Oracle Fusion Middleware 11*g*: Java Programming

Volume II • Student Guide

Oracle University and Global Business Solutions Peru SAC use only

D53983GC11

Edition 1.1

May 2009

D60391

ORACLE®

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Author

Kate Heap

Technical Contributors and Reviewers

Ken Cooper

Clay Fuller

Taj Islam

Peter Laseau

Yvonne Price

Editors

Daniel Milne

Joyce Raftery

Graphic Designer

Satish Bettegowda

Publishers

Pavithran Adka

Nita Brozowski

Copyright © 2009, Oracle. 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 is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Contents

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

11

Arrays and Collections

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Describe how to create arrays of primitives and objects
- Process command-line variables
- Handle groups of objects using the Java Collections Framework

Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

This lesson discusses the manipulation of groups of primitives and objects. The first part of the lesson shows you how to create and use arrays. The second part introduces the Java Collections Framework and shows how you can employ the different interfaces of the framework to satisfy different requirements.

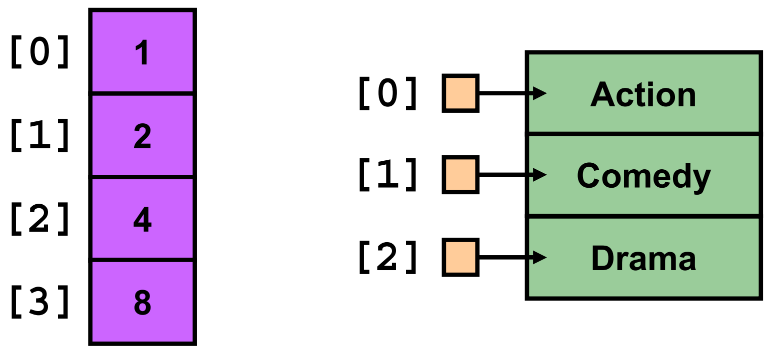
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Arrays

An array is a collection of variables of the same type.

- Each element can hold a single item.
- Items can be primitives or object references.
- The length of the array is fixed when it is created.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Arrays

Arrays are useful when you want a group of objects that you can manipulate as a whole. For example, if you are writing a program to allow users to search for a movie, you would probably store the list of movie categories in an array.

The slide shows an array of four integers and an array of three strings. The following slides show how to create and initialize the arrays. As you will see, an array in Java is an object.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Array of Primitives

1. Declare the array.

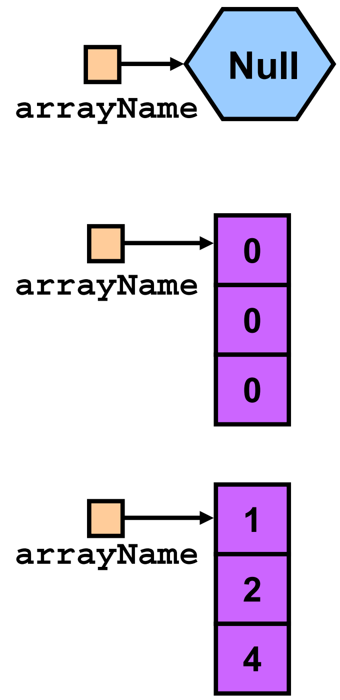
```
type[] arrayName;
… or …
type arrayName[];
```

`type` is a primitive, such as `int` and so on.

2. Create the array object.

```
// Create array object syntax
arrayName = new type[size];
```

3. Initialize the array elements (optional).



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating an Array of Primitives

1. **Declaration:** Create the variable that references the array.
2. **Creation:** Create an array object of the required type and size. Then store a reference to the array in the array variable.
3. **Initialization:** Initialize the array elements to the values that you want. This is optional for an array of primitives because the elements are initialized to default values when the array object is created.

The following slides explain these three steps in detail.

Oracle University and Global Business Solutions Peru SAC use only

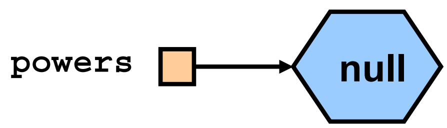
THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Declaring an Array

- Create a variable to reference the array object:

```
int[] powers;   // Example
```

- When an array variable is declared:
  - Its instance variable is initialized to `null` until the array object has been created



  - Its method variable is unknown until the object is created

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Declaring an Array

There are two ways to declare an array:

Syntax	Example
`type[] arrayname;`	*`int[] powers;`*
`type arrayname[];`	*`int powers[];`*

Most Java programmers use the first style because it separates the variable type (in the example, an array of `int`) from the variable name, making the code clearer to read.

When you declare an array variable, it refers to `null` initially until you initialize the array by using `new`.

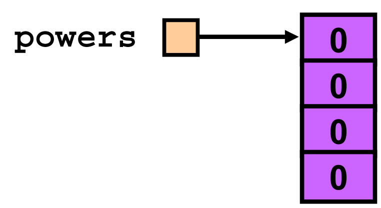
Oracle University and Global Business Solutions Peru SAC use only
THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Array Object

- Create an array of the required length and assign it to the array variable:

```
int[] powers;            // Declare array variable
powers = new int[4];  //Create array object
```

  – Create the array object by using the `new` operator.
- The contents of an array of primitives are initialized automatically.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating an Array Object

When using the `new` operator, you must specify the array size inside the brackets. The size must be an integer but does not have to be a constant number; it could be an expression that is evaluated at run time.

After the array object has been created, its length is fixed for the lifetime of the array.

Default Initialization of Array Elements

All elements in a new array of primitives are initialized automatically with default values, as follows:

- `char` elements are set to `\u0000`.
- `byte`, `short`, `int`, and `long` elements are set to `0`.
- `boolean` elements are set to `false`.
- `float` and `double` elements are set to `0.0`.

Note: `\u0000` is Unicode 0000. Java uses the Unicode character set.

Examples of Valid Array Creation

Example 1

```
final int SIZE = 4;
int[] powers = new int[SIZE];    // SIZE is a constant
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Array Object (continued)

Example 2

```
int[] examMarks;
int num = askUserHowManyStudents(); // set the value of num
examMarks = new int[num];           // array is a fixed size
```

Examples of Invalid Array Creation

Example 1

```
int powers[4]; // Invalid syntax: you can’t set the size of
               // the array in the declaration statement.
```

Example 2

```
int num;
int[] examMarks = new int[num]; // Compilation error: num has
                                // not been initialized
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Initializing Array Elements

- Assign values to individual elements:

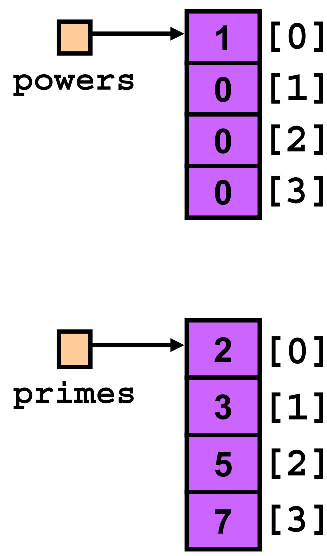
```
arrayName[index] = value;
```

```
powers[0] = 1;
```

- Create and initialize arrays at the same time:

```
type[] arrayName = {valueList};
```

```
int[] primes = {2, 3, 5, 7};
```



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Initializing Array Elements

First Method: Assign a Value to Each Array Element

To refer to an element in an array, use an index in brackets (`[]`), as shown in the slide. Array elements are numbered from `0` to `n-1`, where `n` is the number of elements in the array. In other words, the index of the first element in an array is `0` rather than `1`.

Second Method: Use Array Initializers

As shown in the slide, there is a shorthand technique for creating and initializing an array of primitives. Here, there is no need to use the `new` operator, and the length of the array is set automatically. Note the use of the braces, and remember the semicolon at the end.

Array initializers are very useful for creating lookup tables, as in the following example:

```
int[] daysInMonth = {31, 28, 31, 30, 31, 30,
                     31, 31, 30, 31, 30, 31};
```

This method is useful only if the value of each element is known when the array is created.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Array of Object References

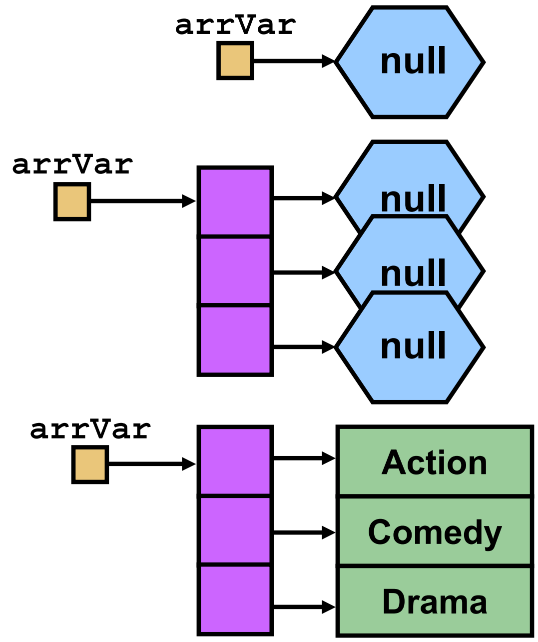
1. Declare the array.

```
ClassName[] arrVar;
… or …
ClassName arrVar[];
```

2. Create the array object.

```
// Create array object syntax
arrVar = new ClassName[size];
```

3. Initialize the objects in the array.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating an Array of Object References

The steps for creating an array of object references are the same as for arrays of primitives, with one exception: You must initialize the elements in the array because this is not done automatically.

1. **Declaration:** The syntax is the same as for arrays of primitive objects. For example,
   `String[] categories;`
   declares a variable that can point to an array of `String`. If the variable is the instance variable, the variable is set to `null` initially.
2. **Creation:** The syntax is the same as for arrays of primitive objects. For example,
   `categories = new String[3];`
   creates an array object of the correct type (`String`) and a size of `3`. Initially, all the elements are set to `null`.
   You can declare and create an array in the same statement.
   Example: `String[] categories = new String[3];`
3. **Initialization:** Initialize the array elements to the values that you want. This is described in the next slide.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Initializing the Objects in an Array

- Assign a value to each array element:

```
// Create an array of four empty Strings
String[] arr = new String[4];
for (int i = 0; i < arr.length; i++) {
      arr[i] = new String();
}
```

- Create and initialize the array at the same time:

```
String[] categories =
            {"Action", "Comedy", "Drama"};
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Initializing the Objects in an Array

As with arrays of primitives, there are two ways of initializing an array of object references. You can initialize the array by assigning a value to each array element or by initializing the array when you create it.

`length` Property

Every array has a `length` attribute that contains the number of elements in the array. By using `length`, you can avoid the need to hardcode or store the size of an array in your code. Because the index of the first element in an array is `0`, the index of its last element is `length – 1`.

The example in the slide uses `length` to loop through all the elements of an array to create an array of empty strings.

`length` Property (continued)

Incidentally, the `System` class provides a useful method for copying all or part of an array to another array. For more information, refer to `System.arraycopy()` in the Java Development Kit (JDK) documentation.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using an Array of Object References

- Any element can be assigned to an object of the correct type:

```
String category = categories[0];
```

- Each element can be treated as an individual object:

```
System.out.println
     ("Length is " + categories[2].length());
```

- An array element can be passed to any method; array elements are passed by reference.

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Passing Arrays to Methods

Arrays behave like objects. When an array is passed into a method, it is therefore passed by reference (like any other object). If the method changes the contents of the array, these changes operate on the original array and not on a copy.

```
For (String category: categories{
     System.out.println("Category: " + category);
}
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Going Through the Array Elements

- Use a Loop to explore each element in the array.

```
for (int i = 0;i < categories.length; i++){
   System.out.println("Category: "+categories[i]);
}
```

- Java 5.0 provides this alternative enhanced syntax.

```
for (String category: categories) {
     System.out.println ("Category: " +category);
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Exploring Array Elements

In most cases, finding the elements that a program wants to manipulate requires that the program loop through the array and examine each element in the array.

The above syntax shows two different ways you can use to go through each array element.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Arrays and Exceptions

- `ArrayIndexOutOfBoundsException` occurs when an array index is invalid:

```
String[] list = new String[4];
//The following throws ArrayIndexOutOfBoundsException
System.out.println(list[4]);
```

- `NullPointerException` occurs when you try to access an element that has not been initialized:

```
Movie[] movieList = new Movie[3];
// The following will throw NullPointerException
String director = movieList[0].getDirector();
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Arrays and Exceptions

The slide shows the exceptions, or errors, that occur when you try to perform an invalid operation on an array. Exceptions are covered in more detail in the lesson titled "Throwing and Catching Exceptions." You are likely to see these errors if your code attempts to perform one of the operations that is described in the slide.

If you try to access an invalid array index, your program will crash with the error "`ArrayIndexOutOfBoundsException`."

If you try to access an array element that has not been initialized, your program will crash with the error "`NullPointerException`."

Oracle University and Global Business Solutions Peru SAC use only

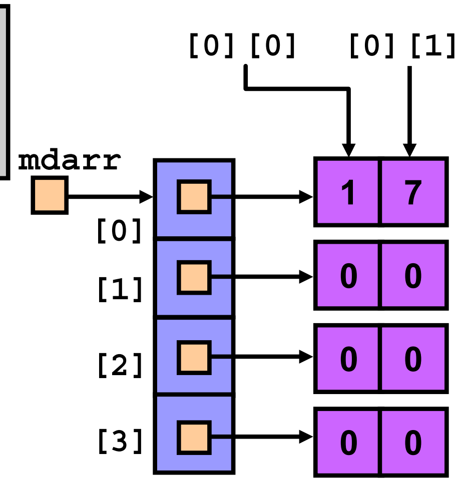
THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Multidimensional Arrays

Java supports arrays of arrays:

```
type[][] arrayname = new type[n1][n2];
```

```
int[][] mdarr = new int[4][2];
mdarr[0][0] = 1;
mdarr[0][1] = 7;
```



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Multidimensional Arrays

Java supports multidimensional arrays (that is, arrays of arrays):

```
int[][] tax = new int[5][4];
```

This declares and creates a two-dimensional matrix; the matrix contains five rows, each of which contains four columns. Individual elements can be accessed as follows:

```
tax[rowIndex][colIndex] = value;
```

Advanced Topic: Nonsquare Multidimensional Arrays

The following example creates a multidimensional array with 10 rows, but the number of columns in each row is different: the first row has one element, the second row has two elements, and so on.

```
int[][] a = new int[10][];
for (int i = 0; i < a.length; i++) {
      a[i] = new int[i + 1];
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Passing Command-Line Parameters to `main()`

- `main()` has a single parameter: `args`.
- `args` is an array of `Strings` that holds command-line parameters:

```
C:\> java SayHello Hello World
```

```
public class SayHello {
  public static void main(String[] args) {
    if (args.length != 2)
     System.out.println("Specify 2 arguments");
    else
      System.out.println(args[0]+" "+args[1]);
  } …
```

Copyright © 2009, Oracle. All rights reserved.

Passing Command-Line Parameters to `main()`

A reference to an array can be passed to any method. A good example is the `main()` method that is used in Java applications. When you start a Java application, as opposed to a Java applet, the system locates and calls the `main()` method for that class.

The `main()` method has a single parameter, which is a reference to an array of `String` objects. Each `String` object holds a command-line parameter; the first element in the array contains the first command-line parameter, not the name of the program as in C and C++.

Command-Line Parameters Are Always Converted to `Strings`

It is important to note that command-line parameters are always represented by `String` objects. Inside the `main()` method, you may need to convert a parameter to a primitive type. For example, if one of the command-line parameters represents a number, you may need to convert it into an `int` to perform some arithmetic with it.

Specifying Command-Line Parameters in JDeveloper

JDeveloper has a dialog box that you can use to specify command-line parameters for a Java application. When you run the application from the JDeveloper environment, JDeveloper passes the parameters into the `main()` method, as usual.

To specify command-line parameters in JDeveloper, select Tools > Project Properties from the menu bar, and then click the Run/Debug node. Click the Edit button and specify the program arguments as command-line parameters.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Collections Framework

Java Collections Framework is an API architecture for managing a group of objects that can be manipulated independently of their internal implementation. It is:

- Found in the `java.util` package
- Defined by six core interfaces and some implementation classes:
  - `Collection` interface: A generic group of elements
  - `Set` interface: A group of unique elements
  - `List` interface: An ordered group of elements
  - `Map` interface: A group of unique keys and their values
  - `SortedSet` and `SortedMap` for a sorted `Set` and `Map`

Copyright © 2009, Oracle. All rights reserved.

Java Collections Framework

The Java Collections Framework is an API architecture for managing a collection of objects that can be manipulated independently of their internal implementation. The framework is a unified architecture for representing and manipulating collections. All collections frameworks contain three things: interfaces, implementations, and algorithms.

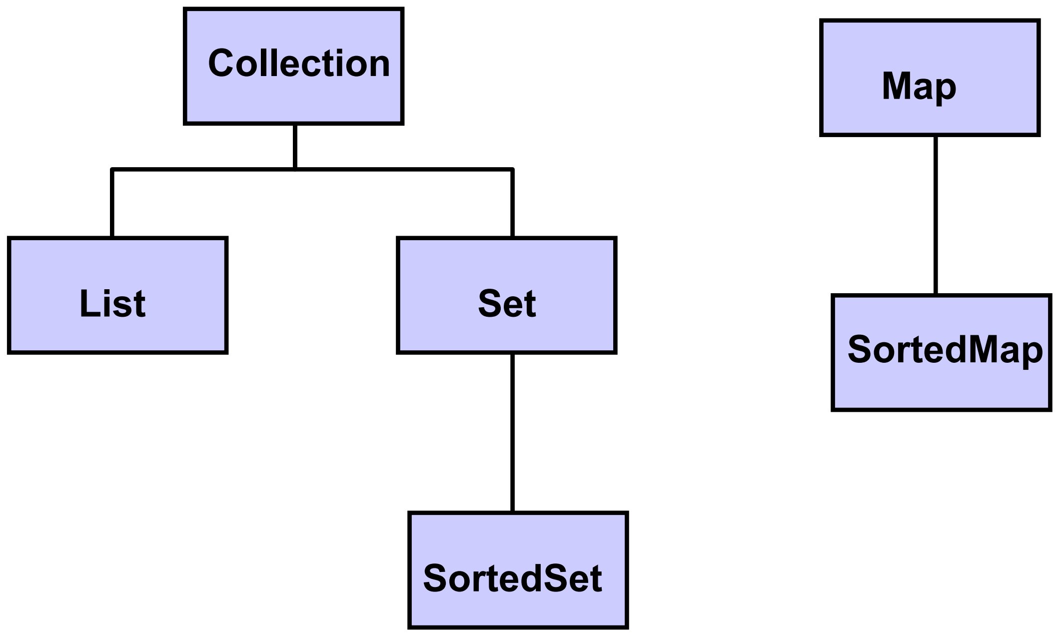
A *collection* is a container object that stores a group of objects, often referred to as *elements*. The Java Collections Framework supports three major types of collections: *set, list* and *map*, which are defined in the interfaces `Set`, `List`, and `Map`.

- The `Collection` interface is an abstraction representing a group of elements.
- The `Set` interface models mathematical *set* abstraction. It is a collection that cannot contain duplicate elements.
- The `List` interface represents an *ordered* collection (or sequence) of elements, including duplicates. Lists provide control over where each element is inserted. Elements can be accessed by their integer index (position).
- The `Map` interface represents an object that maps one or more keys to their values. Maps do not contain duplicate keys, and each key maps to a single value.
- Sorted collections are provided through the `SortedSet` and `SortedMap` interfaces.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Framework Interface Hierarchy



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Framework Interface Hierarchy

As you saw on the earlier slide, the Collections Framework is made up of a set of interfaces for working with groups of objects. The different interfaces describe the different types of groups. Though you always need to create specific implementations of the interfaces, access to the actual collection should be restricted to the use of the interface methods, thus allowing you to change the underlying data structure without altering the rest of your code.

The diagram on the slide shows the framework interface hierarchy. One might think that `Map` would extend `Collection`. In mathematics, a map is just a collection of pairs. In the Collections Framework however, the interfaces `Map` and `Collection` are distinct, with no lineage in the hierarchy. The reasons for this distinction have to do with the ways that `Set` and `Map` are used in the Java technology libraries. The typical application of a `Map` is to provide access to values stored by keys. The set of collection operators are all there, but you work with a key-value pair, instead of an isolated element. `Map` is therefore designed to support the basic operations of `get()` and `put()` which are not required by `Set`.

When designing software with the Collections Framework, it is useful to remember the following hierarchical relationships of the four basic interfaces of the framework:

- The `Collection` interface is a group of objects with duplicates allowed.
- `Set` extends `Collection` but forbids duplicates.
- `List` extends `Collection`, allows duplicates, and introduces positional indexing.
- `Map` extends neither `Set` nor `Collection`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Collections Framework Components

The Java Collections Framework is a set of interfaces and classes used to store and manipulate groups of data as a single unit.

- Core interfaces are the interfaces used to manipulate collections and to pass them from one method to another.
- Implementations are the actual data objects used to store collections; the data structures implement the core collection interface.
- Algorithms are pieces of reusable functionality provided by the JDK.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Collections Framework Components

The design of programs often requires handling groups of objects. The Collections Framework offers a set of standard utility classes to manage the collection of these objects. The framework is made up of three main components:

- **Core interfaces:** These allow collections to be manipulated independently of their implementation. These interfaces describe a common set of functionality, displayed by collections, and enhance data exchange between collections. In object-oriented languages, these interfaces are generally contained within a hierarchy.
- **Implementations:** A small set of implementations exist as concrete implementations of the core interfaces, which provide a data structure that a program can use. In a sense, these are reusable data structures. The implementations come in three flavors: general-purpose, wrapper, and convenience.
- **Algorithms:** Methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. These algorithms are said to be polymorphic because the same method can be used on many different implementations of the appropriate collections interface. In essence, algorithms are reusable functionality.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

The `Collection` Interface and the `AbstractCollection` Class

- The `Collection` interface:
  - Is the root interface for manipulating a collection of objects.
  - Provides the basic operations for adding and removing elements in a collection.
  - Provides various query operations.
  - Provides the `toArray` method that returns an array representation for the collection.
- The `AbstractCollection` class is a convenience class that provides partial implementation for the `Collection` interface. It implements all the methods in `Collection`, except the `size` and `iterator` methods.
- The `Iterator` interface is used for traversing elements in a collection.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

The `Collection` Interface and the `AbstractCollection` Class

The `Collection` interface provides the basic operations for adding and removing elements in a collection. The `add` method adds an element to the collection, the `addAll` method adds all the elements in the specified collection to this collection. `remove()` removes an element from the collection, and `removeAll()` removes the elements from this collection that are present in the specified collection. The `retainAll` method retains the elements in this collection that are also present in the specified collection. All these methods return `boolean`. The return value is `true` if the collection is changed as a result of the method execution. The `clear` method simply removes all the elements from the collection.

The `Collection` interface also provides various query operations. The `size` method returns the number of elements in the collection. The `contains` method checks whether the collection contains all the elements in the specified collection. The `isEmpty` method returns `true` if the collection is empty.

The `Iterator` interface provides a uniform way for traversing elements in various types of collections. The `iterator` method in the `Collection` interface returns an instance of the `Iterator` interface, which provides sequential access to the elements in the collection using the `next()` method.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`Iterator` Interface

The `Iterator` interface can be used to process a series of objects. The `java.util.Iterator` interface:

- Implements an object-oriented approach for accessing elements in a collection
- Replaces the `java.util.Enumeration` approach
- Contains the following methods:
  - `hasNext()` returns `true` if more elements exist.
  - `next()` returns the next `Object`, if any.
  - `remove()` removes the last element returned.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

`Iterator` Interface

Collections differ from arrays in that the members of a collection are not directly accessible using indices, as you would with arrays. When using `Enumeration` or `Iterator`, you can move the current item pointer to only the first or next element of a collection. `Enumeration` was part of the standard Java API, and `Iterator` was added with Java Collections Framework. `Iterator` supports the removal of an object from the collection, whereas `Enumeration` can only traverse the collection.

The following example creates an `ArrayList` containing several `String` elements, and then calls the `iterator()` method to return an `Iterator` object. The loop uses the `next()` method of `Iterator` to get elements and display their string value in uppercase.

When `hasNext()` is `false`, the loop terminates.

```
import java.util.ArrayList;
import java.util.Iterator;
:
ArrayList al = new ArrayList();
al.add("Jazz");
al.add("Classical");
al.add("Rock 'n Roll");
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`Iterator` Interface (continued)

```
for (Iterator e = al.iterator();
e.hasNext(); ) {
String s = (String) e.next();
  System.out.println(s.toUpperCase());
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Sets

- The `Set` interface extends the `Collection` interface.
- The concrete classes that implement `Set` must ensure that no duplicate elements can be added to the set.
- `AbstractSet` extends `AbstractCollection` and implements `Set`.
- Three concrete classes of `Set` are:
  - `HashSet`
  - `LinkedHashSet`
  - `TreeSet`

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Sets

The `Set` interface extends the `Collection` interface. It does not introduce new methods or constants, but it stipulates that an instance of `Set` contains no duplicate elements.

The `AbstractSet` class is a convenience class that extends `AbstractCollection` and implements `Set`. The `AbstractSet` class provides concrete implementations for the `equals` method and the `hashCode` method. The hash code of a set is the sum of the hash codes of all the elements in the set.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

HashSet

- A `HashSet` can be used to store duplicate-free elements.
- You can create an empty hash set using its no-arg constructor, or create a hash set from an existing collection.
- Objects added to a hash set need to implement the `hashCode` method in a way that properly disperses the hash code.
- The hash codes of two objects must be the same if the objects are equal.
- Two unequal objects may have the same hash code, but you need to implement the `hashCode` method to avoid having too many such cases.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

HashSet

HashSet example.

The following program finds all the words used in a piece of text. The program creates a hash set to store the words extracted from the text and uses an iterator to traverse the elements in the set.

```
import java.util.*;

public class TestHashSet {
  public static void main(String[] args) {
     //Create a hash set
     Set<String> set = new HashSet<String>();

     //Add strings to the set
     set.add("London");
     set.add("Paris");
     set.add("New York");
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`HashSet` (continued)

```
            set.add("San Francisco");
            set.add("Beijing");
            set.add("New York");

            System.out.println(set);

            //Display the elements in the hash set
            for (Object element: set)
                System.out.print(element.toString() + " ");
        }
    }
```

Note: The program adds string elements to a hash set, displays the elements using the `toString` method, and traverses the elements using an iterator. "New York" is added to the set more than once, but only one is stored because a set does not allow duplicates.
When you run the program you see that the strings are not stored in the order in which they were inserted into the set. There is no particular order for the elements in a hash set. To impose an order on them, you need to use the `LinkedHashSet` class, which is introduced in the next slide.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

LinkedHashSet

- `LinkedHashSet` was added in JDK 1.4.
- It supports the ordering of elements in a set.
- A `LinkedHashSet` can be created by using its `no-arg` constructor.

Copyright © 2009, Oracle. All rights reserved.

LinkedHashSet

LinkedHashSet example.

This example rewrites the previous example using `LinkedHashSet`.

```
import java.util.*;

public class TestLinkedHashSet {
  public static void main(String[] args) {
    //Create a linked hash set
    Set<String> set = new LinkedHashSet<String>();

    //Add strings to the set
    set.add("London");
    set.add("Paris");
    set.add("New York");
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`LinkedHashSet` (continued)

```
            set.add("San Francisco");
            set.add("Beijing");
            set.add("New York");

            System.out.println(set);

            //Display the elements in the linked hash set
            for (Object element: set)
                System.out.print(element.toString() + " ");
        }
    }
```

Note: When you run this program, you see that the `LinkedHashSet` maintains the order in which the elements are inserted. To impose a different order, you need to use the `TreeSet` class, which is introduced on the next slide.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

TreeSet

- `TreeSet` is a concrete class that implements the `SortedSet` interface.
- `SortedSet` is a subinterface of `Set` that guarantees that the elements in the set are sorted.
- `SortedSet` provides the methods:
  - `first()` and `last()` for returning the first and last elements in the set
  - `headSet(toElement)` and `tailSet(fromElement)` for returning a portion of the set whose elements are less than `toElement` and greater than `fromElement`
- You can add elements into a tree set as long as they can be compared with each other.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

TreeSet

You must be able to compare the elements in a tree set with each other. There are two ways to compare objects:

- Use the `Comparable` interface. Since the objects added to the set are instances of `Comparable`, they can be compared using the `compareTo` method. Several classes in the Java API—such as `String`, `Date`, `Calendar`, and all wrapper classes for the primitive types—implement the `Comparable` interface.
- If the class for the elements does not implement the `Comparable` interfaces, or if you do not want to use the `compareTo` method in the class that implements the `Comparable` interface, specify a comparator for the elements in the set.

TreeSet Example

This example orders elements using the `Comparable` interface. It rewrites the previous example to display the words in alphabetical order using the `TreeSet` class.

```
import java.util.*;
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`TreeSet` (continued)

```
public class TestTreeSet {
  public static void main(String[] args) {
     //Create a hash set
     Set<String> set = new HashSet<String>();

     //Add strings to the set
     set.add("London");
     set.add("Paris");
     set.add("New York");
     set.add("San Francisco");
     set.add("Beijing");
     set.add("New York");

     TreeSet<String> treeSet = new TreeSet<String>(set);
     System.out.println(treeSet);

     //Display the elements in the hash set
     for (Object element: set)
        System.out.print(element.toString() + " ");
     }
}
```

Note: The program creates a hash set filled with strings, and then creates a tree set for the same strings. The strings are sorted in the tree set using the `compareTo` method in the `Comparable` interface.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Lists

- The `List` interface extends `Collection` to define an ordered collection with duplicates allowed.
- The `List` interface adds position-oriented operations.
- A list iterator enables you to traverse the list in both directions.
- The `AbstractList` class provides a partial implementation for the `List` interface.
- The `AbstractSequentialList` class extends `AbstractList` to provide support for linked lists.
- The `ArrayList` class and the `LinkedList` class are two concrete implementations of the `List` interface.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Lists

A set stores nonduplicate elements. To allow duplicate elements to be stored in a collection, you need to use a list. A list can not only store duplicate elements, but also allows the user to specify where they are stored. The user can access elements via an index.

The `List` interface has the following methods:

• `add(index, element)`	Adds an new element at the specified index
• `addAll(index, collection)`	Inserts a collection at the specified index
• `remove(index)`	Removes an element at the specified index from the list
• `set(index, element)`	Sets a new element at the specified index
• `listIterator`	Returns the list iterator for the elements in this list
• `listIterator(startIndex)`	Returns the iterator for the elements from `startIndex`

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

ArrayList

The ArrayList class:

- Stores elements in an array
- Is a resizable implementation of the List interface
- Allows manipulation of the array size
- Has capacity that grows as elements are added to the list

To create an empty ArrayList:

```
ArrayList members = new ArrayList();
```

To create an ArrayList with an initial size:

```
// Create an ArrayList with 10 elements.
ArrayList members = new ArrayList(10);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ArrayList

The `ArrayList` class, belonging to the `java.util` package, provides a resizable collection of objects. Remember that Java arrays are fixed in size, so an `ArrayList` is useful when you do not know how large an array will be at the time that you create it. For example, you may get a list of names from a server and want to store the names in a local array. Before you fetch the data from the server, you have no idea how large the list is.

The `ArrayList` class provides methods to modify and access the `ArrayList`.

This class is roughly equivalent to Vector, except that it is unsynchronized.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Modifying an `ArrayList`

- Add an element to the end of the `ArrayList`:

```
String name = MyMovie.getNextName();
members.add(name);
```

- Add an element at a specific position:

```
// Insert a string at the beginning
members.add(0, name);
```

- Remove the element at a specific index:

```
// Remove the first element
members.remove(0);
```

Copyright © 2009, Oracle. All rights reserved.

Modifying an `ArrayList`

When you add an element to an `ArrayList`, the `ArrayList` is expanded by one element. When you remove an element from an `ArrayList`, the size of the `ArrayList` decreases. When you insert an element at a specific position, all elements after that position increase their indexes by 1.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Accessing an `ArrayList`

- Get the first element:

```
String s = members.get(0);
```

- Get an element at a specific position:

```
String s = members.get(2);
```

- Find an object in an `ArrayList`:

```
int position = members.indexOf(name);
```

- Get the size of an `ArrayList` :

```
int size = members.size();
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

`ArrayList` Indexing

The index of the first element of an `ArrayList` is 0.

Example of an `ArrayList` Containing Different Objects

You can combine different types in an `ArrayList`. Here is a simple example:

```
ArrayList al = new ArrayList();
al.add('pat');
al.add(10);
al.add(123456.789);
System.out.println(al); // to print the ArrayList
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`LinkedList`

- `LinkedList`:
  - Is an implementation of the `List` interface
  - Provides the methods for retrieving, inserting, and removing elements from both ends of the list
  - Can be constructed using its no-arg constructor or `LinkedList(Collection)`
- `ArrayList` and `LinkedList` are operated similarly; the critical difference is internal implementation that affects their performance.
  - `ArrayList` is efficient for retrieving elements and for inserting and removing elements from the end of the list.
  - `LinkedList` is efficient for inserting and removing elements anywhere in the list.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

`LinkedList` Example

The following example creates an `ArrayList` filled with numbers and inserts new elements into specified locations in the list. It then creates a linked list from the `ArrayList` and inserts and removes elements from the list. Finally, it traverses the list forwards and backwards:

```
import java.util.*;

public class TestArrayAndLinkedList {
   public static void main(String[] args) {
     List<Integer> arrayList = new ArrayList<Integer>();
     arrayList.add(1);  //1 is autoboxed to new Integer(1)
     arrayList.add(2);
     arrayList.add(3);
     arrayList.add(1);
     arrayList.add(4);
     arrayList.add(0, 10);
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
arrayList.add(3, 30);

System.out.println("A list of integers in the array
list:");
System.out.println(arrayList);

LinkedList<Object> linkedList = new
LinkedList<Object>(arrayList);
linkedList.add(1, "red");
linkedList.removeLast();
linkedList.addFirst("green");
System.out.println("Display the linked list
forwards:");
ListIterator listIterator = linkedList.listIterator();
while (listIterator.hasNext()) {
  System.out.print(listIterator.next() + " ");
}
System.out.println();

System.out.println("Display the linked list
backwards:");
listIterator =
linkedList.listIterator(linkedList.size());
while (listIterator.hasPrevious()) {
  System.out.print(listIterator.previous() + " ");
}
}
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Maps

- The `Collection` interface represents a collection of elements stored in a set or a list.
- The `Map` interface maps keys to the elements.
- The keys are like indexes.
- The keys can be any objects.
- Each key maps to one value.
- The `Map` interface provides the methods for querying, updating, and obtaining a collection of values and a set of keys.

Copyright © 2009, Oracle. All rights reserved.

Maps

The query methods include `containsKey`, `containsValue`, `isEmpty`, and `size`. The `containsKey(Object key)` method checks whether the map contains a mapping for the specified key. The `containsValue(Object value)` method checks whether the map contains a mapping for this value. The `isEmpty()` method checks whether the map contains any mappings. The `size()` method returns the number of mappings in the map.

The update methods include `clear`, `put`, `putAll`, and `remove`. The `clear()` method removes all mappings from the map. The `put(Object key, Object value)` method associates the specified value with the specified key in the map. If the map formerly contained a mapping for this key, the old value associated with the key is returned. The `putAll(Map m)` method adds the specified map to this map. The `remove(Object key)` method removes the map elements for the specified key from the map.

You can obtain a set of the keys in the map using the `keySet` method and a collection of the values in the map using the values method. The `entrySet` method returns a collection of objects that implement the `Map.Entry` interface, where `Entry` is an inner interface for the `Map` interface. Each object in the collection is a specific key-value pair in the underlying map.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Types of Maps

`HashMap`, `LinkedHashMap`, and `TreeMap` are three concrete implementations of the `Map` interface.

- `HashMap` is efficient for locating a value, inserting a mapping and deleting a mapping.
- `LinkedHashMap` supports ordering of the entries in a map.
- `TreeMap` is efficient for traversing the the keys in a sorted order.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Types of Maps

Three types of maps are supported: `HashMap`, `LinkedHashMap`, and `TreeMap`.

- `HashMap` is efficient for locating a value, inserting a mapping, and deleting a mapping.
- `LinkedHashMap` supports ordering of the entries in the map. The entries in a `HashMap` are are not ordered, but the entries in a `LinkedHashMap` can be retrieved either in the order in which they were inserted into the map (known as the *insertion order*), or in the order in which they were last accessed, from the least recently accessed to the most recently (*access order*) accessed.
- The `TreeMap` class, implementing `SortedMap`, is efficient for traversing the keys in a sorted order. The keys can be sorted using the `Comparable` interface or the `Comparator` interface. If you create a `TreeMap` using its `no-arg` constructor, the `compareTo` method in the `Comparable` interface is used to compare the elements in the set, assuming that the class of the elements implements the `Comparable` interface. To use a comparator, you have to use the `TreeMap(Comparator comparator)` constructor to create a sorted map that uses the `compare` method in the comparator to order the elements in the map, based on the keys.

Note: Prior to JDK 1.2, Map was supported in `java.util.Hashtable`. Since then, `Hashtable` was redesigned to fit into the Java Collections Framework with all its methods retained for compatibility. `Hashtable` implements the `Map` interface and is used the same way as `HashMap`, except that `Hashtable` is synchronized.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Example of Using Maps

The example below creates a hash map, a linked map, and a tree map that map students to ages.

Copyright © 2009, Oracle. All rights reserved.

Types of Maps (continued)

Map Example

The program first creates a hash map with the student's name as its key and the age as its value. It then creates a tree map from the hash map and displays the mappings in ascending order of the keys. Finally, it creates a linked hash map, adds the same entries to the map, and displays the entries.

```
import java.util.*;

public class testMap {
   public static void main (String[] args) {
     // Create a HashMap
     Map<String, Integer> hashMap = new HashMap<String,
                                Integer>();
     hashMap.put("Smith", 30);
     hashMap.put("Anderson", 31);
     hashMap.put("Lewis", 29);
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
            hashMap.put("Cook", 29);

            System.out.println("Display entries in HashMap");
            System.out.println(hashMap);

            //Create a TreeMap from the HashMap
            Map<String, Integer> treeMap = new TreeMap<String,
                                       Integer>(hashMap);

            System.out.println("\nDisplay entries in ascending
                                       order of key");
            System.out.println(treeMap);

            //Create a LinkedHashMap
            Map<String, Integer> linkedHashMap = new
           LinkedHashMap<String, Integer>(16, 0.75f, true);
            linkedHashMap.put("Smith", 30);
            linkedHashMap.put("Anderson", 31);
            linkedHashMap.put("Lewis", 29);
            linkedHashMap.put("Cook", 29);

            //Display the age for Lewis
            System.out.println("The age for " + "Lewis is " +
                  linkedHashMap.get("Lewis").intValue());

            System.out.println("\nDisplay entries in
                                            LinkedHashMap");
            System.out.println(linkedHashMap);
    }
}
```

Note1: When you run the program you will see that the entries in the `HashMap` are in random order. The entries in the `TreeMap` are in increasing order of the keys. The entries in the `LinkedHashMap` are in the order of their access, from the least recently accessed to the most recently accessed.

Note2: The example above uses generics. You will learn about generics in the next lesson.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Create Java arrays of primitives
- Create arrays of object references
- Initialize arrays of primitives or object references
- Process command-line arguments in the `main()` method
- Use the Java Collections Framework to manage collections of objects
- Use the core interfaces of the Java Collections API: the `Collection`, `Set`, `List`, `Map`, `SortedSet`, and `SortedMap` interfaces

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 11 Overview: Using Arrays and Collections

This practice covers the following topics:

- Modifying the `DataMan` class
- Creating an array to hold the `Customer`, `Company`, and `Individual` objects
- Adding a method to ensure that the array is successfully created and initialized
- Adding a method to find a customer by an ID value

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Practice 11 Overview: Using Arrays and Collections

The goal of this practice is to gain experience with Java array objects and work with collection classes such as the `java.util.ArrayList` class. You also work with command-line arguments.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les11` directory, load the `OrderEntryApplicationLes11` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, load the `OrderEntryApplicationLes11` application. In the Applications – Navigator node, expand `OrderEntryApplicationLes11 – OrderEntryProjectLes11 - Application Sources – oe` and double-click the `UML Class Diagram1` entry. This diagram displays all the classes created up to this point in the course.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

12

Using Generic Types

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this course, you should be able to do the following:

- Identify how generics can be used to improve software reliability and readability
- Declare and use generic classes and interfaces
- Declare and use generic methods
- Use wildcard types

Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

This lesson describes how to declare and use generic classes, interfaces, and methods, and explains how they can be used to improve software reliability and readability.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Generics

- Generics is the ability to parameterize types.
- The most common examples are container types such as those in the Collection hierarchy.
- The use of generic types eliminates the need for unnecessary casting when dealing with objects in a Collection.
- Angle brackets are used to provide type parameters to parameterized types.

```
List<String> al = new ArrayList<String>();
```

- Using generic types helps to avoid unexpected type errors that can occur at run time, thus making your code more robust.

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Generics

Generics was added to the Java language as part of Java SE 5.0. It allows applications to create classes and objects that can operate on any defined types.

Example of the benefits of generics:

Consider the following four lines of code:

```
List al = new ArrayList();
al.add("test");
String s = (String) al.get(0);
Integer i = (Integer) al.get(0);
```

`ArrayList` contains a list of untyped objects, but in order to use the objects the program needs to cast them to their correct data type. If the casting is incorrect, the error will not be detected by the compiler, but it will fail at run time with a `java.lang.ClassCastException`. It is cumbersome and error-prone to have to cast types each time you use an object from the collection.

Using generics the code can be rewritten as follows:

```
List<String> al = new ArrayList <String> ();
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Generics (continued)

```
al.add("test");
String s = al.get(0);
Integer i = al.get(0);
```

Note the code in the angle brackets <>. The angle brackets are the syntax for providing type parameters to parameterized types. You now do not need to cast the result into a `String` type because the `al` reference is of type `List<String>`, so you know its method `.get()` returns a `String`.

Compiling the above code with JDK 1.5 will now cause a type error in the final line, trying to cast a `String` to an `Integer`. This type of error is now caught at compile time instead of at run time.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Declaring Generic Classes

- Imagine a simple class called `ObjectHolder` that can hold any type of Java object.
- Declare the `ObjectHolder` class:

```
public class ObjectHolder<O> { }
```

- `ObjectHolder` is a parametric class.
- `O` is a type parameter.
- `O` serves as a place holder for holding any type of object.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Declaring Generic Classes

The `ObjectHolder` class can hold any type of Java object. It provides methods for getting and setting the current object.

```
public class ObjectHolder<O> {
     private O anyObject;
     public O getObject(); {
          return anyObject;
     }
     public void setObject(O anyObject) {
          this.anyObject = anyObject;
     }
     public String toString() {
          return anyObject.toString();
     }
}
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using Generic Classes

- To instantiate a `String` instance of the `ObjectHolder` class:

```
ObjectHolder<String> stringHolder = new
                        ObjectHolder<String>();
```

- *Type substitution* is used to replace the type parameter `O` with the type `String`.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Using Generic Classes

In the example below, the `ObjectHolderClient` class uses the `ObjectHolder` class to create a `stringHolder` object for holding a `String` and a `urlHolder` object to hold a URL:

```
public class ObjectHolderClient  {
     public static void main(String [] args) throws
                                  Exception  {
        ObjectHolder<String> stringHolder = new
                          ObjectHolder<String>();
        stringHolder.setObject(new String("String"));
        System.out.println(stringHolder.toString());


        ObjectHolder<URL> urlHolder = new
                             ObjectHolder<URL>();
        urlHolder.setObject(new
          URL("http://www.oracle.com/technology/
                      products/jdev/11/index.html"));
        System.out.println(urlHolder.toString());
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Generic Methods

- You can also use generic types to declare generic methods.

```
public static <E> void print(E[] list)
```

- A generic method containing one or more type parameters affects that method only.
- A non-generic class can contain a mixture of generic and non-generic methods.
- To invoke a generic method you prefix the method name with the actual type in angle brackets.

```
GenericMethod.<String>print(strings);
```

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Generic Methods

Example: Generic Method

The following program declares a generic method `print` to print an array of objects. First an array of integer objects is passed to invoke the generic `print` method, and then `print` is invoked with an array of strings:

```
public class GenericMethod  {
     public static void main(String [] args) {
        Integer[] integers = {1,2,3,4,5};
        String[] strings = {"Dracula", "Titanic",
                  "Deliverance", "Casablanca"};

        GenericMethod.<Integer>print(integers);
        GenericMethod.<String>print(strings);
     }

     public static <E> void print(E[] list) {
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
         System.out.println();
       }
        }
```

Example: Generic Method Used With a Collection

This example uses the generic `max` method to compute the greatest value in a collection of elements of an unknown type `A`.

```
class Collections {
    public static <A extends Comparable<A>> A
                              max(Collection< A > xs) {
     Iterator< A > xi = xs.iterator();
     A w = xi.next();
     while (xi.hasNext()) {
       A w = xi.next();
       if (w.compareTo(x) < 0) w = x;
     }
     return w;
    }
}
```

The max method has one parameter named `A`. It is a placeholder for the element type of the collection that the method works on. The type parameter has a bound; it must be a type that is a subtype of `Comparable<A>`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Wildcards

- The character `?` is a wildcard character.
- `?` stands for any Java type, a placeholder that can have any type assigned to it.

```
List<?> anyObjects = null;
```

- `List<?>` indicates a list which has an unknown object type.
- The use of wildcards is necessary because objects of one type parameter cannot be converted to objects of another parameter.

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Wildcards

Generic type parameters in Java are not limited to specific classes. Java allows the use of wildcards to specify bounds on the type of parameters a given generic object may have. Wildcards are type parameters of the form "`?`."

Because the exact element type of an object with a wildcard is unknown, restrictions are placed on the type of methods that may be called on the object. As an example of an unbounded wildcard, `List<?>` indicates a list that has an unknown object type. Methods that take such a list as an argument, can take any type of list, regardless of parameter type. Reading from the list will return objects of type `Object`, and writing non-null elements to the list is not allowed, since the parameter type is not known.

To specify the upper bound of a generic element, the `extends` keyword is used, indicating that the generic type is a subtype of the bounding class. Thus it must either extend the class, or implement the interface of the bounding class. So `List<? extends Number>` means that the list contains objects of some unknown type that extends the `Number` class; for example, the list could be `List<Float>` or `List<Number>`.

The use of wildcards is necessary since objects of one type parameter cannot be converted to objects of another parameter. Neither `List<Float>` nor `List<Number>` is subtype of the other (even though `Float` is a subtype of `Number`). So code that deals with `List<number>` does not work with `List<Float>`. The solution with wildcards works because it disallows operations that would violate type safety.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Wildcards (continued)

To specify the lower bound of a generic element, the `super` keyword is used, indicating that the generic type is a supertype of the bounding class. So `List<? super Number>` could be `List<Number>` or `List<Object>`. Reading from the list returns objects of type `Object`. Any element of type `Number` can be added to the list since it is guaranteed to be a valid type to store in the list.

Example: Using a Generic Method and Wildcards

The following method prints out all the elements in a collection. Note that the `Collection<?>` is the collection of an unknown type.

```
void printCollection(Collection<?> c) {
     for(Object o:c) {
     System.out.println(o);
     }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Raw Types

- Are nonparameterized types

```
ArrayList rawlist = new ArrayList();
```

- Are allowed in JDK 1.5 (and later) for backwards compatibility
- Are assignment-compatible with all instantiations of the generic type

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Raw Types

The generic type without any type arguments, like `ArrayList`, is called a *raw type*. Raw types are permitted in the language predominantly to facilitate interfacing with non-generic (legacy) code. If, for instance, you have a non-generic legacy method that takes a `List` as an argument, you can pass a parameterized type such as `List<String>` to that method. Conversely, if you have a method that returns a `List`, you can assign the result to a reference variable of type `List<String>`, provided you know, for some reason, that the returned list really is a list of strings.

Raw types are assignment-compatible with all instantiations of the generic type. Assignment of an instantiation of a generic type to the corresponding raw type is permitted without warnings; assignment of the raw type to an instantiation results in an "unchecked conversion" warning:

```
ArrayList rawList = new ArrayList():
ArrayList<String> stringList = new
                         ArrayList<String>();
rawList = stringList;
stringList = rawList;     //unchecked warning
```

The "unchecked" warning indicates that the compiler does not know whether the raw type `ArrayList` really contains strings. A raw type `ArrayList` can, in principle, contain any type of object and is similar to an `ArrayList<Object>`.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Type Erasure

Type erasure is the way in which the compiler implements generics:

- Removes all information that is related to type parameters and type arguments
- Can be thought of as a translation from generic Java source code back into regular Java code
  - `List<String>` becomes `List`
  - `Set<Long>` becomes `Set`
  - `Map<String>` becomes `Map`
- Generates casts to the appropriate type

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Type Erasure

Type erasure is an automatic process, internal to the compiler, that gets rid of (*erases*) all generic type information. All the type information between the angle brackets is removed, so, for example, a parameterized type like `List<String>` is converted into `List`. All remaining uses of type variables are replaced by the upper bound of the type variable (usually `Object`). And whenever the resulting code is not type-correct, a cast to the appropriate type is generated.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Use generics to help avoid casting errors and make your code more robust
- Declare a generic type in a class, interface, or method
- Use wildcards to specify bounds on the type of parameters a given generic object may have
- Identify raw types—nonparameterized types that are allowed for backwards compatibility
- Describe type erasure—the translation process that the compiler uses to implement generics

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

13

Structuring Code Using Abstract Classes and Interfaces

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Define abstract classes
- Define abstract methods
- Define interfaces
- Identify the similarities and differences between an abstract class and an interface
- Implement interfaces

Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

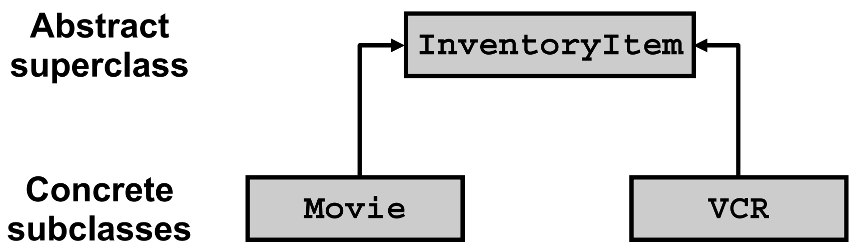
In Java, interfaces can be used as an effective alternative to multiple inheritances. This lesson shows you how to do this. You also learn how abstract classes and abstract methods can be defined and used in Java.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Abstract Classes

- An abstract class cannot be instantiated.
- Abstract methods must be implemented by subclasses.
- Interfaces support multiple inheritance with regard to type.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Abstract Classes

In Java, you can define classes that are high-level abstractions of real-world objects. Using these high-level classes gives the designer control over what subclasses look like and even which methods are mandatory in the subclass.

