11/30/22

Notes: Creating and Using Arrays

Printing Arrays

```
boolean[] bs = { true, false };
```

```
System.out.println(bs[0]);
System.out.println(bs[1]);
> true
> false
```

```
import java.util.Arrays;
System.out.println(Arrays.toString(bs));
> [true, false]
```

Notes: Creating and Using Arrays

Copying Arrays

```
boolean[] bs = { true, false, true };
```

```
boolean[] bTemp = new boolean[5];
bTemp[0] = bs[0];
bTemp[1] = bs[1];
bTemp[2] = bs[2];
bs = bTemp;
System.out.println(Arrays.toString(bs));
> [true, false, true, false, false]
```

```
import java.util.Arrays;
bs = Arrays.copyOf(bs, 5);
System.out.println(Arrays.toString(bs));
> [true, false, true, false, false]
```

Notes: Creating and Using Arrays

Re-Allocating and Initializing Array Variables
`boolean[] bs = { true, false, true };`

`bs = new boolean[] { true, false, true, false, false };`
`System.out.println(Arrays.toString(bs));`
`> [true, false, true, false, false]`

6.2: Traversing Arrays with `for` loops

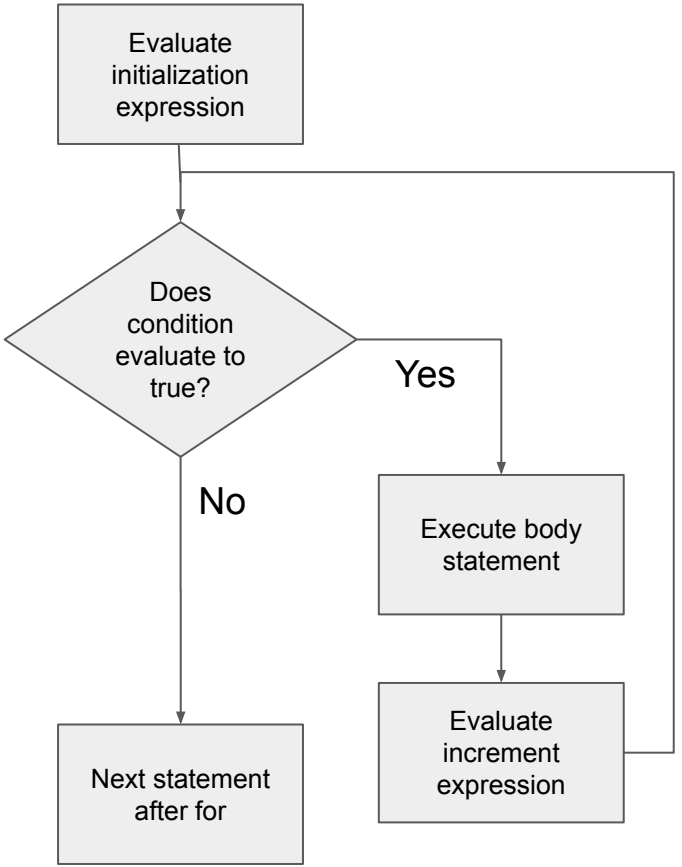
Traversing Arrays with `for` loops

- Remember what we learned about `for` loops in Section 4

`for` (*initialization; condition; increment*)
    statement

Example:

```
for (int i = 1; i <= 100; i++) {
    System.out.println(i);
}
```



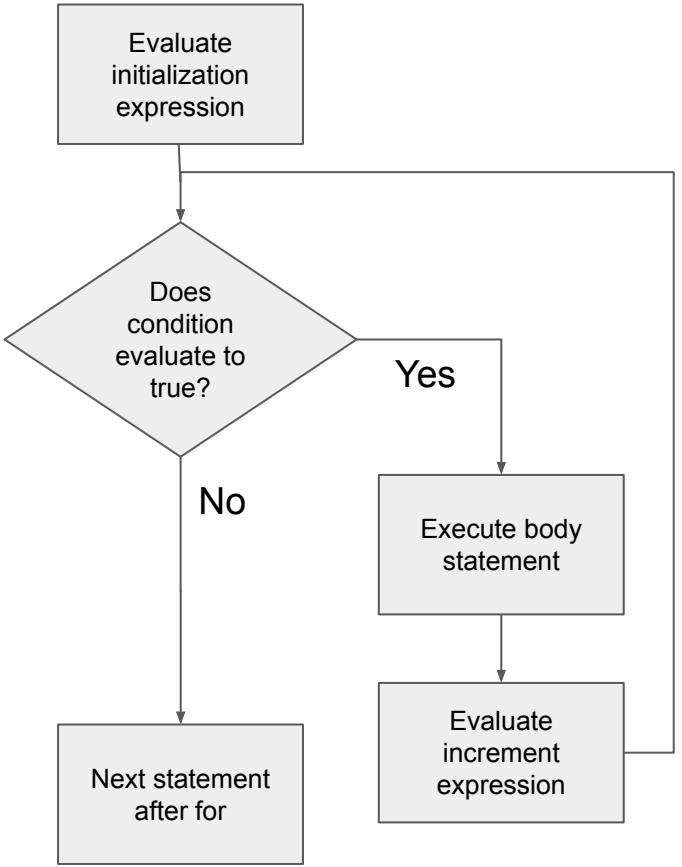
Traversing Arrays with `for` loops

