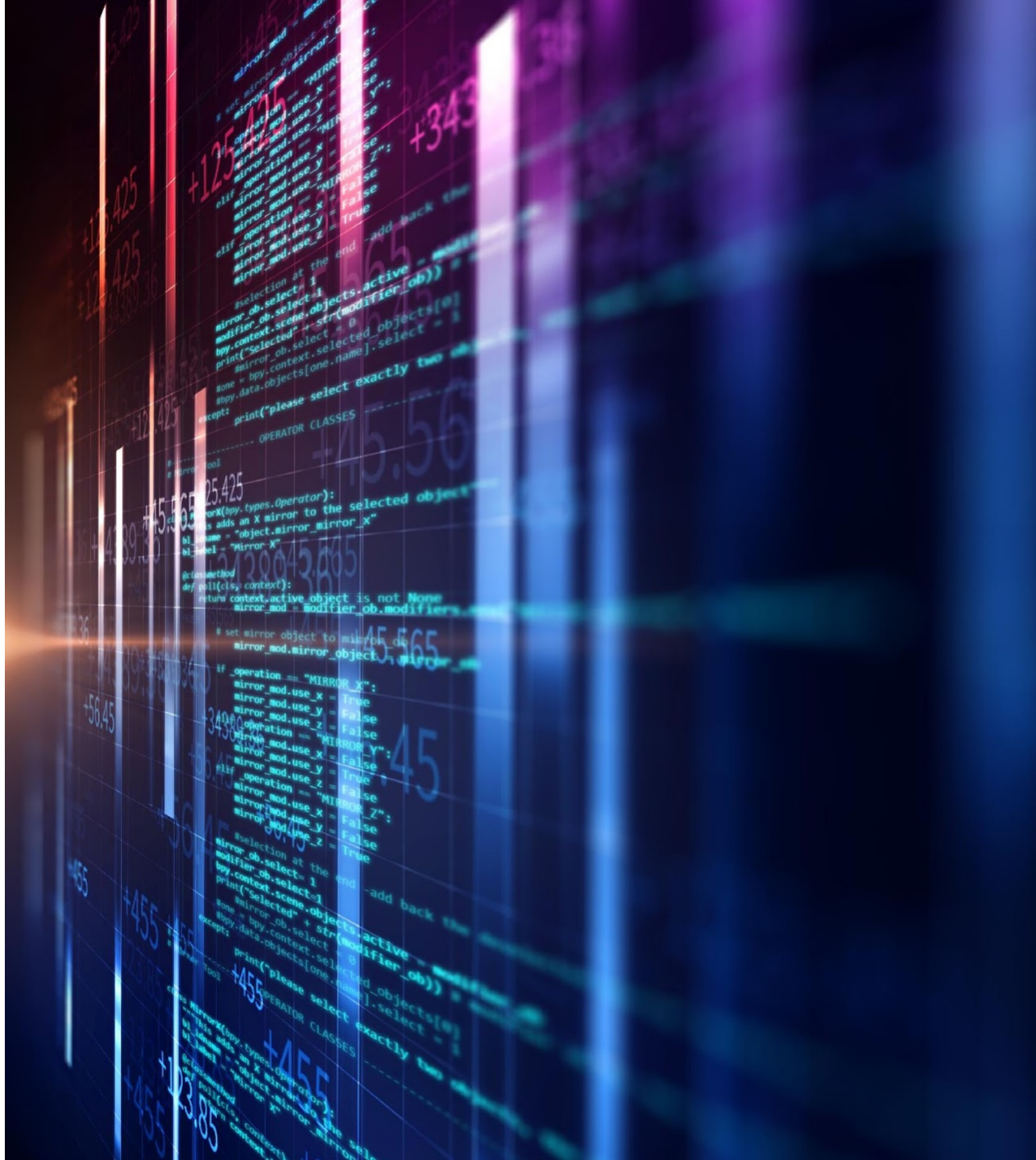
2D Arrays

Brandon Krakowsky

Penn Engineering



2D Arrays

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

- 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
- To print an array of arrays, you can use the *Arrays.deepToString* method

```
System.out.println(Arrays.deepToString(table));
```

