

# Working with Arrays, Loops, and Dates

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Kevin Gómez González (kev.gomez97@hotmail.com) has a non-transferable license to use this Student Guide.

## Interactive Quizzes

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Before you start today's lessons, test your knowledge by answering some quiz questions that relate to yesterday's lessons. Open your quiz file from labs > Quizzes > Java SE 8 Fundamentals Quiz.html. Click the links for the lessons titled "Using Encapsulation" and "More on Conditionals."

## Objectives

After completing this lesson, you should be able to:

- Create a `java.time.LocalDateTime` object to show the current date and time
- Parse the `args` array of the `main` method
- Correctly declare and instantiate a two-dimensional array
- Code a nested `while` loop to achieve the desired result
- Use a nested `for` loop to process a two-dimensional array
- Code a `do/while` loop to achieve the desired result
- Use an `ArrayList` to store and manipulate lists of Objects
- Evaluate and select the best type of loop to use for a given programming requirement



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The first topic, “Working with dates,” focuses on the new Date Time API. This is a new feature of Java SE 8.

## Displaying a Date

```
LocalDate myDate = LocalDate.now();  
System.out.println("Today's date: " + myDate);
```

Output: 2013-12-20

- `LocalDate` belongs to the package `java.time`.
- The `now` method returns today's date.
- This example uses the default format for the default time zone.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `now` static method returns an object of type `LocalDate`. Of course, `System.out.println` calls the `toString` method of the `LocalDate` object. Its `String` representation is `2013-12-20` in this example.

## Class Names and the Import Statement

- Date classes are in the package `java.time`.
- To refer to one of these classes in your code, you can fully qualify `java.time.LocalDate` or, add the import statement at the top of the class.

```
import java.time.LocalDate;  
public class DateExample {  
    public static void main (String[] args) {  
        LocalDate myDate;  
    }  
}
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

The `import` statement can be:

- For just the class in question  
`java.time.LocalDate;`
- For all classes in the package  
`java.time.*;`

## Working with Dates

`java.time`

- Main package for date and time classes

`java.time.format`

- Contains classes with static methods that you can use to format dates and times

Some notable classes:

- `java.time.LocalDate`
- `java.time.LocalDateTime`
- `java.time.LocalTime`
- `java.time.format.DateTimeFormatter`

Formatting example:

```
myDate.format(DateTimeFormatter.ISO_LOCAL_DATE);
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The Java API has a `java.time` package that offers many options for working with dates and times. A few notable classes are:

- `java.time.LocalDate` is an immutable date-time object that represents a date, often viewed as year-month-day. Other date fields, such as day-of-year, day-of-week, and week-of-year, can also be accessed. For example, the value “2nd October 2007” can be stored in a `LocalDate`.
- `java.time.LocalDateTime` is an immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second. Other date and time fields, such as day-of-year, day-of-week, and week-of-year, can also be accessed. Time is represented to nanosecond precision. For example, the value “2nd October 2007 at 13:45.30.123456789” can be stored in a `LocalDateTime`.
- `java.time.LocalTime` is an immutable date-time object that represents a time, often viewed as hour-minute-second. Time is represented to nanosecond precision. For example, the value “13:45.30.123456789” can be stored in a `LocalTime`. It does not store or represent a date or time-zone. Instead, it is a description of the local time as seen on a wall clock. It cannot represent an instant on the time-line without additional information such as an offset or time zone.

- `java.time.format.DateTimeFormatter` provides static and nonstatic methods to format dates using a specific format style. It also provides static constants (variables) that represent specific formats.
- The example in the slide uses a static constant variable, `ISO_LOCAL_DATE`, from the `DateTimeFormatter` class. It is passed as an argument into the `format` method of `Date` object:

```
myDate.format(DateTimeFormatter.ISO_LOCAL_DATE)
```

- A formatted `String` representing the `LocalDateTime` is returned from the `format` method. For a more complete discussion of date formatting, see “Some Methods of `LocalDate`” later in the lesson.



