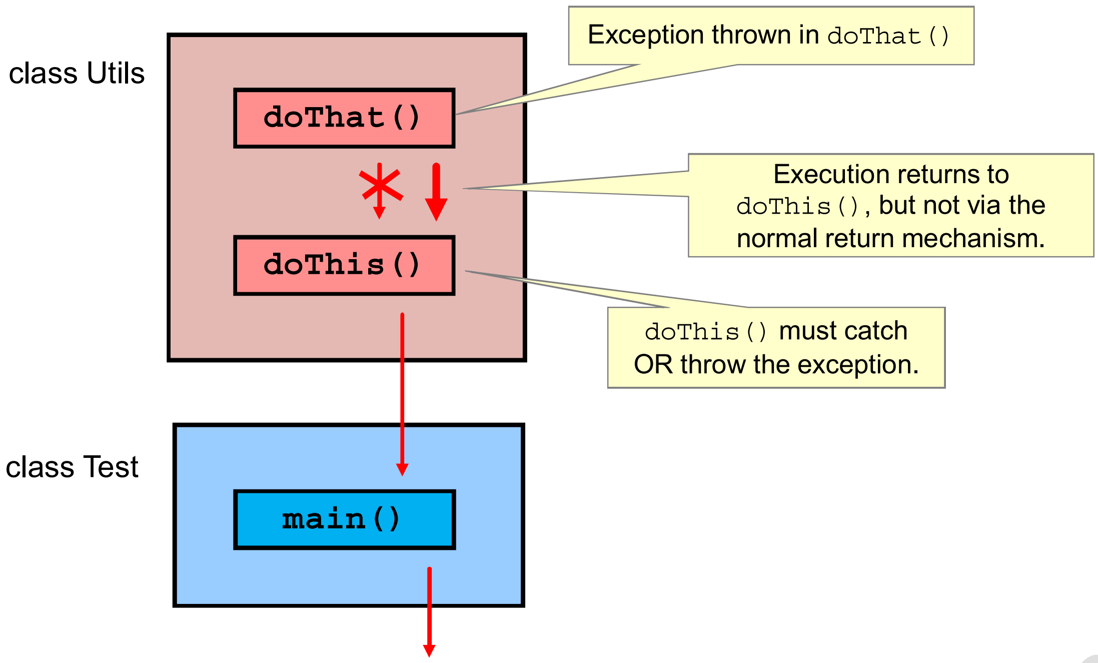
```
  public void doThis() {
     ...< code to do something >...
     doThat();
  return;

  public void doThat() throws Exception{
     ...< code to do something >...
     if (some_problem) throw new Exception();
  return;
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code shown in this slide is possible code for the example illustrated in the previous slide.

Oracle Internal & Oracle Academy Use Only

Throwing Throwables



Exception thrown in doThat()

Execution returns to doThis(), but not via the normal return mechanism.

doThis() must catch OR throw the exception.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

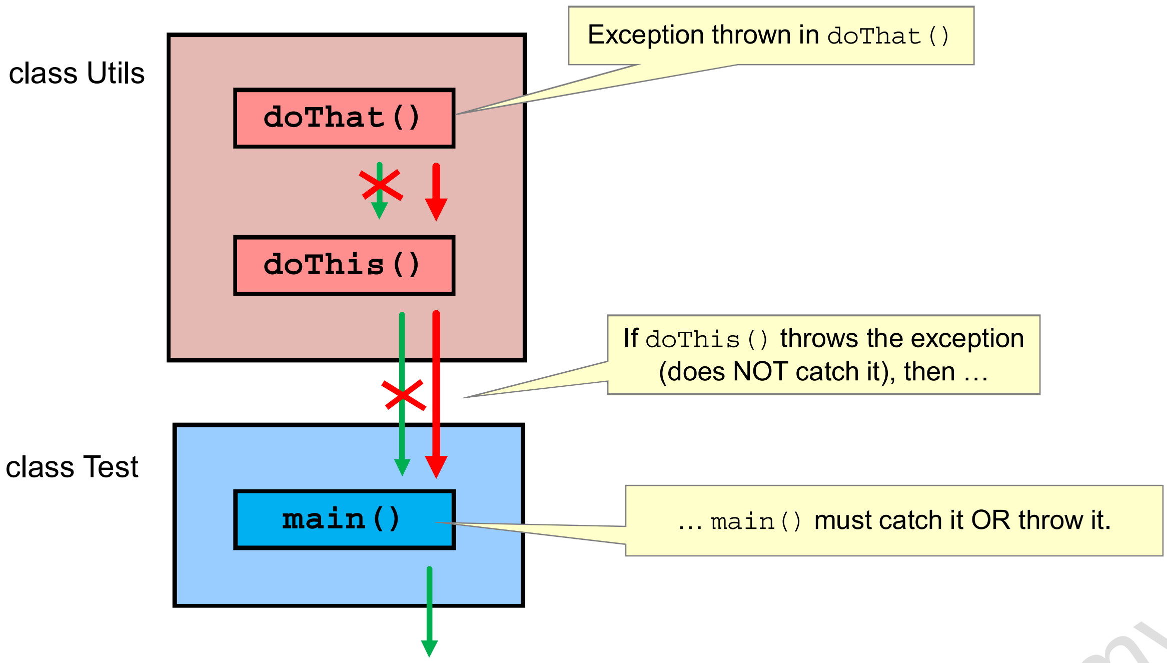
Oracle Internal & Oracle Academy Use Only

When a method finishes executing, the normal flow (on completion of the method or on a return statement) goes back to the calling method and continues execution at the next line of the calling method.

When an exception is thrown, program flow returns to the calling method, but not to the point just after the method call. Instead, if there is a try/catch block, it is thrown back to the catch block that is associated with the try block that contains the method call. If there is no try/catch block in the calling method, the exception is thrown back to its calling method.

In the case of a checked exception, this happens because the programmer is forced to explicitly throw the exception if the programmer chose not to catch it. In the case of an exception that is a RuntimeException or an error, the throwing of the exception happens automatically where no try/catch exists.

Throwing Throwables



Exception thrown in `doThat()`

If `doThis()` throws the exception (does NOT catch it), then …

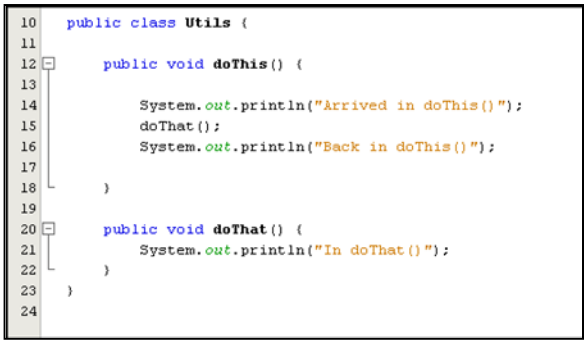
… `main()` must catch it OR throw it.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide illustrates an exception originally thrown in `doThat()` being thrown to `doThis()`. The error is not caught there, so it is thrown to its caller method, which is the main method.

Oracle Internal & Oracle Academy Use Only

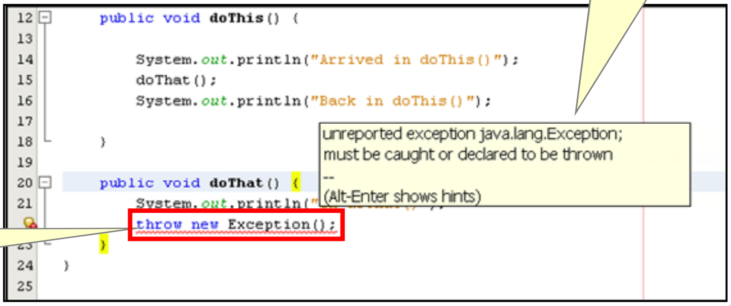
Working with Exceptions in NetBeans



```
public class Utils {

    public void doThis() {

        System.out.println("Arrived in doThis()");
        doThat();
        System.out.println("Back in doThis()");

    }

    public void doThat() {
        System.out.println("In doThat()");
    }
}

```

No exceptions thrown; nothing needs be done to deal with them.

NetBeans uses a tooltip to give you your two options.



```
    public void doThis() {

        System.out.println("Arrived in doThis()");
        doThat();
        System.out.println("Back in doThis()");

    }

    public void doThat() {
        System.out.println("In doThat()");
        throw new Exception();
    }
}

```

unreported exception java.lang.Exception;
must be caught or declared to be thrown
--
(Alt-Enter shows hints)

Throwing an exception within the method requires further steps.

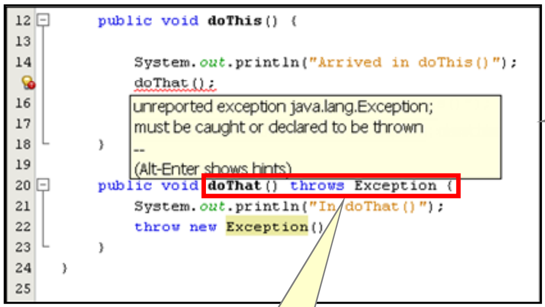
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Here you can see the code for the Utils class shown in NetBeans. In the first screenshot, no exceptions are thrown, so NetBeans shows no syntax or compilation errors. In the second screenshot, `doThat()` throws an exception, and NetBeans flags this as something that needs to be dealt with by the programmer. As you can see from the tooltip, it gives the two options that a programmer must choose from if handling checked exceptions.

In these early examples, for simplicity we use the Exception superclass. However, as you will see later, you should not throw so general an exception. Where possible, when you catch an exception, you should try to catch a specific exception.

Oracle Internal & Oracle Academy Use Only

Catching an Exception

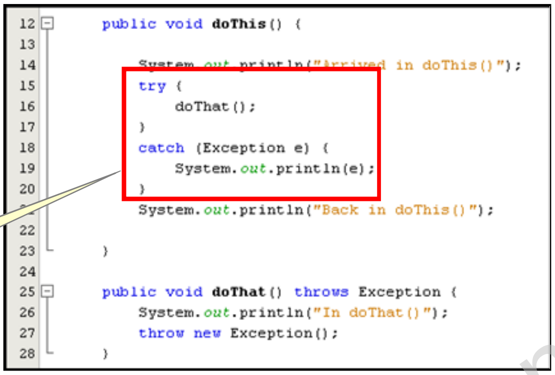


```
    public void doThis() {

        System.out.println("Arrived in doThis()");
        doThat();

    }

    public void doThat() throws Exception {
        System.out.println("In doThat()");
        throw new Exception()
    }
}
```

unreported exception java.lang.Exception;
must be caught or declared to be thrown
--
(Alt-Enter shows hints)

Now exception needs to be dealt with in `doThis()`.

`doThat()` now throws an exception.



```
    public void doThis() {

        System.out.println("Arrived in doThis()");
        try {
            doThat();
        }
        catch (Exception e) {
            System.out.println(e);
        }
        System.out.println("Back in doThis()");

    }

    public void doThat() throws Exception {
        System.out.println("In doThat()");
        throw new Exception();
    }
```

The try/catch block catches exception and handles it.

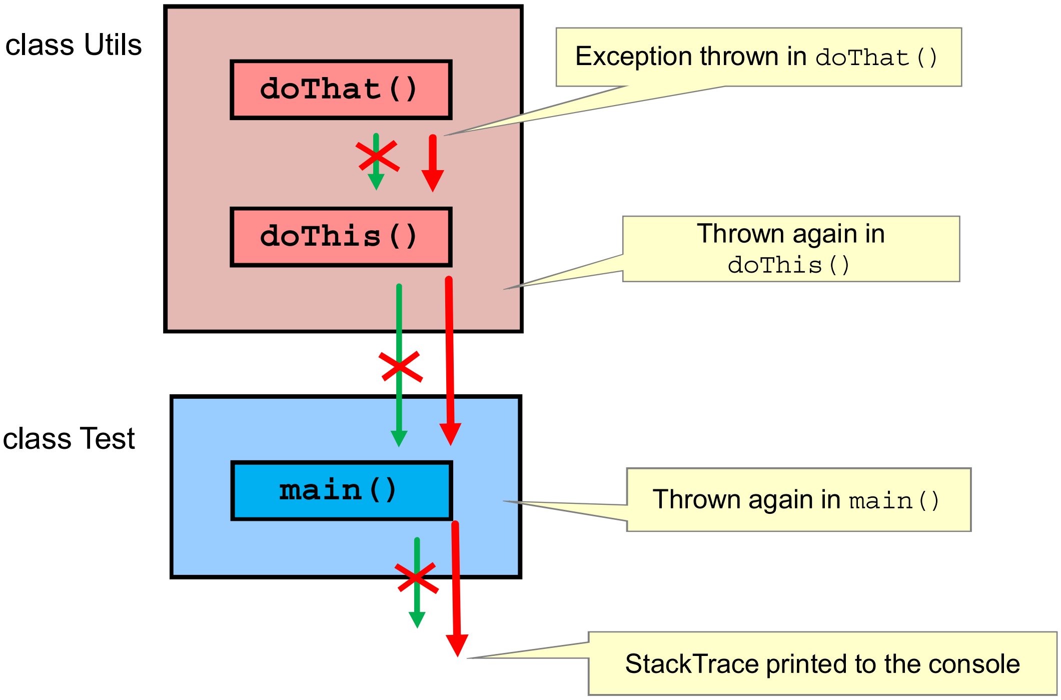
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Here you can see that the exception thrown in `doThat()` has been handled by:

- Adding `throws Exception` to the `doThat()` method signature, ensuring that it is thrown to the caller, `doThat()`
- Adding a try/catch block to `doThis()` so that:
  - The try block contains the call to `doThat()`
  - The catch block is set up with the parameter `Exception`

Oracle Internal & Oracle Academy Use Only

Uncaught Exception



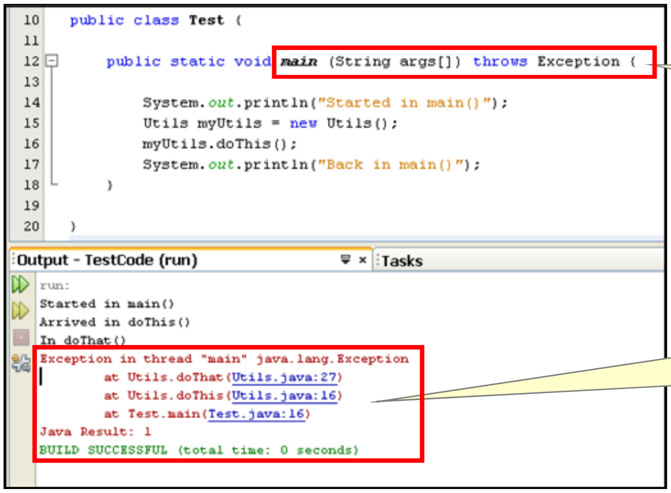
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

But what happens if none of the methods in the call stack have try/catch blocks? That situation is illustrated by the diagram shown in this slide. Because there are no try/catch blocks, the exception is thrown all the way up the call stack. But what does it mean to throw an exception from the `main()` method? This causes the program to exit, and the exception, plus a stack trace for the exception, is printed to the console.

Oracle Internal & Oracle Academy Use Only

Exception Printed to Console

Example of `main()` throwing exception



```
public class Test {

    public static void main (String args[]) throws Exception {

        System.out.println("Started in main()");
        Utils myUtils = new Utils();
        myUtils.doThis();
        System.out.println("Back in main()");
    }

}
```

```
run:
Started in main()
Arrived in doThis()
In doThat()
Exception in thread "main" java.lang.Exception
        at Utils.doThat(Utils.java:27)
        at Utils.doThis(Utils.java:16)
        at Test.main(Test.java:16)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

`main()` is now set up to throw exception.

Because `main()` throws the exception, it now prints call stack to console.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

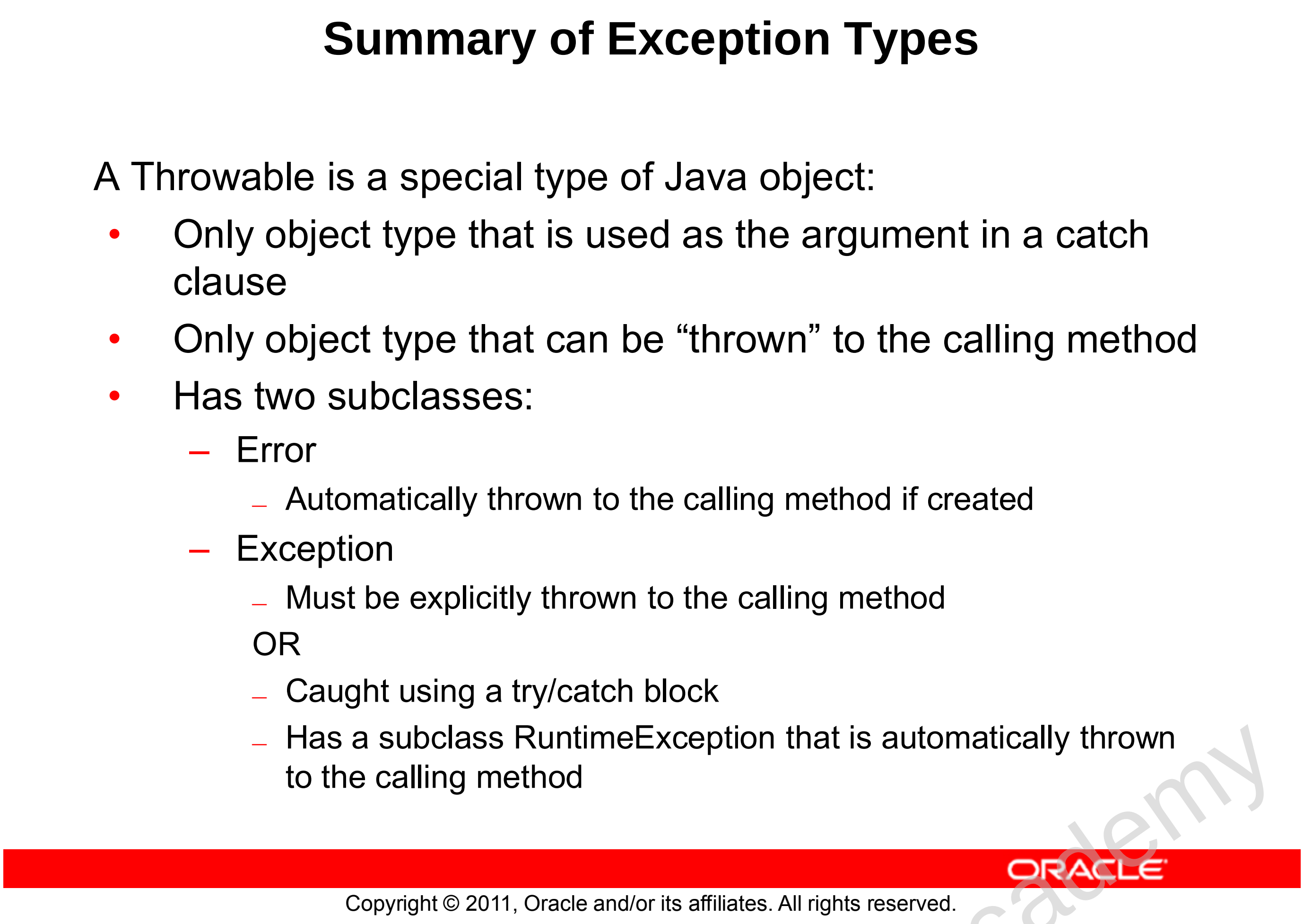
In the example, you can see what happens when the exception is thrown up the call stack all the way to the `main()` method, and it throws the exception too.

Did you notice how similar this looks to the first example you saw of an ArrayIndexOutOfBoundsException? In both cases, the exception is displayed as a stack trace to the console.

However, there was something different about the ArrayIndexOutOfBoundsException: None of the methods threw that exception! So how did it get passed up the call stack?

The answer is that ArrayIndexOutOfBoundsException is a RuntimeException. The RuntimeException class is a subclass of the Exception class. It has the additional functionality that its exceptions are automatically thrown up the call stack without this being explicitly declared in the method signature.