An abstract class is simply a class that cannot be instantiated; only its nonabstract subclasses may be instantiated. For example, an `InventoryItem` does not contain sufficient detail to provide anything meaningful to the business. It must be either a movie or a VCR. An `InventoryItem` does, however, serve as a collection of data and behaviors that are common to all items that are available for rent.

Abstract Methods

Abstract methods go a step beyond standard inheritance. An abstract method is defined only within an abstract class and must be implemented by a subclass. The class designer can use this technique to decide exactly what behaviors a subclass must be able to perform. The designer of the abstract class cannot determine how the behaviors will be implemented—only that they will be implemented. Abstract classes may, however, contain subclasses, which are also declared abstract. In such cases the abstract methods will not be implemented by the (abstract) subclasses.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Abstract Classes (continued)

Interfaces

An interface is the specification of a set of methods, which is similar to an abstract class. In addition to what an abstract class offers, an interface can effectively provide multiple inheritances. A class can implement an unlimited number of interfaces but can extend only one superclass.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating Abstract Classes

Use the `abstract` keyword to declare a class as abstract.

```
public abstract class InventoryItem {
  private float price;
  public boolean isRentable()…
}
```

```
public class Movie
extends InventoryItem {
 private String title;
 public int getLength()…
```

```
public class Vcr
extends InventoryItem {
 private int serialNbr;
 public void setTimer()…
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating Abstract Classes

Java provides the `abstract` keyword, which indicates that a class is abstract. For example, the `InventoryItem` class in the slide has been declared as abstract:

```
public abstract class InventoryItem {
  …
}
```

`InventoryItem` is declared abstract because it does not possess enough intelligence or detail to represent a complete and stand-alone object. The user must not be allowed to create `InventoryItem` objects because `InventoryItem` is only a partial class. The `InventoryItem` class exists only so that it can be extended by more specialized subclasses, such as `Movie` and `Vcr`.

What Happens If You Try to Instantiate an Abstract Class?

If you try to create an `InventoryItem` object anywhere in the program, the compiler flags an error:

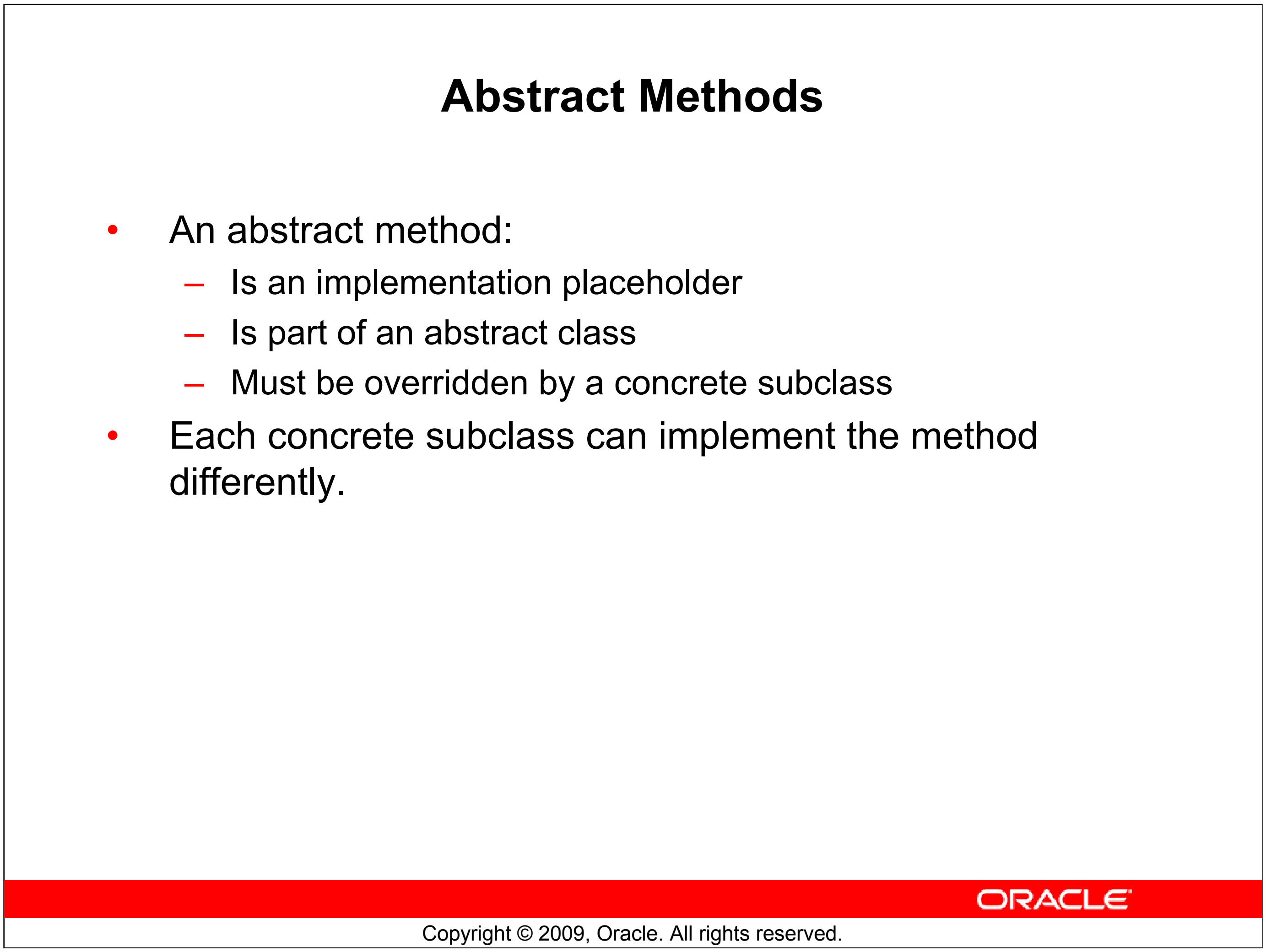
```
InventoryItem i = new InventoryItem (…);// Compiler error
```

The user can only create objects of the concrete subclasses:

```
Movie   m = new Movie(…);        // This is fine
Vcr v = new Vcr(…);          // This is fine too
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED



Oracle University and Global Business Solutions Peru SAC use only

Abstract Methods

When you design an inheritance hierarchy, there will probably be some operations that all classes perform, each in its own way. For example, in a video rental business, the vendor must know whether each item is rentable or not. Each type of item, however, determines whether the item is rentable in a specific way.

To represent this concept in Java, the common "is this item rentable?" method is defined in the `InventoryItem` class. However, there is no sensible implementation for this method in `InventoryItem` because each kind of item has its own requirements. One approach may be to leave the method empty in the `InventoryItem` class:

```
public abstract class InventoryItem{
  public boolean isRentable() {
    return true;
  }
}
```

This approach is not good enough because it does not force each concrete subclass to override the method. For example, in the `Vcr` class, if the user forgets to override the `isRentable()` method, what will happen if the user calls the method on a `Vcr` object? The `isRentable()` method in `InventoryItem` will be called and always return `true`. This is not the desired outcome. The solution is to declare the method as abstract, as shown on the next slide.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Defining Abstract Methods

- Use the `abstract` keyword to declare a method as abstract.
  - Provide the method signature only.
  - Ensure that the class is also abstract.
- Why is this useful?
  - You can declare the structure of a given class without providing complete implementation of every method.

```
public abstract class InventoryItem {
  public abstract boolean isRentable();
  …
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Defining Abstract Methods

To declare a method as abstract in Java, prefix the method name with the `abstract` keyword as follows:

```
public abstract class InventoryItem {
  abstract boolean isRentable();
  …
}
```

When you declare an `abstract` method, you provide only the signature for the method, which comprises its name, its argument list, and its return type. You do not provide a body for the method. Each concrete subclass must override the method and provide its own body.

Now that the method is declared as abstract, a subclass must provide an implementation of that method.

Abstract classes can contain methods that are not declared as abstract. Those methods can be overridden by the subclasses, but this is not mandatory.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Defining and Using Interfaces

- An interface is like a fully abstract class.
  - All its methods are abstract.
  - All variables are `public static final`.
- An interface lists a set of method signatures without code details.
- A class that implements the interface must provide code details for all the methods of the interface.
- A class can implement many interfaces but can extend only one class.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Defining and Using Interfaces

An interface is similar to an abstract class, except that it cannot have concrete methods or instance variables. It is a collection of abstract method declarations and constants—that is, `static final` variables. It is like a contract that the subclass must obey.

Any class that implements an interface must implement some or all of the methods that are specified in that interface. If it does not implement all the methods, then the class is an abstract class; a subclass of the abstract class must implement the remaining abstract methods.

A class can implement many interfaces but can extend only one class. Java does not support inheritance from multiple classes, but it does support implementing multiple interfaces. For example:

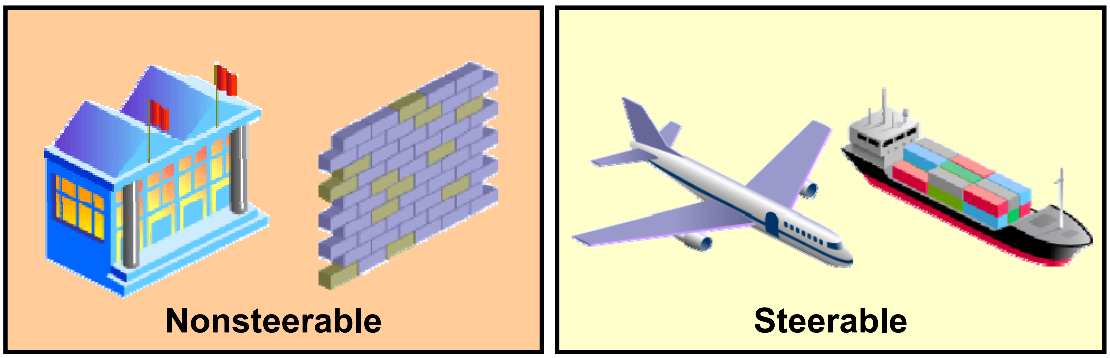
```
class Movie extends InventoryItem implements Sortable, Listable {
    …
  }
```

As demonstrated earlier, `Movie` inherits all the attributes and behaviors of `InventoryItem`. In addition, it now must provide implementation details for all the methods that are specified in the `Sortable` and `Listable` interfaces. Those methods can be used by other classes to implement specific behaviors (such as a sort routine).

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Examples of Interfaces

- Interfaces describe an aspect of behavior that different classes require.
- For example, classes that can be steered support the “steerable” interface.
- Classes can be unrelated.



Copyright © 2009, Oracle. All rights reserved.

Examples of Interfaces

Interfaces describe an aspect of behavior that many different classes require. The name of an interface is often an adjective such as `Steerable`, `Traceable`, or `Sortable`. This is in contrast to a class name, which is usually a noun such as `Movie` or `Customer`.

The `Steerable` interface may include such methods as `turnRight()`, `turnLeft()`, or `returnCenter()`. Any class that needs to be steerable can implement the `Steerable` interface.

The classes that implement an interface may be completely unrelated. The only thing that they may have in common is the need to be steered.

For example, the core Java packages include a number of standard interfaces such as `Runnable`, `Cloneable`, and `ActionListener`. These interfaces are implemented by all types of classes that have nothing in common except the need to be cloneable or to implement an action listener.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating Interfaces

- Use the `interface` keyword:

```
public interface Steerable {
  int MAXTURN = 45;
  void turnLeft(int deg);
  void turnRight(int deg);
}
```

- All methods are `public abstract`.
- All variables are `public static final`.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating Interfaces

You can define an interface by using the `interface` keyword. All methods that are specified in an interface are implicitly `public` and `abstract`. Any variables that are specified in an interface are implicitly `public`, `static`, and `final`—that is, they are constants.

Therefore, the interface definition that is shown in the slide is equivalent to the following definition, where the `public`, `static`, `final`, and `abstract` keywords have been specified explicitly:

```
public interface Steerable {
  public static final int MAXTURN = 45;
  public abstract void turnLeft(int deg);
  public abstract void turnRight(int deg);
}
```

Because interface methods are implicitly `public` and `abstract`, it is a generally accepted practice not to specify those access modifiers. The same is true for variables. Because they are implicitly `public`, `static`, and `final` (in other words, constants), you must not specify those modifiers.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Interfaces Versus Abstract Classes

	Variables	Constructors	Methods
Abstract Class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the `new` operator.	No restrictions
Interface	All variables must be `public static final`.	No constructors. An interface cannot be instantiated using the `new` operator.	All methods must be public abstract methods.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Interfaces Versus Abstract Classes

An interface can be used in the same way as an abstract class, but declaring an interface is different from declaring an abstract class. The table in the slide summarizes the differences.

Abstract classes and interfaces can both be used to model common features. How then do you decide whether to use an interface or a class? In general, a **strong** *is-a* relationship that clearly describes a parent-child relationship should be modeled using classes. For example, a customer is a person, so the relationship between them should be modeled using class inheritance. A **weak** *is-a* relationship, also known as an *is-a-kind-of* relationship, indicates that an object possesses a particular property. A weak *is-a* relationship can be modeled using interfaces. For example, all strings are comparable, so the `String` class implements the `Comparable` interface.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Implementing Interfaces

Use the `implements` keyword:

```
public class Yacht extends Boat
      implements Steerable {
  public void turnLeft(int deg) {…}
  public void turnRight(int deg) {…}
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Implementing Interfaces

The slide shows an example of a `Yacht` class that implements the `Steerable` interface. `Yacht` must implement some or all of the methods in any interface that it implements; in this case, `Yacht` can implement `turnLeft()` and `turnRight()`.

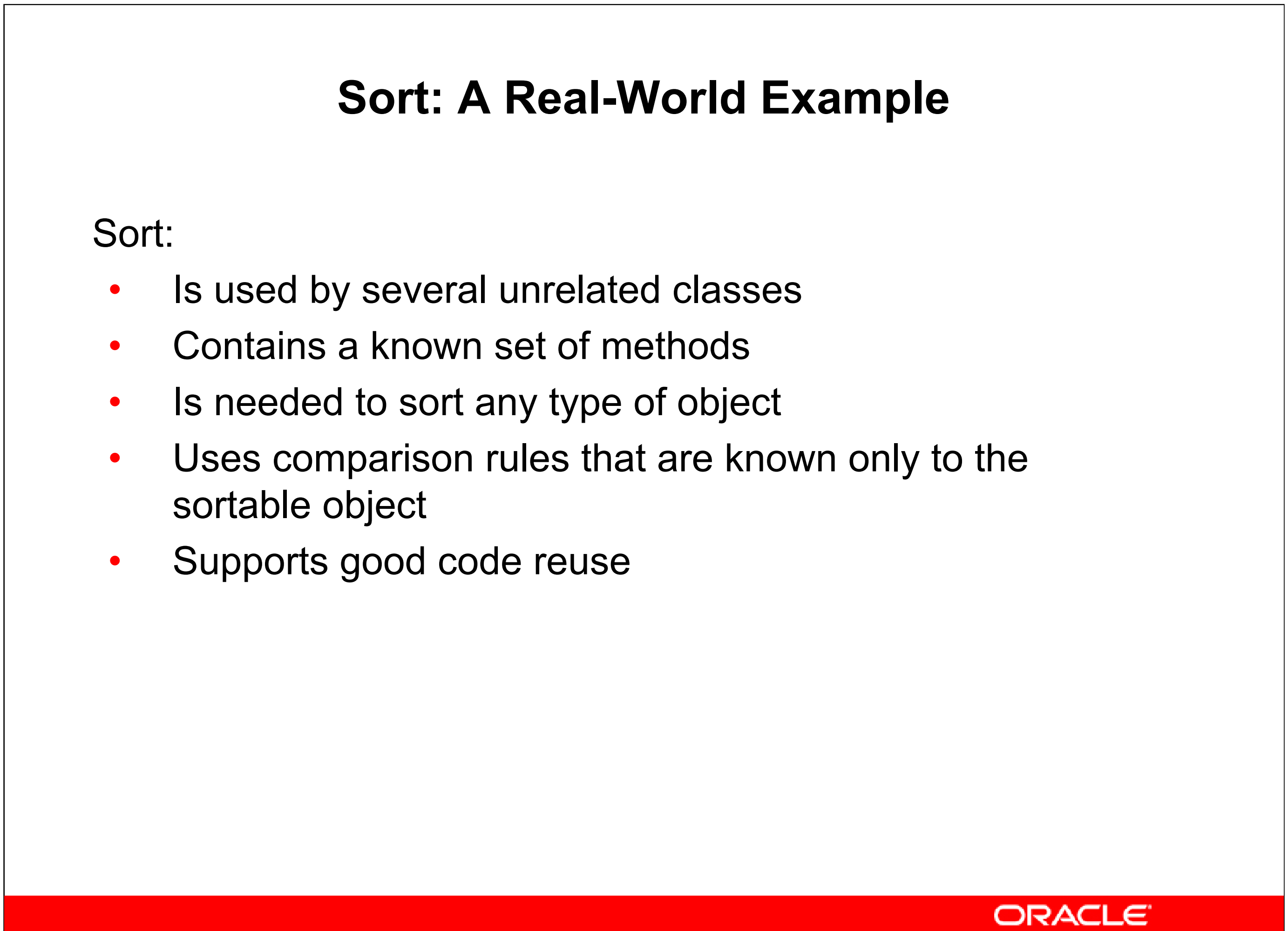
A class can implement more than one interface by specifying a list of interfaces separated by commas. Consider the following example:

```
public class Yacht
    extends Boat
    implements Steerable, Taxable {
  …
}
```

Here, the `Yacht` class implements two interfaces: `Steerable` and `Taxable`. This means that the `Yacht` class must implement all the methods that are declared in both `Steerable` and `Taxable`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED



Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Sort: A Real-World Example

A sort is a classic example of the use of an interface. Many completely unrelated classes must use a sort. A sort is a well-known and well-defined process that does not need to be written repeatedly.

A sort routine must provide the ability to sort any object in the way that fits that particular object. The traditional programming approach dictates several subroutines and an ever-growing decision tree to manage each new object type. By using good object-oriented programming techniques and interfaces, you can eliminate all the maintenance difficulties that are associated with the traditional approach.

The `Sortable` interface specifies the methods that are required to make the sort work on each type of object that needs to be sorted. Each class implements the interface based on its specific sorting needs. Only the class needs to know its object comparison, or sorting rules.

Implementing the sort in an object-oriented fashion provides a model that supports very good code reuse. The sort code is completely isolated from objects that implement the sort.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Overview of the Classes

- Created by the sort expert:

```
public interface
   Sortable
```

```
public abstract
   class Sort
```

- Created by the movie expert:

```
public class Movie
   implements Sortable
```

```
public class
   MyApplication
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Overview of the Classes

The slide shows the three classes and one interface that are involved in sorting a list of videos. The classes are divided into two categories:

- Classes that are created by the sort expert, who knows all about sort algorithms but nothing about individual objects that people may want to sort
- Classes that are created by the movie expert, who knows all about movies but nothing about sort algorithms

You see how interfaces can separate these two types of developers, enabling the separation of unrelated areas of functionality.

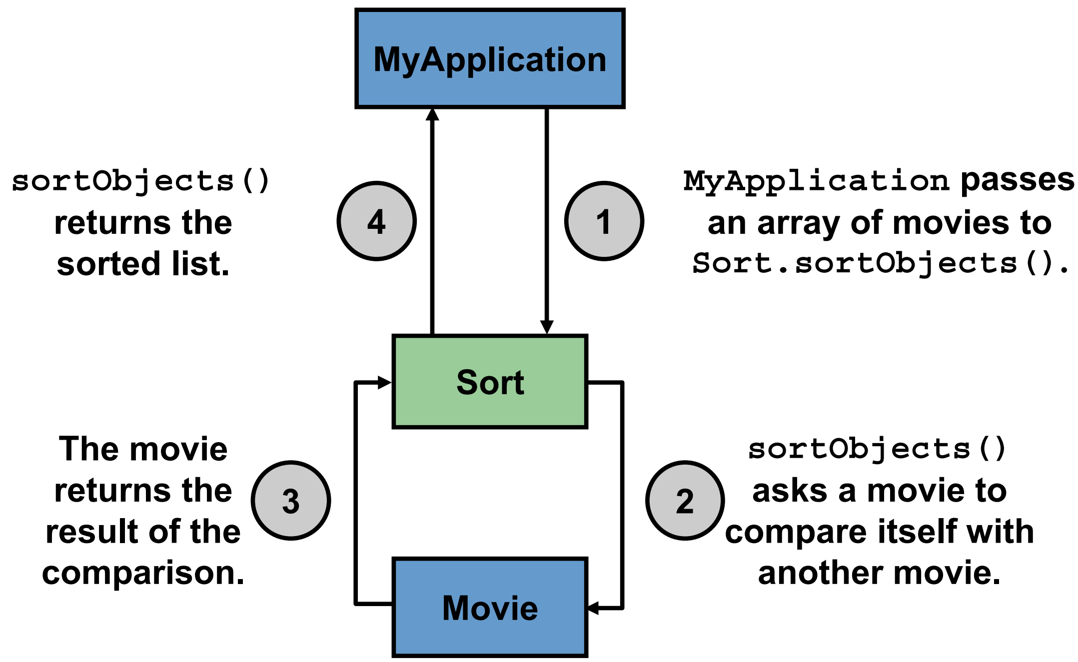
Classes and Interfaces Used by the Example

- The `Sortable` interface declares one method: `compare()`. This method must be implemented by any class that wants to use the sort class methods.
- The `Sort` class is an `abstract` class that contains `sortObjects()`, which is a method to sort an array of objects. Most sort algorithms work by comparing pairs of objects. `sortObjects()` does this comparison by calling the `compare()` method on the objects in the array.
- The `Movie` class implements the `Sortable` interface. It contains a `compare()` method that compares two `Movie` objects.
- `MyApplication` represents any application that must sort a list of movies. It can be a form displaying a sortable list of movies.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

How the Sort Works



Copyright © 2009, Oracle. All rights reserved.

How the Sort Works

The slide shows the process of sorting a list of objects. The steps are as follows:

1. The main application passes an array of movies to `Sort.sortObjects()`.
2. `sortObjects()` sorts the array. Whenever `sortObjects()` needs to compare two movies, it calls the `compare()` method of one movie, passing it with the other movie as a parameter.
3. The movie returns the results of the comparison to `sortObjects()`.
4. `sortObjects()` returns the sorted list.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`Sortable` Interface

Specifies the `compare()` method:

```
public interface Sortable {
  // compare(): Compare this object to another object
  // Returns:
  //     0 if this object is equal to obj2
  //     a value < 0 if this object < obj2
  //     a value > 0 if this object > obj2
  int compare(Object obj2);
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

`Sortable` Interface

The `Sortable` interface specifies all the methods and constants that are required for a class to be sortable. In the example, the only method is `compare()`.

Any class that implements `Sortable` must provide a `compare()` method that accepts an `Object` argument and returns an `int`.

The result of the `compare()` method is as follows:

Value	Meaning
Positive integer	This object is greater than the argument.
Negative integer	This object is less than the argument.
Zero	This object is equal to the argument.

Note: It is entirely up to the implementer of `compare()` to determine the meaning of "greater than," "less than," and "equal to."

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`Sort` Class

Holds `sortObjects()`:

```
public abstract class Sort {
  public static void sortObjects(Sortable[] items) {
    // Step through the array comparing and swapping;
    // do this length-1 times
    for (int i = 1; i < items.length; i++) {
      for (int j = 0; j < items.length - 1; j++) {
        if (items[j].compare(items[j+1]) > 0) {
          Sortable tempitem = items[j+1];
          items[j+1] = items[j];
          items[j] = tempitem; } } } } }
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

`Sort` Class

The `Sort` class contains the `sortObjects()` method, which sorts an array of `Sortable` objects. `sortObjects()` accepts an array of `Sortable` as its argument. It is legal syntax to specify an interface type for a method's argument; in this case, it ensures that the method will be asked to sort only objects that implement the `Sortable` interface. In the example, `sortObjects()` executes a simple sort that steps through the array several times and compares each item with the next one, swapping them if necessary.

When `sortObjects()` needs to compare two items in the array, it calls `compare()` on one of the items, passing the other item as the argument.

Note that `sortObjects()` knows nothing about the type of object that it is sorting. It knows only that they are `Sortable` objects, and therefore it knows that it can call a `compare()` method on any of the objects. It also knows how to interpret the results.

Interface as a Contract

You can think of an interface as a contract between the object that uses the interface and the object that implements the interface. In this case, the contract is as follows:

- The `Movie` class (the implementer) agrees to implement a method called `compare()`, with parameters and a return value specified by the interface.
- The `Sort` class (the user) agrees to sort a list of objects in the correct order.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Movie Class

Implements `Sortable`:

```
public class Movie extends InventoryItem
                        implements Sortable {
  String title;
  public int compare(Object movie2) {
    String title1 = this.title;
    String title2 = ((Movie)movie2).getTitle();
    return(title1.compareTo(title2));
  }
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

`Movie` Class

The `Movie` class implements the `Sortable` interface. To call `Sort.sortObjects()`, it must implement the `Sortable` interface, and if it implements the `Sortable` interface, then it must implement the `compare()` method. This is the contract. The `compare()` method takes an `Object` as an argument and compares it with the object on which it was called.

In this case, you use the `String compareTo()` method to compare the two title strings. `compareTo()` returns a positive integer, a negative integer, or zero depending on the relative order of the two objects. When implementing `compare()`, you can compare the two objects in any way you like, as long as you return an integer that indicates their relative sort order.

Note: In the example, `movie2` is an `Object`, so it must be cast to `Movie` before you can call `getTitle()` to get its title.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using the Sort

Call `Sort.sortObjects(Sortable [])` with an array of `Movie` as the argument:

```
class myApplication {
  Movie[] movielist;
  …      // build the array of Movie
  Sort.sortObjects(movielist);
}
```

Copyright © 2009, Oracle. All rights reserved.

Using the Sort

To use the sort, you call `Sort.sortObjects(Sortable [])` from your application, passing the array of objects that you want sorted. Each object that you want to sort must implement the `Sortable` interface and provide the required `compare()` method. Only the class implementing `Sortable` knows exactly how its objects are sorted.

You can make other types of objects in your application sortable. For example, you can make the `Rental` and `Member` classes implement the `Sortable` interface and add a `compare()` method to each class. Then you can sort an array of `Rental` or `Member` by calling `Sort.sortObjects()`. The `compare()` method in each of the classes can be radically different or fundamentally the same. The only requirement is that the `compare()` methods return an integer to indicate the relative sort order of the objects.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using `instanceof` with Interfaces

- Use the `instanceof` operator to determine whether an object implements an interface.
- Use downcasting to call methods that are defined in the interface:

```
public void aMethod(Object obj) {
  …
  if (obj instanceof Sortable)
    ((Sortable)obj).compare(obj2);
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Using `instanceof` with Interfaces

In the lesson about inheritance, you learned how to use the `instanceof` operator to test whether the run-time type of an object matched a certain type.

You can also use `instanceof` with interfaces, as shown by the method in the slide. The method takes an argument whose compile-time type is `Object`. At run time, the argument can be any kind of object inherited from `Object`. The `instanceof` operator tests the object to see whether it is an `instanceof Sortable`. In other words, it verifies whether the object supports the `Sortable` interface.

This means that you do not care what kind of object you are dealing with. Your concern is whether the object is capable of having the `compare()` method called on it.

If the object does implement the `Sortable` interface, you cast the object reference into `Sortable` so that the compiler lets you call the `compare()` method.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Define abstract classes
- Define abstract methods
- Define interfaces
- Identify the similarities and differences between an abstract class and an interface
- Implement interfaces

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 13 Overview: Structuring Code Using Abstract Classes and Interfaces

This practice covers the following topics:

- Creating an interface and abstract class
- Implementing the `java.lang.Comparable` interface to sort objects
- Testing the abstract and interface classes

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 13: Overview

The goal of this practice is to learn how to create and use an abstract class and how to create and use an interface.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les13` directory, load the `OrderEntryApplicationLes13` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, in the Applications – Navigator node, expand `OrderEntryApplicationLes13 – OrderEntryProjectLes13 - Application Sources – oe` and double-click the `UML Class Diagram1` entry. This diagram displays all the classes created to this point in the course.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

14

Throwing and Catching Exceptions

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Explain the basic concepts of exception handling
- Write code to catch and handle exceptions
- Write code to throw exceptions
- Create your own exceptions

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

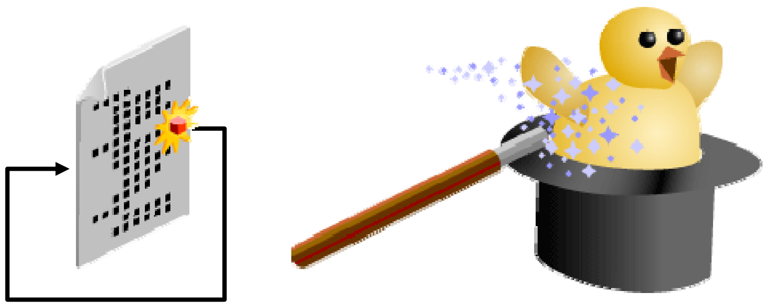
Many Java methods in the Java Development Kit (JDK) class library throw an exception when they encounter a serious problem that they do not know how to handle. This lesson explains how exceptions work in Java and shows you how to handle such exceptions in your applications.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

What Is an Exception?

An exception is an unexpected event.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

What Is an Exception?

An exception is an event during program execution that disrupts the normal flow of instructions. For example, trying to access an element outside the bounds of an array, trying to divide a number by zero, and trying to access a URL with an invalid protocol are all exceptions.

What Is an Error?

An error in Java is an unrecoverable abnormal condition. For example, an error condition exists if there is some internal error in the Java Virtual Machine (JVM) or if the JVM runs out of memory.

What Is the Difference?

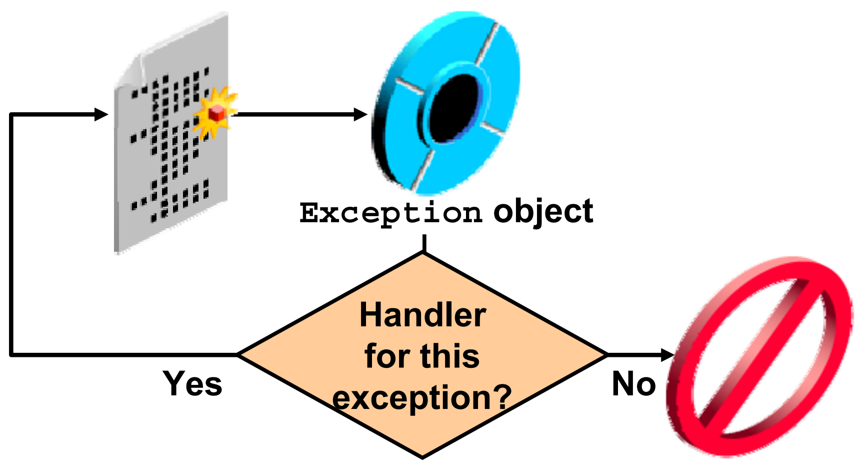
Your code can handle an exception and move on; if an error occurs, your program must exit.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Exception Handling in Java

1. A method throws an exception.
2. A handler catches the exception.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Exception Handling in Java

When an exception occurs within a Java method, the method creates an `Exception` object and hands it off to the run-time system. This process is called throwing an exception. The `Exception` object contains information about the exception, including its type and the state of the program when the error occurred.

When a Java method throws an exception, the run-time system searches all the methods in the call stack in sequence to find one that can handle this type of exception. In Java terminology, this method is called *catching the exception*.

If the run-time system does not find an appropriate exception handler, the whole program terminates.

The following slides discuss some of the advantages of Java's exception handling over traditional error handling in other languages.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Advantages of Java Exceptions: Separating Error-Handling Code

- In traditional programming, error handling often makes code more confusing to read.
- Java separates the details of handling unexpected errors from the main work of the program.
- The resulting code is clearer to read and, as a result, less prone to bugs.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Advantage 1: Separating Error-Handling Code from Other Code

In traditional programming, error handling often makes code more confusing to read.
For example, if you want to write a function that reads the first line of a file, the pseudocode for the function might be the following:

```
readFirstLine {
     open the file;   // the open could fail
     read the first line;  // the read could fail
     close the file; // the close could fail
}
```

The traditional way of checking for the potential errors in this function is to test each possible error and set an error code. The table on the next page compares the traditional method with Java's exception handling; the original three statements are in bold.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Advantage 1: Separating Error-Handling Code from Other Code (continued)

Traditional Error Handling	Java Exception Handling

`readFirstLine {`
`  int errcode = 0;`
`  `**`open the file;`**
`  if (openError) {`
`    errcode = OPEN_ERR;`
`  }`
`  else {`
`    `**`read the first line;`**
`    if (readError) {`
`      errcode = READ_ERR;`
`    }`
`    `**`close the file;`**
`    if (closeError) {`
`      errcode = errcode`
`        and CLOSE_ERR;`
`    }`
`  }`
`  return errcode;`
`}`
	`readFirstLine {`
	`  try {`
	`    `**`open the file;`**
	`    `**`read the first line;`**
	`    `**`close the file;`**
	`  }`
	`  catch (openError) {`
	`    handle error;`
	`  }`
	`  catch (readError) {`
	`    handle error;`
	`  }`
	`  catch (closeError) {`
	`    handle error;`
	`  }`
	`}`

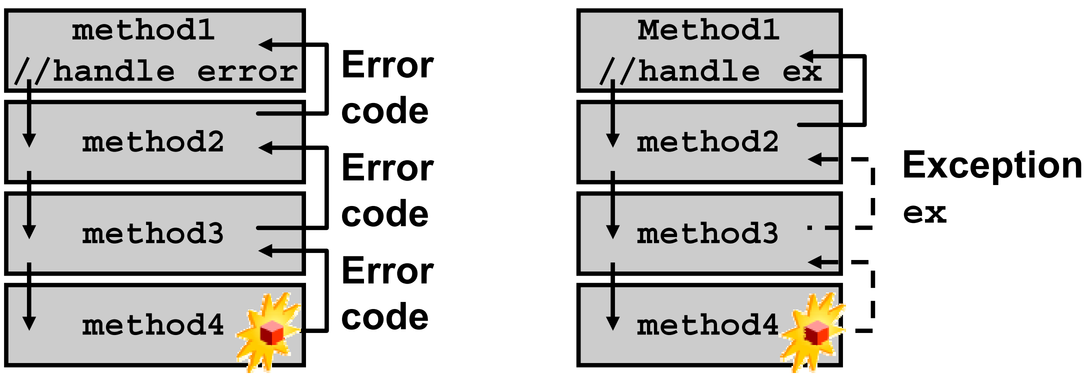
Java separates the details of handling unexpected errors from the main work of the program, thereby making the code clearer to read (which, in turn, makes it less prone to bugs).

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Advantages of Java Exceptions: Passing Errors Up the Call Stack

Traditional error handling | **Java exceptions**



Each method checks for errors and returns an error code to its calling method.

`method4` throws an exception; eventually `method1` catches it.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Advantage 2: Passing Errors Up the Call Stack

A Java exception is sent immediately to the appropriate exception handler; there is no need to have `if` statements at each level to pass the error up the call stack. For example, a series of nested methods can handle errors as follows:

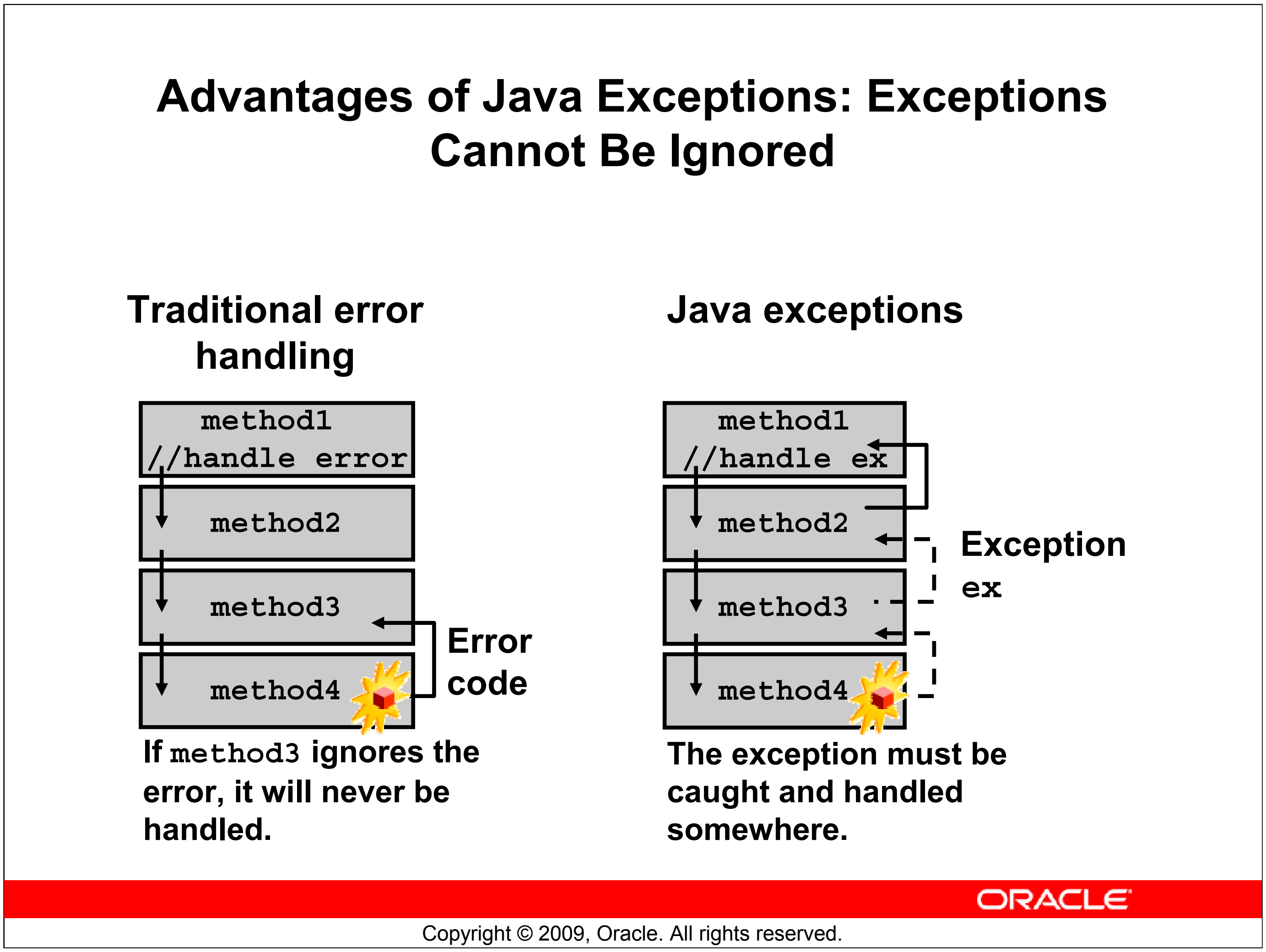
`method1` handles all errors
`method1` calls `method2`
`method2` calls `method3`
`method3` calls `method4`

The following table shows the steps that are taken by traditional error handling and by Java exception handling if an error occurs in `method4`. Exception handling requires fewer steps.

Traditional Error Handling	**Java Exception Handling**
1. `method4` returns an error code to `method3`. 2. `method3` checks for errors and passes the error code to `method2`. 3. `method2` checks for errors and passes the error code to `method1`. 4. `method1` handles the error.	1. `method4` throws an exception that is propagated to `method3`. 2. `method3` receives a return from `method 4` and propagates it to `method2`. 3. `method2` receives a return from `method3` and propagates it to `method1`. 4. `method1` catches and handles the exception.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED



Advantage 3: Exceptions Cannot Be Ignored

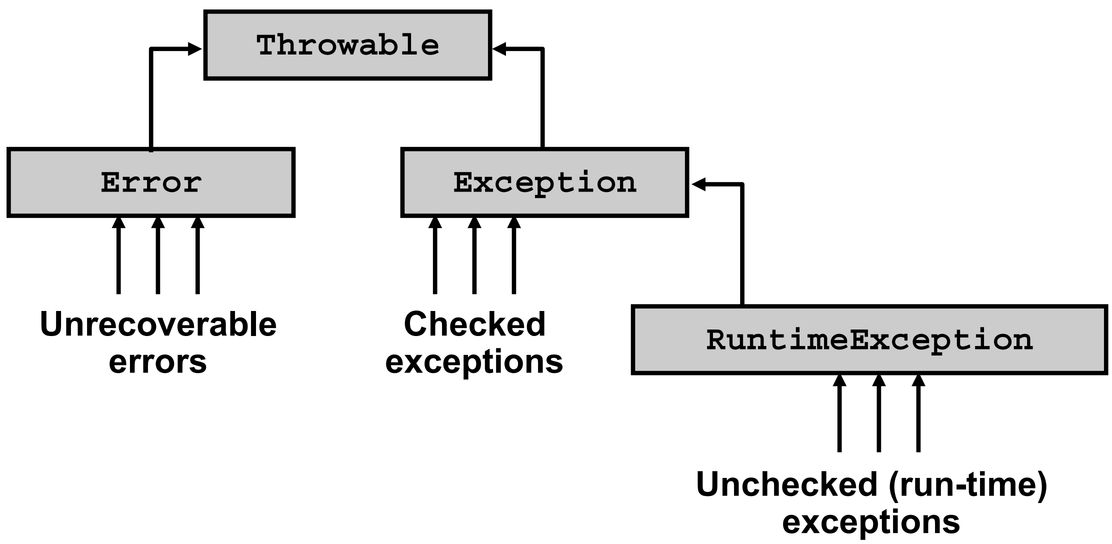
After a method has thrown an exception, the exception cannot be ignored; it must be caught and handled somewhere. In the example in the slide, the programmer writing `method3`, `method2`, or `method1` can choose to ignore the error code that is returned by `method4`, in which case the error code is lost.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Checked Exceptions, Unchecked Exceptions, and Errors

All errors and exceptions extend the `Throwable` class.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Standard Error and Exception Classes

All the standard errors and exceptions in Java extend the `Throwable` class.

Errors

Errors are extensions of the `Error` class. If an error is generated, it normally indicates a problem that will be fatal to the program. Examples of this type of error include running out of memory, being unable to load a class, stack overflow and even hard disk crash. Do not catch `Errors` in your Java code.

Unchecked Exceptions

Unchecked (or run-time) exceptions are extensions of the `RuntimeException` class. All the standard run-time exceptions (for example, dividing by zero or attempting to access an array beyond its last element) are extensions of `RuntimeException`. You can choose what to do with run-time exceptions; you can check for them and handle them, or you can ignore them. If a run-time exception occurs and your code does not handle it, the JVM terminates your program and prints the name of the exception and a stack trace.

Common examples of run-time exceptions are `ArithmeticException`, `IndexOutOfBoundsException`, `NullPointerException`, and `IllegalArgumentException`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Standard Error and Exception Classes (continued)

Checked Exceptions

Checked exceptions are extensions of the `Exception` class. Checked exceptions must be caught and handled somewhere in your application; this rule is enforced by the compiler. Exceptions that you create yourself must extend the `Exception` class. Common examples of checked exceptions are `ClassNotFoundException` and `IOException`.

Note: Run-time exceptions do not need to be caught, but they cannot be ignored. If they are not caught, the program terminates with an error.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Handling Exceptions

Three choices:

- Catch the exception and handle it.
- Allow the exception to pass to the calling method.
- Catch the exception and throw a different exception.

Copyright © 2009, Oracle. All rights reserved.

Handling Exceptions

If you call a method that may throw a checked exception, you have three choices:

- Catch the exception and handle it.
- Allow the exception to pass through your method; another handler somewhere else must handle it.
- Catch the exception and throw a different exception; the new exception must be handled by another handler somewhere else.

Run-Time Exceptions

Your code does not need to handle run-time exceptions; these are handled by the JVM.
The JVM handles run-time exceptions by terminating your program. If you do not want a run-time exception to have this effect, you must handle it.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Catching and Handling Exceptions

- Enclose the method call in a `try` block.
- Handle each exception in a `catch` block.
- Perform any final processing in a `finally` block.

```
try {
  // call the method
}
catch (exception1) {
  // handle exception1
}
catch (exception2) {
  // handle exception2
}…
finally {
  // any final processing
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Catching and Handling Exceptions

If a block of code calls one or more methods that may throw exceptions, enclose the code in a `try` block with one or more `catch` blocks immediately after it. Each `catch` block handles a particular exception.

You can add a `finally` block after all the `catch` blocks. A `finally` block is executed depending on what happens before the block.

How Do You Know Whether a Particular Java Method Throws an Exception?

All the standard Java classes are documented in the JDK documentation; part of the specification for each method is a list of exceptions that the method may throw. Whenever you call a Java method, you must know what exceptions may arise as a consequence. For example, the following are declarations that are taken from the JDK documentation for `java.io.FileInputStream`:

```
public FileInputStream(String name)
     throws FileNotFoundException …
public int read() throws IOException
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Catching and Handling Exceptions (continued)

General Guidelines for `try-catch-block` Structures

- A `try` block must have at least one `catch` block or a `finally` block.
- A `catch` block is required for checked exceptions, unless it is propagated.
- A `try` block can have more than one `catch` block.
- The `finally` block always executes, whether exceptions occur or not.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Catching a Single Exception

```
int qty;
String s = getQtyFromForm();
try {
  // Might throw NumberFormatException
  qty = Integer.parseInt(s);
}
catch ( NumberFormatException e ) {
  // Handle the exception
}
// If no exceptions are thrown, we end up here
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Catching a Single Exception

The example in the slide uses `Integer.parseInt()` to process the value that an end user has entered on a form. `parseInt()` throws a `NumberFormatException` if the string is not an integer value. The catch block can handle this exception by prompting the user to enter the value again.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Catching Multiple Exceptions

```
try {
  // Might throw MalformedURLException
  URL u = new URL(str);
  // Might throw IOException
  URLConnection c = u.openConnection();
}
catch (MalformedURLException e) {
  System.err.println("Could not open URL: " + e);
}
catch (IOException e) {
  System.err.println("Could not connect: " + e);
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Catching Multiple Exceptions

The example in the slide constructs a URL and then connects to it. The example uses two `catch` blocks because there are two possible exceptions that can occur. If an exception occurs in the `try` block, the JVM checks each `catch` handler in sequence until it finds one that deals with that type of exception; the rest of the `try` block is not executed.

A `catch` statement catches the exception that is specified as well as any of its subclasses. For example, the javadoc for `MalformedURLException` shows that it extends `IOException`; thus, you can replace the two `catch` blocks with one:

```
catch (IOException e) {
     System.err.println("Operation failed: " + e);
}
```

You use a single `catch` block if you want your code to behave in the same way for either exception.

Order of `catch` Statements

Note that you get a compiler error if you specify a `catch` handler for a superclass first, followed by a `catch` handler for a subclass. This is because the superclass `catch` handler hides the subclass `catch` handler, which will therefore never see any exceptions. For example, reversing the two `catch` blocks in the example causes a compiler error.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Cleaning Up with a `finally` Block

```
FileInputStream f = null;
try {
  f = new FileInputStream(filePath);
  while (f.read() != -1)
    charcount++;
}
catch(IOException e) {
  System.out.println("Error accessing file " + e);
}
finally {
  // This block is always executed
  f.close();
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Cleaning Up with a `finally` Block

The example in the slide opens a file and counts the characters in the file. The file is then closed, even if the read operation causes an exception. A `finally` block is useful when you want to release system resources, such as open files.

A `finally` block is executed regardless of how the `try` block exits:

- Normal termination, by falling through the end brace
- Because of return or break statement
- Because an exception was thrown

Note: `f.close()` can throw an `IOException` and, therefore, must be enclosed in its own `try…catch` block inside the `finally` block.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Guided Practice: Catching and Handling Exceptions

```
void makeConnection(String url) {
  try {
    URL u = new URL(url);
  }
  catch (MalformedURLException e) {
    System.out.println("Invalid URL: " + url);
    return;
  }
  finally {
    System.out.println("Finally block");
  }
  System.out.println("Exiting makeConnection");
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Guided Practice: Catching and Handling Exceptions

Examine the code that is shown in the slide and describe what it is doing. Note that the `url` argument is a string such as `http://www.oracle.com`.

1. What is printed to standard output if the `URL` constructor executes without throwing an exception?
2. What is printed to standard output if the `URL` constructor throws `MalformedURLException`?

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Guided Practice: Catching and Handling Exceptions

```
void myMethod () {
  try {
    getSomething();
  } catch (IndexOutOfBoundsException e1) {
     System.out.println("Caught IOBException ");
  } catch (Exception e2) {
     System.out.println("Caught Exception ");
  } finally {
     System.out.println("No more exceptions ");
  }
  System.out.println("Goodbye");
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Guided Practice: Catching and Handling Exceptions (continued)

3. What is printed to standard output if `getSomething()` throws `IllegalArgumentException`?
4. Does anything change if the order of the two `catch` blocks is reversed? That is:

```
…
try …
catch (Exception e) {…}
catch (IndexOutOfBoundsException e) {…}
…
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Allowing an Exception to Pass to the Calling Method

- Use `throws` in the method declaration.
- The exception propagates to the calling method.

```
public int myMethod() throws exception1 {
  // code that might throw exception1
}
```

```
public URL changeURL(URL oldURL)
                 throws MalformedURLException {
  return new URL("http://www.oracle.com");
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Allowing an Exception to Pass to the Calling Method

If you cannot meaningfully handle an exception locally, or if you choose not to, it can be passed back to the code that called your method.

In the example in the slide, the URL constructor can throw `MalformedURLException`, but the method does not catch this exception locally. Instead, the exception passes automatically to the method that called `changeURL()`.

