CSCI 161
Introduction to Computer Science
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
Motivation for Arrays

- Need a way to store and access large numbers of values
Thinking about Arrays (Hotel Analogy)

A Hotel has four rooms...

• The number of rooms cannot change once the hotel is built
• Each room has a room number, which serves as an *address* or *index*
• Behind each door stores an occupant, or it can be empty
  - Contents (what's behind the door?) can change over time.
Declaring Arrays with the [] Notation

- **Array Declaration**
  - Syntax: `DataType[] variable_name;
  - An array is an object, so it points to null until instantiation
  - Like before, DataType is a primitive or a Class

- **Declaring fields as arrays**
  ```java
  private double[] acct_balance;  // Field that is an array of doubles
  ```

- **Declaring local variable as arrays**
  ```java
  Circle[] my_circs;  // Local variable that is an array of Circle objects
  String[] names;    // Local variable that is an array of Strings
  ```
Instantiating Arrays

- **Array instantiation** tells Java the number of elements you need to store

  - Syntax: `arrayName = new DataType[length];`

**Examples**

```java
this.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)
Array, Illustrated

- An **array** is an **object** that can store an **indexed list** of values
  - Indices (positions) are zero-based
  - Length of an array you named *Arr* is found in *Arr.length*
    - This value is a constant or a **final** value (can't be assigned new values)

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

```java
int[] age = new int[10]; // declare and instantiate an array of length 10
```

![Diagram of an array storing 10 people's ages with indices and values]
Array Access

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

```java
int[] age = new int[10];
age[0] = 25;
```
Array Access

For example, an array storing 10 people's ages

```java
int[] age = new int[10];
age[0] = 25;
age[1] = 43;
```

![Array diagram showing ages](image)
For example, an array storing 10 people's ages

```java
int[] age = new int[10];
age[0] = 25;
age[1] = 43;
age[0] += 10;
```
Array Access

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

```java
int[] age = new int[10];
age[0] = 25;
age[1] = 43;
age[0] += 10;
System.out.println(age[0]); // output?
System.out.println(age[6]); // output?
System.out.println(age[10]); // output?
```
Let's Start Writing the Bank Class

- Suppose we want to program a bank
  - Stores the $ balances of N accounts (N is given by user)
  - Account ID go from 0 to N-1

- Can:
  - public void deposit(int acctID, double amount)
  - public double getBalance(int acctID)
  - public double getAvgBalance()
  - public void lowBalance()
    - Prints the account IDs with low balance (< $100)
**Looping through Arrays**

- **Important**: 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

- **Example**:

```java
// Print out everyone's age!
for (int i = 0; i < age.length; i++) {
    System.out.println(age[i]);
}
```
Let's Start Writing the Bank Class

- Suppose we want to program a bank
  - Stores the $ balances of \( N \) accounts (\( N \) is given by user)
  - Account ID go from 0 to \( N-1 \)

- Can:
  - public void deposit(int acctID, double amount)
  - public double getBalance(int acctID)
  - public double getAvgBalance()
  - public void lowBalance()
    - Prints the account IDs with low balance ($ < 100)
Outline

- Motivation for Collections
- Arrays: Declaration, Creation, and Initialization
  - Storing Objects in Arrays
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
Arrays can also store objects, not just primitives!

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

Create a TurtleMob class with \textbf{N} turtles, and the following API:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public TurtleMob(int \textit{N})</td>
<td>Creates an array with \textit{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 \textit{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 goExplore()</td>
<td>Repeatedly turn all and draw random amounts</td>
</tr>
<tr>
<td>public Turtle[] subset(int \textit{start}, int \textit{end})</td>
<td>Returns an array of Turtles in the specified range</td>
</tr>
<tr>
<td>public void addMob(TurtleMob \textit{other})</td>
<td>Adds the Turtles from another mob to this collection</td>
</tr>
</tbody>
</table>
Recall How to Control a Single Turtle

Here's Turtle's API again

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Turtle()</td>
<td>Constructor to create a Turtle object</td>
</tr>
<tr>
<td>public void forward(distance)</td>
<td>Move turtle forward by specified distance</td>
</tr>
<tr>
<td>public void penDown()</td>
<td>Lower the pen to draw</td>
</tr>
<tr>
<td>public void penUp()</td>
<td>Lift pen up</td>
</tr>
<tr>
<td>public void left(angle)</td>
<td>Turn turtle left by given angle</td>
</tr>
<tr>
<td>public void right(angle)</td>
<td>Turn turtle right by given angle</td>
</tr>
<tr>
<td>public void setPenColor(color)</td>
<td>Sets the color of the turtle's pen.</td>
</tr>
</tbody>
</table>
Example: goExplore()
Outline

- Motivation for Arrays
- Declaration, Creation, and Initialization
- Array Access
- Multi-Dimensional Arrays
- The ArrayList Class
- Conclusion
How Do Maps Work?
Sometimes, a table is a more way convenient to represent certain information

<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>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., matrix) has both row and column indices

<table>
<thead>
<tr>
<th></th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>

Floors are "rows"
Room in each floor are "columns"
Making a 2D Array

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

- **Instantiation:**
  
  ```java
  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];
  ```
Default Values within a 2D Array

- 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];
    ```
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
};
```
Accessing 2D Arrays

