11

Working with Arrays, Loops, and Dates

ORACLE

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
- 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

Topics

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

Displaying a Date

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

Output: 2018-12-20

- `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.

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;
    }
}
```

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);
```

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`

Some Methods of `LocalDate`

`LocalDate` overview: A few notable methods and fields

- Instance methods:
  - `myDate.minusMonths (15);` ——— `(long monthsToSubtract)`
  - `myDate.plusDays (8);` ——— `(long daysToAdd)`

- Static methods:
  - `of(int year, Month month, int dayOfMonth)`
  - `parse(CharSequence text, DateTimeFormatter formatter)`
  - `now()`

Formatting Dates

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

Format the date in standard ISO format.

Localized date time in Medium format

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
```

Exercise 11-1: Declare a `LocalDateTime` Object

1. Open the project **`Exercise_11-1`** or create your own project with a Java Main Class named `TestClass`.
2. Declare a `LocalDateTime` object to hold the order date.
3. Initialize the object to the current date and time by using the now() static method of the class.
4. Print the `orderDate` object with a suitable label.
5. Format `orderDate` by using the `ISO_LOCAL_DATE` static constant field of the `DateTimeFormatter` class.
6. Add the necessary package imports.
7. Print the formatted `orderDate` with a suitable label.

Topics

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

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[0]

Goes into args[1]

- 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]);
    }
}
```

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`:

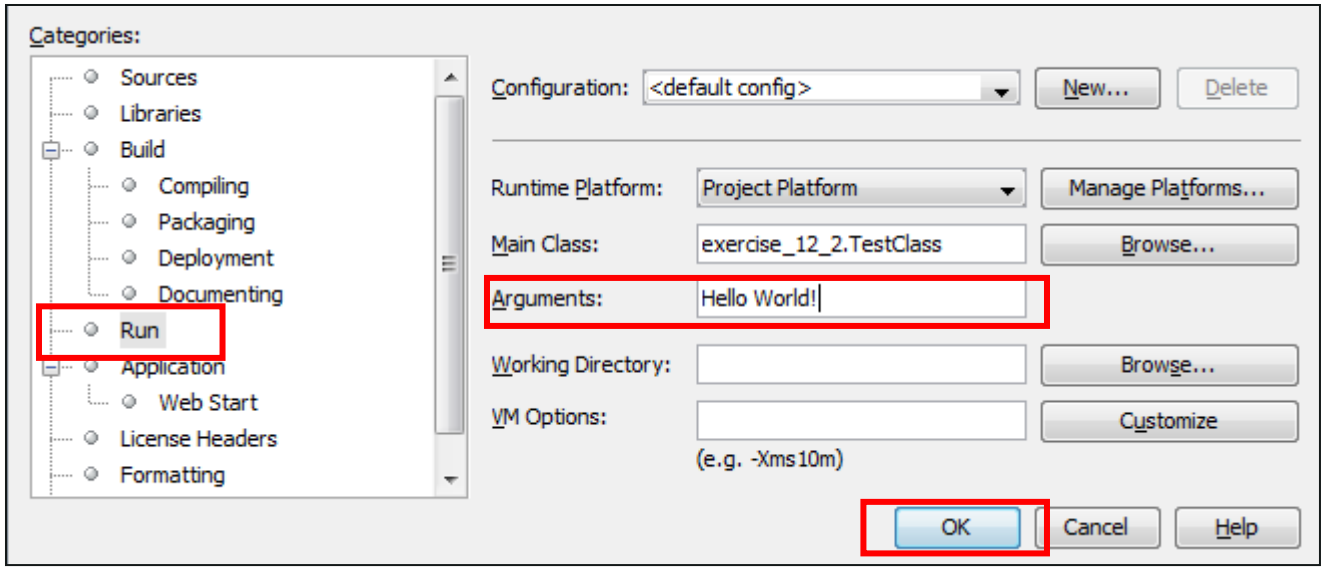
```
public class ArgsTest {
   public static void main (String[] args) {
     System.out.println("Total is:"+(args[0]+args[1]));
      int arg1 = Integer.parseInt(args[0]);
      int arg2 = Integer.parseInt(args[1]);
      System.out.println("Total is: " + (arg1+arg2));
   }
}
```

Strings

Note the parentheses.