If you want an exception to propagate to the calling method, you must declare the exception in your method declaration:

```
public URL changeURL(URL oldURL) throws MalformedURLException
{
  …
}
```

The method that calls `changeURL()` can catch `MalformedURLException`, or it can also let the exception pass through. If the calling method allows the exception to pass through, it must also contain `throws MalformedURLException` in its declaration.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Throwing Exceptions

- Throw exceptions by using the `throw` keyword.
- Use `throws` in the method declaration.

```
throw new Exception1();
```

```
public String getValue(int index) throws
                       IndexOutOfBoundsException {
  if (index < 0 || index >= values.length) {
    throw new IndexOutOfBoundsException();
  }
  …
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Throwing Exceptions

You can throw exceptions in your own code to indicate some abnormal situation. The exceptions that you throw can be standard system exceptions, or you can create your own.

If you decide to throw exceptions, remember that what you are really doing is creating an object and passing it to a higher-level method. Therefore, you must create this exception object by using the `new` operator, as shown in the slide.

A method can throw multiple exceptions, in which case the exception names are separated by commas.

There are four types of exception:

- System
- Application
- Run-time
- Custom

The `java.lang.IndexOutOfBoundsException` is a run-time exception.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating Exceptions

Extend the `Exception` class:

```
public class UserFileException extends Exception {
  public UserFileException (String message) {
     super(message);
  }
}
```

Copyright © 2009, Oracle. All rights reserved.

Creating Exceptions

You can create your own exceptions by extending the `Exception` class. You must not extend the `RuntimeException` class because this is for common exceptions that need not be checked.

The example creates an exception called `UserFileException` with one constructor that just calls the constructor of the superclass.

You can create multiple exceptions for different circumstances in your code. For example, if your code accesses different files, you can throw a different exception for each file. This approach is useful for several reasons:

- You can handle each exception differently.
- If your exception handlers print the exception, this gives you or your users more information about where the exception occurred.
- You can customize your exception. For example, you can add a `UserFileException` constructor that sets an attribute for the line number of the file and a method that prints the line number.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Catching an Exception and Throwing a Different Exception

```
catch (exception1 e) {
  throw new exception2(…);
}
```

```
void readUserFile() throws UserFileException {
  try {
    // code to open and read userfile
  }
  catch(IOException e) {
    throw new UserFileException(e.toString());
  }
}
```

Copyright © 2009, Oracle. All rights reserved.

Catching an Exception and Throwing a Different Exception

The example catches `IOException` and handles it by throwing `UserFileException`. You do this if you want this method to throw a different exception from other methods. The method uses the `throws` keyword in its declaration to indicate that it throws a `UserFileException`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Use Java exceptions for robust error handling
- Handle exceptions by using `try`, `catch`, and `finally`
- Use the `throw` keyword to throw an exception
- Use a method to declare an exception in its signature to pass it up the call stack

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 14 Overview: Throwing and Catching Exceptions

This practice covers the following topics:

- Creating a custom exception
- Changing `DataMan` finder methods to throw exceptions
- Handling the exceptions when calling `DataMan` finder methods
- Testing the changes to the code

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Practice 14 Overview: Throwing and Catching Exceptions

The goal of this practice is to learn how to create your own exception classes, throw an exception object by using your own class, and handle the exceptions.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les14` directory, load the `OrderEntryApplicationLes14` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, load the `OrderEntryApplicationLes14` application. In the Applications – Navigator node, expand `OrderEntryApplicationLes14 – OrderEntryProjectLes14 - Application Sources – oe` and double-click the `UML Class Diagram1` entry. This diagram displays all the classes created up to this point in the course.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

15

Using JDBC to Access the Database

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Describe how Java code connects to the database
- Describe how Java database functionality is supported by the Oracle database
- Load and register a JDBC driver
- Connect to an Oracle database
- Perform a simple `SELECT` statement
- Map simple Oracle database types to Java types
- Use a pooled connection

Copyright © 2009, Oracle. All rights reserved.

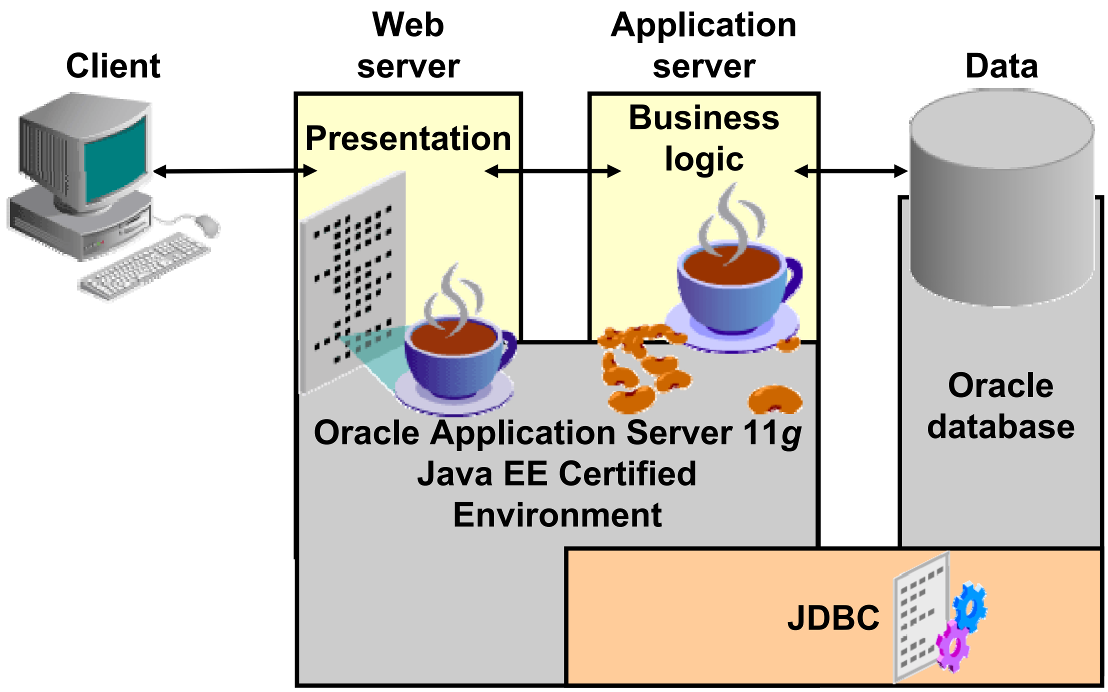
Lesson Objectives

If your business uses an Oracle database to store its data, your Java application must be able to interact with that database. In this lesson, you learn how to use Java Database Connectivity (JDBC) to query a database from a Java class.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java, Java EE, and Oracle 11*g*



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Java, Java EE, and Oracle 11*g*

Oracle provides a complete and integrated platform called Oracle 11*g*, which supports all of the server-side requirements for Java applications. The Oracle 11*g* platform comprises the following:

Oracle Database 11*g*

In addition to its database management features, the Oracle database (currently Oracle Database 11*g*) provides support for a variety of Java-based structures, including Java components and Java stored procedures. These Java structures are executed in the database by its built-in Java Virtual Machine, called the Oracle Java Virtual Machine (Oracle JVM).

Oracle Application Server 11*g*

Oracle Application Server 11*g* maintains and executes all your application logic, including Enterprise JavaBeans, through its own built-in JVM, the Enterprise Java Engine.

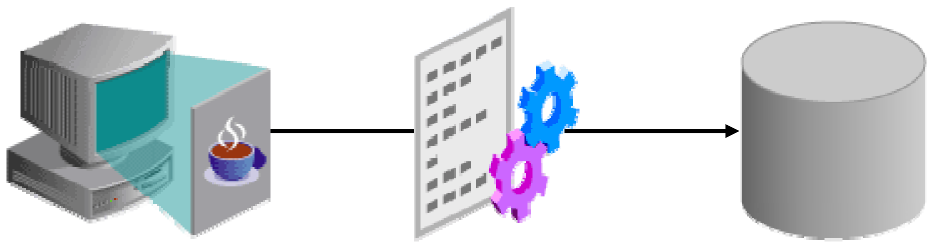
Using Java EE with Oracle 11*g*

Java EE is a standard technology that provides a set of APIs and a run-time infrastructure for hosting and managing applications. It specifies roles and interfaces for applications and the run time onto which applications can be deployed. Application developers can focus only on the application logic and related services, while leveraging the run time for all infrastructure-related services.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Connecting to a Database with Java

Client applications, JSPs, and servlets use JDBC.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Connecting to a Database with Java

To query an Oracle database, a Java application must have a way to connect to the database. This is performed with Java Database Connectivity (JDBC), which is a standard application programming interface (API) that is used for connecting a Java application to relational databases. The networking protocol depends on the JDBC driver you are using. For example, the OCI driver uses Oracle Net; the Thin driver uses TCP/IP.

Running SQL from a Server-Side Application

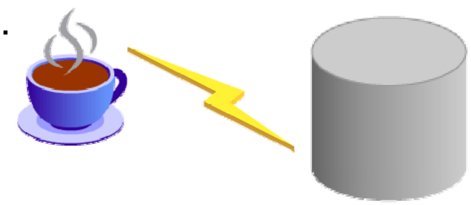
Java procedures inside the database use JDBC to execute their SQL queries. This includes Java stored procedures.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Database Connectivity (JDBC)

- JDBC is a standard API for connecting to relational databases from Java.
  - The JDBC API includes the Core API Package in `java.sql`.
  - JDBC 2.0 API includes the Optional Package API in `javax.sql`.
  - JDBC 3.0 API includes the Core API and Optional Package API.



- You must include the Oracle JDBC driver archive file in the `CLASSPATH`.
- The JDBC class library is part of Java Platform, Standard Edition (Java SE).

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Java Database Connectivity (JDBC)

The `java.sql` package contains a set of interfaces that specify the JDBC API. This package is part of Java 2, Standard Edition. Database vendors implement these interfaces in different ways, but the JDBC API itself is standard.

Using JDBC, you can write code that:

- Connects to one or more data servers
- Executes any SQL statement
- Obtains a result set so that you can navigate through query results
- Obtains metadata from the data server

Each database vendor provides one or more JDBC drivers. A JDBC driver implements the interfaces in the `java.sql` package, providing the code to connect to and query a specific database.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Preparing the Environment

- Import the JDBC packages:

```
// Standard packages
import java.sql.*;
import java.math.*; // optional
// Oracle extension to JDBC packages
import oracle.jdbc.*;
import oracle.sql.*;
```

- Include the JDBC driver classes in the classpath settings.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Requirements for Using Oracle JDBC

Your Java class must import `java.sql.*` to be able to use the JDBC classes, and you must include the JDBC driver classes from your database vendor in the classpath settings.

In JDeveloper, you add the Oracle JDBC library to your project. This adds the necessary `.jar` files to your classpath.

You add a library in the Project Properties dialog box.

JDBC OCI Driver

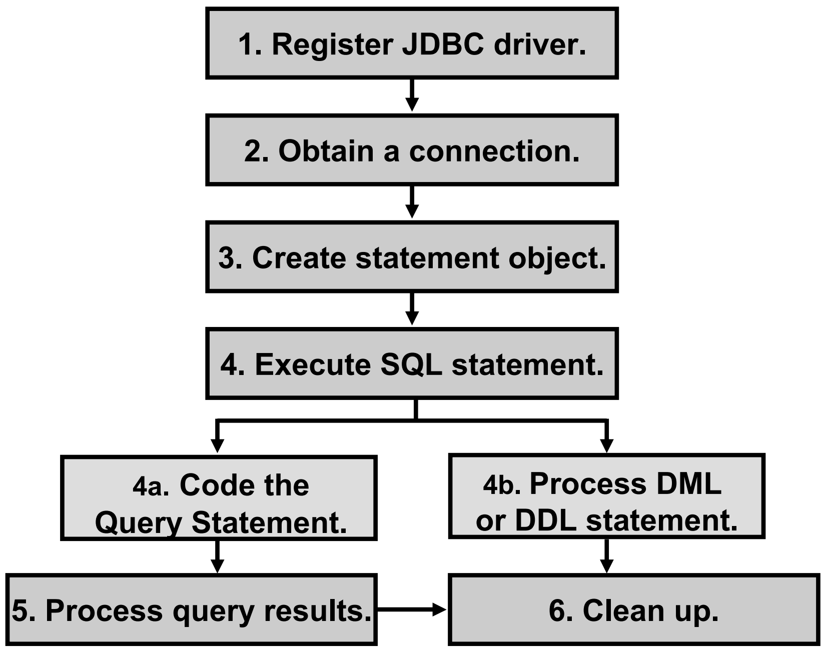
If you are installing the JDBC OCI driver, you must also set the following value for the library path environment variable: `[Oracle Home]/lib`.

Note about "Oracle extension to JDBC packages": *Optional packages* are the new name for what used to be known as standard extensions. An optional package is a group of packages housed in one or more JAR files that implement an API that extends the Java platform. An implementation of an optional package may consist of code written in the Java programming language and, less commonly, platform-specific native code. In this case the optional package contains code specific to the Oracle's implementation of JDBC.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Steps for Using JDBC to Execute SQL Statements



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Steps for Using JDBC to Execute SQL Statements

The following are the key steps:

1. Load and register the driver. (Use the `java.sql.DriverManager` class.)
2. Obtain a connection object. (Use the `getConnection()` method of the `java.sql.DriverManager` class to do this.)
3. Create a statement object. (Use the `Connection` object.)
4. Execute a query, DML, or DDL. (Use the `Statement` object.)
5. If you obtain a `ResultSet` object while executing a query, iterate through the `ResultSet` to process the data for each row that satisfies the query.
6. Close the `ResultSet`, `Statement`, and `Connection` objects when finished.

Dealing with Exceptions

When you use JDBC, all the methods that access the database throw `SQLException` if anything goes wrong. You must put the code in a `try-catch` block to deal with such errors.

`SQLException` has a number of methods that you can call to get information about the exception, including the following:

- `getMessage()` returns a string that describes the error.
- `getErrorCode()` retrieves the vendor-specific exception code.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 1: Register the Driver

- Register the driver in the code:
  - `DriverManager.registerDriver (new oracle.jdbc.OracleDriver());`
  - `Class.forName ("oracle.jdbc.OracleDriver");`
- Register the driver when launching the class:
  - `java -Djdbc.drivers=oracle.jdbc.OracleDriver <ClassName>;`

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Step 1: Registering the Driver

JDBC drivers must register themselves with the driver manager. There are two ways to perform this:

- Use the `registerDriver()` method of `DriverManager`.
- Use the `forName()` method of the `java.lang.Class` class to load the JDBC drivers directly, as follows:

```
try {
Class.forName("oracle.jdbc.OracleDriver");
}
 catch (ClassNotFoundException e) {}
```

Using the `Class.forName()` method calls the static initializer of the driver class. The driver class does not need to be present at compile time. However, this method is valid only for JDK-compliant Java Virtual Machines.

You can register the driver at execution time. In this case, the registering statements that may exist in your Java class are ignored.

Example of using the –`Djdbc` option in the command line:

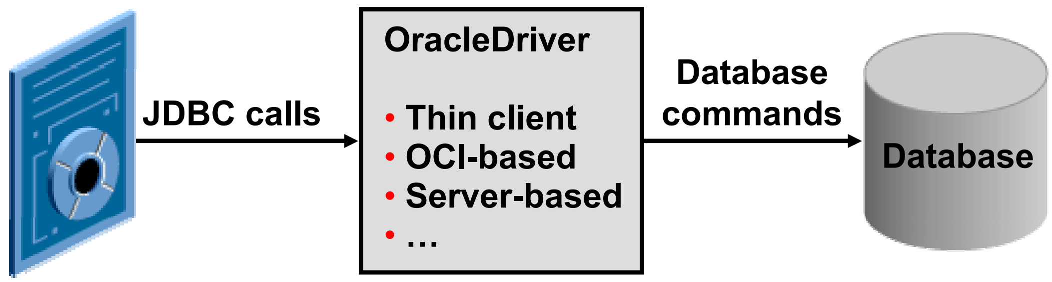
```
C:>java -Djdbc.drivers=oracle.jdbc.OracleDriver MyClass
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Connecting to the Database

Using the package `oracle.jdbc.driver`, Oracle provides different drivers to establish a connection to the database.



Copyright © 2009, Oracle. All rights reserved.

About JDBC Drivers

A JDBC driver implements the interfaces in the `java.sql` package, thereby providing the code to connect to and query a specific database. A JDBC driver can also provide a vendor's own extensions to the standard; Oracle drivers provide extensions to support special Oracle data types.

Oracle provides three drivers:

- Thin-client driver
- OCI-based driver
- Server-based driver

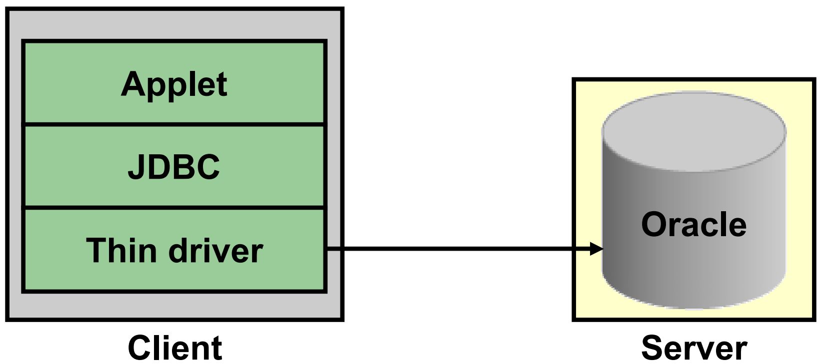
The Oracle JDBC driver is located in the `classes12.jar` file for JDBC 2.0 (and later versions). This archive file contains supporting classes for both the Thin and OCI JDBC drivers.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle JDBC Drivers: Thin-Client Driver

- Is written entirely in Java
- Must be used by applets



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Thin-Client Driver

This driver can connect to an Oracle 11*g* database but also to an Oracle8*i* database or an Oracle9*i* database. To provide maximum portability, you must use this driver if you are developing a client application that can connect to different versions of the Oracle database.

To communicate with the database, the thin-client driver uses a lightweight version of Oracle*Net over TCP/IP that can be downloaded at run time to the client.

The Oracle JDBC Thin driver is a 100% pure Java, Type IV driver. It is targeted to Oracle JDBC applets but can be used for applications as well. Because it is written entirely in Java, this driver is platform independent. It does not require additional Oracle software on the client side. The Thin driver communicates with the server by using Two Task Common (TTC), a protocol developed by Oracle to access the Oracle Relational Database Management System (RDBMS).

The JDBC Thin driver allows a direct connection to the database by providing an implementation of TCP/IP that emulates Oracle Net and TTC (the wire protocol used by OCI) on top of Java sockets. Both of these protocols are lightweight implementation versions of their counterparts on the server. The Oracle Net protocol runs over TCP/IP only.

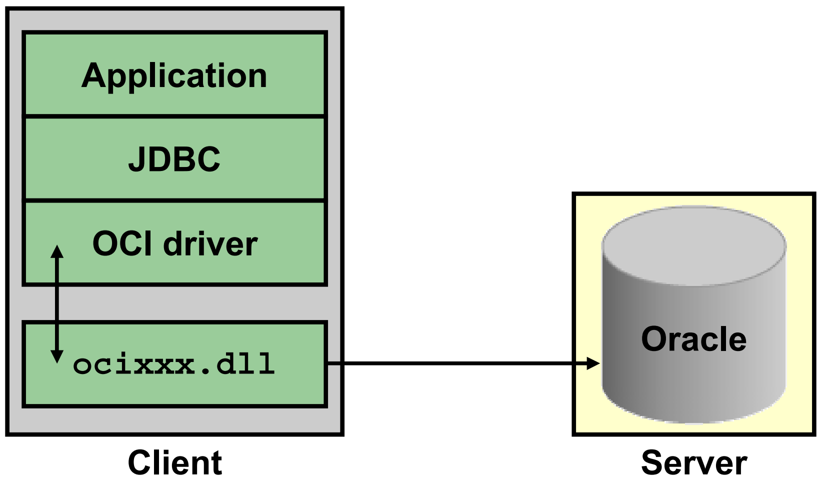
Note: When the JDBC Thin driver is used with an applet, the client browser must have the capability to support Java sockets.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle JDBC Drivers: OCI Client Driver

- Is written in C and Java
- Must be installed on the client



ORACLE
Copyright © 2009, Oracle. All rights reserved.

OCI Client Driver

The JDBC OCI driver:

- Is a Type II driver for use with client/server Java applications
- Requires an Oracle client installation and therefore is specific to the Oracle platform and not suitable for applets
- Provides OCI connection pooling functionality, which can be part of either the JDBC client or a JDBC stored procedure
- Supports Oracle7, Oracle8/8*i*, Oracle9*i*, Oracle 10*g*, and Oracle 11*g* with the highest compatibility. It also supports all installed Oracle Net adapters, including IPC, named pipes, TCP/IP, and IPX/SPX.
- Is written in a combination of Java and C. It converts JDBC invocations to calls to the Oracle Call Interface (OCI) by using native methods to call C-entry points. These calls are then sent over Oracle Net to the Oracle database server. The JDBC OCI driver communicates with the server by using the Oracle-developed TTC protocol.
- Uses OCI libraries, C-entry points, Oracle Net, CORE libraries, and other necessary files on the client machine on which it is installed

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Choosing the Right Driver

<table>
<tr><th>Type of Program</th><th colspan="2">Driver</th></tr>
<tr><td>Applet</td><td colspan="2">Thin</td></tr>
<tr><td>Client application</td><td>Thin</td><td>OCI</td></tr>
<tr><td rowspan="2">EJB, servlet
(on the middle tier)</td><td colspan="2">Thin</td></tr>
<tr><td colspan="2">OCI</td></tr>
<tr><td>Stored procedure</td><td colspan="2">Server side</td></tr>
</table>

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Choosing the Appropriate Driver

Consider the following when choosing a JDBC driver to use for your application or applet:

- If you are writing an applet, you must use the JDBC Thin driver. JDBC OCI-based driver classes do not work inside a Web browser because they call native (C-language) methods.
- If you want maximum portability and performance under the Oracle 10*g* platform (and earlier), use the JDBC Thin driver. You can connect to an Oracle server from either an application or an applet by using the JDBC Thin driver.
- If you are writing a client application for an Oracle client environment and need maximum performance, choose the JDBC OCI driver.
- If performance is critical to your application, if you want maximum scalability of the Oracle server, or if you need enhanced availability features such as Transparent Application Failover (TAF) or the enhanced proxy features such as middle-tier authentication, choose the JDBC OCI driver.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 2: Obtain a Database Connection

- In JDBC 1.0, use the `DriverManager` class, which provides overloaded `getConnection()` methods.
  - All connection methods require a JDBC URL to specify the connection details.
- Example:

```
Connection conn =
DriverManager.getConnection(
   "jdbc:oracle:thin:@myhost:1521:ORCL",
      "scott","tiger");
```

- Vendors can provide different types of JDBC drivers.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Obtaining a Database Connection

Use the `DriverManager` class to create a connection by calling the `getConnection()` method.

The `getConnection()` method is overloaded, as shown by the following example:

- `getConnection(String url)`
- `getConnection(String url, Properties props)`, where properties must include at least a value for the following key names: `user` and `password`
- `getConnection(String url, String user, String password)`

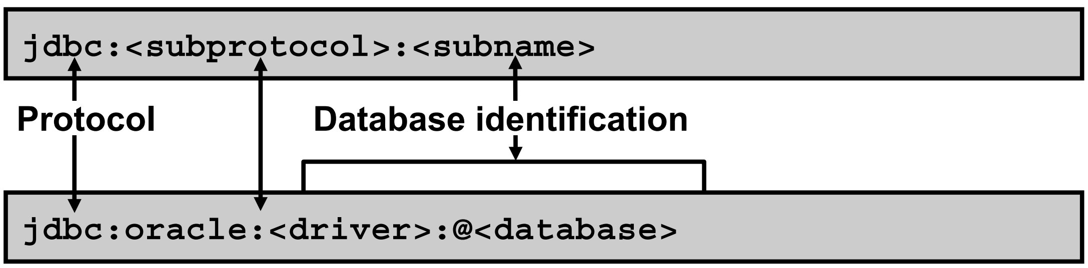
In each case, you must supply a URL-like string (identifying the registered JDBC driver to use) and the database connection string and security credentials, if required.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

JDBC URLs

- JDBC uses a URL-like string; the URL identifies:
  - The JDBC driver to use for the connection
  - Database connection details, which vary depending on the driver used



- Example using the Oracle JDBC Thin driver:
  - jdbc:oracle:thin:@myhost:1521:ORCL

ORACLE
Copyright © 2009, Oracle. All rights reserved.

JDBC URLs

JDBC uses a URL to identify the database connection. A JDBC URL looks different from an HTTP or FTP URL. But, like any URL, it is a locator for a particular resource (in this case, a database). The structure of a JDBC URL is flexible, enabling the driver writer to specify what to include in the URL. End users need to learn what structure their vendor uses.

The slide shows the general syntax for a JDBC URL and the syntax that Oracle uses for connecting with an Oracle driver. The general syntax of a JDBC URL is as follows:

jdbc:<*subprotocol*>:<*subname*>

- jdbc is the protocol. All URLs start with their protocol.
- <*subprotocol*> is the name of a driver or database connectivity mechanism. Driver developers register their subprotocols with JavaSoft to make sure that no one else uses the same subprotocol name. For all Oracle JDBC drivers, the subprotocol is oracle.
- <*subname*> identifies the database. The structure and contents of this string are determined by the driver developer. For Oracle JDBC drivers, the subname is <*driver*>:@<*database*>, where:
  - <*driver*> is the driver
  - <*database*> provides database connectivity information

The following slides describe the syntax of an Oracle JDBC URL for the different JDBC drivers for client-side Java application code.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

JDBC URLs with Oracle Drivers

- Oracle JDBC Thin driver

```
Syntax:   jdbc:oracle:thin:@<host>:<port>:<SID>
Example: "jdbc:oracle:thin:@eduhost:1521:orcl"
```

- Oracle JDBC OCI driver

```
Syntax:   jdbc:oracle:oci:@<tnsname entry>
Example: "jdbc:oracle:oci:@orcl"
```

Copyright © 2009, Oracle. All rights reserved.

JDBC URLs with Oracle Drivers

The basic structure of the JDBC URL for connecting to a database by using one of the Oracle JDBC drivers is `jdbc:`*`<subprotocol>`*`:`*`<driver>`*`:`*`<database>`*.

Oracle JDBC Thin driver

`<driver>` is thin.

`<database>` is a string of the form *`<host>:<port>:<sid>`*. That is, it is the host name, TCP/IP port, and Oracle SID of the database to which you want to connect.

Example: *`jdbc:oracle:thin:@eduhost:1521:ORCL`*

Oracle JDBC OCI driver

`<driver>` is oci, oci8, or oci7, depending on which OCI driver you are using.

`<database>` is a `TNSNAMES` entry from the `tnsnames.ora` file.

Example: *`jdbc:oracle:oci:@eduhost`*

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 3: Create a Statement

JDBC statement objects are created from the `Connection` instance.

- Use the `createStatement()` method, which provides a context for executing a SQL statement.
- Example:

```
Connection conn =
DriverManager.getConnection(
       "jdbc:oracle:thin:@myhost:1521:ORCL",
       "scott","tiger");
Statement stmt = conn.createStatement();
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating a Statement

The `execute()` method is useful for dynamically executing an unknown SQL string.

JDBC provides two other statement objects:

- `PreparedStatement`, for precompiled SQL statements
- `CallableStatement`, for statements that execute stored procedures

Objects and Interfaces

`java.sql.Statement` is an interface rather than a class. When you declare a `Statement` object and initialize it with the `createStatement()` method, you are creating the implementation of the `Statement` interface supplied by the Oracle driver that you are using.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using the `Statement` Interface

The `Statement` interface provides three methods to execute SQL statements:

- `executeQuery(String sql)` for `SELECT` statements
  - Returns a `ResultSet` object for processing rows
- `executeUpdate(String sql)` for DML or DDL.
  - Returns an `int`
- `execute(String)` for any SQL statement
  - Returns a `boolean` value

Copyright © 2009, Oracle. All rights reserved.

Using the `Statement` Interface

Use `executeQuery(String sql)` for `SELECT` statements.

- Returns a `ResultSet` object for processing rows

Use `executeUpdate(String sql)` for DML or DDL.

- Returns an `int` value indicating the number of rows affected by the DML; otherwise, returns `0` for DDL

Use `execute(String)` for any SQL statement.

- Returns a `boolean` value of `true` if the statement returns a `ResultSet` (such as a query); otherwise, returns a value of `false`

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 4a: Code the Query Statement

Provide a SQL query string, without semicolon, as an argument to the `executeQuery()` method.

- Returns a `ResultSet` object:

```
Statement stmt = null;
ResultSet rset = null;
  stmt = conn.createStatement();
  rset = stmt.executeQuery
      ("SELECT ename FROM emp");
```

Copyright © 2009, Oracle. All rights reserved.

Executing a Query

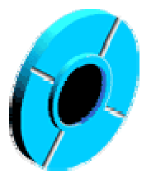
To query the database, use the `executeQuery()` method of your `Statement` object. This method takes a SQL statement as input and returns a JDBC `ResultSet` object.
The statement follows standard SQL syntax.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`ResultSet` Object

- The JDBC driver returns the results of a query in a `ResultSet` object.
- `ResultSet`:
  - Maintains a cursor pointing to its current row of data
  - Provides methods to retrieve column values



ORACLE
Copyright © 2009, Oracle. All rights reserved.

`ResultSet` Object

A `ResultSet` object is a table of data representing a database result set, which is generated by executing a statement that queries the database.

A `ResultSet` object maintains a cursor pointing to its current row of data. Initially, the cursor is positioned before the first row.

A default `ResultSet` object is not updatable and has a cursor that moves forward only. Thus, it is possible to iterate through the object only once, and only from the first row to the last row. With the release of the JDBC 2.0 API, it is now possible to produce `ResultSet` objects that are scrollable and updatable.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 4b: Submit DML Statements

1. Create an empty statement object.

```
Statement stmt = conn.createStatement();
```

2. Use `executeUpdate` to execute the statement.

```
int count = stmt.executeUpdate(SQLDMLstatement);
```

Example:

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("DELETE FROM order_items
      WHERE order_id = 2354");
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Submitting DML Statements

The slide shows the syntax for the methods that execute a database update by using a DML statement. Whereas `executeQuery` returns a `ResultSet` object containing the results of the query sent to the DBMS, the return value for `executeUpdate` is an `int` that indicates how many rows of a table were updated.

When the `executeUpdate` method is used to execute a DDL statement, such as in creating a table, it returns the `int 0`.

A return value of `0` for `executeUpdate` can mean one of the following: the statement executed was an update statement that affected zero rows, or the statement executed was a DDL statement.

Example: By using the `executeUpdate()` method, the `PICTURES` table is populated with the `region_id` from the `regions` table:

```
System.out.println("Table Insert");
stmt.executeUpdate ("INSERT INTO pictures (id)
SELECT region_id FROM regions");
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 4b: Submit DDL Statements

1. Create an empty statement object.

```
Statement stmt = conn.createStatement();
```

2. Use `executeUpdate` to execute the statement.

```
int count = stmt.executeUpdate(SQLDDLstatement);
```

Example:

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
  ("CREATE TABLE temp (col1 NUMBER(5,2),
      col2 VARCHAR2(30)");
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Submitting DDL Statements

The slide shows the syntax for the methods that execute a DDL statement.

`executeUpdate()` returns an `int` containing `0` for a statement with no return value, such as a SQL DDL statement.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 5: Process the Query Results

The `executeQuery()` method returns a `ResultSet`.

- Use the `next()` method in loop to iterate through rows.
- Use `getXXX()` methods to obtain column values by column name or by column position in query.

```
stmt = conn.createStatement();
rset = stmt.executeQuery(
         "SELECT ename FROM emp");
while (rset.next()) {
    System.out.println
(rset.getString("ename"));
}
```

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Processing the Query Results: `getXXX()` Methods

The `ResultSet` class has several methods that retrieve column values for the current row. Each of these `getXXX()` methods attempts to convert the column value to the specified Java type and return a suitable Java value. For example, `getInt()` gets the column value as an integer, `getString()` gets the column value as a string, and `getDate()` returns the column value as a date.

- The `next()` method returns `true` if a row was found; otherwise, it returns `false`. Use it to check whether a row is available and to step through subsequent rows.
- There are many `getXXX()` methods to get the column values, where *XXX* is a Java data type. For example, `getString(pos)` returns a column in `pos` as a `String`, and `getInt(pos)` returns a column in `pos` as an `int`.
- There is a potential problem of database null values when using `getXXX` methods (for example, `getInt`) because Java primitives do not support null values. You should typically use `java.sql.ResultSet.wasNull()` to determine if the database value is NULL.
- `getXXX(x)` is overloaded. `x` can be an `int` (position in `Select`), or `String` (name of column or expression returned). In the case of `String`, the value is not case-sensitive.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Mapping Database Types to Java Types

`ResultSet` maps database types to Java types:

```
ResultSet rset = stmt.executeQuery
  ("SELECT empno, hiredate, job
  FROM emp");
while (rset.next()){
int id = rset.getInt(1);
Date hiredate = rset.getDate(2);
String job = rset.getString(3);
```

Column Name	Type	Method
`empno`	`NUMBER`	`getInt()`
`hiredate`	`DATE`	`getDate()`
`job`	`VARCHAR2`	`getString()`

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Mapping Database Types to Java Types

In many cases, you can get all the columns in your result set by using the `getObject()` or `getString()` methods of `ResultSet`. For performance reasons, or because you want to perform complex calculations, it is sometimes important to have your data in a type that exactly matches the database column.

The JDBC section of the Java tutorial contains a matrix that maps `ResultSet.get`*`XXX`* methods to ANSI SQL types. For each SQL type, the matrix shows:

- Which `get`*`XXX`* methods can be used to retrieve the SQL type
- Which `get`*`XXX`* method is recommended to retrieve the SQL type

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Mapping Database Types to Java Types (continued)

Table of ANSI SQL Types and Java Types

The following table lists the ANSI SQL types, the corresponding data type to use in Java, and the name of the method to call in `ResultSet` to obtain that type of column value:

ANSI SQL Type	Java Type	`ResultSet Method`
CHAR, VARCHAR	`java.lang.String`	`getString()`
LONGVARCHAR	`java.io.InputStream`	`getAsciiStream()`
NUMERIC, DECIMAL	`java.math.BigDecimal`	`getBigDecimal()`
BIT	`boolean`	`getBoolean()`
TINYINT	`byte`	`getByte()`
SMALLINT	`short`	`getShort()`
INTEGER	`int`	`getInt()`
BIGINT	`long`	`getLong()`
REAL	`float`	`getFloat()`
DOUBLE, FLOAT	`double`	`getDouble()`
BINARY, VARBINARY	`byte[]`	`getBytes()`
LONGBINARY	`java.io.InputStream`	`getBinaryStream()`
DATE	`java.sql.Date`	`getDate()`
TIME	`java.sql.Time`	`getTime()`
TIMESTAMP	`Java.sql.Timestamp`	`getTimestamp()`

Table of Oracle SQL Types

ORACLE SQL Type	Oracle Type	Type Extension
NUMBER	`Oracle.Types.NUMBER`	`oracle.sql.NUMBER`
CHAR	`Oracle.Types.CHAR`	`oracle.sql.CHAR`
RAW	`Oracle.Types.RAW`	`oracle.sql.RAW`
DATE	`Oracle.Types.DATE`	`oracle.sql.DATE`
ROWID	`Oracle.Types.ROWID`	`oracle.sql.ROWID`
BLOB	`Oracle.Types.BLOB`	`oracle.sql.BLOB`
CLOB	`Oracle.Types.CLOB`	`oracle.sql.CLOB`
BFILE	`n/a`	`oracle.sql.BFILE`

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 6: Clean Up

Explicitly close a `Connection`, `Statement`, and `ResultSet` object to release resources that are no longer needed.

- Call their respective `close()` methods:

```
Connection conn = ...;
Statement stmt = ...;
ResultSet rset = stmt.executeQuery(
          "SELECT ename FROM emp");
...
// clean up
rset.close();
stmt.close();
conn.close();
...
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Closing the `ResultSet`, `Statement`, and `Connection` Objects

You must explicitly close all `ResultSet` and `Statement` objects after you finish using them. The `close()` methods clean up memory and release database cursors. So if you do not explicitly close your `ResultSet` and `Statement` objects, serious memory leaks may occur, and you may run out of cursors in the database. You then need to close the connection.

The server-side driver runs within a default session. You are already connected, and you cannot close the default connection made by the driver. Calling `close()` on the connection does nothing.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Basic Query Example

```
import java.sql.*;
import oracle.jdbc.driver.OracleDriver;
  class TestJdbc {
    public static void main (String args [ ]) throws
                                  SQLException {
      DriverManager.registerDriver (new
                            oracle.jdbc.OracleDriver());
      Connection conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@myHost:1521:ORCL","scott",
                                         "tiger");
      Statement stmt = conn.createStatement();
      ResultSet rset = stmt.executeQuery
                    ("SELECT ename FROM emp");
      while (rset.next())
        System.out.println (rset.getString ("ename"));
      rset.close();
      stmt.close();
      conn.close();
      }
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Handling an Unknown SQL Statement

1. Create an empty statement object.

```
Statement stmt = conn.createStatement();
```

2. Use `execute` to execute the statement.

```
boolean isQuery = stmt.execute(SQLstatement);
```

3. Process the statement accordingly.

```
if (isQuery) { // was a query - process results
  ResultSet r = stmt.getResultSet(); ...
}
else { // was an update or DDL - process result
 int count = stmt.getUpdateCount(); ...
}
```

Copyright © 2009, Oracle. All rights reserved.

Handling an Unknown SQL Statement

An application may not know whether a given statement returns a result set until the statement has been executed. In addition, some stored procedures may return several different result sets and update counts.

JDBC provides a mechanism so that an application can execute a statement and then process an arbitrary collection of result sets and update counts. The mechanism is based on the use of a general `execute()` method and calls to three other methods: `getResultSet`, `getUpdateCount`, and `getMoreResults`. These methods enable an application to explore the statement results one at a time and determine whether a given result is a result set or an update count.

`execute()` returns `true` if the result of the statement is a result set; it returns `false` if the result of the statement is an update count. You can then call either `getResultSet()` or `getUpdateCount()` on the statement.

The following example uses `execute()` to dynamically execute an unknown statement:

```
public void executeStmt (String statement) throws SQLException {
 Statement stmt = conn.createStatement(); // Execute the statement
 boolean isQuery = stmt.execute(statement);
   if (isQuery ) {    <statement was a query; process the results>
 ...   }
   else {          <statement was an update or DDL>
    int updateCount = stmt.getUpdateCount(); // Process the results
    ... }
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

Handling Exceptions

- SQL statements can throw a `java.sql.SQLException`.
- Use standard Java error handling methods.

```
 try  {
     rset = stmt.executeQuery("SELECT empno,
       ename FROM emp");
       }
       catch (java.sql.SQLException e)
       { ... /* handle SQL errors */ }
...
finally {  // clean up
 try { if (rset != null) rset.close(); }
       catch (Exception e)
        { ... /* handle closing errors */ }
...
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Handling Exceptions

You can use the `try-catch-finally` block structure for closing resources.

Code Example

```
Connection conn = null; Statement stmt = null;
ResultSet rset = null; // initialize
stmt = conn.createStatement();
try {
 rset = stmt.executeQuery("SELECT empno, ename FROM emp");
    }
catch (java.sql.SQLException e)
    { ... /* handle errors */ }
...
// Clean up resources
finally {
 try { if (rset != null) rset.close(); } catch (Exception e) {}
 try { if (stmt != null) stmt.close(); } catch (Exception e) {}
 try { if (conn != null) conn.close(); } catch (Exception e) {}
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Transactions with JDBC

- By default, connections are in autocommit mode.
- Use `conn.setAutoCommit(false)` to disable autocommit.
- To control transactions when you are not in autocommit mode, use:
  - `conn.commit()` to commit a transaction
  - `conn.rollback()` to roll back a transaction
- Closing a connection commits the transaction even with autocommit disabled.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Transactions with JDBC

After you perform an `UPDATE` or `INSERT` operation in a result set, you propagate the changes to the database in a separate step that you can skip if you want to cancel the changes.

With JDBC, database transactions are managed by the `Connection` object. When you create a `Connection` object, it is in autocommit mode (which means that each statement is committed after it is executed).

You can change the connection's autocommit mode at any time by calling `setAutoCommit()`. The following is a full description of autocommit mode:

- If a connection is in autocommit mode, all its SQL statements are executed and committed as individual transactions.
- If a statement returns a result set, the statement finishes when the last row of the result set has been retrieved or when the result set has been closed.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Transactions with JDBC (continued)

- If autocommit mode has been disabled, its SQL statements are grouped into transactions, which must be terminated by calling either `commit()` or `rollback()`. The `commit()` method makes permanent all changes because the previous commit or rollback releases any database locks held by the connection.
- `rollback()` drops all changes because the previous commit or rollback releases any database locks. `commit()` and `rollback()` must be called only when you are not in autocommit mode.

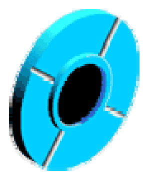
Note: The server-side driver does not support autocommit mode. You must control transactions explicitly.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`PreparedStatement` Object

- A prepared statement prevents reparsing of SQL statements.
- Use the `PreparedStatement` object for statements that you want to execute more than once.
- A prepared statement can contain variables that you supply each time you execute the statement.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

`PreparedStatement` Object

`PreparedStatement` is inherited from `Statement`; the difference is that `PreparedStatement` holds precompiled SQL statements.

If you execute a `Statement` object many times, its SQL statement is compiled each time. `PreparedStatement` is more efficient because its SQL statement is compiled only once, when you first prepare `PreparedStatement`. After that, the SQL statement does not have to be recompiled every time you execute it in `PreparedStatement`.

Therefore, if you need to execute the same SQL statement several times in an application, it is more efficient to use `PreparedStatement` than `Statement`.

`PreparedStatement` Parameters

`PreparedStatement` does not have to execute exactly the same query each time. You can specify parameters in the `PreparedStatement` SQL string and supply the actual values for these parameters when the statement is executed.

The next slide shows how to supply parameters and execute a prepared statement.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating a `PreparedStatement` Object

1. Register the driver and create the database connection.
2. Create the `PreparedStatement` object, identifying variables with a question mark (`?`).

```
PreparedStatement pstmt =
  conn.prepareStatement
("UPDATE emp SET ename = ? WHERE empno = ?");
```

```
PreparedStatement pstmt =
  conn.prepareStatement
("SELECT ename FROM emp WHERE empno = ?");
```

Copyright © 2009, Oracle. All rights reserved.

Creating a `PreparedStatement` Object

To write changes to the database, such as for `INSERT` or `UPDATE` operations, you typically create a `PreparedStatement` object. You can use the `PreparedStatement` object to execute a statement with varying sets of input parameters. The `prepareStatement()` method of your JDBC `Connection` object enables you to define a statement that takes bind variable parameters and returns a JDBC `PreparedStatement` object with your statement definition.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Executing a `PreparedStatement` Object

1. Supply values for the variables.

```
pstmt.setXXX(index, value);
```

2. Execute the statement.

```
pstmt.executeQuery();
pstmt.executeUpdate();
```

```
int empNo = 3521;
PreparedStatement pstmt =
  conn.prepareStatement("UPDATE emp
  SET ename = ? WHERE empno = ? ");
pstmt.setString(1, "DURAND");
pstmt.setInt(2, empNo);
pstmt.executeUpdate();
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Specifying Values for the Bind Variables

You use the `PreparedStatement.setXXX()` methods to supply values for the variables in a prepared statement. There is one `setXXX()` method for each Java type: `setString()`, `setInt()`, and so on.

You must use the `setXXX()` method that is compatible with the SQL type of the variable. In the example in the slide, the first variable is updating a `VARCHAR` column, and so you must use `setString()` to supply a value for the variable. You can use `setObject()` with any variable type.

Each variable has an index. The index of the first variable in the prepared statement is 1, the index of the second variable is 2, and so on. If there is only one variable, its index is 1. The index of a variable is passed to the `setXXX()` method.

Closing a `PreparedStatement`

A `PreparedStatement` object is not cached. If you close it, you must start again.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

What Is a `DataSource`?

A `DataSource` object:

- Is the representation of a data source—a facility for storing data—in the Java programming language.
- Can reside on a remote server or on a local desktop machine.
- Can be thought of as a factory for connections to the particular data source that the `DataSource` instance represents.
- Can optionally be bound to Java Naming and Directory (JNDI) entities so that you can access databases by logical names for convenience and portability.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

What Is a `DataSource`?

`DataSources` are Java objects that represent physical data storage systems such as relational databases. It is via `javax.sql.DataSource` objects that an application can retrieve underlying connections to the databases being represented by the `DataSource` object.

The `DataSource` interface provides an alternative to using the `DriverManager` class for establishing a connection with a data source. (You saw an example of using `DriverManager` in step 2 earlier in this lesson). The `DataSource` mechanism is now the preferred way to make a connection because it offers a more standard and versatile alternative to the `DriverManager` connection functionality. You can use both facilities in the same application but ultimately it is recommended that you transition to using `DataSources`. Eventually Sun will probably deprecate `DriverManager` and its associated classes and functionality.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Advantages of Using a `DataSource`

There are a number of advantages to using a `DataSource` object for establishing a connection to the database:

- Applications do not need to hard code a driver class.
- Changes can be made to a data source's properties without changing application code.
- Connection pooling and distributed transactions are available through a `DataSource` object that is implemented to work with the middle-tier infrastructure.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Advantages of Using a `DataSource`

Using `DriverManager` to connect to a data source reduces portability, in that the application must identify a specific JDBC driver class name and driver URL (as you saw earlier in this lesson). The driver class name and driver URL are specific to a JDBC vendor, driver implementation, and data source. Therefore, if something about the data source or driver changes, the application code has to be amended. If your applications need to be portable among data sources, the `DataSource` interface offers distinct advantages. A `DataSource` object works with a Java Naming and Directory Interface (JNDI) naming service and is created, deployed, and managed separately from the applications that use it. Being registered with a JNDI naming service gives a `DataSource` object two major advantages over `DriverManager`. Instead of hardcoding driver information, as with the `DriverManager`, a programmer can choose a logical name for the data source and register the logical name with a JNDI naming service. The application uses the logical name, and the JNDI naming service supplies the `DataSource` object associated with the logical name. The `DataSource` object can then be used to create a connection to the data source it represents.

The second major advantage is that the `DataSource` facility allows developers to implement a `DataSource` class to take advantage of features like connection pooling and distributed transactions. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested. The ability to use distributed transactions enables an application to do the heavy-duty database work of large enterprises.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using an `OracleDataSource` to Connect to a Database

There are three steps that must be performed to use an `OracleDataSource`:

1. Create an `OracleDataSource` object of the `oracle.jdbc.pool.OracleDataSource` class.
2. Set the `OracleDataSource` object attributes using the `set` methods defined in the class.
3. Connect to the database via the `OracleDataSource` object using the `getConnection()` method.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using an OracleDataSource to Connect to a Database

Driver vendors provide `DataSource` implementations. A particular `DataSource` object represents a particular physical data source, and each connection to that `DataSource` object creates a connection to that physical data source. Oracle provides the `OracleDataSource` implementation.