- Recall to access a 1D array element, we use: \texttt{arrayName[index]}
- In contrast, each 2D element has two indices: \texttt{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>

- 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 are great, but cannot be (easily) resized
  - But in many problems, size is unknown or can grow/shrink during execution

- **Motivating example:**
  - Want a *resizable* mob of Turtles
  - New Turtles can be added over time
  - Turtles can also be removed from the mob
Creating ArrayLists

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

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

- Syntax (**E** is the *class of objects* to be stored in the list):

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

  - To create an ArrayList of Strings, we use:

```java
ArrayList<String> roster = new ArrayList<>();
```

  - To create an ArrayList of Turtles

```java
ArrayList<Turtle> mob = new ArrayList<>();
```
## Useful ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean add(E e)</td>
<td>Inserts item e to the end of this list</td>
</tr>
<tr>
<td>public boolean add(int index, E e)</td>
<td>Inserts item e at the specified position in the list</td>
</tr>
<tr>
<td>public void clear()</td>
<td>Removes all items</td>
</tr>
<tr>
<td>public E get(int index)</td>
<td>Returns the item at the 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 item at given index. Returns the deleted item</td>
</tr>
<tr>
<td>public E set(int index, E e)</td>
<td>Replaces the item at the specified index</td>
</tr>
<tr>
<td>public int size()</td>
<td>Returns the number of items</td>
</tr>
<tr>
<td>public String toString()</td>
<td>Returns the String representation of the list</td>
</tr>
</tbody>
</table>
ArrayList Internals

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

![Diagram of ArrayList with null values and size/capacity information]
ArrayList Internals

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

```
| 0 | 1   | 2   | 3   | 4   | ...
|---|-----|-----|-----|-----|-----
| "Bashful" | "Jumpy" | null | null | null |
```

size: 2
capacity: 10
ArrayList Internals

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

![ArrayList structure diagram]

- `dwarves` is an instance of `ArrayList<String>`
- `size`: 3
- `capacity`: 10
- Elements: "Bashful", "Awful", "Jumpy"
ArrayList Internals

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");

```
  0  1  2  ...  9
"Doc" "Jumpy" "Snorty" ...
```

size: 10
capacity: 10
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");

dwarves

<table>
<thead>
<tr>
<th></th>
<th>&quot;Doc&quot;</th>
<th>&quot;Jumpy&quot;</th>
<th>&quot;Snorty&quot;</th>
<th>...</th>
<th>&quot;Frumpy&quot;</th>
<th>&quot;Jittery&quot;</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

size: 11
capacity: 20
Problem: ArrayLists can only store objects (specified as E)

- What about primitives?
- This won't compile!

```java
ArrayList<int> list = new ArrayList<>();
```
Wrapper Classes (2)

- **Problem:** ArrayLists can only store objects (specified as E)
  - What about primitives?
  - This won't compile! `ArrayList<int> list = new ArrayList<>();`

- **Important:** As a workaround, Java created *wrapper classes* for each primitive data type:
  - `Integer` (corresponds to `int`)
  - `Double` (corresponds to `double`)
  - `Boolean` (corresponds to `boolean`)
  - `Character` (corresponds to `char`)
  - ...
Problem: ArrayLists can only store objects (specified as E)
- What about primitives?
- This won't compile!