Pass Arguments to the `args` Array in NetBeans

1. Right-click on your project.
2. Select **Properties**.
3. Select **Run**.
4. Type your arguments into the **Arguments** field.
   - Separate each argument with a space, not a comma.
5. Click **OK**.



args[0] is Hello
args[1] is World!

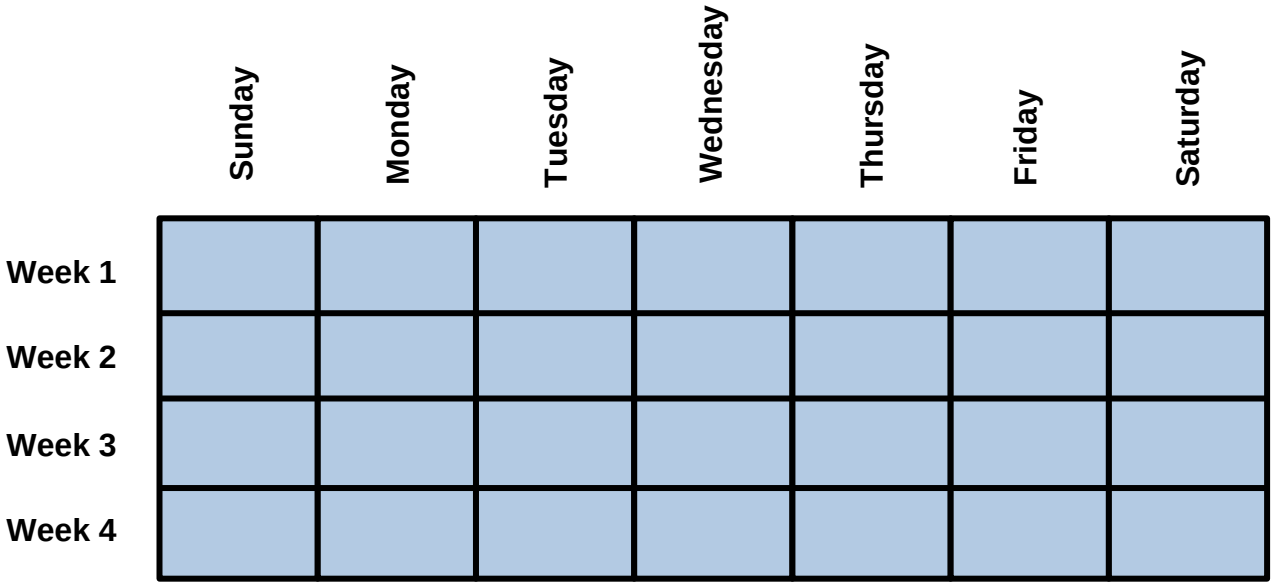
Exercise 11-2: Parsing the `args` Array

1. Open the project **`Exercise_11-2`** or create your own project with a **Java Main Class** named `TestClass`.
2. Parse the `args` array to populate name and age.
   - If `args` contains fewer than two elements, print a message telling the user that two arguments are required.
   - Remember that the age argument will have to be converted to an `int`.
     - **Hint:** Use a static method of the Integer class to convert it.
3. Print the `name` and `age` values with a suitable label.