The three steps involved in connecting to a database using an `OracleDataSource`:

1. Create an object of the `oracle.jdbc.pool.OracleDataSource` class:
   `OracleDataSource myDataSource = new OracleDataSource();`
2. Before you can use your `OracleDataSource` object, you must set a number of attributes to specify the connection details, using the various `set` methods in the class. These details include items like the database name and the JDBC driver to use. The `oracle.jdbc.pool.OracleDataSource` class actually implements the `javax.sql.DataSource` interface, which defines a set of attributes.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using an `OracleDataSource` to Connect to a Database (continued)

```
myDataSource.setServerName("localhost");
myDataSource.setDatabaseName("ORCL");
myDataSource.setDriverType("thin");
myDataSource.setNetworkProtocol("tcp");
myDataSource.setPortNumber(1521);
myDataSource.setUser("scott");
myDataSource.setPassword("tiger");
```

The next few lines illustrate the use of some of the `get` methods used to read the attributes previously set:

```
String serverName = myDataSource.getServerName();
String databaseName = myDataSource.getDatabaseName();
String driverType = myDataSource.getDriverType();
```

3. The third step is to connect to the database using the `OracleDataSource` object. You do this by calling the `getConnection()` method:

```
Connection myConnection = myDataSource.getConnection();
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Maximizing Database Access with Connection Pooling

- Use connection pooling to minimize the operation costs of creating and closing sessions.
- Use explicit data source declaration for physical reference to the database.
- Use the `getConnection()` method to obtain a logical connection instance.

Copyright © 2009, Oracle. All rights reserved.

Maximizing Database Access with Connection Pooling

A connection pool is a cache of database connections. It is maintained in memory, which enables the connections to be reused. This technique is important for increasing performance, especially when the JDBC API is used in a middle-tier environment.

Connection pooling does not affect application code. The application simply accesses a JDBC data source and uses it in the standard way. The data source implements connection pooling transparently to the application by using the `PooledConnection` and `ConnectionPoolDataSource` facilities provided by the JDBC 2.0 driver.

- `javax.sql.ConnectionPoolDataSource`: A `DataSource` implementation that provides pooling capabilities, uses a class that implements the `ConnectionPoolDataSource` interface. A `ConnectionPoolDataSource` is a factory for `PooledConnection` objects.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

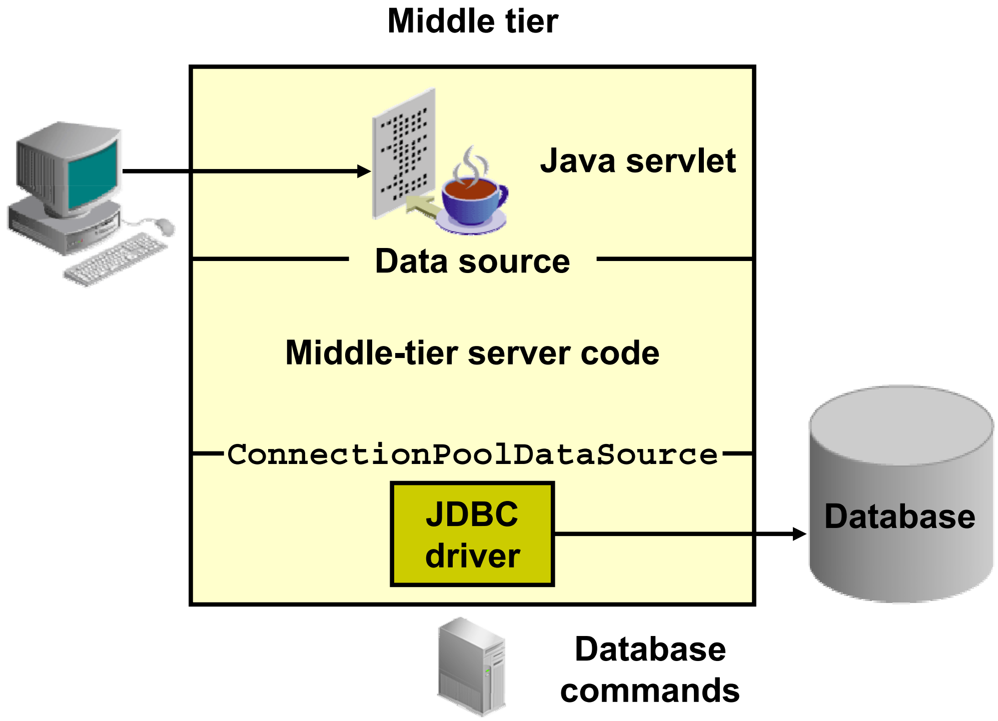
Maximizing Database Access with Connection Pooling (continued)

- `javax.sql.PooledConnection`: The objects that a `DataSource` with pooling capabilities keeps in its pool implement the `PooledConnection` interface. When the application asks the `DataSource` for a connection, it locates an available `PooledConnection` object, or, if the pool is empty, gets a new one from its `ConnectionPoolDataSource`. The `PooledConnection` provides a `getConnection()` method that returns a `Connection` object. The `DataSource` calls this method and returns the `Connection` to the application. This `Connection` object behaves like a regular connection with one exception—when the application calls the `close()` method, instead of closing the connection to the database, it informs the `PooledConnection` that it belongs to that it is no longer used. The `PooledConnection` relays this information to the `DataSource`, which returns the `PooledConnection` to the pool.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Connection Pooling



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Connection Pooling

When using pooled connections, you must use a `DataSource` object rather than the `DriverManager` class to get a connection. The `DataSource` object is implemented and deployed so that it creates pooled connections.

Note: Connection pooling is supported in Thin and OCI drivers in both JDK1.1 and JDK 1.2. Connection pooling is not supported for the server driver because the server driver can have only one connection, which is to the logged-in session in which it is running.

Simple Pooled Connection Example

```
import java.sql.*;
import javax.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.*;

class PooledConnection1
{
  public static void main (String args [])
       throws SQLException     {
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Connection Pooling (continued)

Simple Pooled Connection Example (continued)

```
// Create an OracleConnectionPoolDataSource instance
   OracleConnectionPoolDataSource ocpds =
                                new
   OracleConnectionPoolDataSource();
   String url = "jdbc:oracle:thin:@myhost";
   try {
     String url1 = System.getProperty("JDBC_URL");
     if (url1 != null)
       url = url1;
   } catch (Exception e) {
// If there is any security exception,
// ignore it and use the default
   }
// Set connection parameters
   ocpds.setURL(url);
   ocpds.setUser("scott");
   ocpds.setPassword("tiger");
// Create a pooled connection
   PooledConnection pc  = ocpds.getPooledConnection();
// Get a logical connection
   Connection conn = pc.getConnection();
// Create a Statement
   Statement stmt = conn.createStatement ();
// Select the ENAME column from the EMP table
   ResultSet rset = stmt.executeQuery ("select ENAME from
        EMP");

// Iterate through the result set and print the employee  names
   while (rset.next ())
     System.out.println (rset.getString (1));
// Close the ResultSet
   rset.close();
   rset = null;
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Connection Pooling (continued)

Simple Pooled Connection Example (continued)

```
// Close the Statement and logical connection
     stmt.close();
     stmt = null;
     conn.close();
     conn = null;
// Get another logical connection using the same pooled
    connection
      conn = pc.getConnection();
// Create another statement and run another query against the
    database
    …
    …
// Close the ResultSet
// Close the Statement and logical connection
// Finally close the pooled connection
     pc.close();
     pc = null;
  }
}
```

Note: A pooled connection instance will typically be asked to produce a series of connection instances during its existence, but only one of these connection instances can be open at any one time. Each time a pooled connection instance `getConnection()` method is called, it returns a new connection instance and it closes any previous connection instance that still exists and has been returned by the same pooled connection instance. However you should explicitly close any previous connection instance before opening a new one. Calling the `close()` method of a pooled connection instance closes the physical connection to the database.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Load and register a JDBC driver
- Use the driver to connect to an Oracle database
- Perform a simple `SELECT` statement
- Process the results of the query by iterating through the rows of a result set
- Create and use a `PreparedStatement` object
- Use an `OracleDataSource` to connect to a database
- Use a pooled connection

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 15 Overview: Using JDBC to Access the Database

This practice covers the following topics:

- Setting up the Java environment for JDBC
- Adding JDBC components to query the database

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Practice 15 Overview: Using JDBC to Access the Database

The goal of this practice is to use the course application to interact with the Oracle database. During this practice, you establish a connection to the database, perform query statements to access the database, and retrieve information to integrate into the application.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les15` directory, load the `OrderEntryApplicationLes15` application, and continue with this practice.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

16

User Interface Design: Swing Basics for Planning the Application Layout

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Explain Abstract Window Toolkit (AWT), Swing, and Java Foundation Classes (JFC)
- Detail the Swing UI containment hierarchy
- Describe how to use layout managers
- Use UI containers to group components within an application
- Embed UI components into UI containers
- Use the Swing pluggable look and feel

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Objectives

This lesson shows you how to add rich graphics functionality and interactivity to your Java applications.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

AWT, Swing, and JFC

- AWT, or Abstract Window Toolkit (`java.awt`):
  - A graphical user interface library
  - The predecessor to Swing components and the foundation for Swing and JFC
- Swing (`javax.swing`):
  - A more powerful graphical user interface library
  - Built on top of the AWT class hierarchy
- Java Foundation Classes (JFC):
  - A collection of APIs including AWT, Swing, Accessibility API, Pluggable Look and Feel
  - Java 2D API, drag-and-drop support (since JDK 1.2)

ORACLE
Copyright © 2009, Oracle. All rights reserved.

AWT, Swing, and JFC

Abstract Window Toolkit (`java.awt`)

The AWT was Java's original set of visual components for the development of graphical user interface (GUI) applications. The AWT is the foundation upon which Swing and the rest of Java Foundation Classes are constructed. The AWT was not designed for high-powered UI development, which can be appreciated when you understand that it has a smaller set of components. AWT classes are found in the `java.awt` package and its subpackages.

Swing (`javax.swing`)

Swing provides lightweight components built on top of the AWT library. Intended for high-powered user interfaces, it provides many more components, which are more efficient than their AWT counterparts. Swing components adhere to the AWT event-handling model that was introduced in JDK 1.1. Swing classes are found in the `javax.swing` package and its subpackages.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

AWT, Swing, and JFC (continued)

Java Foundation Classes (JFC)

Java Foundation Classes is a set of classes and APIs that was first released with JDK 1.1. The version of JFC that is included in JDK 1.2 contains:

- AWT and Swing GUI components
- Accessibility API for people with disabilities
- Pluggable look and feel, to adapt the UI to an operating system look and feel
- Java 2D API for two-dimensional graphics and imaging
- Drag-and-drop support

Because Swing is the major component of JFC, the terms Swing and JFC are often used interchangeably. Oracle JDeveloper (11g) supports JDK 1.5, so Swing is fully supported.

Note: You can create Java client applications that rely on standard Swing components in your application. The Java client application in JDeveloper is known as a *Client* application. When a Client form has been deployed to a client machine, users can use it to display and manipulate data in the form. This course does not teach you about Java Client;
it simply shows you how to develop Java applications by using JDeveloper.

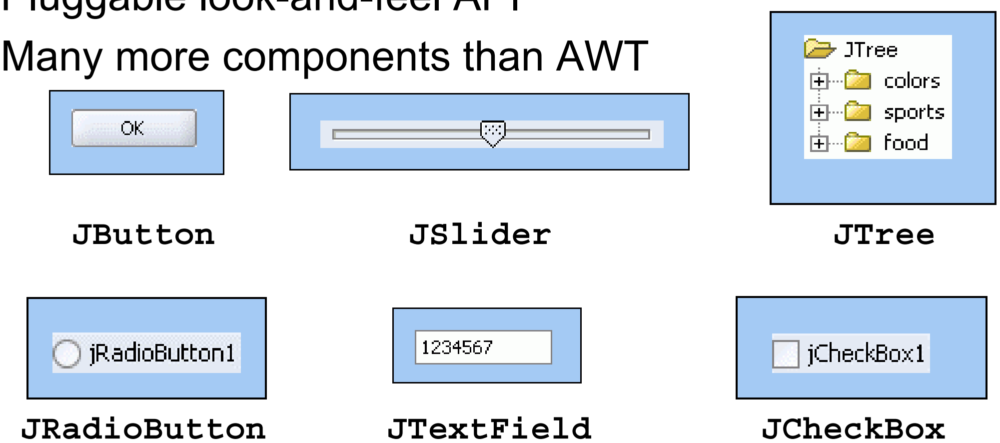
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Features

Swing is a set of visual components that have been available since JDK 1.1 and have been part of the core JDK since version 1.2.

- Lightweight components compared to AWT
- Pluggable look-and-feel API
- Many more components than AWT



Copyright © 2009, Oracle. All rights reserved.

Swing Features

Swing components are all part of the `javax.swing` package, which was added in JDK 1.2. Swing transformed Java UI development by providing lightweight components that could adapt to the look and feel of the operating system in which the application executed. Swing provides many more types of UI components for user interaction than are found in AWT.

Lightweight Versus Heavyweight Components

Swing components are considered lightweight, which means that they are rendered (visually constructed) in their container window. That is, they are created within the Java environment. The container window is usually a native (operating system) window. By contrast, AWT components are heavyweight, meaning that each component is rendered in its own native window. This makes Swing components smaller and more efficient than their AWT counterparts.

Pluggable Look and Feel

Developers can use the pluggable look-and-feel features of Swing to specify the look and feel of applications that are developed with Swing components. The default is to use the Java look and feel (called the Metal look and feel). By using the pluggable look-and-feel API, you can develop your application to use the native look and feel of whatever platform the application happens to be running on, or you can develop your own look and feel.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Lightweight and Heavyweight Components

Heavyweight components
- Strong dependency on native peer code
- Each rendered in *its own* opaque window
- Early AWT components were mostly heavyweight
- Include some top-level Swing components (`JFrame`, `JApplet`, `JDialog`)

Lightweight components
- No dependence on native peer code
- Can have transparent backgrounds
- Most Swing components are lightweight
- When displayed, can appear nonrectangular
- Must be displayed in heavyweight container

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Lightweight and Heavyweight Components

Heavyweight components were developed in early releases of the AWT. Each heavyweight component is tightly associated with a native peer component in the client environment. When rendered, each heavyweight component relies on the creation of *its own* native opaque window. All top-level containers are heavyweight and provide the context for lightweight containers and components.

Lightweight components must ultimately be displayed in heavyweight top-level containers, such as `JFrame`, `JApplet`, or `JDialog`. However, lightweight components are visually more flexible because they can be transparent and they appear nonrectangular. These features enable lightweight components to be easily adapted to a different look and feel. Lightweight components do not have a native peer because they are rendered directly by the Java code. Therefore, lightweight components are more portable.

Note: As a general rule, avoid mixing heavyweight and lightweight low-level components, such as buttons, text fields, and so on. In other words, avoid using AWT and Swing components in the same visual container or application.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Planning the UI Layout

Building a UI application involves planning, even more so when building Swing applications. Planning requires understanding the following concepts and their relationships:

- UI containment hierarchy (a root component that comprises nested containers and components)
- Container levels and types (such as top-level and intermediate containers)
- Layout managers and their types (used by each container)
- Components that can be added to containers

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Planning the UI Layout

Building a Java UI application, whether it is a stand-alone application or an application such as an applet embedded in a browser, requires some basic understanding of:

- The Java UI containment hierarchy that provides a layer of containers nested in containers, and components nested in containers
- The types of Java containers and their relationship in the containment hierarchy
- The concept of layout managers and their types
- Components that can be added to containers

Most graphical applications have a main display area (usually a main window or the applet display area in a Web browser). In Java, the main window or applet display area is called a top-level container. A top-level container is considered to be the root of the containment hierarchy for that window or area.

Note: An application can comprise many top-level windows.

A main window can be divided into regions or sections (which are represented by intermediate containers) and ultimately into components that contain the user data or accept user input. These components are positioned within the top-level or intermediate containers. Together, the top-level and intermediate containers with their components form a containment hierarchy.

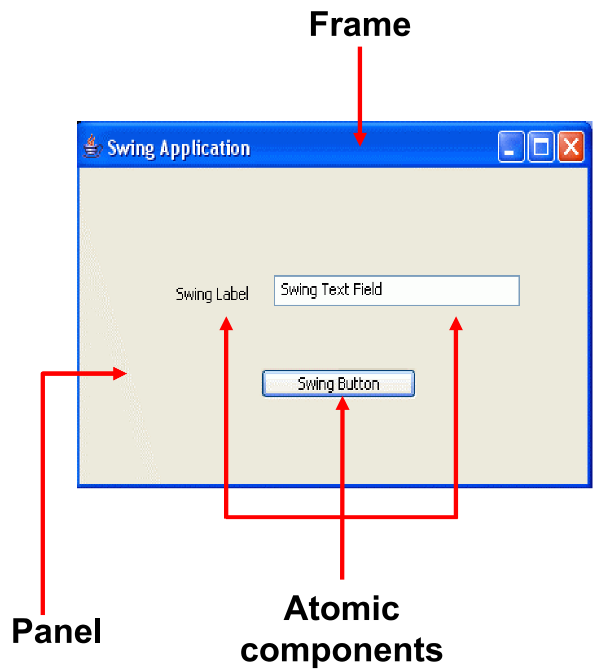
Each container uses a layout manager to control the size and placement of components within a container.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Containment Hierarchy

- Top-level containers
  - Frame
  - Dialog
  - Applet
- Intermediate containers
  - Panel
  - Scroll Pane
- Atomic components
  - Label
  - Text item
  - Button



Copyright © 2009, Oracle. All rights reserved.

Swing Containment Hierarchy

The slide lists the three levels of containers for Swing components that are commonly used in applications: top-level containers, intermediate containers, and atomic components.

Top-level containers provide a place, such as a main application window, for other Swing components to display or paint themselves. Top-level containers cannot be placed within another top-level container, and they usually contain an intermediate container called a *content pane*. Commonly used top-level containers are `JFrame`, `JDialog`, and `JApplet`.

Intermediate containers simplify the way you organize visual items within a top-level container and can contain other intermediate containers and lower-level atomic components. For example, a panel (sometimes called a *pane*) can be nested within another panel. Common intermediate containers are `JPanel`, `JScrollPane`, `JSplitPane`, and `JToolBar`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Containment Hierarchy (continued)

Atomic components are self-sufficient entities (or widgets) that are used to present information to, or receive data from, the user. Common atomic components are `JButton`, `JLabel`, and `JTextField` (as shown in the slide). Many atomic components exist for text, combination boxes, check boxes, tables, and lists, to name a few.

Note: The slide shows the following containment hierarchy:

- Frame (top-level container contains a …)
  - Panel (intermediate container, which contains …)
    - Label
    - Text field
    - Button

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Top-Level Containers

- Swing provides `JFrame`, `JDialog`, and `JApplet`, which have changeable properties such as:
  - Content panes for holding intermediate containers or components, using the `getContentPane()` or `setContentPane()` methods
  - Borders, using a `setBorder()` method
  - Titles, using a `setTitle()` method
  - Window decorations, such as buttons for closing and minimizing (excludes applets)
- AWT provides `Frame`, `Dialog`, and `Applet`
  - These do not provide properties such as a content pane or borders.

Copyright © 2009, Oracle. All rights reserved.

Top-Level Containers

Each Swing application has at least one top-level container or frame. The top-level container can be an instance of `JFrame`, `JDialog`, or `JApplet`. It is easy to customize the top-level content pane to add a border or set the layout manager. However, using the top-level content pane methods is tricky. The methods of the top-level containers return a `Container` object, and not a `JComponent` object. This means that you must typecast the return value of the methods in order to use them.

An easier way to achieve the same results is to create your own content pane, typically by using a `JPanel` object. You then call the `JFrame setContentPane()` method to set the top-level content pane to be your customized `JPanel`. You now have complete control of the content pane without the restrictions of the top-level or root container.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Top-Level Containers (continued)

The following example creates a top-level container by using a `JFrame` object, and an intermediate container as a `JPanel`. After the code customizes the panel by changing its layout manager and applying a border, the top-level container's content pane is modified to use the panel by calling the `setContentPane()` method:

```
JFrame topLevelContainer = new JFrame();
JPanel myContentPane = new JPanel();
myContentPane.setLayout(new BorderLayout());
myContentPane.setBorder(new
LineBorder(Color.lightGray,0));
topLevelContainer.setContentPane(myContentPane); // or
topLevelContainer.getContentPane().add(myContentPane);
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Intermediate Containers

- Designed to contain components (or containers); can be nested within other containers
- Types of intermediate containers:
  - Panels for grouping containers or components
  - Scroll panes to add scroll bars around components that can grow, such as a list or a text area
  - Split panes to display two components in a fixed area that is adjustable by the user
  - Tabbed panes for containing multiple components, showing only one at a time based on user selection
  - Toolbars for grouping components, such as buttons
  - Internal frames for nested windows

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Intermediate Containers

The next level of containers in Swing is designed for the sole purpose of containing other components. These containers may hold any other Swing component, including other containers. By nesting intermediate containers within other containers, you can control the layout of your application. This technique is described later in this lesson.

The intermediate containers are the following:

- **Panels:** These are the most frequently used intermediate containers. They are implemented with the `JPanel` class. They are generally used to group components for logical presentation to the user. A `JPanel` can use any layout manager; by default, it uses the `FlowLayout`, and you can set its border to any border.
- **Scroll panes:** These provide scroll bars around any large component or one that may grow. They are implemented with `JScrollPane`.
- **Split panes:** These are used to present two components in a fixed amount of space while letting the user adjust the space that is devoted to each item. Split panes are implemented with `JSplitPane`.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Intermediate Containers (continued)

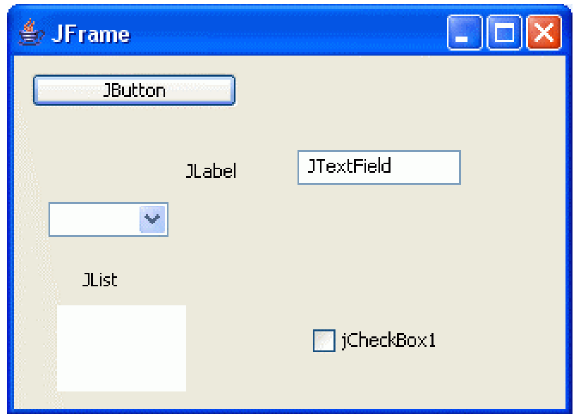
- **Tabbed panes:** This container possesses multiple components, but the user can see only one at a time. The user can switch between the components by clicking the tabs. Tabs are implemented with `JTabbedPane`.
- **Toolbars:** In addition to holding multiple components, instances of `JToolBar` can be repositioned by the user.
- **Internal frames:** Top-level containers can support internal windows or frames, which are implemented by `JInternalFrame` and best used with a `JDesktopPane`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Atomic Components

- Buttons
- Check boxes
- Combo boxes
- Text
- Lists
- Labels



Copyright © 2009, Oracle. All rights reserved.

Atomic Components

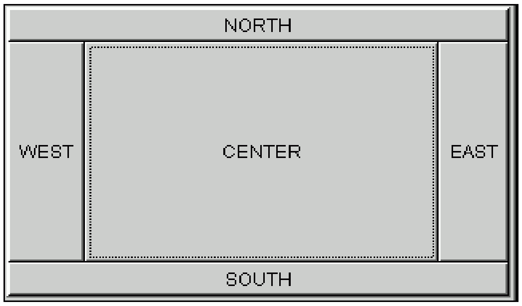
Atomic components exist solely to present or accept information. They do not serve as containers for other components. Atomic components inherit from `JComponent` and thus support standard component features such as borders and tool tips.

Oracle University and Global Business Solutions Peru SAC use only

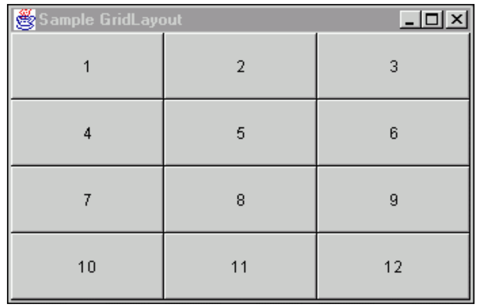
THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Layout Management Overview

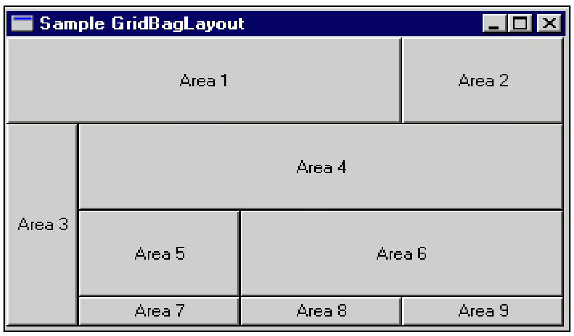
Border layout



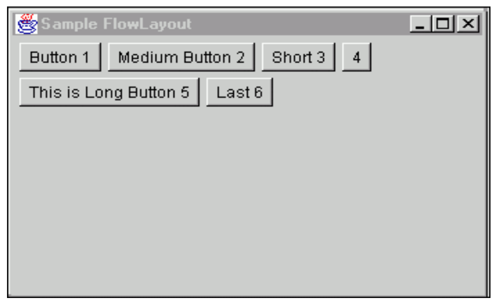
Sample grid layout



Sample gridbag layout



Sample flow layout



Copyright © 2009, Oracle. All rights reserved.

Layout Management Overview

You can use layout managers to control the process of placing components onto a container at run time. Each container has a layout manager by default. The layout manager ultimately controls the layout and position of components within the container. However, each component can provide hints about itself to assist the layout manager, such as its preferred size and position.

Java provides many layout managers. The following are commonly used:

- `java.awt.BorderLayout`, which is the default for `JFrame` containers, arranges the container into five areas that are called North, South, East, West, and Center.
- `java.awt.FlowLayout`, which is the default for `Jpanel`, organizes items from left to right and then from top to bottom. The rows can be centered (default), right-justified, or left-justified.
- `java.awt.GridLayout` arranges items in a grid in rows and columns with cells of the same size.
- `java.awt.GridBagLayout` arranges items in a grid of rows and columns with different cell sizes. This is the most flexible and complex of all the layout managers, and it enables components to span multiple rows and column cells.
- `javax.swing.BoxLayout` arranges items in a stack horizontally or vertically.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Layout Management Overview (continued)

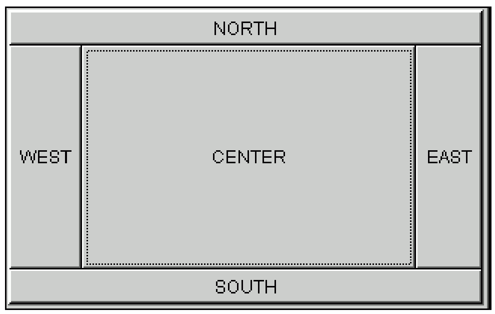
Null: You can set a container layout property to `null`, thereby forcing the container not to use any layout manager with the rules described. In this case, *absolute positioning*, specific position, and size in pixels control the UI component. Absolute positioning is inflexible to changes in the shape of the top-level container at run time. However, it can be useful in design stages to provide precise control over the placement and size of each component.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Border Layout

- Has five areas: north, south, east, west, and center
- Has center area that expands to fill the available space
- Displays only one component in each area
- Makes each area useful for holding intermediate panels



Copyright © 2009, Oracle. All rights reserved.

Border Layout

Border layout provides five areas for components: north, south, east, west, and center. If the user enlarges the window, the center area expands to use as much of the space as possible. The other areas expand only as much as necessary to fill the available space. For example, if the user makes the frame wider, the center expands horizontally but the east and west areas do not; however, the south area expands to fill the new window size.

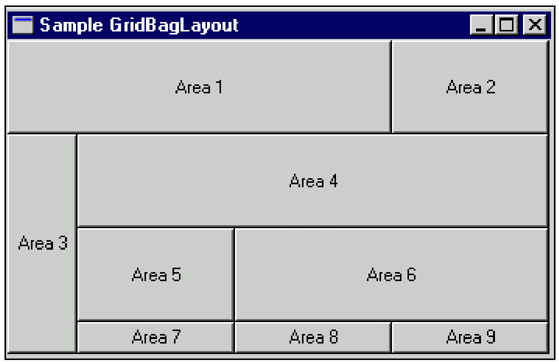
Each area displays only one component. To overcome this restriction and make Border a useful layout manager, add containers to the areas instead of atomic components. Most panels that use Border use only one or two of the areas, such as center and south. South may be used for a toolbar or navigation, whereas center may contain a panel that holds all the atomic data components. This technique is useful in creating a resizable frame.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

GridBag Layout

- Is based on a grid
- Allows components to span multiple rows and columns
- Allows rows and columns to differ in size
- Uses the component's preferred size to control cell size



ORACLE
Copyright © 2009, Oracle. All rights reserved.

GridBag Layout

GridBag layout is the most flexible and most complex of the layout managers. The flexibility comes from its ability to enable components to span multiple rows and columns. In addition to spanning multiple columns and rows, the components can provide hints or suggestions about how the component should appear. For instance, a component can specify how much space to automatically set around the component, both inside and outside the component's cell. You can also specify minimum, maximum, and preferred sizes for each component.

Components can span multiple cells in both directions, in both rows and columns. Row and column size is determined by the size of the components that occupy the row and column.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

GridBag Constraints

External insets

Cell position

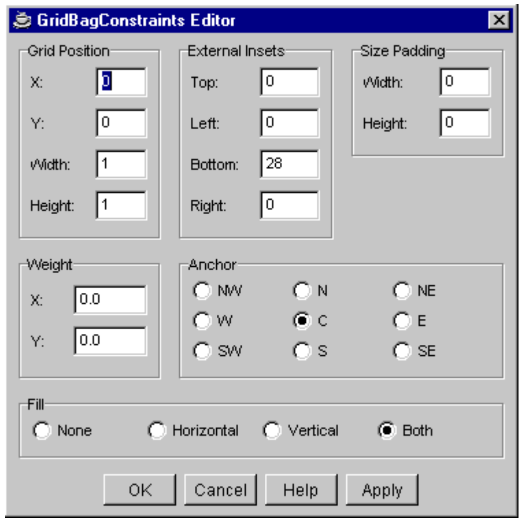
Cell span

Expansion weighting

Fill rules

Component padding

Anchoring



ORACLE

Copyright © 2009, Oracle. All rights reserved.

GridBag Constraints

Each component in a `GridBagLayout` container has properties that you can set to control the layout behavior for the component. You edit the constraints by selecting the component and clicking constraints in the Properties Inspector window. Alternatively, you can right-click the component and select constraints from the context menu.

Layout Constraints

- **Cell position:** The X and Y properties specify the grid cell for the upper-left corner of the component. The values are integers and represent the cell number in a row and column.
- **Cell span:** These properties specify how many columns (width) and rows (height) the component occupies.
- **External insets:** These values specify the amount of space between the component and the edge of its display area. You can specify a value for the top, bottom, left, and right.
- **Component padding:** These values specify the amount of space around a component within a cell. The width of the component is calculated as the minimum width plus the width property. The height is calculated as the minimum height plus the height property.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

GridBag Constraints (continued)

- **Expansion weighting:** This specifies how extra space is distributed vertically (X) and horizontally (Y). The range of values is 0 through 1.0. Weight determines what share of the extra space is allocated to each component.
- **Fill rules:** These values specify what to do if the display area is larger than the component.
- **Anchoring:** This indicates where to anchor the component if the component is smaller than the display area.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using Layout Managers

- Layout managers are designed to manage multiple components simultaneously.
- Using a layout manager with containers requires:
  – Creating a container and a layout manager object
  – Setting the layout property of the container
  – Adding items (components or other containers) to the regions that are defined by the layout manager
- Different layout managers require different arguments to control component placement.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Using Layout Managers

These layout managers are designed to manage multiple components at once. The basic steps to use a layout manager are shown in the slide. The examples in the slide show creating a frame by using the `javax.swing.JFrame` class, to which you apply a `java.awt.BorderLayout` manager.

You create the layout manager object. Then you call the frames `setLayout()` method. Finally, you start adding components or other containers to the regions that are provided by the layout manager.

Create Container and Manager

```
JFrame myFrame = new JFrame();
BorderLayout layoutMgr = new BorderLayout();
```

Set Properties

```
myFrame.setLayout(layoutMgr);
```

Add Items

```
myFrame.add(new JButton(), BorderLayout.NORTH);
myFrame.add(new JTextField(), BorderLayout.SOUTH);
myFrame.add(new JPanel(), BorderLayout.CENTER);
```

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using Layout Managers (continued)

Adding Components to Containers

When adding a component to a container, you must always consider the layout manager that is used by the container. Each type of layout manager may require different arguments to control the placements and/or size of component that is added. For example, when adding a component to a container by using the `BorderLayout` manager, you are required to specify the border area in which you want the component placed—north, south, and so on. `FlowLayout` does not require a placement parameter, and merely appends components in the order in which they are added to the container.

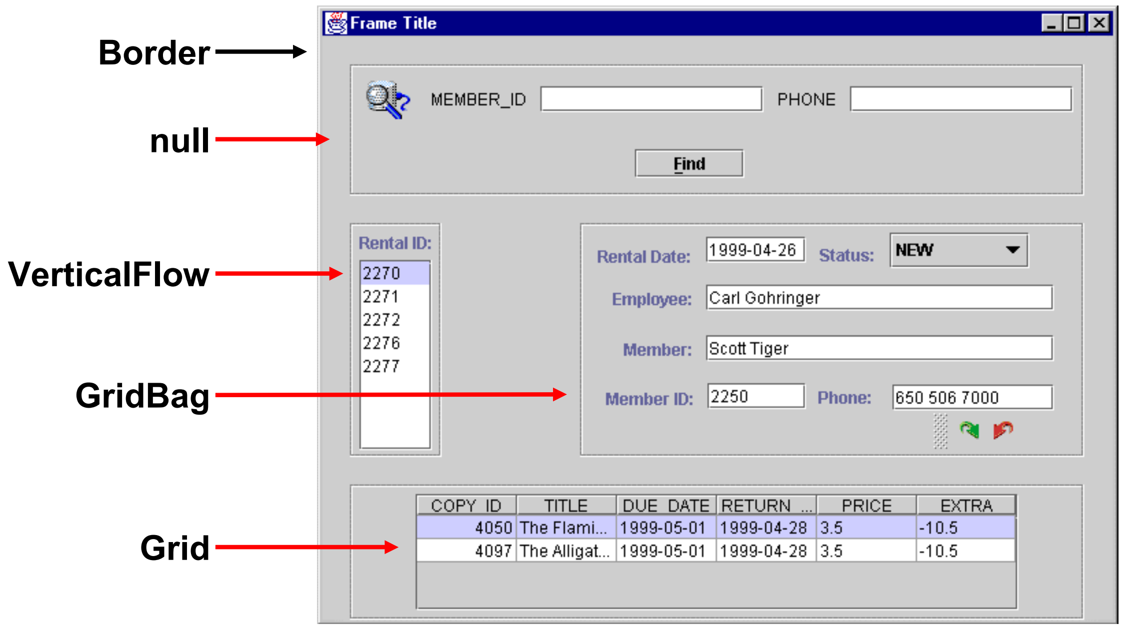
For most Swing code, such as the examples shown in the slide, you should import classes from the following packages: `javax.swing`, `java.awt`, and `java.awt.event`.

Note: If you are using an IDE tool like JDeveloper, you can set the layout property of a container to `null` to force the absolute position to be used. This makes it very convenient for you when designing and prototyping a user interface. Later, you can switch the layout property to a suitable Java layout manager class for the application.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Combining Layout Managers



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Combining Layout Managers

Different layout managers are good at different tasks. Because you can place multiple panels in a frame and each one may have a different layout manager, you have a lot of control over the ultimate layout.

Nesting panels and layout managers is a common practice. In the example in the slide, you use the Border, null, VerticalFlow, GridBag, and Grid layout managers.

The top-level frame uses Border, which enables you to specify what nested panels go in the north, south, east, and west areas. The top panel uses null, which enables you to place the components where you want them to be displayed.

The RentalID panel uses VerticalFlow, which displays the items that are stacked vertically. You use GridBag in the Rental panel (east) so that you can align components of differing sizes. Lastly, you use Grid in the bottom panel, which contains only one component.

Using a combination of these layout managers offers very fine control over the layout of your application. You can create a form that is resizable without losing its general look and feel.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Frame Classes

A Java frame is equivalent to an application window.

- Use `JFrame` for a main window.
  - It has properties for icons, title, and the buttons to minimize, maximize, and close.
  - It uses `BorderLayout` by default.
  - It provides a default content pane that occupies the center region of the layout.
  - You can set the frame size with the `setSize()` method and make it visible by using the `setVisible()` method.
- Use `JDialog` for a modal window.
  - You must dismiss a modal window before the application that invokes it can become active.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Java Frame Classes

Java frames are analogous to top-level application windows. These windows contain all the functionalities that are provided by the operating system to manage the window, such as a bar containing title string, an icon, and the minimize, maximize, and close buttons. These windows are also resizable unless you programmatically disable this feature.

When you create a Java `JFrame` object, you automatically get a content pane that provides the container for the window objects or components. As stated in a previous slide, you typically replace the content pane with an intermediate component like a `JPanel` to simplify management of the visual contents of the container.

The `JFrame` uses a `BorderLayout` manager by default, where the default content pane is located in the center region. You can alter the frame to have a menu and/or a toolbar that is commonly placed in the north region and a status bar that would typically be placed in the south.

`JDialog` classes can be modal or nonmodal. They tend to be modal in nature. It is more common to create dialog boxes by using the `JOptionPane` class methods, such as the `showMessageDialog()`. Otherwise, you can use the JDialog class to create custom dialog boxes.

Note: On the next page, there is an example of a simple frame application with a default content pane but no intermediate containers or components.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Frame Classes (continued)

Creating a Simple Frame Application

This example shows how to build a Java GUI application by using the `JFrame` class. The code example illustrates the points that are discussed in the slide of the previous page.

```
import java.awt.Color;
import javax.swing.JFrame;
public class MyFrame extends JFrame
{
  public MyFrame()
  {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    getContentPane().setBackground(Color.blue);
    setTitle("Default Frame Title");
    setLocation(50, 50);
    setSize(600, 400);
  }

  public static void main(String[] args)
  {
    JFrame f = new MyFrame();
    f.setResizable(true);
    f.setVisible(true);
  }
}
```

Note

- By default, the `JFrame` default operation on a close event is to hide the window. The call to `setDefaultCloseOperation(EXIT_ON_CLOSE)` changes the default operation that is performed by the JVM when the window is closed. The `EXIT_ON_CLOSE` constant is defined in `javax.swing.WindowConstants`, which is an interface implemented by `JFrame`.
- The `getContentPane()` method is used to access the frame's default container and change the background color to blue.
- The `setLocation()` determines the top left x and y coordinates (in pixels) of the window relative to the upper-left corner of the screen.
- The `setSize()` method sets the width and height of the window (in pixels).
- The `setLocation()` and `setSize()` can be done in one step by calling `setBounds(x, y, width, height)`.
- The example shows how you can set properties of the frame either in the constructor or by using a reference to the frame (as shown in the `main()` method).

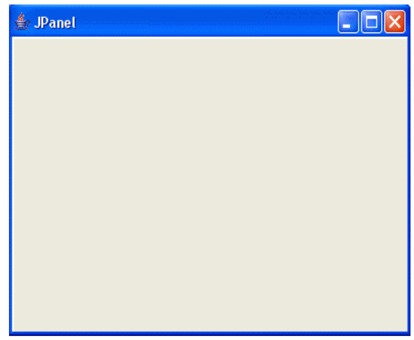
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`JPanel` Containers

`JPanel` is a general-purpose container; `JPanel` can:

- Use any layout manager (uses `Flowlayout` by default)
- Use any border
- Have added components or other panels or containers by using the `add()` method




```
JPanel myPanel = new JPanel(new BorderLayout());
JTextArea jTextArea1 = new JTextArea();
myPanel.setBorder(BorderFactory.createRaisedBevelBorder());
myPanel.add(jTextArea1, BorderLayout.SOUTH);
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

`JPanel` Containers

`JPanel` is a general-purpose container that is designed to hold other components. You can use `JPanel` containers to group components within an area of an application. You may add a border to the panel to help visually separate the components from other components in the application.

Setting the Layout Manager

The default layout manager for `JPanel` is `FlowLayout`, which places all the components within the container in a row. You can make the panel use another layout manager by calling the `setLayout()` method or by specifying the layout manager when you create the panel.

Examples:

```
JPanel myPanel = new JPanel();
myPanel.setLayout(new BorderLayout());
```
or
```
JPanel myPanel = new JPanel(new BorderLayout());
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

`JPanel` Containers (continued)

Adding Components

You can add components to the panel by using the `add()` method. The arguments that are provided to the `add()` method depend on which layout manager is used by the panel. For example, `FlowLayout`, `GridLayout`, and `GridBagLayout` typically accept one argument. If the layout manager is `BorderLayout`, additional arguments are used to specify the position of the contained components.

Examples:

```
myPanel.add(button);                      // if FlowLayout
myPanel.add(button, BorderLayout.NORTH); // if BorderLayout
myFrame.getContentPane().add(myPanel);    // add to a frame
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Internal Frames

An internal frame is the equivalent of a document window that is contained in an application window for multiple-document interface (MDI) window applications.

- Use `JInternalFrame` for an internal window:
  - Like a `JFrame`, a `JInternalFrame` can contain intermediate containers and components and use a layout manager.
  - By default, a `JInternalFrame` is not “closable,” “iconifiable,” “maximizable,” or visible.
- Use a `JDesktopPane` as the content pane in which the internal frames are added; the `JDesktopPane`:
  - Controls the size and placement of internal frames
  - Uses a null layout manager by default

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Internal Frames

The Swing API also provides an internal frame, which is implemented by the `JInternalFrame` class. An internal frame creates a window in another window that you can use to build an application that conforms to the popular multiple document interface (MDI) model of the Windows platform.

When using an internal frame, the `JDesktopPane` class is provided as a container to manage the size and placement of the internal frames within the containing window. Therefore, you would normally create a `JDesktopPane` object to replace the existing frame’s default content pane. The internal frames are then added to the desktop pane.

Like `JFrame`, `JInternalFrame` has a window title bar with a title, icon, and window decorations such as buttons to maximize, “iconify,” and close (which by default are disabled). The internal frames can be dragged over each other, and an internal frame provides methods to control whether it is on top, selected, and so on. Some examples:

```
setResizable(boolean), setIconifiable(boolean)
setMaximizable(boolean), setVisible(boolean)
toFront(), toBack()
```

The desktop pane provides methods to obtain an array of internal frame objects that have been added to its container, as in the following examples:

`getAllFrames()` returns an array of JInternalFrame objects.
`getSelecedFrame()` returns the currently selected JInternalFrame.

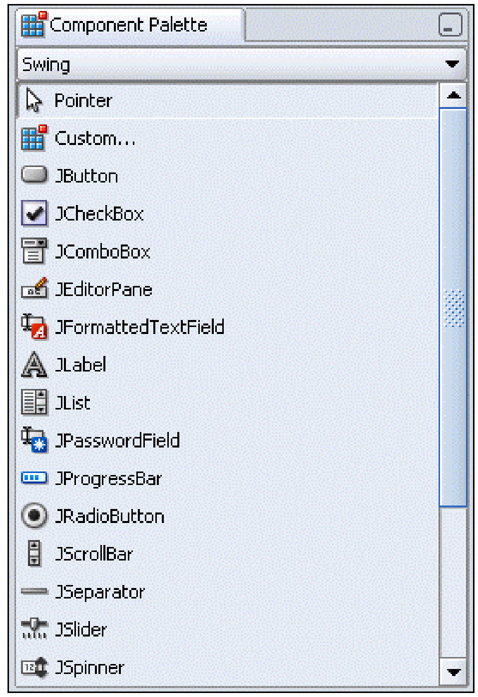
Note: Most `get`*`XXX`*`()` methods have a corresponding `set`*`XXX`*`()` method.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Adding Components with Oracle JDeveloper

1. Create a `JFrame`.
2. Select a layout manager.
3. Add components from the Component Palette.
4. Fine-tune component properties.



Copyright © 2009, Oracle. All rights reserved.

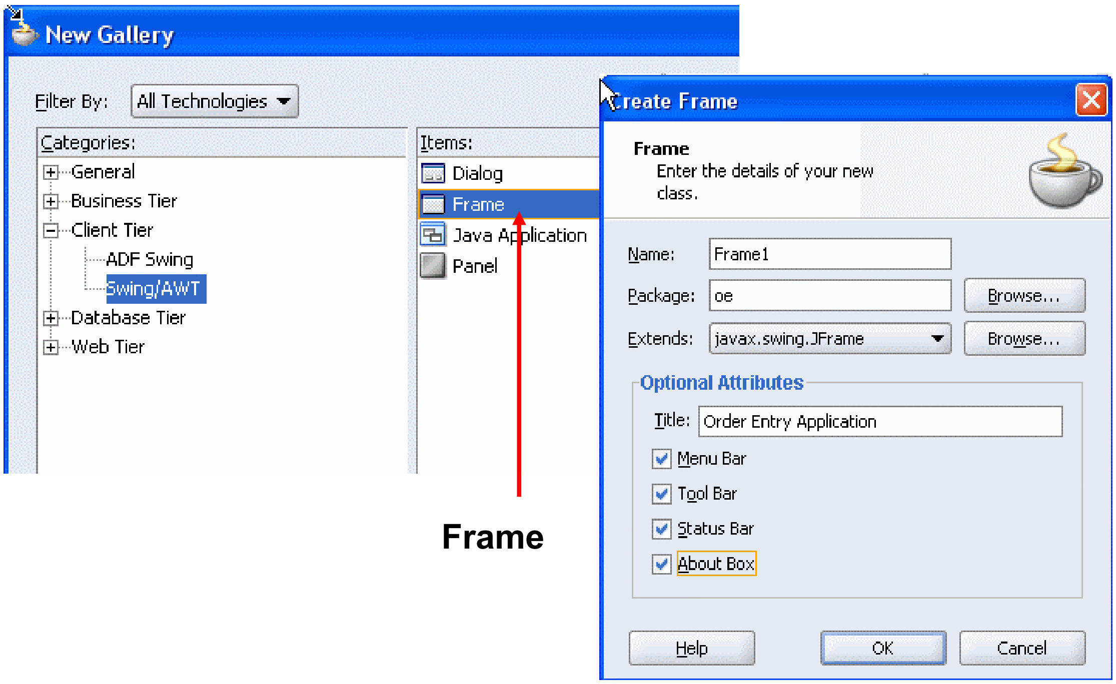
Adding Components with Oracle JDeveloper

Adding components to an application is a straightforward process with Oracle JDeveloper. You create an empty frame and then add the components that you want. The components can be Swing containers or Swing atomic components, such as text fields, buttons, and check boxes. After you add the components, you can fine-tune the components by using the Properties Inspector or by adding or changing the code in the Code Editor window.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating a Frame



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating a Frame

To create a new `JFrame`, select File > New from the JDeveloper menu. Expand the Client Tier, and then select the SWING/AWT node. Look for the Frame item in the Items list. In the Create Frame dialog box, change the name of the class and the frame title to something meaningful. Select `javax.swing.JFrame` as the base class. You can select options to create a menu bar, status bar, toolbar, and "About" box. These are all optional attributes.

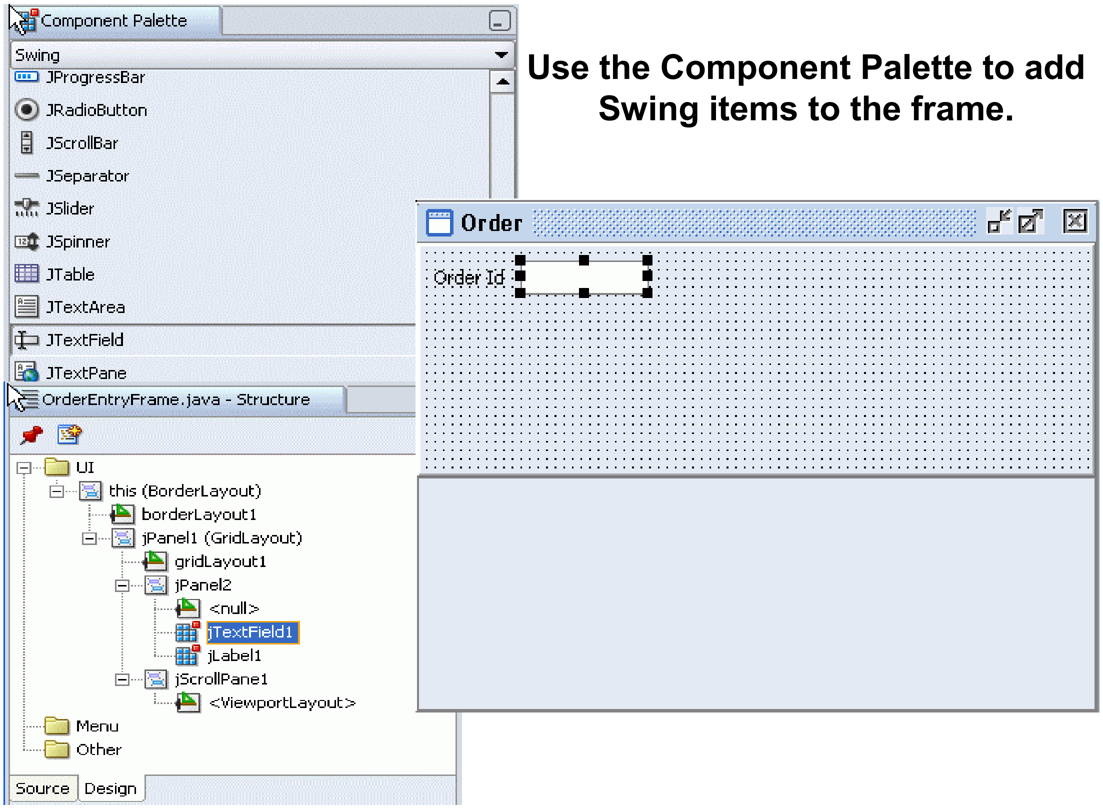
You can specify the preferred superclass for the new frame. JDeveloper generates a class with the required import statements and the code that is necessary to create a usable frame. Now you can use the JDeveloper UI Editor to construct the application UI structure visually.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Adding Components

Use the Component Palette to add Swing items to the frame.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adding Components

The `JFrame` that is created by the Frame Builder Wizard includes the frame and an intermediate `JPanel` container. The wizard does not set the layout manager for the generated `JPanel`. It uses the default layout manager for its type of container (`FlowLayout`).

Because it is a default layout manager, JDeveloper cannot provide the ability to alter the properties of the layout manager. It is best to change the layout manager so that you can manipulate the layout properties.

After setting the layout manager, you can add a component by selecting it from the Swing page of the Component Palette and dragging it onto the `JPanel` in the Design window. Alternatively, you can click the component in the Component Palette and then click JPanel in the structure window. If you choose the latter, JDeveloper uses default sizes for components. In either case, the layout manager affects the final location of the component.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Adding Components (continued)

To invoke the UI Editor, select a class in the navigator, right-click, and select UI Editor.

In general, add components to the structure window instead of directly to the panel. This approach is best if you want to avoid adding a component to the wrong panel by accident. For instance, adding components to a `JTabbedPane` inside a panel can be done in an easier manner by using the structure pane.

When visually adding a component into a frame or panel with JDeveloper, it generates code to:

- Declare and instantiate the selected component object
- Set minimal properties for the default state
- Add the component to the chosen container

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Pluggable Look and Feel

Swing applications provide support for a different look and feel to adapt to the visual environment of the operating system. The look and feel:

- Is application-specific:
  - Can be initialized when the application starts
  - Can change dynamically
- Affects lightweight Swing components
- Supports Windows, Macintosh, Java (Metal), and Motif platforms
- Uses the `javax.swing.UIManager` class
  - Provides the `setLookAndFeel()` method, which accepts a look-and-feel class name string

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Pluggable Look and Feel

Java provides a set of classes in the `javax.swing.plaf` package and subpackages that help render components in a platform-specific manner. The default Java look and feel is known by its code name "Metal," which is the name of the project that was assigned to a team at Sun Microsystems, Inc. to create a unique and distinctive look and feel for Swing 1.0 (JFC 1.1).

Setting UI Look and Feel

Use the `javax.swing.UIManager` class to initialize, or dynamically change, the look and feel of your application. For example, in the `main()` method of your frame application, you can add the following code:

```
try { UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) { }
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Setting UI Look and Feel (continued)

The value that is returned by `getSystemLookAndFeelClassName()` is a string representing a fully qualified class name that implements the look and feel of the current platform. The class name string is provided as the parameter to `UIManager.setLookAndFeel()`. Some possible values for the class names of different platforms are:

```
javax.swing.plaf.metal.MetalLookAndFeel
com.sun.java.swing.plaf.windows.WindowsLookAndFeel
com.sun.java.swing.plaf.motif.MotifLookAndFeel
```

Note: If you want to change the look and feel dynamically, you can call the `SwingUtilities.updateComponentTreeUI(getContent Pane())` method.

This method makes the existing components reflect the new look and feel.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Plan the layout of a Swing-based UI
- Add UI components with JDeveloper
- Manage the look and feel of a Swing application

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 16 Overview: Swing Basics for Planning the Application Layout

This practice covers the following topics:

- Creating a class based on `JFrame` for the main window of the `OrderEntry` application
  - Adding a default menu and status bar
  - Adding a `JDesktopPane` and setting it as the content pane
- Creating a class based on `JInternalFrame` to manage order creation and data entry
  - Creating the container layout hierarchical structure for the order-entry frame components
  - Adding some of the components to this frame
- Setting layout managers for each container

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 16 Overview: Swing Basics for Planning the Application Layout

The goal of this practice is to use JDeveloper to create the main application frame as an MDI window and to create the internal order frames that are contained in the main window. Working with these frames helps you explore Swing classes and the ways to build GUI applications.

Note: For this practice, you use the `OrderEntryApplicationLes16` application.

Viewing the model: To view the course application model up to this practice, in the Applications – Navigator node, expand `OrderEntryApplicationLes16 – OrderEntryProjectLes16 - Application Sources – oe` and double-click the `UML Class Diagram1` entry. This diagram displays all the classes created up to this point in the course.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Adding User Interface Components and Event Handling

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Add Swing components to an application
- Get and modify the contents of the components
- Provide event handlers for common types of events
- Create a menu bar

Copyright © 2009, Oracle. All rights reserved.

Objectives

This lesson also deals with the user interface for your Java applications. You learn how to use standard Java Swing components as well as the more sophisticated controls provided by Oracle JDeveloper. You also learn how to provide event handler methods to deal with events such as button clicks and text changes.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Components

- Text controls
  - `JTextField`
  - `JPasswordField`
  - `JTextArea`
  - `JEditorPane`
  - `JTextPane`
- Graphic controls
  - `JTree`
  - `JTable`
  - `JToggleButton`

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Swing Components

Text Components

Swing text components display text and can (optionally) enable users to edit text. There are five text components that support varying complexities and requirements:

- `JTextField`: Can display and edit only one line of text at a time
- `JPasswordField`: Subclass of `JTextField` (It works in the same way as `JTextField` except that the input is hidden from the user.)
- `JTextArea`: Can display and edit multiple lines of text (This component is used to enable users to enter text of any length. It can display text in any font.)
- `JEditorPane`: Enables the use of more sophisticated text styles, including multiple fonts and embedded images (`JEditorPane` can read and write plain text, HTML, and RTF text.)
- `JTextPane`: In addition to the facilities that are provided by `JEditorPane`, this component allows embedded components.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Components (continued)

Graphic Components

- `JTree`: Control that displays a set of hierarchical data as a tree diagram
- `JTable`: Component that displays data in a two-dimensional table
- `JToggleButton`: Toggle buttons are similar to `JCheckBox`. When they are clicked (set to `true`), they remain `true` until they are programmatically set to `false`.

