CSCI 161
Introduction to Computer Science

Department of Mathematics and Computer Science

Lecture 5
Arrays and ArrayList
Motivation: Bank

- Suppose we want to program a bank
  - Has a name
  - Stores the balances of multiple accounts

- Can:
  - Deposit some $$ into an account
  - Return the balance given an account number
  - Return the average account balance
  - Return the highest or lowest balance account balance
  - Prints the account numbers with a low balance (< $100)
Outline

- Motivation for Collections
- **Arrays: Declaration, Creation, and Initialization**
- Array Access
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
Declaring Arrays

- **Array declaration** creates a reference that points to an array object in the computer's memory
  - Syntax: `DataType[] variable_name;`
  - An array is an object, so it initially points to `null` until instantiation

**Examples**

```java
double[] acct_balance; // References an array of doubles
Circle[] my_circs; // References an array of Circle objects
String[] names; // References an array of Strings
```
Instantiating Arrays

- **Array instantiation** tells Java the number of elements you need to store.
  - Syntax: `variable_name = new DataType[size];`
    - `size` must be a positive integer.

**Examples**

```java
acct_balance = new double[20000];  // Holds 20000 doubles
names = new String[2600];          // Holds 2600 Strings
my_circs = new Circle[5];          // Holds 5 references to Circles
```
Array, Illustrated

- An **array** is an **object** that can store an **indexed list** of values
  - Indices (positions) are zero-based
  - Length of an array named `A` is stored in `A.length`
    - This value is a constant or a `final` value (read-only)

- For example, an array storing 10 people's ages

  ```java
  int[] age = new int[10]; // declare and instantiate an array of length 10
  ```
If you *already know* the contents of an array as you're coding:

```
DataType[] arrayName = { val0, val1, ..., val_n };  
```
If you *already know* the contents of an array as you're coding:

```java
DataType[] arrayName = { val0, val1, ..., val_n };  
```

```java
int[] faceValues = { 1,2,3,4,5,6,7,8,9,10,11,12,13 };  
```
Array Initialization Shortcut

- If you *already know* the contents of an array as you're coding:

```
DataType[] arrayName = { val0, val1, ..., val_n };
```

```
String[] names = { "Lilly", "Max", "Ben" };
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>&quot;Lilly&quot;</td>
<td>&quot;Max&quot;</td>
<td>&quot;Ben&quot;</td>
</tr>
</tbody>
</table>

length: 3
Outline

- Motivation for Arrays
- Declaration, Creation, and Initialization
- Array Access
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
Accessing Elements (Looking in the Rooms)

- Important: The *ith* array element is accessed by the syntax:

  ```java
  arrayName[i]
  ```

- Example:

  ```java
  int[] age = new int[10];
age[0] = 25;
age[1] = 43;
age[0]++;  
System.out.println(age[0]);  // output?
System.out.println(age[6]);  // output?
System.out.println(age[10]); // output?
  ```
Looping through Arrays

- Arrays and loops go hand-in-hand
  - Remember: `arrayName.length` holds the size of that array
    - Notice, length is not a method!!!
    - In fact, length is field of array objects declared using: `public final length;`
    - A `final` variable cannot be changed once it's given a value

```java
//Find the average balance
int sum = 0;
for (int i = 0; i < balance.length; i++) {
    sum += balance[i];
}
double average = (double) sum / balance.length;
```
Consider the following class:

- Write a constructor that:
  - Creates an array of the given size
  - Initializes all the elements in the array to 1

```java
public class Mystery {
    private int[] list;

    public Mystery(int size) {
        // TODO: See BlueJ demo
    }

    ...
}
```
Looping through Arrays

- Consider the following class:

  - Write a constructor that:
    - Creates an array of the given size
    - Initializes all the elements in the array to 1

```java
public class Mystery {
    private int[] list;
    public Mystery(int size) {
        if (size > 0) {
            list = new int[size];
            //initialize every element in the list to 1
            for (int i = 0; i < list.length; i++) {
                list[i] = 1;
            }
        }
    }
    ...
}
```
Looping through Arrays

- In the same Mystery class:
  - Now write a method that returns the average of the numbers in the array