## Working with Different Calendars

- The default calendar is based on the Gregorian calendar.
- If you need non-Gregorian type dates:
  - Use the `java.time.chrono` classes
  - They have conversion methods.
- Example: Convert a `LocalDate` to a Japanese date:

```
LocalDate myDate = LocalDate.now();  
JapaneseDate jDate = JapaneseDate.from(myDate);  
System.out.println("Japanese date: " + jDate);
```

- Output:  
Japanese date: Japanese Heisei 26-01-16



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the above example, `JapaneseDate` is a class belonging to the `java.time.chrono` package. `myDate` is passed to the static `from` method, which returns a `JapaneseDate` object (`jDate`). The result of printing the `jDate` object is shown as output.

## Some Methods of LocalDate

LocalDate overview: A few notable methods and fields

- Instance methods:
  - `myDate.minusMonths (15);`
  - `myDate.plusDays (8);`
    - `(long monthsToSubtract)`
    - `(long daysToAdd)`
- Static methods:
  - `of(int year, Month month, int dayOfMonth)`
  - `parse(CharSequence text, DateTimeFormatter formatter)`
  - `now()`

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

LocalDate has many methods and fields. Here are just a few of the instance and static methods that you might use:

- `minusMonths` returns a copy of this `LocalDate` with the specified period in months subtracted.
- `plusDays` returns a copy of this `LocalDate` with the specified number of days added.
- `of(int year, Month month, int dayOfMonth)` obtains an instance of `LocalDate` from a year, month, and day.
- `parse(CharSequence text, DateTimeFormatter formatter)` obtains an instance of `LocalDate` from a text string using a specific formatter.

Read the `LocalDate` API reference for more details.

## Formatting Dates

```

1 LocalDateTime today = LocalDateTime.now();
2 System.out.println("Today's date time (no formatting): "
3     + today);
4
5                                     Format the date in
6 String sdate = today.format(DateTimeFormatter.ISO_DATE_TIME);
7                                     standard ISO format.
8 System.out.println("Date in ISO_DATE_TIME format: "
9     + sdate);
10
11                                     Localized date time in
12 String fdate = today.format(DateTimeFormatter.ofLocalizedDateTime
13                                     Medium format
14     (FormatStyle.MEDIUM));
15 System.out.println("Formatted with MEDIUM FormatStyle: "
16     + fdate);

```

Output:

Today's date time (no formatting):	2013-12-23T16:51:49.458
Date in ISO_DATE_TIME format:	2013-12-23T16:51:49.458
Formatted with MEDIUM FormatStyle:	Dec 23, 2013 4:51:49 PM

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The code example in the slide shows you some options for formatting the output of your dates.

- **Line 1:** Get a `LocalDateTime` object that reflects today's date.
- **Lines 6 - 7:** Get a `String` that shows the date object formatted in standard `ISO_DATE_TIME` format. As you see in the output, the default format when you just print the `LocalDateTime` object uses the same format.
- **Lines 11 - 12:** Call the `ofLocalizedDateTime` method of the `DateTimeFormatter` to get a `String` representing the date in a medium localized date-time format. The third line of the output shows this shorter version of the date.

## Exercise 11-1: Declare a LocalDateTime Object

1. Declare and initialize a `LocalDateTime` object.
2. Print and format the `OrderDate`.
3. Run the code.

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- Open the Java Code Console and access 11-ArraysLoopsDates > Exercise1.
- Follow the instructions below the code editor.
- If you need help, click the Solution link. To go back to your code, click the Exercise link again. Any changes that you have made will have been saved.

## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Using the args Array in the main Method

- Parameters can be typed on the command line:

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

*Goes into args[1]*  
*Goes into args[0]*

- Code for retrieving the parameters:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

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

NetBeans does not allow you a way to run a Java class from the command line, but you can set command-line arguments as a property of the NetBeans project.

## Converting String Arguments to Other Types

- Numbers can be typed as parameters:

```
> java ArgsTest 2 3
```

```
Total is: 23
```

```
Total is: 5
```

Concatenation, not addition!