Oracle Internal & Oracle Academy Use Only



Summary of Exception Types

A Throwable is a special type of Java object:

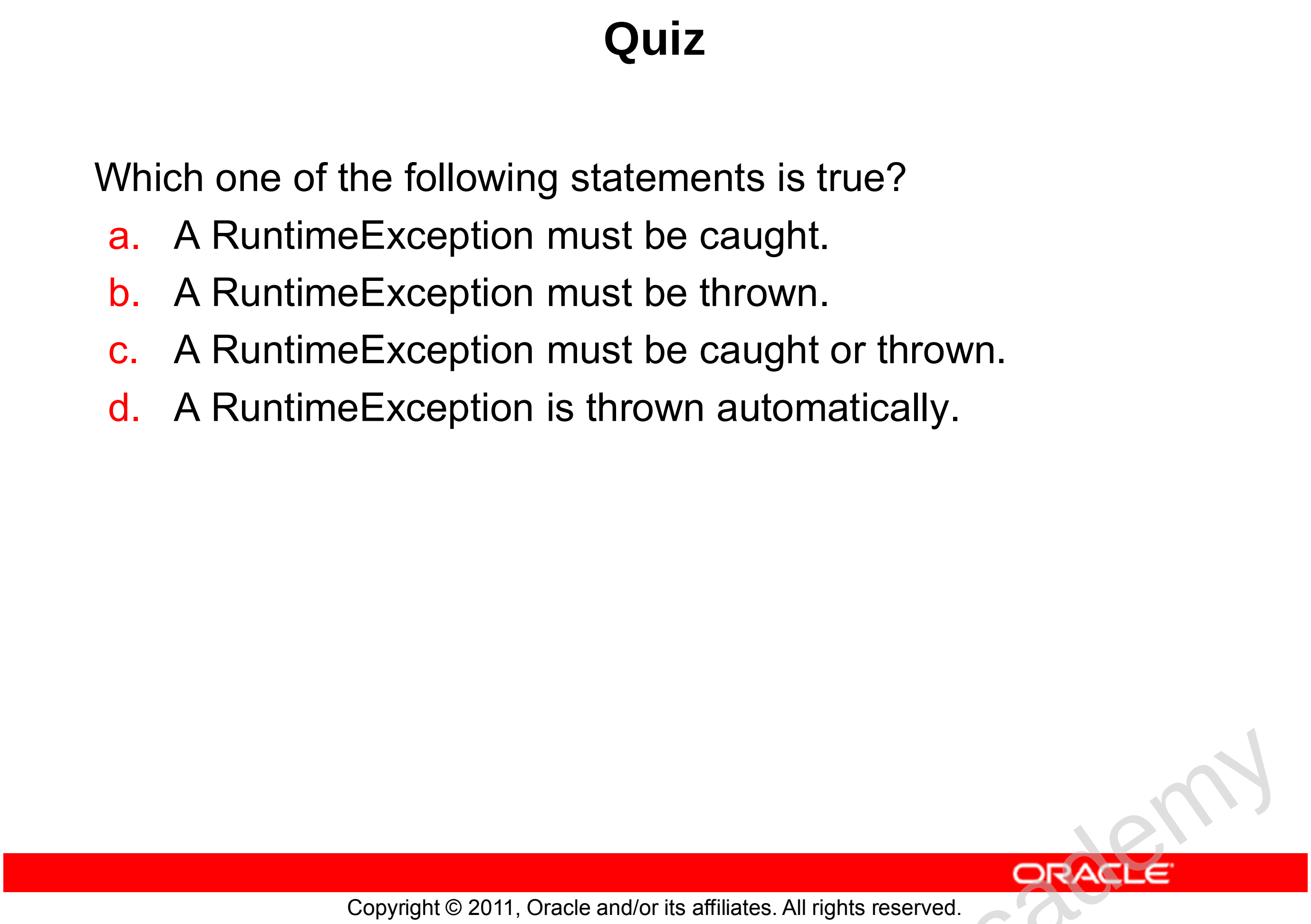
- Only object type that is used as the argument in a catch clause
- Only object type that can be “thrown” to the calling method
- Has two subclasses:
  - Error
    - Automatically thrown to the calling method if created
  - Exception
    - Must be explicitly thrown to the calling method

      OR
    - Caught using a try/catch block
    - Has a subclass RuntimeException that is automatically thrown to the calling method

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Exceptions that are not also RuntimeExceptions must be explicitly handled. The examples later in this lesson show you how to work with an IOException.

Oracle Internal & Oracle Academy Use Only



Quiz

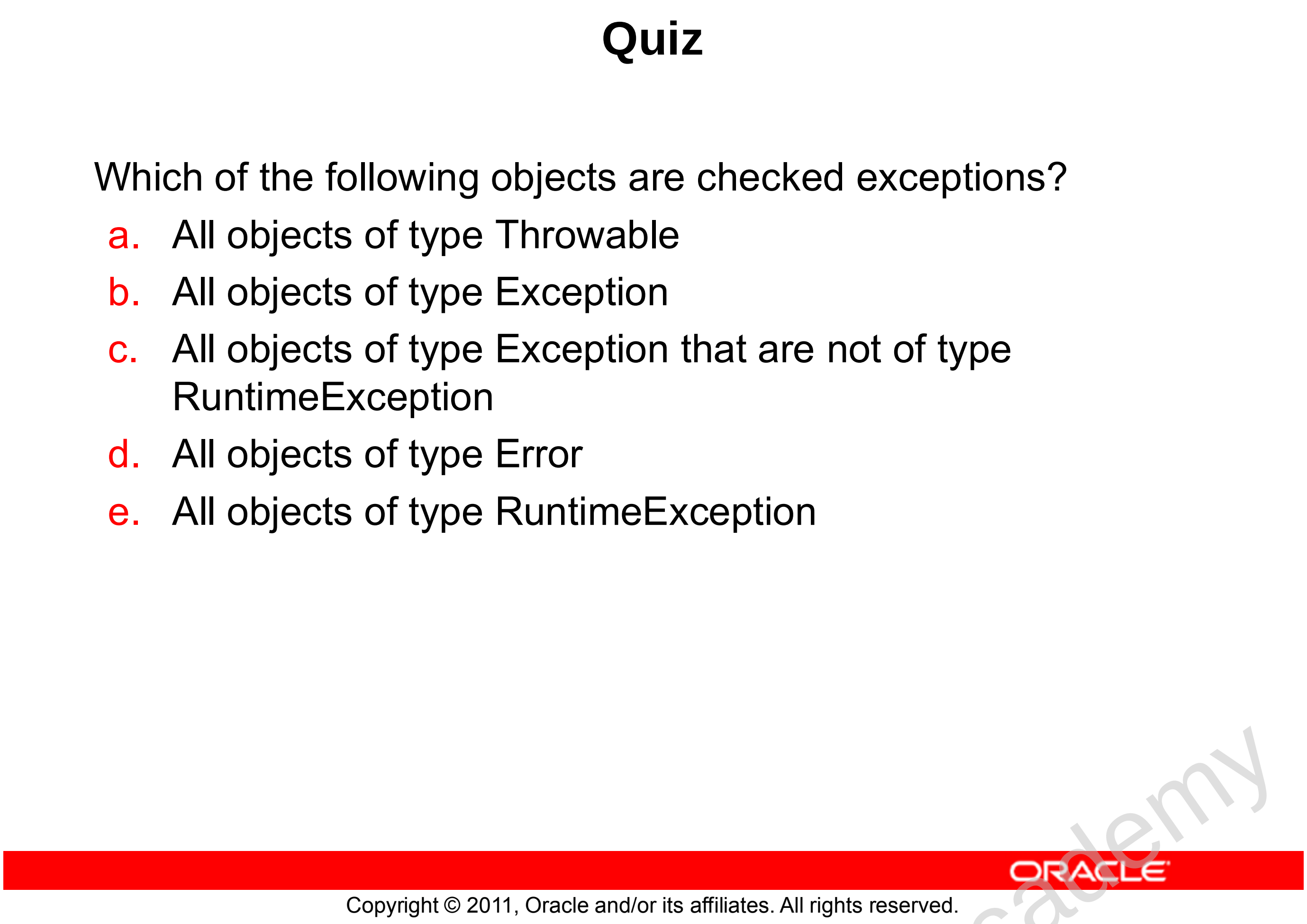
Which one of the following statements is true?

a. A RuntimeException must be caught.
b. A RuntimeException must be thrown.
c. A RuntimeException must be caught or thrown.
d. A RuntimeException is thrown automatically.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: d

Oracle Internal & Oracle Academy Use Only



Quiz

Which of the following objects are checked exceptions?

a. All objects of type Throwable
b. All objects of type Exception
c. All objects of type Exception that are not of type RuntimeException
d. All objects of type Error
e. All objects of type RuntimeException

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Answer: c

Oracle Internal & Oracle Academy Use Only

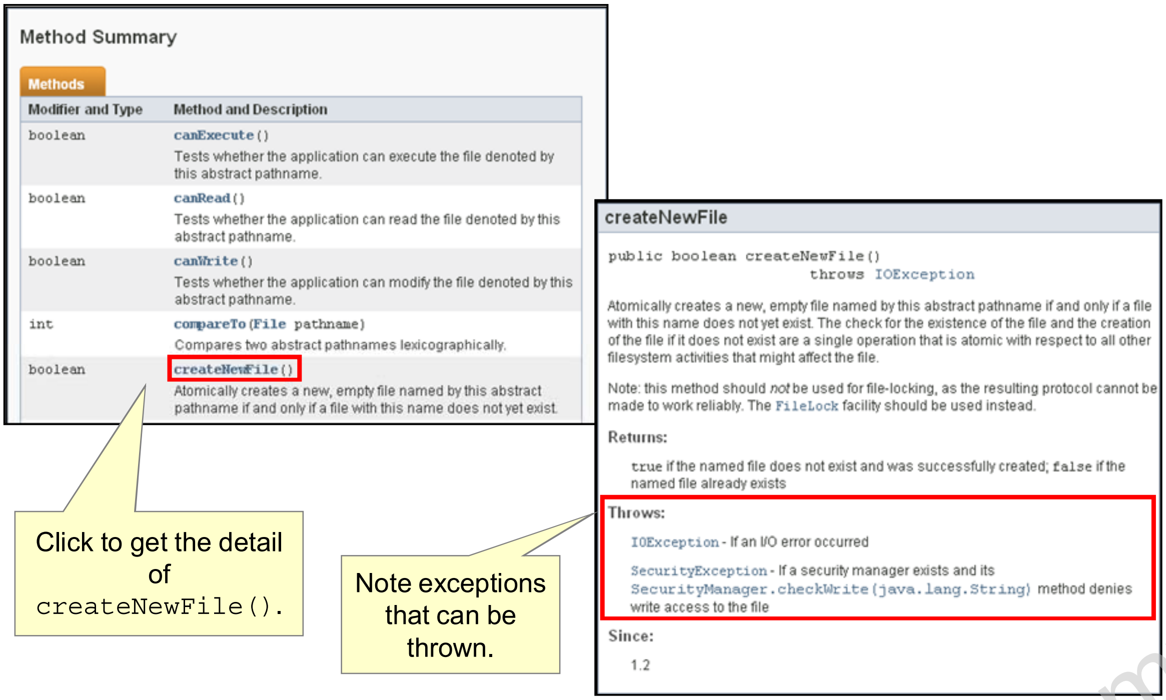
Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Exceptions in the Java API Documentation



Method Summary

Modifier and Type	Method and Description
boolean	canExecute() Tests whether the application can execute the file denoted by this abstract pathname.
boolean	canRead() Tests whether the application can read the file denoted by this abstract pathname.
boolean	canWrite() Tests whether the application can modify the file denoted by this abstract pathname.
int	compareTo(File pathname) Compares two abstract pathnames lexicographically.
boolean	createNewFile() Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

createNewFile

```
public boolean createNewFile()
                      throws IOException
```

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.

Note: this method should *not* be used for file-locking, as the resulting protocol cannot be made to work reliably. The FileLock facility should be used instead.

Returns:

true if the named file does not exist and was successfully created; false if the named file already exists

Throws:

IOException - If an I/O error occurred

SecurityException - If a security manager exists and its SecurityManager.checkWrite(java.lang.String) method denies write access to the file

Since:

1.2

Click to get the detail of `createNewFile()`.

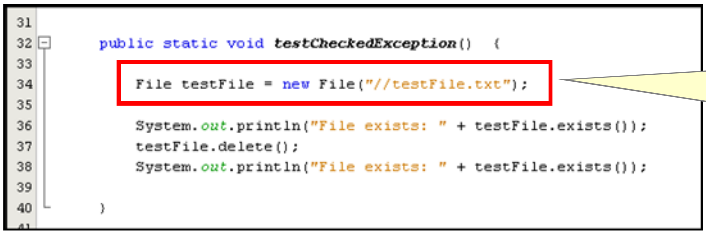
Note exceptions that can be thrown.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

When working with any API, it is necessary to determine what exceptions are thrown by the object's constructors or methods. The example in the slide is for the File class. File has a `createNewFile()` method that can throw an IOException or a SecurityException. SecurityException is a RuntimeException, so SecurityException is unchecked but IOException is a checked exception.

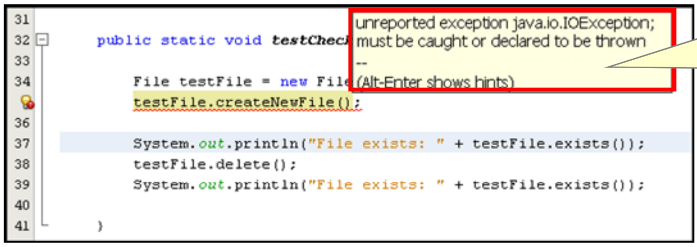
Oracle Internal & Oracle Academy Use Only

Calling a Method That Throws an Exception



```
public static void testCheckedException()  {

    File testFile = new File("//testFile.txt");

    System.out.println("File exists: " + testFile.exists());
    testFile.delete();
    System.out.println("File exists: " + testFile.exists());

}
```

Constructor causes no compilation problems.



```
public static void testChec

    File testFile = new File
    testFile.createNewFile();

    System.out.println("File exists: " + testFile.exists());
    testFile.delete();
    System.out.println("File exists: " + testFile.exists());

}
```

unreported exception java.io.IOException;
must be caught or declared to be thrown
--
(Alt-Enter shows hints)

`createNewFile()` can throw a checked exception, so it must throw or catch.

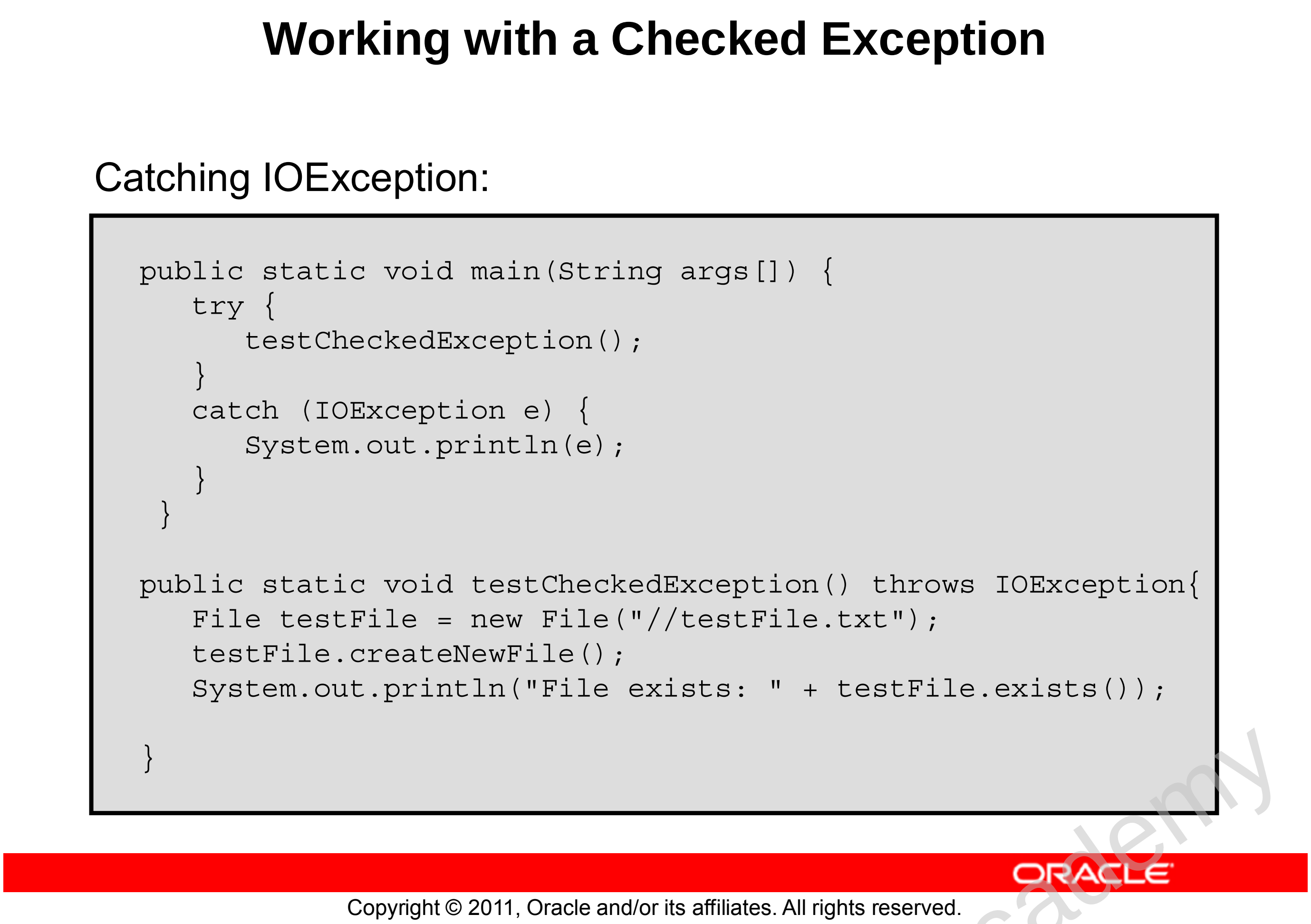
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The two screenshots in the slide show a simple `testCheckedException()` method. In the first example, the File object is created using the constructor. Note that even though the constructor can throw a NullPointerException (if the constructor argument is null), you are not forced to catch this exception.

However, in the second example, `createNewFile()` can throw an IOException, and NetBeans shows that you must deal with this.

Note that File is introduced here only to illustrate an IOException. In the next course (Programming 2), you learn about the File class and a new set of classes in the package `nio`, which provides more sophisticated ways to work with files.

Oracle Internal & Oracle Academy Use Only



Working with a Checked Exception

Catching IOException:

```
public static void main(String args[]) {
   try {
      testCheckedException();
   }
   catch (IOException e) {
      System.out.println(e);
   }
 }

public static void testCheckedException() throws IOException{
   File testFile = new File("//testFile.txt");
   testFile.createNewFile();
   System.out.println("File exists: " + testFile.exists());

}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide is handling the possible raised exception by:

- Throwing the exception from the `testCheckedException()` method
- Catching the exception in the caller method

In this example, the catch method catches the exception because the path to the text file is not correctly formatted. `System.out.println(e)` calls the `toString()` method of the exception, and the result is as follows:

```
java.io.IOException: The filename, directory name, or volume label
syntax is incorrect
```

Oracle Internal & Oracle Academy Use Only

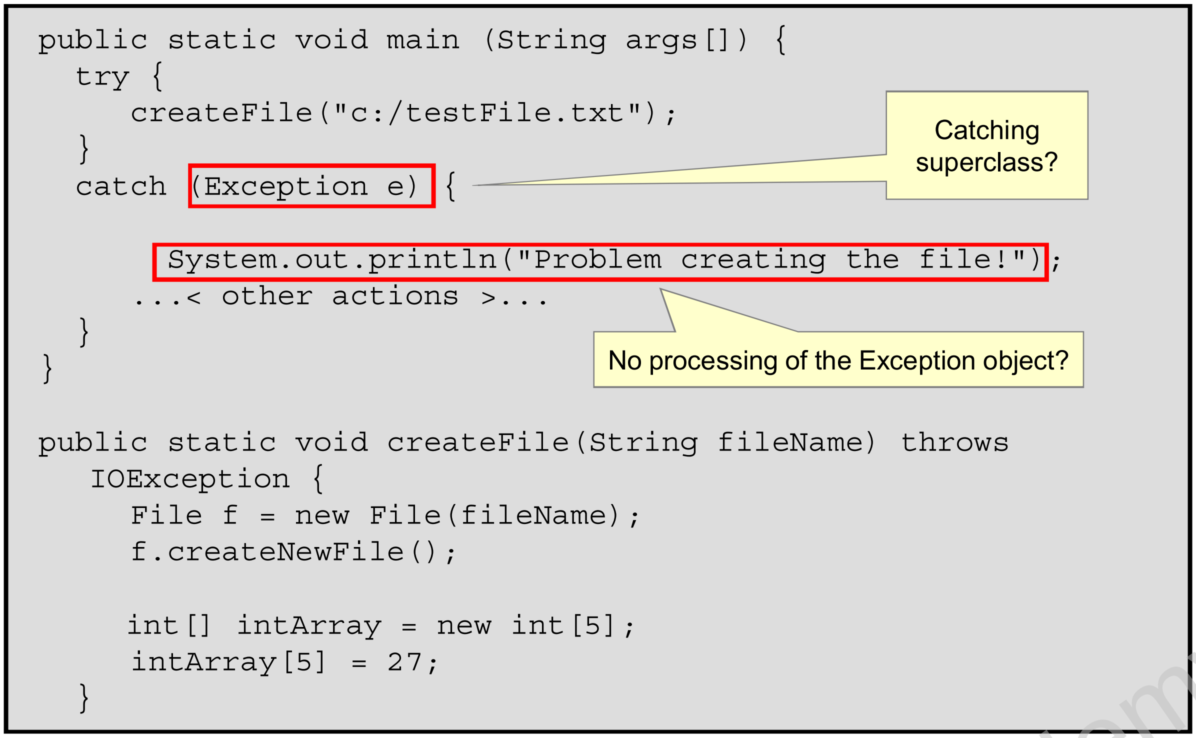
Best Practices

- Catch the actual exception thrown, not the exception or Throwable superclass.
- Examine the exception to find out the exact problem so you can recover cleanly.
- You do not need to catch every exception.
  - A programming mistake should not be handled. It must be fixed.
  - Ask yourself, "Does this exception represent behavior I want the program to recover from?"

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Bad Practices



```
public static void main (String args[]) {
  try {
     createFile("c:/testFile.txt");
  }
  catch (Exception e) {

