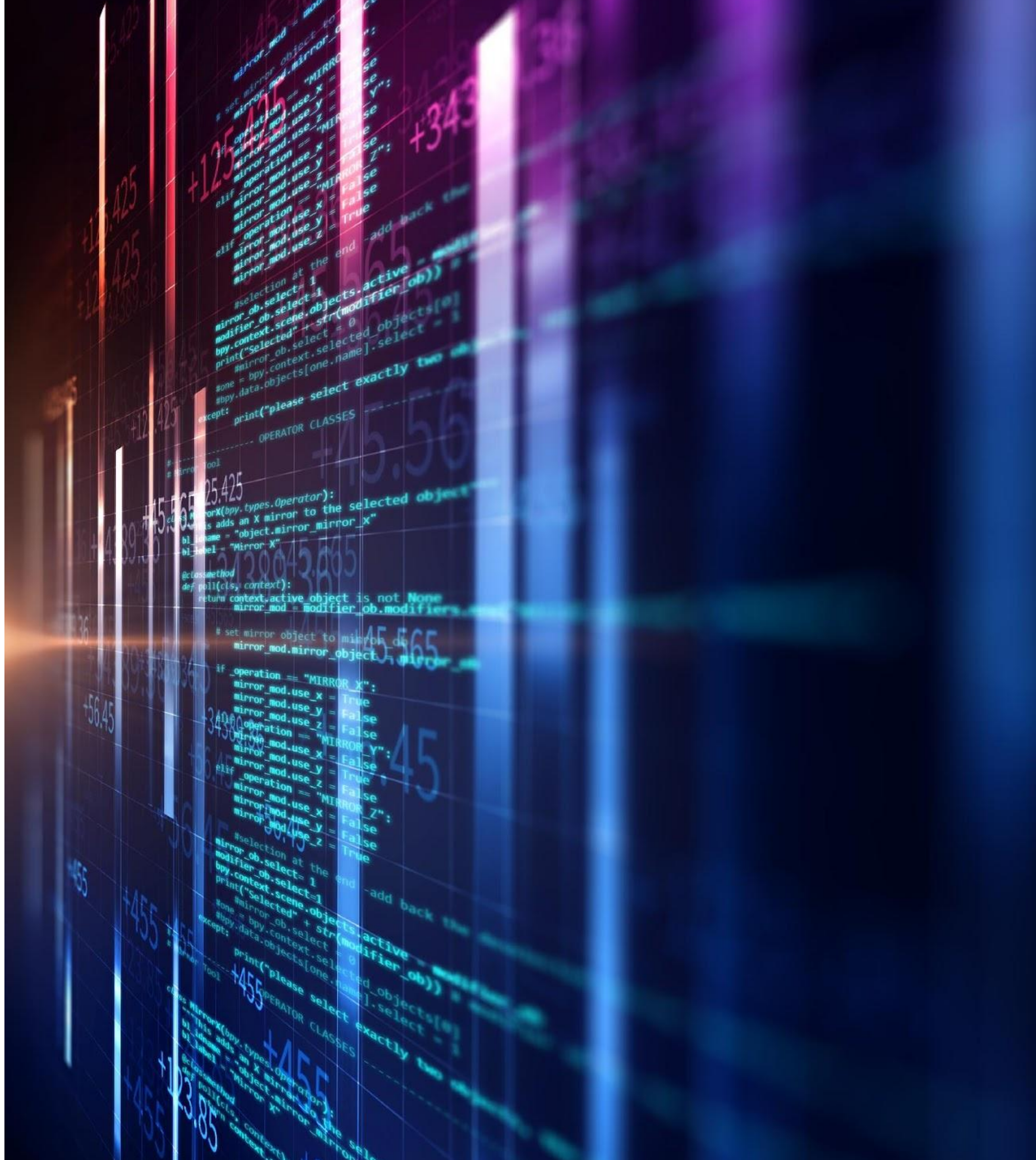
Arrays & ArrayLists

Brandon Krakowsky



Arrays

Arrays

- An array is a way to store a collection of items, as a single unit
  - It's like a list in Python, but not exactly …

Arrays

- An array is a way to store a collection of items, as a single unit
  - It’s like a list in Python, but not exactly …
- Arrays are defined with a *fixed number of slots*, each of which holds an individual item
  - You can add and delete items from those slots as needed

Arrays

- An array is a way to store a collection of items, as a single unit
  - It's like a list in Python, but not exactly …
- Arrays are defined with a *fixed number of slots*, each of which holds an individual item
  - You can add and delete items from those slots as needed
- Each item in an array is distinguished by a numerical *index* between 0 and the array size minus 1
  - This works exactly the same as list indexing in Python

Arrays

- An array is a way to store a collection of items, as a single unit
  - It's like a list in Python, but not exactly …
- Arrays are defined with a *fixed number of slots*, each of which holds an individual item
  - You can add and delete items from those slots as needed
- Each item in an array is distinguished by a numerical *index* between 0 and the array size minus 1
  - This works exactly the same as list indexing in Python
- Arrays *can* contain any type of element value (primitive types or Objects), but you *can't* store different types in a single array
  - You can have an array of ints, an array of Strings, or even an array of arrays, but you can't have an array that contains, for example, both Strings and ints

Arrays

- An array is a way to store a collection of items, as a single unit
  - It's like a list in Python, but not exactly …
- Arrays are defined with a *fixed number of slots*, each of which holds an individual item
  - You can add and delete items from those slots as needed
- Each item in an array is distinguished by a numerical *index* between 0 and the array size minus 1
  - This works exactly the same as list indexing in Python
- Arrays *can* contain any type of element value (primitive types or Objects), but you *can't* store different types in a single array
  - You can have an array of ints, an array of Strings, or even an array of arrays, but you can't have an array that contains, for example, both Strings and ints
- An array itself, is an Object

Creating Arrays – 1st Step

To create an array in Java, you first declare a variable to hold the array

- Array variables indicate the type of object the array will hold, followed by empty brackets [], and the name of the array
- For example, this declares an array of ints:
  `int[] myArrayOfInts;`

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 1st Step

To create an array in Java, you first declare a variable to hold the array

- Array variables indicate the type of object the array will hold, followed by empty brackets [], and the name of the array
- For example, this declares an array of ints:
  `int[] myArrayOfInts;`
- This declares an array of Strings:
  `String[] myArrayofStrings;`

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 1st Step

To create an array in Java, you first declare a variable to hold the array

- Array variables indicate the type of object the array will hold, followed by empty brackets [], and the name of the array
- For example, this declares an array of ints:
  `int[] myArrayOfInts;`
- This declares an array of Strings:
  `String[] myArrayofStrings;`
- Imagine we have a Customer class. This declares an array of Customers:
  `Customer[] myArrayOfCustomers;`

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 2nd Step

- The second step is to create an array object and assign it to that variable. There are two ways to do this.
- One way is to use the *new* operator to create a new instance of an array

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 2nd Step

- The second step is to create an array object and assign it to that variable. There are two ways to do this.
- One way is to use the *new* operator to create a new instance of an array
- This creates a new array of Strings with 10 slots (sometimes called elements)

```
String[] names = new String[10]; //declare and create instance of array
of 10 Strings
```

  - When you create an array object using *new*, you must indicate how many slots that array will hold, inside the brackets []
  - This does not put actual String values in the slots -- you'll have to do that later

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 2nd Step

- The second step is to create an array object and assign it to that variable. There are two ways to do this.
- One way is to use the *new* operator to create a new instance of an array
- This creates a new array of Strings with 10 slots (sometimes called elements)

```
String[] names = new String[10]; //declare and create instance of array of 10 Strings
```

  - When you create an array object using *new*, you must indicate how many slots that array will hold, inside the brackets []
  - This does not put actual String values in the slots -- you'll have to do that later
- This creates a new array of ints with 99 slots

```
int[] temps; //declare array
temps = new int[99]; //create instance of array with 99 slots
```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 3rd Step