- Conversion of String to int:

```
public class ArgsTest {
    public static void main (String[] args) {
        System.out.println("Total is:"+(args[0]+args[1]));
        int arg1 = Integer.parseInt(args[0]);
        int arg2 = Integer.parseInt(args[1]);
        System.out.println("Total is: " + (arg1+arg2));
    }
}
```

Strings

Note the parentheses.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `main` method treats everything you type as a literal string. If you want to use the string representation of a number in an expression, you must convert the string to its numerical equivalent.

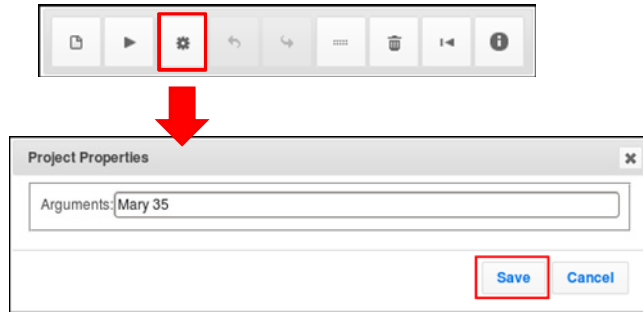
The `parseInt` static method of the `Integer` class is used to convert the `String` representation of each number to an `int` so they can be added.

Note that the parentheses around `arg1 + arg2` are required so that the `+` sign indicates addition rather than concatenation. The `System.out.println` method converts any argument passed to it to a `String`. We want it to add the numbers first, and *then* convert the total to a `String`.

## Exercise 11-2: Parsing the `args` Array

In this exercise, you parse the `args` array in the `main` method to get the command-line arguments and assign them to local variables.

- To enter command-line arguments, click the Configure button and enter the values, separated by a space.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- In the Java Code Console, access Lessons > 11-ArraysLoopsDates > Exercise2.
- Follow the instructions below the code editor to parse the `args` array in the `main` method, saving the arguments to local variables and then printing them.
- Enter command arguments as described in the slide above.
- Run the file to test your code.
- If you need help, click the Solution link. To go back to your code, click the Exercise link again. Any changes that you have made will have been saved.



## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Describing Two-Dimensional Arrays

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

## Declaring a Two-Dimensional Array

Example:

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

Syntax:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

where:

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

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

## Instantiating a Two-Dimensional Array

Example:

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

Syntax:

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

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

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The syntax for instantiating a two-dimensional array is:

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

where:

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

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

## Initializing a Two-Dimensional Array

Example:

```
int[][] yearlySales = new int[5][4];
yearlySales[0][0] = 1000;
yearlySales[0][1] = 1500;
yearlySales[0][2] = 1800;
yearlySales[1][0] = 1000;
yearlySales[3][3] = 2000;
```

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

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

**Note:** When you choose to draw a chart based on a 2D array, the way you orient the chart is arbitrary. That is, you have the option to decide if you would like to draw a chart corresponding to `array2DName[x][y]` or `array2DName[y][x]`.



## Quiz

A two-dimensional array is similar to a \_\_\_\_\_.

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Some New Types of Loops

Loops are frequently used in programs to repeat blocks of code while some condition is true.

There are three main types of loops:

- A `while` loop repeats *while* an expression is true.
- A `for` loop simply repeats a *set number* of times.
  - \* A variation of this is the **enhanced** `for` loop. This loops through the elements of an array.
- A `do/while` loop executes once and then continues to repeat *while* an expression is true.

*\*You have already learned this one!*

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Up to this point, you have been using the enhanced for loop, which repeats a block of code for each element of an array.

Now you can learn about the other types of loops as described above.



## Repeating Behavior



```
while (!areWeThereYet) {  
    read book;  
    argue with sibling;  
    ask, "Are we there yet?";  
}  
  
Woohoo!;  
Get out of car;
```

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A common requirement in a program is to repeat a number of statements. Typically, the code continues to repeat the statements until something changes. Then the code breaks out of the loop and continues with the next statement.