      System.out.println("Problem creating the file!");
     ...< other actions >...
  }
}

public static void createFile(String fileName) throws
   IOException {
     File f = new File(fileName);
     f.createNewFile();

     int[] intArray = new int[5];
     intArray[5] = 27;
  }
```

Catching superclass?

No processing of the Exception object?

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code in the slide illustrates two poor programming practices.

1. The `catch` clause catches an exception rather than the expected exception from calling the `createFile` method (IOException).
2. The catch clause does not analyze the Exception object and instead simply assumes that the expected exception has been thrown from the File object.

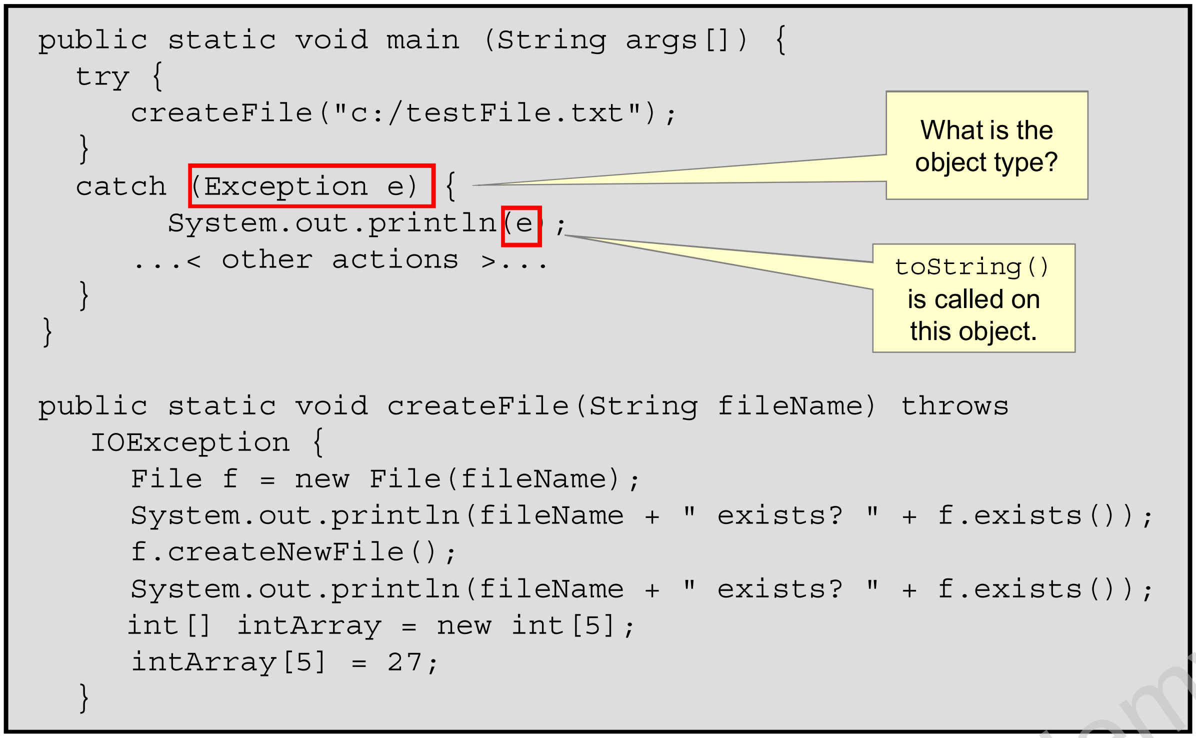
A major drawback of this careless programming style is shown by the fact that the code prints the following message to the console:

```
There is a problem creating the file!
```

This suggests that the file has not been created, and indeed any further code in the catch block will run. But what is actually happening in the code?

Oracle Internal & Oracle Academy Use Only

Bad Practices



```
public static void main (String args[]) {
  try {
     createFile("c:/testFile.txt");
  }
  catch (Exception e) {
         System.out.println(e);
     ...< other actions >...
  }
}

public static void createFile(String fileName) throws
   IOException {
     File f = new File(fileName);
     System.out.println(fileName + " exists? " + f.exists());
     f.createNewFile();
     System.out.println(fileName + " exists? " + f.exists());
     int[] intArray = new int[5];
     intArray[5] = 27;
  }
```

What is the object type?

`toString()` is called on this object.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Putting in a few `System.out.println()` calls in the `createFile` method may help clarify what is happening. The output now is:

```
C:/testFile.txt exists? false
C:/testFile.txt exists? true
java.lang.ArrayIndexOutOfBoundsException: 5
```

So the file is being created! And you can see that the exception is actually an ArrayIndexOutOfBoundsException that is being thrown by the final line of code in `createFile()`.

In this example, it is obvious that the array assignment can throw an exception, but it may not be so obvious. In this case, the `createNewFile()` method of File actually throws another exception—a SecurityException. Because it is an unchecked exception, it is thrown automatically.

If you check for the specific exception in the `catch` clause, you remove the danger of assuming what the problem is.

Oracle Internal & Oracle Academy Use Only

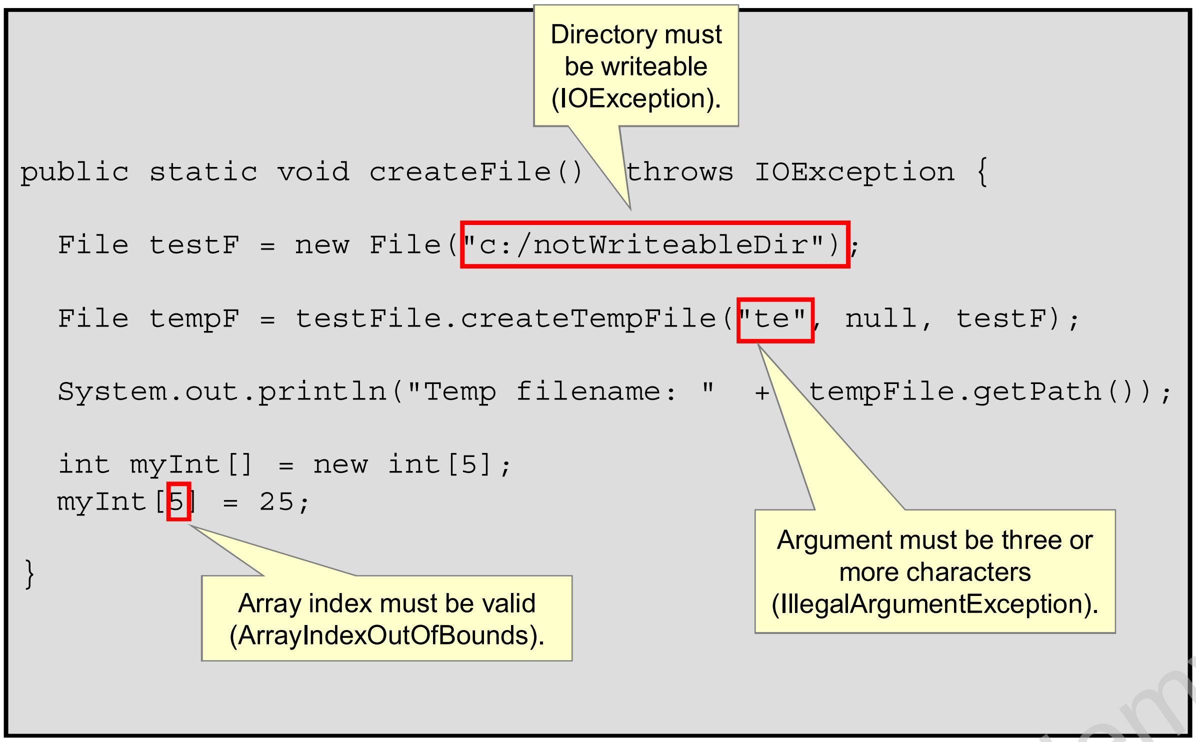
Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Multiple Exceptions



Directory must be writeable (IOException).

```
public static void createFile()  throws IOException {

   File testF = new File("c:/notWriteableDir");

   File tempF = testFile.createTempFile("te", null, testF);

   System.out.println("Temp filename: "  + tempFile.getPath());

   int myInt[] = new int[5];
   myInt[5] = 25;

}
```

Argument must be three or more characters (IllegalArgumentException).

Array index must be valid (ArrayIndexOutOfBounds).

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows a method that could potentially throw three different exceptions. It uses the `createTempFile()` File method, which creates a temporary file. (It ensures that each call creates a new and different file and also can be set up so that the temporary files created are deleted on exit.)

The three different exceptions are the following:

`IOException`

`c:\notWriteableDir` is a directory, but it is not writeable. This causes `createTempFile()` to throw an IOException (checked).

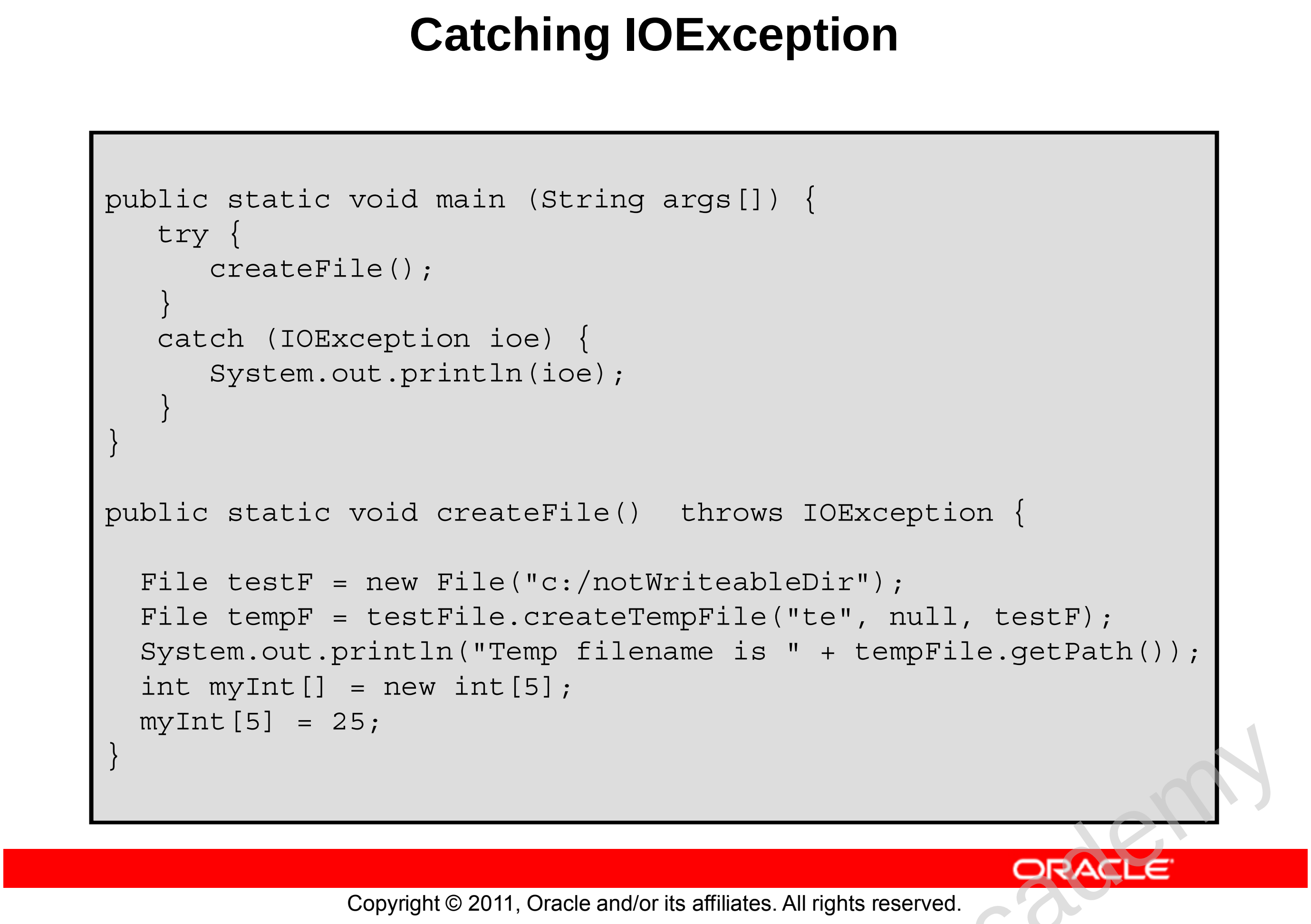
`IllegalArgumentException`

The first argument passed to createTempFile should be three or more characters long. If it is not, the method throws an IllegalArgumentException (unchecked).

`ArrayIndexOutOfBoundsException`

As in previous examples, trying to access a nonexistent index of an array throws an ArrayIndexOutOfBoundsException (unchecked).

Oracle Internal & Oracle Academy Use Only



Catching IOException

```
public static void main (String args[]) {
   try {
      createFile();
   }
   catch (IOException ioe) {
      System.out.println(ioe);
   }
}

public static void createFile()  throws IOException {

  File testF = new File("c:/notWriteableDir");
  File tempF = testFile.createTempFile("te", null, testF);
  System.out.println("Temp filename is " + tempFile.getPath());
  int myInt[] = new int[5];
  myInt[5] = 25;
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows the minimum exception handling (the compiler insists on at least the IOException being handled).

With the directory set as shown at `c:/notWriteableDir`, the output of this code is:

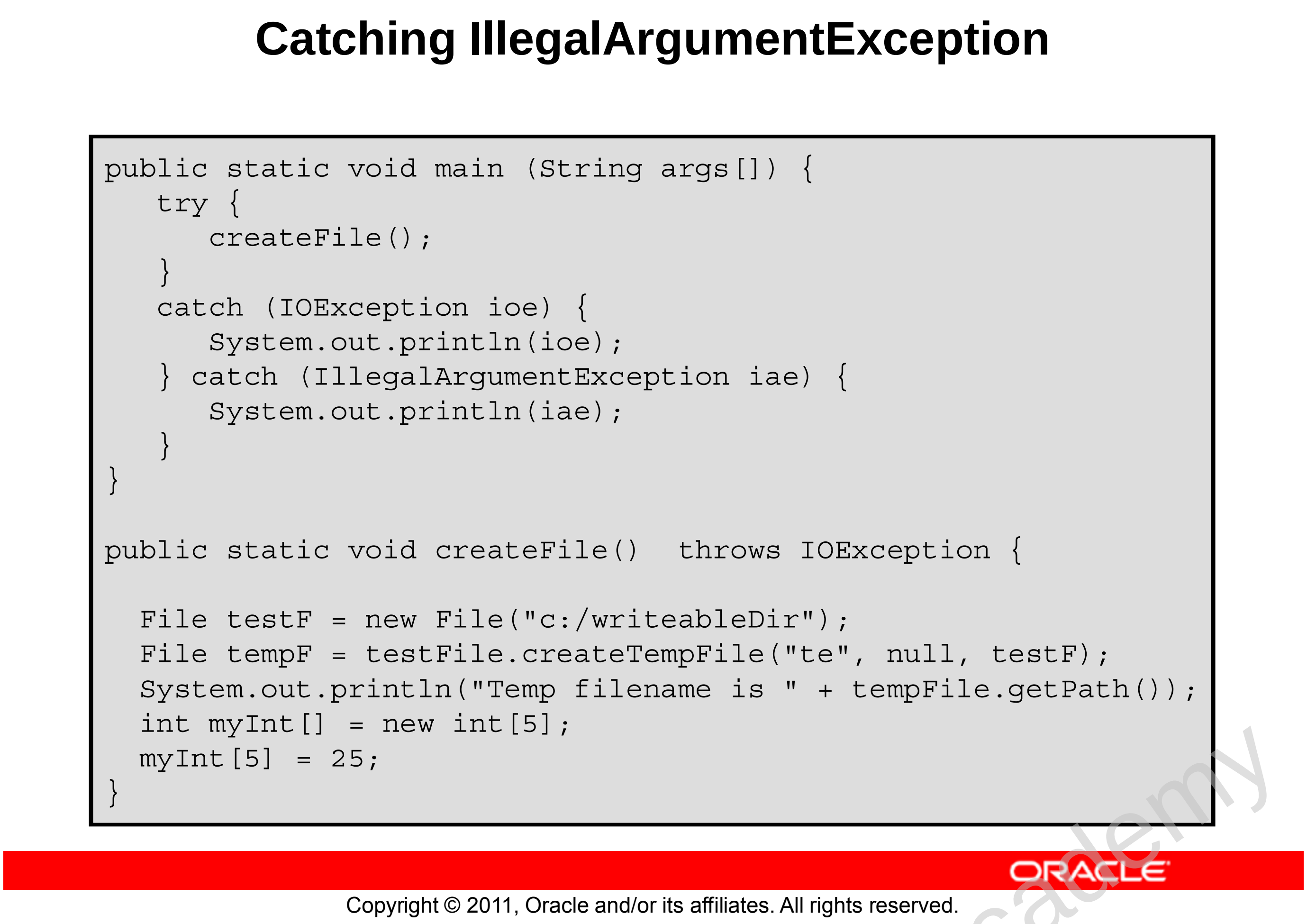
```
java.io.IOException: Permission denied
```

However, if the file is set as `c:/writeableDir` (a writeable directory), the output is now:

```
Exception in thread "main" java.lang.IllegalArgumentException: Prefix
string too short
   at java.io.File.createTempFile(File.java:1782)
   at
MultipleExceptionExample.createFile(MultipleExceptionExample.java:34)
   at MultipleExceptionExample.main(MultipleExceptionExample.java:18)
```

The argument `"te"` causes an IllegalArgumentException to be thrown, and because it is a RuntimeException, it gets thrown all the way out to the console.

Oracle Internal & Oracle Academy Use Only



Catching IllegalArgumentException

```
public static void main (String args[]) {
   try {
      createFile();
   }
   catch (IOException ioe) {
      System.out.println(ioe);
   } catch (IllegalArgumentException iae) {
      System.out.println(iae);
   }
}

public static void createFile()  throws IOException {

  File testF = new File("c:/writeableDir");
  File tempF = testFile.createTempFile("te", null, testF);
  System.out.println("Temp filename is " + tempFile.getPath());
  int myInt[] = new int[5];
  myInt[5] = 25;
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows an additional `catch` clause added to catch the potential IllegalArgumentException.

With the first argument of the `createTempFile()` method set to `"te"` (fewer than three characters), the output of this code is:

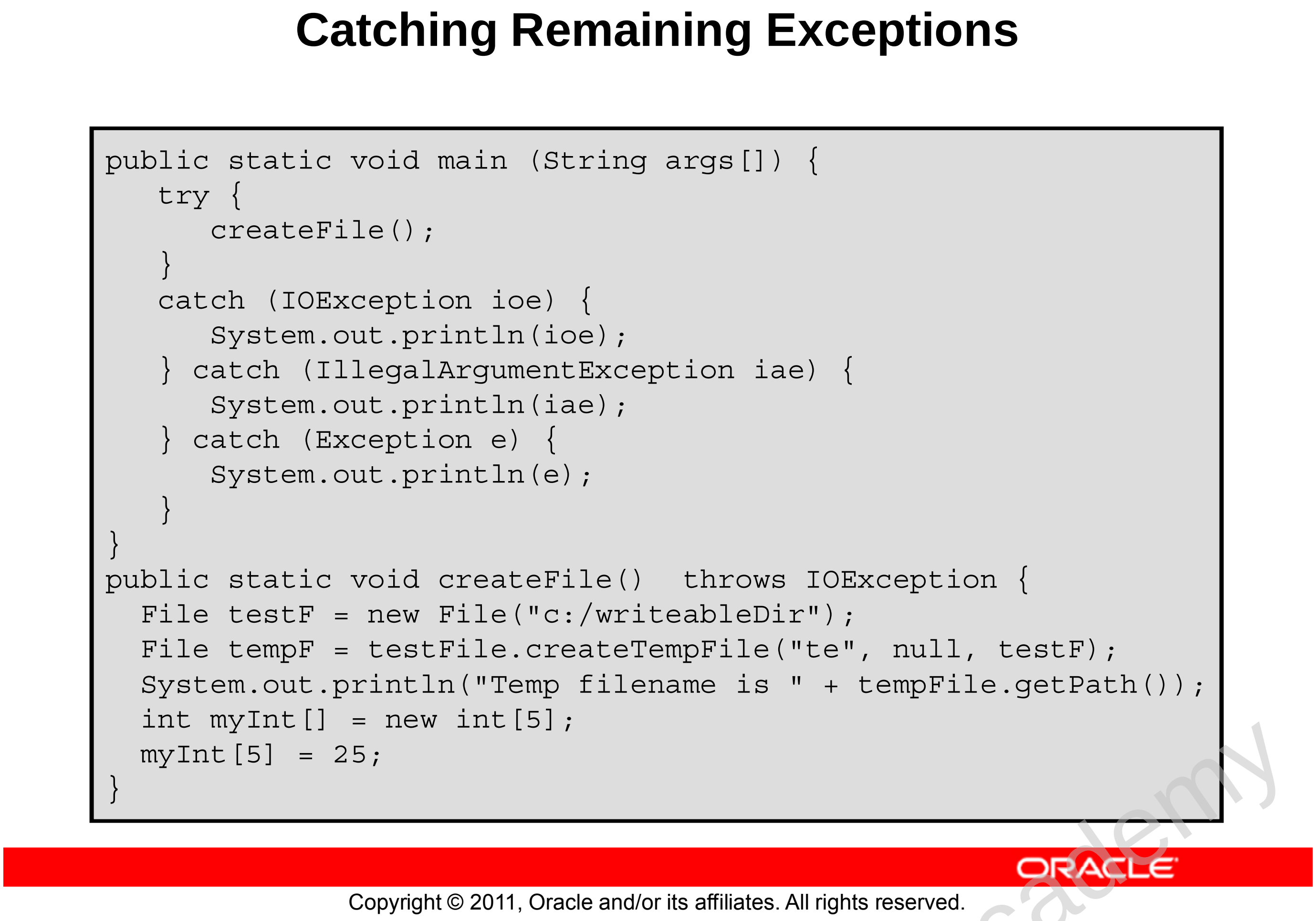
```
java.lang.IllegalArgumentException: Prefix string too short
```

However, if the argument is set to `"temp"`, the output is now:

```
Temp filename is /Users/kenny/writeableDir/temp938006797831220170.tmp
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
... < some code omitted > ...
```

Now the temporary file is being created, but there is still another argument being thrown by the `createFile()` method. And because ArrayIndexOutOfBoundsException is a RuntimeException, it is automatically thrown all the way out to the console.

Oracle Internal & Oracle Academy Use Only



Catching Remaining Exceptions

```
public static void main (String args[]) {
   try {
      createFile();
   }
   catch (IOException ioe) {
      System.out.println(ioe);
   } catch (IllegalArgumentException iae) {
      System.out.println(iae);
   } catch (Exception e) {
      System.out.println(e);
   }
}
public static void createFile()  throws IOException {
  File testF = new File("c:/writeableDir");
  File tempF = testFile.createTempFile("te", null, testF);
  System.out.println("Temp filename is " + tempFile.getPath());
  int myInt[] = new int[5];
  myInt[5] = 25;
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows an additional `catch` clause to catch all the remaining exceptions.