```java
public class Mystery {
    private int[] list;

    public double getAvg() {
        // TODO
    }
}
```
Accessing Elements (Looking in the Rooms)

- Remember that arrays can also store objects

```java
Pig[] myPigs = new Pigs[2]; // an array capable of storing 2 Pig objects
```
Example of array storing objects:

```java
Pig[] myPigs = new Pigs[2];  // an array capable of storing 2 Pig objects
myPigs[0] = new Pig();       // create a Pig object and store it in 0th position
```
Accessing Elements (Looking in the Rooms)

- Example of array storing objects:

```java
Pig[] myPigs = new Pigs[2];  //an array capable of storing 2 Pig objects
myPigs[0] = new Pig();       //create a Pig object and store it in 0th position
myPigs[1] = new Pig();       //create a Pig object and store it in 1st position
```
Example of array storing objects:

```java
Pig[] myPigs = new Pigs[2]; // an array capable of storing 2 Pig objects
myPigs[0] = new Pig(); // create a Pig object and store it in 0th position
myPigs[1] = new Pig(); // create a Pig object and store it in 1st position

myPigs[0].sleep();
myPigs[1].eat(50);
System.out.println("There are " + myPigs.length + " pigs");
```

"Honk Shooo"
"Nom Nom Nom"
Another Example of Array Access

- Remember, arrays can hold any data type, including objects.
  - If we have a list of Pigs, we could, for instance put them all to sleep in one go:

```java
//Put every pig to sleep!
for (int i = 0; i < myPigs.length; i++) {
    //important: one of the elements might not have a Pig assigned to it yet!
    //check to see if it's still pointing to null
    if (myPigs[i] != null) {
        myPigs[i].sleep();
    }
}
```
Example: Multiple Turtles

- Create a TurtleDrawer class with N turtles, and the following API:
  - Final code on course page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public TurtleDrawer()</td>
<td>Creates 3 turtles</td>
</tr>
<tr>
<td>public TurtleDrawer(int N)</td>
<td>Creates N turtles</td>
</tr>
<tr>
<td>public void allUp()</td>
<td>Lift all pens</td>
</tr>
<tr>
<td>public void allDown()</td>
<td>Lower all pens</td>
</tr>
<tr>
<td>public void allForward(int dist)</td>
<td>Move all turtles forward by given distance</td>
</tr>
<tr>
<td>public void randomTurn()</td>
<td>Randomly turn all turtles' directions</td>
</tr>
<tr>
<td>public void penUp(int i)</td>
<td>Lift pen for ith turtle</td>
</tr>
<tr>
<td>public void penDown(int i)</td>
<td>Lower pen for ith turtle</td>
</tr>
<tr>
<td>public void goofyDrawing()</td>
<td>Repeatedly turn all and draw random amounts</td>
</tr>
</tbody>
</table>
Example: goofyDrawing()
Outline

- Motivation for Collections
- Arrays: Declaration, Creation, and Initialization
- Array Access
- Arrays as Objects
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
Arrays are objects, so an array variable is just a reference.

```java
//declare an reference called names
String[] names;
```

```java
//just like objects, 'new' sets aside storage, and actually creates the array
names = new String[10];
```

Each element is an uninitialized String... *(What does each of these array elements currently store?)*
Array Variables Store Object References!

- We can change which array that the `names` variable refers to:

```java
String[] names = new String[10];
```
Array Variables Store Object References!

- We can change which array that the names variable refers to:

```java
String[] names = new String[10];
names = new String[1000];
String[] roster = names;
```

No one is referencing me anymore! :(
Gone forever byeeeeee!
(garbage collection)
Outline

- Motivation for Arrays
- Declaration, Creation, and Initialization
- Array Access
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
How Do Maps Work?

- Chicago
- New York City (NYC)
- Atlanta
- Boston

Distances:
- Chicago to NYC: 787.8
- Chicago to Atlanta: 714.1
- Chicago to Boston: 983.5
- NYC to Atlanta: 888.2
- NYC to Boston: 214
- Atlanta to Boston: 1102
Sometimes, a table is a more convenient way to represent certain information.

<table>
<thead>
<tr>
<th></th>
<th>Chicago (0)</th>
<th>Boston (1)</th>
<th>NYC (2)</th>
<th>Atlanta (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago (0)</td>
<td>0</td>
<td>983.5</td>
<td>787.8</td>
<td>714.1</td>
</tr>
<tr>
<td>Boston (1)</td>
<td>983.5</td>
<td>0</td>
<td>214</td>
<td>0.0</td>
</tr>
<tr>
<td>NYC (2)</td>
<td>787.8</td>
<td>214.0</td>
<td>0</td>
<td>888.2</td>
</tr>
<tr>
<td>Atlanta (3)</td>
<td>714.1</td>
<td>1102</td>
<td>888.2</td>
<td>0</td>
</tr>
</tbody>
</table>
Two Dimensional (2D) Arrays

- Other examples:
  - Store 50 golfers' scores
  - My favorite stocks' prices over an 8-hour day
  - Store a list of temperatures for multiple days
  - Store the pixel values of a canvas

- A 2D array \( (i.e., \text{matrix}) \) has both row and column indices

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Making a 2D Array

- Declaration: `dataType[][] arrayName;`

- Instantiation: `arrayName = new dataType[numRows][numColumns];`

- Example:

```java
//make a table of scores (18 holes) for 4 golfers
int[][] scores;
scores = new int[4][18];

//make 3 class rosters, with room for 25 names
String[][] classRoster;
classRoster = new String[3][25];
```
Recall that array elements are *initialized* to their data type's default values, or `null` if storing objects.

- Example:

```java
//make 3 class rosters, with room for 25 names
String[][] classRoster;
classRoster = new String[3][25];
```
2D Array Initialization

- If elements' values are known in advance, you can specify them during declaration:

```java
String[][] reportCard = {
    {"A", "B", "A"},  // 1st semester
    {"C-", "C", "B+"}, // 2nd semester
};
```

![Diagram of 2D array initialization](image)
Accessing 2D Arrays

- Recall to access a 1D array element, we use: `arrayName[index]`
- In contrast, each 2D element has two indices: `arrayName[rowIndex][colIndex]`

Example:

```java
int[][] table = new int[3][4];
table[0][0] = 90;
table[0][1] = 43;
table[0][2] = 10;
table[0][3] = 21;
table[1][1] = 20;
table[3][3] = 100; // runtime error
int Y = table[0][2] + table[1][1];
System.out.println(Y);
> 30
```
Getting 2D Dimensions

- We know that every 1D array has a `.length` field that stores its size.
- Given a 2D array, how do we get the following?
  - How many rows does table have?
  - How many columns?
- Important to remember that a 2D array is an array of arrays.
- Knowing this: print out all elements in a 2D array.
Back to Our Example

- Assign each city some **ID** starting from 0

<table>
<thead>
<tr>
<th>Distance</th>
<th>Chicago (0)</th>
<th>Boston (1)</th>
<th>NYC (2)</th>
<th>Atlanta (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago (0)</td>
<td>0</td>
<td>983.5</td>
<td>787.8</td>
<td>714.1</td>
</tr>
<tr>
<td>Boston (1)</td>
<td>983.5</td>
<td>0</td>
<td>214</td>
<td>1102</td>
</tr>
<tr>
<td>NYC (2)</td>
<td>787.8</td>
<td>214.0</td>
<td>0</td>
<td>888.2</td>
</tr>
<tr>
<td>Atlanta (3)</td>
<td>714.1</td>
<td>1102</td>
<td>888.2</td>
<td>0</td>
</tr>
</tbody>
</table>