The third step is to store things in the array

- You can access the value in any slot of an array by specifying the index number inside brackets []
    - Again, this works exactly the same as list indexing in Python
    - Remember, indexing starts at 0

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 3rd Step

The third step is to store things in the array

- You can access the value in any slot of an array by specifying the index number inside brackets []
  - Again, this works exactly the same as list indexing in Python
  - Remember, indexing starts at 0
- This creates an array of 3 doubles and sets the values

```
double[] myDoubleArray = new double[3];
myDoubleArray[0] = 5.0; //sets 1st value to 5.0
myDoubleArray[1] = 4.1; //sets 2nd value to 4.1
myDoubleArray[2] = 3.9; //sets 3rd  value to 3.9
```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Creating Arrays – 3rd Step

- This creates an array of 2 booleans and sets the values

```
boolean[] myBoolArray = new boolean[2];
myBoolArray[1] = true; //sets 2nd value to true
myBoolArray[0] = false; //sets 1st value to false
```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

PennEngineering

Creating Arrays – 3rd Step

- This creates an array of 2 booleans and sets the values
  ```
  boolean[] myBoolArray = new boolean[2];
  myBoolArray[1] = true; //sets 2nd value to true
  myBoolArray[0] = false; //sets 1st value to false
  ```
- Note, if you use an index outside of 0 up to myArray.length - 1, you'll get an *ArrayIndexOutOfBoundsException*

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Initializing Arrays in One Line

Another way to create an array is to enclose the elements of the array inside curly braces {}, separated by commas

- This initializes the contents (values) of the array in the array declaration
- For example, this creates an array of ints with actual prime numbers
```
int[] primes = {2, 3, 5, 7, 11, 13, 19};
```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Initializing Arrays in One Line

Another way to create an array is to enclose the elements of the array inside curly braces {}, separated by commas

- This initializes the contents (values) of the array in the array declaration
- For example, this creates an array of ints with actual prime numbers
  `int[] primes = {2, 3, 5, 7, 11, 13, 19};`
- This creates an array of Strings, with actual programming languages
  `String[] languages = {"Java", "C", "C++”};`

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Initializing Arrays in One Line

Another way to create an array is to enclose the elements of the array inside curly braces {}, separated by commas

- This initializes the contents (values) of the array in the array declaration
- For example, this creates an array of ints with actual prime numbers
  ```
  int[] primes = {2, 3, 5, 7, 11, 13, 19};
  ```
- This creates an array of Strings, with actual programming languages
  ```
  String[] languages = {"Java", "C", "C++”};
  ```
- Imagine we have a Customer class. This creates an array of Customers, with actual customers
  ```
  Customer[] customers = {new Customer(“Brandon”), new
  Customer(“Betsy”)};
  ```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Initializing Arrays in One Line

Another way to create an array is to enclose the elements of the array inside curly braces {}, separated by commas

- This initializes the contents (values) of the array in the array declaration
- For example, this creates an array of ints with actual prime numbers
  ```
  int[] primes = {2, 3, 5, 7, 11, 13, 19};
  ```
- This creates an array of Strings, with actual programming languages
  ```
  String[] languages = {"Java", "C", "C++”};
  ```
- Imagine we have a Customer class. This creates an array of Customers, with actual customers
  ```
  Customer[] customers = {new Customer(“Brandon”), new
  Customer(“Betsy”)};
  ```
- The syntax above can *only* be used in the array declaration. You can’t do this:
  ```
  int[] composites;
  composites = {4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20}; //illegal
  ```

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

Length of An Array

Every array has a *length* variable, that tells how large the array is

- This array of ints has a length of 10

```
int[] scores = new int[10];
System.out.println(scores.length); //length of 10
```

Length of An Array

Every array has a *length* variable, that tells how large the array is

- This array of ints has a length of 10
  ```
  int[] scores = new int[10];
  System.out.println(scores.length); //length of 10
  ```
- And this array of Customers has a length of 2
  ```
  Customer[] customers = {new Customer(“Brandon”), new
  Customer(“Betsy”)};
  System.out.println(customers.length); //length of 2
  ```

Length of An Array

Every array has a *length* variable, that tells how large the array is

- This array of ints has a length of 10

```
int[] scores = new int[10];
System.out.println(scores.length); //length of 10
```

- And this array of Customers has a length of 2

```
Customer[] customers = {new Customer(“Brandon”), new
Customer(“Betsy”)};
System.out.println(customers.length); //length of 2
```

- *length* is an instance variable, not a method
  - On the other hand, Strings have a *length()* method

Length of An Array

Every array has a *length* variable, that tells how large the array is

- This array of ints has a length of 10

```
int[] scores = new int[10];
System.out.println(scores.length); //length of 10
```

- And this array of Customers has a length of 2

```
Customer[] customers = {new Customer(“Brandon”), new
Customer(“Betsy”)};
System.out.println(customers.length); //length of 2
```

- *length* is an instance variable, not a method
  - On the other hand, Strings have a *length()* method
- Arrays cannot be easily resized
  - You’d have to create a new array, copy everything from the old array, and add the new elements to the new array

Stepping Through an Array – *for* Loop

You can use a *for* loop to visit (and/or set) every element in an array, using its index

- Here we iterate over an array using its *length*, where i represents the index of each item

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

Stepping Through an Array – *for* Loop

You can use a *for* loop to visit (and/or set) every element in an array, using its index

- Here we iterate over an array using its *length*, where i represents the index of each item

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

- The name i is traditional for loops
    - i is instantly recognizable as the index of an enclosing for loop
    - Note, inner (nested) loops typically use j, then k

Stepping Through an Array – *for* Loop

You can use a *for* loop to visit (and/or set) every element in an array, using its index

- Here we iterate over an array using its *length*, where i represents the index of each item

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

- The name i is traditional for loops
  - i is instantly recognizable as the index of an enclosing for loop
  - Note, inner (nested) loops typically use j, then k
- Use of *length* is always preferred over using a constant (hard-coded) value (such as 10)
  - Try not to do this: `for (int i = 0; i < 10; i++) { … }`

Stepping Through an Array – *Enhanced for* Loop

You can also use an *enhanced for* loop to visit every element in an array

- This is like the *for x in list* syntax in Python
- Here we iterate over an array of Strings, where `n` represents each item in the array

```
for (String n : names) {
    System.out.println(“Name: “ + n);
}
```

Stepping Through an Array – *Enhanced for* Loop

You can also use an *enhanced for* loop to visit every element in an array

- This is like the *for x in list* syntax in Python
- Here we iterate over an array of Strings, where `n` represents each item in the array

```
for (String n : names) {
    System.out.println(“Name: “ + n);
}
```

- This simple structure allows you to visit each element of an array without explicitly expressing how to go from element to element using an index

Default Values for primitives

- If you declare a variable to have a given primitive type, for example:
  ```
  int age;
  double weight;
  boolean graduated;
  ```
- ... and if you have not yet assigned a value to it, for example, with:
  ```
  age = 23;
  weight = 145.6;
  graduated = true;
  ```
- … then the default value of that variable is:
  0 for non floating-point types (e.g. int, byte, short, long)
  0.0 for floating-point types (e.g. double, float)
  false for booleans

Default Values for Objects

- If you declare a variable to have a given Object (not primitive) type, for example:
```
Person john;
String name;
```
- ... and if you have not yet assigned a value to it, for example, with:
```
john = new Person();
name = “John Smith”;
```
- … then the default value of that variable is null
- null is a legal value, but there isn’t much you can do with it
  - It’s an error to refer to its fields, because it has none
  - It’s an error to send a message to it, because it has no methods
  - null is basically a pointer that doesn’t point to anything
  - It’s very similar to Python’s *None*

Default Values for Arrays

- When you create an array of primitives or Objects, but you haven't yet given values to each element in that array, the slots will have the default values associated with the type of array
  - primitives in a numeric array will default to 0 (or 0.0)
  - primitives in a boolean array will default to false
  - Objects in an array of Objects will default to null