For the example code, the output of this code is:

```
Temp filename is /Users/kenny/writeableDir/temp7999507294858924682.tmp
java.lang.ArrayIndexOutOfBoundsException: 5
```

Finally, the `catch` exception clause can be added to catch any additional exceptions.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Describe the different kinds of errors that can occur and how they are handled in Java
- Describe what exceptions are used for in Java
- Determine what exceptions are thrown, for any foundation class
- Write code to handle an exception thrown by the method of a foundation class

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 13-1 Overview: Using a Try/Catch Block to Handle an Exception

In this practice, you handle an exception thrown by the `parse()` method of `SimpleDateFormat`. During the practice, you:

- Use the Java API documentation to examine the `SimpleDateFormat` class and find the exception thrown by its `parse()` method
- Create a class that calls the `parse()` method
- Write a try/catch block to catch the exception thrown by `parse()`

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Practice 13-2 Overview:
Catching and Throwing a Custom Exception

In this practice, you use a custom exception named `InvalidSkillException`. You use this with the Employee Tracking application that you designed and built in Practices 12-1 and 12-2.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

14

Deploying and Maintaining the Duke's Choice Application

ORACLE

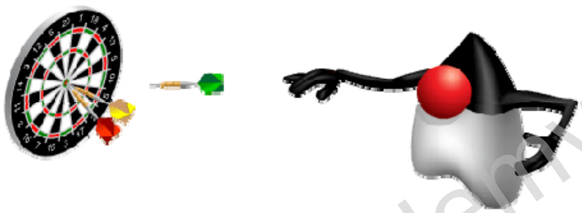
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to do the following:

- Deploy a simple application as a JAR file
- Describe the parts of a Java application, including the user interface and the back end
- Describe how classes can be extended to implement new capabilities in the application



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

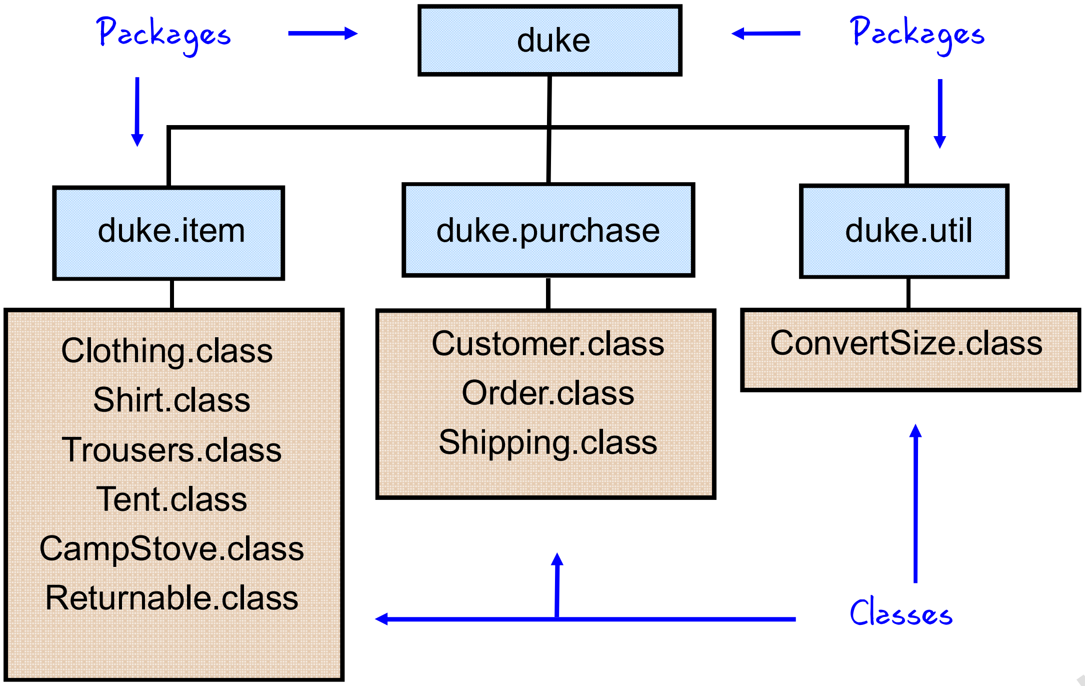
Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Duke's Choice application
- Application modifications and enhancements

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Packages



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

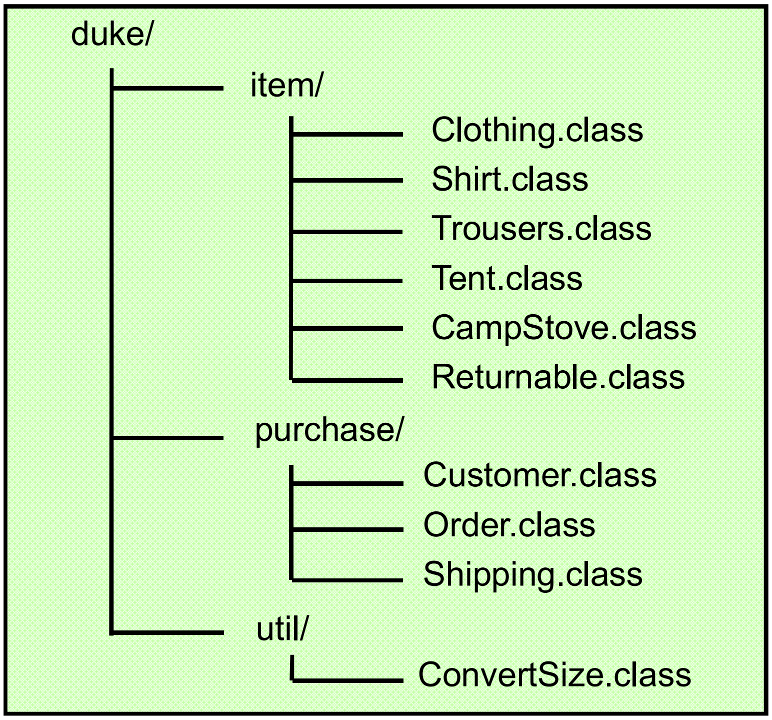
Classes are grouped into packages to ease management of the system.

There are many ways to group classes into meaningful packages. There is no right or wrong way, but a common technique is to group classes into a package by semantic similarity.

For example, the software for Duke's Choice could contain a set of item classes (such as `Shirt`, `Trousers`, `Tent`, the superclasses `Clothing` and `Camping`, and so on), a set of classes that use these item classes to arrange purchases, and a set of utility classes. All these packages are contained in the top-level package called `duke`.

Oracle Internal & Oracle Academy Use Only

Packages Directory Structure



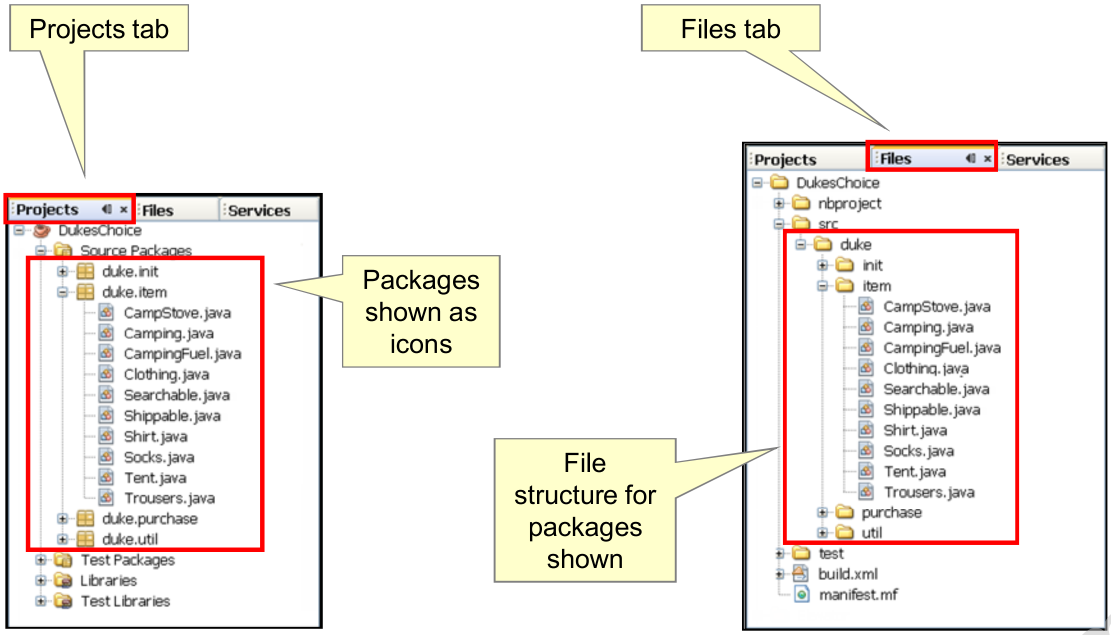
```
duke/
 ├── item/
 │    ├── Clothing.class
 │    ├── Shirt.class
 │    ├── Trousers.class
 │    ├── Tent.class
 │    ├── CampStove.class
 │    └── Returnable.class
 ├── purchase/
 │    ├── Customer.class
 │    ├── Order.class
 │    └── Shipping.class
 └── util/
      └── ConvertSize.class
```

ORACLE
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Packages are stored in a directory tree containing directories that match the package names. For example, the `Clothing.class` file should exist in the directory `item`, which is contained in the directory `duke`.

Oracle Internal & Oracle Academy Use Only

Packages in NetBeans



Projects tab

Files tab

Packages shown as icons

File structure for packages shown

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

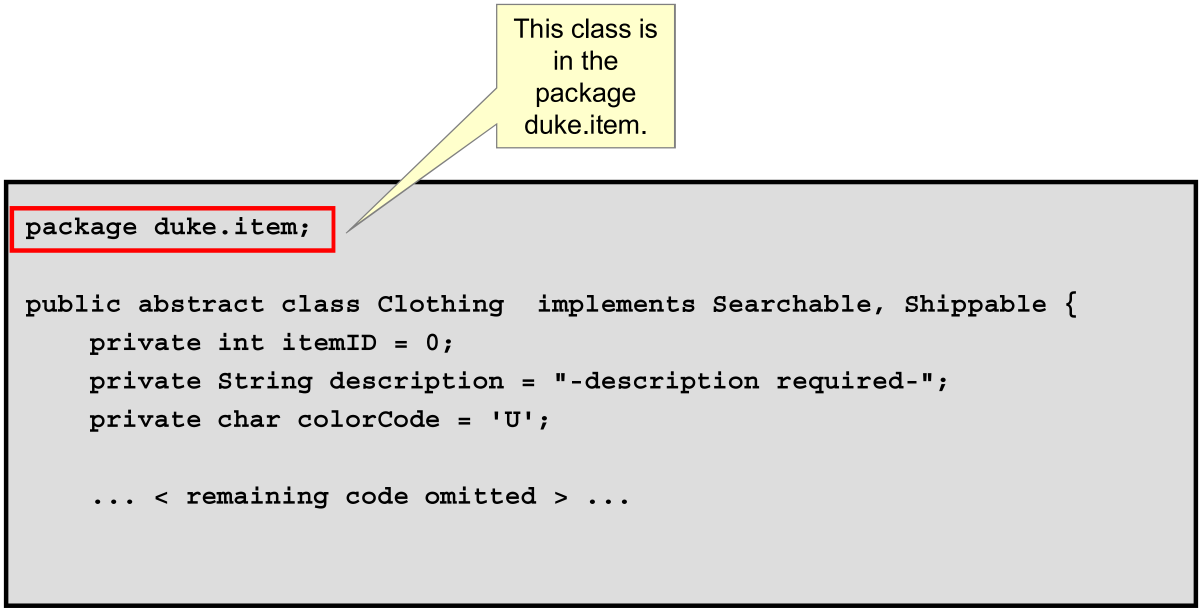
The left panel in NetBeans has three tabs. Two of these tabs, Projects and Files, show how packages relate to the file structure.

The Projects tab shows the packages and libraries for each project (the screenshot shows only `DukesChoice`). The source package shown is the one containing the packages and classes for Duke's Choice, and the screenshot shows the four packages `duke.init`, `duke.item`, `duke.purchase`, and `duke.util`. Each of these packages can be expanded to show the source files within, as has been done for the `duke.item` package in the screenshot.

The Files tab shows the directory structure for each project. In the screenshot, you can see how the packages listed on the Projects tab have a corresponding directory structure. For example, the `duke.item` package has the corresponding file structure of the `duke` directory just under the `src` directory and contains the `item` directory, which in turn contains all the source files in the package.

Oracle Internal & Oracle Academy Use Only

Packages in Source Code



This class is in the package duke.item.

```
package duke.item;

public abstract class Clothing  implements Searchable, Shippable {
    private int itemID = 0;
    private String description = "-description required-";
    private char colorCode = 'U';

    ... < remaining code omitted > ...
```

The package that a class belongs to is defined in the source code.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The example code in the slide shows the package statement being used to define the package that the Clothing class is in. Just as the class itself must be in a file of the same name as the class, the file (in this case, `Clothing.java`) must be contained in a directory structure that matches the package name.

Oracle Internal & Oracle Academy Use Only

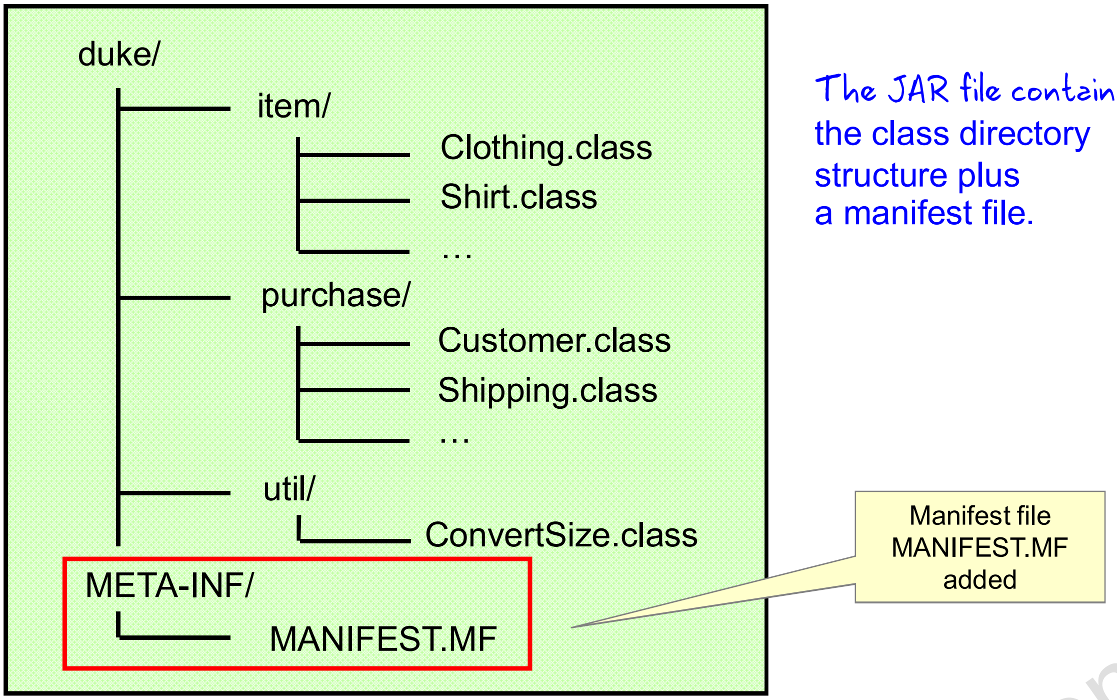
Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Duke's Choice application
- Application modifications and enhancements

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

DukesChoice.jar



The JAR file contains the class directory structure plus a manifest file.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

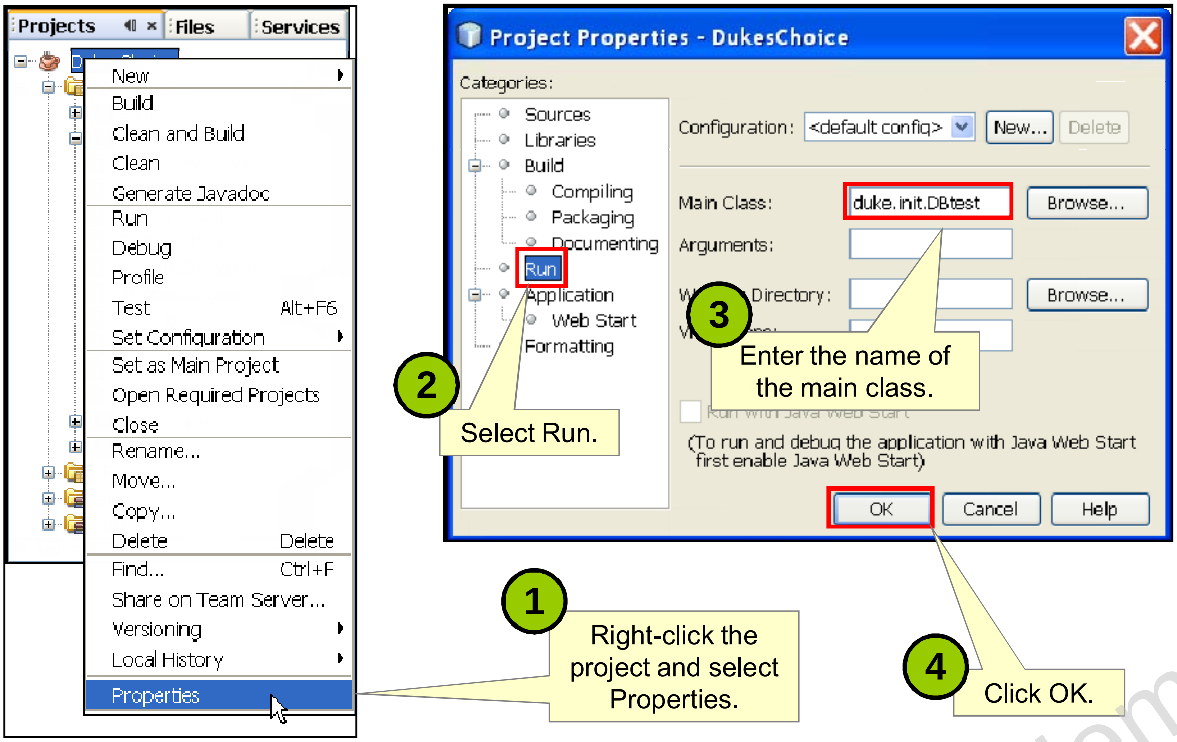
To deploy a Java application, you typically put the necessary files into a JAR file. This greatly simplifies running the application on another machine.

A JAR file is much like a zip file (or a tar file on UNIX) and contains the entire directory structure for the compiled classes plus an additional `MANIFEST.MF` file in the `META-INF` directory. This `MANIFEST.MF` file tells the Java runtime which file contains the `main()` method.

You can create a JAR file by using a command-line tool called jar, but most IDEs make the creation easier. In the following slides, you see how to create a JAR file using NetBeans.

Oracle Internal & Oracle Academy Use Only

Set Main Class of Project



Main Class: duke.init.DBtest

1 Right-click the project and select Properties.

2 Select Run.

3 Enter the name of the main class.

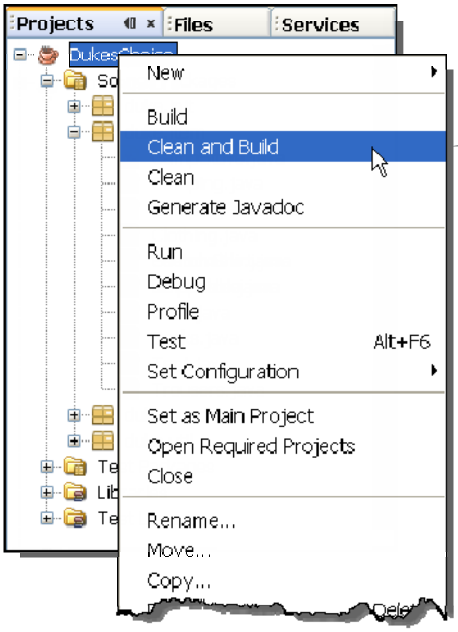
4 Click OK.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Before you create the JAR file, you need to indicate which file contains the `main()` method. This is subsequently written to the `MANIFEST.MF` file.