The pseudocode example above, shows a `while` loop that loops until the `areWeThereYet` boolean is true.

## A while Loop Example

```

01 public class Elevator {
02     public int currentFloor = 1;
03
04     public void changeFloor(int targetFloor){
05         while (currentFloor != targetFloor){
06             if(currentFloor < targetFloor)
07                 goUp();
08             else
09                 goDown();
10         }
11     }

```

Boolean expression

Body of the loop



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

- The goUp and goDown methods increment or decrement the currentFloor variable.
- The boolean expression returns true if currentFloor is not equal to targetFloor. When these two variables are equal, this expression returns false (because the elevator is now at the desired floor), and the body of the while loop is not executed.

## Coding a `while` Loop

Syntax:

```
while (boolean_expression) {  
    code_block;  
}
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `while` loop first evaluates a boolean expression and, while that value is true, it repeats the code block. To avoid an infinite loop, you need to be sure that something will cause the boolean expression to return false eventually. This is frequently handled by some logic in the code block itself.

## while Loop with Counter

```
01 System.out.println("/*");
02 int counter = 0;
03 while (counter < 3) {
04     System.out.println(" *");
05     counter++;
06 }
07 System.out.println("*/");
```

Output:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

Although this works, the standard `for` loop is ideal for this purpose.

## Coding a Standard `for` Loop

The standard `for` loop repeats its code block for a set number of times using a counter.

Example:

```
01 for(int i = 1; i < 5; i++){
02     System.out.print("i = " + i + "; ");
03 }
```

Output: i = 1; i = 2; i = 3; i = 4;

Syntax:

```
01 for (<type> counter = n;
02     <boolean_expression>;
03     <counter_increment>){
04     code_block;
05 }
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The three essential elements of a standard `for` loop are the counter, the boolean expression, and the increment. All of these are expressed within parentheses following the keyword `for`.

1. A counter is declared and initialized as the first parameter of the `for` loop. (`int i = 1`)
2. A boolean expression is the second parameter. It determines the number of loop iterations. (`i < 5`)
3. The counter increment is defined as the third parameter. (`i++`)

The code block (shown on line 2) is executed in each iteration of the loop. At the end of the code block, the counter is incremented by the amount indicated in the third parameter.

As you can see, in the output shown above, the loop iterates four times.

## Standard for Loop Compared to a while loop

while loop

```
01  int i = 0;
02  while (i < 3) {
03      System.out.println(" *");
04      i++;
05  }
```

Initialize  
counter

Increment  
counter

for loop

```
01  for (int num = 0; num < 3; num++) {
02      System.out.println(" *");
03  }
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In this slide, you see a `while` loop example at the top of the slide. At the bottom, you see the same logic implemented using a standard `for` loop.

The three essential elements of a `while` loop are also present in the `for` loop, but in different places.

1. The counter (`i`) is declared and initialized outside the `while` loop on line 1.
2. The counter is incremented in the `while` loop on line 4.
3. The boolean expression that determines the number of loop iterations is within the parentheses for the `while` loop on line 2.
4. In the `for` loop, all three elements occur within the parentheses as indicated in the slide.

The output for each statement is the same.

## Standard for Loop Compared to an Enhanced for Loop

### Enhanced for loop

```
01 for(String name: names){
02     System.out.println(name);
03 }
```

### Standard for loop

```
01 for (int idx = 0; idx < names.length; idx++){
02     System.out.println(names[idx]);
03 }
```

boolean expression

Counter used as the  
index of the array

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This slide compares the standard `for` loop to the enhanced `for` loop that you learned about in the lesson titled “Managing Multiple Items.” The examples here show each type of `for` loop used to iterate through an array. Enhanced `for` loops are used only to process arrays, but standard `for` loops can be used in many ways.

- **The enhanced for loop example:** A `String` variable, `name`, is declared to hold each element of the array. Following the colon, the `names` variable is a reference to the array to be processed. The code block is executed as many times as there are elements in the array.
- **The standard for loop example:** A counter, `idx`, is declared and initialized to 0. A boolean expression compares `idx` with the `length` of the `names` array. If `idx < names.length`, the code block is executed. `idx` is incremented by one at the end of each code block.
- Within the code block, `idx` is used as the array index.

The output for each statement is the same.

## do/while Loop to Find the Factorial Value of a Number