- For example, here we create an array of 100 ints

```
int[] count = new int[100];
```

  - Every value in this array will default to 0

- We can then iterate over the array to set each value to a new int

```
for (int i = 0; i < count.length; i++) {
    count[i] = i + 1; //set each value to index i plus 1
}
```

Copying Arrays

- Array assignment does not copy array values
  - This is equivalent to the concept of assignment *by reference* in Python

Copying Arrays

- Array assignment does not copy array values
    - This is equivalent to the concept of assignment *by reference* in Python
- Given an array, we *can not* copy it directly

```
//Define array a
int[] a = {1, 8, 3};

//Create an array b of same size as a
int[] b = new int[a.length];

//Set b = a
//This does NOT copy elements of a to b
//It only makes b refer to same array object [1, 8, 3]
b = a;
```

Copying Arrays

- Array assignment does not copy array values
  - This is equivalent to the concept of assignment *by reference* in Python
- Given an array, we *can not* copy it directly

```
//Define array a
int[] a = {1, 8, 3};

//Create an array b of same size as a
int[] b = new int[a.length];

//Set b = a
//This does NOT copy elements of a to b
//It only makes b refer to same array object [1, 8, 3]
b = a;
```

- How do we know? Use == to compare object references

```
System.out.println(a == b); //true
```

Copying Arrays – Copy Elements

- You *can*, however, create a new array and copy the elements directly

```
//Define array a
int[] a = {1, 8, 3};

//Create an array b of same size as a
int[] b = new int[a.length];

//Copy elements of a to b
for (int i = 0; i < b.length; i++) {
    b[i] = a[i];
}
```

Copying Arrays – Copy Elements

- You *can*, however, create a new array and copy the elements directly

```
//Define array a
int[] a = {1, 8, 3};

//Create an array b of same size as a
int[] b = new int[a.length];

//Copy elements of a to b
for (int i = 0; i < b.length; i++) {
    b[i] = a[i];
}
```

- Use == to compare the objects

```
System.out.println(a == b); //false
```

Copying Arrays – Copy Elements

- You *can*, however, create a new array and copy the elements directly

```
//Define array a
int[] a = {1, 8, 3};

//Create an array b of same size as a
int[] b = new int[a.length];

//Copy elements of a to b
for (int i = 0; i < b.length; i++) {
    b[i] = a[i];
}
```

- Use == to compare the objects

```
System.out.println(a == b); //false
```

- And use the *Arrays.equals* method to compare the actual array contents (values)

```
System.out.println(Arrays.equals(a, b)); //true
```

Copying Arrays - Cloning

- You can also clone (create an exact copy of) an array using the *clone* method
  - Many Java Objects support cloning

```
//Define array a
int[] a = {1, 8, 3};

//Copy elements of a to b
int[] b = a.clone();
```

Copying Arrays - Cloning

- You can also clone (create an exact copy of) an array using the *clone* method
  - Many Java Objects support cloning

```
//Define array a
int[] a = {1, 8, 3};

//Copy elements of a to b
int[] b = a.clone();
```

- Use == to compare the objects

```
System.out.println(a == b); //false
```

Copying Arrays - Cloning

- You can also clone (create an exact copy of) an array using the *clone* method
  - Many Java Objects support cloning

```
//Define array a
int[] a = {1, 8, 3};

//Copy elements of a to b
int[] b = a.clone();
```

- Use == to compare the objects

```
System.out.println(a == b); //false
```

- Compare the actual array contents (values)

```
System.out.println(Arrays.equals(a, b)); //true
```

Array of Arrays

- Elements of an array can be arrays themselves
- The following creates an array of 3 arrays, each of which points to an array of 2 ints

```
int[][] table = new int[3][2];
```

Array of Arrays

- Elements of an array can be arrays themselves
- The following creates an array of 3 arrays, each of which points to an array of 2 ints
  `int[][] table = new int[3][2];`
- Then populates each slot in the array with an incremented count

```
int count = 1;
for (int i = 0; i < table.length; i++) { //get the length of the rows (vertical)
    for (int j = 0; j < table[i].length; j++) { //get the length of the columns
(horizontal)
        table[i][j] = count++; //set count value in each array slot, then increment
    }
}
```

Array of Arrays

- Elements of an array can be arrays themselves
- The following creates an array of 3 arrays, each of which points to an array of 2 ints

```
int[][] table = new int[3][2];
```

- Then populates each slot in the array with an incremented count

```
int count = 1;
for (int i = 0; i < table.length; i++) { //get the length of the rows (vertical)
    for (int j = 0; j < table[i].length; j++) { //get the length of the columns 
(horizontal)
        table[i][j] = count++; //set count value in each array slot, then increment
    }
}
```

- This is like a “table” of 3 rows and 2 columns
  - table.length is 3
  - table[0].length is 2

Array of Arrays

- This is the same as defining and directly populating a 2-dimensional array like so

```
int[][] table2 = {
    {1, 2},
    {3, 4},
    {5, 6}
};
```

Array of Arrays

- This is the same as defining and directly populating a 2-dimensional array like so

```
int[][] table2 = {
    {1, 2},
    {3, 4},
    {5, 6}
};
```

- To compare 2 nested arrays, you can use the *Arrays.deepEquals* method to do a deep comparison

```
System.out.println(Arrays.deepEquals(table, table2)); //true
```

Array of Arrays

- You can also have non-rectangular arrays
- Here we update the 3rd element in the array to point to an array of 20 ints

```
table2[2] = new int[20];
```

Array of Arrays

- You can also have non-rectangular arrays
- Here we update the 3rd element in the array to point to an array of 20 ints
```
table2[2] = new int[20];
```
- This is the same as initializing a 2-dimensional array like so
```
int[][] table2 = {
    {1, 2},
    {3, 4},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

Accessing Values in Array of Arrays

- To access the value in any slot in a 2-dimensional array, specify the index number of the row inside brackets [], followed by the index number of the column inside brackets []
- For example:

```
int[][] table2 = {
    {1, 2},
    {3, 4},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

- To print the value in row 0 and column 0, you'd use:

```
System.out.println(table2[0][0]); //prints 1
```

Accessing Values in Array of Arrays

- To access the value in any slot in a 2-dimensional array, specify the index number of the row inside brackets [], followed by the index number of the column inside brackets []
- For example:

```
int[][] table2 = {
    {1, 2},
    {3, 4},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

- To print the value in row 0 and column 0, you'd use:

```
System.out.println(table2[0][0]); //prints 1
```

- To print the value in row 2 and column 2, you'd use:

```
System.out.println(table2[2][2]); //prints 0
```

Array Methods

- Arrays have very few attributes/methods
- *length* is useful, but there is no *add*, *remove*, *reverse*, etc.

Ref: https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

ArrayLists

ArrayLists

- An ArrayList is like an array, but much more flexible
  - It’s just like a list in Python

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- An ArrayList is like an array, but much more flexible
  - It’s just like a list in Python
- ArrayLists are *not* defined with a fixed number of slots – they have a *variable* length

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- An ArrayList is like an array, but much more flexible
  - It’s just like a list in Python
- ArrayLists are *not* defined with a fixed number of slots – they have a *variable* length
- ArrayLists can *only* contain Objects and you *can't* store different types in a single ArrayList

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- An ArrayList is like an array, but much more flexible
  - It’s just like a list in Python
- ArrayLists are *not* defined with a fixed number of slots – they have a *variable* length
- ArrayLists can *only* contain Objects and you *can't* store different types in a single ArrayList
- ArrayLists are part of Java’s *Collections Framework*
  - *Collections* are defined in java.util
  - To use ArrayLists specifically, you have to import java.util.ArrayList
  - All *Collections* share similar methods (*add*, *remove*, *size*, etc.)
  - We’ll learn more about *Collections* later in this course

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- The syntax to create an ArrayList is:
  ```
  ArrayList<Data Type> myArrayList = new ArrayList<Data Type>();
  ```

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- The syntax to create an ArrayList is:
  `ArrayList<Data Type> myArrayList = new ArrayList<Data Type>();`
- Since you can't store primitive values in an ArrayList, you have to use the *wrapper* classes associated with primitive types
  - For int, use Integer; for double use Double, etc.
  - These are essentially the Object versions of the primitive data types, with additional methods/attributes

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- The syntax to create an ArrayList is:
  `ArrayList<Data Type> myArrayList = new ArrayList<Data Type>();`
- Since you can't store primitive values in an ArrayList, you have to use the *wrapper* classes associated with primitive types
  - For int, use Integer; for double use Double, etc.
  - These are essentially the Object versions of the primitive data types, with additional methods/attributes
- This creates an ArrayList of Integers
  `ArrayList<Integer> numberList = new ArrayList<Integer>();`

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

ArrayLists

- The syntax to create an ArrayList is:
  `ArrayList<Data Type> myArrayList = new ArrayList<Data Type>();`
- Since you can't store primitive values in an ArrayList, you have to use the *wrapper* classes associated with primitive types
  - For int, use Integer; for double use Double, etc.
  - These are essentially the Object versions of the primitive data types, with additional methods/attributes
- This creates an ArrayList of Integers
  `ArrayList<Integer> numberList = new ArrayList<Integer>();`
- And this creates an ArrayList of Strings
  `ArrayList<String> stringList = new ArrayList<String>();`

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Size of An ArrayList

- Every ArrayList has a *size* method, that tells how large the ArrayList is
- This ArrayList of Integers has a size of 3

```
ArrayList<Integer> scores = new ArrayList<Integer>();
scores.add(23); //adds element 23
scores.add(15); //adds element 15
scores.add(0); //adds element 0
System.out.println(scores.size()); //size of 3
```

Size of An ArrayList

- Every ArrayList has a *size* method, that tells how large the ArrayList is
- This ArrayList of Integers has a size of 3

```
ArrayList<Integer> scores = new ArrayList<Integer>();
scores.add(23); //adds element 23
scores.add(15); //adds element 15
scores.add(0); //adds element 0
System.out.println(scores.size()); //size of 3
```

- *size* is a method, not a variable

Size of An ArrayList

- Every ArrayList has a *size* method, that tells how large the ArrayList is
- This ArrayList of Integers has a size of 3

```
ArrayList<Integer> scores = new ArrayList<Integer>();
scores.add(23); //adds element 23
scores.add(15); //adds element 15
scores.add(0); //adds element 0
System.out.println(scores.size()); //size of 3
```

- *size* is a method, not a variable
- ArrayLists can be easily resized
  - You don't initialize ArrayLists with a specific size
  - You can add/remove elements without worrying about it
  - ArrayLists will take care of the resizing for you

ArrayList Methods

- ArrayLists have *many* attributes/methods
- There is *add*, *remove*, *size*, *get*, etc.

Ref: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

toString

Printing primitives vs. Objects

- You can *easily* print primitive values in Java

```
int a = 5;
System.out.println(“a = “ + a); //prints “a = 5”
```

Printing primitives vs. Objects

- You can *easily* print primitive values in Java
```
int a = 5;
System.out.println("a = " + a); //prints "a = 5"
```
- You know what to expect when you print a primitive
```
boolean b = false;
System.out.println("b = " + b); //prints "b = false"
```

Printing primitives vs. Objects

- You can *easily* print primitive values in Java
```
int a = 5;
System.out.println("a = " + a); //prints "a = 5"
```
- You know what to expect when you print a primitive
```
boolean b = false;
System.out.println("b = " + b); //prints "b = false"
```
- Printing Objects is not as straightforward
```
Customer c = new Customer("Brandon");
System.out.println("c = " + c); //what would you expect this to print?
```

Printing primitives vs. Objects

- You can *easily* print primitive values in Java
  ```
  int a = 5;
  System.out.println(“a = “ + a); //prints “a = 5”
  ```
- You know what to expect when you print a primitive
  ```
  boolean b = false;
  System.out.println(“b = “ + b); //prints “b = false”
  ```
- Printing Objects is not as straightforward
  ```
  Customer c = new Customer(“Brandon”);
  System.out.println(“c = “ + c); //what would you expect this to print?
  ```
- You need to tell Java what to print by defining the *toString* method in the class
  - For reference, this is the same as defining the __*str*__ method in a Python class
  - The *toString* method must return a String

Printing primitives vs. Objects

- You can *easily* print primitive values in Java
```
int a = 5;
System.out.println("a = " + a); //prints "a = 5"
```
- You know what to expect when you print a primitive
```
boolean b = false;
System.out.println("b = " + b); //prints "b = false"
```
- Printing Objects is not as straightforward
```
Customer c = new Customer("Brandon");
System.out.println("c = " + c); //what would you expect this to print?
```
- You need to tell Java what to print by defining the *toString* method in the class
  - For reference, this is the same as defining the *__str__* method in a Python class
  - The *toString* method must return a String
- The syntax of the *toString* method is:
```
public String toString() {
    return "someString";
}
```

Printing primitives vs. Objects

- Below is an implementation of the *toString* method in a Customer class

```
public class Customer {

    //Name of Customer
    String name;

    public Customer(String name) {
        this.name = name;
    }

    //toString method must return a String
    public String toString() {
       return this.name; //return name, to be printed by Java
    }
}
```

Printing primitives vs. Objects

- Below is an implementation of the *toString* method in a Customer class

```
public class Customer {

    //Name of Customer
    String name;