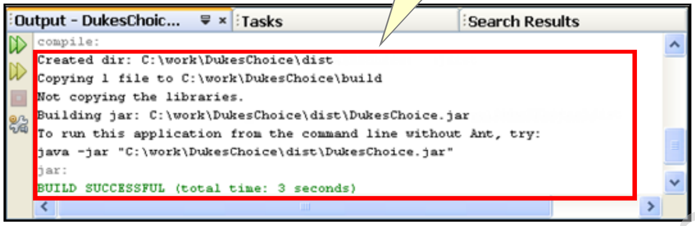
Oracle Internal & Oracle Academy Use Only

Creating the JAR File with NetBeans



1 Right-click the project and select “Clean and Build.”

2 Check the output to ensure the build is successful.



```
compile:
Created dir: C:\work\DukesChoice\dist
Copying 1 file to C:\work\DukesChoice\build
Not copying the libraries.
Building jar: C:\work\DukesChoice\dist\DukesChoice.jar
To run this application from the command line without Ant, try:
java -jar "C:\work\DukesChoice\dist\DukesChoice.jar"
jar:
BUILD SUCCESSFUL (total time: 3 seconds)
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

You create the JAR file by right-clicking the project and selecting “Clean and Build.” For a small project such as `DukesChoice`, this should take only a few seconds.

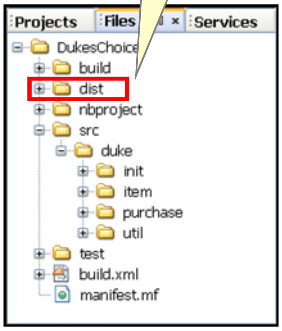
- Clean removes any previous builds.
- Build creates a new JAR file.

You can also run “Clean” and “Build” separately.

Oracle Internal & Oracle Academy Use Only

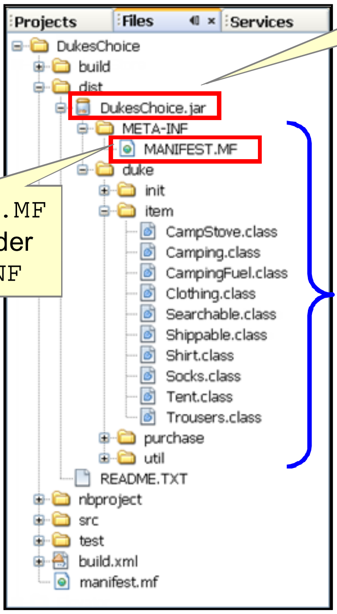
Creating the JAR File with NetBeans

Now a new directory in the Project



```
Projects | Files | Services
⊟ DukesChoice
  ⊞ build
  ⊞ dist
  ⊞ nbproject
  ⊟ src
    ⊟ duke
      ⊞ init
      ⊞ item
      ⊞ purchase
      ⊞ util
  ⊞ test
  ⊞ build.xml
    manifest.mf
```

`MANIFEST.MF` added under `META-INF`



```
Projects | Files | Services
⊟ DukesChoice
  ⊞ build
  ⊟ dist
    ⊟ DukesChoice.jar
      ⊟ META-INF
          MANIFEST.MF
      ⊟ duke
        ⊞ init
        ⊟ item
            CampStove.class
            Camping.class
            CampingFuel.class
            Clothing.class
            Searchable.class
            Shippable.class
            Shirt.class
            Socks.class
            Tent.class
            Trousers.class
        ⊞ purchase
        ⊞ util
      README.TXT
  ⊞ nbproject
  ⊞ src
  ⊞ test
  ⊞ build.xml
    manifest.mf
```

`DukesChoice.jar` under `dist` directory

The JAR file contains the class directory structure plus a manifest file.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

By default, the JAR file will be placed in the dist directory. (This directory is removed in the clean process and re-created during build.) Using the files tab of NetBeans, you can look inside the JAR file and make sure that all the correct classes have been added.

Oracle Internal & Oracle Academy Use Only

Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Duke's Choice application
- Application modifications and enhancements

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Client/Server Two-Tier Architecture

Client/server computing involves two or more computers sharing tasks:

- Each computer performs logic appropriate to its design and stated function.
- The front-end client communicates with the back-end database.
- Client requests data from back end.
- Server returns appropriate results.
- Client handles and displays data.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

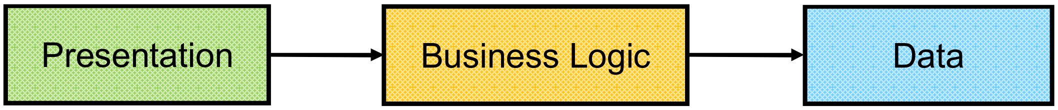
A major performance penalty is paid in two-tier client/server. The client software ends up larger and more complex because most of the logic is handled there. The use of server-side logic is limited to database operations. The client here is referred to as a *thick client.*

Thick clients tend to produce frequent network traffic for remote database access. This works well for intranet-based and local area network (LAN)–based network topologies, but produces a large footprint on the desktop in terms of disk and memory requirements. Also, not all back-end database servers are the same in terms of server logic offered, and all of them have their own API sets that programmers must use to optimize and scale performance.

Oracle Internal & Oracle Academy Use Only

Client/Server Three-Tier Architecture

- Three-tier client/server is a more complex, flexible approach.
- Each tier can be replaced by a different implementation:
  - Presentation can be GUI, web, smartphone, or even console.
  - Business logic defines business rules.
  - Data tier is an encapsulation of all existing data sources.



Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The three components or tiers of a three-tier client/server environment are *presentation, business logic or functionality,* and *data.* They are separated so that the software for any one of the tiers can be replaced by a different implementation without affecting the other tiers.

For example, if you want to replace a character-oriented screen (or screens) with a GUI (the presentation tier), you write the GUI using an established API or interface to access the same functionality programs in the character-oriented screens.

The business logic offers functionality in terms of defining all of the business rules through which the data can be manipulated. Changes to business policies can affect this layer without having an impact on the actual databases.

The third tier, or data tier, includes existing systems, applications, and data that have been encapsulated to take advantage of this architecture with minimal transitional programming effort.

Oracle Internal & Oracle Academy Use Only

Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Duke's Choice application
- Application modifications and enhancements

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

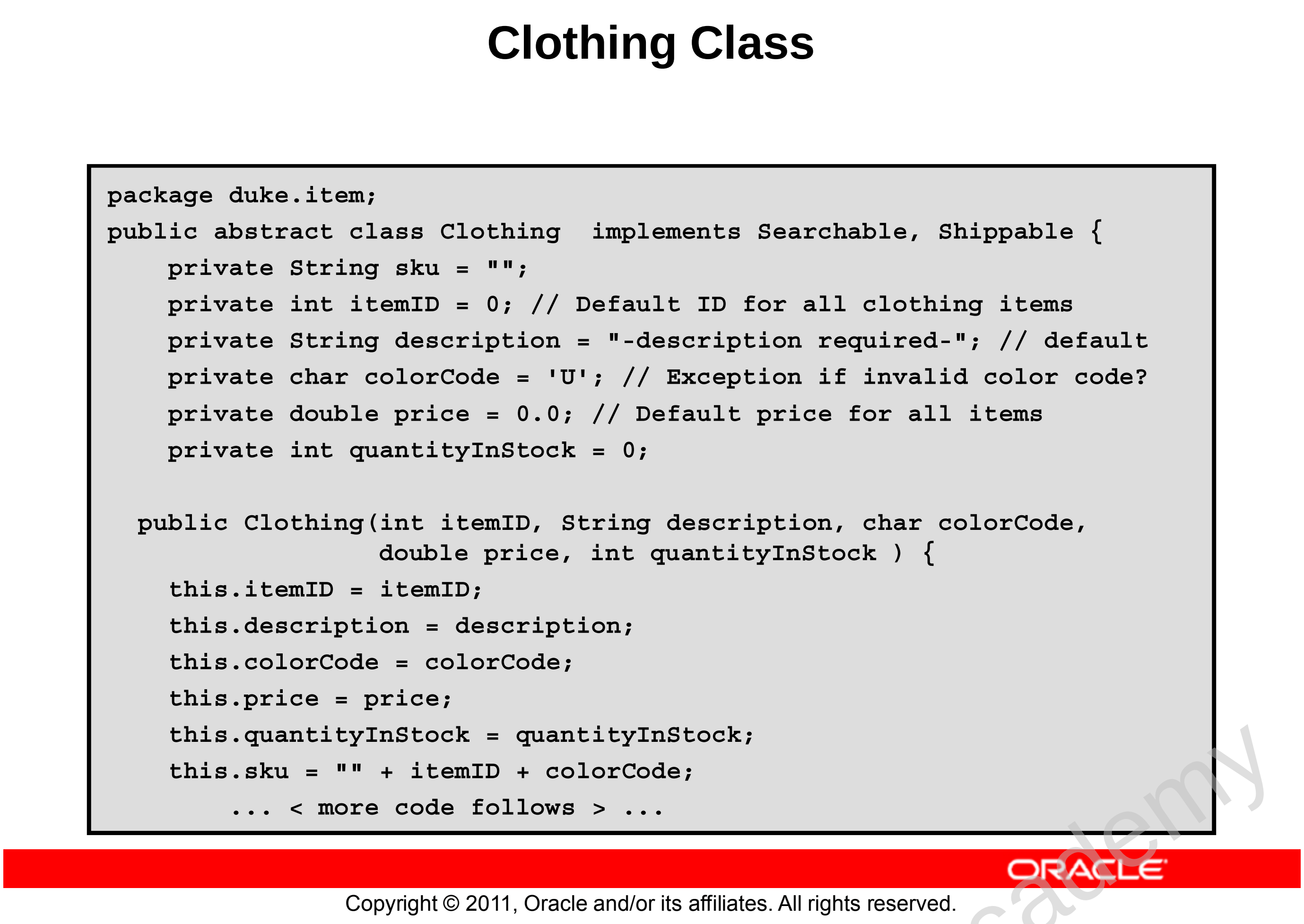
The Duke's Choice Application

- Abstract classes
  - Clothing
    - Extended by Shirt and other clothing classes
  - Camping
    - Extended by Tent and other camping classes
- Interfaces
  - Searchable
    - All purchasable items implement Searchable.
  - Returnable
    - Items that can be returned implement Returnable.
  - Shippable
    - Items that can be shipped implement Shippable.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

A version of the Duke's Choice application has been created to illustrate object-oriented programming in Java.

Oracle Internal & Oracle Academy Use Only



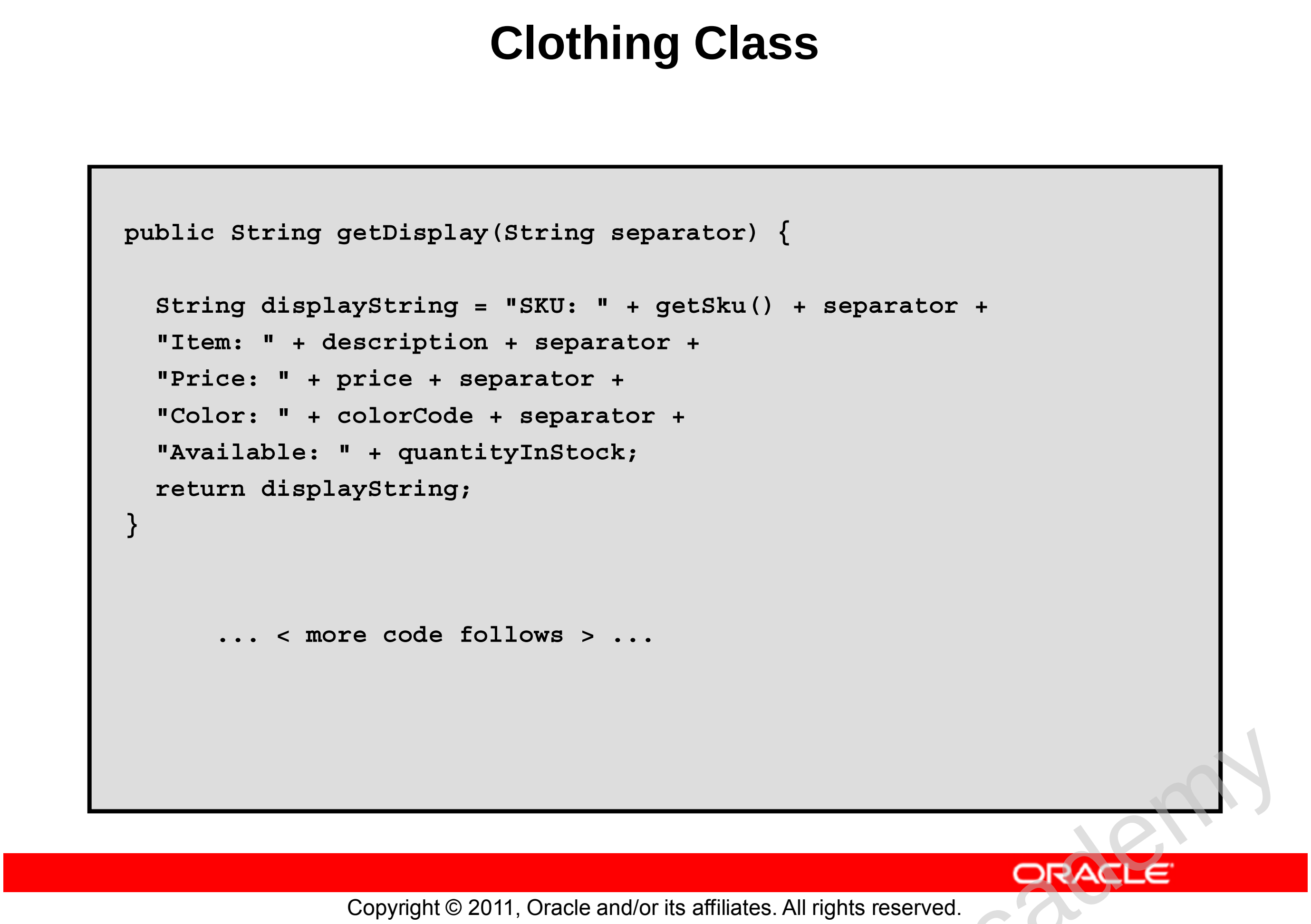
Clothing Class

```
package duke.item;
public abstract class Clothing  implements Searchable, Shippable {
    private String sku = "";
    private int itemID = 0; // Default ID for all clothing items
    private String description = "-description required-"; // default
    private char colorCode = 'U'; // Exception if invalid color code?
    private double price = 0.0; // Default price for all items
    private int quantityInStock = 0;

  public Clothing(int itemID, String description, char colorCode,
                  double price, int quantityInStock ) {
    this.itemID = itemID;
    this.description = description;
    this.colorCode = colorCode;
    this.price = price;
    this.quantityInStock = quantityInStock;
    this.sku = "" + itemID + colorCode;
        ... < more code follows > ...
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The `Clothing` class is very similar to the `Shirt` class you have seen earlier in the course. However, to ensure that there is a unique code for every type of item, a field SKU (Stock Keeping Unit) has been added.

Oracle Internal & Oracle Academy Use Only



Clothing Class

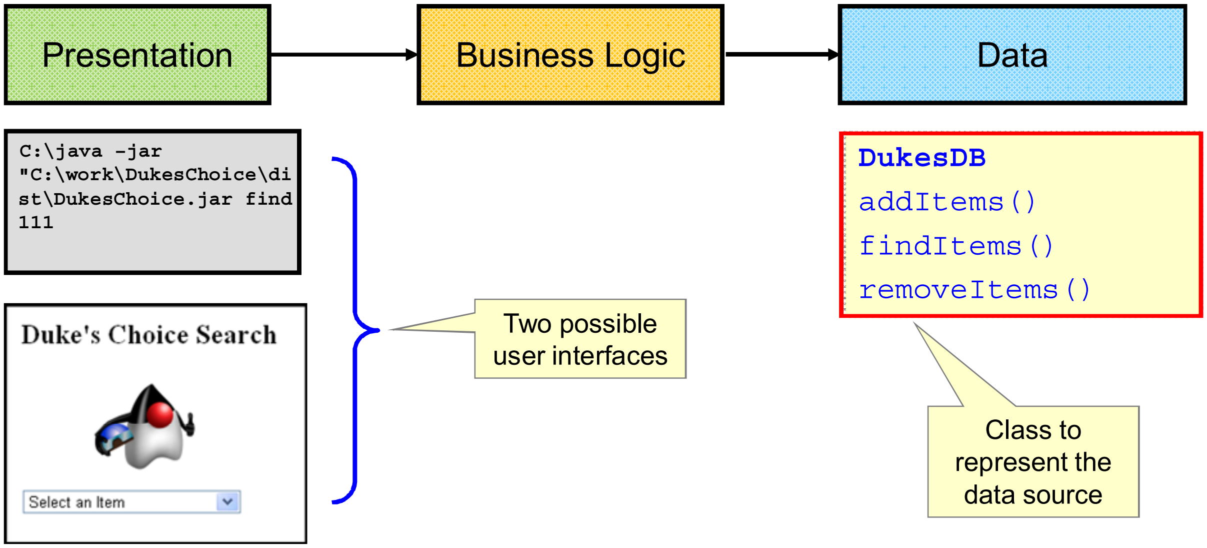
```
public String getDisplay(String separator) {

  String displayString = "SKU: " + getSku() + separator +
  "Item: " + description + separator +
  "Price: " + price + separator +
  "Color: " + colorCode + separator +
  "Available: " + quantityInStock;
  return displayString;
}


      ... < more code follows > ...
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In addition to the previous method, `display()`, to display details of the item, a `getDisplay()` method has been added that returns a String. This allows the method to be called by different clients. It takes one argument: a String that determines how the individual attributes of the item are separated. For example, they could be separated with a new line in the console version of the application, or with an HTML element for the web application.

Oracle Internal & Oracle Academy Use Only

Tiers of Duke's Choice



```
C:\java -jar
"C:\work\DukesChoice\di
st\DukesChoice.jar find
111
```

Duke's Choice Search

Two possible user interfaces

```
DukesDB
addItems()
findItems()
removeItems()
```

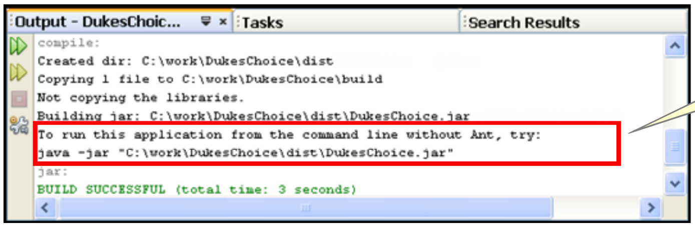
Class to represent the data source

ORACLE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Running the JAR File from the Command Line



```
compile:
Created dir: C:\work\DukesChoice\dist
Copying 1 file to C:\work\DukesChoice\build
Not copying the libraries.
Building jar: C:\work\DukesChoice\dist\DukesChoice.jar
To run this application from the command line without Ant, try:
java -jar "C:\work\DukesChoice\dist\DukesChoice.jar"
jar:
BUILD SUCCESSFUL (total time: 3 seconds)
```

The command to run the JAR file

```
C:\java -jar "C:\work\DukesChoice\dist\DukesChoice.jar
```

Output:

```
Please add parameters in the format:
   find <item id number>
   OR
   remove <sku> <number to remove>
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Running the command-line application using the JAR file is very straightforward and the instructions are actually given in the output window for the build process. (If it were implemented as a GUI application, it would be run the same way.)

Assuming the application is an early command-line version of the software that has been sent to Duke's Choice for testing, you run it as shown in the slide. Because it is an early version, assume that it is only for the use of Duke's Choice employees and requires parameters to be added at the command line to do anything.

Oracle Internal & Oracle Academy Use Only

Listing Items from the Command Line

Casual Shirt: 111
Dress Trousers: 120
Sports Socks: 131
...

```
C:\java -jar "C:\work\DukesChoice\dist\DukesChoice.jar find 111
```

Output:

```
------------------------------------------------------------------------
SKU: 111R | Item: Casual Shirt | Price: 34.29 | Color: R | Available: 63
------------------------------------------------------------------------
SKU: 111B | Item: Casual Shirt | Price: 25.05 | Color: B | Available: 20
------------------------------------------------------------------------
```

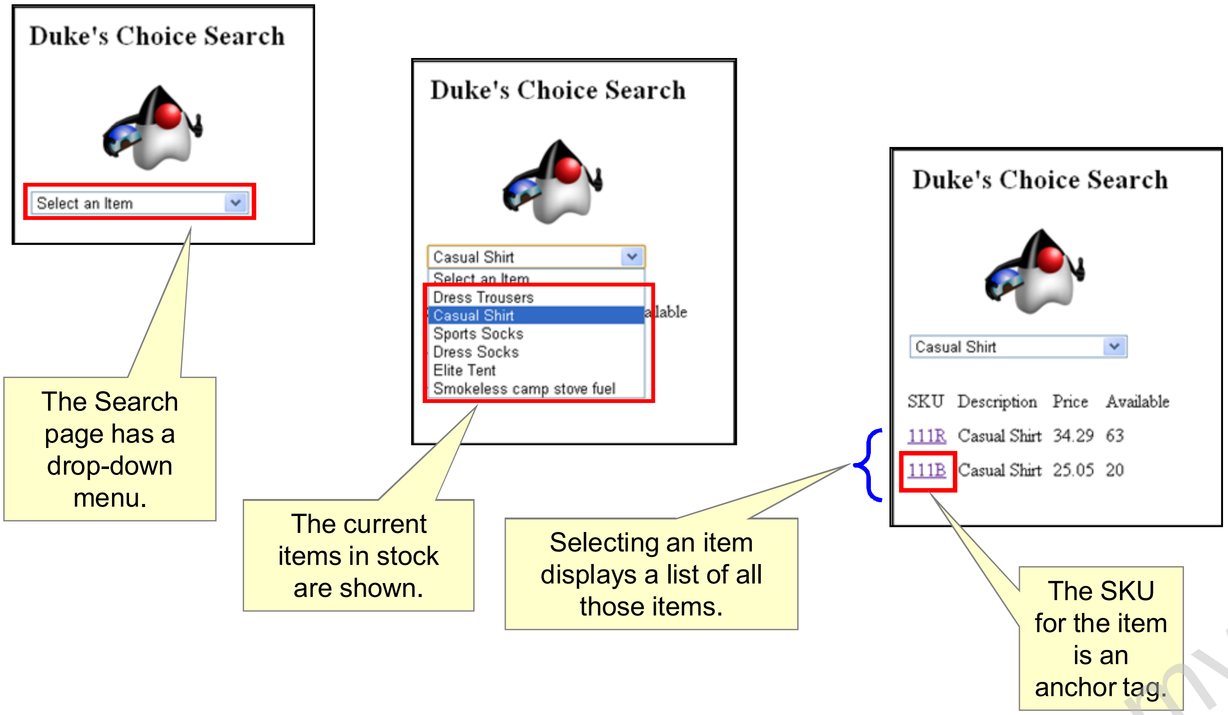
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In this simple application, commands are entered using command-line parameters and SKU or item IDs. So you can assume that Duke's Choice employees have been given a list of the appropriate item IDs so that they can try the application.