  - This prints a String representation of each nested array

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
```

Iterate Over Array of Arrays

- Here's how to iterate over an arrays of arrays using a nested for loop

```
//iterate over the length of the array (the rows)
for (int i = 0; i < table2.length; i++) {
    //for each of those, iterate over the length of the row (the columns)
    for (int j = 0; j < table2[i].length; j++) {
        //print the item at index i (row) and index j (column)
        System.out.print(table2[i][j] + " ");
    }

    System.out.println(); //add line break after printing each row
}
```

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
```

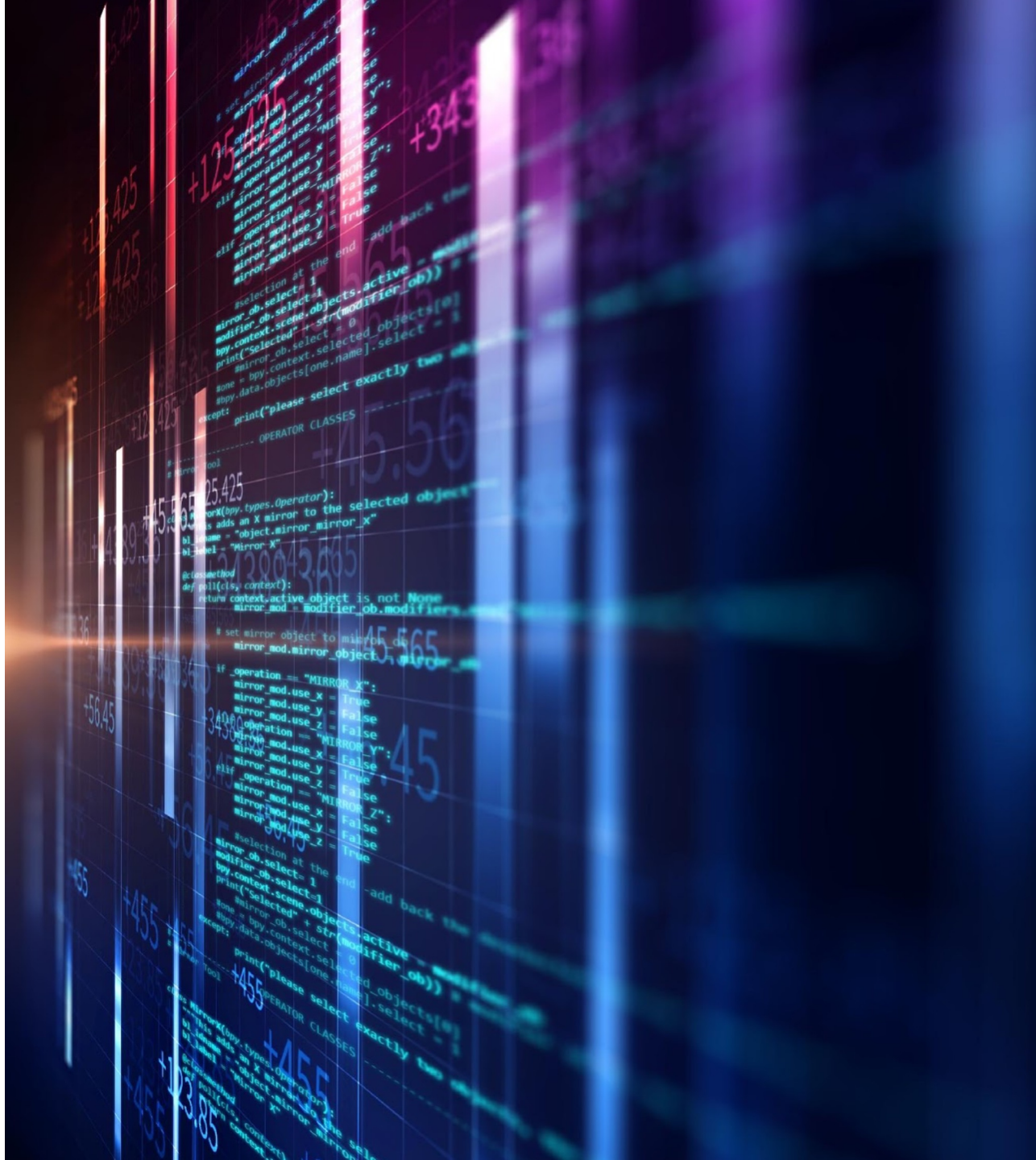
- To print the value in row 2 and column 2, you'd use:

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

ArrayLists

Brandon Krakowsky

Penn Engineering



ArrayLists

ArrayLists

- An ArrayList is like an array, but much more flexible
  - For reference, 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
  - For reference, 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
  - For reference, 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
  - For reference, 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

ArrayList Examples

- Define a method *uniqueArrayList* that takes an array of doubles as a *parameter* and *returns* an ArrayList with the unique values

ArraysAndArrayLists.java

```
1  import java.util.ArrayList;
2
3  /**
4   * Examples with arrays and arraylists.
5   * @author lbrandon
6   *
7   */
8  public class ArraysAndArrayLists {
9
```

ArrayList Examples

- Define a method *uniqueArrayList* that takes an array of doubles as a *parameter* and *returns* an ArrayList with the unique values

```
10    /**
11     * Returns an arraylist of unique values from the given array.
12     * @param arr of values
13     * @return unique values from given array
14     */
15    ArrayList<Double> uniqueArrayList(double[] arr) {
16
17        //create empty arraylist of doubles
18        ArrayList<Double> uniqueValues = new ArrayList<Double>();
19
20        //iterate over given array
21        for (double i : arr) {
22            //if arraylist does not already contain the double
23            if (!uniqueValues.contains(i)) {
24                //add to arraylist
25                uniqueValues.add(i);
26            }
27        }
28
29        return uniqueValues;
30    }
31
```

ArrayList Examples

- Define a method *uniqueArrayList* that takes an array of doubles as a *parameter* and *returns* an ArrayList with the unique values

```
51    public static void main(String[] args) {
52
53        ArraysAndArrayLists a = new ArraysAndArrayLists();
54
55        double[] arr = {1.0,2.9,3.1,3.1,3.1,3.1,4.3,5.8};
56        ArrayList<Double> uniqueArrValues = a.uniqueArrayList(arr);
57        System.out.println(uniqueArrValues);
58
```

ArrayList Examples

- Define a method *getFactors* that takes an integer as a *parameter* and *returns* an ArrayList of factors of that number
  - Basically, find the numbers between 1 and the given integer that divide the number evenly

```
32    /**
33     * Returns an arraylist of factors of the given number.
34     * @param i value of which to get the factors
35     * @return factors
36     */
37    ArrayList<Integer> getFactors(int x) {
38
39        //create empty arraylist of integers
40        ArrayList<Integer> factors = new ArrayList<Integer>();
41
42        //To find the possible factors, check for division by the numbers 1 to x
43        for (int i = 1; i < x + 1; i++) {
44            if (x % i == 0) {
45                factors.add(i); //append to arraylist
46            }
47        }
48
49        return factors;
50    }
51
```

ArrayList Examples

- Define a method *getFactors* that takes an integer as a *parameter* and *returns* an ArrayList of factors of that number
  - Basically, find the numbers between 1 and the given integer that divide the number evenly

```
60
61          System.out.println(a.getFactors(21));
```

ArrayList Examples

- Define a method *createArrayListFromArray* that takes an array of integers as a *parameter* and *returns* an ArrayList of integers

```
53
54    /**
55     * Creates an ArrayList from array of integers.
56     * @param arr of integers
57     * @return ArrayList of given integers
58     */
59    ArrayList<Integer> createArrayListFromArray(Integer[] arr) {
60
61        //create arraylist from a list of the given array of integers
62        //use Arrays.asList to convert array to list
63        ArrayList<Integer> arrayListofIntegers = new ArrayList<Integer>(Arrays.asList(arr));
64
65        return arrayListofIntegers;
66    }
67
```