```java
private double[][] distances = {
    {0, 983, 787, 714},
    {983, 0, 214, 1102},
    {787, 214, 0, 888},
    {714, 1102, 888, 0}
};
```

- Open up BlueJ...
  
  - `public double getDistance(int cityID1, int cityID2)` that returns the distance between any two cities.
  
  - `public int getNearestCity(int cityID)` that returns the city nearest to the city with the given ID.
  
  - `public double getPathDistance(int[] paths)` that returns the distance of the given path.
### On Your Own: Make a Multiplication Table

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
<td>33</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
<td>40</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
<td>66</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
<td>70</td>
<td>77</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
<td>88</td>
<td>96</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td>90</td>
<td>99</td>
<td>108</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>88</td>
<td>99</td>
<td>110</td>
<td>121</td>
<td>132</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
<td>72</td>
<td>84</td>
<td>96</td>
<td>108</td>
<td>120</td>
<td>132</td>
<td>144</td>
</tr>
</tbody>
</table>
Make a Multiplication Table

- How should this class behave? (Code pad)

```java
MultTable myTable = new MultTable(5,5);  //creates a 5 x 5 multiplication table
myTable.printTable();
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25

MultTable myTable = new MultTable(4,8);  //creates a 4 x 8 multiplication table
myTable.printTable();
1 2 3 4 5 6 7 8
2 4 6 8 10 12 14 16
3 6 9 12 15 18 21 24
4 8 12 16 20 24 28 32
```
Higher Dimensionality Arrays

- Most applications can get by using 1D and 2D arrays, but some might call for even higher dimensionality.
  - Anything higher than 3D is extremely rare

- Be mindful of space... this is a billion doubles (8 GB)!!!
Outline

- Motivation for Arrays
- Declaration, Creation, and Initialization
- Array Access
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
Motivation for ArrayLists

- Problem: Arrays cannot be (easily) resized
  - But in many problems, size is unknown or can grow/shrink during execution

- Motivating example:
  - Want an "army" of Robots
    - Manage multiple Robots
    - Make them all march forward, backward, etc.
  - New Robots can be enlisted (added) over time
  - Robots can be discharged (removed) over time
Using ArrayLists

- To use this class, you must first import it at the top of your file:

```java
import java.util.ArrayList;
```

- ArrayList's declaration and instantiation syntax looks wonky

```java
ArrayList<E> list_name;
list_name = new ArrayList<>();
```

- E is the class name for the type of elements we will store

- To create an ArrayList of Integers, we use:

```java
ArrayList<Integer> peoplesAge = new ArrayList<>();
```

- To create an ArrayList of Dice

```java
ArrayList<Die> diceCollection = new ArrayList<>();
```

Note:

Use ArrayList<Double> to store doubles
Use ArrayList<Boolean> to store booleans
Use ArrayList<Integer> to store ints
Useful ArrayList Methods

Some key methods from the **ArrayList API**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public ArrayList&lt;&gt;()</td>
<td>Constructor. Creates a new ArrayList that holds objects of type E</td>
</tr>
<tr>
<td>public boolean add(E e)</td>
<td>Inserts element e to end of this list</td>
</tr>
<tr>
<td>public boolean add(int index, E e)</td>
<td>Inserts element e at the specified position in the list</td>
</tr>
<tr>
<td>public void clear()</td>
<td>Removes all elements from list</td>
</tr>
<tr>
<td>public boolean contains(E e)</td>
<td>Searches for element e in list, returns true if found, false otherwise</td>
</tr>
<tr>
<td>public E get(int index)</td>
<td>Returns the element at given index</td>
</tr>
<tr>
<td>public int indexOf(E e)</td>
<td>Searches for e in the list, returns the index of the first occurrence if found, or -1 if not found.</td>
</tr>
<tr>
<td>public E remove(int index)</td>
<td>Removes the element at given index. Returns the deleted element</td>
</tr>
<tr>
<td>public E set(int index, E e)</td>
<td>Replaces the element at the specified index.</td>
</tr>
<tr>
<td>public int size()</td>
<td>Returns the number elements in the current list</td>
</tr>
<tr>
<td>public String toString()</td>
<td>Returns the String representation of the current list</td>
</tr>
</tbody>
</table>

For the full ArrayList API: [http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)
ArrayList Internals

ArrayList<String> dwarves = new ArrayList<>();

<table>
<thead>
<tr>
<th>null</th>
<th>null</th>
<th>null</th>
<th>...</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>...</td>
<td>9</td>
</tr>
</tbody>
</table>

size: 0
capacity: 10
ArrayList Internals

ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
ArrayList Internals