Swing Containers Toolbar

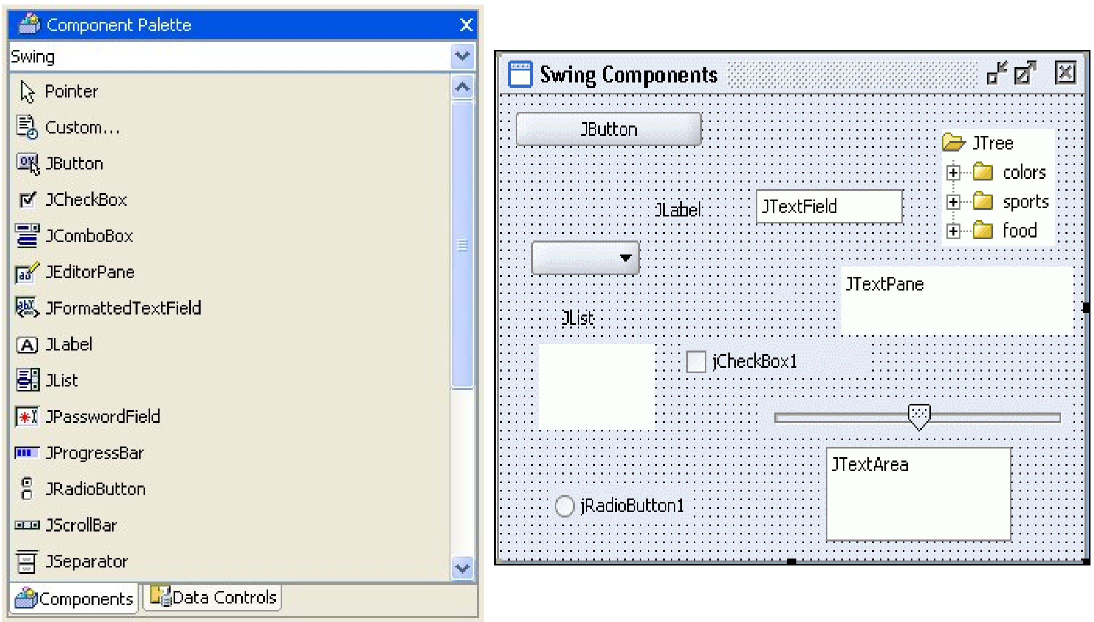
The Swing Containers toolbar holds components that are intended to contain other components.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Components in JDeveloper

Use the Swing Component Palette to add items.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Swing Components in JDeveloper

The main window contains the Component Palette, which holds all the graphical and nongraphical controls that are available. The Swing page of the palette contains the Swing components. Here is a brief summary:

- `JButton`: Push button
- `JCheckBox`: Check box that can be selected or cleared
- `JComboBox`: Combination of text field and drop-down list
- `JEditorPane`: Styled text area that can display text in different formats, including RTF and HTML
- `JLabel`: Short text string or an image that cannot be selected
- `JList`: List of items from which the user can select
- `JPasswordField`: Text field that displays a character such as an asterisk (*) instead of showing what the user enters
- `JProgressBar`: Graphic display showing how much of a task is completed
- `JRadioButton`: One of a group of option buttons
- `JScrollBar`: Horizontal or vertical scroll bar
- `JSeparator`: Component that draws a straight line
- `JSlider`: Component with which users can select a value by sliding a knob

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Components in JDeveloper (continued)

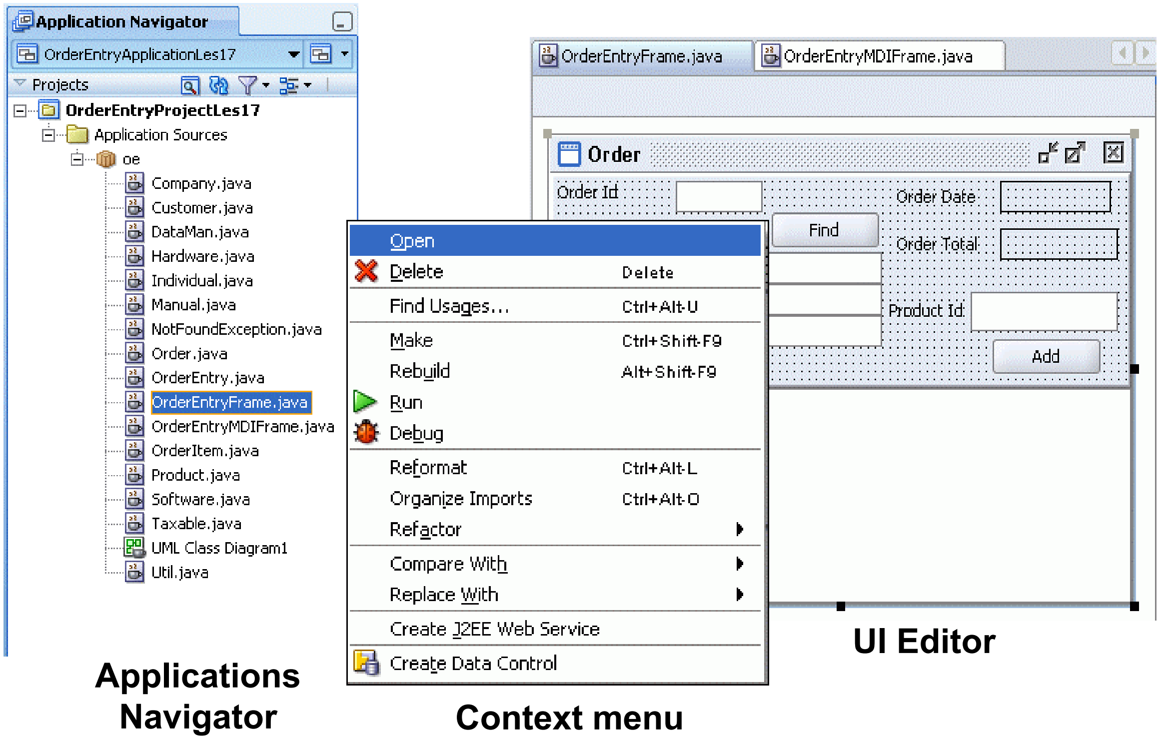
- `JTextArea`: Multiline text field
- `JTextField`: Single-line text field
- `JTextPane`: Styled text area that you can use to define your own text formats
- `JTree`: Control that displays a set of hierarchical data as a tree diagram
- `JTable`: Component that displays data in a two-dimensional table
- `JToggleButton`: Toggle buttons are similar to `JCheckBox`. When they are clicked (set to `true`), they remain `true` until they are programmatically set to `false`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Invoking the UI Editor

Right-click and select Open in the context menu.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Invoking the UI Editor

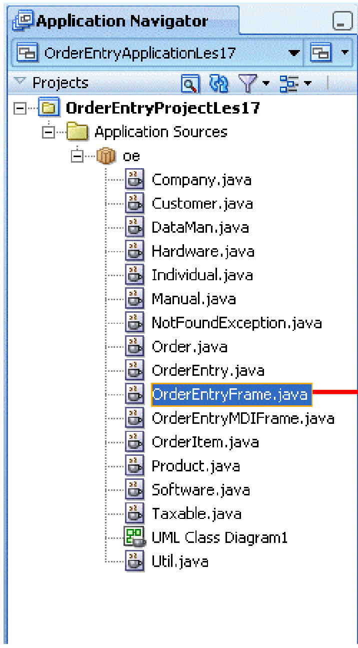
You can right-click the class in the Applications Navigator and then select Open to view the class. The class is displayed in the Code Editor. At the bottom of the pane are three tabs: Source, Design, and History. Clicking the Design tab displays the class in the UI Editor.

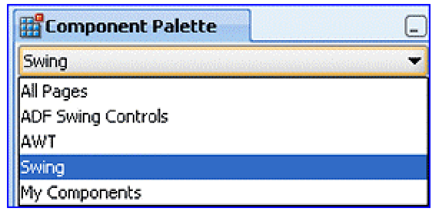
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

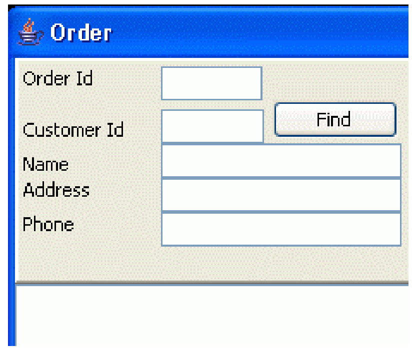
Adding a Component to a Form

1. Open the Component Palette and select the Swing category.





2. Drag the component to the form. The class updates automatically.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Adding a Component to a Form

Note that you can add components to the structure window as well as directly to the panel. Adding components to the structure window is best if you want to avoid adding a component to the wrong panel by accident, particularly when the panel regions are not visible in the UI Editor. Make sure that you drag the component to its container in the structure window. For example, adding components to a `JTabbedPane` inside a panel is easily done using the structure window.

Changes to the Source Code When Adding Components

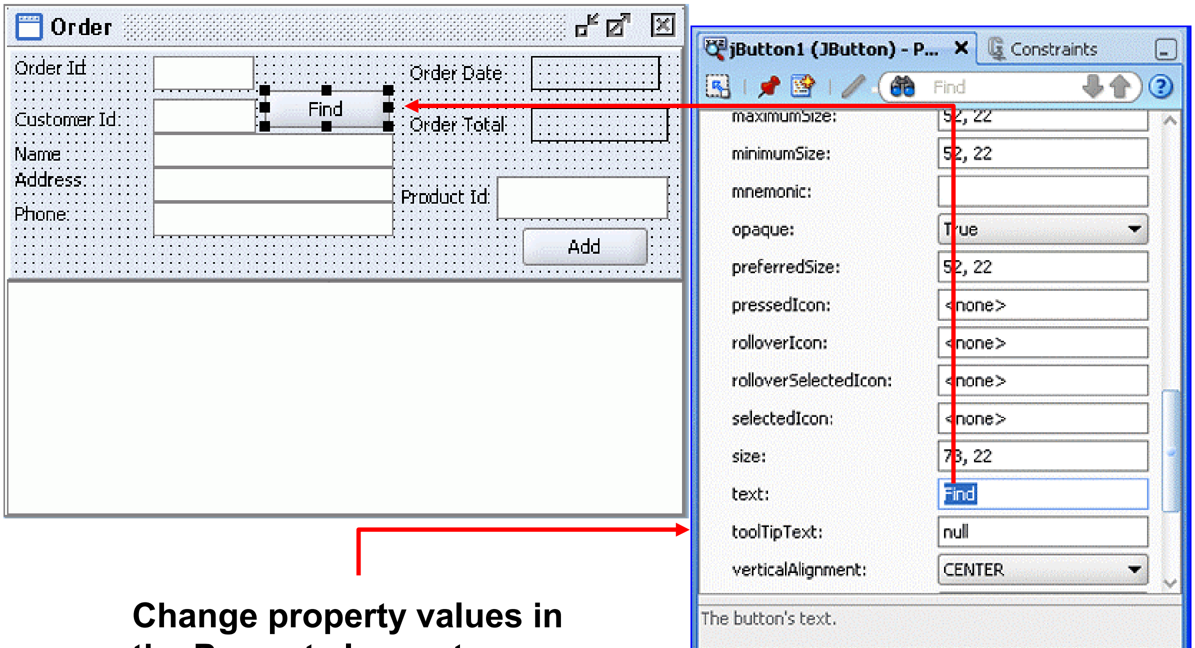
If you examine the source code changes before and after you add a component, you notice that JDeveloper makes the following changes to the class that is being edited; JDeveloper:

- Adds an import statement for the component's class (if not already present)
- Creates an instance variable by using the component class name as the type, and creates a default instance variable name (by using default/package-level access)
- Adds lines to the `jbInit()` method to set the default properties for the component, such as the initial text value of a `JTextField`. In addition, a code line is generated to add the component to its container.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Editing the Properties of a Component



Change property values in the Property Inspector.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Editing the Properties of a Component

Select a Swing component. In the JDeveloper menu, select View > Property Inspector to view the component's properties.

Changes that are made to the properties of a component modify the source code to reflect the changes that are made in the Inspector window.

Note: When changing a text field that requires you to enter the value, press the Enter key to accept the change that is made.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Code Generated by JDeveloper

Example: Adding `JButton` to `JFrame`

```
import javax.swing.JButton;
public class JFrame1 extends JFrame {
 private JButton jButton1 = new JButton();
 ...
     public void jbInit() throws Exception {
        this.setLayout(null);
        jButton1.setText("jButton1");
        jButton1.setBounds(new Rectangle(25, 140,
                73, 22));
        this.add(jButton1, null);
   }
```

Copyright © 2009, Oracle. All rights reserved.

Code Generated by JDeveloper

Whenever you modify a UI, JDeveloper updates the source code for that class to achieve the design that you specify. In fact, the source code is all that matters; the Designer tool is just an easy way to generate the source code that is required to achieve a certain visual appearance.

Instance Variables and Their Names

Each time you add a component to the UI, JDeveloper adds a corresponding instance variable to your class. By default, the instance variable is assigned a default name based on the class name and number—for example, `jButton1` (as shown in the slide). To give the variable a more meaningful name, change its `name` property in the Inspector to replace all usages of the name in the class. If you change the variable name manually in the source code, you must remember to replace *all* occurrences of the variable in the class.

The code lines that are generated in the `jbInit()` method to initialize the component and add it to its container will vary based on the type of layout manager that is used by the container.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Code Generated by JDeveloper (continued)

Methods That Set Properties

JDeveloper calls a method to set each property that you edited in the Property Inspector. In the example, the button's text is changed to `Find` and the size of the button's text is changed to 16 points. The two methods that are called are `setText()` and `setFont()`:

```
jButton1.setText("Find");
jButton1.setFont(new Font("Dialog", 1, 16));
```

Component Objects Added to the Container

The `jbInit()` method adds each component object to the container. The location and the size is specified as follows:

```
jButton1.setBounds(new Rectangle(25, 140, 73, 22));
```

The following are the parameters for the `Rectangle` constructor:

```
new Rectangle(X, Y, width, height);
```

Where *X* and *Y* are the coordinates of the component relative to the upper-left corner of its container.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating a Menu

- Select Create Menu Bar during application creation.
- Add `JMenuBar` from the Component Palette.
- JDeveloper creates:
  - `JMenuBar` for a visual container for menus
  - `JMenu`, which represents a menu of items added to a menu bar
  - `JMenuItems`, which are placed in a `JMenu`
- Each `JMenuItem` supports events, interfaces, and handler methods in the same way as with other Swing UI components.
- `JMenuBar` can be added to any top-level container, such as frames, dialog boxes, and applets.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Menu Manually

Follow these steps to create a menu bar manually with a single menu and single item:

1. Create a `JMenuBar` object.
2. Create a `JMenu` object.
3. Create a `JMenuItem` object.
4. Add the menu item to `JMenu`.
5. Add the `JMenu` object to the `JMenuBar` object.

Add the Menu Bar to a Container

Associate the `JMenuBar` object with a frame, dialog, or applet by calling its `setJMenuBar()` method. Write menu event-handling code for `JMenuItem` by registering the appropriate event listeners or by using the Swing `Action` objects.

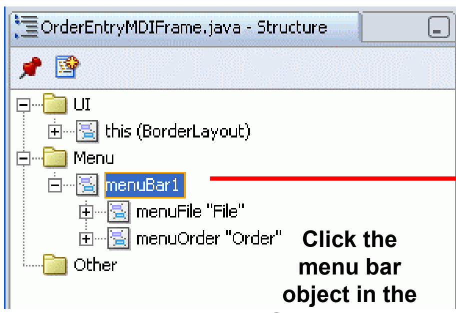
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

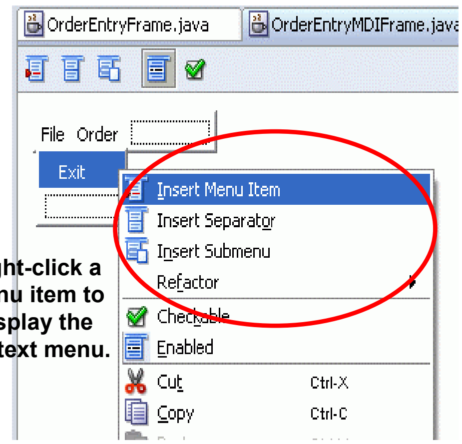
Using the JDeveloper Menu Editor

In the Structure pane of the JDeveloper Menu Editor:

1. Expand the Menu node.
2. Click the menu bar object for a visual representation.
3. Right-click menu or menu items to display context menu options.



Click the menu bar object in the Structure pane to display the menu bar.



Right-click a menu item to display the context menu.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Using the JDeveloper Menu Editor

In the UI Editor, the menu bar can be altered by adding, deleting, and moving components.

To Add a Menu or Menu Item

You can right-click a menu item to add another menu to the menu bar, as shown in the slide. If you right-click a menu item, you can use the context menu selection to:

- Add another menu item
- Add a separator
- Add a submenu
- Mark a menu item as a check box menu item
- Disable the menu item

To Delete Menu Items

Press the Delete key after selecting a menu component.

To Rearrange the Menu Structure

Drag the components visually.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-1 Overview: Adding User Interface Components

This practice covers the following topics:

- Creating the `OrderEntryMDIFrame` menu
- Adding menu items and a separator to the Order menu
- Adding components to `OrderEntryFrame` to create its visual structure

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

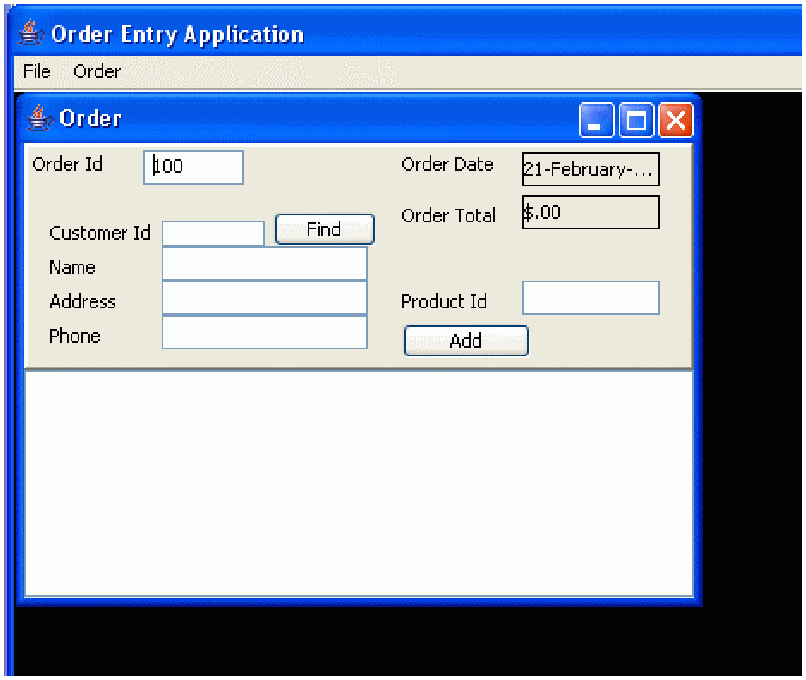
Practice 17-1 Overview: Adding User Interface Components

In this practice, you create the menu and visual components so that users can manage order entry details. The application includes a button to find the customer assigned to the order, and buttons to add and remove products as items in the order. You learn how to build a Swing-based UI application by using the JDeveloper UI Editor to construct the user interface. You also learn how to handle events for the Swing components that are added to the application.
Note: Whenever you create a UI component, JDeveloper declares it as private and you can remove that if required.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les17` directory, load the `OrderEntryApplicationLes17` application, and continue with this practice.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

UI for the Order Entry Application



ORACLE
Copyright © 2009, Oracle. All rights reserved.

UI for the Order Entry Application

The slide shows a snapshot of the final visual appearance of the application's main window, `OrderEntryMDIFrame`, and a sample `OrderEntryFrame` for an order that is created as an internal frame.

Using the Application

`OrderEntryMDIFrame` provides the main application menu, from which users select the Order > New menu option to create a new order for a customer.

The new order request should create the internal `OrderEntryFrame` and a new `Order` object (whose ID sets the Order Id text field in the frame).

You enter customer details by providing an ID value in the Customer Id field and clicking the Find button. The Find button event validates if the customer exits (by using the `DataMan.findCustomerById()` method). When the event validates, it assigns the customer to the order and displays the customer details in the fields provided; otherwise, an error message is displayed.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Event Handling Model

- How it works:
  - An event originates from a source and generates an event object.
  - An event listener hears a specific event.
  - An event handler determines what to do.
- Setting it up:
  1. Create an event source object.
  2. Create an event listener object implementing an interface with methods to handle the event object.
  3. Write an event-specific method to handle the event.
  4. Register the listener object with the event source for the specified event.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Java Event Handling Model

There are four components of the Java event handling model:

- **Event source:** The object or component from which the event comes (For example, a mouse-click event could originate from a button.)
- **Event object:** The object that is generated when the event occurs (This object is passed to an event listener.)
- **Event listener:** A method whose job is to listen for a specific event and then run an event handler when the event occurs by receiving the event object from the event source
- **Event handler:** A method whose job is to handle a specific event and event object

Interfaces involved in handling events:

- `ActionListener` (extends `EventListener`): The listener interface for receiving action events. When an action event occurs, the `ActionPerformed` method is invoked.
- `MouseListener` (extends `EventListener`): The listener interface for receiving "interesting" mouse events (press, release, click, enter, exit) on a component.

The following slides illustrate the event handling model in detail.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Event Listener Handling Code Basics

- Create the event source.

```
JButton findBtn = new JButton("Find");
```

- Create the event listener implementing the required event interface.

```
class MyListener implements ActionListener {
 public void actionPerformed(ActionEvent e) {
   // handler logic
 }
}
```

- Register the listener with the event source.

```
findBtn.addActionListener(new MyListener());
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Java Event Listener Handling Code Basics

The slide shows the key steps to create an event source object—for example, a `JButton`. When the button (event source) is pressed with an Enter key (if it has focus), or if you click the button with the mouse, the button creates the event object, a `java.awt.eventActionEvent` object. If listeners are registered with a button to listen for the `ActionEvent`, their handler method is called by the event firing mechanism of the button.

The second code example in the slide shows creating the class for the event listener that implements the `java.awt.event.ActionListener` interface, requiring that you write a single method with the following signature:

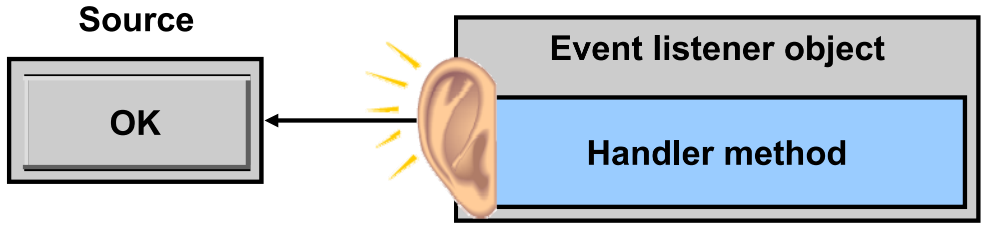
```
public void actionPerformed(ActionEvent e);
```

The `actionPerformed()` method receives an event object reference that is created by the event source (in this case, an `ActionEvent`). The handler code can optionally use the event object to get information from or find out about the event source. The body of the method effectively handles the event by implementing the code to manage the event. The event handling code executes on the Java event handling thread.

The third code example shows the final piece to the puzzle, where the listener object is created in the argument of the `addActionListener()` method, thereby registering the listener object with event source to handle the `ActionEvent` (that is, the button-clicked event).

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Event Handling Process: Registration




```
MyListener actionListenerObj = new MyListener();
public void jbInit() {
  button1.addActionListener(actionListenerObj);
  …
}
```

Copyright © 2009, Oracle. All rights reserved.

Event Handling Process: Registration

Registering the Listener

As already seen, the final stage in the event handling coding process is the registration of the listener with the event source. The event listener registers "interested" in a particular type of event. For example, "I am interested in button clicks."

Registering Listener Objects

An event listener receives events from a source only if it registers with that source as a listener for a particular type of event. For each type of event that it can generate, a source object provides a method that enables objects to register themselves as listeners for that event.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Event Handling Process: Registration (continued)

For example, consider a button. A `Button` object can generate `ActionEvents`, so the `Button` class provides a method called `addActionListener()`. The example in the slides shows how to call this method to register the listener object that was created in the previous slide. This `MyListener` object is added to a list of listeners that are informed when the `button` is clicked. This listener object contains the code to handle the event.

Note: There is a method called `removeActionListener()`, which allows a listener to be removed from the list of registered listeners, as in the following example:

```
findBtn.removeActionListener( actionListenerObj);
```

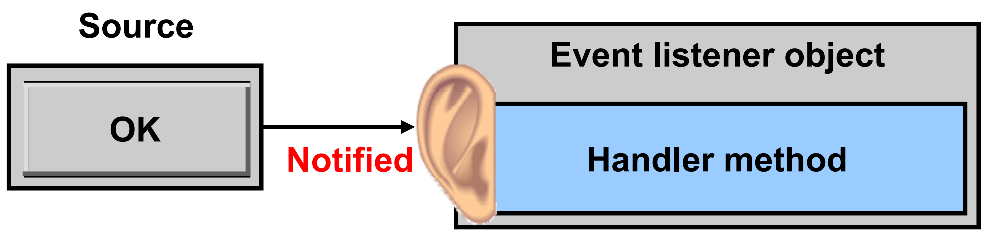
The event handling model is enforced because the classes follow coding rules as follows:

- The event object class is called *XXX*`Event`.
- The listener implements an interface called *XXX*`Listener` interface.
- The event source provides an `add`*XXX*`Listener()` method, which accepts an object argument that implements the appropriate *XXX*`Listener` interface.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Event Handling Process: The Event Occurs




```
public void jbInit() {
  button1.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    // Your code to handle the ActionEvent
   }
  }); … }
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Notifying the Listener of the Event

When the event occurs, the listener is notified that the event it is interested in has occurred. An event source notifies an event listener object by invoking a method on it and passing it an event object. Events are delivered only to registered listener objects. An object never receives unsolicited events, and events that are not handled are simply ignored.

Listening for an Event

When an event occurs, the event source notifies an event listener by calling an event-specific method on the listener; all listeners for a particular type of event must provide the appropriate method. For example, the event that occurs when a button is clicked is `ActionEvent`. All listeners for an `ActionEvent` must provide an `actionPerformed()` method, because this is what the event source will try to call.

Implementing the Event Listener as an Inner Class

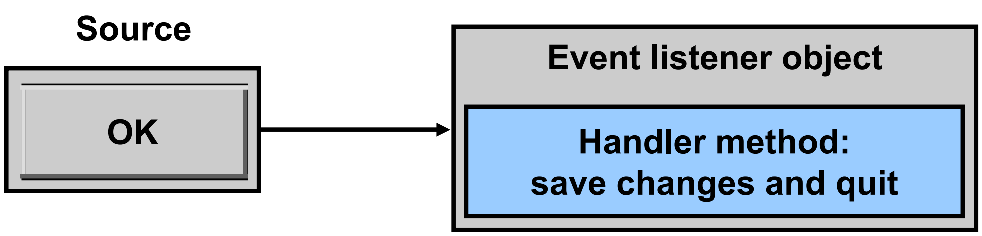
The example in the slide shows how to implement the event listener object and register it with the event source in one step. You can do this by implementing the event listener object as an anonymous inner class.

The particular type of inner class event adapters that JDeveloper generates by default are known as anonymous adapters. This implementation of anonymous adapters avoids the creation of a separate (named) adapter class. The resulting code is compact and elegant.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Event Handling Process: Running the Event Handler



Copyright © 2009, Oracle. All rights reserved.

Running the Event Handler

The event listener contains an event handler. After the event listener receives notification, it runs its event handler. For example, if the button is a Save button, the event handler saves the data on the form.

How Is This Enforced?

All listeners for an `ActionEvent` must implement the `ActionListener` interface.

The various listener interfaces specify the methods that you must implement in your listener class. The `ActionListener` interface stipulates only one method for you to implement:

```
public interface ActionListener {
public void actionPerformed(ActionEvent e);
}
```

Note that event handlers should not contain any business logic code. Instead they are used to call business logic in other classes e.g EJBs, JavaBeans etc.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using Adapter Classes for Listeners

Adapter classes are "convenience" classes that implement event listener interfaces:

- They provide empty method implementations.
- They are extended, and the desired method is overridden.

```
interface MouseListener {
  // Declares five methods
}
```

```
class MouseAdapter implements MouseListener {
  // Empty implementations of all five methods
}
```

```
public class MyListener extends MouseAdapter {
  // Override only the methods you need
}
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adapter Classes

Some of the event listener interfaces contain more than one method. For each of these interfaces, there is a simple adapter class that provides an empty body for each method in the interfaces.

For example, `MouseListener` contains five methods. If you implement `MouseListener` directly, you must implement all five of its methods, even if you are only interested in one of them. Alternatively, you can extend the `MouseAdapter` class.
If you do this, you can override the methods that you need and ignore the rest.

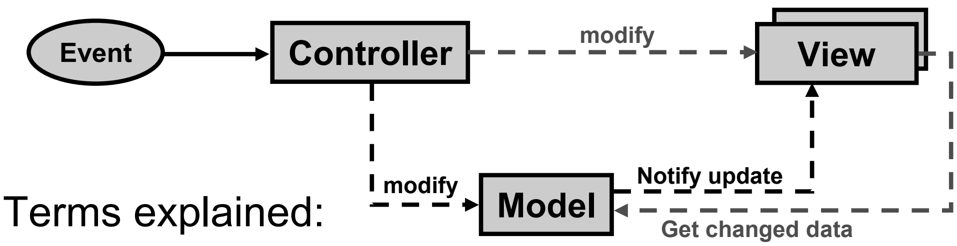
There is no adapter class for the `ActionListener` interface because the interface has only one method.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Swing Model-View-Controller Architecture

- Model-View-Controller principles



- Terms explained:
  - Model represents the data or information.
  - View provides a visual representation of the data.
  - Controller handles events modifying the view or model.
- Always update Swing components on the event thread queue, or use `SwingUtilities` methods.

Copyright © 2009, Oracle. All rights reserved.

Importance of Swing Model-View-Controller Architecture

The Model-View-Controller (MVC) design pattern forms dynamic associations among the visual representations of the data object, the data, and events. The MVC design allows multiple views to the same data (model), keeping the various views synchronized as the data is modified. This forms the foundation for creating data-aware components.

The slide shows that when an event occurs that causes changes to a visual component, the view requests the appropriate data from the model. If the event causes a change to the model, the model notifies the view that a change occurred and the view in turn makes a request for the changed data from the model.

In Swing terms, a UI component represents a view. Each component passes an event object to a "listener" object or controller to handle the event. Depending on the event, the controller modifies the view or model. If the model class is one of those provided by the Swing API that is suitable for the associated UI component, changes to the model are automatically visible through the view.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Importance of Swing Model-View-Controller Architecture (continued)

Using a `JList` with a `Vector` is possible, but adding elements to the `Vector` does not update the `JList` contents for two reasons:

- The `Vector` is not the appropriate model class for a `JList`.
- The `JList` creates an internal `ListModel` from the vector items.

However, if you explicitly create a `DefaultListModel` object and associate it with a `JList`, the `JList` will reflect the changes as items are added to the `DefaultListModel` object.

Note: The diagram in the slide represents classic MVC. Swing components use a modified form of MVC to support a pluggable look and feel.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Importance of Swing Model-View-Controller Architecture (continued)

Swing Components (View) and Model Classes

The following table shows the Swing components and interfaces that can be implemented by model classes to provide MVC functionality. Each interface has been implemented by a class in the Swing API packages; the implementing classes are also shown.

Swing components	**Model interface**	**Class implementing model interface**
JTextField, JPasswordField, JTextArea, JTextPane, JEditorPane	Document (found in javax.swing.text package).	PlainDocument, and DefaultStyledDocument
JButton, JCheckBox, JCheckBoxMenuItem, JMenu, JMenuItem, JRadioButton, JRadioButtonMenuItem, JToggleButton	ButtonModel	DefaultButtonModel
JComboBox	ComboBoxModel	DefaultComboBoxModel
JProgessBar, JScrollBar, JSlider	BoundedRangeModel	DefaultBoundedRangeModel
JList	ListModel, ListSelectionModel	DefaultListModel, DefaultListSelectionModel
JTable	TableModel (found in javax.swing.table package) ListSelectionModel	DefaultTableModel, DefaultListSelectionModel
JTree	TreeModel, TreeSelectionModel (both interfaces found in javax.swing.table package)	DefaultTreeModel, DefaultTreeSelectionModel

Swing components, by default, implicitly create and store data in a default model that suits their requirement. You can use all components to explicitly create and use an appropriate model, usually based on those shown in the preceding table.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Basic Text Component Methods

- Text item (`JLabel`, `JTextField`, and `JButton`) methods:

```
void setText(String value)
String getText()
```

- Additional methods in `JTextArea`:

```
void append(String value)
void insert(String value, int pos)
```

- Changes to component contents are usually made in the event handling thread.

Note: Consult the Java API documentation for details about each component's capabilities.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Getting and Setting Properties

The various Swing components have different methods to populate them with values and to retrieve the values contained in the component. In general, most text items, such as labels, text fields, and text areas, have a `setText(String value)` method that sets the contents to the specified string value, or a `String getText()` method that returns the contents as a `String` object.

Note: You must use the `setText()` and `getText()` methods to change or get the label of a `JButton` object, respectively. Do not use the `setLabel()` and `getLabel()` methods that are now deprecated.

`JTextArea` objects are multiline text items and thus enable you to append to the existing contents, insert at a particular position in the text, or replace text. See the Java API documentation for information about the methods.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Basic `JList` Component Methods

Subset of `JList` component methods include:

- `void setListData(Vector)`
  - Copies `Vector` to a `ListModel` applied with `setModel`
- `void setModel(ListModel)`
  - Sets model representing the data and clears selection. Uses `DefaultListModel` class for the model
- `Object getSelectedValue()`
  - Returns the selected object, or returns `null` if nothing is selected
- `int getSelectedIndex()`
  - Returns the index of the selected item, or returns `-1` if nothing is selected

ORACLE

Copyright © 2009, Oracle. All rights reserved.

List Components

The `JList` and `JComboBox` are Swing components that handle lists of data. This slide discusses some of the methods that are provided by the `JList` class.

```
Vector vector = new Vector();
Jlist list = new JList(vector);
```

For example, if you create a `JList` object with the constructor accepting a vector, the vector elements are copied to an internally created `DefaultListModel` object. Thus, if you add elements to the vector, with the `addElement()` method, the new elements will *not* be visible through the `Jlist` unless you call the `JList.setListData()` method passing the updated vector object as an argument. This is inefficient because the elements are copied from the vector again and a new `DefaultListModel` object is created internally.

It is better to create the `DefaultListModel` object first, use it as you would use a `Vector`, and create the `JList` with the appropriate constructor. Here is an example:

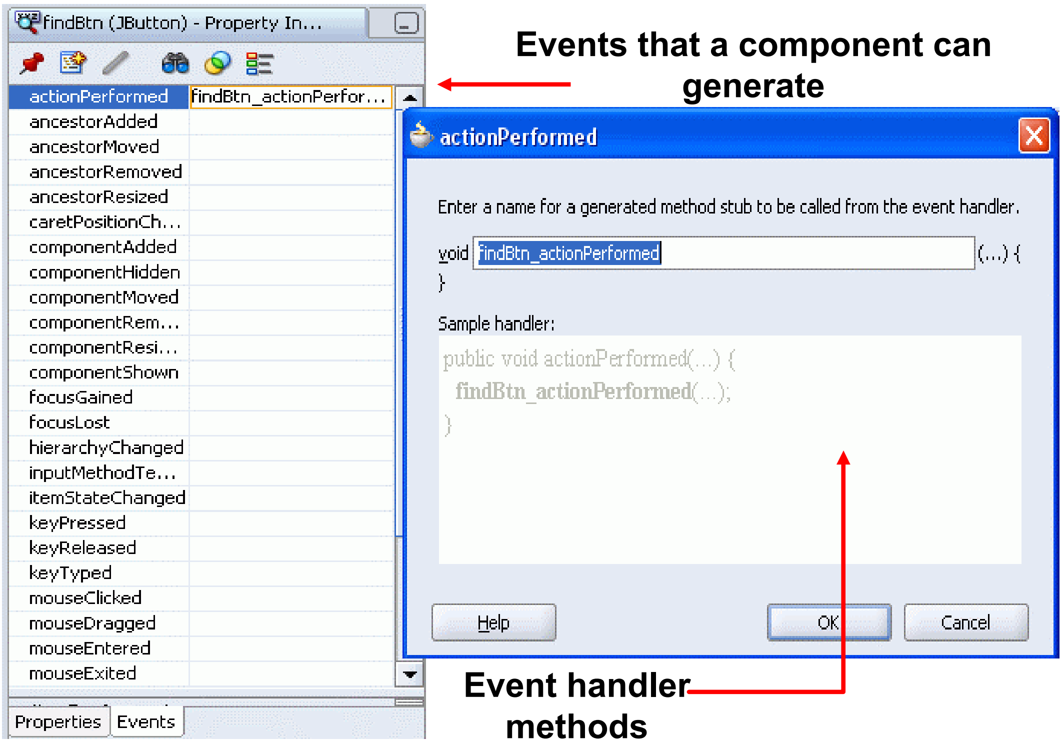
```
DefaultListModel model = new DefaultListModel();
Jlist list = new JList(model);
```

Or you can call the `setModel()` method. As elements are added to the `DefaultListModel` object by using the `addElement()` method, they automatically appear in the `JList` visual display.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

What Events Can a Component Generate?



Copyright © 2009, Oracle. All rights reserved.

Each Type of Component Generates Different Events

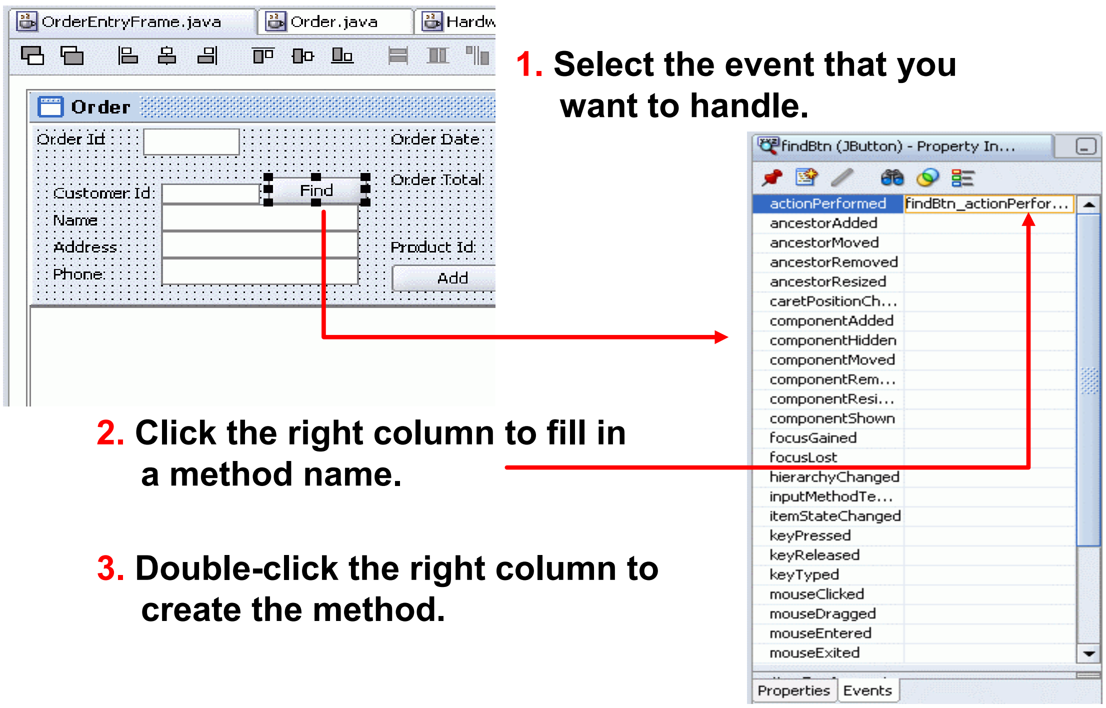
To find out what events a component is capable of generating with JDeveloper, select the component in the Designer pane and click the Events tab in the Inspector window. This shows a list of all the events that the component can generate.

For each event, the Inspector also shows whether an event handler method has been installed yet. In the slide, the Inspector shows all the events for the `JButton` component, `findButton`. No event handlers have yet been installed, so the Inspector does not have any event handler methods to advertise in the list.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Defining an Event Handler in JDeveloper



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Defining an Event Handler in JDeveloper

JDeveloper makes it easy to define event handler methods in your code:

- In the UI Editor, select the component for which you want to provide an event handler.
- In the Inspector, the Events tab shows a list of all the events that the component can generate. Click the event that you want to handle.
- Click the right column for that event; the Inspector suggests a name for the event handler method that it is about to generate. In the example, the event handler method is called `findButton_actionPerformed`.
- Click the "…" button, which suggests a name for the event handler method in the dialog box. JDeveloper then generates the event handler method in your code.

Note that double-clicking the button in the UI Editor creates the listener and handler. It is an alternative to clicking the Events tab of the Inspector and double-clicking the name of the listener.

`actionPerformed` Event

Many UI components have a special event called `actionPerformed`. For most components, `actionPerformed` is the most commonly used event. For example, a `JButton` generates an `actionPerformed` event when it is clicked, whereas a `JList` generates an `actionPerformed` event when it is double-clicked. Use `actionPerformed` when possible, rather than an event such as `mouseClicked`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Default Event Handling Code Style Generated by JDeveloper

```
public void jbInit() throws Exception
…
findButton.addActionListener(
new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  findButton_actionPerformed(e);
 }
}); …
```


```
void findButton_actionPerformed(ActionEvent e) {
 // Your code to handle the ActionEvent
}
```

Copyright © 2009, Oracle. All rights reserved.

What Happens When the Event Is Fired?

- When the button is clicked, it examines its list of registered listener objects and calls the `actionPerformed()` method on each listener object. One of the listener objects is the new (nameless) `ActionListener` object, created and registered in the applet's `jbInit()` method.
- The listener object's implementation of `actionPerformed()` calls the handler method.

By default, JDeveloper uses anonymous inner classes in the event handling code that it generates, but you can configure JDeveloper to create a separate listener class instead, called a Standard Adapter style.

The procedure for selecting this code style option is as follows:

- Select Tools > Preferences.
- Select the Java Visual Editor node. Then, in the Event Settings pane, choose one of the following:
  - Anonymous Inner Class button
  - Standard Adapter button.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Completing the Event Handler Method

```
public class JFrame1 extends JFrame {
 …
 void findButton_actionPerformed(ActionEvent e){
  // When the user clicks the button, display
  // the list of customers in jTextArea1
 String findList = ("Supplies-R-Us " + "\n" +
                     "Consulting Inc. " + "\n" +
                     "Just-In-Time Training ");
 jTextArea1.setText(findList);
 }
}
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Completing the Event Handler Method

When you add an event handler method in JDeveloper, it defines a skeleton method in your program and also generates the event listener code to make sure that the method is called when the event occurs.

Using Methods in the `Button` and `Label` Classes

The event handler that is shown in the slide is called when `findButton` is clicked. When that happens, the event handler method performs the following tasks:

- Constructs a string that contains a list of customers. In a real application, the string would be built by a call to a method that returns a string. That method could retrieve the data from a database, a file, or another source.
- Calls `JTextArea1.setText()` to set the `text` property of the form's text area to the string list of customers

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this lesson, you should have learned how to:

- Add a Swing component to a visual container
- Get and modify the contents of the components
- Use the AWT event handling model to:
  - Create an event source
  - Create an event listener and handler code
  - Register an event listener to handle the event
- Create a menu bar with menus and menu items
- Handle events

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-2 Overview: Adding Event Handling

This practice covers adding event handling for:

- The Order > New menu
- The Find Customer button
- The Add Product and Remove Product buttons

Copyright © 2009, Oracle. All rights reserved.

Practice 17-2 Overview: Adding Event Handling

In this practice, you create the order entry details. You add event handling code for the Order > New menu, the Find Customer button, and the Add Product and Remove Product buttons.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you would like to move on to this practice, change to the `les17-2` directory, load the `OrderEntryApplicationLes17-2` application, and continue with this practice.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

18

Deploying Java Applications

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Objectives

After completing this lesson, you should be able to do the following:

- Package programs in `.jar` files
- Describe the benefits of using Java Web Start
- Deploy an application using Java Web Start

Copyright © 2009, Oracle. All rights reserved.

Objectives

Once you have built your application, you will want to deploy it. This lesson discusses the options available for deploying Java applications.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Packaging and Deploying Java Projects

- Java supports an archive file that can be used to group all project files in a compressed file.
- This single file can be deployed on an end user's machine as an application.
- It can also be downloaded to a browser in a single HTTP transaction.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Packaging and Deploying Java Projects

A project may consist of many classes and supporting files, such as image files and audio files. To make your programs run on the end-user side, you need to provide end users with all these files. For convenience, Java supports an archive file that can be used to group all the project files in a compressed file.

Archiving makes it possible for Java applications, applets, and their requisite components (`.class` files, images, and sounds) to be transported in a single file, which can be deployed on an end user's machine as an application. You can also download it to a browser in a single HTTP transaction, rather than opening a new connection for each piece. This greatly simplifies application deployment and improves the speed with which an applet can be loaded onto a Web page and begin functioning.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Deploying a `.jar` File

- You can make your simple archive into an executable `.jar` file that you can launch with the `java` command.
- Before deploying an executable `.jar` file, you must first create a deployment profile.
- Deployment profiles are named sets of properties stored as part of the application or project's properties that govern the deployment of a project or application.
- A deployment profile specifies the format and contents of the archive file that will be created.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Deploying a `.jar` File

Deployment profiles are application or project properties that govern the deployment of a project or application. A deployment profile names the source files, deployment descriptors, and other auxiliary files that will be packages; the type and name of the archive file to be created; dependency information; platform-specific instructions; and other information. JDeveloper provides a wizard to help you create the deployment profile.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Deploying Applications with JDeveloper

The JDeveloper Deployment Profile wizard:

- Detects interclass dependencies
- Creates `.ear`, `.war`, `.jar`, or `.zip` files
- Enables you to have control over other files added to the deployed archive
- Enables you to save deployment profile settings in project files:
  - That simplify redeployment when code changes
  - That can be automatically updated with new classes as they are added to the project

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Deploying Applications with JDeveloper

Oracle JDeveloper provides a deployment wizard that helps with most of the tedious tasks associated with deploying your application.

The wizard detects classes that are used in your application and proposes that they are included in the archive file it creates for you. It also allows you to specify rules for the inclusion of files into the archive. These rules provide an easy way to control which file types are automatically selected for inclusion.

The wizard also provides a dialog box for you to manually add other files that were not detected by the wizard. The wizard can detect dependencies only between classes recognized at compile time. It does not recognize required resource files, such as image and sound files, or dynamically loaded classes. These files are proposed by the wizard only if they are included in the project.

Any settings you choose or set with the deployment wizard are maintained in profiles and are accessible any time you run the wizard.

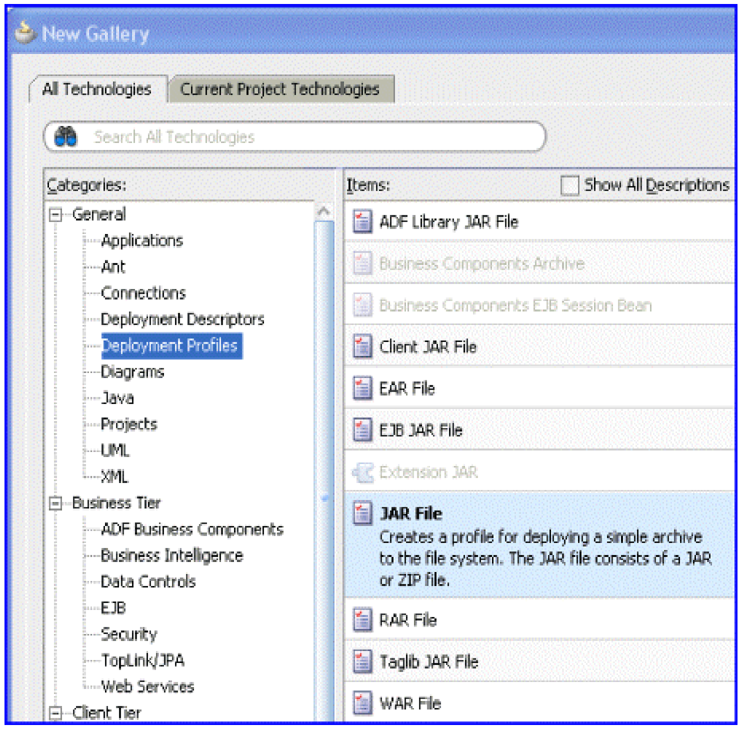
It is a good idea to rebuild the project before deploying your application.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating the Deployment Profile

1. Select File > New.
2. In the New Gallery, select Deployment Profiles in the General category, and JAR File in the Items pane.
3. Click OK.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating the Deployment Profile

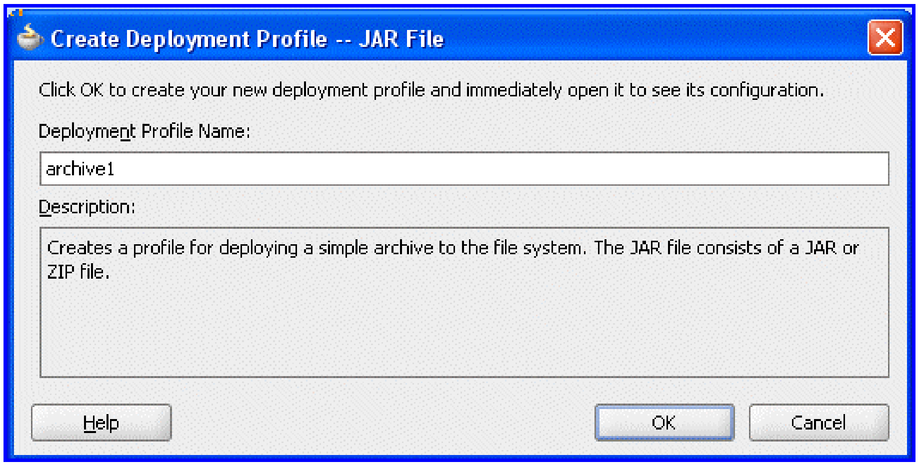
1. In the Application Navigator, select the application or project for which you want to create a profile**.**
2. Select **File** > **New** to open the New Gallery.
3. In the Categories tree, expand General and select Deployment Profiles. In the Items pane, select **JAR File.**
4. Click **OK**.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating the Deployment Profile

Name the deployment profile.



