11/28/22

6.1: Creating and Using Arrays

Arrays - Declaration

- An Array variable is is a collection of values **of the same type** and is declared like this

```
type[] name;
```

```
boolean[] answers;
String[] questions;
int[] scores;
Student[] students;
```

Arrays - Declaration

- An Array variable is is a collection of v and is declared like this

```
type[] name;
```

Java also supports this Array declaration syntax

```
type name[];
```

But is not as common

```
boolean[] answers;
String[] questions;
int[] scores;
Student[] students;
```

Arrays - Declaration

- An Array variable is is a collection of values **of the same type** and is declared like this

```
type[] name;
```

```
boolean[] answers;
String[] questions;
int[] scores;
Student[] students;
```

Note: *Arrays in Java are Object types. As-written - these Array variables are undefined and your code will fail if you attempt to access them.*

So...

Arrays - Creation

- Arrays are created with an **initializer list** or **`new`**

```
boolean[] answers = {true, false, false, true};
int[] scores = {100, 84, 95, 78};

double[] prices = new double[20];
String[] questions = new String[5];

int numStudents = 10;
Student[] students = new Student[numStudents];
```

Arrays - Creation

- Arrays are created with an **initializer list** or **`new`**

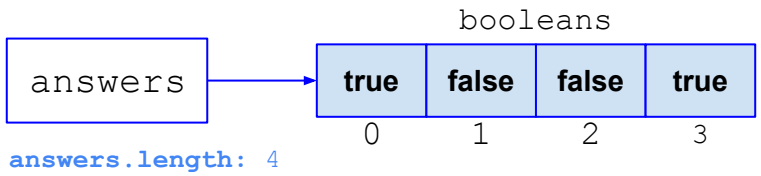
```
boolean[] answers = {true, false, false, true};
int[] scores = {100, 84, 95, 78};

double[] prices = new double[20];
String[] questions = new String[5];

int numStudents = 10;
Student[] students = new Student[numStudents];
```

Note 1: *Each of these Array variables now have a value assigned to them - And can be referenced by your code.*

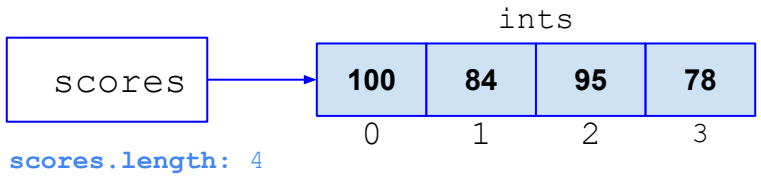
Note 2: *After creation every Array has an available `length` property (which never changes)*

Arrays - Creation

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

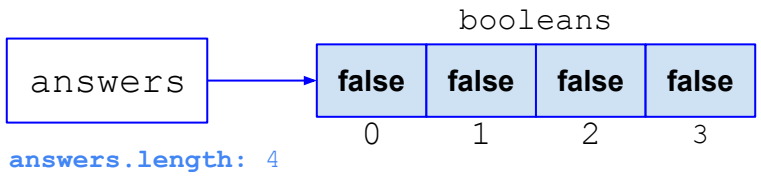



```
int[] scores = {100, 84, 95, 78};
```

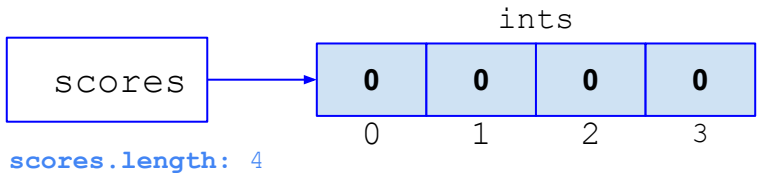


Arrays - Creation - Primitive Defaults

```
boolean[] answers = new boolean[4];
```




```
int[] scores = new int[4];
```



Arrays - Creation

A note about initializer lists...

These can **ONLY** be used when declaring an Array variable

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

Attempting to assign an initializer list to an **EXISTING** Array variable will **FAIL**

```
answers = {false, false, false, false};
```

*** ERROR ***

Arrays - Creation

A note about initializer lists...

These can **ONLY** be used when declaring an Array variable

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

Attempting to assign an initializer list to an EXISTING Arra

```
answers = {false, false, false, f
```

*** ERROR ***

This is different behavior than what you will find in JavaScript, Ruby, and Python

Arrays - Creation

Using `new` to re-assign an Array is allowed

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

This works!

```
answers = new boolean[4];
```

This also works...

```
answers = new boolean[] { true, false, false, true };
```

Arrays - Access - `length`

- The length of an Array can be determined via the `length` **property**. **Note:** The length of a String is accessed via the `String.length()` **method**.

```
boolean[] answers = {true, false, false, true};
System.out.println(answers.length);
A> ?

String[] questions = new String[5];
System.out.println(questions.length);
B> ?
```

Arrays - Access - `length`

- The length of an Array can be determined via the `length` **property**. **Note:** The length of a String is accessed via the `String.length()` **method**.

```
boolean[] answers = {true, false, false, true};
System.out.println(answers.length);
A> 4

String[] questions = new String[5];
System.out.println(questions.length);
B> 5
```

Arrays - Access - READING

- Items in an Array can be read via the **[index]** property. **Note:** Like `String`-`index` is zero-based and the range of valid `index` values is `0` to `length-1`

```
boolean[] answers = {true, false, false, true};
System.out.print(answers[2] + ", " + answers[0]);
C> ?

int[] scores = {100, 84, 95, 78};
System.out.print(scores[1] + ", " + scores[3]);
D> ?
```

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

Arrays - Access - READING

- Items in an Array can be read via the **[index]** property. **Note:** Like `String`-`index` is zero-based and the range of valid `index` values is `0` to `length-1`

```
boolean[] answers = {true, false, false, true};
System.out.print(answers[2] + ", " + answers[0]);
C> false, true

int[] scores = {100, 84, 95, 78};
System.out.print(scores[1] + ", " + scores[3]);
D> 84, 78
```

Arrays - Access - WRITING

- Items in an Array can be written via the `[index]` property. **Note:** Unlike `String` - you can change the values in an Array after it is created (however you cannot change its `length` after creation)

```
boolean[] answers = {true, false, false, true};
answers[2] = true; answers[0] = false;
System.out.print(answers[2] + ", " + answers[0]);
C> ?

int[] scores = {100, 84, 95, 78};
scores[1] = 48; scores[3] = 87;
System.out.print(scores[1] + ", " + scores[3]);
D> ?
```

Arrays - Access - WRITING

- Items in an Array can be written via the `[index]` property. **Note:** Unlike `String` - you can change the values in an Array after it is created (however you cannot change its `length` after creation)

```
boolean[] answers = {true, false, false, true};
answers[2] = true; answers[0] = false;
System.out.print(answers[2] + ", " + answers[0]);
C> true, false

int[] scores = {100, 84, 95, 78};
scores[1] = 48; scores[3] = 87;
System.out.print(scores[1] + ", " + scores[3]);
D> 84, 87
```

Arrays - Access - Object Types

- Arrays that hold Object types work a little differently than those that hold primitive types
- We already saw that the length properly works

```
String[] questions = new String[5];
System.out.println(questions.length);
B> 5
```

Arrays - Access - Object Types

- Arrays that hold Object types work a little differently than those that hold primitive types
- We already saw that the length properly works

```
String[] questions = new String[5];
System.out.println(questions.length);
B> 5
```

- But what about reading and writing values in an Array that holds Object types?

Arrays - Access - Object Types

- But what about reading and writing values in an Array that holds Object types?

```
String[] questions = new String[5];
System.out.println(questions[1]);
E> ?
```

```
Student[] students = new Student[10];
System.out.println(students[1]);
F> ?
```

Arrays - Access - Object Types

- But what about reading and writing values in an Array that holds Object types?

```
String[] questions = new String[5];
System.out.println(questions[1]);
E> null
```

```
Student[] students = new Student[10];
System.out.println(students[1]);
F> null
```

For Arrays that hold Object types - each slot will be be initialized to `null`

Arrays - Access - Object Types

- But what about reading and writing values in an Array that holds Object types?

```
String[] questions = new String[5];
System.out.println(questions[1]);
E> null
```

```
Student[] students = new Stude
System.out.println(students[1]
F> null
```

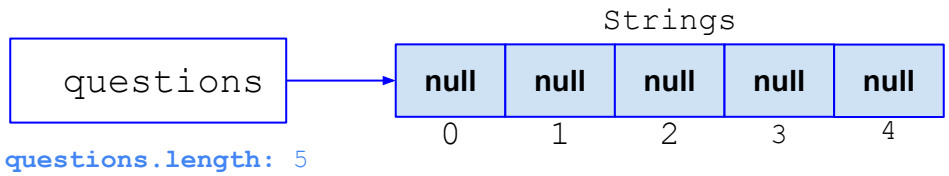
For Arrays that hold Object types - each slot will be be initialized to `null`