```java
ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
dwarves.add(1, "Awful");
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;Bashful&quot;</td>
<td>&quot;Awful&quot;</td>
<td>&quot;Jumpy&quot;</td>
<td>...</td>
<td>null</td>
</tr>
</tbody>
</table>
```

size: 3
capacity: 10
ArrayList Internals

```java
ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
dwarves.add(1, "Awful");
dwarves.remove(0);
```
ArrayList Internals

```java
ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
dwarves.add(1, "Awful");
dwarves.remove(0);
dwarves.set(0, "Doc");
```
ArrayList Internals

ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
dwarves.add(1, "Awful");
dwarves.remove(0);
dwarves.set(0, "Doc");
dwarves.add("Snorty");
dwarves.add("Sneezy");
dwarves.add("Coughy");
dwarves.add("Snoozy");
dwarves.add("Grumpy");
dwarves.add("Dopey");
dwarves.add("Happy");
dwarves.add("Frumpy");
ArrayList Internals

ArrayList<String> dwarves = new ArrayList<>();
dwarves.add("Bashful");
dwarves.add("Jumpy");
dwarves.add(1, "Awful");
dwarves.remove(0);
dwarves.set(0, "Doc");
dwarves.add("Snorty");
dwarves.add("Sneezy");
dwarves.add("Coughy");
dwarves.add("Snoozy");
dwarves.add("Grumpy");
dwarves.add("Dopey");
dwarves.add("Happy");
dwarves.add("Frumpy");
dwarves.add("Jittery");
Example Usage of ArrayList of Doubles

```java
ArrayList<Double> gpas;
gpas = new ArrayList<>();

System.out.println(gpas.toString()); // calling gpas.toString()
> []
```
Example Usage of ArrayList of Doubles

```
ArrayList<Double> gpas;
gpas = new ArrayList<>();

System.out.println(gpas.toString());
> []

gpas.add(4.0);
gpas.add(2.5);
gpas.add(3.0);
System.out.println(gpas.toString());
> [4.0, 2.5, 3.0]
```
Example Usage of ArrayList of Doubles

```java
ArrayList<Double> gpas;
gpas = new ArrayList<>();

System.out.println(gpas.toString()); //calling gpas.toString()
> []

gpas.add(4.0);
gpas.add(2.5);
gpas.add(3.0);
System.out.println(gpas.toString());
> [4.0, 2.5, 3.0]

gpa.add(0, 3.8); //use overloaded add() method
System.out.println(gpas.toString());
> [3.8, 4.0, 2.5, 3.0]

System.out.println(gpa.get(2));
> 2.5

System.out.println(gpa.contains(0.0));
> false
```
Example Usage of ArrayList of Dice

```java
ArrayList<Die> listOfDice;
System.out.println(listOfDice);
> null
```
Example Usage of ArrayList

```java
ArrayList<Die> listOfDice;
System.out.println(listOfDice);
> null

listOfDice = new ArrayList<>(); // instantiate an empty list of dice
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.size()
> 3 (int)
```
Example Usage of ArrayList

```java
ArrayList<Die> listOfDice;
System.out.println(listOfDice);
> null

listOfDice = new ArrayList<>(); // instantiate an empty list of dice
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.size()
> 3 (int)

for (int i = 0; i < listOfDice.size(); i++) {
    // roll each die and print its face value
    listOfDice.get(i).roll();
    System.out.println(listOfDice.get(i).getFaceValue());
}
> 3
> 5
> 2
```
Example Usage of ArrayList