- Now we can now combine this with what we have learned about accessing Arrays

`for` (*initialization; condition; increment*)

statement

Example:



Traversing Arrays with `for` loops

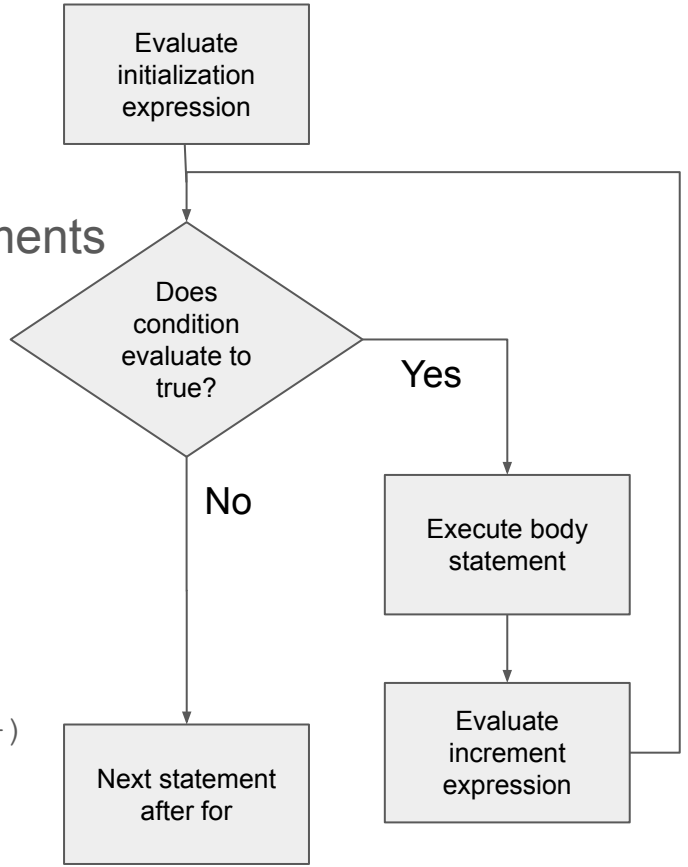
- Now we can now combine this with what we have learned about accessing Arrays
- Arrays have a property called **`length`** and elements can be access **via `[]` and an `index`**

`for` (*initialization; condition; increment*)

statement

Example:

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[idx]);
}
```



Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[ idx]);
}
```

Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[ idx]);
}
```

Note: *Passing an out of range index will cause a* `ArrayIndexOutOfBoundsException`*!*

Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[idx]);
}
```

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[idx]);
}
```

This loop also skips the first element in the Array!

Note: *Passing an out of range index will cause a* `ArrayIndexOutOfBoundsException`*!*

Traversing Arrays with `for` loops

- You can use a `for` loop to traverse an Array from back to front!

```
int[] scores = {95, 100, 91, 85 };
for (int idx = scores.length - 1; idx >= 0; idx--) {
    System.out.println(scores[ idx]);
}
```

Traversing Arrays with `for` loops

- You can use a `for` loop to traverse an Array from back to front!

