YEAH Session #5

Tuesday, February 21

Max Wang and Ashley Taylor
Slides courtesy of Nick Troccoli

YEAH Hours Schedule

Topic	Date	Time	Location
Assignment 5	Today!	Now!	Here!
Assignment 6	3/1 (Wed)	7:30-9:30PM	Bishop Auditorium
Assignment 7	3/9 (Thu)	7:30-9:30PM	Bishop Auditorium
Final Exam	3/20 (Wed)	8:30-11:30AM	TBD

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

137	42	314	271	160	178
0	1	2	3	4	5

- 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`**:

```
Type[] arr = 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)

Arrays holds a **sequence** of information

0	1	2	3	4

2D Arrays holds a **grid/matrix** of information

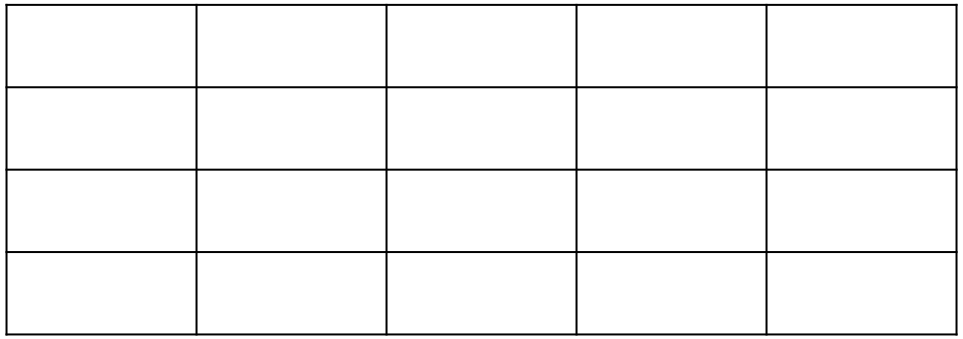
[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	...	...	...	...
[2][0]	...	...	...	...
[3][0]	...	...	...	...

Interpreting Multidimensional Arrays

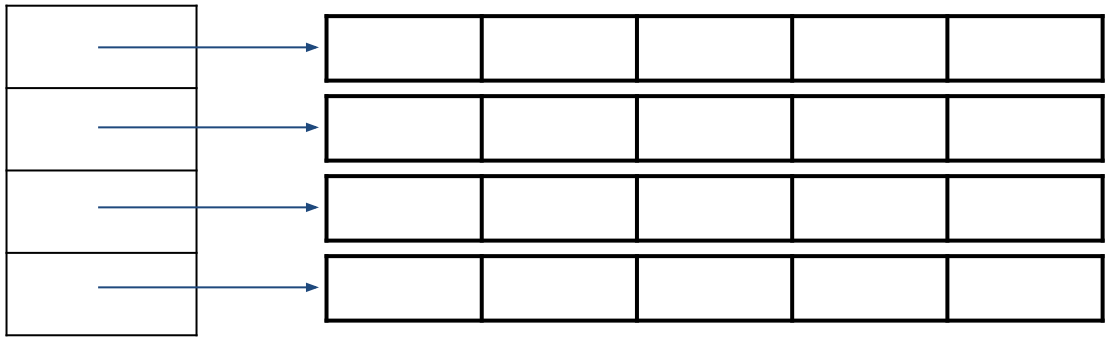
- There are two main ways of intuiting a multidimensional array.
- **As a 2D Grid:**
  - Looking up ***arr*[*row*][*col*]** selects the element in the array at position (***row***, ***col***).
- **As an array of arrays:**
  - Looking up ***arr*[*row*]** gives back a one-dimensional consisting of the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

Interpreting Multidimensional Arrays

Grid



Array of arrays



Declaring 2D Arrays

```
Type[][] a = new Type[rows][cols];
```

For example:

```
int[][] twoDArray = new int[4][5];
```

rows columns

Indexing into 2D Arrays

```
twoDArray[3]; // 4th row of array
```

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	...	...	...	...
[2][0]	...	...	...	...
[3][0]	...	...	...	...

Indexing into 2D Arrays

`twoDArray[3][1]; // element at 4th row, 2nd column`

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	...	...	...	...
[2][0]	...	...	...	...
[3][0]	...	...	...	...

Remember the order of the indices is ***`[row][column]`***

Iterating through a 2-D array

```
Type[][] arr = /* … */
for (int row = 0; row < arr.length; row++) {
  for (int col = 0; col < arr[row].length; col++) {
    /* … access arr[row][col] … */
  }
}
```

2D Array Example

```
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;
  }
}
```

	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	5
2	2	3	4	5	6
3	3	4	5	6	7

Yahtzee!

- Due at 10:30AM on Monday, Feb. 27
- 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. Rounds alternate between players, 13 rounds per player total (same number as categories)
2. Each round:
   - Roll dice
   - Pick 0-5 dice to reroll
   - Pick 0-5 dice to reroll again (total of three rolls)
   - Choose a category
   - Add appropriate points to that category

YahtzeeDisplay

- Graphics are taken care of for you
- Manipulate onscreen graphics via your YahtzeeDisplay instance variable (`display._`)
- Methods on YahtzeeDisplay (from **handout**):
  - `void waitForPlayerToClickToRoll(int player)`
  - `void displayDice(int[] dice)`
  - `void waitForPlayerToSelectDice()`
  - `boolean isDieSelected(int index)`
  - `int waitForPlayerToSelectCategory()`
  - `void updateScorecard(int category, int player, int score)`
  - `void printMessage(String message)`
- Player indices start at 1!!

Constants

```
/** The width of the application window */
   public static final int APPLICATION_WIDTH = 600;

/** The height of the application window */
   public static final int APPLICATION_HEIGHT = 350;

/** The number of dice in the game */
   public static final int N_DICE = 5;

/** The maximum number of players */
   public static final int MAX_PLAYERS = 4;

/** The total number of categories */
   public static final int N_CATEGORIES = 17;

/** The number of categories in which the player can score */
   public static final int N_SCORING_CATEGORIES = 13;

/* The constants that specify categories on the scoresheet */
   public static final int ONES = 1;
   public static final int TWOS = 2;
   public static final int THREES = 3;
   public static final int FOURS = 4;
   public static final int FIVES = 5;
   public static final int SIXES = 6;
   public static final int UPPER_SCORE = 7;
   public static final int UPPER_BONUS = 8;
   public static final int THREE_OF_A_KIND = 9;
   public static final int FOUR_OF_A_KIND = 10;
   public static final int FULL_HOUSE = 11;
   public static final int SMALL_STRAIGHT = 12;
   public static final int LARGE_STRAIGHT = 13;
   public static final int YAHTZEE = 14;
   public static final int CHANCE = 15;
   public static final int LOWER_SCORE = 16;
   public static final int TOTAL = 17;
```

Calculating Scores

- Given: a set of **dice** and the chosen **category**. Calculate: the **score**
  - Categories: 1s, 2s, 3s, full house, small straight…
- Update total score each time a player makes a move!

Are these dice valid for this category?

- 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!)

Are these dice valid for this category?

- 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?)
- For <u>testing only</u>: Use YahtzeeMagicStub initially/for testing, but *don't use it for your final submission*!

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

Testing only

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)

Development Tips

- Start with always having 1 player, then expand to multiplayer later
- Start by using YahtzeeMagicStub, then implement each category one by one
  - You can directly test your code by comparing what YahtzeeMagicStub returns for checkCategory versus what your code returns

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

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

- Have fun!