    public Customer(String name) {
        this.name = name;
    }

    //toString method must return a String
    public String toString() {
       return this.name; //return name, to be printed by Java
    }
}
```

- Now, print a customer

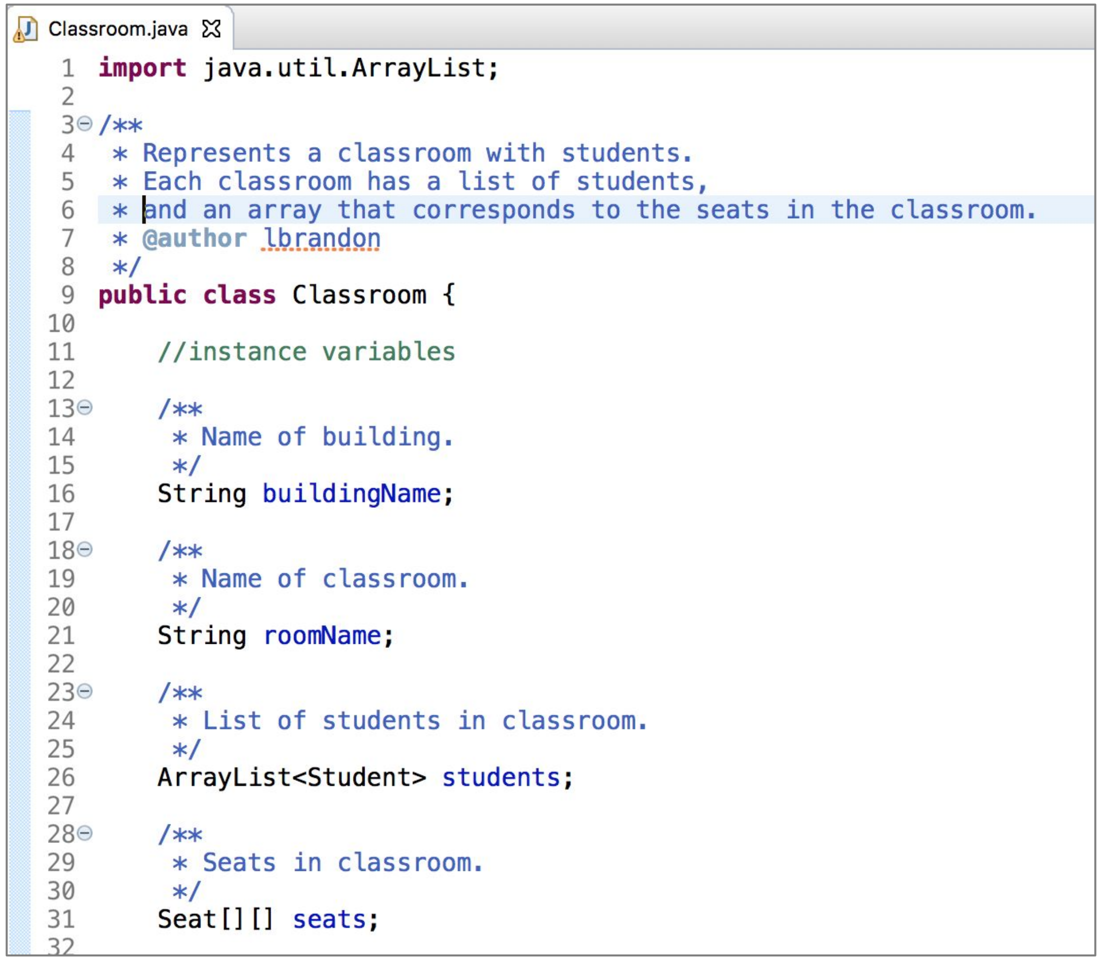
```
Customer c = new Customer(“Brandon”);
System.out.println(“c = “ + c); //prints “c = Brandon”
```

Classroom Project

Classroom Project

- In Eclipse, create a new “Classroom” project
    - This program will represent an actual classroom, with ways (methods) of adding students and assigning seats
- Create 3 classes:
    - Classroom
        - The classroom itself, with seats and students
        - Make sure public static void main(String[] args) IS checked
    - Seat
        - A seat in the classroom
    - Student
        - A student assigned to the classroom and sitting in a seat

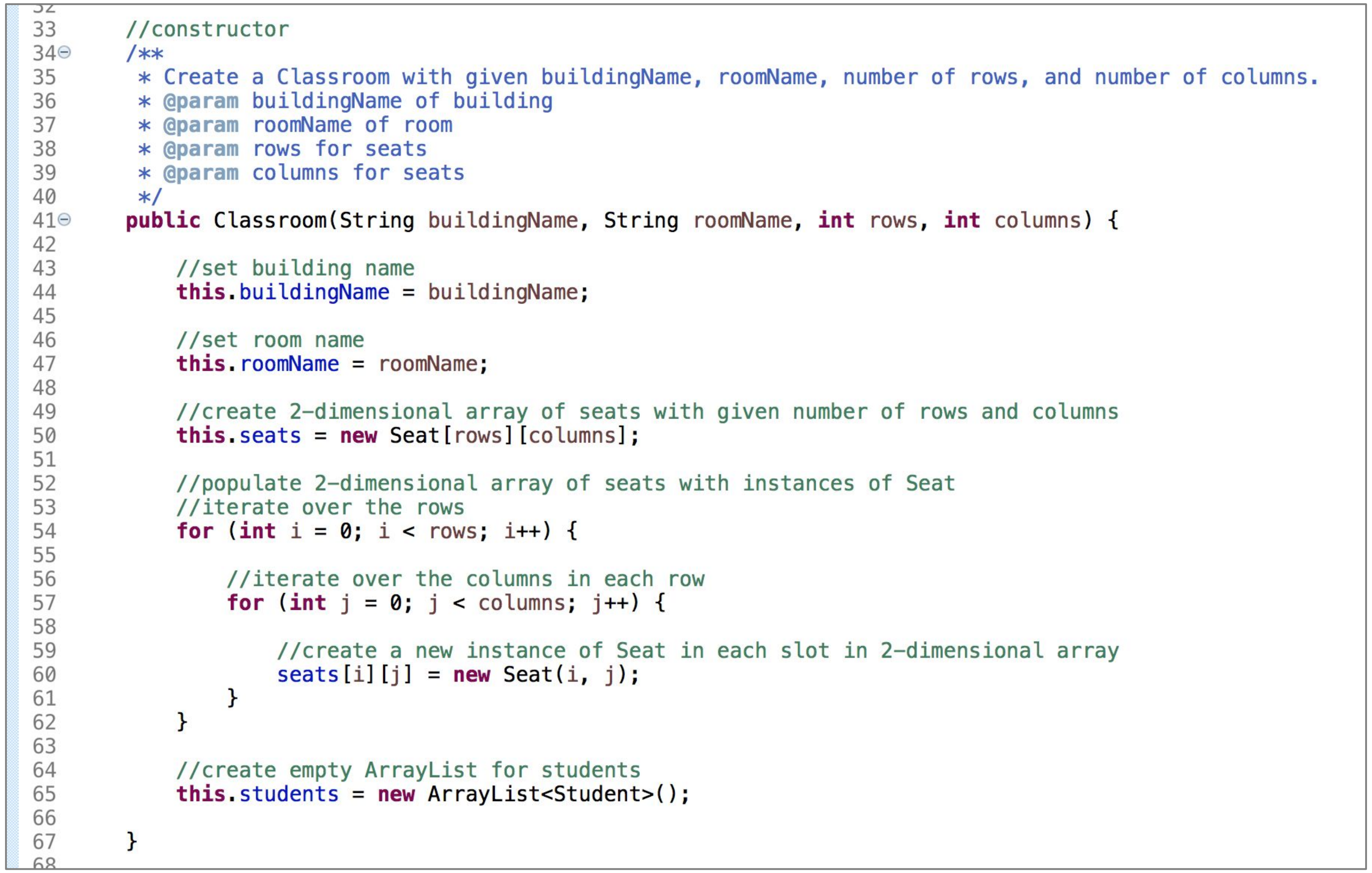
Classroom Project – Classroom Class



Classroom.java

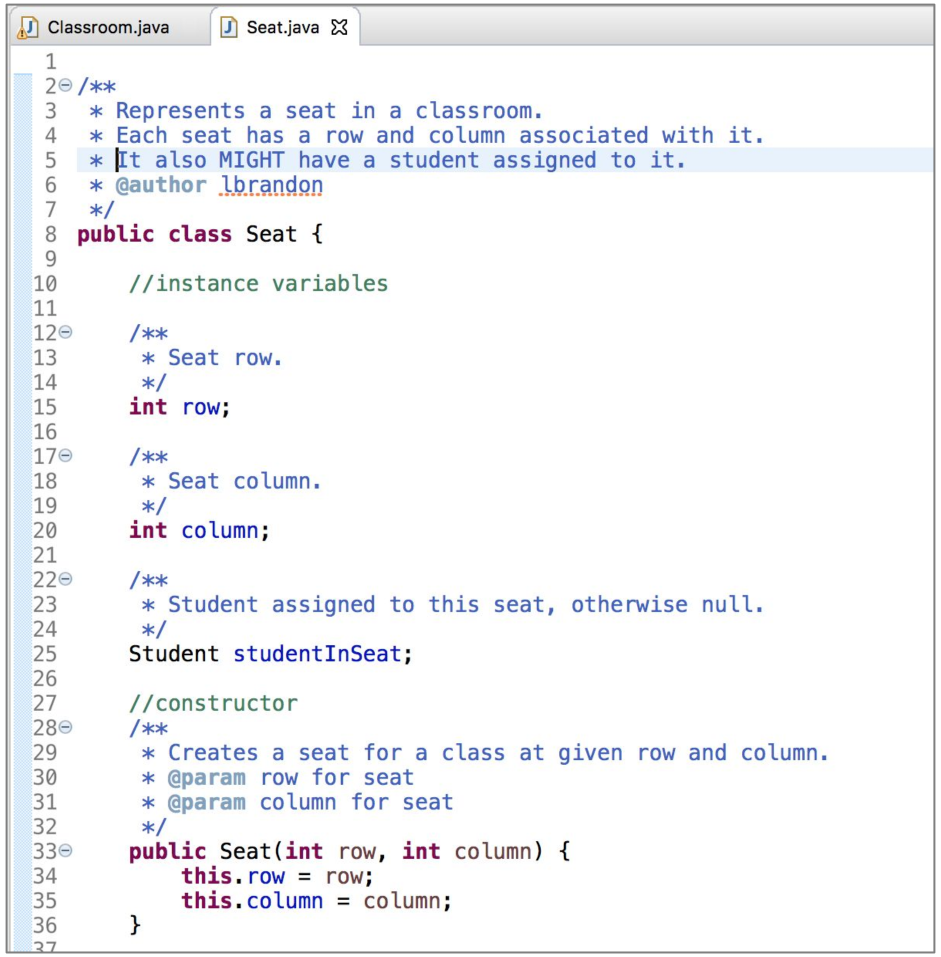
```
import java.util.ArrayList;