```
int[] scores = {95, 100, 91, 85 };
for (int idx = scores.length - 1; idx >= 0; idx--) {
    System.out.println(scores[ idx]);
}
```

- ...or to traverse any arbitrary range of elements

```
int[] scores = {95, 100, 91, 85 };
for (int idx = 1; idx <= 2; idx++) {
    System.out.println(scores[ idx]);
}
```

6.3: Traversing Arrays with `for-each` loops

Traversing Arrays with `for-each` loops

- An alternate way to loop through Objects that support the Iterable interface

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable is a copy of arrayVariable[0]
    arrayItemVariable is a copy of arrayVariable[1]
    arrayItemVariable is a copy of arrayVariable[...]
    arrayItemVariable is a copy of arrayVariable[arrayVariable.length-1]
    then the loop terminates
}
```

Traversing Arrays with `for-each` loops

```
for (type arrayItemVariable : arrayVariable) {
    itemVariable resolves to Array[...]
}
```

```
String[] colors = {"red", "orange", "purple"};
```

```
System.out.println("begin");
for(String color: colors){
  System.out.println(" " + color);
}
System.out.println("end");
```

Traversing Arrays with `for-each` loops

```
for (type arrayItemVariable : arrayVariable) {
    itemVariable resolves to Array[...]
}
```

```
String[] colors = {"red", "orange", "purple"};
```

```
System.out.println("begin");
for(String color: colors){
  System.out.println(" " + color);
}
System.out.println("end");
```

Output:

```
begin
  red
  orange
  purple
end
```

Traversing Arrays with `for-each` loops

- The type of the `for-each` variable MUST match the type of the values stored in the Array

```
String[] colors = {"red", "orange", "purple"};

for(int color: colors){
  System.out.println(" " + color);
}
```

Traversing Arrays with `for-each` loops

- The type of the `for-each` variable MUST match the type of the values stored in the Array

```
String[] colors = {"red", "orange", "purple"};

for(int color: colors){
  System.out.println(" " + color);
}
```

Note: *color must be of type String since colors is an Array that contains Strings*

Comparing `for` and `for-each` loops

- `for`
  - Direct access to any element in the Array - in any order - using zero-based index and `[]`
  - You always know the index - so using parallel Arrays is easy!
  - May require more variables to efficiently operate
  - Can change the value of an Array element during the loop
- `for-each`
  - Sequential access to the elements in the Array - **must always go from first to last**
  - You do not know the index - so using Parallel Arrays is harder (impossible?)
  - May eliminate the need for extra variables (no need to use indexes to access an item)
  - Cannot change the value of an Array element during the loop

Comparing `for` and `for-each` loops

	`String[] colors = {"red", "red", "purple", "blue", "red"};`

`for`
 	
Direct access to any element in the Array - in any order
	`for (int idx = 1 ; idx < colors.length ; idx++) {`
	`  System.out.println(colors[0].equals(colors[idx]));`
	`}`

`for-each` Sequential access to the elements in the Array - must go from first to last	

Comparing `for` and `for-each` loops

```
String[] colors = {"red", "red", "purple", "blue", "red"};
```

`for`

Direct access to any element in the Array - in any order

```
for (int idx = 1 ; idx < colors.length ; idx++) {
  System.out.println(colors[0].equals(colors[idx]));
}
```

Question: *What is this code doing?*

`for-each`

Sequential access to the elements in the Array - must go from first to last

Comparing `for` and `for-each` loops

```
String[] colors = {"red", "red", "purple", "blue", "red"};
```

`for`

Direct access to any element in the Array - in any order

```
for (int idx = 1 ; idx < colors.length ; idx++) {
  System.out.println(colors[0].equals(colors[idx]));
}
```

Determine if the colors in an Array match the first color in the Array

`for-each`

Sequential access to the elements in the Array - must go from first to last

Comparing `for` and `for-each` loops

```
String[] colors = {"red", "red", "purple", "blue", "red"};
```

`for`

Direct access to any element in the Array - in any order

```
for (int idx = 1 ; idx < colors.length ; idx++) {
  System.out.println(colors[0].equals(colors[idx]));
}
```

Determine if the colors in an Array match the first color in the Array

`for-each`

Sequential access to the elements in the Array - must go from first to last

```
String firstColor = colors[0];
boolean isFirstItem = true;
for (String color : colors) {
  if (isFirstItem) {
    isFirstItem = false;
  } else {
    System.out.println(firstColor.equals(color));
  }
}
```

Comparing `for` and `for-each` loops

```
String[] students = {"Julie", "Anne","Roscoe"};
int[] scores = {100, 95, 90};
```

`for`

You always know the index - so using Parallel Arrays is easy

```
for (int idx = 0 ; idx < students.length ; idx++) {
  System.out.println(students[idx] + ": " + scores[idx]);
}
```

`for-each`

You do not know the index - so using Parallel Arrays is challenging.

