Arrays y Loops

JoeDayz | 2024

Objectives

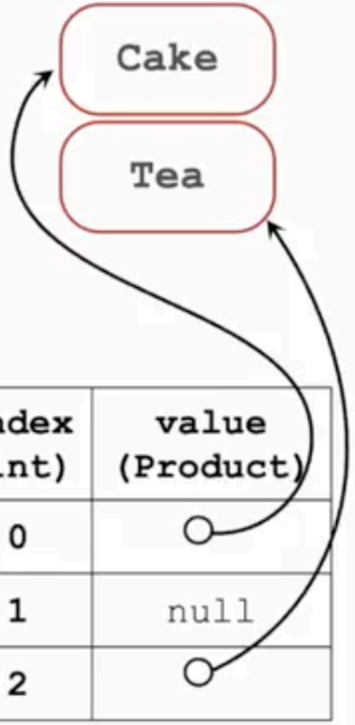
After completing this lesson, you should be able to:

- Declare, initialize, and access arrays of object and primitive types
- Use the `while`, `do/while`, `for`, and `forEach` loops
- Process arrays by using a loop
- Use multidimensional arrays
- Use embedded loops
- Use break and continue operators

Arrays

An array is a fixed-length collection of elements of the same type indexed by `int`.

- Declare array object: Determine the type of elements to be stored in the array
- Create array object: Determine the length (number of elements) in the array
  - Once an array is created, its length cannot be changed.
  - An array of object references is filled with `null` values.
  - An array of primitive values is filled with `0` values (`false` values if it is of Boolean type).
- Initialize array content: Assign values to array positions in any order
  - Index starts at `0`.
  - Last valid index position is `array.length-1`.
- Access elements in the array using index.
  - No need to "extract" an array element to operate on it.
  - ⊗ Accessing outside of the valid index boundaries produces `ArrayIndexOutOfBoundsException`.




```
int[] primes;
// int primes[];
primes = new int[4];
primes[1] = 3;
primes[2] = 7;
primes[0] = primes[1]-1;
⊗ primes[4];
```

index (int)	value (int)
0	2
1	3
2	7
3	0

```
Product[] products;
// Product products[];
products = new Product[3];
products[0] = new Food("Cake");
products[2] = new Drink("Tea");
products[2].setPrice(1.99);
⊗ products[3];
```

index (int)	value (Product)
0	○
1	null
2	○

Combined Declaration, Creation, and Initialization of Arrays

Array objects can be declared, created, and initialized at the same time.

- Combine declaration and creation of the array object:

```
int[] primes = new int[3];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
```

```
Product[] products = new Product[3];
products[0] = new Food("Cake");
products[1] = new Drink("Tea");
products[2] = new Food("Cookie");
```

- Combine creation of the array object and initialization of the array content:

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

```
Product[] products;
products = new Product[]{new Food("Cake"),
                         new Drink("Tea"),
                         new Food("Cookie")};
```

- Combine declaration and creation of the array object as well as the initialization of the array content:

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

```
Product[] products = {new Food("Cake"),
                      new Drink("Tea"),
                      new Food("Cookie")};
```

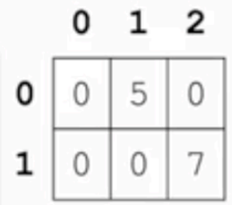
✣ **Note:** Array content can be provided as a comma-separated list, wrapped up in a block of code **{,,,}**

Multidimensional Arrays

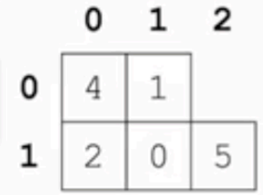
Java arrays can have multiple dimensions:

- Can use normal or short-hand initializations
- Accessed by indicating each dimension index
- Can be of nonsquare shapes
- Can have more than two dimensions

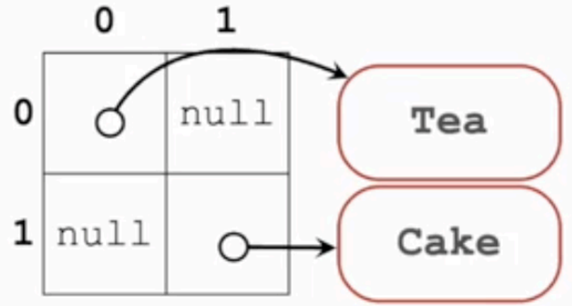
```
int[][] matrix = new int[2][3];
matrix[0][1] = 5;
matrix[1][2] = 7;
```