/**
 * Represents a classroom with students.
 * Each classroom has a list of students,
 * and an array that corresponds to the seats in the classroom.
 * @author lbrandon
 */
public class Classroom {

    //instance variables

    /**
     * Name of building.
     */
    String buildingName;

    /**
     * Name of classroom.
     */
    String roomName;

    /**
     * List of students in classroom.
     */
    ArrayList<Student> students;

    /**
     * Seats in classroom.
     */
    Seat[][] seats;

```

Classroom Project – Classroom Class



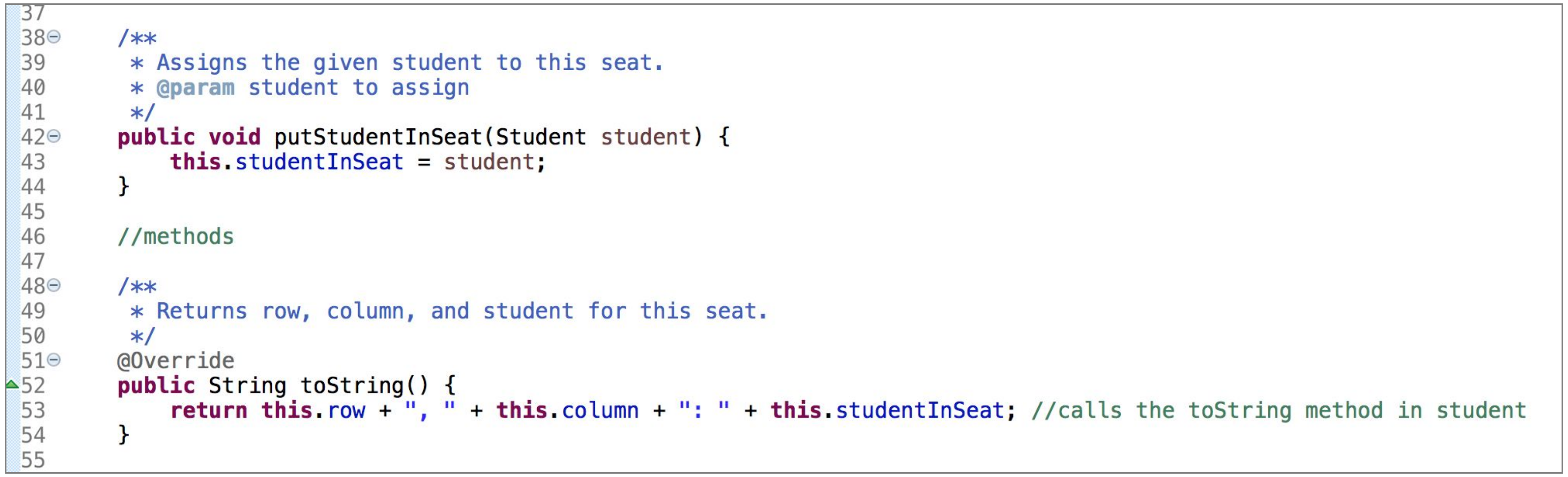
```
    //constructor
    /**
     * Create a Classroom with given buildingName, roomName, number of rows, and number of columns.
     * @param buildingName of building
     * @param roomName of room
     * @param rows for seats
     * @param columns for seats
     */
    public Classroom(String buildingName, String roomName, int rows, int columns) {

        //set building name
        this.buildingName = buildingName;

        //set room name
        this.roomName = roomName;

        //create 2-dimensional array of seats with given number of rows and columns
        this.seats = new Seat[rows][columns];

        //populate 2-dimensional array of seats with instances of Seat
        //iterate over the rows
        for (int i = 0; i < rows; i++) {

            //iterate over the columns in each row
            for (int j = 0; j < columns; j++) {

                //create a new instance of Seat in each slot in 2-dimensional array
                seats[i][j] = new Seat(i, j);
            }
        }

        //create empty ArrayList for students
        this.students = new ArrayList<Student>();

    }
```

Classroom Project – Seat Class



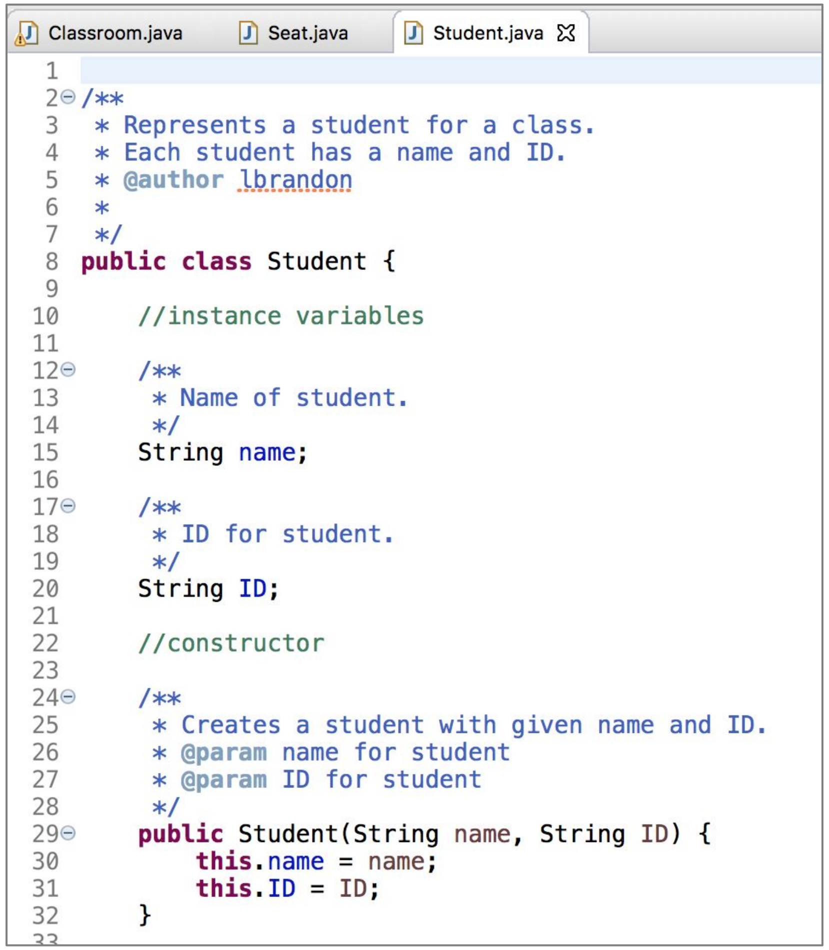
```java

/**
 * Represents a seat in a classroom.
 * Each seat has a row and column associated with it.
 * It also MIGHT have a student assigned to it.
 * @author lbrandon
 */
public class Seat {

    //instance variables

    /**
     * Seat row.
     */
    int row;

    /**
     * Seat column.
     */
    int column;

    /**
     * Student assigned to this seat, otherwise null.
     */
    Student studentInSeat;