To store ints in an ArrayList, use its wrapper class instead:
AutoBoxing and Wrapper Classes

- **Weirdness:** This ArrayList stores `Integer` objects
  - So the `add()` method accepts `Integers`, not `ints`!
  - So why does the following work?

```java
ArrayList<Integer> list = new ArrayList<>();
list.add(5);  // This now works! ... but why should it?
             // 5 is an int, not Integer
```

- **AutoBoxing** - Java is **actually** doing this hidden background work:

```java
ArrayList<Integer> list = new ArrayList<>();
list.add(new Integer(5));
```
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

```java
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

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: Diary

- Write a class Diary that can store a collection of entries (Strings).
  - It has the following API

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void addEntry(String entry)</code></td>
<td>Adds a new entry to the diary</td>
</tr>
<tr>
<td><code>public String getEntry(int index)</code></td>
<td>Returns the entry at the specified index, or null if index is invalid</td>
</tr>
<tr>
<td><code>public void printAll()</code></td>
<td>Prints all diary entries</td>
</tr>
<tr>
<td><code>public ArrayList&lt;String&gt; search(String str)</code></td>
<td>Returns a list of entries containing the search string</td>
</tr>
<tr>
<td><code>public int countEntriesContaining(String str)</code></td>
<td>Returns the number of entries containing the search string</td>
</tr>
<tr>
<td><code>public void removeEntriesContaining(String str)</code></td>
<td>Deletes all entries containing the search string</td>
</tr>
</tbody>
</table>
Another Example: RobotGroup

- Motivating example:
  - Want a collection of Robots
    - Manage multiple Robots
    - Make them all march forward, backward, etc.
  - Robots can be added or removed over time
  - Robots can also be removed
Implement the following: (Code on web page)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean addRobot()</td>
<td>Creates a new robot and adds it to the group</td>
</tr>
<tr>
<td>public Robot discharge(int index)</td>
<td>Removes and returns the robot at the specified index.</td>
</tr>
<tr>
<td>public void dischargeAll()</td>
<td>Removes all robots</td>
</tr>
<tr>
<td>public void showAll()</td>
<td>Makes all robots visible</td>
</tr>
<tr>
<td>public void hideAll()</td>
<td>Makes all robots invisible</td>
</tr>
<tr>
<td>public int getSize()</td>
<td>Returns the size of the group</td>
</tr>
<tr>
<td>public void scatterAll()</td>
<td>Moves all robots to a random location</td>
</tr>
<tr>
<td>public void march(int steps)</td>
<td>Moves all robots forward/backward concurrently</td>
</tr>
</tbody>
</table>
Conclusion

- **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
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
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
Reminders:

- Hwk 5 (Loops) due Friday!
  - Prime number checker found in Lec 4 notes

Last time...

- Loops are one of the most important structures
  - Get strong, if you're not strong in loops!

Today:

- Finding our way around Zoom
  - How to mute/unmute, how to ask questions

- Arrays
  - Notes already posted!!!!!!
Changes to Zoom logistics: chat, recording, 'whiteboard'

Reminders:

- Hwk 5 due Friday!
  - Questions so far?
- Lab tomorrow:
  - Get on Zoom at 12; I'll try using breakout groups on Zoom

Last time...

- Started the Bank class

Today:

- Hwk 4 Robots Solution posted
- A review on arrays
Administrivia 3/13

- Update on recording: You can't, only I can.
  - I guess I will continue doing this?

- Reminders:
  - Hwk 5 extended to Monday 3/23
  - Next Meeting is Wednesday 3/25

- Last time...
  - A review on arrays
  - Started the TurtleMob class

- Today:
  - Finish TurtleMob
Welcome back!

- Let's discuss the big news: Credit/No-Credit option for all students
  - Look for the Google form: April 6-17 (After Exam 2)

- What about our Exams?
  - Just like Exam 1; I'll give you 2hrs, should only need 1
  - Remotely through gradescope.com
  - Open-book, open-notes, open-internet, access to BlueJ
  - Extended time for students with accommodations

- Lab tomorrow(!): More work on loops and arrays
Lab 8 (ComboGuesser) Post-mortem

- Weaknesses with arrays/loops still

Homework 5 solution posted

Reminder: Hwk 6 due 4/3

Last time...

- Weighed pros/cons of Arrays
- Introduced ArrayLists, "diamond notation"

Today

- More on ArrayLists
- RobotGroup class