Copyright © 2009, Oracle. All rights reserved.

Deployment Profile Wizard: Create Deployment Profile – JAR File

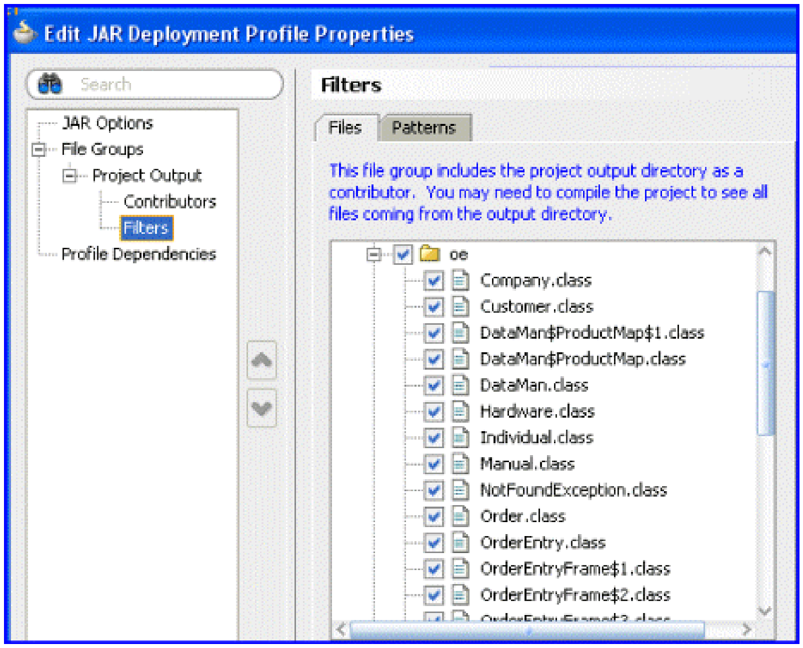
5. In the Create Deployment Profile dialog, enter the name for the profile, and click **OK.**

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Selecting Files to Deploy

Select the file types to include. This process is called *Configuring.*



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Selecting Project Files to Deploy

6. Select the files that you wish to include in your deployment profile. Click **OK** to close the dialog and create the deployment profile.
   - The process of assembling an archive file from its component files is called *configuring*. Configuring is specified in the File Groups branch of deployment profile properties. The File Groups branch consists of a list of file groups, each specifying some components. The packaged archive will be the union of all the file groups. The order of the file groups resolves name collisions: if two files have the same name, the one from the file group higher in the list is included, and the one from the lower file group is omitted.
   - A newly created deployment profile will include one or more predefined file groups. You can add, delete, or edit file groups.
   - File groups are defined by a set of *contributors*, pruned by a set of *filters*. Contributors are source files, `.jar` files, and directories that are selected for inclusion. Filters are rules that are applied to the contributors or contributors component subdirectories and files to identify the set that will be packaged.

   There are three kinds of file groups:
   - The **Packaging** file group type allows you to select contributors, project directories and other directories, and .jar files and filters. The file group mechanism is flexible and transparent, and is suitable for most projects.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

Selecting Project Files to Deploy (continued)

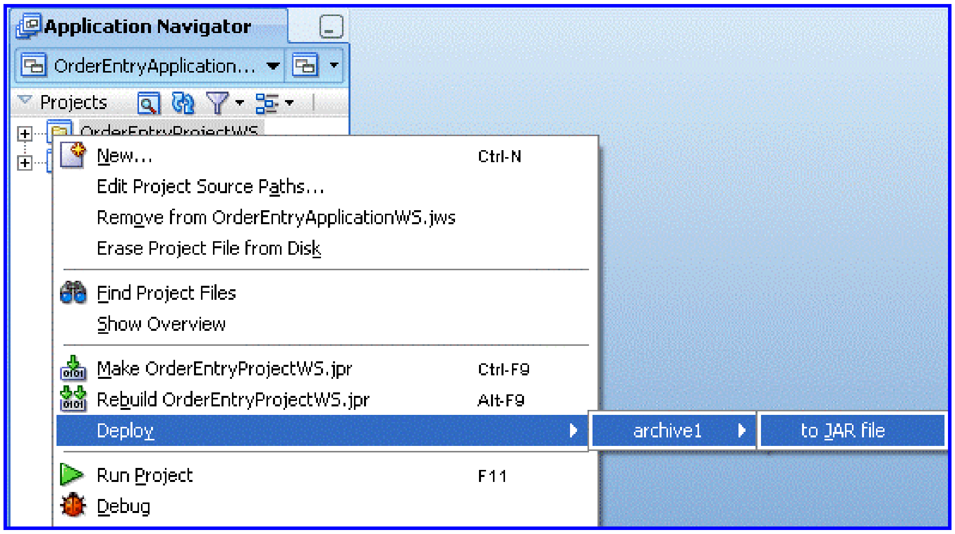
- The **Dependency analysis** file group type allows you to select contributors that are project files and their dependencies. The dependency analysis is the packaging mechanism provided in JDeveloper prior to 9.0.5.1. Profiles created in 9.0.5.1 or subsequent releases will not contain a Dependency Analysis file group by default**.**
- The **Libraries** file group type allows you to select contributors that are project libraries. A libraries file group is created for WAR deployment profiles. Libraries file groups are useful in other projects that need to repackage existing `.jar` files.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating and Deploying the Archive File

1. Right-click the project name.
2. Select Deploy > [profile name] > to JAR file.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Deploying the `.jar` File

1. Right-click the project in which you created the deployment profile and select **Deploy** > **[profile name]** > **to JAR file** from the context menu.
2. The Java archive is placed in the directory listed in the JAR Options, JAR File properties of the archive.

Editing the Profile File

To edit the simple archive, double-click the project name to invoke Project Properties. Select Deployment in the list on the left, and in the Deployment Profiles pane, select the profile you wish to edit, and click the Edit button. The Jar File field in the JAR Options window indicates the location of the archive file.

Deploying an Application or WAR file to WebLogic Server

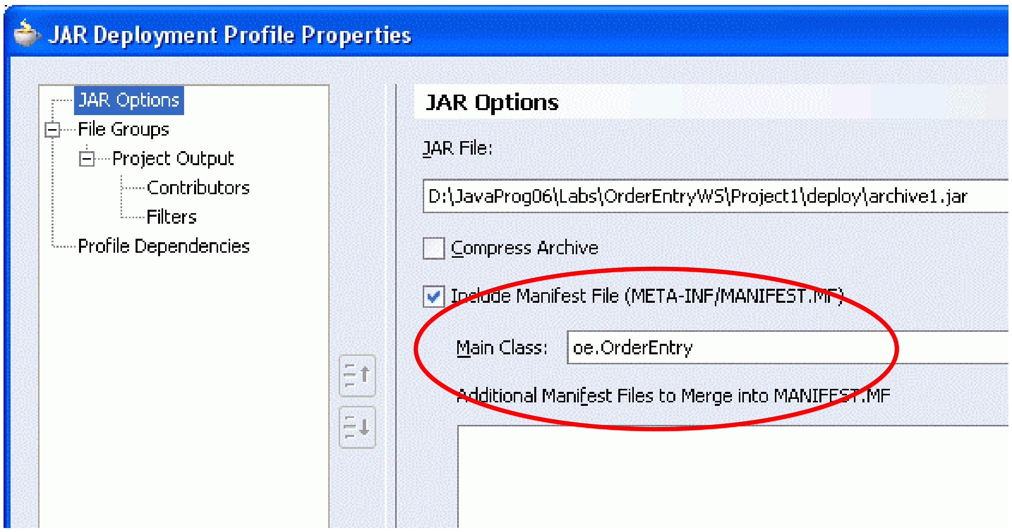
If you need to deploy the application to an application server, such as WebLogic Server, you must use the Oracle Connection Manager to form a connection with the application server. The instructor and practices provide you with the instructions on how to accomplish this task.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Executable `.jar` File

Set the Main Class field to the class name containing a `main()` method in JAR Options.



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Creating an Executable `.jar` File

You can make your simple archive into an executable JAR file that you can launch with the `java` command:

1. Right-click the project in the Application Navigator and select **Project Properties**.
2. Select the name of the profile in the Deployment section of the Project Properties dialog and click **Edit**.
3. Click **JAR Options** in the tree.
4. Select **Include Manifest File**. The manifest is a special file that contains information about the files packaged in a `.jar` file.
5. In the **Main Class** field, enter the fully qualified name of the application class without the .class extension for the class containing the main()method that you invoke when using the java –jar command-line option. Specifying the Main Class main attribute is the only way to make an executable `.jar` file.
6. Click **OK**.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Creating an Executable `.jar` File (continued)

Example

If the generated JAR archive file is called `OrderEntry.jar` , execute the main application class as follows:

```
java -jar OrderEntry.jar
```

If you do not enter a name in the Main Class field for the JAR archive, you can still execute any class with a `main()` method contained in the JAR file from the command line as follows:

Set the `CLASSPATH` to include the Java Archive file named

```
java package.ClassName
```

Or use the `-classpath` command-line option as follows:

```
java -classpath {archivefilename}.jar package.ClassName
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Java Web Start

- Is an application-deployment technology based on the Java 2 platform.
- Launches full-featured applications via any browser on any platform, from anywhere on the Web.

ORACLE
Copyright © 2009, Oracle. All rights reserved.

What Is Java Web Start?

Java Web Start is an application deployment technology created by Sun Microsystems, Inc. You can use Java Web Start to launch full-featured applications via any browser on any platform, from anywhere on the Web. After your application is launched and is running, the browser can be closed and your application will continue to run. Because Java Web Start applications are not tied to the Web browser, one benefit is that you can keep your old applications without having to trade them for a version based on an HTML interface that runs in a Web browser.

JDeveloper supports the creation of the XML-based JNLP (Java Network Launching Protocol) definition on which the Java Web Start technology is based.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Advantages of Web Start

- Is as easy to deploy as HTML
- Launches applications from the Start menu on the desktop
- Does not require browser to be running
- Allows applications to work offline
- Automatically updates applications when invoked

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

Advantages of Web Start

Java Web Start Features

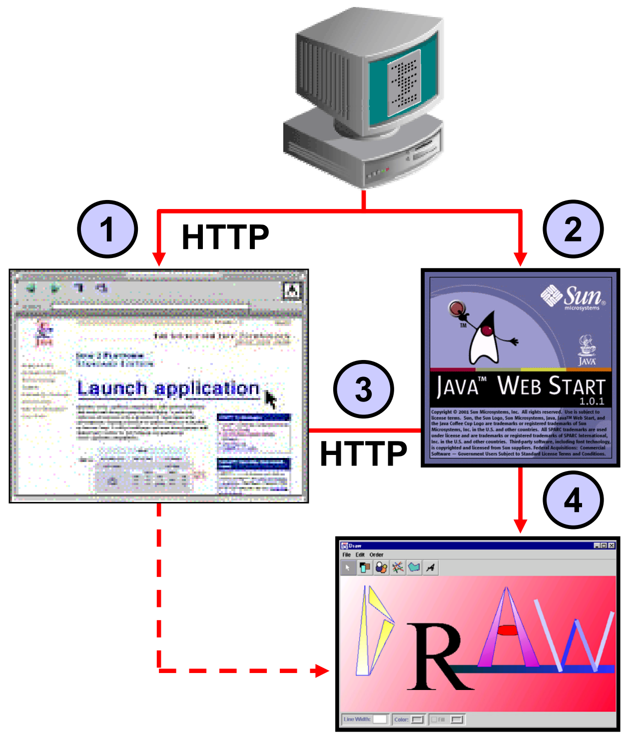
Web Start is as easy to deploy as HTML with the richness of full-featured GUI applications. Applications can be launched using a browser, on any type of platform, from anywhere on an intranet or the Internet. A small, one-time download of the application is required on initial launching. Subsequent access is provided from the local cache and the application launches more quickly. It works like a browser plug-in, similar to RealAudio. With the Java Web Start software installed once on the user's machine, individual users can run applications and applets simply by clicking a Web page link. If the application is not present on their computer, Java Web Start downloads all necessary files from the Web server where the application libraries reside. It then caches the files on the client computer so the application is always ready to be relaunched anytime either from an icon on your desktop or from the browser link. The most current version of the application is always presented to the user since Java Web Start performs updates as needed.

For more information on Java Web Start, go to this Sun Microsystems Web page: http://java.sun.com/products/javawebstart/

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Running a Web Start Application

1. Request the application.
2. Launch Web Start on the local machine.
3. Download the application.
4. Launch the application (Draw).



ORACLE
Copyright © 2009, Oracle. All rights reserved.

Java Web Start Architecture: Overview

The Java Web Start software must be installed on your machine before you can launch an application.

1. When you click a download link, the link instructs the browser to invoke Java Web Start. A JNLP file runs the application.
2. Java Web Start technology queries the Web to determine whether all the resources that are needed for the application are already loaded. If they are, and if the most recent version of the application is present, the application will be launched.
3. The application is launched.

If the application is not present on your computer, Java Web Start automatically downloads all the necessary files from the Web server where the application libraries reside. These files are cached on the client machine so that the application is always ready to be relaunched any time, either from an icon on your desktop or from the browser link. The most current version of the application is always presented to you because Java Web Start performs updates as needed.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Examining the JNLP File

The JNLP file defines:

- The location of the application resources
- Information that appears while the application loads
- What the application resources are

OrderEntryWS.jnlp

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<jnlp spec="1.0+" codebase="http://127.0.0.1:7101" href="/OrderEntry
  <information>
    <title>Kate's App</title>
    <vendor>My Company</vendor>
    <homepage href="OrderEntryWS.html"/>
    <description>My application description</description>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="1.4+"/>
    <jar href="/OrderEntryApplicationWS-Project1-context-root/apps,
  </resources>
  <application-desc main-class="oe.OrderEntry"/>
</jnlp>
```

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Using the JNLP File

Java Web Start technology enables Web deployment by using existing Internet protocols: applications are launched when a client accesses (typically, by clicking a link in an HTML page) a special launch file with a `.jnlp` file name extension.

The Web Start technology is built on the JNLP API, which provides services that enable applications to obtain information not normally available using the Java 2, Standard Edition (Java SE) platform API. These services include accessing the system clipboard, controlling resource caching, importing files from a local disk, and so on.

The task of packaging for deployment is where the JNLP comes into play. In addition to a JAR file for the application classes, JNLP requires that you create a descriptor file on how to start up the application.

JNLP also requires that you provide the location of the application resources, what information must be displayed in the window that appears while the application loads, and what the application resources are.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using JDeveloper to Deploy an Application for Java Web Start

Step 1: Generate deployment profiles and archive the application.

Step 2: Start the WebLogic server.

Step 3: Use Web Start Wizard to create a JNLP file.

Step 4: Archive and deploy your application to the WebLogic server.

Copyright © 2009, Oracle. All rights reserved.

How to Deploy a Java Web Application for Java Web Start

You can use JDeveloper's simple Web deployment process to set up the Web server before downloading and running the application using Java Web Start. The process of deploying is similar whether you intend to deploy to the JDeveloper's integrated WebLogic Server or a production server.

First, you create a deployment archive of your Java application using the deployment wizard. The result is the creation of a `.jar` file that contains all your application files.

Second, you start the application server that is used to deliver the application. In JDeveloper, select Run > Start Server Instance.

Third, you use the wizard in JDeveloper to create a JNLP file. It is a good idea to store all the deployment files (HTML, JNLP, XML) in a separate JDeveloper project.

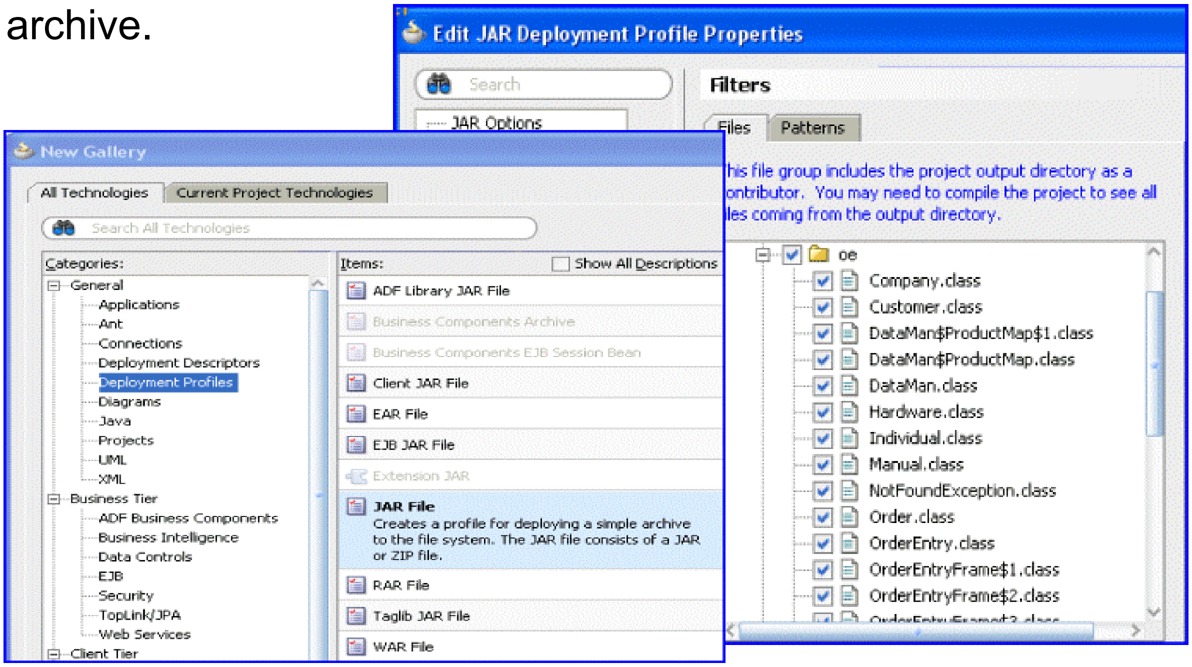
Finally, create a Web deployment file containing deployment-specific information and the appropriate deployment descriptor.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 1: Generate Deployment Profiles and Application Archive

Package all the Java application files into a simple `.jar` archive.



Copyright © 2009, Oracle. All rights reserved.

Step 1: Generate Deployment Profiles and Archive Application

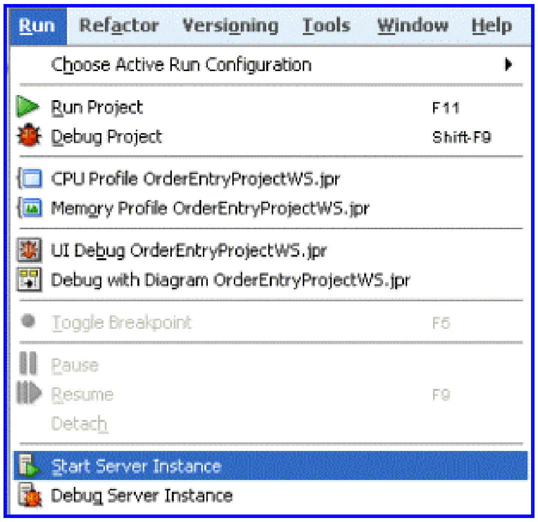
Follow the steps described earlier in this lesson to create the deployment profile and the `.jar` file for your Java application.

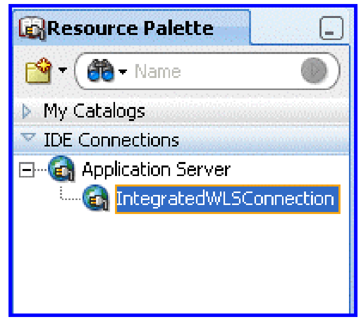
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 2a: Start the Server

Select Run > Start Server Instance to start WebLogic Server.





A connection to the server is automatically created in the Resource Palette.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Step 2a: Start the Server

Oracle JDeveloper (11*g*) is packaged with WebLogic Server 10.3. You select Run > Start Server Instance to start WebLogic Server. Once the server is running, a reference to the server connection is displayed in the Resource Palette

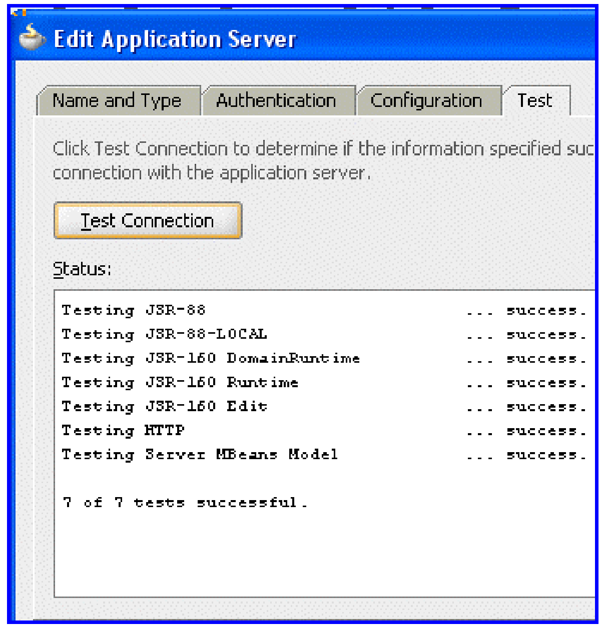
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 2b: Test the Connection

To view details of the connection and test it:

- Right-click the connection name and select Properties.
- Check the automatically generated properties, and then click the Test tab.
- Click the Test Connection button.



Copyright © 2009, Oracle. All rights reserved.

Step 2b: Test the Connection

Right-clicking the automatically generated server connection and selecting Properties from the context menu allows you both to view the properties for the application server connection and to test it.

The Edit Application Server dialog has three tabs. Click each of the tabs to see the automatically generated properties of the server connection. Click the Test tab, and then the Test Connection button. The Status box displays the results of the tests.

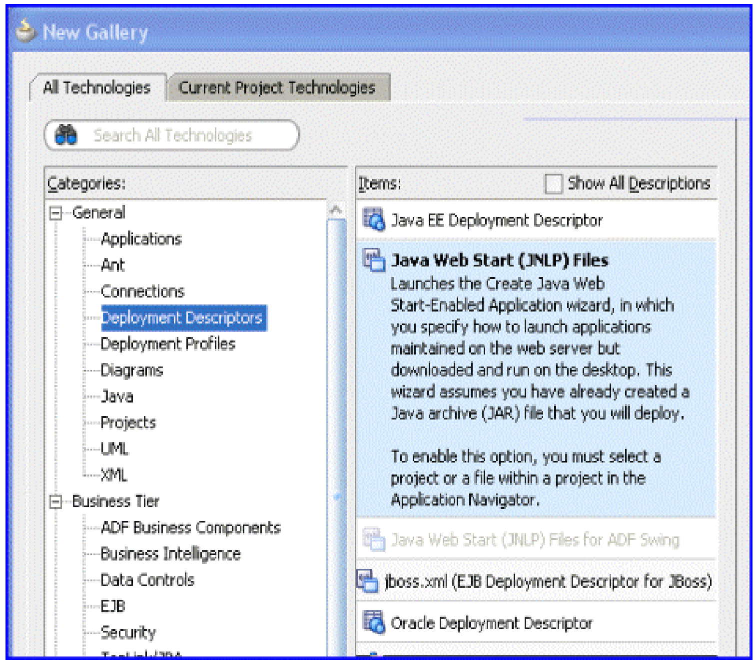
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 3: Use the Web Start Wizard to Create a JNLP File

To invoke the Web Start wizard:

- Select Deployment Descriptors > Java Web Start (JNLP) Files.



Copyright © 2009, Oracle. All rights reserved.

Step 3: Use Web Start Wizard to Create a JNLP file

You use the Java Web Start wizard to create an XML-based JNLP definition, which the Java Web Start software uses to download and run the application on the client machine.

To invoke the wizard, select Deployment Descriptors > Java WebStart (JNLP) Files.

The pages of the wizard enable you to include information to be displayed to the user while downloading (for example, application title, vendor, and brief description).

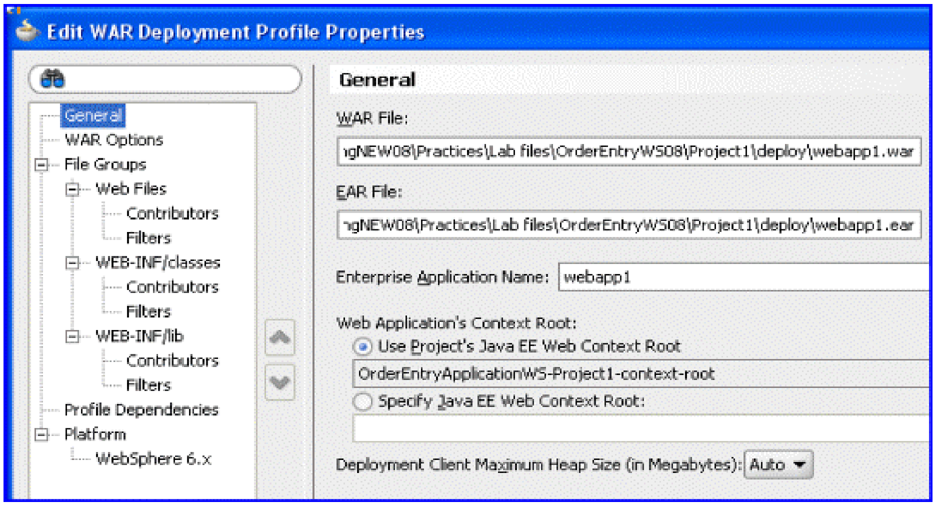
The wizard creates the complete JNLP file and an optional HTML file to launch your Web Start application.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Step 4: Archive and Deploy the Application to the WebLogic Server

- Specify properties of the Web components and deployment description.



- Deploy to the server connection described in step 2.
- Run the generated HTML file.

Copyright © 2009, Oracle. All rights reserved.

Step 4: Archive and Deploy the Application to the WebLogic Server

Create a Web Application Archive (.war) file to deploy to the Web server. It contains the contents of the `public_html` directory in your JDeveloper `mywork` folder, including the `.jar`,`.html`, and `.jnlp` files.

To deploy the `.war` file:

- Right-click the project containing your deployment files and from the context menu, select Deploy > [.war deployment profile name] > to > [server connection], for example, Deploy > webapp1 > to > IntegratedWLSConnection.

Run the generated HTML file.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Summary

In this module, you should have learned how to:

- Create an executable `.jar` file containing your application
- Describe the role of Java Web Start in deployment and outline the benefits of using it
- Describe how a Java Web Start application runs
- Use JDeveloper to deploy an application using Java Web Start

ORACLE
Copyright © 2009, Oracle. All rights reserved.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Appendix A: Practices

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Table of Contents

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 1: *Introducing the Java and Oracle Platforms*

There is no practice for this lesson.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 2: *Basic Java Syntax and Coding Conventions*

Goal

The goal of this practice is to create, examine, and understand Java code. You start by editing and running a very simple Java application. In addition, in this practice you become acquainted with the Order Entry application, which you will use throughout this course. You will use a UML model of the Order Entry application as a guide to creating additional class files for it, and you will run some simple Java applications, fixing any errors that occur.

The practices in this and the next two lessons are written to help you understand the syntax and structure of Java. Their sole purpose is to instruct rather than to reflect any set of best practices for application development. The goals of the practices from Lesson 5 to the end of the course are different. Starting in Lesson 5, you use JDeveloper to build an application by using techniques you would use during real-world development. The practices continue to support the technical material presented in the lesson while incorporating some best practices that you use while developing a Java application.

Your Assignment

In this practice you edit and run a very simple Java application. You then start to get familiar with the Order Entry application.

Editing and Running a Simple Java Application

Note: If you close a DOS window or change the location of the `.class` files, you must set the `CLASSPATH` variable *again*.

1. Open a DOS window, navigate to the **`C:\labs\D53983GC11\temp`** directory (or the location specified by your instructor), and create a file called **`HelloWorld.java`** using **Notepad** with the following commands:

```
cd \labs\D53983GC11\temp

notepad HelloWorld.java
```

2. In **Notepad**, enter the following code, placing your name in the comments (after the double slashes). Also, ensure that the case of the code text after the comments is preserved (remember that Java is case-sensitive):

```
// File:   HelloWorld.java
// Author: <Enter Your Name>
   public class HelloWorld {
     public static void main(String[] args)
     {
```


THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

```
            System.out.println("Hello World!");
        }
    }
```

3. Save the file to the C:**labs\D53983GC11\temp** directory by using the File > Save menu option, but keep Notepad running in case of compilation errors that require you to edit the source to make corrections.
4. Compile the HelloWorld.java file (file name capitalization is important).
   a. In the DOS window, ensure that the current directory is **C:\labs\D53983GC11\temp** (or the directory specified by your instructor) and that the **PATH** system variable references **JDeveloper\jdk\bin**.
   b. Check that the Java source file is saved to disk. (**Hint:** Enter the command **dir Hello***.)
   c. Compile the file using the command **javac HelloWorld.java**.
   d. Name the file that is created if you successfully compiled the code. (**Hint:** Enter the command **dir Hello***.)
5. Run the HelloWorld application (Again, remember that capitalization is important.), and examine the results.
   a. Run the file using the command java HelloWorld.
   b. What is displayed in the DOS window?
6. Modify the CLASSPATH session variable to use the directory where the **.class** file is stored. In the DOS window, use the **set CLASSPATH=C:\labs\D53983GC11\temp** command to set the variable. The variable will be set for the duration of the DOS session. If you open another DOS window, you must set the CLASSPATH variable again.
7. Run the HelloWorld application again.
   a. Use the command java HelloWorld.
   b. What is displayed in the DOS window?
8. Close Notepad but do *not* exit the DOS window because you continue to work with this environment in the following practice exercises.

Creating Order Entry Class Files (Examining the Customer Class)

The practices throughout this course are based on the Order Entry application. Turn to the end of Lesson 2 in the student guide to see the UML model of the classes in the Order Entry application.

In this practice you examine some of the class files used in the application.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 2: Basic Java Syntax and Coding Conventions (continued)

1. Copy the **`Customer.java`** file from the `c:\labs\D53983GC11` directory to your **`C:\labs\D53983GC11\OrderEntry\src\oe`** directory.
2. In the **DOS window**, change your current working directory to: **`C:\labs\D53983GC11\OrderEntry\src\oe`**.
3. Using Notepad, review the `Customer` class and provide answers to the following:
   a. Name all of the instance variables in `Customer`.
   b. How many instance methods are there in `Customer`?
   c. What is the return type of the method that gets the customer's name?
   d. What is the access modifier for the class?
4. Close the file and at the DOS prompt, compile the `Customer.java` file using the following command as a guide:

```
javac -d C:\labs\D53983GC11\OrderEntry\classes
Customer.java
```

   Where is the compiled `.class` file created?

   (**Hint:** Enter **`cd ..\..\classes\oe`**, and then type **`dir`**.)

Incorporating `Order.java` into your Application Files

Add the `Order.java` file to your application structure, review the code, and compile it.

1. In Notepad, open the `\labs\D53983GC11\Order.java` file and save it to the directory for your OE package source code (`C:\labs\D53983GC11\OrderEntry\src\oe` or the directory specified by your instructor). The attributes are different from those in the UML model. The customer and item information are incorporated later.
2. Notice that two additional attributes (getters and setters) have been added:
   - shipmode (String) is used to calculate shipping costs.
   - status (String) is used to determine the order's place in the order fulfillment process.
3. Ensure that you are in the `C:\labs\D53983GC11\OrderEntry\src\oe` directory. Use the following command to compile the Order.java file, which places the .class file in the directory with the compiled version of the `Customer` class:

```
javac –d C:\labs\D53983GC11\OrderEntry\classes Order.java
```

Creating and Compiling the Application Class with a `main()` Method

1. Create a file called `OrderEntry.java` containing the main method as follows. Place the source file in the source directory that contains the `java` files (`C:\labs\D53983GC11\OrderEntry\src\oe`). This file is a skeleton that is used for launching the course application. Use the following code to create the file:

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
package oe;
public class OrderEntry {
   public static void main(String[] args)
   {
   System.out.println("Order Entry
Application");
   }
}
```

2. Save and compile **`OrderEntry.java`** with the following command line:

```
javac -d C:\labs\D53983GC11\OrderEntry\classes
OrderEntry.java
```

3. Run the `OrderEntry` application.

   a. Open a DOS window and use the `cd` command to change the directory to `C:\labs\D53983GC11\OrderEntry\classes`.

   b. Run the file using the command **`java oe.OrderEntry`**.

   Note: To ensure that the correct version of code is run, irrespective of the working directory, include classpath information in the run command as follows:

```
java –classpath
C:\labs\D53983GC11\OrderEntry\classes
oe.OrderEntry
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: *Exploring Primitive Data Types and Operators*

Goal

The goal of this practice is to declare and initialize variables and use them with operators to calculate new values. You also categorize the primitive data types and use them in code.

Note: If you have successfully completed the previous practice, you should continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les03` directory and continue with this practice.

Remember that if you close a DOS window or change the location of the `.class` files, you must set the `CLASSPATH` variable again.

Your Assignment

Add some code to the simple `main()` method in the `OrderEntry` class created in the last practice: declare variables to hold the costs of some rental items, and after displaying the contents of these variables, perform various tests and calculations on them and display the results.

Note: Ensure that the `CLASSPATH` variable points to the location of your `.class` files`(C:\labs\D53983GC11\OrderEntry\classes` or the location specified by your instructor`)`.

Modifying the OrderEntry Class and Adding Some Calculations

1. Declare and initialize two variables in the `main()` method to hold the cost of two rental items. The values of the two items are 2.95 and 3.50. Name the items anything you like, but do not use single-character variable names; instead, use longer meaningful names such as `item1` and `item2`. Also, think about your choice of variable type.

   Note: Recompile the class after each step, fix any compiler errors that may arise, and run the class to view any output.

   a. Use four different statements: two to declare your variables and two more to initialize them, as follows:
   ```
   double item1;
   double item2;
   item1 = 2.95;
   item2 = 3.50;
   ```

   b. However you can also combine the declaration and initialization of both variables into a single statement:
   ```
   float item1 = 2.95, item2 = 3.50;
   ```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 3: Exploring Primitive Data Types and Operators (continued)

2. Use `System.out.println()` to display the contents of your variables. After recompiling the class, run the class and see what is displayed. Then, modify the code to display more meaningful messages.

   a. To simply display the contents of the variables:
   ```
   System.out.println(item1);
   System.out.println(item2);
   ```

   b. To display more useful information:
   Hint: Use the + operator.
   ```
   System.out.println("Item costs " + item1);
   System.out.println("Item costs " + item2);
   ```

3. Now that you have the costs for the items, calculate the total charge for the rental. Declare and initialize a variable to hold the number of days and to track the line numbers. This variable holds the number of days for which the customer rents the items, and initializes the value to `2` for two days. Display the total in a meaningful way such as `Total cost: 6.982125`.

   a. Create a variable to hold the item total:
   ```
   double itemTotal;
   ```

   b. Declare and initialize variables to hold the number of days (initialized to 2) and to keep track of line numbers:
   ```
   int line = 1, numOfDays = 2;
   ```

   c. Calculate the total charge for the rental:
   ```
   itemTotal = ((item1 * numOfDays) + (item2 *
   numOfDays));
   System.out.println("Total cost: " + itemTotal);
   ```

4. Display the item total in such a way that the customer can see how it has been calculated. To do so, display the item total as the item cost multiplied by the number of rental days:
   ```
   System.out.println(
      "Item " + line++ +" is " + item1 + " * " +
   numOfDays + " days = " +(item1 * numOfDays));

   System.out.println(
     "Item " + line++ +" is " + item2 + " *   " + numOfDays
   + " days = " +(item2 * numOfDays));
   ```

5. Compile and run the **`OrderEntry`** class. Ensure that the `.class` file has been placed in the correct directory **(`C:\labs\\D53983GC11\OrderEntry\classes\oe`).**

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 4*: Controlling Program Flow*

Goal

The goal of this practice is to make use of flow-control constructs that provide methods to determine the number of days in a month and to handle leap years.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les04` directory and continue with this practice.

Remember that if you close a DOS window or change the location of the `.class` files, you *must* set the `CLASSPATH` variable again.

Your Assignment

Create a program that calculates the return date of a rental item based on the day it was rented and on the total number of days before it must be returned. You must determine how many days are in the month and whether the year is a leap year.

Modifying the `OrderEntry` Class to Calculate Dates

1. Modify the number of days in a month. Use a `switch` statement to set an integer value to the number of days in the month that you specify. For now, add all of the code in the `main()` method of the `OrderEntry.java` application.

   a. Declare three integers to hold the day, month, and year. Initialize these variables with a date of your choice.
   `int day = 25, mth = 5, yr = 2000;`

   b. Add a simple statement to display the date. Select a format that you prefer, such as day/month/year or month/day/year.
   `System.out.println(day + "/" + mth + "/" + yr);`

   c. Declare a variable to hold the number of days in the current month. Then, using a `switch` statement, determine the value to store in this variable. Use **`daysInMonth`** as the name of the variable.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Note: The hardest part of this exercise is remembering how many days there really are in each month. Here is a reminder if you need it: There are 30 days in September, April, June, and November. All other months have 31 days, except for February, which has 28 days (ignore leap years for now).

```
int daysInMonth;
switch (mth) {
  case 4:
  case 6:
  case 9:
  case 11: daysInMonth = 30;
           break;
  case 2:  daysInMonth = 28;
           break;
  default: daysInMonth = 31;
           break;
    }
```

d. Add a simple statement to display the number of days for the current month.

```
System.out.println(daysInMonth + " days in
month");
```

2. Ensure that your `CLASSPATH` is set correctly (**`C:\labs\D53983GC11\OrderEntry\classes`**), and then compile and test the program. Experiment with different values for the month. What happens if you initialize the month with an invalid value, such as 13?

   For January 27, 2000, the output should look something like the following:

```
27/1/2000
31 days in the month
```

3. Use a `for` loop to display dates.

   a. Using a **`for`** loop, extend your program so that it prints out all of the dates between your specified day/month/year and the end of the month. Here is an example:

      If your day variable is 27, your month variable is 1 (January), and your year variable is 2000, then your program must display all of the dates between January 27 and January 31 (inclusive) as follows:

```
27/1/2000
28/1/2000
29/1/2000
30/1/2000
31/1/2000
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Hint: You must use the result of the `switch` statement in step 1 to determine the last day in the month.

```
System.out.println("Printing all days to end of
month using for loop...");
for (int temp1 = day;
         temp1 <= daysInMonth;
         temp1++) {
     System.out.println(temp1 + "/" + mth + "/" +
yr);
}
```

b. Compile and test your program, making sure that it works with a variety of dates.

c. Modify your program so that it outputs a maximum of 10 dates. For example, if your day/month/year variables are 19/1/2000, the output must now be as follows:

```
19/1/2000
20/1/2000
21/1/2000
22/1/2000
23/1/2000
24/1/2000
25/1/2000
26/1/2000
27/1/2000
28/1/2000
```

Ensure that your program works for dates near the end of the month, such as 30/1/2000. In this situation, it must output only the following:

```
30/1/2000
31/1/2000
```

To do this, use the following code:

```
// Print maximum of 10 dates, using a for
loop
System.out.println("Printing maximum of 10
days using for loop...");
  for (int temp3 = day, iter = 0;
     temp3 <= daysInMonth && iter < 10;
     temp3++, iter++) {
     System.out.println(temp3 + "/" + mth +
"/" + yr);
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

d. Compile your program. Then, test it with a variety of dates to ensure that it still works.

4. Determine whether the `year` you specify is a leap year. Use the `boolean` operators `&&` and `||`.

a. Build a `boolean` statement that tests `year` to see whether it is a leap year. A year is a leap year if it is divisible by 4, *and* if it is either not divisible by 100 *or* it is divisible by 400.

```
boolean isLeapYear = (yr % 4 == 0) &&
     // year divisible by 4?  AND
          (  (yr % 100 != 0) ||
// year not divisible by 100 OR
             (yr % 400 == 0)  );
// year divisible by 400
```

b. Modify your `switch` statement from step 1 to apply to leap years. Remember that February has 29 days in leap years and 28 days in nonleap years.

```
switch (mth) {
 case 4:
 case 6:
 case 9:
 case 11: daysInMonth = 30;
          break;
 case 2:  daysInMonth = ((isLeapYear) ? 29 : 28);
          break;
 default: daysInMonth = 31;
          break;
}
```

c. Build and test your program with a variety of dates. The following table includes some sample leap years and nonleap years that you may want to use as test data:

Leap years	**Nonleap years**
1996	1997
1984	2001
2000	1900
1964	1967

5. Calculate the date on which each rental is due. The due date is the current date plus three days. For this test, you use a number of different dates for the current date, not just today's date.

a. Declare three variables to hold the due date.
`int dueDay, dueMth, dueYr;`

b. Add a variable to hold the rental period of three days.
`int rentDays = 3;`

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. Add to your program the due date calculation that adds the rental period to the date you used in step 1. Display your original date and the due date in a meaningful way. The output should look something like:

```
Rental Date:              27/2/2001
Number of rental days:    3
Date Due:                 2/3/2001
```

```
dueDay = rentDays + day;
System.out.println("Rental Date:   " + day + "/"
                   + mth + "/" + yr);
System.out.println("Number of rental days: " +
rentDays);
System.out.println("Date Due back:       " +
dueDay + "/" + dueMth + "/" + dueYr);
```

d. Test your routine with several dates. For example, try February 29, 2001.

e. What are the problems you must address?

f. Modify your program to catch input dates with invalid months (not 1–12).

```
// Determine invalid months
if ((mth > 0 )& (mth <13))
  System.out.println (mth + " is a valid month");
else
  System.out.println (mth + " is not a valid month");
```

6. When building a software solution to a problem, you must determine the size and scope of the problem and address all of the pertinent issues. One of the issues is what to do if the rental period extends beyond the current month. For example, if the rental date is August 29 and the rental is for three days, the return date must be September 1, but with the current solution, it is August 32, which is an obvious error. Acme Video store rents items only for 10 or fewer days.
To handle such issues, follow these steps:

a. Add code to test whether the calculation results in a valid day.

```
// is dueDay valid for the current month?
if (dueDay <= daysInMonth)
     System.out.println(dueDay + "/" + dueMth + "/" +
     dueYr);
```

b. If the rental period crosses into a new month, be sure to increment the month.

```
else {
// set dueDay to a day in the next month
dueDay = (dueDay - daysInMonth);
// increment the month
dueMth = (dueMth + 1);
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. If the rental period crosses into a new year, be sure to increment the year.

```
// is the new month in a new year
if (dueMth > 12) {
dueMth = 1;
dueYr += 1;
}
```

d. Test your routine with various dates.

Optional (Do if you have time.)

1. Replace the `for` loop that prints all days to the end of the month with a `while` loop.

   Print all days to the end of the month using a `while` loop:

```
// initialize temp2 to day of the month
int temp2 = day;
System.out.println("Printing all days to end of month
using while loop...");
while (temp2 <= daysInMonth) {
   System.out.println(temp2 + "/" + mth + "/" +
yr);
   temp2++;
}
```

2. Replace the `for` loop that prints a maximum of 10 days using a `while` loop.

   Print a maximum of ten days using a `while` loop:

```
System.out.println("Printing maximum of 10 days using
while loop...");
// initialize temp4 to day of the month
int temp4 = day;
int numSoFar = 0;
while (temp4 <= daysInMonth) {
   System.out.println(temp4 + "/" + mth + "/" +  yr);
   temp4++;
   if (++numSoFar == 10)
     break;
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 5: *Building Java with Oracle JDeveloper 11g*

Starting in Practice 5, you use JDeveloper to build an application using techniques you would use during real-world development. The practice supports the technical material presented in the lesson and incorporates best practices to use while developing a Java application.

Goal

In this practice, you explore using the Oracle JDeveloper IDE to create an application and a project so that you can manage your Java files more easily during the development process. You learn how to create one or more Java applications and classes using the rapid code-generation features.

More importantly, you now start using JDeveloper for most of the remaining lab work for this course (occasionally returning to the command line for various tasks). By the end of the course, you will have built and deployed the course GUI application while continuing to develop your Java and JDeveloper skills.

In this practice, you use the files found in the **`C:\labs\D53983GC11\les05`** directory (or the location specified by your instructor). They are similar to the ones you created in earlier practices (with subtle differences).

Your Assignment

- In the first section, you explore JDeveloper's rapid code-generation features by using the default JDeveloper paths to create a new default application. You then create a project from existing code in the `C:\labs\D53983GC11\les05` directory. You also view a UML diagram that displays the classes that have been created up to this point in the course.
- In the optional section, you run and test the application with the debugger.

Creating an Application and Project

Launch Oracle JDeveloper 11*g* from the desktop icon provided, or ask your instructor how to start JDeveloper. (In this practice, you must use the `C:\labs\D53983GC11\les05` directory.)

1. Create a new application.

   a. In the Applications Navigator, on the left hand side of the screen, click **New Application**.

   b. In the Create Application dialog, enter the following application name: **`OrderEntryApplication`**.

   c. Change the Directory Name field to **`C:\labs\D53983GC11\les05`** (or the directory specified by your instructor). You can use the Browse button to locate the directory.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

   d. In the Application Template field, ensure that the default value **Generic Application** is selected. Click **OK** to create your application definition.

   e. Click **Finish**. You create a project explicitly in the next step.

2. Create a new project in the new application, and populate the project with files from the `C:\labs\D53983GC11\les05\src\oe` directory.

   a. Notice that the new `OrderEntryApplication` application has been created and appears in the Navigator. Right-click `OrderEntryApplication` and select the **New Project** menu item. The New Gallery dialog displays. Select **Project from Existing Source** in the Items section of the New Gallery window and click **OK** to invoke the Create Project from Existing Source wizard.

   b. Click the **Next** button on the Welcome screen. In the Location page of the wizard, change the name of the project to **`OrderEntryProject`** and select the **`C:\labs\D53983GC11\les05OrderEntryProject`** directory (or the directory you have been using). Click the **Next** button.

   c. On the Specify Source page of the wizard, click the **Add** button next to Java Source Paths. Navigate to the subdirectory containing the Java source files (which are in the `src\oe` subdirectory of the `C:\labs\D53983GC11\les05` directory tree). Click **Select**.

   d. On the Included tab, ensure that the **Include Content from Subfolders** check box is checked. Confirm that the output directory is **`C:\labs\D53983GC11\les05\classes`**, and then click **Add**. Check that all the `.java` files in the `C:\labs\D53983GC11\les05\src\oe` directory are to be included, and click the **Finish** button.

The new project is displayed in the Navigator. Double-click the name to invoke the Project Properties dialog. In the **Project Source Paths** page, set the Default Package field to **`oe`** and click **OK**.

   e. Note that the `OrderEntryApplication` and `OrderEntryProject` names are in italics in the Navigator. This is because the application is not yet saved. Select it and then select **File > Save All**.
   The font reverts to normal after the application is saved. Expand the application and project nodes to examine their contents.

   f. Compile the files in the project. Right-click **`OrderEntryProject`** and select the **Rebuild** option.
   Observe the compilation progress in the Log window.

   g. Right-click the project again and select **Run** from the context menu. In the Choose Default Run Target dialog box, browse to the **`oe`** package and select **`OrderEntry.java`**. Click **OK**.
   View the output results of your application in the Log window.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Examining a UML Diagram

View a UML diagram showing the classes that were created in the lessons up to this point in the course.

1. In the Applications Navigator, click **Open Application**.
2. Browse to locate **`C:\labs\D53983GC11\les05`**, select **`OrderEntryWorkspaceLes05.jws`** and click Open. If you get a message asking if you want to migrate application files, click **Yes**.
3. In the Navigator select the **`OrderEntryProjectLes05`** project, and then select **File > Open.**
4. In the **Open** dialog box, double-click **model**, and then double-click **`oe`**.
5. Select **`UML Class Diagram1.java_diagram`** and click **Open**. The diagram displays the classes created up to this point in the course.

Optional: Debugging the Course Application

Run the `OrderEntryApplication` application in debug mode and examine how the debugger works.

1. Expand the Application Sources and oe nodes in the Navigator, and then open the **`Order.java`** file in the Code Editor by double-clicking the file name.
2. Scroll down to lines 67 and 68. Remove the comment marks from the `System.out.println`, and then set breakpoints on the following two statements:
   ```
   item1Total = item1.getItemTotal();
   System.out.println("Item 2 Total: " +
   item2Total);
   ```
   Note: To set a breakpoint on a line, click the left margin next to the line.
3. In the Navigator, select the **`OrderEntry.java`** file, right-click, and then select **Debug** from the context menu.
   JDeveloper creates a new debugger tab that opens at the lower-right portion of the JDeveloper window. The execution of the code stops at your first breakpoint, as indicated by a red arrow. The red arrow indicates the next line that is about to be executed when you resume debugging.
   The Log/Debug window is modified to contain two tabs—a Log tab and a Breakpoints tab—in which you can view all of the breakpoints that you have set. The Log tab must display the output results generated by the application. Resize the windows if required.
4. Visually select the **Smart Data** tab in the lower-right window, which is called the Debug window.
   Note: If the Debug window is not visible, display it by selecting the View > Debugger > Smart Data menu item. The check box next to the Data item must be selected to make it visible. Otherwise, the tab is removed from the Debug window.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

5. Locate the **`item1`** variable in the Smart Data tab and expand it. Using the values of **`quantity`** and **`unitPrice`**, calculate the **`item1Total`** of the order. What is the present value of **`item1Total`**?
   (**Hint:** The value for `quantity` is displayed as 2 and the value `for unitPrice` is displayed as 2.95.) However the value for `item1Total` displays as "null" in the Smart Data window.
6. Select **Debug > Step Over** (alternatively, press F8 or click the appropriate toolbar icon) to calculate `item1Total`. Note the changes to the `item1Total` instance variable in the Smart Data tab of the Debug window. Was your calculation in the previous step correct?
7. In the toolbar at the top of the screen, click **Resume** (F9 or select Debug > Resume). The red arrow in the Code Editor advances and highlights the line with the next breakpoint detected in the code-execution sequence.
8. Continue by selecting the **Debug|Resume** menu (press the F9 key, or click the toolbar button) until the program is completed. You need to select it only once.
9. Remove the breakpoints from the **`Order.java`** source file by clicking each breakpoint entry (red dot) in the margin for each line with a breakpoint.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: *Creating Classes and Objects*

Goal

The goal of this practice is to complete the basic functionality for existing method bodies of the Customer class. You then create customer objects and manipulate them by using their public instance methods. You display the Customer information back to the JDeveloper message window.

Note: For this practice you need to change to the `les06` directory, and load the `OrderEntryApplicationLes06` application. (This workspace file contains the solution to the previous practice, Practice 5.)

Your Assignment

In this practice, you begin refining the application for the Order Processing business area. These classes continue to form the basis for the application that you are building for the remainder of the course. After creating one or more Customer objects, you associate a customer with an order.