```

1 // Finds the product of a number and all integers below it
2 static void factorial(int target){
3     int save = target;
4     int fact = 1;
5     do {
6         fact *= target--;
7     }while(target > 0);
8     System.out.println("Factorial for "+save+": "+ fact);
9 }

```

Executed once before evaluating the condition

Outputs for two different targets:

Factorial value for 5: 120

Factorial value for 6: 720

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The do/while loop is a slight variation on the while loop.

The example above shows a do/while loop that determines the factorial value of a number, called target. The factorial value is the product of an integer, multiplied by each positive integer smaller than itself. For example if the target parameter is 5, this method multiplies 1 \* 5 \* 4 \* 3 \* 2 \* 1 resulting in a factorial value of 120.

do/while loops are not used as often as while loops. The code above could be rewritten as a while loop like this:

```

while (target > 0) {
    fact *= target--;
}


```

The decision to use a do/while loop instead of a while loop usually relates to code readability.



## Coding a do/while Loop

Syntax:

```
do {  
    code_block;  This block executes at least once.  
}  
while (boolean_expression); // Semicolon is mandatory.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In a do/while loop, the condition (shown at the bottom of the loop) is evaluated *after* the code block has already been executed once. If the condition evaluates to true, the code block is repeated continually until the condition returns false.

The *body of the loop* is, therefore, processed at least once. If you want the statement or statements in the body to be processed at least once, use a do/while loop instead of a while or for loop.

## Comparing Loop Constructs

- Use the `while` loop to iterate indefinitely through statements and to perform the statements zero or more times.
- Use the standard `for` loop to step through statements a predefined number of times.
- Use the enhanced `for` loop to iterate through the elements of an Array or ArrayList (discussed later).
- Use the `do/while` loop to iterate indefinitely through statements and to perform the statements *one* or more times.

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

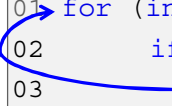
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## The `continue` Keyword

There are two keywords that enable you to interrupt the iterations in a loop of any type:

- `break` causes the loop to exit. \*
- `continue` causes the loop to skip the current iteration and go to the next.

```
01 for (int idx = 0; idx < names.length; idx++) {  
02     if (names[idx].equalsIgnoreCase("Unavailable"))  
03         continue;  
04     System.out.println(names[idx]);  
05 }
```



\* Or any block of code to exit

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- `break` allows you to terminate an execution of a loop or switch and skip to the first line of code following the end of the relevant loop or switch block.
- `continue` is used only within a loop. It causes the loop to skip the current iteration and move on to the next. This is shown in the code example above. The `for` loop iterates through the elements of the `names` array. If it encounters an element value of "Unavailable", it does not print out that value, but skips to the next array element.

## Exercise 11-3: Processing an Array of Items

In this exercise you:

- Process an array of items to calculate the Shopping Cart total
- Skip any items that are back ordered
- Display the total

```

1 package ex11_2_exercise;
2
3 public class ShoppingCart {
4     Item[] items = {new Item("Shirt",25.60),
5                     new Item("WristBand",1.00),
6                     new Item("Pants",35.99)};
7
8     public static void main(String[] args){
9         ShoppingCart cart = new ShoppingCart();
10        cart.displayTotal();
11    }
12
13    // Use a standard for loop to iterate through the items array, adding up the total price
14    // Skip any items that are back ordered. Display the Shopping Cart total.
15    public void displayTotal(){
16    }
17
18 }
19
20

```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- In the Java Code Console, access Lessons > 11-ArraysLoopsDates > Exercise3.
- Follow the instructions below the code editor. Code the `displayTotal` method of the `ShoppingCart` class so that it will iterate through the items array and print out the total for the Shopping Cart.
- Skip any items that are back ordered. (You might want to examine the `Item` class to see how this setting is determined.)
- If you need help, click the Solution link. To go back to your code, click the Exercise link again. Any changes that you have made will have been saved.

## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Nesting Loops

All types of loops can be nested within the body of another loop. This is a powerful construct used to:

- Process multidimensional arrays
- Sort or manipulate large amounts of data



How it works:

1<sup>st</sup> iteration of outer loop triggers:

Inner loop

2<sup>nd</sup> iteration of outer loop triggers:

Inner loop

3<sup>rd</sup> iteration of outer loop triggers:

Inner loop

and so on...

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Nested for Loop

Example: Print a table with 4 rows and 10 columns:

```
01 int height = 4, width = 10;
02
03 for(int row = 0; row < height; row++){
04     for (int col = 0; col < width; col++){
05         System.out.print("@");
06     }
07     System.out.println();
08 }
```

Output:

```
run:
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
@@@@@@@@@@
BUILD SUCCESSFUL (total time: 0 seconds)
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The code in the slide shows a simple nested loop to output a block of @ symbols with height and width given in the initial local variables.

- The outer `for` loop produces the rows. It loops four times.
- The inner `for` loop prints the columns for a given row. It loops ten times.
- Notice how the outer loop prints a new line to start a new row, whereas the inner loop uses the `print` method of `System.out` to print an @ symbol for every column. (Remember that, unlike `println`, `print` does not generate a new line.)
- The output is shown at the bottom: a table containing four rows of ten columns.

## Nested while Loop

Example:

```

01 String name = "Lenny";
02 String guess = "";
03 int attempts = 0;
04 while (!guess.equalsIgnoreCase(name)) {
05     guess = "";
06     while (guess.length() < name.length()) {
07         char asciiChar = (char) (Math.random() * 26 + 97);
08         guess += asciiChar;
09     }
10     attempts++;
11 }
12 System.out.println(name+" found after "+attempts+" tries!");

```

Output:

Lenny found after 20852023 tries!

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

- Looking at the inner loop first, the code initializes `char asciiChar` to a lowercase letter randomly. These `chars` are then added to `String guess`, until that `String` is as long as the `String` that it is being matched against. Notice the convenience of the concatenation operator here, allowing concatenation of a `String` and a `char`.
- The outer loop tests to see whether `guess` is the same as a lowercase version of the original name. If it is not, `guess` is reset to an empty `String` and the inner loop runs again, usually millions of times for a five-letter name. (Note that names longer than five letters will take a very long time.)



## Processing a Two-Dimensional Array

### Example: Quarterly Sales per Year