In the example, the application is finding all kinds of casual shirts in stock. Currently there are two kinds of casual shirt in stock: red and blue. You can also see that 63 red shirts and 20 blue ones are in stock.

Oracle Internal & Oracle Academy Use Only

Listing Items in Duke's Choice Web Application



SKU	Description	Price	Available
111R	Casual Shirt	34.29	63
111B	Casual Shirt	25.05	20

The Search page has a drop-down menu.

The current items in stock are shown.

Selecting an item displays a list of all those items.

The SKU for the item is an anchor tag.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

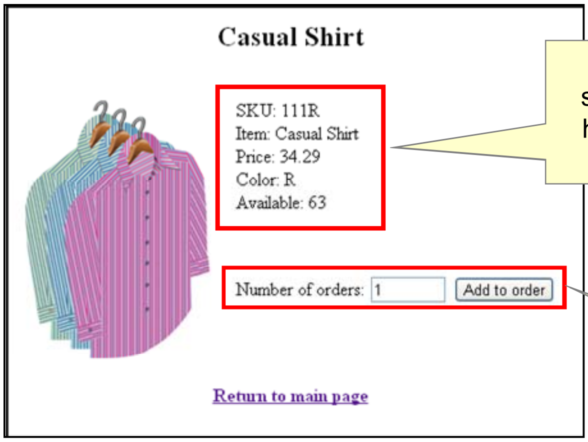
Here is the other possible application for Duke's Choice—a simple web application. In this case, `DukesChoice.jar` is copied to the application server, where it can be accessed by the UI components of the application (in this case, by Java Server Pages [JSP] files).

The screenshot shows the main search page that allows customers to search for a particular item. They can pick an item from a drop-down list, and all of the varieties of the item are then listed. In the example in the slide, the list shows the same information as the command-line application: two colors of shirt and the available quantity of each.

The web application also allows a customer to click the SKU number of a particular item and, by doing so, navigate to a page that shows further details about that item.

Oracle Internal & Oracle Academy Use Only

Listing Items in Duke's Choice Web Application



Casual Shirt

SKU: 111R
Item: Casual Shirt
Price: 34.29
Color: R
Available: 63

Number of orders: 1

Details of the shirt, including how many are available

Click to order

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The screen shown in the slide shows the details of the item that the customer selected. On this page, customers can add a specific number of shirts to their orders.

The two applications shown (the command-line application and this web application) use classes very similar to the Shirt class you were introduced to at the very beginning of the course. Even though the user interface of the command-line version is very different from the web version, the item classes (Shirt, Trousers, Socks, Tent, and Fuel) are not in any way involved in the presentation of the data, so it is possible to modify any of these classes or add additional classes without having to change the user interface.

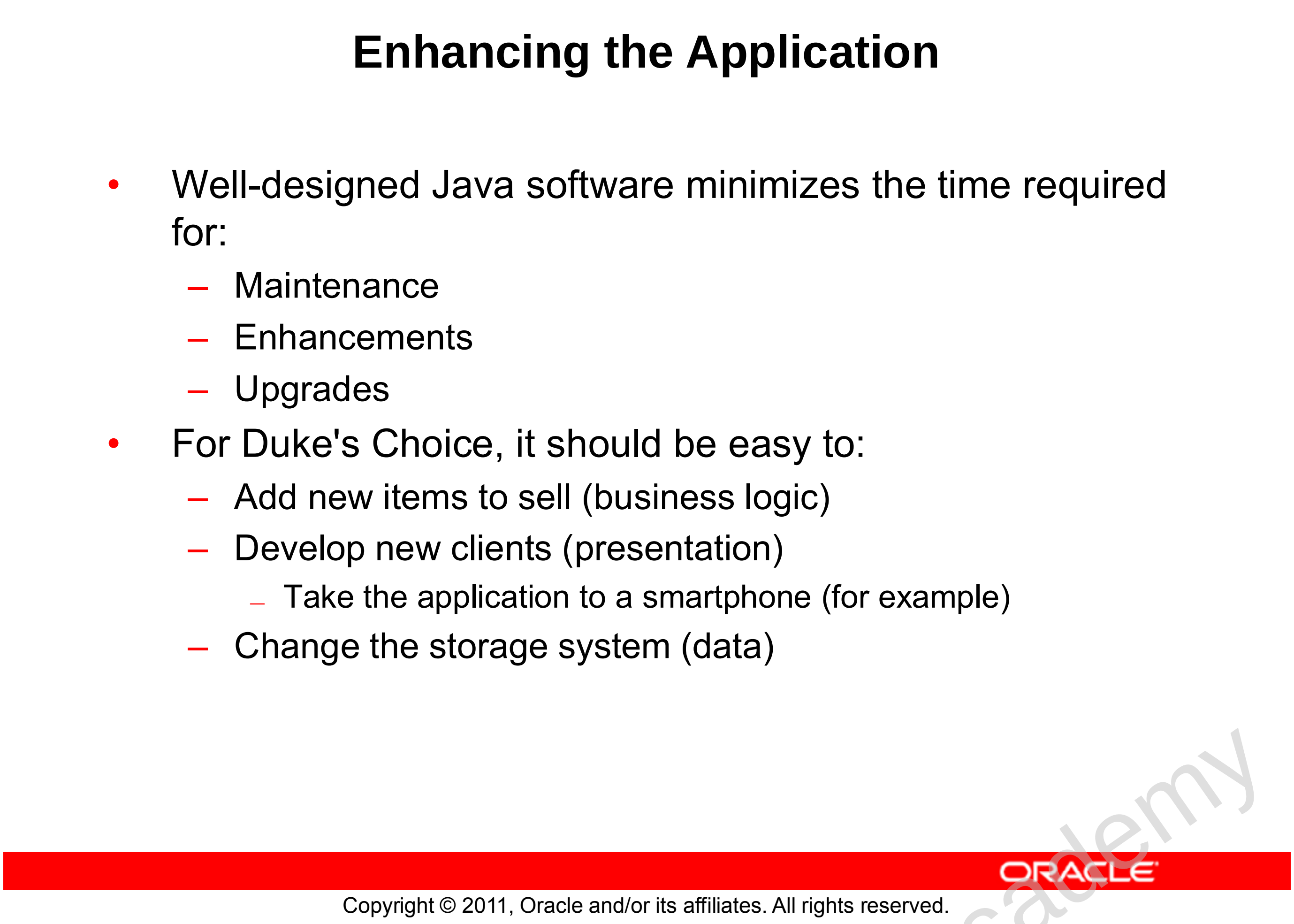
Oracle Internal & Oracle Academy Use Only

Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- Duke's Choice application
- Application modifications and enhancements

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only



Enhancing the Application

- Well-designed Java software minimizes the time required for:
  - Maintenance
  - Enhancements
  - Upgrades
- For Duke's Choice, it should be easy to:
  - Add new items to sell (business logic)
  - Develop new clients (presentation)
    - Take the application to a smartphone (for example)
  - Change the storage system (data)

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

In the following slides, you see what is involved in adding another item class to represent a dress suit.

Oracle Internal & Oracle Academy Use Only

Adding a New Item for Sale

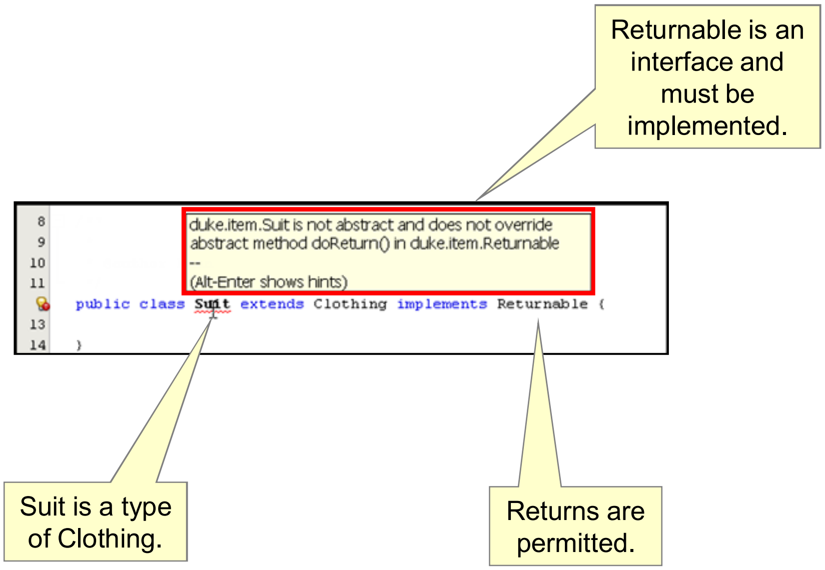
It is possible to add a new item for sale by:

- Extending the Clothing or Camping class, or even creating a new category (for example, Books)
- Adding any new unique features for the item
- Adding some of the new items to the data store

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Adding a New Item for Sale



```
duke.item.Suit is not abstract and does not override
abstract method doReturn() in duke.item.Returnable
--
(Alt-Enter shows hints)
```

```
public class Suit extends Clothing implements Returnable {

}
```

Returnable is an interface and must be implemented.

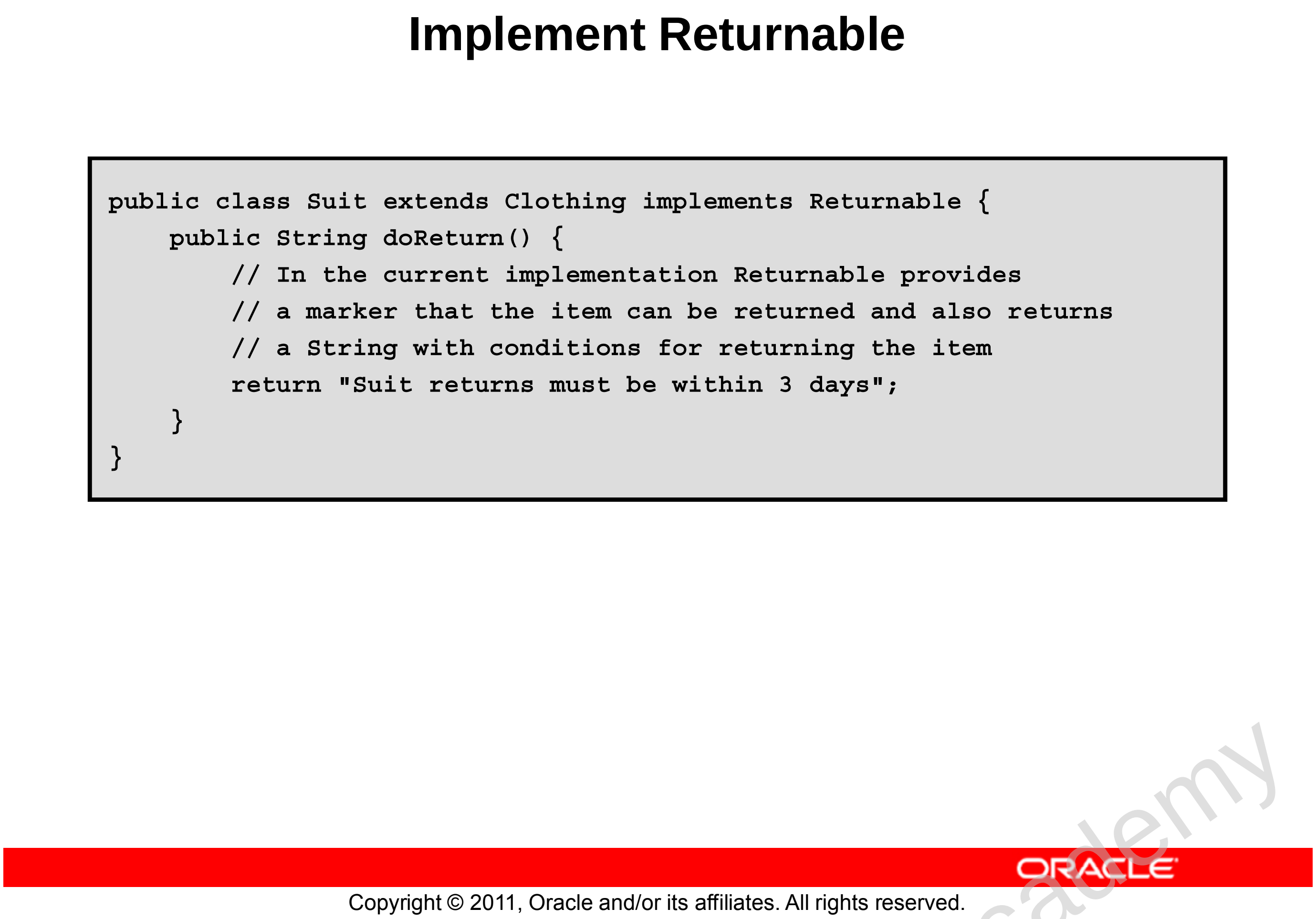
Suit is a type of Clothing.

Returns are permitted.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

NetBeans is helpful when extending abstract classes and implementing interfaces because it gives you hints about what you need to do. In the example in the slide, the new class `Suit` extends `Clothing` and implements `Returnable`. NetBeans flags that you need to implement the methods of the `Returnable` interface (in this case, the `doReturn()` method).

Oracle Internal & Oracle Academy Use Only



Implement Returnable

```
public class Suit extends Clothing implements Returnable {
    public String doReturn() {
        // In the current implementation Returnable provides
        // a marker that the item can be returned and also returns
        // a String with conditions for returning the item
        return "Suit returns must be within 3 days";
    }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code shows a simple example of implementing the `doReturn()` method of the `Returnable` interface.

Oracle Internal & Oracle Academy Use Only

Implement Constructor

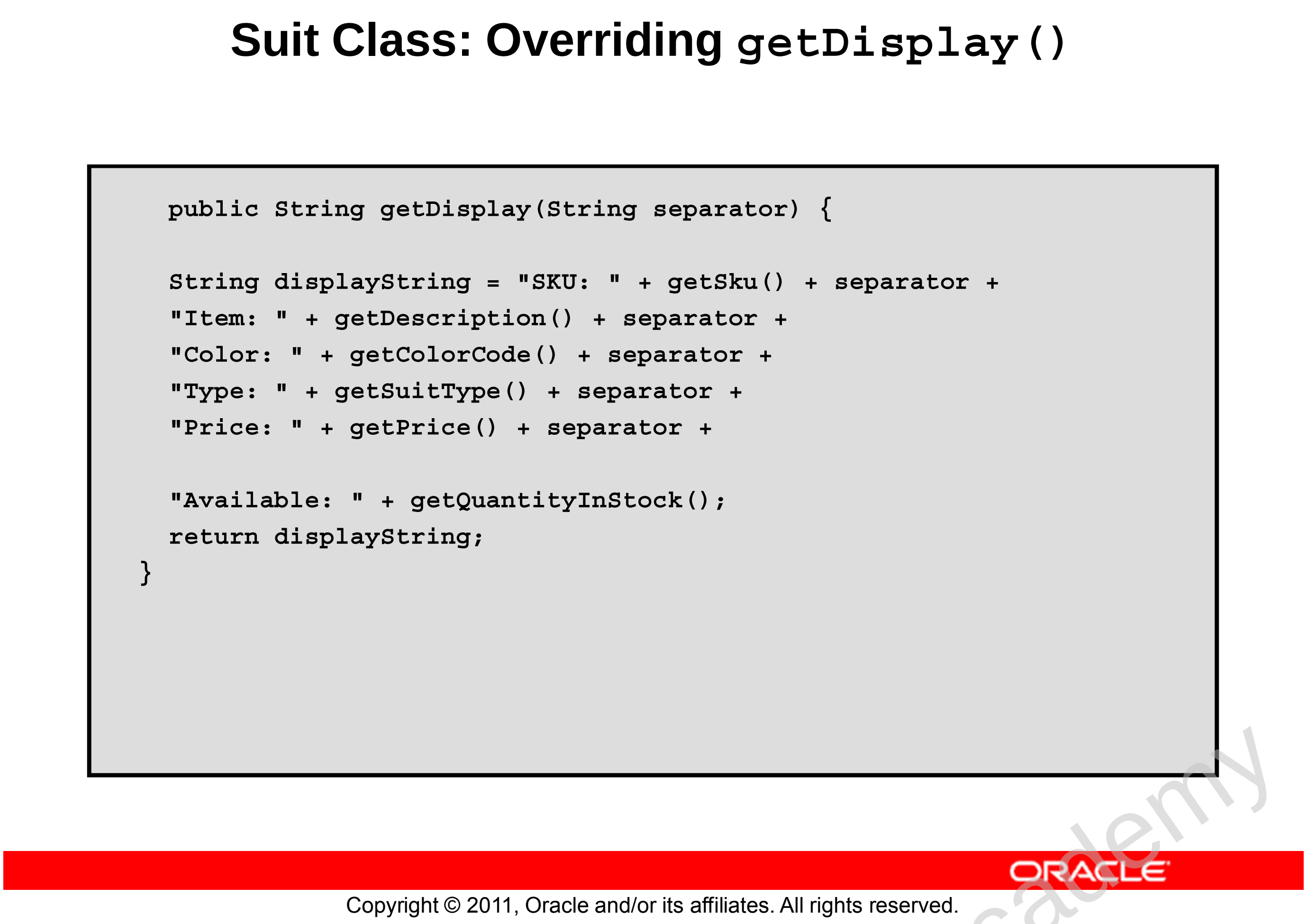
```
public class Suit extends Clothing implements Returnable {

  ...< code omitted > ...

  // Types are D = Double-breasted, S = Single-breasted, U=Unset
  private char suitType = 'U'; //

  // Constructor
  public Suit(int itemID, String description, char colorCode,
             double price,  char type, int quantityInStock) {
    super( itemID,  description,  colorCode, price, quantityInStock);
    setSuitType(type);
    setSku(getSku() + type); // To create a unique SKU
  }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

This code shows the implementation of the constructor for the Suit type, given that:

- Suit has an extra attribute, suitType, that is not in the superclass Clothing
- This extra attribute is combined with the SKU (generated in the `Clothing` superclass) to create a unique SKU for this item

Oracle Internal & Oracle Academy Use Only



Suit Class: Overriding `getDisplay()`

```
   public String getDisplay(String separator) {

   String displayString = "SKU: " + getSku() + separator +
   "Item: " + getDescription() + separator +
   "Color: " + getColorCode() + separator +
   "Type: " + getSuitType() + separator +
   "Price: " + getPrice() + separator +

   "Available: " + getQuantityInStock();
   return displayString;
 }
```

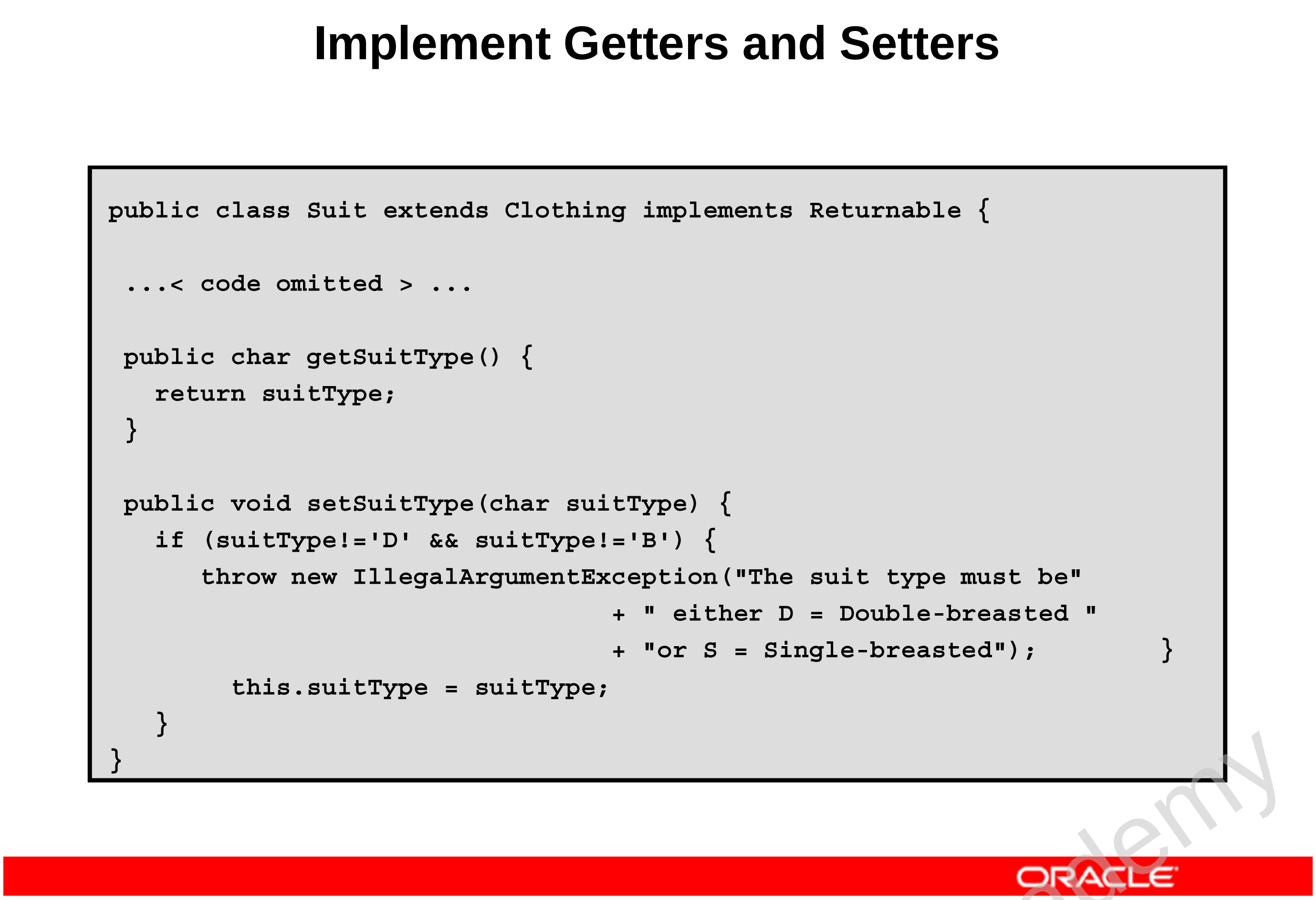
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Clothing class has a `getDisplay(String separator)` method where a separator can be specified so that the attributes of the item can be written on one line and separated by a separator character, or written line by line using newline as the separator character.