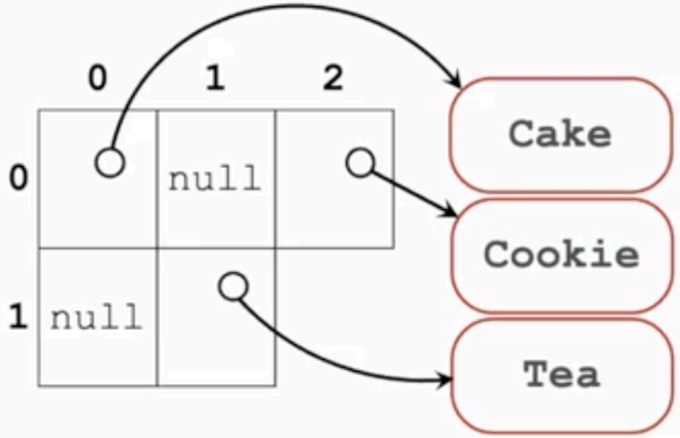

```
int[][] matrix = {{4,1},{2,0,5}};
```




```
Product[][] products = new Product[2][2];
matrix[1][1] = new Food("Cake");
matrix[0][0] = new Drink("Tea");
```




```
Product[][] products =
            {{new Food("Cake"), null, new Food("Cookie"},
             {null, new Drink("Tea")}};
```



Copying Array Content

Array content can be copied into another array.

- Java arrays are of fixed length (cannot be resized).
- However, resize can be emulated by creating a new array with a different size and copying all or partial content from the original array to the new array object, using:

```
System.arraycopy(<source array>, <source position>,
                 <destination array>, <destination position>,
                 <length of content to copy from source>);
```

- or:

```
<new array> = Arrays.copyOf(<source array>, <new array length>);
<new array> = Arrays.copyOfRange(<source array>, <start position>, <end postion>);
```

index int	value char
0	a
1	c
2	m
3	e

index int	value char
0	a
1	c
2	m
3	e
4	

```
char[] a1 = {'a','c','m','e'};
char[] a2 = {'t','o','','','',''};
System.arraycopy(a1, 2, a2, 3, 2);
```

```
char[] b1 = {'a','c','m','e'};
char[] b2 = Arrays.copyof(b1,5);
```

index int	value char
0	a
1	c
2	m
3	e

index int	value char
0	t
1	o
2	
3	m
4	e

Arrays Class

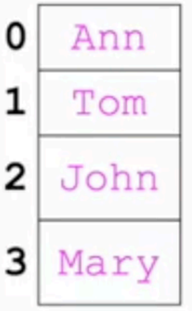
The `java.util.Arrays` class provides convenient methods for handling arrays, such as:

- Filling an array with values
- Searching through the array
- Comparing content
- Sorting array content using:
  - `Comparable` (as implemented by objects within the array)
  - `Comparator` interfaces

```
String[] values = new String[5];
Arrays.fill(values, 2, 4, "aaa");
int x = Arrays.binarySearch(values, "aaa");
```


```
String[] names1 = {"Mary","Ann","Jane","Tom"};
String[] names2 = {"Mary","Ann","John","Tom"};
boolean isTheSame = Arrays.equals(names1, names2);
Arrays.sort(names2);
Arrays.sort(names2, new LengthCompare());
```


```
public class LengthCompare implements Comparator<String> {
  public int compare(String s1, String s2) {
    if (s1.length() > s2.length()) { return 1; }
    if (s1.length() < s2.length()) { return -1; }
    return 0;
  }
}
```



Loops

Java loops contain the following parts:

- Declaration of the iterator
- Termination condition
- Iterator (typically increment or decrement)
- Loop body

Java loop constructs:

- `while` provides a simple iterator.
- `do while` guarantees to get through the loop body at least once.
- `for` keeps declaration, termination condition, and iterator together, on three positions separated with the ";" symbol.
- `forEach` provides a convenient way of iterating through the array or collection (see next page).

```
int i = 0;
while (i < 10) {
  // iterative logic
  i++;
}
```

```
int i = 0;
do {
  // iterative logic
  i++;
} while (i < 10);
```

```
for (int i = 0; i < 10; i++) {
  // iterative logic
}
```

```
while (someMethod()) {
  // iterative logic
}
```

```
int i = 0;
for ( ; i < 10 ; ) {
  // iterative logic
  i++;
}
```

- **Note:** `while` or `do/while` loops are best used with a method call that returns Boolean to control iterations.
- **Note:** The example demonstrates the positional nature of the `for` loop construct (not a recommended way of writing a `for` loop.)