```java
ArrayList<Die> listOfDice;
System.out.println(listOfDice);
> null

listOfDice = new ArrayList<>(); // instantiate an empty list of dice
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.add(new Die());
listOfDice.size() > 3

for (int i = 0; i < listOfDice.size(); i++) {
  // roll each die and print its face value
  listOfDice.get(i).roll();
  System.out.println(listOfDice.get(i).getFaceValue());
}
> 3
> 5
> 2

listOfDice.remove(listOfDice.size() - 1);
for (int i = 0; i < listOfDice.size(); i++) {
  // print out each die's face value
  System.out.println(listOfDice.get(i).getFaceValue());
}
> 3
> 5
```
Implement the following: (Code on web page)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public RobotArmy()</code></td>
<td>Constructor. Creates a new (empty) collection of Robots</td>
</tr>
<tr>
<td><code>public boolean addRobot()</code></td>
<td>Creates a new robot and adds it to the army</td>
</tr>
<tr>
<td><code>public boolean addRobot(Robot r)</code></td>
<td>Adds the specified robot to the army</td>
</tr>
<tr>
<td><code>public Robot discharge(int index)</code></td>
<td>Removes and returns the robot at the specified index.</td>
</tr>
<tr>
<td><code>public void dischargeAll()</code></td>
<td>Removes all robots</td>
</tr>
<tr>
<td><code>public void showAll()</code></td>
<td>Makes all robots visible</td>
</tr>
<tr>
<td><code>public void hideAll()</code></td>
<td>Makes all robots invisible</td>
</tr>
<tr>
<td><code>public int getSize()</code></td>
<td>Returns the size of the army</td>
</tr>
<tr>
<td><code>public void scatterAll()</code></td>
<td>Moves all robots to a random location</td>
</tr>
<tr>
<td><code>public void march(int steps)</code></td>
<td>Moves all robots forward/backward concurrently</td>
</tr>
</tbody>
</table>
Arrays are useful for storing collections of related data, which happens often in computer applications

- 1D array: list
- 2D array: a list of lists
- 3D array: a list of list of lists
- and so on...

Every dimension can be indexed:

- Indices always start from 0
Conclusion (Cont.)

- Things to remember for 1D array A:
  - Indices start from 0 and end at $A.length-1$
  - $A.length$ constant always holds the size of the array

- 2D arrays are row-major, so $A[i][j]$ is the element at $ith$ row, $jth$ column
  - $A.length$ constant holds the size of rows
  - $A[i].length$ constant holds size of columns in row $i$
Conclusion (Cont.)

- Summary of ArrayLists
  - Unlike arrays, they can be resized
  - Declaration uses `diamond` notation to specify what objects it will store
  - Each element holds an object reference
    - Wait, then how do you store primitives? (Find out in Lab 8!)
When to Use Arrays vs. ArrayLists?

- Arrays are fast, but cannot be re-sized
- ArrayLists are slow in comparison, but are flexible
  - Indeed, the ArrayList class is implemented using arrays!

Rules of thumb:

- Use an array if:
  - Your collection will not change in size
  - Speed matters
- Otherwise, use an ArrayList
Administrivia 3/11

- Hwk 3: OrcaCard graded
- Hwk 5 due Friday!
- Talk tonight!
  - "Survey Data Science"
  - Prof. Kelly McConville
  - Assistant Professor of Statistics, Reed College
  - 4pm, TH 391
Hwk 5 due Friday!

- Questions?

Last time…

- Introduction to arrays
- Started the Bank class
- A quick and dirty review on arrays
Administrivia 3/15

- Reminder: Hwk 5 due tonight
- Solutions posted for:
  - Hwk 4, Lab 6, & Lab7

- Last time...
  - More array practice
  - A taste of complexity

- Today...
  - Classic array-based algorithms: Find largest, Merge
  - 2D arrays
Adminstrivia 3/25

- To be posted later today
  - Hwk 5 solutions
  - Hwk 6 to be posted

- Today:
  - 2D Arrays