Impossible?

Comparing `for` and `for-each` loops

```
String[] students = {"Julie", "Anne","Roscoe"};
int[] scores = {100, 95, 90};
```

`for`

You always know the index - so using Parallel Arrays is easy

```
for (int idx = 0 ; idx < students.length ; idx++) {
  System.out.println(students[idx] + ": " + scores[idx]);
}
```

Question: *What is this code doing?*

`for-each`

You do not know the index - so using Parallel Arrays is challenging.

Impossible?

Comparing `for` and `for-each` loops

```
String[] students = {"Julie", "Anne","Roscoe"};
int[] scores = {100, 95, 90};
```

`for`

You always know the index - so using Parallel Arrays is easy

```
for (int idx = 0 ; idx < students.length ; idx++) {
  System.out.println(students[idx] + ": " + scores[idx]);
}
```

Print the score for each Student

`for-each`

You do not know the index - so using Parallel Arrays is challenging.

Impossible?

Comparing `for` and `for-each` loops

```
String[] students = {"Julie", "Anne","Roscoe"};
int[] scores = {100, 95, 90};
```

for

You always know the index - so using Parallel Arrays is easy

```
for (int idx = 0 ; idx < students.length ; idx++) {
  System.out.println(students[idx] + ": " + scores[idx]);
}
```

Print the score for each Student

for-each

You do not know the index - so using Parallel Arrays is challenging.

Impossible?

```
?
```

Comparing `for` and `for-each` loops

```
String[] colors = {"red", "red", "purple", "blue", "red"};
```

`for`

May require more variables to efficiently operate

```
int length = colors.length;
for (int idx = 0 ; idx < length ; idx++) {
  String color = colors[idx];
  System.out.println(color);
}
```

`for-each`

May eliminate the need for extra variables (and no need to use indexes to access an item)

Comparing `for` and `for-each` loops

	`String[] colors = {red, red, purple, blue, red};`

`for`
 	
May require more variables to efficiently operate
	`int length = colors.length;`
	`for (int idx = 0 ; idx < length ; idx++) {`
	`  String color = colors[idx];`
	`  System.out.println(color);`
	`}`

`for-each`
 	
May eliminate the need for extra variables (and no need to use indexes to access an item)
	`for (String color : colors) {`
	`  System.out.println(color);`
	`}`

Comparing `for` and `for-each` loops

	`boolean[] verified = {true, false, false, true};`

`for`
 	
Can change the value of an Array element during the loop
	`for (int idx = 0 ; idx < verified.length ; idx++) {`
	`  if (false == verified[idx]) {`
	`  System.out.println("Verified Item!");`
	`  verified[idx] = true;`
	`  }`
	`}`
	`// verified == {true, true, true, true};`

`for-each` Cannot change the value of an Array element during the loop	

Comparing `for` and `for-each` loops

```
boolean[] verified = {true, false, false, true};
```

`for`

Can change the value of an Array element during the loop

```
for (int idx = 0 ; idx < verified.length ; idx++) {
  if (false == verified[idx]) {
    System.out.println("Verified Item!");
    verified[idx] = true;
  }
}
// verified == {true, true, true, true};
```

`for-each`

Cannot change the value of an Array element during the loop

```
for (boolean item : verified) {
  if (false == item) {
    System.out.println("Verified Item!");
    item = true;
  }
}
// verified == {true, false, false, true};
```

Practice on your own

- CSAwesome 6.2 - Traversing Arrays with For Loops
- CSAwesome 6.3 - Enhanced For-Loop (For-Each) for Arrays
- Replit - RPNCalculator
- Replit - GrowableArray