Topics

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

Describing Two-Dimensional Arrays



Declaring a Two-Dimensional Array

Syntax:

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

Example:

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

Instantiating a Two-Dimensional Array

Syntax:

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

Example:

```
// Instantiates a 2D array: 5 arrays of 4 elements each

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

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

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				

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

Topics

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

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!*

Repeating Behavior


```
while (!areWeThereYet) {

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

}

Woohoo!;
Get out of car;
```

Coding a `while` Loop

Syntax:

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

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

`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:

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

Coding a Standard `for` Loop

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

- Syntax:

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

- 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;

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      }
```

boolean expression

Initialize counter

Increment counter

`for` loop

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

boolean expression

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

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
```

Coding a `do/while` Loop

Syntax:

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

This block executes at least once.

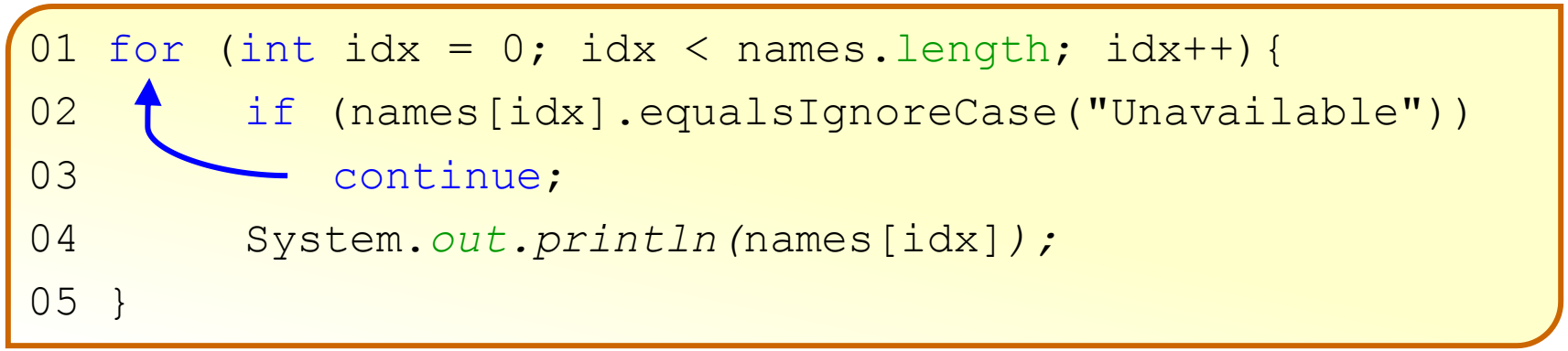
Comparing Loop Constructs

- 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 `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.



** Or any block of code to exit*

Exercise 11-3: Processing an Array of Items

1. Open the project **`Exercise_11-3`** in NetBeans:

In the `ShoppingCart` class:

2. Code the `displayTotal` method. Use a standard `for` loop to iterate through the items array.
3. If the current item is out of stock (call the `isOutOfStock` method of the item), skip to the next loop iteration.
4. If it is not out of stock, add the item price to a total variable that you declare and initialize before the `for` loop.
5. Print the Shopping Cart total with a suitable label.

Topics

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

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:

1st iteration of outer loop triggers:
    Inner loop
2nd iteration of outer loop triggers:
    Inner loop
3rd iteration of outer loop triggers:
    Inner loop
and so on…

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)
```

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!
```

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  }
```

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
```

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

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

Topics

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

`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`

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
```

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.

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.

Exercise 11-4: Working with an `ArrayList`

1. Open the project **`Exercise_11-4`**.
2. Create a String `ArrayList` with at least three elements.
   - Be sure to add the correct import statement.
   - Print the `ArrayList` and test your code.
3. Add a new element to the middle of the list.
   - **Hint:** Use the overloaded `add` method that takes an index number as one of the arguments.
   - Print the list again to see the effect.
4. Test for a particular value in the ArrayList and remove it.
   - **Hint:** Use the `contains` method. It returns a boolean and takes a single argument as the search criterion.
   - Print the list again.

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

Practices Overview

- 11-1: Iterating Through Data
- 11-2: Working with `LocalDateTime`