Refining the Customer Class

1. In the `OrderEntryProjectLes06` in the Application Navigator, make the following changes to the Customer class:

   a. Make all instance variables private. To do this, double-click the **`Customer.java`** file to open it in the Source Editor. Make your changes directly in the code.

   b. Assign each of the **`setXXX()`** methods to its appropriate field.

   c. The `get()` methods must be assigned. Confirm whether the **`getXXX()`** methods return their appropriate field values.

   Note: The naming convention—such as `setId(), setName()`, and so on—for these methods makes the classes more intuitive and easier to use.

2. At the moment, there is no way to display most or all details for a Customer object by calling one method. You need to address this deficiency.

   a. Add a new **`toString()`** public method to the class, without arguments, and return a String containing the customer's ID, name, address, and phone number. The resultant string should be a concatenation of the attributes that you want to display, as in the following example:

   ```
   public String toString() {
   return property1 + " " + property2;
         }
   ```

   Note: The `toString()` method is a special method that is called whenever a String representation of an object is needed. The `toString()` method is very

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

useful to add to any class, and thus it is added to almost all of the classes that you create. When adding the `toString` method, a dialog box appears with the message, "OK to override method." Click **Yes**.

b. Save the Customer class and compile it to remove any syntax errors. Compile by right-clicking the **`Customer.java`** file and selecting the **Make** option.

Creating Customer Objects (`OrderEntry` Class)

3. Modify the `main()` method in the `OrderEntry` class to create two customer objects.

a. In the `main()` method of `OrderEntry.java` create two customer objects using the **`new`** operator, assigning each one to a different object reference (use `customer1` and `customer2`).

```
Customer customer1 = new Customer();
Customer customer2 = new Customer();
```

b. At the end of the `main()` method, initialize the state of each customer object by calling its public **`setXXX()`** methods to set the ID, name, address, and phone. Use the data in the following table:

Id	Name	Address	Phone
1	Gary Williams	Houston,TX	713. 555. 8765
2	Lynn Munsinger	Orlando, FL	407.695.2210

```
customer1.setId(1);
customer1.setName("Gary Williams");
customer1.setAddress("Houston, TX");
customer1.setPhone("713.555.8765");

customer2.setId(2);
customer2.setName("Lynn Munsinger");
customer2.setAddress("Orlando, FL");
customer2.setPhone("407.695.2210");
```

c. Print the two customer objects created, under a printed heading of "**Customers:**" by calling the **`toString()`** method inside the argument of the `System.out.println(…)` method. For example:

```
System.out.println("\nCustomers:");
System.out.println(customer1.toString());…
```

Note: Alternatively, you can just print the customer object reference variable to achieve the same result, as in the following example:
`System.out.println(customer1);`

The latter technique is a feature of Java that is discussed in a subsequent lesson.

d. Save the `OrderEntry` class and compile and run the class to view the results.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 6: Creating Classes and Objects (continued)

Modifying `OrderEntry` to Associate a Customer to an Order

4. In the `main()` method of the `OrderEntry` class, associate one of the customer objects with the order object, and then display the order details.

    a. Call the **`setCustomer()`** method of the `order` object passing in the object reference of **`customer1`** (or **`customer2`**).
    `order.setCustomer(customer1);`

    b. After setting the customer, call the **`showOrder()`** method of the **`order`** object.
    `order.showOrder();`

    c. Save, compile and run the **`OrderEntry`** class.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 7: *Object Life Cycle Classes*

Goal

The goal of this practice is to provide experience with creating and using constructors, class-wide methods, and attributes. You also use an existing `DataMan` class to provide a data-access layer for finding customers and products in the OrderEntry application. Part of the practice is designed to help you understand method overloading by creating more than one constructor and/or method with the same name in the same class.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les07` directory, load the `OrderEntryApplicationLes07` application, and continue with this practice. (This application contains the solution to the previous practice, Practice 6).

Viewing the model: To view the course application model up to this practice, expand `OrderEntryApplicationLes07 application – OrderEntryProjectLes07 - Application Sources – oe`, and double-click the UML Class Diagram1 entry. This diagram displays all of the classes created up to this point in the course.

Your Assignment

Create one or more suitable constructors to properly initialize the customer objects when instantiated. Examine the `Order` class and the new instantiations. Copy and examine the `DataMan` class to provide class-wide (static) attributes of customer objects to be used by the OrderEntry application when it associates a customer object with an order.

Modifying Customer Information

1. Create two constructors for the `Customer` class. Create a `no-arg` constructor to provide default initialization, and another constructor to set the actual name, address, and phone properties. The `no-arg` constructor is invoked by the second constructor.
   a. Add a `no-arg` constructor to the `Customer` class; the constructor is used to generate the next unique ID for the customer object by first declaring a class variable called **`nextCustomerId`** as a private static integer initialized to zero.
   `private static int nextCustomerId = 0;`
   b. In the `OrderEntry` class, comment out the **`customer.setId`**, **`customer.setName`**, **`customer.setAddress`** **and** **`customer.setPhone`** statements for both `customer1` and `customer2`.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. In the `Customer` class, create a `no-arg` constructor, increment the **nextCustomerId**, and use the **setId()** method with **nextCustomerId** to set the ID of the customer.

```
public                              Customer()
{
    nextCustomerId++;
    setId(nextCustomerId);
}
```

d. Add a second constructor that accepts a **name, address**, and **phone** as `String` arguments. This constructor must set the corresponding properties to these values.

```
public    Customer(String    theName,    String
theAddress, String thePhone)
```

e. In the first line of the second constructor, chain it to the first constructor by invoking the `no-arg` constructor by using the **this()** keyword. This is done to ensure that the ID of a customer is always set regardless of the constructor used.

```
{
  this();
  name              =              theName;
  address           =           theAddress;
  phone             =             thePhone;
}
```

f. Save, compile, and run the OrderEntry class to check the results. Including the order and item details that are displayed as output, you should see "**Customer: 1 null null null**".

Replacing and Examining the `Order.java` File

1. In Windows Explorer, copy the `Order.java` class from the `C:\labs\D53983GC11\Les07Adds` directory into your current working …`\src\oe` directory. For example, if you are working in `les07` directory, copy the files under `C:\Labs\D53983GC11\les07\src\oe`.

   a. In the Application Navigator, select your application (**OrderEntryApplication**) and select the **File** > **Open** menu option. Navigate to your current …`\src\oe` directory and select the **Order.java** file. Click the **Open** button. The file is now included in the list of files. If needed, select **View** > **Refresh** to see the new file in the navigator.

2. The new version of the Order class also has one constructor. Examine the way in which the order date information is managed.

   a. Note that the **OrderDate** variable that was commented out is now a `private` variable.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

b. After the package statement at the top of the class, notice the import statements (before the class declaration):

```
import java.util.Date;
     import java.util.Calendar;
```

c. Note that the `orderDate` type is `Date` instead of `String`, and that the three `integer` variables (`day, month`, and `year`) have been removed.

3. Examine the methods that depend on the three integer date variables to use `orderDate`.

a. The return type and value of the **`getOrderDate()`** method have been replaced as follows:

```
public Date getOrderDate()
{
return orderDate;
}
```

Also included is an overloaded void `setOrderdate()` method that accepts a `Date` as its argument and sets the `orderDate` variable.

b. The **`getShipDate()`** method had used the `Calendar` class to calculate the ship date. The body of `getShipDate()` has been replaced with the following:

```
int daysToShip = Util.getDaysToShip(region);
     Calendar c = Calendar.getInstance();
     c.setTime(orderDate);
     c.add(Calendar.DAY_OF_MONTH, daysToShip);
     return c.getTime().toString();
```

c. The **`setOrderDate()`** method body is coded to set the `orderDate` by using the `Calendar` class methods, using the three input arguments. The following date initialization code has been deleted:

```
day = 0;
month = 0;
year = 0;
```

d. Note that the **`setOrderDate(int,int,int)`** method has been modified. The following three bold lines of code:

```
if ((m > 0 && m <= 12) && (y > 0   )) {
 day = d;
 month = m;
 year = y;
}
```

have been replaced with these three lines of code:

```
     Calendar c = Calendar.getInstance();
c.set(y, m - 1, d);
orderDate = c.getTime();
```

4. A `no-arg` constructor has been created to initialize the order number, date, and total. Note the following:

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

- A new class variable, **`nextOrderId`** has been declared and initialized to **100**.
- In the `no-arg` constructor, the **ID** of the order is set to the value in **`nextOrderId`**, and then the **`nextOrderId`** value is incremented by **1**. The **`orderTotal`** value is set to **0**, and the **`orderDate`** value is set as follows:
  `orderDate  = new Date;`

Loading the `Dataman.java` Class File into JDeveloper

The `DataMan` class is used to create the data that is used to test the application. The file creates the customer objects and later is used to access a database for information. This class is really a convenience class that simplifies your application testing. However, after this class is completed, it can be changed to retrieve data from a database without impacting your application.

1. In Windows Explorer, copy the **`DataMan.java`** class from the `C:\labs\D53983GC11` directory into your current working …`\src\oe` directory.
2. Select your application and select the **File > Open** menu option. Navigate to your current **`...\src\oe`** directory and select the **`DataMan.java`** file. Click **OK**. The file is now included in the list of classes. If needed, select **View > Refresh** to see the new file in the navigator.
3. Save and compile the **`DataMan`** class.
   Note: You can compile DataMan.java by right-clicking the file and selecting the Make menu option.
4. Save, compile, and run the **`OrderEntry`** class to verify that the code still works. You can compile `OrderEntry.java` by right-clicking the file and selecting the **Make** menu option.

Modifying `OrderEntry` to Use `DataMan`

Modify the `main()` method in `OrderEntry` to use customer objects from the `DataMan` class.

1. Use the class name **`DataMan.`** as the prefix to all customer reference variables customer1 and customer2. For example, change the code:
   `order.setCustomer(customer1);`
   to become:
   `order.setCustomer(DataMan.customer1);`

   Note: You are accessing a class variable via its class name—that is, there is no need to create a `DataMan` object. In addition, the customer variables in `DataMan` are visible to `OrderEntry` because they have default (package) access.
2. Save, compile and run the **`OrderEntry`** class to verify that the code still works. Replace **`customer1`** with **`customer3`** or **`customer4`** from `DataMan` to confirm that your code now uses the customer objects from `DataMan`.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

Practice 8: *Using Strings and the `StringBuffer`, Wrapper, and Text-Formatting Classes*

Goal

The goal of this practice is to modify the `Util` class to provide generic methods to support formatting the order details, such as presenting the total as a currency and controlling the date string format that is displayed. This should give you exposure to using some of the java text formatting classes.

In this practice, you use the `GregorianCalendar` class. This class enables you to obtain a date value for a specific point in time. You can specify a date and time and see the behavior of your class respond to that specific date and time. The class can then be based on the values you enter rather than on the system date and time.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les08` directory, load the `OrderEntryApplicationLes08` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, load the OrderEntryApplicationLes08 application. In the Applications – Navigator node, expand OrderEntryApplicationLes08 – OrderEntryProjectLes08 - Application Sources – oe, and double-click the UML Class Diagram1 entry. This diagram displays all of the classes created up to this point in the course.

Your Assignment

Create a method called `toMoney()` to return a currency-formatted string for the order total. Also create a method called `toDateString()` that formats the date in a particular way. Then, modify the `Order` class to use these methods to alter the display of order details, such as the order date and total.

Adding Formatting Methods to the `Util` Class

1. Create a static method called `toMoney()` that accepts an amount as a double and returns a `String`.

   a. Open `Util.java` and add the following import statement to the class:
   ```
   import java.text.DecimalFormat;
   ```

   b. Add the following `toMoney()` method code to the class to format a double:
   ```
   public static String toMoney(double amount) {
   DecimalFormat df = new DecimalFormat("$##,###.00");
   return df.format(amount);
   }
   ```

   c. Save and compile the **`Util`** class.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

2. Us the static `toDateString()` method to format a date.

   a. Add the following import statements to the **`Util`** class:
   ```
   import java.util.Date;
   import java.text.SimpleDateFormat;
   ```

   b. Use the following code for your method:
   ```
   public static String toDateString(Date d) {
   SimpleDateFormat df = new SimpleDateFormat("dd-  MMMM-
   yyyy");
   return df.format(d);
   }
   ```

   c. Save and compile the **`Util`** class.

3. In this step, you use the `GregorianCalendar` class. This class enables you to obtain a date value for a specific point in time. You can specify that date and time and then see the behavior of your class based on the values you enter (and not simply the system date and time).

   Create another static method called `getDate()` that accepts three integers representing the day, month, and year, and returns a `java.util.Date` object representing the specified date (for example, month = 1 represents January on input). Because many of the methods in the `Date` class that could have been used are deprecated, you use the `GregorianCalendar` class to assist with this task.

   a. Import the **`java.util.GregorianCalendar`** class.

   b. Use the following for the method:
   ```
   public static Date getDate(int day,int month,int year)
   {
        // Decrement month, Java interprets 0 as
        // January.
     GregorianCalendar gc =
         new GregorianCalendar(year, --month, day);
    return gc.getTime();
   }
   ```

   c. Save and compile the **`Util`** class.

Using the Util Formatting Method in the `Order` Class

In the `Order` class, modify the `toString()` method to use the `Util` class methods `toMoney()` and `toDateString()` altering the display format.

1. In the **`toString()`** method, replace the return value with the following text. When `shipMode` is not specified, you do not need to display the information for "`Shipped:` ".

   ```
   return "Order: " + id +
     " Date: " + Util.toDateString(orderDate) +
   ```


Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
" Shipped: " + shipMode +
" (" + Util.toMoney(getOrderTotal()) + ")";
```

2. Save and compile the **Order** class, and then run **OrderEntry** to view the changes to the displayed order details.
3. Import the **java.text.MessageFormat** class in the `Order` class and use this class to format the **toString()** return value as follows:

```
import java.text.MessageFormat;
Object[] msgVals = {new Integer(id),
  Util.toDateString(orderDate), shipMode,
  Util.toMoney(getOrderTotal()) };
return MessageFormat.format(
                "Order: {0} Date: {1} Shipped:
     {2} (Total: {3})",msgVals);
```

4. Save and compile the **Order** class, and then run the `OrderEntry` class to view the results of the displayed order. The change to the displayed total should appear.

Optional: Using Formatting in the `OrderItem` Class

In the `OrderItem` class, modify the `toString()` method to use the `Util.toMoney()` methods to alter the display format of the item total.

1. In the **toString()** method, replace the **return** statement with the following:

```
return lineNbr + " " + quantity + " " +
                Util.toMoney(unitPrice);
```

2. Save and compile the **OrderItem** class, and then run the **OrderEntry** class to view the changes to the order item total.

Optional: Using Util.`getDate()` to Set the Order Date

1. In the OrderEntry class, alter the **second order object creation statement** to use the **Util.getDate()** method to provide the value for the first argument in the constructor. Select the previous day's date for the values of the day, month, and year arguments supplied to the `Util.getDate()` method.

   The call to the constructor should look like the following:
   ```
   Order order2 = new Order(Util.getDate(7, 3,   2002),
   "overnight");
   ```

2. Save, compile and run the **OrderEntry** class to confirm that the order date has been set correctly.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 9: *Using Streams for I/O*

Goal

The goal of this practice is to use some of the byte- and character-based stream classes to read and write application data. You also use Object Serialization to save and restore objects.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les09` directory, load the `OrderEntryApplicationLes09` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, load the `OrderEntryApplicationLes09` application. In the Applications – Navigator node, expand `OrderEntryApplicationLes09 - OrderEntryProjectLes09 - Application Sources - oe` and double-click the `UML Class Diagram1` entry. This diagram displays all of the classes created up to this point in the course.

Your Assignment

In this practice you use some of the stream classes to manipulate data. First you use the `PrintWriter` class to write a file containing customer information. Then, you use various classes – `FileInputStream, InputStreamReader` and `Scanner` - to read from this file and output the values. Finally, you use object serialization to save and restore customer and order information. Import the necessary I/O classes when prompted to do so by JDeveloper.

Using PrintWriter to Create a File Containing Customer Data

Create a file to contain the customer information that is hard-coded in the `DataMan` class.

1. At the end of the `OrderEntry` class `main` method, declare a `String` variable for the name of the file that will hold the customer information. Call the file **`Customers.txt`**.
   ```
   String fileName = "customers.txt";
   ```
2. Declare an instance of the **`PrintWriter`** class to write to the file.
   ```
   PrintWriter pw = new PrintWriter(fileName);
   ```
3. Write a record for each of the customers in the `DataMan` class, using the value returned by the **`toString()`** method.
   Note that you do not have to explicitly use the `toString()` method.
   ```
   pw.println(DataMan.customer1);
   pw.println(DataMan.customer2);
   pw.println(DataMan.customer3);
   pw.println(DataMan.customer4);
   ```


THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

4. Add the following statement to the main method declaration. (Exception handling is discussed in Lesson 14).
   ```
   throws Exception
   ```
5. Close the instance of **PrintWriter**.

Using Different Classes to Read the `Customers.txt` File and Print the Values.

Oracle University and Global Business Solutions Peru SAC use only

Notice the different syntax used in the following steps, and (in one case) the different output.

1. Use `FileInputStream` to read and output the contents of the Customers file. Remember that `FileInputStream` is a class that is used for byte-based streams.
   a. Declare an instance of **FileInputStream** to read the file that you created.
   ```
   FileInputStream fis = new   FileInputStream(fileName);
   ```
   b. Declare a **variable** of type `int` to hold the size of the buffer and set it to the value returned by the `available()` method.
   ```
   int fileSize = fis.available();
   ```
   c. Create a **byte array** of the size of the buffer to store the bytes read in from the file.
   ```
   byte[] bbuf = new byte[fileSize];
   ```
   d. Read the file in from the buffer.
   e. Close **FileInputStream**.
   f. Try to print the buffer as a `String`.
   g. Run **OrderEntry**.
   What does the output look like? The result is unrecognizable as customer information because output from `FileOutputStream` is not a Java character string.
   h. Replace the print buffer instruction with a loop to print out each byte from the array individually.
   ```
   for (int cx = 0; cx < fileSize; cx++) {
      System.out.print(bbuf[cx]);
    }
   ```
   i. Rerun **OrderEntry**.
   What does the output look like this time? The output is simply a list of the decimal values of the byte stream returned (the stored ASCII characters). In order to view the output as text, you would need to cast each byte to `char`.
2. Use `InputStreamReader` to read and output the values from the Customers file. Remember that `InputStreamReader` handles character-based data.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

a. Declare an instance of **InputStreamReader** that reads an input stream containing the **Customers** file.
```
InputStreamReader isr = new InputStreamReader(new
FileInputStream(fileName));
```

b. Using the buffer size stored in 2b (above), create a character array to hold the file contents.
```
char[] cbuf = new char[fileSize];
```

c. Read the file into the buffer and print it as a single unit.
```
isr.read(cbuf);
System.out.println(cbuf);
```

d. Close the instance of **InputStreamReader**.

e. Run `OrderEntry`. You should now see customer information correctly displayed as earlier in the program.

3. Use `Scanner` to read and print the contents of the Customer file.

a. Declare an instance of **Scanner** to read the file you created earlier.
```
Scanner sc = new Scanner(new File(fileName));
```

b. Define **a loop** to read in and print one line of the file at a time until the end of file is reached. (Use the `Scanner nextLine()` method).
```
while (sc.hasNext()) {
  System.out.println(sc.nextLine());
 }
```

c. Close your instance of **Scanner**.

d. Run **OrderEntry**. You should see the same correct customer information output as before.

Using Serialization to Save and Restore Objects.

In this practice you use serialization to save and restore first a simple object, and then a more complex one containing nested objects. You then mark one of the nested objects as "transient" to specify that it should not be saved when the owning object is written to file. The files created by this approach are a permanent copy of the object data, which can be used elsewhere in this application, or in another one.
(**Hint:** Refer to the serialization example in the lesson if you need help.)

1. **Use serialization to save and restore a Customer object:** Save the `customer1` object to a stream and then restore and run it.

a. Ensure that the `Order,OrderItem` and `Customer` classes can use object serialization, by specifying that they implement the **Serializable** interface.

b. In the **OrderEntry** class, declare a new **ObjectOutputStream** instance based on a new **FileOutputStream** instance, referencing the file you want to save the object to. (Call the file **customers.ser**).
```
ObjectOutputStream cs = new   ObjectOutputStream(new
FileOutputStream("customers.ser"));
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. Write the **DataMan customer1** object to the file.
```
cs.writeObject(DataMan.customer1);
//entire object is written
```

d. Close the **ObjectOutputStream** instance.

e. Create an **ObjectInputStream** instance based on a new **FileInputStream** instance, referencing the file you just created, to enable you to read the object back.
```
ObjectInputStream ois =
new ObjectInputStream(new
FileInputStream("customers.ser"));
```

f. Read the saved **Customer** object from **customers.ser** into a different `Customer` variable.
```
Customer restCust1 = (Customer)ois.readObject();
//entire object is read
```

g. Close the **ObjectInputStream** instance.

h. Print out the restored `Customer` object.

i. Run `OrderEntry`.

In the Log window, you should see the same information as displayed from the original `customer1` earlier. This is a very simple object that does not contain any nested objects or object references within it.

2. **Use serialization to save and restore an Order object:** Save `the` `order2` object to a stream and then restore and run it. The `Order` class is more complex, and contains `OrderItem` and `Customer` classes nested within it, enabling you to see the power of object serialization.

   a. In the `OrderEntry` class, declare a new **ObjectOutputStream** instance as before, referencing the file you want to save the object to. (Call the file **orders.ser**.)

   b. Write the **order2** object to the file. This saves the complete `Order` class structure, known as a "graph."

   c. Close the **ObjectOutputStream** instance.

   d. Create an **ObjectInputStream** instance as before to enable you to read the object back from the **orders.ser** file.

   e. Create a new instance of `Order` called **restOrd2** to hold the restored order object, and read the saved **order2** object into it.

   f. Close the **ObjectInputStream** instance.

   g. Print out the **restOrd2** object.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 9: Using Streams for I/O (continued)

h. Run `OrderEntry`. You should see the details for `restOrd2` in the Log window—the same information as displayed from the original `order2` earlier in the Log.

Using the “transient” Modifier to Prevent Fields being Saved and Restored

If a nested object’s class is not marked as serializable, or you do not want it to be stored with the “owning” object, you mark it as “transient.” This tells the JVM to ignore it when writing the object to an object stream.

It is essential that references to a transient object must include a test for a null value, in order to be safe when processing a restored copy of the owning object.

It has been decided that, in the `OrderEntry` application, customer information will no `longer` be stored in the order graph. To accomplish this, mark the customer variable as “transient.”

1. In **`Order.java`**, add the modifier **`transient`** to the customer variable.
   `private transient Customer customer;`
2. Scroll down to the `showOrder()` method, and check that references to customer are conditional on it having a non-null value.
3. Run `OrderEntry`. In the Log window, you should see that the information displayed for `restOrd2` no longer contains the customer information.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 10: *Inheritance and Polymorphism*

Goal

The goal of this practice is to understand how to create subclasses in Java and use polymorphism with inheritance through the `Company` and `Individual` subclasses of the `Customer` class. You refine the subclasses, override some methods, and add some new attributes using the Class Editor in JDeveloper.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les10` directory, load the `OrderEntryApplicationLes10` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, examine the `UML Class Diagram`. This diagram displays all of the classes created up to this point in the course.

Business Scenario

The owners of the business have decided to expand their business and sell their products to companies as well as individuals. Both are customers, but because companies have slightly different attributes to individuals, there is a need to hold separate company information and individual information. It therefore makes sense to create subclasses for `Company` and `Individual`. Each of the subclasses will have a few of their own methods and will override the `toString()` method of `Customer`.

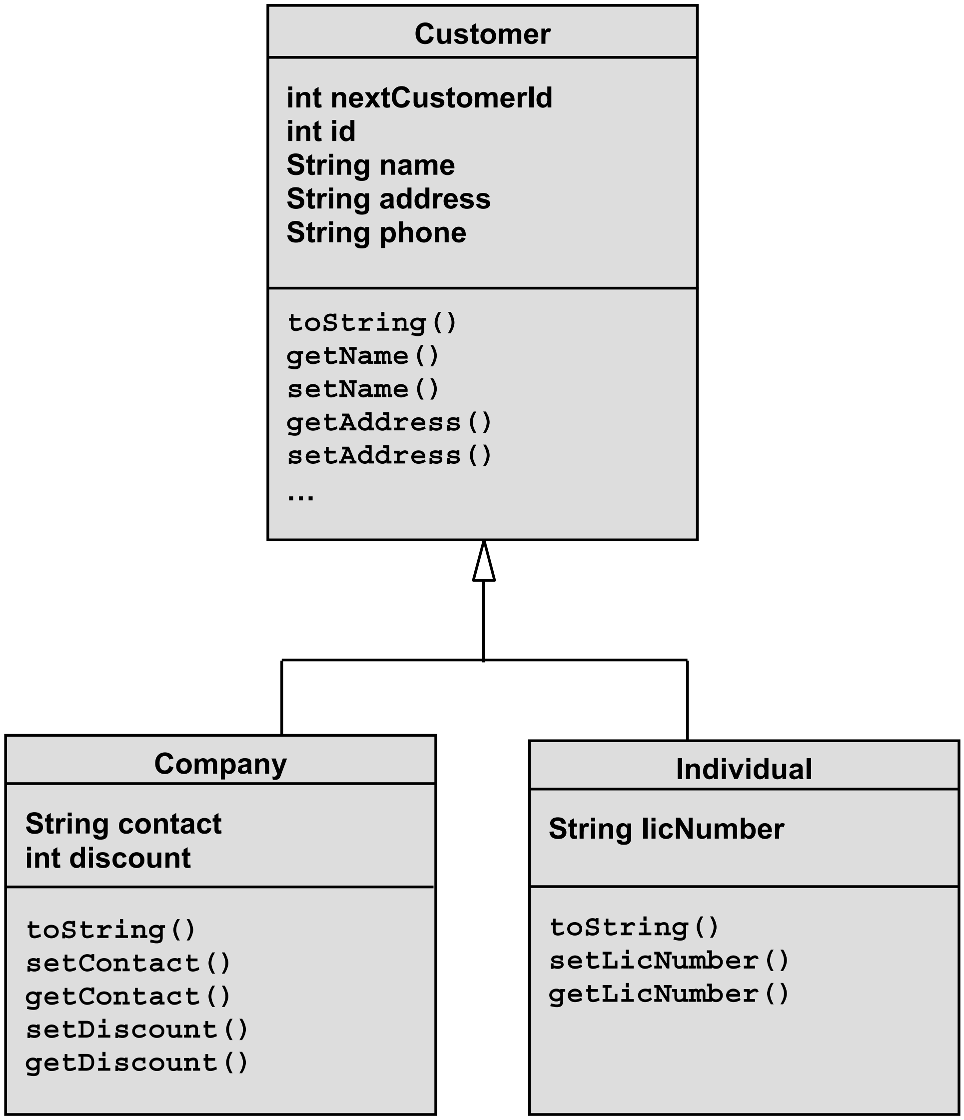
Your Assignment

Add two new classes as subclasses. The added classes are `Company` and `Individual`, and they both inherit from the `Customer` class. Here is a class diagram to show the relationships between `Customer`, `Company`, and `Individual`. Each box represents a class. The name of the class appears at the top of each box. The middle section specifies the attributes in the class, and underlined attributes represent class variable. The lower section specifies the methods in the class.

Notice the arrow on the line connecting `Company` and `Individual` to `Customer`. This is the UML notation for inheritance.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED



Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Defining a New `Company` Class

1. Define a `Company` class that extends `Customer` and includes the attributes and methods that were defined in the class diagram on the preceding page of this practice.

   a. Right-click the **`OrderEntryProject`** project and select **New** from the context menu. In the New Gallery window, select the General category (if not selected by default) and **Java Class** from the **Items** list. Click **OK**.

   b. In the Create Java Class wizard, enter **`Company`** in the **Name** field, set the package to **`oe`**, and then click the **Browse** button next to the Extends field. In the Class Browser window, click the **Hierarchy** tab and locate and expand the `oe` package. Select the **`Customer`** class and click the **OK** button. The `oe.Customer` class is displayed in the Extends field. Leave the Optional Attributes in their default state and click **OK**. When the source code for the generated class is displayed, save your work.

   c. In the Code Editor for the Company.java file, declare the following attributes:
   ```
   private String contact;
   private int discount;
   ```

   d. To generate the methods for the attributes, right-click in the Code Editor and select **Generate Accessors** from the context menu.

   e. In the **Generate Accessors** dialog, check the check box to the left of the class name. Expand the nodes beside each of the attributes to see the names of the methods that are to be generated for them. Click **OK** to generate the methods.

   f. Save your changes.

2. Modify the `Company` constructor to add arguments.

   a. Add the following arguments to the `no-arg` constructor:
   ```
   public Company(String aName, String aAddress,
               String aPhone, String aContact, int
               aDiscount) { …
     }
   ```

   b. Use the arguments to initialize the object state (including the superclass state). **Hint:** Use the `super(…)` method syntax to pass values to an appropriate superclass constructor to initialize the superclass attributes. Here is an example:
   ```
   super(aName, aAddress, aPhone);
   contact = aContact;
   discount = aDiscount; ...
   ```

3. Add a public String `toString()` method in the `Company` class to return the contact name and discount as in the following example: `(Scott Tiger, 20%)`.

   a. Include in the return value the superclass details, and format them as follows: `<`*`company info`*`> (<`*`contact`*`>, <`*`discount`*`>%)` using the following statement:

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
return super.toString() + " ("+ contact + ", "+
discount + "%) ";
```

b. Save and compile the **Company.java** class.

Defining a New `Individual` Class as a Subclass of `Customer`

Define an `Individual` class that extends `Customer` and includes the attributes and methods that were defined in the class diagram on the preceding page of this practice.

1. Create the **Individual** class as you did for the `Company` class in step 1.a. Add the **licNumber** attribute as a **String** with a **private** scope, and ensure that the `get` and `set` methods are created to retrieve the values.
2. Alter the `no-arg` constructor to accept four arguments for the name, address, phone, and license number.
3. Complete the constructor body initialization by assigning the arguments to the appropriate instance variables in the Individual class and its superclass.
4. Override the `toString()` method that is defined in the superclass, and append the license number enclosed in parentheses to the superclass information.
5. Save and compile the **Individual** class.

Modifying the `DataMan` Class to Include `Company` and `Individual` Objects

Add two new class variables to the `DataMan` class: one for a `Company` object and the other for an `Individual`. Open `DataMan` in the Code Editor and add two new class variables called `customer5` and `customer6`.

1. Create a **Company** variable called `customer5`, and initialize it by using the `Company` constructor. Here is an example:

   ```
   static Company customer5 =
   newCompany("Oracle","Redw…","80…","Larry…",20);
   ```

2. Create an **Individual** variable called **customer6** and initialize it using the constructor from the `Individual` class.
3. Save and compile `DataMan.java` by right-clicking the file and selecting **Make** from the context menu.

Testing Your New Classes in the OrderEntry Application

1. In these steps you modify the `OrderEntry` code that assigns a customer object to each of the two order objects in the `main()` method. You then run the application to see the results of your work.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

a. Open **OrderEntry.java** in the Code Editor. Locate the line assigning **`customer3`** with the first **`order`** object.
For example, find:
`order.setCustomer(DataMan.customer3);`
Hint: Press **Ctrl** + **F** to display a search dialog box.
Replace `customer3` with `customer5` (the company in `DataMan`).

b. Compile the code and, if successful, explain why the code was successful.

c. Now replace **`customer4`** in the **`order2.setCustomer(DataMan.customer4)`** argument with **`customer6`** (the individual in `DataMan`).

d. Compile and run the `OrderEntry` application. What is displayed in the customer details for each order?
Explain the results that you see. If you are using the same application you used in the previous practice, remember that if you want the customer information to appear as part of the stored order (2), you need to remove the `transient` modifier on the customer declaration in `Order.java`.

Optional: Refining the `Util` and `Customer` Classes and Testing the Results

It is not obvious to the casual user that the data that is printed for the customer, company, or individual objects represents different objects, unless the user is made aware of the meaning of the subtle differences in the displayed data. Therefore, modify your code to explicitly indicate the object type name in the text that is printed before the rest of the object details, as follows:

```
[Customer] <customer details>
[Company] <company details>
[Individual] <individual details>
```

If you manually add the bracketed text string before the return values of the `toString()` methods in the respective classes, then `[Company]` is concatenated to `[Customer]` and `[Individual]` is concatenated to `[Customer]` for the subclasses of `Customer`. Therefore, the solution is to use inherited code called from the `Customer` class that dynamically determines the run-time object type name.

You can determine the run-time object type name of any Java object by calling its `getClass()` method, which is inherited from the `java.lang.Object` class. The `getClass()` method returns a `java.lang.Class` object reference, through which you can call a `getName()` method returning a `String` containing the fully qualified run-time object name. For example, suppose that you add the following line to the `Customer` class:

```
String myClassName = this.getClass().getName();
```

The variable `myClassName` will contain a fully qualified class name that includes the package name. The value that is stored in `myClassName` will be `oe.Customer`.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

To extract only the class name, you must strip off the package name and the dot that precedes the class name. This can be done by using a `lastIndexOf()` method in the `String` class to locate the position of the last dot in the package name, and then extract the remaining text thereafter.

1. Add the `getClassName()` method to the `Util` class, and call it from the `toString()` method in the `Customer` class.

   a. Open **`Util.java`** in the Code Editor and add a **`static String getClassName()`** method that determines the run-time object type name and returns only the class name.

   ```
   public static String getClassName(Object o) {
     String className = o.getClass().getName();
     return className.substring(
                    className.lastIndexOf('.')+1,
                    className.length());
   }
   ```

   b. Save and compile **`Util.java`**. Note that JDeveloper automatically recompiles other classes that are dependent on code in `Util.java`. JDeveloper has a class-dependency checking mechanism.

2. Open **`Customer.java`** in the Code Editor.

   a. Prefix a call to the **`Util.getClassName()`** method before the rest of the return value data in the `toString()` method, as follows:

   ```
   return "[" + Util.getClassName(this) + "]
                                            " + id+...;
   ```

   b. Save and compile **`Customer.java`**.

   c. Run the OrderEntry application to view the results.

   d. In the preceding step a, what does this represent? Why do you pass the parameter value **`this`** to the `Util.getClassName()` method?
   Explain why the compiler accepts the syntax.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practices for Lesson 11

Practice 11*: Using Arrays and Collections*

Goal

The goal of this practice is to gain experience with Java array objects and work with collection classes such as the `java.util.ArrayList` class. You also work with command-line arguments.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les11` directory, load the `OrderEntryApplicationLes11` application, and continue with this practice.

Viewing the model: To view the course application model up to this practice, load the `OrderEntryApplicationLes11` application. In the Applications – Navigator node, expand `OrderEntryApplicationLes11 - OrderEntryProjectLes11 - Application Sources - oe` and double-click the `UML Class Diagram1` entry. This diagram displays all of the classes created up to this point in the course.

Your Assignment

Continue to use JDeveloper to build on the application classes from the previous practices. Enhance the `DataMan` class to construct an array of `Customer` objects, and then provide a method to find and return a `Customer` object for a given ID.

The `Order` class needs to be modified to contain an `ArrayList` of order items, requiring a method to add items into the `ArrayList`, and (optionally) another method to remove the items.

Modifying `DataMan` to Hold `Customer` Objects in an Array

1. Modify `DataMan` to build an array of customers.
   a. Define a private static array of `Customer` objects named **`customers`**.
   ```
   private static Customer[] customers;
   ```
   b. Initialize the array to a **`null`** reference.
   ```
   private static Customer[] customers = null;
   ```
2. Create a public static void method called `buildCustomers()` to populate the array of customers. The array must hold six objects using the four `Customer` objects, the `Company` object, and the `Individual` object that you have already created.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

a. In the body of the method, first test whether the `customers` variable is not `null`, and if so, return from the method without doing anything because a non-null reference indicates that the customers array has been initialized. If `customers` is `null`, then you must create the array object to hold the six customer objects that are already created.

```
public static void buildCustomers()
{
    if (customers != null) return;
    customers = new Customer[6];
```

b. Now move (cut and paste) the definitions of the four existing **`Customer`** objects, the **`Company`**, and the **`Individual`** into the body of this method, *after* creating the array object. Then, delete the **`static`** keyword and class name or type before each **`customer<n>`** variable name. Modify each variable to be the name of the array variable followed by brackets enclosing an array element number. *Remember, array elements start with a zero base.*
For example, replace:

```
static Customer customer1 = new Customer(…);
```

with:

```
customers[0] = new Customer(…);
```

The example here assigns the customer object to the first element in the array. Repeat this for each **`customer<n>`** object reference in the code.

c. Create a static block that invokes the **`buildCustomers()`** method to create and initialize the array of customer objects, when the `DataMan` class is loaded. Place the block at the end of the `DataMan` class. (Static blocks in the class definition are sometimes called class constructors.)

```
static
          {
         buildCustomers();
          }
```

d. Save and compile the **`DataMan`** class. What other classes are compiled? Explain the results. (Only fix errors that are related to the `DataMan` class, if any. Any errors pertaining to the `OrderEntry` class will be fixed after doing the next set of questions).
Hint: Look in the Messages and Compiler tabs in the Log Window.

Modifying `DataMan` to Find a Customer by ID

1. Create a public static method called `findCustomerById(int custId)`, which returns a `Customer` object, where the argument represents the ID of the `Customer` object to be found. If found, return the object reference for the matching `Customer`; otherwise, return a `null` reference value.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

a. Why is the customer array guaranteed to be initialized when the `findCustomerById()` method is called? Thus, you can write code assuming that the array is populated.

b. Write a loop to scan through the `customers` array, obtaining each customer object reference to compare the **`custId`** parameter value with the return value from the **`getId()`** method of each customer. If there is a match, return the customer object reference; otherwise, return a null.

c. Save and compile the **`DataMan`** class, only fixing the syntax errors that are reported for the `DataMan` class.

2. Next, fix the syntax errors in the `OrderEntry` class as a result of the changes made to `DataMan`.

   a. In the Code Editor, locate and modify each line that directly refers to the **`DataMan.customer<n>`** variables that previously existed.
   Hint: You can quickly navigate to the error lines by double-clicking the error message line in the Compiler tab of the Log window.
   Replace each occurrence of the **`DataMan.customer<n>`** text with a method call to: **`DataMan.findCustomerById(n)`**. For example, replace:

   ```
   System.out.println(DataMan.customer1.toString());
   ```
   with
   ```
   System.out.println(DataMan.findCustomerById(1).toStrin
   g());
   ```

   b. Save, compile and run the **`OrderEntry.java`** file to test your changes.

Optional: Modifying the Order Class to Hold an `ArrayList` of `OrderItem` Objects

Currently, the `Order` class has hard-coded the creation of two `OrderItem` objects as instance variables, and the details of each `OrderItem` object are set in the `getOrderTotal()` method. This is impractical for the intended behavior of the `Order` class. You must now replace the two `OrderItem` variables with an `ArrayList` that will contain the `OrderItem` objects. Therefore, you must create methods to add and remove `OrderItem` objects to and from the `ArrayList`.

In the `Order` class, define an `ArrayList` of order items, and replace the `OrderItem` instance variables, removing the code dependent on the original `OrderItem` instance variables.

1. Add a statement at the beginning of your class, after the package statement, to

   ```
   import the java.util.ArrayList class
   ```

2. Declare a new instance variable called **`items`** as an **`ArrayList`** object reference. Also remove, or comment out, the declarations of the two instance variables called `item1` and `item2`, and the code that is using these variables.
   Hint: The following methods directly use the `item1` and `item2` variables:

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

getOrderTotal(), showOrder().

3. In the **Order no-arg constructor,** add a line to create the item `ArrayList`, as follows:

   ```
   items = new ArrayList(10);
   ```

4. Save your changes to the `Order` class and compile it.

Optional: Modifying `OrderItem` to Handle Product Information

Before you create the method to add an `OrderItem` object to the `items` ArrayList, you must first modify the `OrderItem` class to hold information about the product being ordered. Each `OrderItem` object represents an order line item. Each order line item contains information about a product that is ordered, its price, and quantity that is ordered.

1. Edit the **OrderItem** class and add a new instance variable called **product**. Declare the variable as a **private int**, and generate or write the **getProduct()** method and **setProduct()** methods. Modify the **toString()** method to add the `product` value between the `lineNbr` and `quantity`.
2. Create an **OrderItem** constructor to initialize the object by using values that are supplied from the following two arguments: **int productId** and **double itemPrice**. Initialize the item **quantity** variable to **1**.
   Note: The `OrderItem` class does not provide a `no-arg` constructor.
3. Save and compile the **OrderItem** class.

Optional: Modifying `Order` to Add Products into the `OrderItem` ArrayList

In the `Order` class, create a new `public void` method called `addOrderItem()` that accepts one argument, an integer called `product`, representing an ID of the product being ordered. This method must perform the following tasks:

1. Search the **items** array list for an **OrderItem** containing the supplied product. To do this, create a loop to get each `OrderItem` element from the `items` array list.
   Hint: Use the **size()** method of the `ArrayList` object to determine the number of elements in the array list.
2. Use the **getProduct()** method of the `OrderItem` class to compare the product value with the existing product value in the order item. If the product with the specified ID is found in an `OrderItem` element from the array list, increment the quantity by using the `setQuantity()` method. If the specified product *does not exist* in any `OrderItem` object in the array list, create a new `OrderItem` object by using the constructor that will accept the product and a price.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Then, add the new `OrderItem` object into the array list.
Note: Because line item numbers are set relative to their order, set the line number for the **`OrderItem`**, by using the **`setLineNbr()`** method, after an item is added to the array list. The line number is set using the **`size()`** of the array list because the elements are added to the end of the array list. For now, assume that all products have a price of $5.00.

```
public void addOrderItem(int product)
{
    OrderItem item = null;
    boolean productFound = false;
    for (int i = 0; i < items.size() &&
!productFound; i++)
   {
        item = (OrderItem) items.get(i);
        productFound = (item.getProduct() ==
product);
   }
   if (productFound)
       {
          item.setQuantity(item.getQuantity() + 1);
       }
       else
           {
              item = new OrderItem(product, 5.0);
              items.add(item);
              item.setLineNbr(items.size());
  }
  }
```

3. The **`orderTotal`** value is now calculated as each product is added to the order. Thus, you must also add the price of each product to **`orderTotal`**.

   Hint: Use the **`getUnitPrice()`** method from the `OrderItem` class. Because the **`orderTotal`** is now updated as each product is added to the order, the **`getOrderTotal()`** method can simply return the **`orderTotal`** value.

   ```
   orderTotal += item.getUnitPrice();
   ```
   Note: This may already be done due to previous changes to the method.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

4. Modify the **`showOrder()`** method to use an iteration technique to loop through the **`items` array list** to display each **`OrderItem`** object by calling the **`toString()`** method.
   Hint: Import `java.util.Iterator`, and use the array list `elements()` method to create an iteration. See your course notes for an example, or ask your instructor for further clarification.

```
public void showOrder()
{
    System.out.println(toString());
    if (customer != null)
   {
       System.out.println("Customer: " + customer);
    }
    System.out.println("Items:");
    for (Iterator it = items.iterator(); it.hasNext(); )
   {
        System.out.println(it.next().toString());
    }
}
```

5. Save and compile the **`Order`** class and remove any syntax errors.

6. Test your changes to the `OrderItem` and `Order` classes by modifying the **`OrderEntry`** class to add products 101 and 102 to the first order object. For example, before the call to **`showOrder()`**, enter the bold lines shown:

```
order.setCustomer(DataMan.findCustomerById(5));
order.addOrderItem(101);
order.addOrderItem(102);
order.addOrderItem(101);
order.showOrder();
```

7. Compile (eliminating syntax errors first), save, and run **`OrderEntry.java`**. Confirm that your results are accurate. For example, verify that the order total for Order 100 is reported as $15.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 12*: Using Generic Types*

There is no practice for this lesson.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 13: *Structuring Code Using Abstract Classes and Interfaces*

Goal

The goal of this practice is to learn how to create and use an abstract class and how to create and use an interface.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les13` directory, load the `OrderEntryApplicationLes13` application, and continue with this practice.

Your Assignment

The `OrderItem` class currently tracks a product only as an integer. This is insufficient for the business, which must know the name, description, and retail price of each product. To meet this requirement, you need to create an abstract class called `Product` and define three concrete subclasses called `Software`, `Hardware`, and `Manual`.

To support the business requirement of computing the sales tax on the hardware products, you create an interface called `Taxable` that is implemented by the `Hardware` subclass.

To test your changes, you modify `DataMan` to build a list of `Product` objects and to provide a method to find a product by its ID.

Creating an Abstract Class and Three Supporting Subclasses

Create a public abstract class called Product in OrderEntryProject.

1. Declare the following attributes and their **`get`*`XXX`*`()`** and **`set`*`XXX`*`()`** methods.
   Note: Remember to add the **`abstract`** keyword before the `class` keyword in the source code after the **`Product.java`** file is created.

```
private static int nextProductId = 2000;
private int id;
private String name;
private String description;
private double retailPrice;
```

2. Define a **`public no-arg`** constructor that assigns the next product ID to the ID of a new product object before incrementing **`nextProductId`**.

```
public Product()
{
    id = nextProductId++;
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

3. Add a **`public String toString()`** method to return the **`ID, name`**, and **`retailPrice`**. Prefix with the class name using **`getClassName(this)`** from the **`Util`** class. Then, format **`retailPrice`** with **`Util.toMoney().`**

```
public String toString()
{
    return "[" + Util.getClassName(this) + "] " + id + "
" + name + " " + Util.toMoney(retailPrice);
}
```

4. Compile and save the **`Product`** class.
5. Create three concrete subclasses of the **`Product`** class, each with attributes and initial values that are listed in the following table. Add the appropriate get and set methods.

Subclass	**Attributes**
Software	String license = "30 Day Trial";
Hardware	int warrantyPeriod = 6;
Manual	String publisher = "Oradev Press";

6. Modify the **`no-arg`** constructor for the **`Software, Hardware`**, and **`Manual`** subclasses to accept three arguments for the product **`name, description,`** and **`price`**. Use this code example for the `Software` class as an example:

```
public Software(String name,String desc,double price)
{
  setName(name);
  setDescription(desc);
  setRetailPrice(price);
}
```

7. Compile and save the new subclasses.

Modifying `DataMan` to Provide a List of Products and a Finder Method

Use the new class definitions in the `DataMan` class to build an inventory of products.

1. In `DataMan`, create an object to hold a collection of products.
   a. Create a private static inner class called **`ProductMap`** that extends `HashMap`. **Note:** Remember to import **`java.util.HashMap`**.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

b. In the **ProductMap** inner class, create the following method to add product objects to the collection. The **Id** is the key and the object reference is the value:

```
public void add(Product p) {
  String key = Integer.toString(p.getId());
  put(key, p);  // use inherited put() method
}
```

c. Declare a **private static ProductMap** variable called **products**, for example:

```
     private static ProductMap products = null;
```

d. Compile and save the **DataMan** class.

2. Create a method to populate the `ProductMap` variable with product objects.

a. Create the method called **buildProducts()** in the **DataMan** class as follows:

```
public static void buildProducts() {
  if (products != null) return;
  products = new ProductMap();
  products.add(new Product();
}
```

b. Save and compile your code. Explain the compilation error that is listed for the line adding the `Product` to the `products` map.

c. Fix the compilation error by adding concrete subclasses of the `Product` class. Replace the line of code `products.add(new Product())` with the following text:

```
products.add(
  new Hardware("SDRAM - 128 MB", null, 299.0));
products.add(new Hardware("GP 800x600", null, 48.0));
products.add(
   new Software("Spreadsheet-SSP/V2.0", null, 45.0));
     products.add(
   new Software("Word Processing-
SWP/V4.5",null,65.0));
products.add(
   new Manual("Manual-Vision OS/2x +", null, 125.0));
```

d. Compile the `DataMan` class and save your changes. Your compilation should work this time.

e. At the end of the file, in the static block of **DataMan**, add a call to the **buildProducts()** method.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

f. Add the following method called **`findProductById()`** to return a `Product` object matching a supplied ID.

```
public static Product findProductById(int id) {
  String key = Integer.toString(id);
  return (Product) products.get(key);
}
```

Note: Because `products` is a `HashMap`, you simply find the product object by using its key—that is, the Id of the product.

g. Save the changes to the `DataMan` class and compile it.

h. Test the **`DataMan`** code and additional classes by printing the product that is found by its Id. Add the following line to the **`OrderEntry`** class at the end of **`main()`**:

```
System.out.println("Product is: " +
                  DataMan.findProductById(2001));
```

i. Compile, save and run the OrderEntry application to test the code.

Optional: Modifying `OrderItem` to Hold `Product` Objects

Replace uses of the `product` variable as an `int` type with the Product class you just created.

1. In **`OrderItem.java`**, change the type declaration for the product instance variable to be **`Product`** instead of **`int.`**
2. Replace the two argument constructors with a single argument called **`newProduct`** whose type is `Product`—that is, remove the **`productId`** and **`itemPrice`** arguments.
3. Change the body of the constructor to store the **`newProduct`** in the **`product`** variable, and set the **`unitPrice`** to be the value that is returned by calling the **`getRetailPrice()`** method of the `product` object.
4. Modify the **`getProduct()`** method to return a **`Product`** instead of an **`int`**, and change the **`setProduct()`** method to accept a **`Product`** instead of an **`int`**.
5. Alter the **`toString()`** method to display the item total instead of the **`unitPrice`**. **Hint:** Use the **`getItemTotal()`** method.
6. Compile and save your code changes. Eliminate syntax errors from the **`OrderItem`** class **only**. Errors that are reported for the `Order` class are corrected in the next step of this lab.

Optional: Modifying `Order` to Add `Product` Objects into `OrderItem`

Alter the `Order.java` class to use the `Product` object instead of an `int` value; to do this, you need to modify the **`addOrderItem()`** method:

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

1. In **`addOrderItem`**, rename the argument to be `productId`, and in the `for` loop replace:

   ```
   productFound = (item.getProduct() == product);
   ```
   with
   ```
   productFound =
                  (item.getProduct().getId() == productId);
   ```

2. In the **`else`** section of the `if` statement, call **`findProductById()`** from `DataMan` by using the **`productId`** value. If a product object is found, create the `OrderItem` using the `product` object; otherwise, do nothing. Here is an example:

   ```
   item = new OrderItem(product, 5.0);
   items.add(item);
   ```
   becomes:
   ```
   Product p = DataMan.findProductById(productId);
   if (p != null) {
     item = new OrderItem(p);
     items.add(item);
   }
   ```

3. Test the changes that are made to the code supporting the `Product` class and its subclasses by modifying `OrderEntry` to use the new `productId` values.

4. Because the ID of `products` (or its subclasses) starts at 2000, edit the **`OrderEntry.java`** file, replacing parameter values in all of the calls to the **`order.addOrderItem()`** method, as shown in the following table:

Replace	With
101	2001
102	2002

5. Save, compile, and run the **`OrderEntryProject`** project, and check the changes to the printed items. Check whether the price calculations are still correct.