Initialize each Array slot with `new`

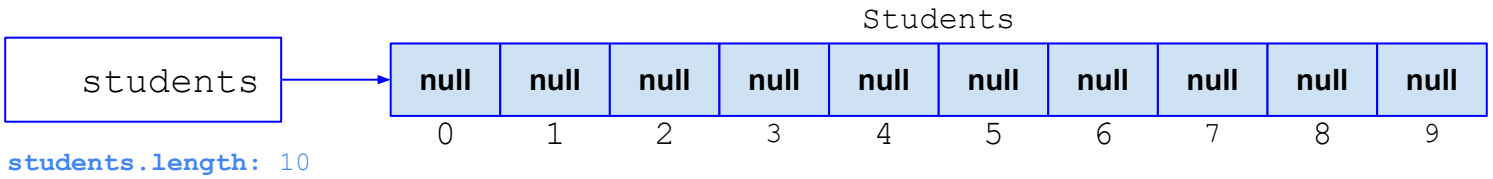
```
students[0] = new Student();
students[1] = new Student();
students[2] = new Student();
...
```

Arrays - Creation

```
String[] questions = new String[5];
```

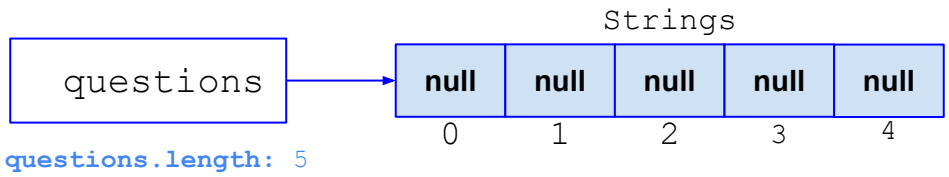



```
Student[] students = new Student[10];
```

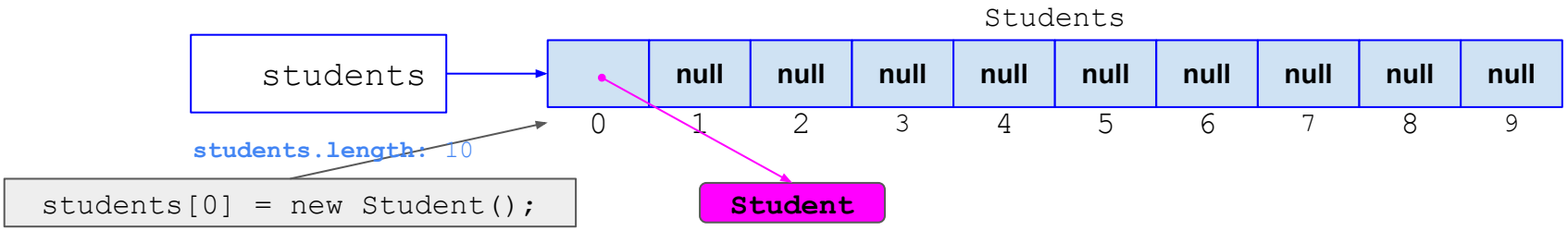


Arrays - Creation

```
String[] questions = new String[5];
```

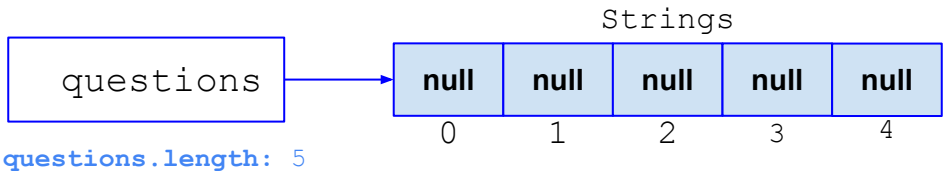



```
Student[] students = new Student[10];
```



Arrays - Creation

```
String[] questions = new String[5];
```

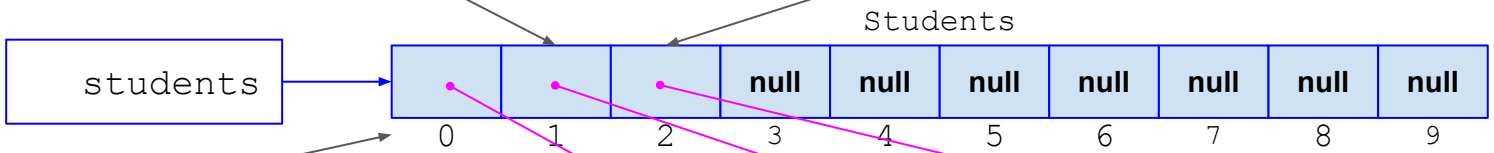


questions.length: 5

```
students[1] = new Student();
```

```
students[2] = new Student();
```

```
Student[] students = new Student[10];
```



students.length: 10

```
students[0] = new Student();
```

Practice on your own

- CSAwesome 6.1 - Array Creation and Access
- Unit 6 - MultiReturn on Replit
- Unit 6 - StudentGrades on Replit