    //constructor
    /**
     * Creates a seat for a class at given row and column.
     * @param row for seat
     * @param column for seat
     */
    public Seat(int row, int column) {
        this.row = row;
        this.column = column;
    }
```

Classroom Project – Seat Class



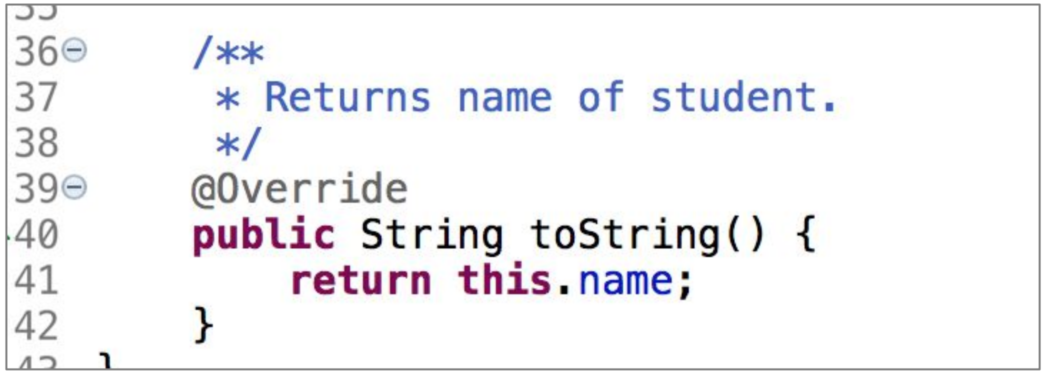
```

    /**
     * Assigns the given student to this seat.
     * @param student to assign
     */
    public void putStudentInSeat(Student student) {
        this.studentInSeat = student;
    }

    //methods

    /**
     * Returns row, column, and student for this seat.
     */
    @Override
    public String toString() {
        return this.row + ", " + this.column + ": " + this.studentInSeat; //calls the toString method in student
    }

```

Classroom Project – Student Class



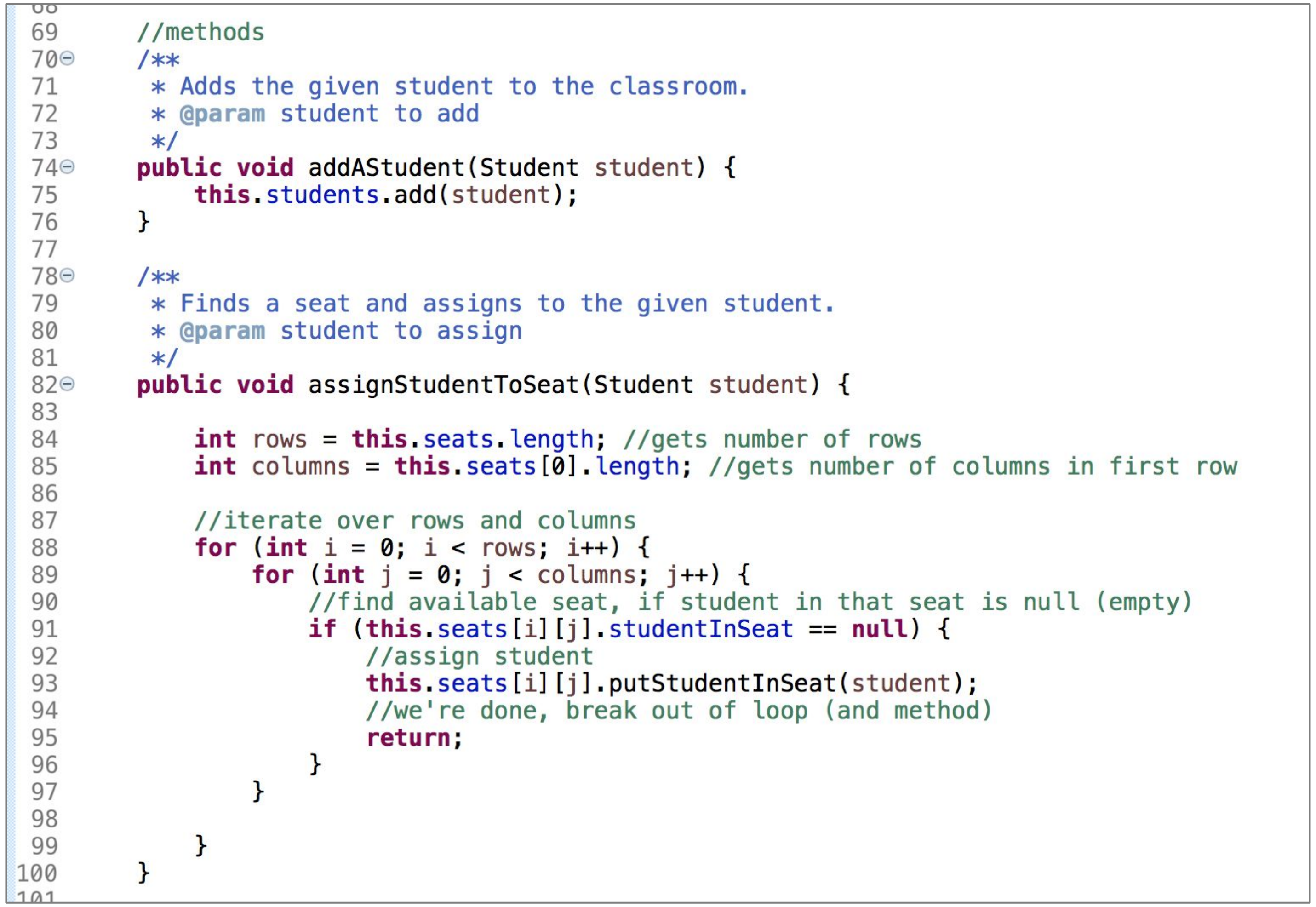
```java

/**
 * Represents a student for a class.
 * Each student has a name and ID.
 * @author lbrandon
 *
 */
public class Student {

    //instance variables

    /**
     * Name of student.
     */
    String name;

    /**
     * ID for student.
     */
    String ID;