ArrayList Examples

- Define a method *createArrayListFromArray* that takes an array of integers as a *parameter* and *returns* an ArrayList of integers

```
81
82          Integer[] arr2 = {1,2,3,3,3,3,4,5};
83          ArrayList<Integer> arrayListofIntegers = a.createArrayListFromArray(arr2);
84          System.out.println(arrayListofIntegers);
85
86      }
87  }
88
```

ArrayList Examples

- Define a method *createArrayListFromArray* that takes an array of integers as a *parameter* and *returns* an ArrayList of integers

```
54    /**
55     * Creates an ArrayList from array of integers.
56     * @param arr of integers
57     * @return ArrayList of given integers
58     */
59    ArrayList<Integer> createArrayListFromArray(Integer[] arr) {
60
61        //create arraylist from a list of the given array of integers
62        //use Arrays.asList to convert array to list
63        //ArrayList<Integer> arrayListofIntegers = new ArrayList<Integer>(Arrays.asList(arr));
64
65        //you can also use List.of to convert array to list
66        ArrayList<Integer> arrayListofIntegers = new ArrayList<Integer>(List.of(arr));
67
68        return arrayListofIntegers;
69    }
70
```

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

```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

```
33      //constructor
34      /**
35       * Create a Classroom with given buildingName, roomName, number of rows, and number of columns.
36       * @param buildingName of building
37       * @param roomName of room
38       * @param rows for seats
39       * @param columns for seats
40       */
41      public Classroom(String buildingName, String roomName, int rows, int columns) {
42
43          //set building name
44          this.buildingName = buildingName;
45
46          //set room name
47          this.roomName = roomName;
48
49          //create 2-dimensional array of seats with given number of rows and columns
50          this.seats = new Seat[rows][columns];
51
52          //populate 2-dimensional array of seats with instances of Seat
53          //iterate over the rows
54          for (int i = 0; i < rows; i++) {
55
56              //iterate over the columns in each row
57              for (int j = 0; j < columns; j++) {
58
59                  //create a new instance of Seat in each slot in 2-dimensional array
60                  seats[i][j] = new Seat(i, j);
61              }
62          }
63
64          //create empty ArrayList for students
65          this.students = new ArrayList<Student>();
66
67      }
68
```

Classroom Project – Seat Class

Classroom.java | Seat.java

```
1
2  /**
3   * Represents a seat in a classroom.
4   * Each seat has a row and column associated with it.
5   * It also MIGHT have a student assigned to it.
6   * @author lbrandon
7   */
8  public class Seat {
9
10     //instance variables
11
12     /**
13      * Seat row.
14      */
15     int row;
16
17     /**
18      * Seat column.
19      */
20     int column;
21
22     /**
23      * Student assigned to this seat, otherwise null.
24      */
25     Student studentInSeat;
26
27     //constructor
28     /**
29      * Creates a seat for a class at given row and column.
30      * @param row for seat
31      * @param column for seat
32      */
33     public Seat(int row, int column) {
34         this.row = row;
35         this.column = column;
36     }
```

Classroom Project – Seat Class

```
37
38      /**
39       * Assigns the given student to this seat.
40       * @param student to assign
41       */
42      public void putStudentInSeat(Student student) {
43          this.studentInSeat = student;
44      }
45
46      //methods
47
48      /**
49       * Returns row, column, and student for this seat.
50       */
51      @Override
52      public String toString() {
53          return this.row + ", " + this.column + ": " + this.studentInSeat; //calls the toString method in student
54      }
55
```

Classroom Project – Student Class

Classroom.java | Seat.java | Student.java