The code in the slide shows `getDisplay(String separator)` being overridden to include the suit type in the display.

```
C:\>java -jar "C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar" find
410
----------------------------------------------------------------------------
SKU: 410BD | Item: Suit | Color: B | Type: D | Price: 999.99 | Available: 21
----------------------------------------------------------------------------
SKU: 410BS | Item: Suit | Color: B | Type: S | Price: 789.99 | Available: 15
----------------------------------------------------------------------------
SKU: 410gD | Item: Suit | Color: G | Type: D | Price: 999.99 | Available: 21
----------------------------------------------------------------------------
SKU: 410WS | Item: Suit | Color: W | Type: S | Price: 789.99 | Available: 15
----------------------------------------------------------------------------
```

Oracle Internal & Oracle Academy Use Only



Implement Getters and Setters

```
public class Suit extends Clothing implements Returnable {

  ...< code omitted > ...

 public char getSuitType() {
   return suitType;
 }

 public void setSuitType(char suitType) {
   if (suitType!='D' && suitType!='B') {
      throw new IllegalArgumentException("The suit type must be"
                                   + " either D = Double-breasted "
                                   + "or S = Single-breasted");        }
        this.suitType = suitType;
   }
}
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The code shows the implementation of the getter and setter methods for the suit type. If `'D'` or `'B'` is not passed into the constructor, the method throws an IllegalArgumentException. Note that IllegalArgumentException is an unchecked exception, so it does not need to be thrown from this method or checked in the calling method.

Assuming it is not caught in the current implementation of the application, if an invalid argument is passed into the method, the Duke's Choice testers see the following:

```
C:\>java -jar
"C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar"
find 410

Exception in thread "main" java.lang.IllegalArgumentException:
The suit type must be either D = Double-breasted or S = Single-breasted

        at duke.item.Suit.setSuitType(Suit.java:43)

        at duke.item.Suit.<init>(Suit.java:20)

        at duke.init.DukesDB.setupDb(DukesDB.java:52)

        at duke.init.DukesDB.<init>(DukesDB.java:84)

        at duke.init.DBtest.main(DBtest.java:29)
```

Oracle Internal & Oracle Academy Use Only

Updating the Applications with the Suit Class

For the command-line application:

- Create a new `DukesChoice.jar` file.
- (Optional) Copy it to a new location on the file system or to another machine.

For the web application:

- Create a new `DukesChoice.jar` file.
- Copy it to the directory that is used by the application server for library files.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Note that the JAR file is exactly the same in either case.

Oracle Internal & Oracle Academy Use Only

Testing the Suit Class: Command Line

```
C:\>java -jar
    "C:\work\Java_fundamentals\DukesChoice\dist\DukesChoice.jar"
    find 410
------------------------------------------------------------
SKU: 410BD | Item: Suit | Price: 999.99 | Color: B | Available: 21
------------------------------------------------------------
SKU: 410BS | Item: Suit | Price: 789.99 | Color: B | Available: 15
------------------------------------------------------------
SKU: 410gD | Item: Suit | Price: 999.99 | Color: G | Available: 14
------------------------------------------------------------
SKU: 410WS | Item: Suit | Price: 789.99 | Color: W | Available: 18
------------------------------------------------------------
```

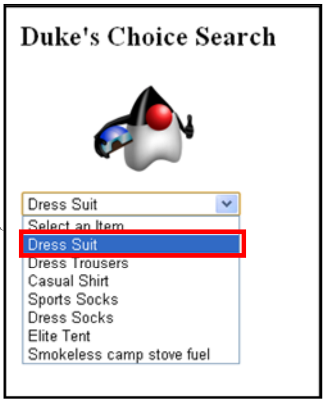
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Now the testers at Duke's Choice can search for suits in stock. However, the display does not let them know if the suit is single-breasted or double-breasted.

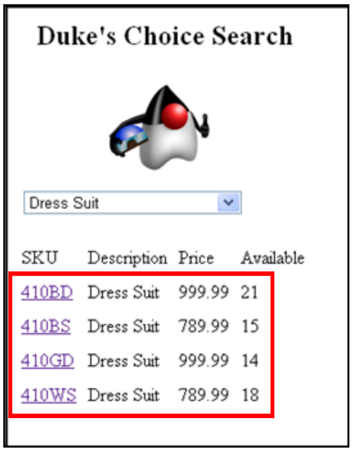
Oracle Internal & Oracle Academy Use Only

Testing the Suit Class: Web Application

A new item appears in the drop-down menu.



The different kinds of suits added to the data store are listed.



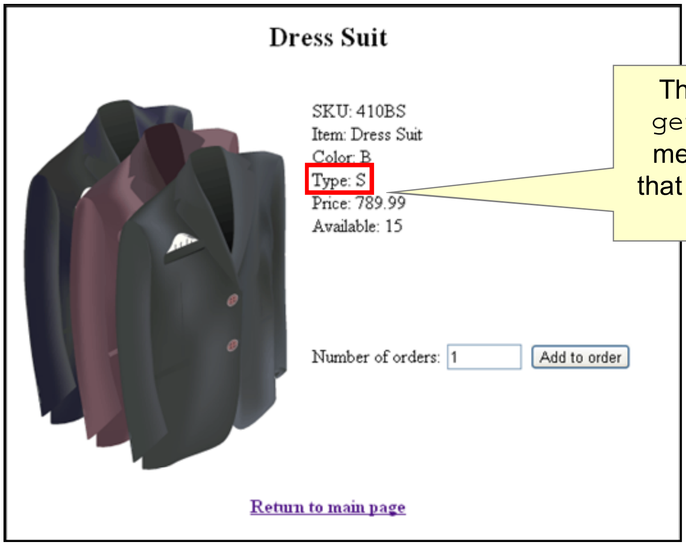
SKU	Description	Price	Available
410BD	Dress Suit	999.99	21
410BS	Dress Suit	789.99	15
410GD	Dress Suit	999.99	14
410WS	Dress Suit	789.99	18

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

After restarting the web application, the testers see an extra item in the drop-down menu for Dress Suit, and the various kinds of Dress Suits that have been added to the data store are listed with their SKUs.

Oracle Internal & Oracle Academy Use Only

Adding the Suit Class to the Web Application



Dress Suit

SKU: 410BS
Item: Dress Suit
Color: B
Type: S
Price: 789.99
Available: 15

Number of orders: 1

The overridden `getDisplay()` method ensures that the suit type is displayed.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

When you click one of the Dress Suits listed, the details are displayed. Notice that because the `getDisplay()` method was overridden, the kind of suit (S for single-breasted) is displayed. No modifications were made to the web application.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Deploy a simple application as a JAR file
- Describe the parts of a Java application, including the user interface and the back end
- Describe how classes can be extended to implement new capabilities in the application

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

No Practice for This Lesson

This lesson has no practices.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Course Summary

In this course, you should have learned how to:

- List and describe several key features of the Java technology, such as that it is object-oriented, multi-threaded, distributed, simple, and secure
- Identify different Java technology groups
- Describe examples of how Java is used in applications, as well as consumer products
- Describe the benefits of using an integrated development environment (IDE)
- Develop classes and describe how to declare a class
- Analyze a business problem to recognize objects and operations that form the building blocks of the Java program design

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Course Summary

- Define the term *object* and its relationship to a class
- Demonstrate Java programming syntax
- Write a simple Java program that compiles and runs successfully
- Declare and initialize variables
- List several primitive data types
- Instantiate an object and effectively use object reference variables
- Use operators, loops, and decision constructs
- Declare and instantiate arrays and ArrayLists and be able to iterate through them

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Course Summary

- Use Javadocs to look up Java foundation classes
- Declare a method with arguments and return values
- Use inheritance to declare and define a subclass of an existing superclass
- Describe how errors are handled in a Java program
- Describe how to deploy a simple Java application by using the NetBeans IDE

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

A

Java Language Quick Reference

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

1. **Declare a class**

```
public class Shirt{ ←class declaration
  }
```

2. **Declare a field/variable**

```
 public char colorCode; ←field variable
  int counter; ←local variable
```

3. **Declare and initialize a primitive variable**

```
 public double price = 0.0; ←field variable
 int hour = 12; ←local variable
```

4. **Declare and instantiate an object reference**

```
 public ArrayList names = new ArrayList();
```

5. **Invoke a method**

```
 displayInformation(); ←method  with no arguments or return value
 setColorCode('R'); ←method  with one argument  and no return value
 int level = getLevel(); ←method  with no arguments  but returning a  value
```

6. **Declare a method**

```
 public void displayInformation(){…} ←method : no args, returns void
 public String getName() {…} ←method: no args, returns String
 public void setName(String name){…} ←method: String arg, returns void
```

7. **If/else block**

```
 If (name1.equals(name2)) {
   System.out.println();
 }
 else {
    System.out.println("Different name.");
 }
```

8. **Switch construct** ←Syntax

```
switch (variable) {
  case literal_value:
       <code_block>
       [break;]
  case another_literal_value:
       <code_block>
       [break;]
  [default:]
       <code_block>
 }
```

Oracle Internal & Oracle Academy Use Only

9. Structure of a Class

```
package myClasses; ←package statement
import java.util.ArrayList; ←import statement

public class NamesList{  ←class declaration
       public ArrayList names = new ArrayList(); ←field

       public void setList(){ ←method
             // code_block;
       } ←end of method
} ←end of class
```

10. While construct ←syntax

```
 while (boolean_expression) {
       // do this while expression remains true
       // code_block;
 } // end of while block
```

11. Do/while construct ←syntax

```
do {    // do the following once before evaluating expression
    // then continue to do this while expression remains true
    // code_block
}
while (boolean_expression);
```

12. For loop ←syntax

```
for (data_type init_var; boolean_expression; increment){
       // code_block;
}
```

←example

```
for (int i = 1; i<10; i++){
       System.out.println("Array element: " + myArray[i]);
}
```

Oracle Internal & Oracle Academy Use Only

13. Enhanced for loop ←syntax

```
for (data_type var : array_name ) {
     // code_block;
}
```

←example

```
for (Object obj : myList){
      System.out.println("List element: "+ obj);
}
```

Oracle Internal & Oracle Academy Use Only

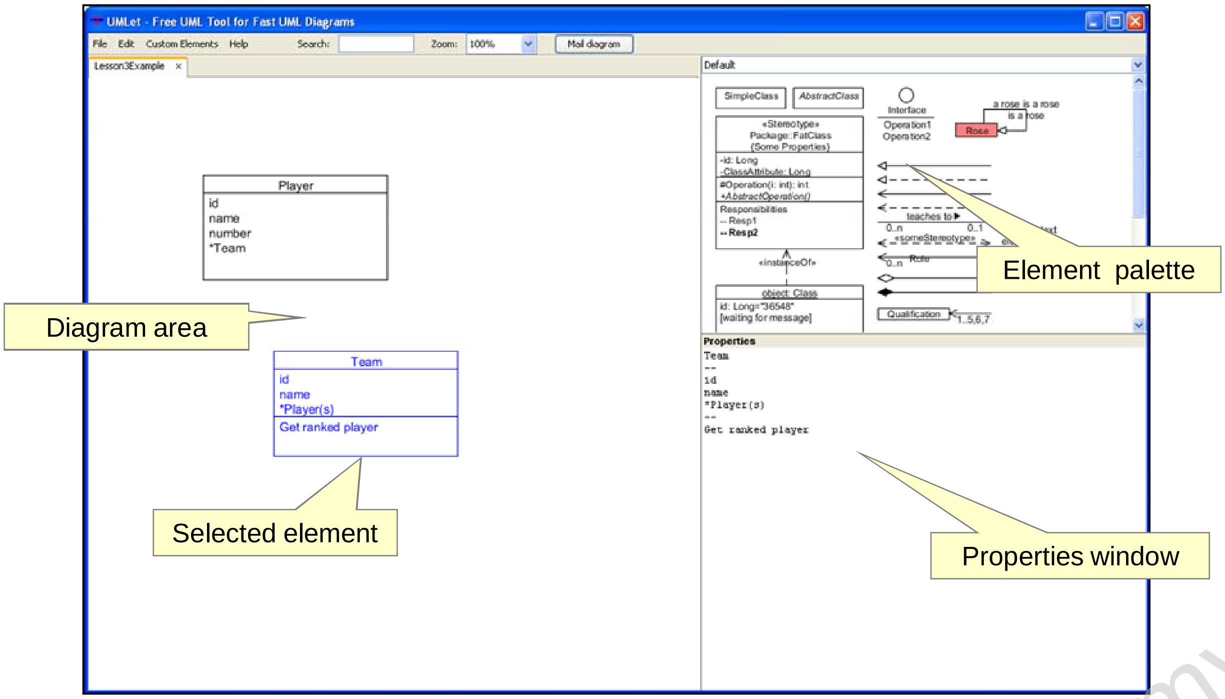
B

UMLet Tips

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

UML Default Interface



Properties