```

01  int sales[] [] = new int[3][4];
02  initArray(sales); //initialize the array
03  System.out.println
04      ("Yearly sales by quarter beginning 2010:");
05  for(int i=0; i < sales.length; i++){
06      for(int j=0; j < sales[i].length; j++){
07          System.out.println("\tQ"+(j+1)+" "+sales[i][j]);
08      }
09      System.out.println();
10  }

```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This example illustrates the process of a two-dimensional array called `sales`. The `sales` array has three rows of four columns. The rows represent years and the columns represent the quarters of the year.

- The `initArray` method is called on line 2 to initialize the array with values.
- An opening message is printed on lines 3 and 4.
- On line 5, you see the outer `for` loop defined. The outer loop iterates through the three years. Notice that `i` is used as the counter for the outer loop.
- On line 6, you see the inner `for` loop defined. The inner loop iterates through each quarter of the year. It uses `j` as a counter. For each quarter of the year, it prints the quarter number (Q1, Q2, ...) plus the quarterly sales value in the element of the array indicated by the row number (`i`) and the column number (`j`).

**Note:** The “\t” character combination creates a tab indent.

- Notice that the quarter number is calculated as `j+1`. Because array elements start with 0, the index number of the first element will be 0, the second element index will be 1, and so on. To translate this into a quarter number, you add 1 to it.
- The output can be seen in the next slide.

## Output from Previous Example

Yearly sales by quarter beginning 2010:

Q1 36631

Q2 62699

Q3 60795

Q4 11975

Q1 72535

Q2 37363

Q3 20527

Q4 36670

Q1 3195

Q2 98608

Q3 21433

Q4 98519

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Kevin Gómez González (kev.gomez97@hotmail.com) has a non-transferable license to use this Student Guide.



## Quiz

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

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

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

**Answer: c**



## Quiz

Which of the following loops always executes at least once?

- a. The `while` loop
- b. The nested `while` loop
- c. The `do/while` loop
- d. The `for` loop

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

Kevin Gómez González (kev.gomez97@hotmail.com) has a non-transferable license to use this Student Guide.

## Topics

- Working with dates
- Parsing the `args` array
- Two-dimensional arrays
- Alternate looping constructs
- Nesting loops
- The `ArrayList` class

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## ArrayList Class

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

- `ArrayList` is one of several list management classes.
- It has a set of useful methods for managing its elements:
  - `add`, `get`, `remove`, `indexOf`, and many others
- It can store *only objects*, not primitives.
  - Example: an `ArrayList` of `Shirt` objects:
    - `shirts.add(shirt04);`
  - Example: an `ArrayList` of `String` objects:
    - `names.remove("James");`
  - Example: an `ArrayList` of ages:
    - `ages.add(5)` //NOT ALLOWED!
    - `ages.add(new Integer(5))` // OK



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `ArrayList` is one of several list management classes included in the `java.util` package. The other classes of this package (often referred to as the “collections framework”) are covered in greater depth in the *Java SE 8 Programming* course.

- `ArrayList` is based on the `Array` object and has many useful methods for managing the elements. Some of these are listed above, and you will see examples of how to use them in an upcoming slide.
- An important thing to remember about `ArrayList` variables is that you cannot store primitive types (`int`, `double`, `boolean`, `char`, and so on) in them—only object types. If you need to store a list of primitives, use an `Array`, or store the primitive values in the corresponding object type as shown in the final example above.

## Benefits of the ArrayList Class

- Dynamically resizes:
  - An ArrayList grows as you add elements.
  - An ArrayList shrinks as you remove elements.
  - You can specify an initial capacity, but it is not mandatory.

- Option to designate the object type it contains:

```
ArrayList<String> states = new ArrayList();
```

*Contains only String objects*

- Call methods on an ArrayList or its elements:

```
states.size(); //Size of list
```

```
states.get(49).length(); //Length of 49th element
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

For lists that are very dynamic, the ArrayList offers significant benefits such as:

- ArrayList objects dynamically allocate space as needed. This can free you from having to write code to:
  - Keep track of the index of the last piece of data added
  - Keep track of how full the array is and determine whether it needs to be resized
  - Increase the size of the array by creating a new one and copying the elements from the current one into it
- When you declare an ArrayList, you have the option of specifying the object type that will be stored in it using the diamond operator (<>). This technique is called “generics.” This means that when accessing an element, the compiler already knows what type it is. Many of the classes included in the collections framework support the use of generics.
- You may call methods on either the ArrayList or its individual elements.
  - Assume that all 50 US states have already been added to the list.
  - The examples show how to get the size of the list, or call a method on an individual element (such as the length of a String object).

## Importing and Declaring an ArrayList

- You must **import** `java.util.ArrayList` to use an `ArrayList`.
- An `ArrayList` may contain any object type, including a type that you have created by writing a class.

```
import java.util.ArrayList;
```

```
public class ArrayListExample {  
    public static void main (String[] args) {  
        ArrayList<Shirt> myList;  
    }  
}
```

*You may specify any object type.*

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- If you forget to import `java.util.ArrayList`, NetBeans will complain but also correctly suggest that you add the **import** statement.
- In the example above, the `myList` `ArrayList` will contain `Shirt` objects. You may declare that an array list contains any type of object.