    //constructor

    /**
     * Creates a student with given name and ID.
     * @param name for student
     * @param ID for student
     */
    public Student(String name, String ID) {
        this.name = name;
        this.ID = ID;
    }
```

Classroom Project – Student Class



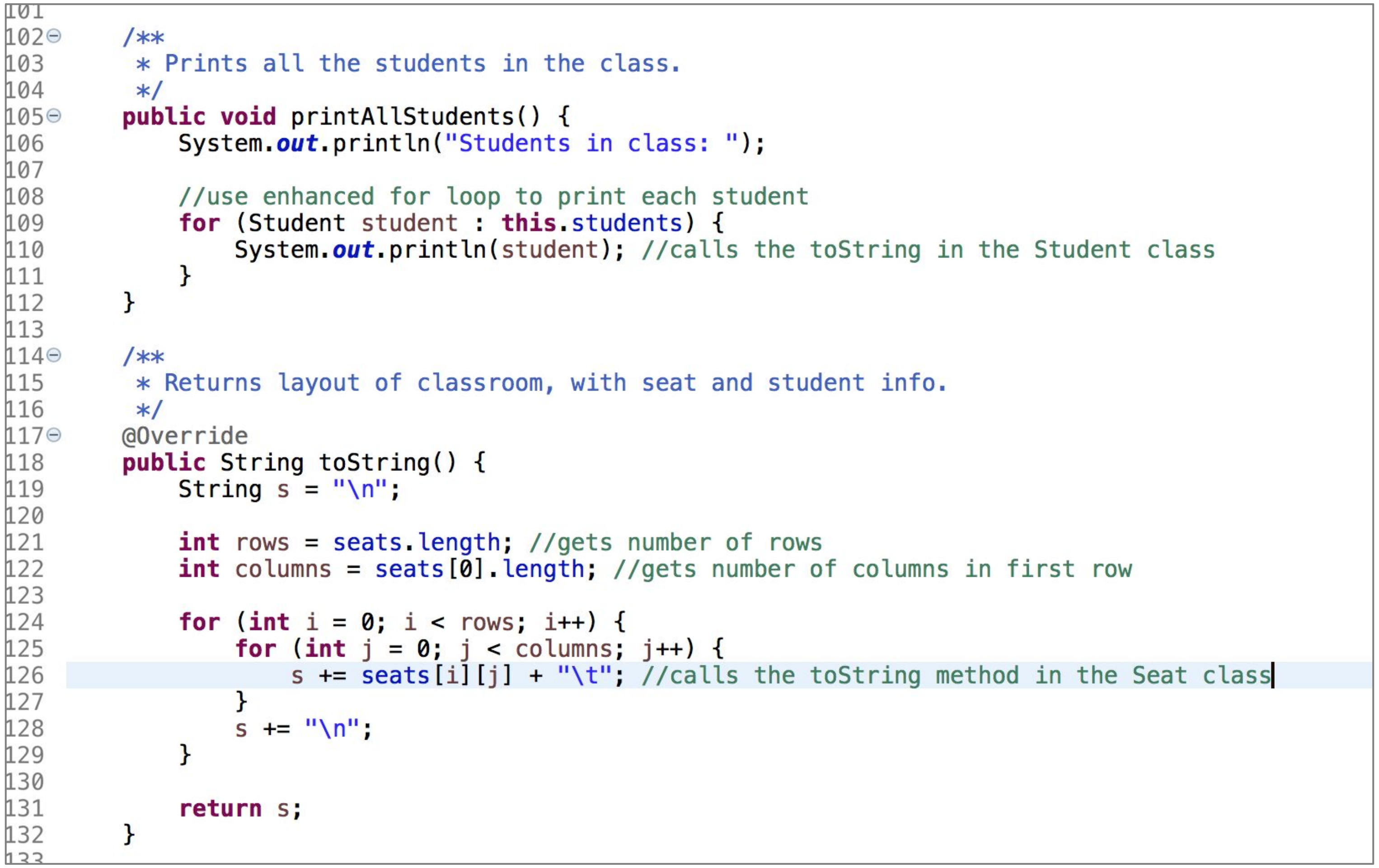
```
    /**
     * Returns name of student.
     */
    @Override
    public String toString() {
        return this.name;
    }
}
```

Classroom Project – Classroom Class



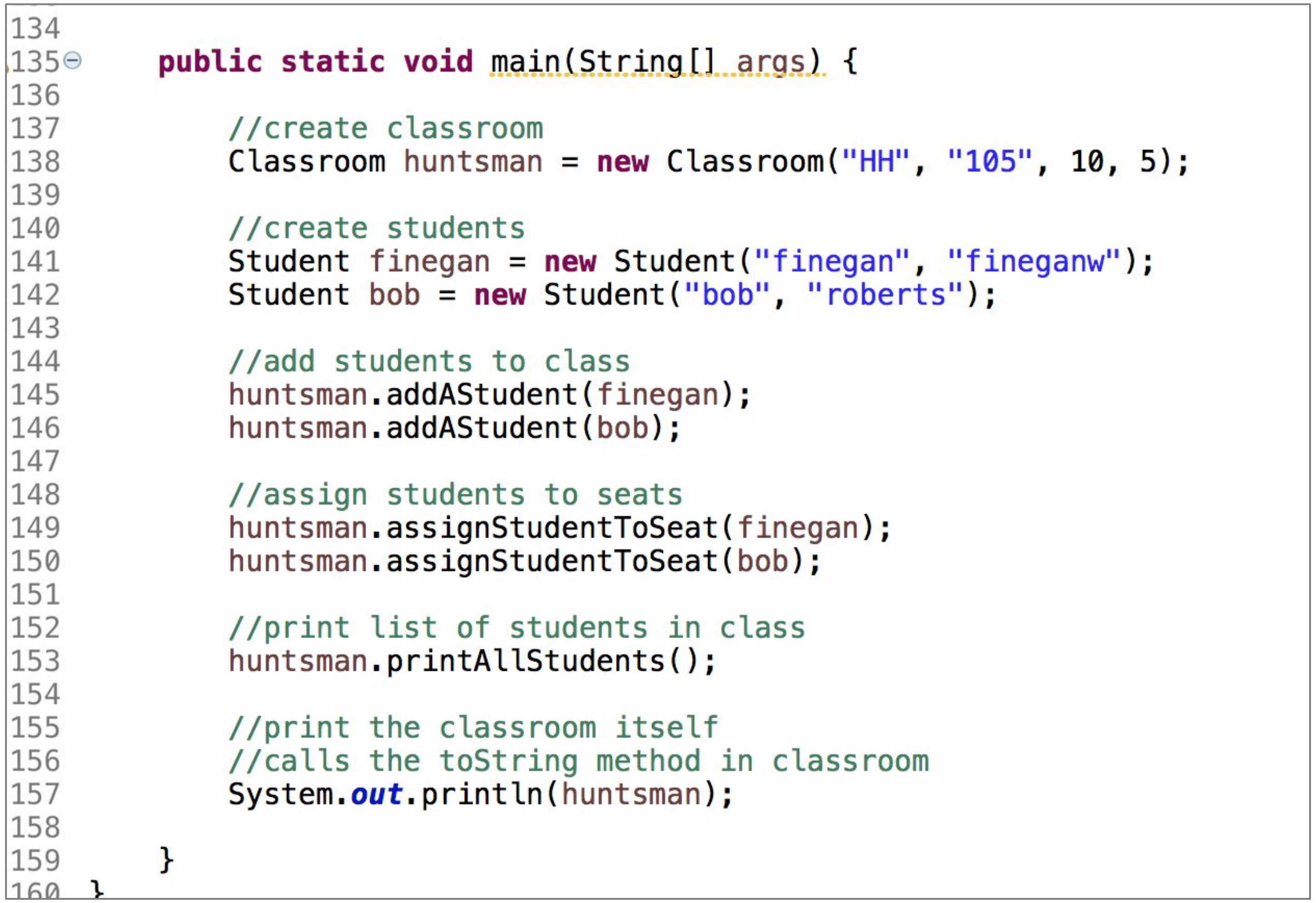
```
    //methods
    /**
     * Adds the given student to the classroom.
     * @param student to add
     */
    public void addAStudent(Student student) {
        this.students.add(student);
    }

    /**
     * Finds a seat and assigns to the given student.
     * @param student to assign
     */
    public void assignStudentToSeat(Student student) {

        int rows = this.seats.length; //gets number of rows
        int columns = this.seats[0].length; //gets number of columns in first row

        //iterate over rows and columns
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                //find available seat, if student in that seat is null (empty)
                if (this.seats[i][j].studentInSeat == null) {
                    //assign student
                    this.seats[i][j].putStudentInSeat(student);
                    //we're done, break out of loop (and method)
                    return;
                }
            }

        }
    }
```

Classroom Project – Classroom Class



```java
    /**
     * Prints all the students in the class.
     */
    public void printAllStudents() {
        System.out.println("Students in class: ");

        //use enhanced for loop to print each student
        for (Student student : this.students) {
            System.out.println(student); //calls the toString in the Student class
        }
    }

    /**
     * Returns layout of classroom, with seat and student info.
     */
    @Override
    public String toString() {
        String s = "\n";

        int rows = seats.length; //gets number of rows
        int columns = seats[0].length; //gets number of columns in first row

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                s += seats[i][j] + "\t"; //calls the toString method in the Seat class
            }
            s += "\n";
        }

        return s;
    }
```

Classroom Project – Classroom Class



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

        //create classroom
        Classroom huntsman = new Classroom("HH", "105", 10, 5);

        //create students
        Student finegan = new Student("finegan", "fineganw");
        Student bob = new Student("bob", "roberts");

        //add students to class
        huntsman.addAStudent(finegan);
        huntsman.addAStudent(bob);

        //assign students to seats
        huntsman.assignStudentToSeat(finegan);
        huntsman.assignStudentToSeat(bob);

        //print list of students in class
        huntsman.printAllStudents();

        //print the classroom itself
        //calls the toString method in classroom
        System.out.println(huntsman);

    }
}
```