```
Team
--
id
name
*Player(s)
--
Get ranked player
```

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

1. **How to add elements to the diagram**
   Double-click any element in the palette; it appears in the upper-left corner of the main diagram window.
2. **How to duplicate elements on the diagram**
   Double-click an element to duplicate it. Alternatively, you can copy and paste (or you can use their respective keyboard equivalents of Ctrl + C and Ctrl + V).
3. **How to select multiple elements**
   Press and hold Ctrl to select multiple elements.
4. **How to lasso-select multiple elements**
   Press Ctrl and click to select a rectangle containing the desired elements.
5. **How to change UML elements**
   Select an element and modify its attributes in the lower-right text panel. Each element type has a simple markup language (for example, the text “/ClassName/” causes “ClassName” to become italic). The markup languages are best explored via the sample UML elements in the palettes.

Oracle Internal & Oracle Academy Use Only

6. **How to enter comments in a UML element description**
   UMLet supports C++-style comments. Starting a line with “//” (for example, “//my comment..”) enables UMLet to ignore that markup line.

7. **How to change the color of UML elements**
   Right-click an element and select its background or foreground color via the context menu.

8. Alternatively, just type the name of the color in the element description (for example, “bg=black”, or “fg=red”).

9. **How to create UML relations**
   Double-click a relation, and then drag its end points to the borders of UML elements; they will stick there.

10. **How to edit the relations**
    Many UML tools make it time consuming to change the type or direction of a relation. In UMLet, simply modify the linetype (that is, by changing the line “lt=“ in the element description). For example, change “lt=<.” to “lt=->>” to change the direction, the arrow type, and the line’s dots at the same time.

11. **How to label relations**
    Edit the name of a relation in the relation’s description.
    Role names can be specified using “r1=” or “r2=”.
    For multiplicities, use “m1=” or “m2=”.
    Qualifiers are done with “q1=” or “q2=”.

12. **How to create sequence diagrams**
    Change the current palette to “Sequence - all in one”. Add the sequence diagram element to the diagram by double-clicking.
    This element’s markup language is slightly more complex. The main idea is that each lane has a name and an ID (defined by the string “_name~ID_”). The IDs can then be used to define messages between lanes (for example, “id1->id3”).

13. **How to create activity diagrams**
    Change the current palette to “Activity - all in one”. Add the activity diagram element to the diagram by double-clicking.
    Here, TABs in the element description are used to define the activity forks.

Oracle Internal & Oracle Academy Use Only

UML Basics

The Unified Modeling Language (UML) is a graphical language for modeling software systems. The UML is not:

- A programming language: It is a set of diagrams that can be used to specify, construct, visualize, and document software designs. Software engineers use UML diagrams to construct and explain their software designs just as building architects use blueprints to construct and explain their building designs. UML has diagrams to assist in every part of application development, from requirements gathering through design, coding, testing, and deployment.
- A process for analysis and design: Its diagrams must be used with a process.

The UML was developed in the early 1990s by three leaders in the object-modeling world: Grady Booch, James Rumbaugh, and Ivar Jacobson. Their goal was to unify the major methods that they had previously developed to create a new standard for software modeling. UML is now the most commonly used modeling language. The UML specification is currently maintained by the Object Management Group (OMG) and is available on the OMG website at http://www.omg.org/uml/.

General Elements

In general, UML diagrams represent:

- Concepts, which are depicted as symbols (also called *nodes*)
- Relationships among those concepts, which are depicted as paths (also called *links*) that connect the symbols

These nodes and links are specialized for each particular diagram. For example, in Class diagrams, the nodes represent object classes and the links represent associations between classes and generalization (inheritance) relationships.

Oracle Internal & Oracle Academy Use Only

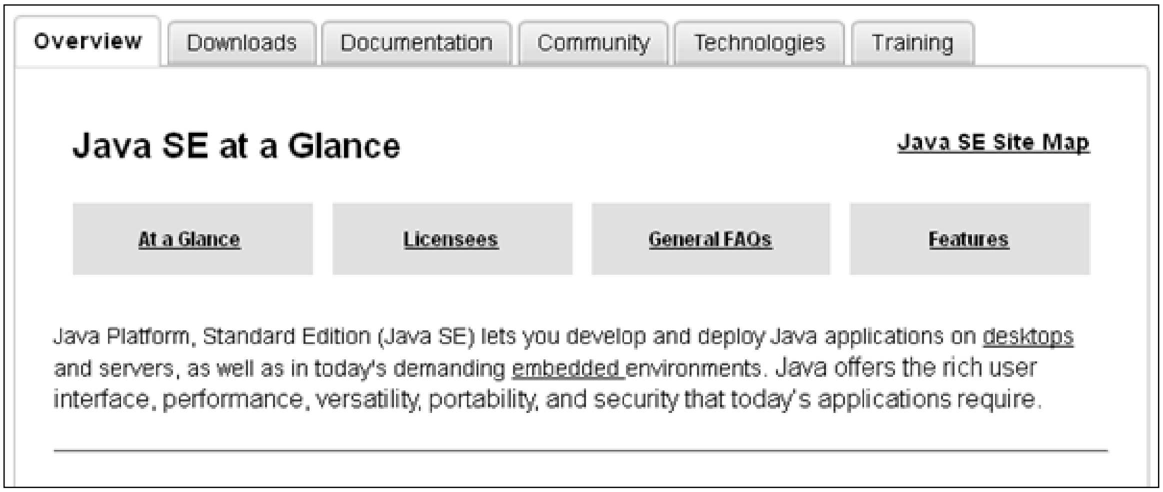
C

Resources

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Java on Oracle Technology Network (OTN)



Java SE at a Glance

Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

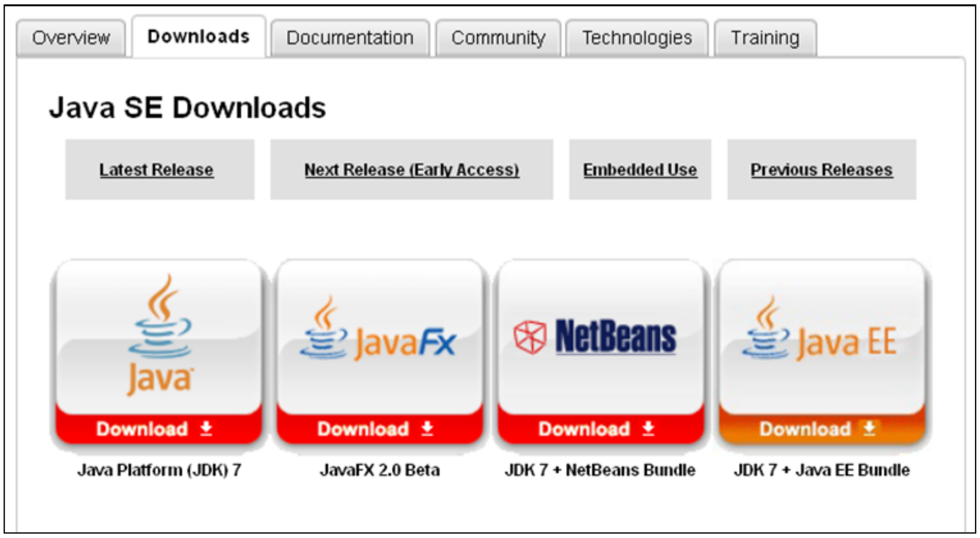
You can find many resources on the Java SE 7 pages of OTN, including:

- Downloads
- Documentation
- Java Community
- Technologies
- Training

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Oracle Internal & Oracle Academy Use Only

Java SE Downloads



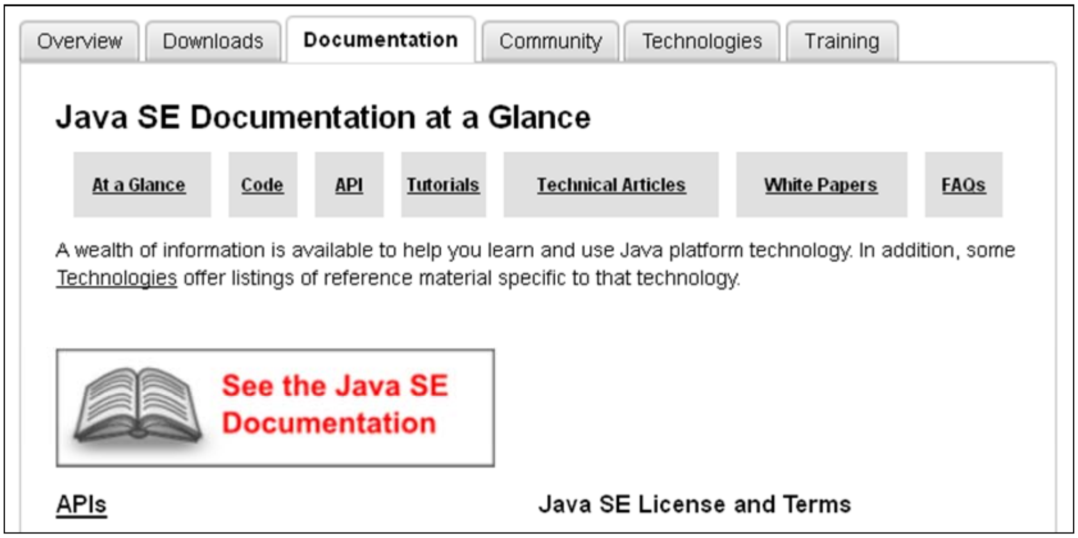
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Downloads link provides the latest and previous releases for Java SE (runtime and JDK), JavaFX, Java EE, and NetBeans.

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Oracle Internal & Oracle Academy Use Only

Java Documentation



Java SE Documentation at a Glance

A wealth of information is available to help you learn and use Java platform technology. In addition, some Technologies offer listings of reference material specific to that technology.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

You can find many resources on the Documentation page, including:

- Code
- API
- Tutorials
- Technical Articles
- …and more

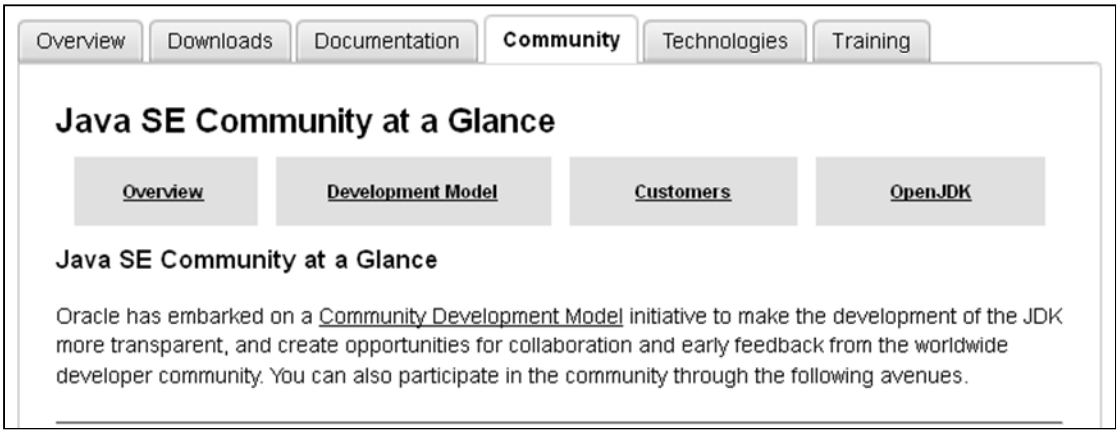
The Java SE Documentation link includes more developer information such as:

- API documentation
- Java language and Virtual Machine specifications
- Developer guides
- JDK / JRE Installation Instructions
- …and more

Documentation page:
http://www.oracle.com/technetwork/java/javase/documentation/index.html

Java SE Technical Documentation page: http://download.oracle.com/javase/

Java Community



Java SE Community at a Glance

Java SE Community at a Glance

Oracle has embarked on a Community Development Model initiative to make the development of the JDK more transparent, and create opportunities for collaboration and early feedback from the worldwide developer community. You can also participate in the community through the following avenues.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Oracle Internal & Oracle Academy Use Only

What is the Java Community? We frequently hear about the Java Community, as well as a variety of acronyms related to Java that you may not be familiar with, such as JUGs, JCP EC, and OpenJDK.

At a very high level, the Java Community is the term used to refer to the many individuals and organizations that develop, innovate, and use Java technology.

The Java Community page includes links to:

- **Forums:** The Java technology discussion forums are interactive message boards for sharing ideas and insights on Java technologies and programming techniques.
- **User groups:** Members of the Java User Groups meet regularly to exchange technical ideas and information.
- **Java Developer Newsletter:** The Java Developer Newsletter is a free, monthly online communication that includes news, technical articles, and events.
- **Blogs** such as the following:
  - The Java Source
  - Java Oracle Blogs
- **Java Developer events**

http://www.oracle.com/technetwork/java/javase/community/index.html

Java Community: Expansive Reach

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

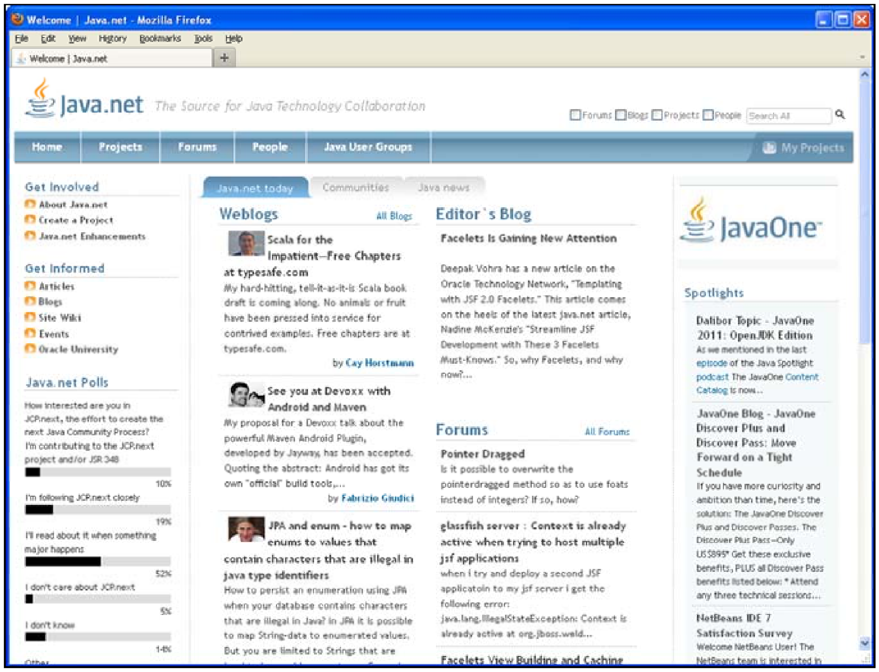
Forums: The Java technology discussion forums are interactive message boards for JUGs. A Java User Group (JUG) is a group of people who share a common interest in Java technology and meet on a regular basis to share technical ideas and information. The actual structure of a JUG can vary greatly—from a small number of friends and coworkers meeting informally in the evening to a large group of companies based in the same geographic area. Regardless of the size and focus of a particular JUG, the sense of community spirit remains the same.

OpenJDK (also known as **Open Java Development Kit):** A free and open source implementation of the Java programming language. In addition to Oracle, other contributors such as RedHat, IBM, and Apple all contribute to OpenJDK.

JCP: JCP stands for Java Community Process, a formalized process that allows interested parties to get involved in the definition of future versions and features of the Java platform. The JCP Executive Committee (EC) is the group of members guiding the evolution of Java technology. The EC represents both major stakeholders and a representative cross-section of the Java Community.

Oracle Internal & Oracle Academy Use Only

Java Community: Java.net


Welcome | Java.net - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Welcome | Java.net
Java.net The Source for Java Technology Collaboration
□Forums □Blogs □Projects □People Search All
Home Projects Forums People Java User Groups
My Projects
Java.net today Communities Java news
JavaOne

Get Involved

- About Java.net
- Create a Project
- Java.net Enhancements

Get Informed

- Articles
- Blogs
- Site Wiki
- Events
- Oracle University

Java.net Polls

How interested are you in JCP.next, the effort to create the next Java Community Process?

I'm contributing to the JCP.next project and/or JSR 348 10%

I'm following JCP.next closely 19%

I'll read about it when something major happens 52%

I don't care about JCP.next 5%

I don't know 14%

Other

Weblogs All Blogs

Scala for the Impatient—Free Chapters at typesafe.com
My hard-hitting, tell-it-as-it-is Scala book draft is coming along. No animals or fruit have been pressed into service for contrived examples. Free chapters are at typesafe.com.
by Cay Horstmann

See you at Devoxx with Android and Maven
My proposal for a Devoxx talk about the powerful Maven Android Plugin, developed by Jayway, has been accepted. Quoting the abstract: Android has got its own "official" build tools,...
by Fabrizio Giudici

JPA and enum - how to map enums to values that contain characters that are illegal in java type identifiers
How to persist an enumeration using JPA when your database contains characters that are illegal in Java? in JPA it is possible to map String-data to enumerated values. But you are limited to Strings that are

Editor`s Blog

Facelets Is Gaining New Attention

Deepak Vohra has a new article on the Oracle Technology Network, "Templating with JSF 2.0 Facelets." This article comes on the heels of the latest java.net article, Nadine McKenzie's "Streamline JSF Development with These 3 Facelets Must-Knows." So, why Facelets, and why now?...

Forums All Forums

Pointer Dragged
Is it possible to overwrite the pointerdragged method so as to use foats instead of integers? If so, how?

glassfish server : Context is already active when trying to host multiple jsf applications
when i try and deploy a second JSF applicatoin to my jsf server i get the following error: java.lang.IllegalStateException: Context is already active at org.jboss.weld...

Facelets View Building and Caching

Spotlights

Dalibor Topic - JavaOne 2011: OpenJDK Edition
As we mentioned in the last episode of the Java Spotlight podcast The JavaOne Content Catalog is now...

JavaOne Blog - JavaOne Discover Plus and Discover Pass: Move Forward on a Tight Schedule
If you have more curiosity and ambition than time, here's the solution: The JavaOne Discover Plus and Discover Passes. The Discover Plus Pass—Only US$895* Get these exclusive benefits, PLUS all Discover Pass benefits listed below: * Attend any three technical sessions...

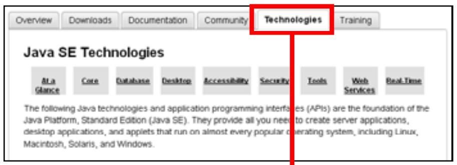
NetBeans IDE 7 Satisfaction Survey
Welcome NetBeans User! The NetBeans team is interested in

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Java.net is a large community of Java developers and their projects. It welcomes anyone interested in Java, related JVM technologies, and education to the discussions and projects on the site. Java.net manages projects in a different way from most groups by maintaining curated communities of projects. That is, projects that use similar technologies or are similar types are grouped together in an area to make it easier to find other developers with similar interests and skills and their projects. The site offers technical articles, news on events, and blogs.

Oracle Internal & Oracle Academy Use Only

Java Technologies

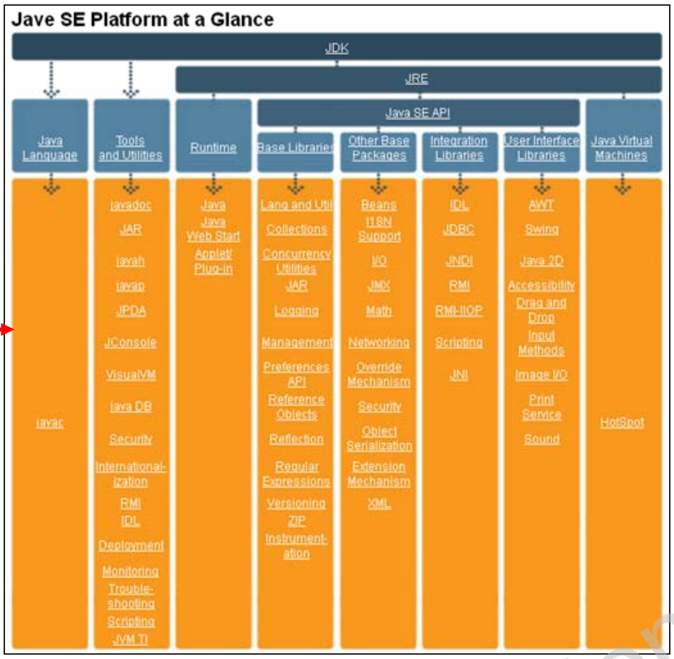


Overview | Downloads | Documentation | Community | Technologies | Training

Java SE Technologies

At a Glance | Core | Database | Desktop | Accessibility | Security | Tools | Web Services | Real-Time

The following Java technologies and application programming interfaces (APIs) are the foundation of the Java Platform, Standard Edition (Java SE). They provide all you need to create server applications, desktop applications, and applets that run on almost every popular operating system, including Linux, Macintosh, Solaris, and Windows.



Jave SE Platform at a Glance

JDK

JRE

Java SE API

Java Language	Tools and Utilities	Runtime	Base Libraries	Other Base Packages	Integration Libraries	User Interface Libraries	Java Virtual Machines
javac	javadoc JAR javah javap JPDA JConsole VisualVM java DB Security International-ization RMI IDL Deployment Monitoring Trouble-shooting Scripting JVM TI	Java Java Web Start Applet/ Plug-in	Lang and Util Collections Concurrency Utilities JAR Logging Management Preferences API Reference Objects Reflection Regular Expressions Versioning ZIP Instrument-ation	Beans I18N Support I/O JMX Math Networking Override Mechanism Security Object Serialization Extension Mechanism XML	IDL JDBC JNDI RMI RMI-IIOP Scripting JNI	AWT Swing Java 2D Accessibility Drag and Drop Input Methods Image I/O Print Service Sound	HotSpot

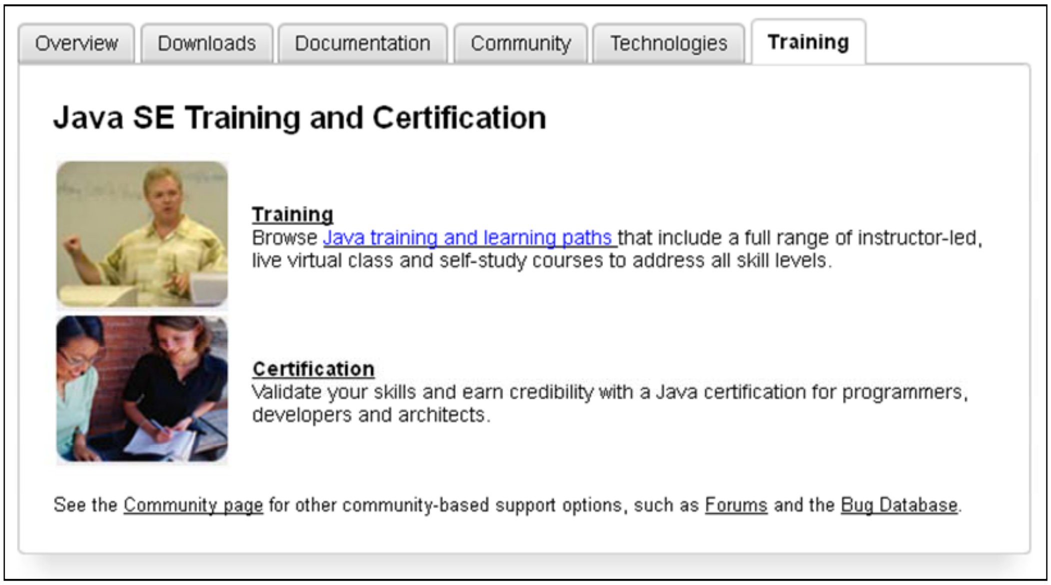
Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Java Technologies page includes a click map that describes all the Java SE Platform technologies in detail.

http://www.oracle.com/technetwork/java/javase/tech/index.html

Oracle Internal & Oracle Academy Use Only

Java Training



Java SE Training and Certification

Training
Browse Java training and learning paths that include a full range of instructor-led, live virtual class and self-study courses to address all skill levels.

Certification
Validate your skills and earn credibility with a Java certification for programmers, developers and architects.

See the Community page for other community-based support options, such as Forums and the Bug Database.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The "Java SE Training and Certification" page describes the available Java training as well as the Java Certification program. Oracle University offers courses that will introduce you to the Java programming language and technology so you can code smarter and develop robust programs and applications more quickly using any platform, including Oracle's application server and web infrastructure software. Validate your competency and dedication with a Java Certification—one of the most recognized credentials in the industry.

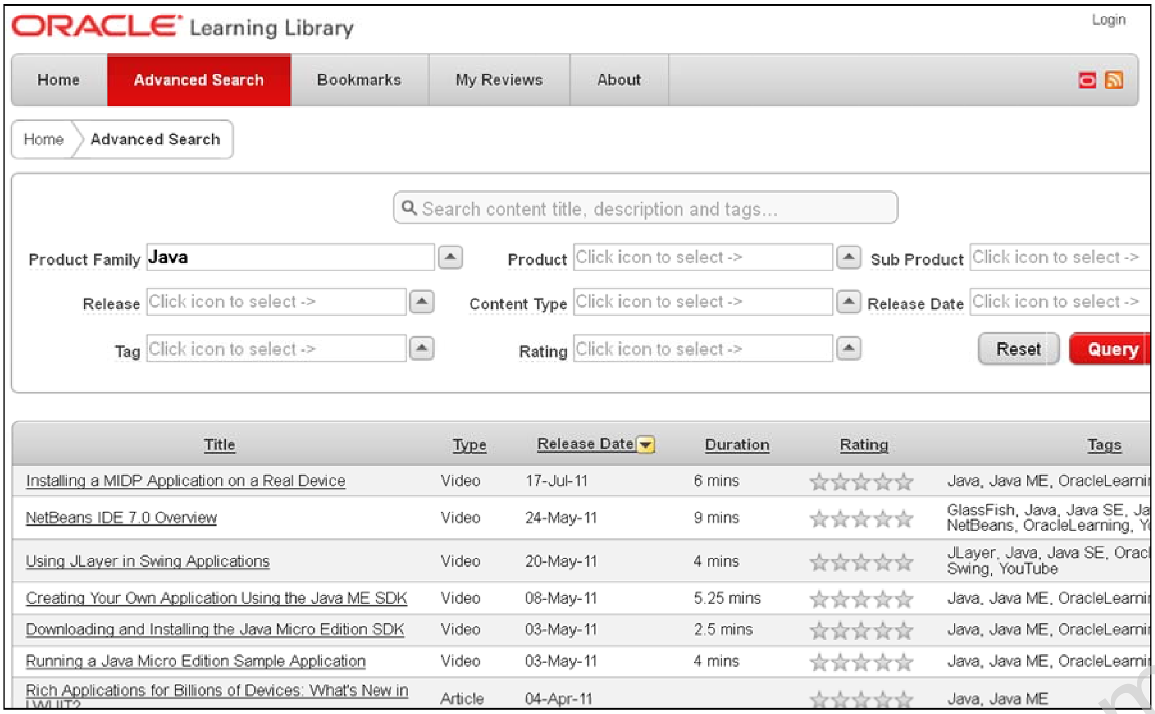
The latest Java SE training courses include:

- *Java SE 7 New Features*
- *Java Performance Tuning and Optimization*
- *Java SE 7 Fundamentals*
- *Java SE 7 Programming*

http://www.oracle.com/technetwork/java/javase/training/index.html

Oracle Internal & Oracle Academy Use Only

Oracle Learning Library



Search content title, description and tags...

Product Family: Java

Product: Click icon to select ->

Sub Product: Click icon to select ->

Release: Click icon to select ->

Content Type: Click icon to select ->

Release Date: Click icon to select ->

Tag: Click icon to select ->

Rating: Click icon to select ->

Reset | Query

Title	Type	Release Date	Duration	Rating	Tags
Installing a MIDP Application on a Real Device	Video	17-Jul-11	6 mins	☆☆☆☆☆	Java, Java ME, OracleLearni
NetBeans IDE 7.0 Overview	Video	24-May-11	9 mins	☆☆☆☆☆	GlassFish, Java, Java SE, Ja NetBeans, OracleLearning, Y
Using JLayer in Swing Applications	Video	20-May-11	4 mins	☆☆☆☆☆	JLayer, Java, Java SE, Orac Swing, YouTube
Creating Your Own Application Using the Java ME SDK	Video	08-May-11	5.25 mins	☆☆☆☆☆	Java, Java ME, OracleLearni
Downloading and Installing the Java Micro Edition SDK	Video	03-May-11	2.5 mins	☆☆☆☆☆	Java, Java ME, OracleLearni
Running a Java Micro Edition Sample Application	Video	03-May-11	4 mins	☆☆☆☆☆	Java, Java ME, OracleLearni
Rich Applications for Billions of Devices: What's New in LWUIT?	Article	04-Apr-11		☆☆☆☆☆	Java, Java ME

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

The Oracle Learning Library (OLL) features technical articles, white papers, videos, demonstrations, and Oracle by Example (OBE) tutorials on many topics, including Java. The site does require an Oracle Technology Network (OTN) login, but all content is free of charge. The OLL is available at oracle.com/oll.

Oracle Internal & Oracle Academy Use Only

Java Magazine

NEW APPS.
Java magazine
By and for the Java community
// PREMIERE ISSUE 2011 /
7 is Here
7 DON'T TOUCH THAT DIAL
Java and Globo TV make interactive television a reality in Brazil
38 RESOURCE INJECTION WITH JAVA EE 6
Master annotations, master configured resources
45 POLYGLOT PROGRAMMER
Dick Wall: What Scala teaches us about the strengths and limits of the JVM
The Java platform and ecosystem move forward. Learn how Java SE 7 addresses new trends in programming and hardware architectures.

Copyright © 2011, Oracle and/or its affiliates. All rights reserved.

Subscribe to *Java Magazine*, a bi-monthly magazine that is an essential source of knowledge about Java technology, the Java programming language, and Java-based applications for people who rely on them in their professional careers—or who aspire to.

http://www.oracle.com/technetwork/java/javamagazine/index.html

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy
Use Only