5

Managing Multiple Items

ORACLE

Objectives

After completing this lesson, you should be able to:

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

Topics

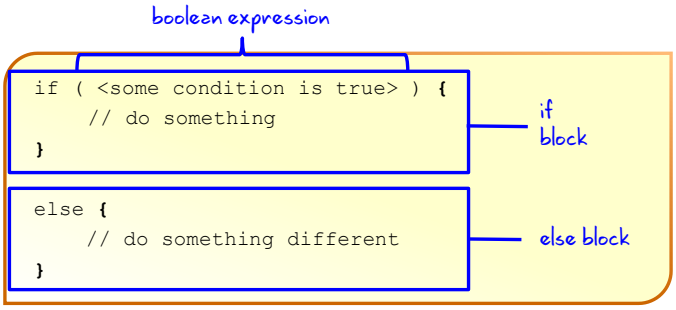
- Working with conditions
- Working with an array of items
- Processing an array of items

Making Decisions

In your daily life, you have to make a lot of decisions, and you often use the word “if” with some condition when making those decisions. For example, “If I can see my destination on the left, I will turn left, otherwise I’ll turn right.”

One of the tasks that programs often perform is to evaluate a condition and, depending on the result, execute different blocks or branches of code. This is called conditional logic, and it is handled through the use of an `if/else` statement.

The `if/else` Statement



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

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

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

Boolean Expressions

Review:

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

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

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

Relational operators

Remember that a `boolean` data type can have only two possible values: `true` and `false`.

In the same way, a boolean expression, made up of some combination of variables, values and operators, must also evaluate to either `true` or `false`.

This usually involves a special kind of operator called a relational operator. Several of these are used in the three examples above:

- Greater than (>)
- Less than or equal to (<=)
- Equal to (==). In the example above, the result of `cost * price` is compared to the value of `total`. If they are equal, the entire expression evaluates to `true`.

Relational Operators

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

Here you see a more complete list of relational operators. The table lists the different conditions you can test by using relational operators. The result of all relational operators is a boolean value. All of the examples in the table yield a boolean result of true.

Note: The equal sign (=) is used to make an assignment, whereas the == sign merely makes a comparison and returns a boolean.

Examples

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

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

Assign a boolean by using an `if` statement.

Assign the boolean directly from the boolean expression.

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

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

Exercise 5-1: Using `if` Statements

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

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

Quiz

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

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

Answer: b

Topics

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

What If There Are Multiple Items in the Shopping Cart?

```
01          // Without an array
02          String itemDesc1 = "Shirt";
03          String itemDesc2 = "Trousers";
04          String itemDesc3 = "Scarf";
05
06          // Using an array
07          String[] items = {"Shirt","Trousers","Scarf"};
```

Not realistic if 100s of items!

Much better!

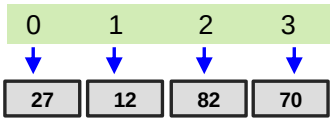
Think about how your code would look if there were multiple items in the shopping cart. You would have to initialize each item description separately. Imagine if you had a thousand items!

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

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

Introduction to Arrays

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



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

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

Array Examples

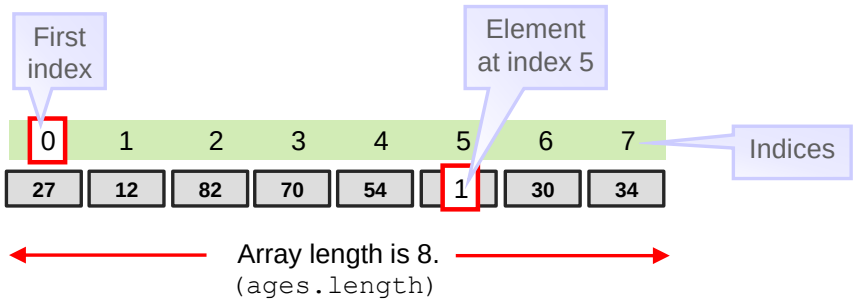
Array of `int` types

Array of `String` types

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

Array Indices and Length

The `ages` array has eight elements.



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

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

Declaring and Initializing an Array

- Syntax:

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

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

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

All in one line

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

- Syntax for declaring an array:

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

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

where:

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

Declaring and Initializing an Array

Examples:

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

Multistep approach

Multistep approach

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

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

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

Accessing Array Elements

- Get values from the `ages` array:

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

- Set values from the `names` array:

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

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

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

Exercise 5-2: Using an Array

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

In this exercise, you declare and initialize a `String` array to hold item descriptions. Then you experiment with accessing the array.

Quiz

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

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

a. `lengths` cannot be used as an array identifier.

b. All of the element values should have the same format (all using `double` values, or all using `int` values).

c. The array was declared to hold `int` values. `double` values are not allowed.

Answer: c

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

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

c is correct.

Quiz

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

```
int[] classSize = {5, 8, 0, 14, 194};
```

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

b. `classSize[5]` is the reference to the last element in the array.

c. There are 5 integers in the `classSize` array.

d. `classSize.length == 5`

Answer: a, c, d

a is correct.

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

c is correct.

d is correct.

Topics

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

Loops

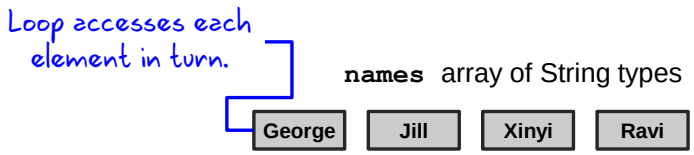
Loops are used in programs to repeat blocks of statements

- Until an expression is false

  or

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

Processing a String Array




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

Each iteration returns the next element of the array.

Output:

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


The `for` loop syntax is:

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

where:

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

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

Using `break` with Loops

`break` example:

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

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

Output:

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

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

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

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

1. In NetBeans,continue editing **`Exercise_05-2`** or open **`Exercise_05-3`.**
2. Create a `for` loop that iterates through the array of item descriptions, displaying each element.
3. Precede the list of elements with the message: "Items purchased:".

In this exercise, you loop through the array of item descriptions, printing each element.

Quiz

Given the following code,

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

which option below shows the correct output?

a. `Size: 4,`

b. `Size: 4`

c. `Size: 4,`
`Size: 5,`

d. There is no output.

Answer: a

a is correct.

b is incorrect because the comma appears within each `println` method.

c is incorrect because when the first size greater than 16 is found, the loop breaks and does not return.

d is incorrect because the first iteration of the loop would print.

Summary

In this lesson, you should have learned how to:

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