YEAH
Session #5

November 7, 2017 7-8PM
Isabella Garcia-Camargo
ArrayLists

• Why ArrayLists?
  – You may not know how much data you have
• ArrayLists can grow as big as you need
• Can check if something is in it with .contains()
• Can only store objects (no primitives!)
  – Need “Integer” instead of “int” for storing integers, etc.

```
ArrayList<Type> list = new ArrayList<Type>();
```
# ArrayList

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(&lt;T&gt; element)</td>
<td>Adds a new element to the end of the ArrayList; the return value is always true.</td>
</tr>
<tr>
<td>void add(int index, &lt;T&gt; element)</td>
<td>Inserts a new element into the ArrayList before the position specified by index.</td>
</tr>
<tr>
<td>&lt;T&gt; remove(int index)</td>
<td>Removes the element at the specified position and returns that value.</td>
</tr>
<tr>
<td>boolean remove(&lt;T&gt; element)</td>
<td>Removes the first instance of element, if it appears; returns true if a match is found.</td>
</tr>
<tr>
<td>void clear()</td>
<td>Removes all elements from the ArrayList.</td>
</tr>
<tr>
<td>int size()</td>
<td>Returns the number of elements in the ArrayList.</td>
</tr>
<tr>
<td>&lt;T&gt; get(int index)</td>
<td>Returns the object at the specified index.</td>
</tr>
<tr>
<td>&lt;T&gt; set(int index, &lt;T&gt; value)</td>
<td>Sets the element at the specified index to the new value and returns the old value.</td>
</tr>
<tr>
<td>int indexOf(&lt;T&gt; value)</td>
<td>Returns the index of the first occurrence of the specified value, or -1 if it does not appear.</td>
</tr>
<tr>
<td>boolean contains(&lt;T&gt; value)</td>
<td>Returns true if the ArrayList contains the specified value.</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>Returns true if the ArrayList contains no elements.</td>
</tr>
</tbody>
</table>
Arrays

• Why Arrays?
  – Arrays are great for representing a fixed-sized list
• Store data at difference indices in the array, and look up data by index
• Can store any type of data (objects & primitives)
Arrays

- An array stores a sequence of multiple objects.
  - Can access objects by index using [ ].
- All stored objects have the same type.
  - You get to choose the type!
- Can store any type, even primitive types.
- Size is fixed; cannot grow once created.
Basic Array Operations

- To create a new array, specify the type of the array and the size in the call to `new`:
  \[
  \text{Type[]} \ arr = \text{new Type[size]}
  \]

- To access an element of the array, use the square brackets to choose the index:
  \[
  arr[index]
  \]

- To read the length of an array, you can read the `length` field:
  \[
  arr.length
  \]
2D Arrays (Grids)
Type[][] a = new Type[rows][cols];
Interpreting Multidimensional Arrays

- There are two main ways of intuiting a multidimensional array.
  - **As a 2D Grid:**
    - Looking up `arr[row][col]` selects the element in the array at position `(row, col)`.
  - **As an array of arrays:**
    - Looking up `arr[row]` gives back a one-dimensional consisting of the columns in row `row`. 
```java
int[][] a = new int[4][5];
```

<table>
<thead>
<tr>
<th>Call</th>
<th>Type?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[1]</td>
<td><code>int[5]</code> – an array of integers that can hold 5 values</td>
</tr>
<tr>
<td>a[2][3]</td>
<td><code>int</code> – a normal integer value</td>
</tr>
</tbody>
</table>

Can iterate through this!!!
2D ARRAYS

int arr[4][5] = new int[4][5]

arr[2][3] = 5

arr[1]
Iterating through a 2-D array

```java
Type[][] arr = /* ... */
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[row].length; col++) {
        /* ... access arr[row][col] ... */
    }
}
```
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
    for (int col = 0; col < arr[row].length; col++) {
        arr[row][col] = row + col;
    }
}
Yahtzee!

Due at 1:30PM on Wednesday, Nov. 15

– Graphics already implemented for you!
– Practice with arrays
– YahtzeeDemo – working demo in assignment folder (double-click to play)
– YahtzeeMagicStub.checkCategory() – provided for you for testing (eventually need to check category yourself!)
DEMO!
Game Flow

1. Roll Dice 1\textsuperscript{st} time
2. Select a set of dice to reroll (if any) and reroll
3. Repeat step 2
4. Choose a category to score in \textit{(make sure they haven’t already used it!)}

Use YahtzeeDisplay!
YahtzeeDisplay

- Graphics are taken care of for you
- Manipulate onscreen graphics via your YahtzeeDisplay instance variable (display.__)
- Methods on YahtzeeDisplay (from handout):
  - waitForPlayerToClickToRoll()
  - displayDice()
  - waitForPlayerToSelectDice()
  - isDieSelected()
  - waitForPlayerToSelectCategory()
  - updateScorecard()
  - printMessage()
- Player indices start at 1!!
import acm.io.*;

public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {

    public static void main(String[] args) {
        new Yahtzee().start(args);
    }

    public void run() {
        IODialog dialog = getDialog();
        nPlayers = dialog.readInt("Enter number of players");
        playerNames = new String[nPlayers];
        for (int i = 1; i <= nPlayers; i++) {
            playerNames[i - 1] = dialog.readLine("Enter name for player " + i);
        }
        display = new YahtzeeDisplay(getGCanvas(), playerNames);
        playGame();
    }

    private void playGame() {
        /* You fill this in */
    }

    /* Private instance variables */
    private int nPlayers;
    private String[] playerNames;
    private YahtzeeDisplay display;
    private RandomGenerator rgen = new RandomGenerator();
}

/* File: Yahtzee.java */
Because Yahtzee implements YahtzeeConstants, you can use these constants directly in your Yahtzee.java code.
Calculating Scores

• Given a set of dice, calculate the score for the chosen category
• 1s, 2s, 3s, full house, small straight...
• Use YahtzeeMagicStub initially/for testing, but don’t use it for your final submission!

```java
boolean matches = YahtzeeMagicStub.checkCategory(dice, YAHTZEE);
```
Calculating Scores

- Any roll is valid for 1s, 2s, 3s, 4s, 5s, 6s, and chance
- 3 Of a Kind, 4 Of a Kind, Yahtzee, Full House, Straights -> not all rolls valid (score = 0 if roll doesn’t fit category!)
- Update total score each time!
- When checking if roll fits category, think about dice value frequencies (e.g. what is 3 of a kind with respect to dice value frequencies?)
Game End

• Tally up Upper Bonus, Upper Score, Lower Score, Final Total

• Report winner!
Arrays Galore!

• Dice (N_DICE)
• Players (array of player names given to you in starter code as instance variable)
• Scorecard for all players (2d array representing scorecard)
Testing Tips

• Use System.out.println() to print testing messages to the Eclipse console (can’t use println() because Yahtzee isn’t a ConsoleProgram!)

• Hardcode dice array so you always control what the dice rolls are (great for testing logic for scoring categories)

• Think about dice value frequencies when checking if a roll fits a given category
Final Tips

READ THE JAVADOC FOR YAHTZEE DISPLAY!!!

• Follow the specifications carefully
• Extensions!
• Comment!
• Go to the LaIR if you get stuck
• **Incorporate IG feedback!**

• Have fun!