Middlebury

CSCI 201: Data Structures

Fall 2024

Lecture 2T: Arrays

Goals for today:

- Use the debugger to inspect the value stored in variables.
- Do a mini-introduction to objects and use the `new` keyword.
- Create and use fixed-size arrays to store many values of the same type.
- Modify objects by using a **reference** to the underlying object.
- Create and use multi-dimensional arrays.

Remember the `String` example from last class? Here's a variation.

```
1  public class StringEquals {
2    public static void main(String[] args) {
3
4      String s = "Middlebury";
5      String a = s.substring(0, 4); // "Midd"
6      String b = "Midd";
7
8      boolean sameByEqualityOperator = (a == b);
9      boolean sameByEqualsMethod = a.equals(b);
10
11     System.out.println("sameByEqualityOperator: " + sameByEqualityOperator);
12     System.out.println("sameByEqualsMethod: " + sameByEqualsMethod);
13   }
14 }
```

Why did this happen? What is `==` actually comparing?

A `String` is a class. An instance of a `String` is an `Object`.

We have been creating `String`s like this (special syntax for `String`s):

```
1 String s = "Middlebury";
```

But we can also create `String`s like this:

```
1 String s = new String("Middlebury");
```

Because strings are *objects*.

- Objects are created using the `new` keyword.
- When objects are created, the resulting variable is a *reference variable.*
- A reference variable is really an address to a place in memory where the object is stored.
- We can pass around this address to make changes to the underlying object.
- The `==` operator will check if the addresses are the same.

Back to our `String` example.

```
1  public class StringEquals {
2    public static void main(String[] args) {
3
4      String a = new String("Midd");
5      String b = new String("Midd");
6
7      boolean sameByEqualityOperator = (a == b);
8      boolean sameByEqualsMethod = a.equals(b);
9
10     System.out.println("sameByEqualityOperator: " + sameByEqualityOperator);
11     System.out.println("sameByEqualsMethod: " + sameByEqualsMethod);
12   }
13 }
```

Introducing fixed-size arrays! Size is fixed upon creating the array.

Unlike lists in **Python**, where the size can change.

```
1 values = []
2 for i in range(10):
3     values.append(i)
```

We'll see dynamically-sized arrays in **Java** next week.

Fixed-sized arrays can be used to store multiple values of the same type.

```java
public class ArrayExamples {
  public static void main(String[] args) {

    int[] values1 = new int[6]; // creates an array of integers with 6 values
    values1[0] = 311;
    values1[1] = 312;
    values1[2] = 315;
    values1[3] = 318;
    values1[4] = 333;
    values1[5] = 467;

    int[] values2 = {311, 312, 315, 318, 333, 467};
    int[] values3 = new int[]{311, 312, 315, 318, 333, 467};
  }
}
```

Retrieve the length using `.length`.

We're not calling a method! So no `()`. We're accessing a property.

Also, here's a fancier way to write `for`-loops!

Called *enhanced* (or *for-each*) loops.

```
1 int[] intValues = {311, 312, 315, 318, 333, 467};
2 for (int value : intValues) {
3   System.out.println("value = " + value);
4 }
5
6 String[] stringValues = {"apple", "banana", "pear", "orange"};
7 for (String value : stringValues) {
8   System.out.println("value = " + value);
9 }
```

This can be done with objects that are *iterable* (like arrays).

The variable we declare for arrays are also reference variables.

```
1 int[] values0 = {311, 312, 315, 318, 333, 467};
2
3 int[] values1 = values0; // values1 has the same reference as values0
4
5 values1[3] = 321; // change item in values1
6
7 System.out.println(values0[3]); // change is reflected in values0
```

Multi-dimensional arrays: can be rectangular or jagged.

```java
// rectangular: all rows have the same size
int nRows = 10;
int nCols = 20;
int[][] matrix = new int[nRows][nCols];
for (int i = 0; i < nRows; i++) {
  for (int j = 0; j < nCols; j++) {
    matrix[i][j] = 1; // some value
  }
}

// jagged: each row has a different size
char[][] triangle = new char[nRows][]; // nRows arrays, initially null
for (int i = 0; i < triangle.length; i++) {
  triangle[i] = new char[i + 1];
  for (int j = 0; j < triangle[i].length; j++)
    triangle[i][j] = '*';
}
```

The Plan for Tuesday.

- You will apply some of this material on some practice problems in class on Tuesday.
- Please submit exit ticket 2T after class (link in the row for today at go/cs201). Questions will be about how the practice problems with arrays went and to upload your progress on solving one of the problems.