```
1
2  /**
3   * Represents a student for a class.
4   * Each student has a name and ID.
5   * @author lbrandon
6   *
7   */
8  public class Student {
9
10     //instance variables
11
12     /**
13      * Name of student.
14      */
15     String name;
16
17     /**
18      * ID for student.
19      */
20     String ID;
21
22     //constructor
23
24     /**
25      * Creates a student with given name and ID.
26      * @param name for student
27      * @param ID for student
28      */
29     public Student(String name, String ID) {
30         this.name = name;
31         this.ID = ID;
32     }
33
```

Classroom Project – Student Class

```
36      /**
37       * Returns name of student.
38       */
39      @Override
40      public String toString() {
41          return this.name;
42      }
```

Classroom Project – Classroom Class

```
69      //methods
70      /**
71       * Adds the given student to the classroom.
72       * @param student to add
73       */
74      public void addAStudent(Student student) {
75          this.students.add(student);
76      }
77
78      /**
79       * Finds a seat and assigns to the given student.
80       * @param student to assign
81       */
82      public void assignStudentToSeat(Student student) {
83
84          int rows = this.seats.length; //gets number of rows
85          int columns = this.seats[0].length; //gets number of columns in first row
86
87          //iterate over rows and columns
88          for (int i = 0; i < rows; i++) {
89              for (int j = 0; j < columns; j++) {
90                  //find available seat, if student in that seat is null (empty)
91                  if (this.seats[i][j].studentInSeat == null) {
92                      //assign student
93                      this.seats[i][j].putStudentInSeat(student);
94                      //we're done, break out of loop (and method)
95                      return;
96                  }
97              }
98
99          }
100     }
101
```

Classroom Project – Classroom Class

```
101
102     /**
103      * Prints all the students in the class.
104      */
105     public void printAllStudents() {
106         System.out.println("Students in class: ");
107
108         //use enhanced for loop to print each student
109         for (Student student : this.students) {
110             System.out.println(student); //calls the toString in the Student class
111         }
112     }
113
114     /**
115      * Returns layout of classroom, with seat and student info.
116      */
117     @Override
118     public String toString() {
119         String s = "\n";
120
121         int rows = seats.length; //gets number of rows
122         int columns = seats[0].length; //gets number of columns in first row
123
124         for (int i = 0; i < rows; i++) {
125             for (int j = 0; j < columns; j++) {
126                 s += seats[i][j] + "\t"; //calls the toString method in the Seat class
127             }
128             s += "\n";
129         }
130
131         return s;
132     }
133
```

Classroom Project – Classroom Class

```
134
135     public static void main(String[] args) {
136
137         //create classroom
138         Classroom huntsman = new Classroom("HH", "105", 10, 5);
139
140         //create students
141         Student finegan = new Student("finegan", "fineganw");
142         Student bob = new Student("bob", "roberts");
143
144         //add students to class
145         huntsman.addAStudent(finegan);
146         huntsman.addAStudent(bob);
147
148         //assign students to seats
149         huntsman.assignStudentToSeat(finegan);
150         huntsman.assignStudentToSeat(bob);
151
152         //print list of students in class
153         huntsman.printAllStudents();
154
155         //print the classroom itself
156         //calls the toString method in classroom
157         System.out.println(huntsman);
158
159     }
160 }
```

Homework 6

Homework 6

Will be assigned by tonight, Wednesday, October 19th at midnight and due Monday, October 31st at midnight

- In this HW, you will **build a Movie Trivia program**
  - We will represent a movie database using classes and arraylists, with the goal of answering simple movie trivia questions
    - For example, “what is the name of the movie that both Tom Hanks and Leonardo DiCaprio acted in?”. Or, “what are the movie ratings for Rocky I?”
- This HW deals with the following topics:
  - More experience with class-based object-oriented programming
  - Arrays and ArrayLists
  - Test-Driven Development (unit testing)
  - Static methods (covered in the next lecture!)
- To complete the assignment:
  - Submit all the classes in your entire Java project. This will include everything in your “src” folder.