## Working with an ArrayList

```

01 ArrayList<String> names;
02 names = new ArrayList();
03
04 names.add("Jamie");
05 names.add("Gustav");
06 names.add("Alisa");
07 names.add("Jose");
08 names.add(2, "Prashant");
09
10 names.remove(0);
11 names.remove(names.size() - 1);
12 names.remove("Gustav");
13
14 System.out.println(names);

```

Declare an ArrayList of Strings.  
 Instantiate the ArrayList.  
 Initialize it.  
 Modify it.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Declaring an ArrayList, you use the diamond operator (<>) to indicate the object type. In the example above:

- Declaration of the names ArrayList occurs on line 1.
- Instantiation occurs on line 2.

There are a number of methods to add data to the ArrayList. This example uses the add method, to add several String objects to the list. In line 8, it uses an overloaded add method that inserts an element at a specific location:

add(int index, E element).

There are also many methods available for manipulating the data. The example here shows just one method, but it is very powerful.

- remove(0) removes the first element (in this case, "Jamie").
- remove(names.size() - 1) removes the last element, which would be "Jose".
- remove("Gustav") removes an element that matches a specific value.
- You can pass the ArrayList to System.out.println. The resulting output is:  
[Prashant, Alisa]

## Exercise 11-4: Working with an ArrayList

In this exercise, you:

- Declare, instantiate, and initialize an `ArrayList` of `Strings`
- Use two different `add` methods
  - `add(E element)`
  - `add(int index, E element)`
- Use the following methods to find and remove a specific element if it exists
  - `contains (Object element)`
  - `remove (Object element)`



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- In the Java Code Console, access Lessons > 11-ArraysLoopsDates > Exercise4.
- Follow the instructions below the code editor. Initialize and manipulate an `ArrayList`, using various methods of the `ArrayList` class.
- Note that, in this case, the `ArrayList` contains `String` objects, so the `Object` reference in the above method signatures can be a `String` literal.
- If you need help, click the Solution link. To go back to your code, click the Exercise link again. Any changes that you have made will have been saved.

## Summary

In this lesson, you should have learned how to:

- Create a `java.time.LocalDateTime` object to show the current date and time
- Parse the `args` array of the `main` method
- Nest a `while` loop
- Develop and nest a `for` loop
- Code and nest a `do/while` loop
- Use an `ArrayList` to store and manipulate objects



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Play Time!

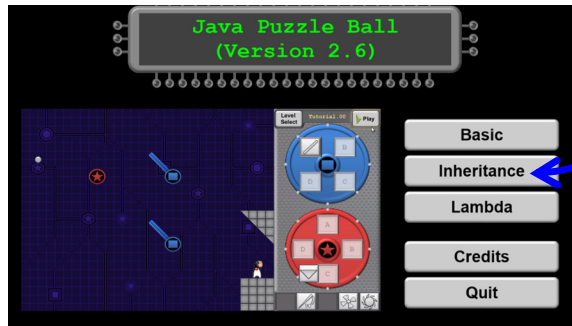


Play **Inheritance Puzzles 1, 2, and 3** before the next lesson titled “Using Inheritance.”

Consider the following:

What is inheritance?

Why are these considered “Inheritance” puzzles?



Play this  
game-mode.

ORACLE

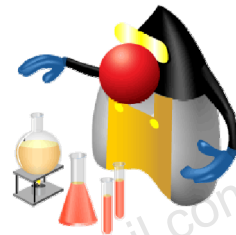
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You will be asked this question in the lesson titled “Using Inheritance.”

## Practice 11-1 Overview: Iterating Through Data

This practice covers the following topics:

- Converting a comma-separated list of names to an array of names
- Processing the array using a `for` loop
- Using a nested loop to populate an `ArrayList` of games

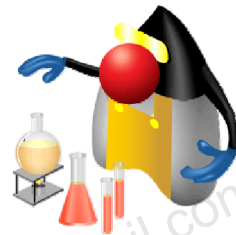


ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

## Practice 11-2 Overview: Working with `LocalDateTime`

This practice covers working with a few classes and methods from the `java.time` package in order to show date information for games played.

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.