Processing Arrays by Using Loops

Processing array in a loop:

- Use `array.length` to determine the boundary for the termination condition.
- Access array values by using an index and an iterator.
- Use the `forEach loop` to auto-extract values.

```
int[] values = {1,2,3};
StringBuilder txt = new StringBuilder();
for (int i = 0; i < values.length; i++) {
  int value = values[i];
  txt.append(value);
}
for (int value: values) {
  txt.append(value);
}
```

- **Note:** "`forEach`" is not an actual operator, or a reserved word, but simply a name that describes a `for` loop that iterates through the array or collection without a need to maintain an index and iterator logic.

Complex `for` Loops

The positional nature of a `for` loop makes it possible to:

- Define multiple iterators separated by ", “.
- Write a loop without a body if its iterator section also contains required actions.

(However, mixing action and iteration logic can be confusing and thus is not a good coding practice.)

```
int[] values = {1,2,3,4,5,6,7,8,9};
int sum = 0;
for (int i = 0 ; i < values.length ; sum += i++);
// sum is 36
```

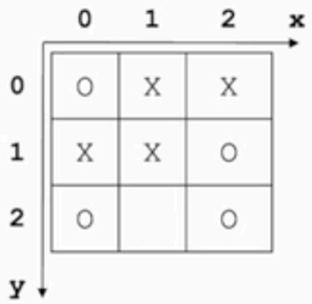
```
int[][] matrix = {{1,2,3},{4,5,6},{7,8,9}};
for (int i = 0, j = 2 ; !(i == 3 || j == -1) ; i++, j--) {
  int value = matrix[i][j];
}
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Embedded Loops

Loops can be placed inside other loops.

- Useful to process multidimensional arrays
- Can combine any loop types: `while`, `do-while`, `for`, and `for-each`




```
char[][] game = {{'O','X','X'},
                 {'X','X','O'},
                 {'O',' ','O'}};
StringBuilder txt = new StringBuilder();
for (int x = 0; x < game.length; x++) {
  int y = 0;
  while (y < game[x].length) {
    txt.append(game[x][y]);
    y++;
  }
  txt.append('\n');
}
for(char[] row : game) {
  for (char value: row) {
    txt.append(value);
  }
  txt.append('\n');
}
```

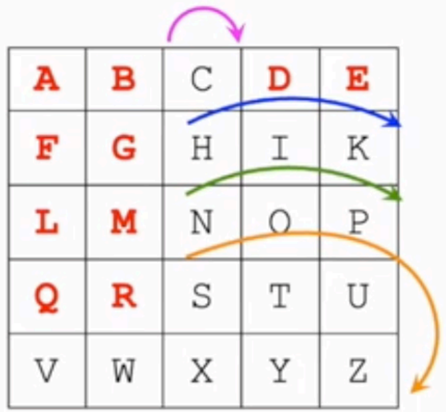
Result:

```
OXX
XXO
O O
OXX
XXO
O O
```

Break and Continue

Breaking out of loops and skipping loop cycles:

- `continue` operator skips the current loop cycle.
- `continue <label>` skips the labeled loop cycle.
- `break` terminates the current loop.
- `break <label>` terminates the labeled loop.



Result:

```
ABDE
FGLM
QR
```

```
char[][] matrix = {{'A','B','C','D','E'},
                   {'F','G','H','I','K'},
                   {'L','M','N','O','P'},
                   {'Q','R','S','T','U'},
                   {'V','W','X','Y','Z'}};
StringBuilder txt = new StringBuilder();
outerLoopLabel:
for(char[] row : matrix) {
  for (char value: row) {
    if (value == 'C') { continue; }
    if (value == 'H') { continue outerLoopLabel; }
    if (value == 'N') { break; }
    if (value == 'S') { break outerLoopLabel; }
    txt.append(value);
  }
  txt.append('\n');
}
```

Summary

In this lesson, you should have learned how to:

- Declare, initialize, and access arrays of object and primitive types
- Use the `while`, `do/while`, `for`, and `forEach` loops
- Process arrays by using a loop
- Use multidimensional arrays
- Use embedded loops
- Use break and continue operators

Practices for Lesson 8: Overview

In this practice, you will:

- Add an array of review objects to the `ProductManager` class
- Compute product rating based on the average value of ratings in all reviews