Optional: Creating and Implementing the `Taxable` Interface

1. Create an interface called `Taxable` to compute the sales tax on hardware products. This interface is to be implemented by the `Hardware` subclass. To do this, follow these steps.

   a. Right-click the **`OrderEntryProject.jpr`** file in the `Navigator` and select **New** from the context menu. In the General category, select **Java Interface** and click **OK**. Enter **`Taxable`** in the class name and click the **OK** button.

   b. In the Code Editor, add the following variable and method definitions to the interface:

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
double TAX_RATE = 0.10;
     double getTax(double amount);
```

Note: Remember that all variables are implicitly `public static final` and that methods are all implicitly `public`. The implementer of the interface must multiply the `amount`, such as a price, by the `TAX_RATE` and return the result as a `double`.

c. Compile and save the interface.

2. Edit the `Hardware` class to implement the Taxable interface.

   a. Add code to the class definition to implement the interface, as follows:

   ```
   public class Hardware extends Product
    implements Taxable {
   ```

   b. Compile the **`Hardware`** class and explain the error.

   c. Add the following method to complete the implementation of the interface:

   ```
   public double getTax(double amount) {
   return amount * TAX_RATE;
      }
   ```

   Note: To perform this step, right-click the arrow in the margin to the left of the interface declaration. From the context menu, select **Implement Methods**. JDeveloper generates all of the code except for the return-value calculation (which you can modify appropriately).

   d. Compile and save the **`Hardware`** class.

3. Modify the `OrderItem` class to obtain the tax for each item.

   a. Add a **`public double getTax()`** method to determine whether the **`Product`** in the item is taxable. If the product is taxable, return the tax amount for the item total (use the **`getItemTotal()`** method); otherwise, return 0.0. Here is an example:

   ```
   double itemTax = 0.0;
   if (product instanceof Taxable)
   {
   itemTax =((Taxable) product).getTax(getItemTotal());
   }
   ```

   b. Modify the **`toString()`** method to display the tax amount for the item, if and only if, the product is taxable. Use the **`getTax()`** method that you created, and format the value with **`Util.toMoney()`**.

   c. View the changes by compiling **`OrderItem.java`** and then running OrderEntry.

4. Modify the `Order` class to display the tax; modify the order total to include the tax.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 13: Structuring Code Using Abstract Classes and Interfaces (continued)

a. In the **`showOrder()`** method, add a **`local double`** variable called **`taxTotal`** initialized to **0.0** that accumulates the total tax for the order.

b. Modify the **`for`** loop by using iteration to call the **`getTax()`** method for each item, and add the value to **`taxTotal`**.
Hint: To do this, you must cast the return value of **`it.next()`** to **`OrderItem`**.

c. Add three **`System.out.println()`** statements after the loop: one to print the **`taxTotal`**, the second to print the **`orderTotal`** including **`taxTotal`**, and the last without a parameter to print a blank line. Use the **`Util.toMoney()`** method to format the totals.

d. To view the results, compile and save **`Order.java`**. Then, run **`OrderEntry`**.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 14: *Throwing and Catching Exceptions*

Goal

The goal of this practice is to learn how to create your own exception classes, throw an exception object by using your own class, and handle the exceptions.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les14` directory, load the `OrderEntryApplicationLes14` application, and continue with this practice.

Your Assignment

The application does not appropriately handle situations when an invalid customer ID is supplied to the `DataMan.findCustomerById()` method, or when an invalid product ID is supplied to the `DataMan.findProductById()` method. In both cases, a `null` value is returned. Your tasks are to:

- Create a user-defined (checked) exception called `oe.NotFoundException`.
- Modify `DataMan.findCustomerById()` to throw the exception when an invalid customer Id is provided.
- Modify `DataMan.findProductById()` in the `DataMan` class to throw the exception if the given product ID is not valid (that is, not found).

Creating the `NotFoundException` Class

In `OrderEntryProject`, create a new class called `NotFoundException`.

1. Right-click the project name in the Navigator, and select **New** from the context menu. From the New Gallery window, ensure that the Category selected is **General** and the Item selected is **Java Class**. Enter the class name **`NotFoundException`**, and make it a subclass of **`java.lang.Exception`**.
2. Modify the default `no-arg` constructor to accept a **message `String`** argument, and pass the string to the superclass constructor, as in the following example:

```
public NotFoundException(String message) {
  super(message);
}
```

3. Compile and save the **`NotFoundException`** class.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Throwing Exceptions in `DataMan;` Finding Methods, and Handling Them in `OrderEntry`

1. Edit `DataMan.java`, and modify the `findCustomerById()` method to throw the `NotFoundException` when the given customer ID is not found in the array.
   a. At the end of the **`for` loop**, if the local customer object reference is `null` (that is, if the customer is not found), throw an **`exception`** object by using the following error message structure in the constructor argument:

      `"Customer with id " + custId + " does not exist"`

   b. Compile the **`DataMan`** class. Explain the error.
   c. Fix the error by modifying the method declaration to propagate the exception.
   d. Compile **`DataMan`** again. What errors do you get this time? Explain the errors.
   e. Fix the compilation errors by handling the exceptions with a **`try-catch` block** in the **`OrderEntry`** class. For simplicity, use one `try-catch` block to handle *all* of the calls to the **`DataMan.findCustomerById()`** methods. Alternatively, if desired, handle each call in its own `try-catch` block.

      ```
      try {  // calls to findCustomerById() here …
      }
      catch (NotFoundException e) {
        // handle the error here …
           }
      ```

      In the **`catch`** block, you can use the exception's inherited methods to display error information. Use the following two methods to display error information:
      - **`e.printStackTrace()`** to display the exception, message, and stack trace
      - **`e.getMessage()`** to return the error message text as a **`String`**.
   f. Compile, save and run **`OrderEntry.java`**. Test your code with the errors.
2. Modify `findProductById()` to throw `NotFoundException` when the supplied product ID is not found in the product map.
   a. The **`findProductById`** method calls **`get(key)`** to find a product from the `HashMap`. If `get(key)` returns **`null`**, throw **`NotFoundException`** by using the following error message; otherwise, return the product object found. You must also add the product declaration line and modify the current return statement.

      `"Product with id " + id + " is not found"`

   b. Modify the **`findProductById`** method to propagate the exception.
   c. Compile and save **`DataMan`** and explain the compile-time error reported.
   d. In the **`Order`** class, modify **`addOrderItem`** to propagate the exception.
   e. Compile the **`Order`** class and explain why it compiles successfully.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

f. In **`OrderEntry.java`**, use a value of **9999** as the product ID in the first call to **`order.addOrderItem(2001)`**. Compile and run **`OrderEntry`**. Explain why the application terminates immediately after adding product 9999.

g. In **`Order.java`**, remove **`throws NotFoundException`** from the end of the **`addOrderItem()`** method declaration. Write a try-catch block to handle the exception in this method.
Hint: You must return from the method in the catch block to ensure that the **`itemTotal`** is not affected.

h. Compile **`Order`**, run **`OrderEntry.java`**, and explain the difference in output results.

i. In **`OrderEntry`**, replace the 9999 product ID with 2001. Compile, save, and run.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 15: *Using JDBC to Access the Database*

Goal

The goal of this practice is to use the course application to interact with the Oracle database. During this practice, you establish a connection to the database, perform query statements to access the database and retrieve information to integrate into the application.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les15` directory, load the `OrderEntryApplicationLes15` application, and continue with this practice.

Your Assignment

In the `DataMan` class, connect to the database and retrieve data from the `Customers` table.

Setting Up the Environment to Use JDBC

Update the project to include the necessary JDBC classes.

1. Open the **`OrderEntryApplicationLes15`**, double-click the project name, in this case **`OrderEntryProjectLes15`**, and select the **Libraries and Classpath** node. This opens the libraries pane.
2. Click the **Add Library** button, and then scroll through the list of displayed libraries and select **Oracle JDBC**. Select it and click **OK**. Click **OK** again to close the Project Properties dialog box.
3. Save the project.

Adding JDBC Components to Query the Database

1. Modify the `DataMan` class to provide connection information.
   a. Specify the package(s) to import:
   ```
   import java.sql.*;
   ```
   b. Add a static variable to hold the connection information:
   ```
   private static Connection conn = null;
   ```
   c. After the static block at the end of the **`DataMan`** class, add a new method that queries the Customers table. This method takes the `customer_id` as the input parameter and queries the **`Customers`** table. Start by setting the connection code as follows (your instructor will provide you with the appropriate connection URL):

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

```
public static Customer selectCustomerById(int id)
throws Exception {
// Register the Oracle JDBC driver
DriverManager.registerDriver(new
oracle.jdbc.OracleDriver());
// Define the connection url
String url = "jdbc:oracle:thin:@myhost:1521:SID";
// Provide db connection information
conn = DriverManager.getConnection (url, "oe", "oe");
     }
```

2. Add statements to execute the select statement based on a customer ID, and return the result to a customer object. In this practice, you populate only two of the items from the database. In the previous practices, the phone number was defined as a `String`. However, in the database it is stored as a complex object type. A utility named JPublisher can be used to convert an object type to a string so that it can be used in your application. This process is much more detailed than can be covered in this course. So in this practice, the phone number item is not populated with any values.

   a. In the **`selectCustomerById`** method, issue the query based on a customer ID:

```
Customer customer = null;
Statement stmt = conn.createStatement();
System.out.println
("Table Customers query for customer with id: " + id);
ResultSet rset = stmt.executeQuery
("SELECT cust_last_name, nls_territory" +
" FROM customers WHERE customer_id = " + id);
```

   b. If a record is returned, populate the `customer` object:

```
if (rset.next ())      {
customer = new Customer();
customer.setId(id);
customer.setName(rset.getString(1));        //holds
first column
customer.setAddress(rset.getString(2));    //holds
second column
System.out.println("Customer found: " + customer);
// prints both columns to the command window
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. Otherwise throw an exception that the customer is not found, close the statement, and return the customer:

```
else {
throw new NotFoundException("Customer with id " + id +
" not found");
}
rset.close();
stmt.close();
return customer;
}
```

d. Save and compile **DataMan.java.** Correct any errors.

3. Add a new method to close the connection at the end of the **DataMan** class.

a. To do this, add the following code:

```
public static void closeConnection()  {
try {
conn.close();
}
catch (Exception e) {
System.out.println("Error: " + e.getMessage());
 }
}
```

b. Save the **DataMan.java** file.

Testing the JDBC Database Access

1. Up to this point in the course you have been using "hard-coded" customer data. You now modify the code that displays the hard-coded customer information to display "real" customer information from the database. In the **OrderEntry** class, add a call to the `selectCustomerById` method, catch the exception if no customer is found, and close the connection. To do this, follow these steps:

a. Add a call to the **selectCustomerById()** method that you just created. In the **for** loop that displays customer information for the customer IDs specified in the command-line arguments, replace the method **DataMan.findCustomerById(custId)** in the following line:

System.out.println("Arg: " + custId...)

with

DataMan.selectCustomerById(custId);

b. To catch the exception from **DataMan** if no customer is found, scroll down to the **catch** block and replace the line:

catch (NotFoundException e)

with

catch (Exception e)

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

c. After the catch block, add a call to the `closeConnection()` method in the `DataMan` class.

d. Save and compile the `DataMan` class and correct any errors.

4. Change the customer IDs that you have used until now (these are specified in the project's run/debug configuration), and then test the application.

a. Right-click the project in the Navigator, and select **Project Properties** from the context menu. Select **Run/Debug** in the left pane and click the **Edit** button. In the **Program Arguments** field, replace the existing values with real IDs as follows: **150, 352, 468, 999** (do not use commas to separate the IDs when you type them in the Program Arguments field). Click **OK** and then **OK** again.

b. Run the application. Did all of the IDs return customer information? If not, did you see the exception raised?

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 16: *Swing Basics for Planning the Application Layout*

Goal

The goal of this practice is to use JDeveloper to start creating the application layout. You create the main application frame as an MDI window and the internal order frames that are contained in the main window. Working with these frames helps you explore Swing classes and the ways to build GUI applications.
Note: For this practice, you use the **`OrderEntryApplicationLes16`** application.

Your Assignment

Start by creating the main window as an extension of the `JFrame` class. This class contains a `JDesktopPane` object to manage the internal frame layout. You also create a class based on the `JInternalFrame` class in which the customer and order details are entered via atomic Swing components. The components layout is managed through the use of panels and associated layout managers. You use the JDeveloper Frame Wizard to create a basic menu for the application, and a status bar in the main application window.

Creating the Main Application Window

1. Start by creating a new subclass of the `JFrame` class in the OrderEntry project.
   a. Right-click the project name in the navigator, and select **New** from the context menu. Select the **Frame** item from the **Client tier > Swing/AWT** category, and click the **OK** button.
   b. In the **Create Frame** dialog, enter the class name **`OrderEntryMDIFrame`**, and extend **`javax.swing.JFrame`**. In the Optional Attributes section, set the **Title** field to: **Order Entry Application**, and select only the **Menu Bar** and **Status Bar** check boxes. Then, click the **OK** button.
   c. Examine the code for the new class that is generated by JDeveloper, by clicking the **Source** tab. Note that JDeveloper creates a **`jbInit()`** method that is called from the default `no-arg` constructor. The `jbInit()` method should contain all code to initialize the user interface structure. You should modify the code if required, to match with the one displayed.
   d. In the Editor window, click the **Design** tab and examine the visual container hierarchy and presentation of the frame. The container hierarchy is visible in the Structure window (located under the Navigator).
   e. Return to the Code Editor, by clicking the **Source** tab and make the following changes:
      - Replace the **`JPanel panelCenter`** variable declaration, with:
        `JDesktopPane desktopPane = new JDesktopPane();`
        Note: You may need to import **`javax.swing.JDesktopPane`**.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

- In the **`jbInit()`** method, replace **`panelCenter`** references with **`desktopPane`**.

f. Save and compile the **`OrderEntryMDIFrame`** class.

2. Make the frame visible. To do this:

a. Modify **`OrderEntry.java`** by renaming the **`main()`** method to **`test1(…).`**

b. At the end of the class, create a new **public static void main(String[] args)** method, which creates an instance of the OrderEntryMDIFrame and makes it visible.

3. Compile, save, and run the OrderEntry application.

Creating the `JInternalFrame` Class for Order Data

This frame contains the bulk of the UI code for data entry and user interaction for an order, and for assigning a customer and adding items to the order.

1. Create the JInternal Frame and name it OrderEntryFrame.

a. Navigate to the **Client Tier > Swing/AWT** node and select the **Frame** item. In the **Create Frame** dialog box, enter the class name **`OrderEntryFrame`**, and extend the **`javax.swing.JInternalFrame`**. Set the **title** to **`Order`**, and then click **OK**.
Note: The `JInternalFrame` class can be selected by clicking the Browse button.
Note: Once you select `javax.swing.JInternalFrame` in the Extends field, the Title field becomes 'grayed out' and is no longer enterable (I think this is a small bug). To get around this, you either need to enter the title into the Title field **before** selecting `javax.swing.JInternalFrame` in the Extends field, or enter the title via the Property Inspector after having created the frame.

b. The OrderEntryFrame that is generated does not have the desired layout manager or content pane. To set the layout manager and to add a panel to the content pane, open **`OrderEntryFrame`** with the UI Editor.
Note: When the UI Editor is activated (the UI Editor can be invoked by clicking the Design tab), the Property Inspector window is also displayed, showing the properties of the object that is selected in the UI Editor.

c. Select the **`frame`** object by clicking the frame title bar in the UI Editor or the node labeled **`this`** in the structure window. (You may have to expand the UI node to view objects in the containment hierarchy.) In the Property Inspector window, locate the **layout** property and select **BorderLayout** from the pop-up list options. (**Note:** The layout property is under the Visual node).

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

d. Examine lines of code that JDeveloper added or changed in your class by clicking the Source tab. When creating a Swing UI using the JDeveloper UI Editor, it is wise to view changes that are made to the source code as an aid to learning what you would need to write yourself when building the UI manually. Remove the **`private`** declaration from **`BorderLayout`**.

2. Add a `JPanel` to the frame.

   a. Return to the Design view. JDeveloper provides a Component Palette in the toolbar (ask the instructor, if needed).

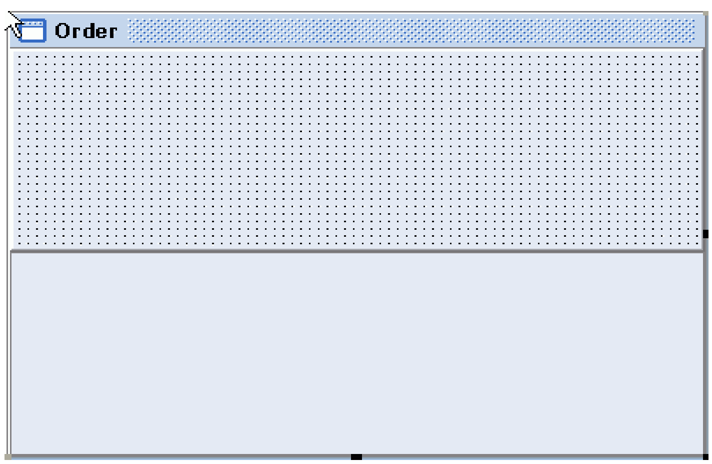
   b. In the Component Palette, select **Swing** from the list, and in the **Containers** sub-group, click the `JPanel` icon, and then click the center of the frame in the UI Editor (or click the node labeled **`this`** in the UI Structure pane). This adds a new panel to the center region of the border layout.
   Note: If the JPanel icon is not visible, expand the Component Palette window by increasing the height.

3. Divide the `JPanel` into two sections by using a GridLayout for the layout, with one column and two rows.

   a. Select **`JPanel1`**, and set its layout property to **GridLayout**.

   b. Expand **JPanel1** in the UI Structure pane, select the **`gridLayout1`** object, and in the **Model** node, set the **columns** property to **1** and set the **rows** property to **2**.

4. Using the following picture as a guide, add another panel to the top and a scroll pane to the bottom of the content panel.



   a. Add a second panel to the top half (or first row) of the first panel by clicking the **`JPanel`** icon in the **Swing Components** palette, and then clicking the **`jPanel1`** object in the UI Editor or in the Structure pane.
   Note: Confirm that the new panel is called **`jPanel2`**, and more importantly, that it is nested inside **`jPanel1`** in the hierarchy.

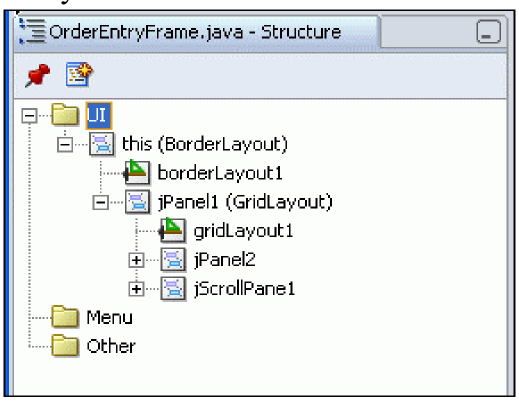
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

b. Add a raised-bevel border to the new panel, **`jPanel2`**, by selecting its **border** property in the Property Inspector and selecting **Swing Border** from the pop-up list. In the **Border dialog box**, select **BevelBorder** and select the **RAISED** option button, and then click the **OK** button.
Note: `jPanel2` should visually occupy the top half of the `jPanel1`.

c. Add a scroll pane to the bottom half (second row) of **`jPanel1`** by clicking the **JScrollPane** button in the **Swing Components** palette and clicking the bottom area of the `jPanel1`. (Alternatively, click the `jPanel1` object in the Structure pane to add the JScrollPane.)

d. Use the Structure window to ensure that you have the following containment hierarchy:



e. Save and compile the **`OrderEntryFrame`** class.

Modifying `OrderEntryMDIFrame` Class to Contain an Internal `OrderEntryFrame`

1. To view the visual results of your internal frame at run time, modify the constructor in `OrderEntryMDIFrame` to create an instance of `OrderEntryFrame`, and then make it visible. To do this, follow these steps.

   a. Edit **`OrderEntryMDIFrame.java`** and, at the end of the constructor, add the following lines of code:

   ```
   OrderEntryFrame iFrame = new OrderEntryFrame();
   iFrame.setVisible(true);
   desktopPane.add(iFrame);
   ```

   Note: The bounds (size and location) of the internal frame must now be set; otherwise, it does not become visible. In addition, you must also alter the dimensions of **`OrderEntryMDIFrame`** to be larger than the initial size of the internal **`OrderEntryFrame`**.

   b. In the jbInit() method of the **OrderEntryMDIFrame** class, locate the following statement:

   ```
   this.setSize(new Dimension(400,300));
   ```

   Then, modify the dimension arguments to be **`700,500`**.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

c. Switch to the **`OrderEntryFrame`**, and add the following line to the **`jbInit()`** method:

```
this.setBounds(0, 0, 400, 300);
```

d. Compile and save **`OrderEntryMDIFrame`** and **`OrderEntryFrame`**.

e. Run **`OrderEntry.java`** to view the results.

2. Notice that the internal frame cannot be maximized, "iconified" (minimized), or closed. Make changes to `OrderEntryFrame` to enable these features.

a. In the **`jbInit()`** method, add the following lines of code to enable the internal frame to be maximized, "iconified," and closed:

```
this.setMaximizable(true);
this.setIconifiable(true);
this.setClosable(true);
```

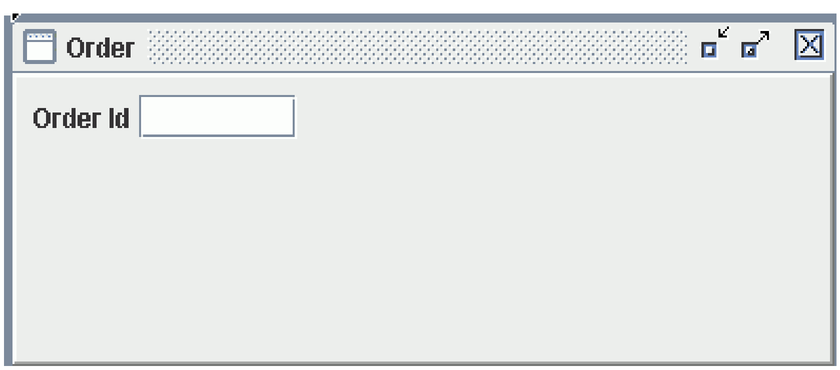
b. Compile and save the changes to **`OrderEntryFrame.java`**.

c. Run the application and notice the changes.

Adding UI Components to `OrderEntryFrame`

1. Before adding UI components to `jPanel2` in OrderEntryFrame, set its layout to `null`.
Note: You could also use the JDeveloper `XYLayout`.
In either case, JDeveloper uses absolute positioning and sizing for components that are added to the panel. It is easier to use absolute positioning when building the initial UI layout. You change the layout again in a subsequent lesson.

Use the following image as a guide to the desired results:



a. In Design mode, select the **Swing** option in the Component Palette pop-up list. Then, add a **`Jlabel`** to **`jPanel2`** and set its **text** property to **Order Id**. Resize the label to see the label value, if needed. What lines of code have been added to your class?

Hint: You should find at least five lines of code (some of them in the `jbInit()` method). Try to identify the three that make the object visible in the

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED
Oracle University and Global Business Solutions Peru SAC use only

panel.
Note: The `setBounds` value can be modified (if required) in the source to make the label clearly visible.

b. From the Swing page of the Component Palette, select a **`JTextField`** component and add it to **`jPanel2`** (to the right of the label).
Note: `setBounds` values can be changed if required.

c. Compile and save **`OrderEntryFrame`**, and then run **`OrderEntry`** to view the results.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-1: *Adding User Interface Components*

Goal

In this practice, you create the menu and visual components so that users can enter order details. The application includes a button to find the customer assigned to the order, and buttons to add and remove products as items in the order. You learn how to build a Swing-based UI application by using the JDeveloper UI Editor to construct the user interface. You also learn how to handle events for the Swing components that are added to the application.

Note: Whenever you create a UI component, JDeveloper declares it as private, but you can remove that if required.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If the compilation from the previous practice was unsuccessful and you want to move on to this practice, change to the `les17` directory, load the `OrderEntryApplicationLes17` application, and continue with this practice.

The UI for the Order Entry Application

The slide in Lesson 17 of your course manual shows a snapshot of the final visual appearance of the application's main window, `OrderEntryMDIFrame`, and a sample `OrderEntryFrame` for an order that is created as an internal frame.

Using the Application

`OrderEntryMDIFrame` provides the main application menu, from which users select the Order > New menu option to create a new order for a customer.

The new order request should create the internal `OrderEntryFrame` and a new `Order` object (whose ID sets the Order Id text field in the frame).

Customer details are entered by providing an ID value in the Customer ID field and clicking the Find button. The Find button event validates whether the customer exits (by using the `DataMan.findCustomerById()` method). When the event validates, it assigns the customer to the order and displays the customer details in the fields provided; otherwise, an error message is displayed.

Products are added to the order by entering a value in the Product ID field and clicking the Add button. Products are found by using the `DataMan.findProductById()` method. They are then added to the order contained in the order item objects that are added to the `JList` in the bottom pane of the `OrderEntryFrame`. Multiple products can be added to the order, but adding a product that already exists in the order increments the item quantity.

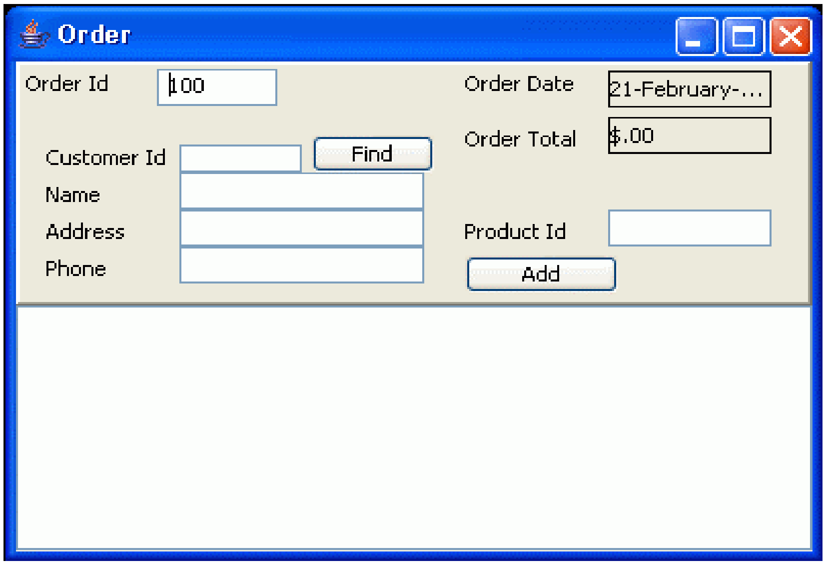
Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-1: Adding User Interface Components (continued)

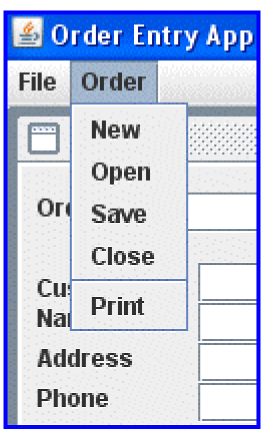
Your Assignment

Use the following screenshot as a guide to modify the menu of `OrderEntryMDIFrame` and add several Swing components to `OrderEntryFrame` to meet user requirements:



Creating the `OrderEntryMDIFrame` Menu

1. The menu structure that is added to the main window should look like the following screenshot:



Use the JDeveloper UI Editor to modify the menu to include the Order menu and its menu items, as shown in the preceding menu.
Note: File > Exit already exists.

   a. Edit the **`OrderEntryMDIFrame`** with the UI Editor. Expand the **Menu** item in the Structure pane, and click the **menuBar** entry to display the initial menu structure in the UI Editor window.

   b. Add the **Order** menu to the right of the **File** menu. Right-click the *outlined* box on the right side of the File menu item, and select the **Insert Menu** option from the context menu.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

c. The new menu should be selected and shown as **`jMenu1`**. With the menu selected, enter the text **`Order`**, overwriting the default menu label text, and then press the **Enter** key.

d. Save your work and compile the class, ensuring that there are no compilation errors. What lines has JDeveloper added to the `OrderEntryMDIFrame` class?

2. Add menu items and a separator to the Order menu, as per the following diagram.

Menu Item `text`
`New`
`Open`
`Save`
`Close`
`<separator>`
`Print`

a. In the Structure window, click the **Order** option that you just created, select the blank outline box at the bottom of the Order menu, and enter the menu label text **New** in the box. Do the same for other menu items in the table. To add a separator (a line that separates menu items), right-click and select **Insert Separator** from the context menu.

b. Save and compile the class. Then, run **`OrderEntry.java`** to view the menu.

Adding Components to `OrderEntryFrame` to Form Its Visual Structure

1. Add text fields and labels for the customer details. Add a Find button for finding a customer by the ID, and add an area in the bottom part of the frame where order information will be displayed.

a. In the top panel, add **`JLabel`** and **`JTextField`** components for the **Customer** details. (These components can be found in the **Swing** list.) Use the sample window on the previous page as a guide for the layout. Create label and text field items as per the following table:

`JLabel` **text** property
Customer Id
Name
Address
Phone

Hint: Aligning UI components works best with **`XYLayout` Manager**, in which alignment is relative to the first component clicked. Select additional components

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

while holding Shift or Control. Right-click a selected component and select an Align option from the context menu.

If you are using the `null` layout manager, JDeveloper generates calls to each component `setBounds()` method, with a `Rectangle` parameter defining the components x, y location, width, and height. You can alter the parameters (x, y, width, height) in the `Rectangle` constructor to manually align and size components, or you can set the `bounds` property for each component in the Inspector.

b. Add a **`JButton`** to the right of the **customer ID** text field, and then set the **text** property to **Find**. Then, save your work.

c. Add a **`JList`** component to the scroll pane in the bottom panel of the `OrderEntryFrame`. The list component should fill the entire bottom section of the frame (just click in the lower pane, and the JList expands and takes up the entire pane).

If you have time…

The following steps take you through adding more Order information to the frame as well as adding and removing products. The process is obviously very similar to what you have already done when adding Customer information to the frame. Therefore, if you are short of time, skip these steps. To see what the completed OrderEntryFrame looks like and how the Add Product functionality works, open `OrderEntryApplication17-2` and run `OrderEntry.java` You can also use this application to start the next practice.

1. Add components for the order information and for the addition of products to an order.
   a. Create a **`JLabel`** and set the **text** property to **Product ID**. To the right of the label, add a **`JTextField`**. Below the product ID label and text field, create a **`JButton`** component, and set the **text** property to **Add**.
   b. Add two **`JLabel`** components at the upper-right side of the top panel, for the order date. Set the first label **text** property to **Order Date**. Set the second label's **text** property to the empty string. In the **Border** property, select the **Swing Border** option in the border property, and then, from the Border dialog box, select the **LineBorder** value. Set the **border thickness** to **1** if it is not already set.
   c. Add two more **`JLabel`** components under the order date labels, for the order total. Set the first label's **text** property to **Order Total**. Set the second label's **text** property to the empty string. In the **Border** property, select the **Swing Border** option in the border property, and then, from the Border dialog box, again select the **LineBorder** value. Set the border thickness to **1** if it is not already set.
   d. Save and compile the OrderEntryFrame class.
   e. Run the OrderEntry application to view the resulting UI layout in the internal frame. Quickly make minor adjustments to the UI layout to make all items clearly visible.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-2: *Adding Event Handling*

Goal

In this practice, you create the order entry details. You add event handling code for the Order > New menu, the Find Customer button, and the Add Product and button.

Note: If you have successfully completed the previous practice, continue using the same directory and files. If you did not complete the practice or if the compilation from the previous practice was unsuccessful and you would like to move on to this practice, change to the `les17-2` directory, load the `OrderEntryApplicationLes17-2` application, and continue with this practice.

Your Assignment

In the previous practice you created the frames to display customer and order information. You now add functionality to the frames, so that a user can display an order, find a customer, and add a product to an order.

Adding Event Handling for the Order > New Menu

1. Modify `OrderEntryFrame.java` in the Source Editor to create an order object and display its initial state in the appropriate components. To do this, do the following:

   a. Create a new instance variable for the order object as follows:
   `Order order = null;`

   b. Add the following method to create a new order object and display its contents in the appropriate components in the frame:

   ```
   private void initOrder() {
    order = new Order();
    jTextField1.setText(
             Integer.toString(order.getId()));
     jLabel8.setText(
      Util.toDateString(order.getOrderDate()));
     jLabel10.setText(
               Util.toMoney(order.getOrderTotal()));
             }
   ```

   Note: The preceding italics represent identifiers for labels and text fields added in the last practice. If you did not add the labels and text fields in the same sequence as the practice steps, you will need to modify the preceding code to refer to the identifiers you have used for the labels and text fields. For example, `jTextField1` refers to the Order Id text field. If you need to check the identifier you have used, select the Order Id field in the Design view and refer to the Property Inspector to see the identifier you have used for that component.

   c. Call the **`initOrder()`** method at end of the **`jbInit()`** method.

   d. To control the x, y location of the upper-left corner of the frame, when it is displayed, declare the following instance and class variables:

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
private static int x = 0;
private static int y = 0;
private static final int OFFSET = 20;
private static final int MAX_OFFSET = 200;
```

and create the following method, to create a cascading effect as new order frames are created:

```
private void setBounds() {
this.setResizable(true);
this.setBounds(x, y,
this.getWidth(), this.getHeight());
x = (x + OFFSET) % MAX_OFFSET;
y = (y + OFFSET) % MAX_OFFSET;
}
```

e. Add a call to your **`setBounds()`** method at the end of the **`jbInit()`** method, after calling **`initOrder()`**.

f. Add one more method to **`OrderEntryFrame`** to make it the active window as follows:

```
public void setActive(boolean active) {
     try {
          this.setSelected(active);
     }
     catch (Exception e) {}
     this.setVisible(active);
     if (active) {
          this.toFront();
     }
}
```

Note: This method will be called from the Order > New menu event handler.

g. Compile and save the **`OrderEntryFrame`** class.

2. Modify `OrderEntryMDIFrame.java` to create the event handler code for the new order menu option.

   a. Open **`OrderEntryMDIFrame.java`** in the UI Editor, expand the menu either in the Visual Editor or the Structure pane, and then select the **New** menu item under the **Order** menu.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

b. Click the **Events** node at the bottom of the Properties Inspector window, click in the text area to the right of the first event called **`actionPerformed`**. The text area will show a button with three dots (ellipses). Click this button to display the `actionPerformed` event generation dialog box. Take note of the **name of the action method** and accept the defaults, and then click the **OK** button.

   Note: JDeveloper generates the event listener code as an anonymous inner class (in the `jbInit()` method) that calls the method that is named in the event dialog window. JDeveloper will position the cursor in the Code Editor inside the empty body of the event handler method created.

c. Move the following lines from the **`OrderEntryMDIFrame()`** constructor to the body of the **`jMenuItem1_actionPerformed()`** method, deleting (or commenting out) the line, making the frame visible, as shown:

```
OrderEntryFrame iFrame = new OrderEntryFrame();
      // iFrame.setVisible(true);
      desktopPane.add(iFrame);
```

   Also add the following line, after adding the frame to the desktop pane in the **`jMenuItem1_actionPerformed`** method:

```
iFrame.setActive(true);
```

d. Compile the **`OrderEntryMDIFrame`** class and save the changes. Run and test the `OrderEntry` application by selecting the **Order** > **New** menu.
   Note: When the application first starts there should not be any order window displayed. Close the internal window by clicking its Close icon (X).

If you are short of time…

The following steps take you through adding event handling for the **Find Customer** button. However if you are short of time, omit the steps and open the OrderEntryApplication17-2Sol application in the Solutions folder, and run `OrderEntry.java` to see how the finished functionality should work.

Adding Event Handling for the Find Customer Button

In this section of the code, you add event handling functionality to the Find button that allows you to display details of a valid customer.

1. Modify `OrderEntryFrame.java` by adding code to do the following:

   – **test** if the customer ID text field has a non-zero length string, and convert it to an integer used in the `DataMan.findCustomerById()` method to return a valid customer. If the customer ID field is empty, or is not a number, the `DataMan.findCustomerById()` method should throw a `NotFoundException`.

   – **display** an error message using the `javax.swing.JOptionPane` class.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

– If the customer is a valid customer, **associate** the customer object with the order and display the customer details in the field that is provided in `OrderEntryFrame`.

The following steps guide you through these tasks.

a. Select the **Find** button and click the **Events** tab in the Property Inspector. Click the ellipses to generate the skeleton code for the **`actionPerformed`** event.

b. In the body of the generated **`jButton1_actionPerformed()`** method, add the following code:

```
int custId = 0;
     Customer customer = null;
     if (jTextField5.getText().length() > 0) {
          try {
     custId = Integer.parseInt(jTextField5.getText());
     customer = DataMan.findCustomerById(custId);
               order.setCustomer(customer);
     jTextField3.setText(customer.getName());
       jTextField4.setText(customer.getAddress());
       jTextField2.setText(customer.getPhone());
            }
          catch (NumberFormatException err) {
               JOptionPane.showMessageDialog(this,
              "The Customer Id: " + err.getMessage() +
              " is not a valid number",
              "Error", JOptionPane.ERROR_MESSAGE);
              jTextField2.setText("");
              }
         catch (NotFoundException err) {
               JOptionPane.showMessageDialog(this,
                         err.getMessage(),
               "Error", JOptionPane.ERROR_MESSAGE);
               jTextField2.setText("");
             }
        }
        else {
             JOptionPane.showMessageDialog(this,
               "Please enter a Customer Id", "Error",
               JOptionPane.ERROR_MESSAGE);
        }
```

Note: As before, pay close attention to the preceding variables in italic type to ensure that you have correctly identified the appropriate labels and text fields.

c. Compile and save your changes. Run the `OrderEntry` application to test your code changes (customer IDs range from 1 to 6).

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

Additional Extra Credit: Adding Event Handling for the Add Product Button

In this section you write code to add products to the order.

1. Modify `OrderEntryFrame.java` by adding code to do the following:
   - Read the product ID that is entered and supply it to the `order.addOrderItem()` method.
   - Update the Order Total field with the latest total after each product is added to the order.
   - Handle errors as appropriate.

   The following steps assist you with these tasks.

   a. Select the **Add** button and create its **`actionPerformed`** event handler using the following code:

```
Product p = null;
            int prodId = 0;
if (jTextField6.getText().length() > 0) {
               try {
      prodId =
     Integer.parseInt(jTextField6.getText();
                         p =
DataMan.findProductById(prodId);
                order.addOrderItem(p.getId());
                jLabel10.setText(

Util.toMoney(order.getOrderTotal()));
                 }
            catch (Exception err) {
               String message = err.getMessage();
               if (err instanceof
               NumberFormatException) {
                 message = "Product id '" +
               message +
             "' is not a valid number";
               }
            JOptionPane.showMessageDialog(this,
     message,"Error", JOptionPane.ERROR_MESSAGE);
            jTextField6.setText("");
               }
            }
            else {
              JOptionPane.showMessageDialog(this,
```


Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

```
                "Please enter a Product Id",
            "Error",JOptionPane.ERROR_MESSAGE);
          }
                                        }
```

Note: As before, pay close attention to the preceding variables in italics to ensure that you have correctly identified the appropriate labels and text fields.

b. Compile and save the code. Run the **`OrderEntry`** application to test the code. Add products to the order (product IDs start at 2000). Did you see the products visually added to the list? If not, explain why. Did the order total get updated?

2. Modify the Order class to support the UI by replacing the `Vector` type for `items` to be a `javax.swing.DefaultListModel`. Provide a method in the `Order` class to return the reference to the model.

a. Modify **`Order.java`** class and replace the `items` declaration as shown:

```
// private ArrayList items = null;
```

replace with ...

```
private DefaultListModel jList1 = null;
```

Note: You will need to import **`javax.swing.DefaultListModel`**.

b. In the **`Order`** `no-arg` constructor, create the **`DefaultListModel`** object to initialize the **`items`** variable, instead of using a Vector, for example:

```
// items = new ArrayList(10);
jList1 = new DefaultListModel();
```

In the **`addOrderItem()`** method, comment out the following statement:

```
//items.add(item);
```

replace with…

```
jList1.addElement(item);
```

In the **`showOrder()`** method, comment out the **`for`** loop statements:

```
/*  for (Iterator it = items; ...) {
          ...
       System.out.println(item.toString());
}*/
```

c. Add a new method with the signature shown to return the `items` reference to the caller:

```
public DefaultListModel getModel() { … }
```

Note: This method will be used as the model for the `JList` causing it to dynamically display `OrderItem` objects as products are added to the order.

d. Modify `OrderEntryFrame` to add the call to use the method in the button.

```
jList1.setModel(order.getModel());
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 17-2: Adding Event Handling (continued)

e. Compile and save the changes to the **`Order`** class.

f. Save and compile **`OrderEntryFrame`**, and run the **`OrderEntry`** application to test if items are dynamically displayed in the list as they are added.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Practice 18: *Deploying Java Applications*

There is no practice for this lesson.

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

B

Java Language Quick-Reference Guide

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Console Output

Java applications and applets can output simple messages to the console as follows:

```
System.out.println("This is displayed on the console");
```

Data Types

`boolean`	Boolean type, can be `true` or `false`
`byte`	1-byte signed integer
`char`	Unicode character (i.e. 16 bits)
`short`	2-byte signed integer
`int`	4-byte signed integer
`long`	8-byte signed integer
`float`	Single-precision fraction, 6 significant figures
`double`	Double-precision fraction, 15 significant figures

Operators

+ - * / %	Arithmetic operators (% means *remainder*)
++ --	Increment or decrement by 1 result = ++i; means increment by 1 first result = i++; means do the assignment first
+= -= *= /= %= etc.	For example, i += 2 is equivalent to i = i + 2
&&	Logical AND. For example, if (i > 50 && i < 70) The second test is only carried out if necessary - use & if the second test should *always* be done
||	Logical OR. For example, if (i < 0 || i > 100) The second test is only carried out if necessary - use | if the second test should *always* be done
!	Logical NOT. For example, if (!endOfFile)
== != > >= < <=	Relational operators
& | ^ ~	Bitwise operators (AND, OR, XOR, NOT)
<< >> >>>	Bitwise shift operators (shift left, shift right with sign extension, shift right with 0 fill)
instanceof	Test whether an object is an instance of a class. For example, if (anObj instanceof BankAccount) System.out.println("$$$");

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Control Flow: `if … else`

`if` statements are formed as follows (the `else` clause is optional). The braces {} are necessary if the if-body exceeds one line; even if the if-body is just one line, the braces {} are worth having to aid readability:

```
String dayname;
…
if (dayname.equals("Sat") || dayname.equals("Sun")){
   System.out.println("Hooray for the weekend");
}
else if (dayname.equals("Mon")) {
   System.out.println("I don’t like Mondays");
}
else {
   System.out.println("Not long for the weekend!");
}
```

Control Flow: `switch`

`switch` is used to check an integer (or character) against a fixed list of alternative values:

```
int daynum;
…
switch (daynum) {
   case 0:
   case 6:
     System.out.println("Hooray for the weekend");
     break;

   case 1:
     System.out.println("I don’t like Mondays");
     break;

   default:
     System.out.println("Not long for the weekend!");
     break;
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Control Flow: Loops

Java contains three loop mechanisms:

```
int i = 0;
while (i < 100) {
  System.out.println("Next square is: " +  i*i);
  i++;
}
```

```
for (int i = 0; i < 100; i++) {
  System.out.println("Next square is: " +  i*i);
}
```

```
int positiveValue;
do {
  positiveValue = getNumFromUser();
}
while (positiveValue < 0);
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Defining Classes

When you define a class, you define the data attributes (usually `private`) and the methods (usually `public`) for a new data type. The class definition is placed in a `.java` file as follows:

```
// This file is Student.java. The class is declared
// public, so that it can be used anywhere in the program

public class Student {

   private String name;
   private int    numCourses = 0;

   // Constructor to initialize all the data members
   public Student(String n, int c) {
      name = n;
      numCourses = c;
   }
   // No-arg constructor, to initialize with defaults
   public Student() {
     this("Anon", 0);      // Call other constructor
   }
   // finalize() is called when obj is garbage collected
   public void finalize() {
     System.out.println("Goodbye to this object");
   }

   // Other methods
   public void attendCourse() {
     numCourses++;
   }
   public void cancelPlaceOnCourse() {
     numCourses--;
   }

   public boolean isEligibleForChampagne() {
     return (numCourses >= 3);
   }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Using Classes

To create an object and send messages to the object:

```
public class MyTestClass {

  public static void main(String[] args) {

    // Step 1 - Declare object references
    // These refer to null initially in this example
    Student me, you;


    // Step 2 - Create new Student objects
    me  = new Student("Andy", 0);
    you = new Student();


    // Step 3 - Use the Student objects
    me.attendCourse();
    you.attendCourse();


    if (me.isEligibleForChampagne())
      System.out.println("Thanks very much");

  }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Arrays

An array behaves like an object. Arrays are created and manipulated as follows:

```
// Step 1 - Declare a reference to an array
int[] squares;          // Could write int squares[];

// Step 2 - Create the array "object" itself
squares = new int[5]; // Creates array with 5 slots

// Step 3 - Initialize slots in the array
for (int i=0; i < squares.length; i++) {
   squares[i] = i * i;
   System.out.println(squares[i]);
}
```

Note that array elements start at [0], and that arrays have a length property that gives you the size of the array. If you inadvertently exceed an array's bounds, an exception is thrown at run time and the program aborts.

Note: Arrays can also be set up by using the following abbreviated syntax:

```
String[] cities = {
      "San Francisco",
      "Dallas",
      "Minneapolis",
      "New York",
      "Washington, D.C."
};
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Inheritance and Polymorphism

A class can inherit all of the data and methods from another class. Methods in the superclass can be overridden by the subclass. Any members of the superclass that you want to access in the subclass must be declared `protected`. The `protected` access specifier allows subclasses, plus any classes in the same package, to access that item.

```
public class Account {
    private double balance = 0.0;

    public Account(double initBal) {
        balance = initBal;
    }
    public void deposit(double amt) {
        balance += amt;
    }
    public void withdraw(double amt) {
        balance -= amt;
    }
    public void display() {
        System.out.println("Balance is: " + balance);
    }
}
public class CheckAccount extends Account {
    private int maxChecks = 0;
    private int numChecksWritten = 0;

    public CheckAccount(double initBal, int maxChk) {
        super(initBal);           // Call superclass ctor
        maxChecks = maxChk;       // Initialize our data
    }
    public void withdraw(double amt) {
        super.withdraw(amt);      // Call superclass
        numChecksWritten++;       // Increment chk. num.
    }
    public void display() {
        super.display();          // Call superclass
        System.out.println(numChecksWritten);
    }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Abstract Classes

An abstract class is one that can never be instantiated; in other words, you cannot create an object of such a class. Abstract classes are specified as follows:

```
// Abstract superclass
public abstract class Mammal {
   …
}


// Concrete subclasses
public class Cat extends Mammal {
   …
}

public class Dog extends Mammal {
   …
}

public class Mouse extends Mammal {
   …
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Abstract Methods

An abstract method is one that does not have a body in the superclass. Each concrete subclass is obliged to override the abstract method and provide an implementation; otherwise, the subclass is itself deemed abstract because it does not implement all its methods.

```
// Abstract superclass
public abstract class Mammal {

   // Declare some abstract methods
   public abstract void eat();
   public abstract void move();
   public abstract void reproduce();

   // Define some data members if you like
   private double weight;
   private int age;

   // Define some concrete methods too if you like
   public double getWeight{} {
     return weight;
   }

   public int getAge() {
     return age;
   }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Interfaces

An interface is similar to an abstract class with 100% abstract methods and no instance variables. An interface is defined as follows:

```
public interface Runnable {
   public void run();
}
```

A class can implement an interface as follows. The class is obliged to provide an implementation for every method specified in the interface; otherwise, the class must be declared `abstract` because it does not implement all its methods.

```
public class MyApp extends Applet implements Runnable {

   public void run() {
      // This is called when the Applet is kicked off
      // in a separate thread
      …
   }


   // Plus other applet methods
   …
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Static Variables

A `static` variable is like a global variable for a class. In other words, you get only one instance of the variable for the whole class, regardless of how many objects exist. `static` variables are declared in the class as follows:

```
public class Account {
  private String accnum;            // Instance var
  private double balance = 0.0;     // Instance var

  private static double intRate = 5.0;  // Class var

  …
}
```

Static Methods

A `static` method in a class is one that can access only `static` items; it cannot access any non-`static` data or methods. `static` methods are defined in the class as follows:

```
public class Account {

  public static void setIntRate(double newRate) {
    intRate = newRate;
  }

  public static double getIntRate() {
    return intRate;
  }
  …
}
```

To invoke a `static` method, use the name of the class as follows:

```
public class MyTestClass {

  public static void main(String[] args) {
    System.out.println("Interest rate is" +
                        Account.getIntRate());
  }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Packages

Related classes can be placed in a common package as follows:

```
// Car.java
package mycarpkg;

public class Car {
   …
}


// Engine.java
package mycarpkg;

public class Engine {
   …
}


// Transmission.java
package mycarpkg;

public class Transmission {
   …
}
```

Importing Packages

Anyone needing to use the classes in this package can import all or some of the classes in the package as follows:

```
import mycarpkg.*;    // import all classes in package
```

or

```
import mycarpkg.Car; // just import individual classes
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

The `final` Keyword

The `final` keyword can be used in three situations:

`final` classes (for example, the class cannot be inherited from)
`final` methods (for example, the method cannot be overridden in a subclass)
`final` variables (for example, the variable is constant and cannot be changed)

Here are some examples:

```
// final classes
public final class Color {
  …
}


// final methods
public class MySecurityClass {
  public final void validatePassword(String password) {
    …
  }
}


// final variables
public class MyTrigClass {
  public static final double PI = 3.1415;
  …
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Exception Handling

Exception handling is achieved through five keywords in Java:

`try` The block of code where statements that can cause an exception are placed

`catch` The block of code where error processing is placed

`finally` An optional block of code after a `try` block, for unconditional execution

`throw` The keyword that is used in the low-level code to generate or throw an exception

`throws` The keyword that specifies the list of exceptions that a method can throw

Here are some examples:

```
public class MyClass {
     public void anyMethod() {
             try {
                func1();
                func2();
                func3();
             }
             catch (IOException e) {
        System.out.println("IOException:" + e);
             }
             catch (MalformedURLException e) {

System.out.println("MalformedURLException:" + e);
             }
             finally {
System.out.println("This is always displayed");
             }
     }

     public void func1() throws IOException {
             …
     }
     public void func2() throws MalformedURLException {
             …
     }
     public void func3() throws IOException,
                           MalformedURLException {
             …
     }
}
```

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Global Business Solutions Peru SAC use only

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED