Welcome to Arrays & Hangman Lab

Learning Outcomes

By the end of this lab:

- Be able to define chapter 7 terms
- Trace, explain and write code using single dimensional arrays.
- Draw a picture of an array.
- Describe attributes of an array such as size, indexes, and element type.
- Describe declaring arrays of a specified size and with explicit values. Describe automatically initialized values when not explicitly initialized.
- Describe parallel arrays (in zyBooks multiple arrays).

Exercise A: Terms

array, element, index, array reference, new operator, brackets [] vs braces {}, swapping, searching, sorting, parallel/multiple arrays

Exercise B: Trace and Explain

For this code:
String[] fruit = new String[]{"apple", "orange", "grape"};

This is a diagram showing memory allocated and references. Each rectangle is an area of memory. The arrows represent references. A reference is information in one area of memory that refers to another area of memory. The numbers 0, 1, & 2 represent array indices.

If this fruit declaration statement was in a method, then the "fruit" variable would be on the stack in the method's stack frame when the method is called. All arrays and strings are allocated on the heap.

To help understand and explain what is happening, draw a diagram showing memory for each Java statement.

1. int [] list;
2. int [] list = new int[5];
3. int [] list = new int[]{5, 4, 3, 2, 1};
System.out.println( list);

4. char [] list1;
   list1 = new int[]{'a', 'b', 'c'};
   char [] list2 = list1;
   System.out.println( list2);

5. double [] nums = new double[]{5.0, 4.3, 2.3};
   nums[1] = 9.9;

6. char [] lettersA = {'C','B','A'};
   char [] lettersB = lettersA;

7. String []strs = new String[]"world's", "most", "popular", "language", "is", "music"};
   System.out.println( strs[3]);
   System.out.println( strs);
   for ( int i = 1; i < strs.length; ++i) {
      System.out.print( str[i] + " ");
   }

8. int size = 10;
   boolean flag = true;
   boolean [] flags = new boolean[size];
   for ( int i = flags.length -1; i>=0; --i) {
      flag = flag != true;
      flags[i] = flag;
   }

9. int []first;
   int []second = new int[]{4,5,6};
   first = second;
   second[1] = 7;
   System.out.println( first[1]);

10. int [] third = new int[];
    third[0] = 1;

11. int fourth = new int[];

12. int [] fifth = {1, 2, 3};

13. int [] list = new int[]{1,2,3,4,5,6,7};
    int temp = list[0];
    for ( int i = 1; i < list.length; ++i){
       list[i-1] = list[i];
    }
list[list.length-1] = temp;

14. int [] playerNumber = new int[]{17, 2, 12};
String [] playerName = new String[]{"Adams","Crosby","Rodgers"};
for ( int i = 0; i < playerNumber.length; ++i) {
    System.out.println("#" + playerNumber[i] + " is " + playerName[i]);
}

15. String []words = {"strength", "and", "growth", "come", "only", "through", "continuous", "effort", "and", "struggle"};
java.util.Arrays.sort( words);
System.out.println( java.util.Arrays.toString(words));

Exercise C: Default Initialization

What will the elements in this array be initialized to?
double [] items = new double[5];

If the array element data type was each of the following, what would be the default initialization values?

<table>
<thead>
<tr>
<th>data type</th>
<th>default initialization value for an element of an array</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td></td>
</tr>
</tbody>
</table>

Exercise D: Hangman Game

In this exercise you get to finish a Hangman game. Some of the code for this lab is provided via the link below, and the rest of this code will be written by you in the following steps.

The goal of Hangman is for the player to determine a secret word by guessing its letters. Initially, the player only knows how many letters are in a word. For example, the four letter word is displayed as "__ __ __", where each underscore ("_") represents a letter in the word. The player guesses one letter at a time, and if that letter is in the secret word, all occurrences of that letter are revealed. For example, if the player guesses 'a', then assume the following is revealed: "a _ a a". If the guessed letter is not in the secret word, that letter is added to
a list of incorrect guesses, and a part of the Hangman picture is drawn. If a player can determine the word before making seven incorrect guesses, s/he wins.

Step 1: Getting Started

1. Begin by starting a new java project in Eclipse, and naming it something memorable like Hangman. Next, download the following file and add it to your project: Hangman.java.
2. The Hangman class implements this game in the main method, which uses two additional public static methods. The code for this class is organized into sections. You'll be completing the code in the sections indicated below:
   1. Secret Words List (done)
   2. Constant (done)
   3. Variables (3B to be completed in Task 2)
   4. Main Loop for Game (4B, part of 4E, and 4C to be completed in Task 2)
   5. End of Game Message (done)
   6. Additional Static Methods (6A to be completed in Task 3)
3. Review the code provided, **but do not change the code at this point - you'll be doing that in the tasks below.** Familiarize yourself with the code sections, the flow of execution, and the constants and variables used. The following is a summary of the program's main loop that repeats so long as the game hasn't been won and the player hasn't used all their guesses:
   1. Call the printHangman method (which we've written) to display the current hangman picture.
   2. Print the secret word so far showing the letters that the user has guessed and the other letters as underscores.
   3. Print the incorrect letters so far.
   4. Ask the user to guess a letter.
   5. Use the Scanner object to read in the letter and then handle the guessed letter to check whether or not it is in the secret word.
4. Note we're using a separate method, named printHangman, to output the hangman picture using characters. We've put this code in its own method to make the main method more readable. Now, in the main method we only have to ask that the printHangman be done with the method call: printHangman(badGuessesCount);. Below shows what this output looks like for a point during the execution of a Hangman game (with parts of the hangman filled in for each wrong guess):
5. ____
   |    |
   | 0  |
   | \ /|
   |
   |
   |____|
   geg _ _ p _ _
Wrong guesses: t l s b
6. **TIP:** Note the blue boxes just to the right of the scroll bar in Eclipse's editor window. These boxes indicate sections that have been tagged as **TODO** items. Click on one of the blue boxes and it will take you to the corresponding "TODO" item. You can use this feature in your own programs by
putting comments in your code that begin with TODO like the comments you see in the Hangman program.

Step 2: Working with Arrays

1. Go to section 3B. of the program and complete the code as indicated for each of the TODO items in that section.
2. Go to section 4B of the program and complete the code as indicated for this TODO item. You should be able to run your program to see if the new code you added works as expected (but the game isn't finished).
3. Go to section 4E of the program and review the code there. Once you understand what this code is doing then complete the code as indicated for this TODO item. Run your program and make sure that the code you've added works correctly.
4. Go to section 4C of the program and complete the code as indicated for this TODO item. Run your program and make sure that the code you've added works correctly.

Check with your Lab TA that your code is correct before moving on to the next task. Be prepared to discuss:

- What size you chose to make the arrays.
- Which loop you chose to access the elements of the arrays
- How to refer to the size of the array.

Step 3: Playing a Complete Game

Switch so that your partner is in control of the keyboard

At this point our game is nearly complete, but it doesn't detect when a game has been won. We'll use a separate method to detect this. You'll be learning more about methods in lecture soon, but we'll give them a try now. They're a very useful way to reduce redundancy and make our programs more organized. (You only needed to use a main method in programming assignment 1. In program 2 you'll be using methods.)

In addition to main, we've already been using another method to display the hangman picture. We use the method by coding the method call statement: printHangman(badGuessesCount);. A method call has the name of the method possibly followed by additional information in parentheses that we call arguments (aka parameter values). Our hangman picture displaying method needs one argument, the number of bad guesses, so that it can determine how much of the hangman picture to draw.

The return type of this method is void indicating it doesn't return any answer. Often methods return some answer to the calling program using a return statement, which we'll describe further below. Methods that return something would need to say the type of the answer they return instead of void. Take a look at this print method to see the layout used for coding methods.

For this task, you'll complete the gameWon method that is passed the array of correct guesses and returns a boolean value (i.e., true/false). If the array of correct guesses doesn't have any blanks (i.e., '_') then all of the characters in the word have been guessed and the game is won. Go to section 6A of the program and complete the gameWon method as indicated for this TODO item. We've already coded the call of method
gameWon in the condition of the program's main game loop (while loop at top of section 4). We've also provided the shell for the method in which you'll fill the method's body.

For our main game loop to work, we need gameWon to return either true or false depending on whether or not the game has been won. We'll use a return statement as in this example: return false;. The method ends as soon as a return statement is executed and the method's call is replaced with its return value. In our program main calls gameWon, which determines if the game was won, and then returns either true or false accordingly.

**Step 4: Improving that Game**

In the lab time remaining, make the following improvements to the program:

1. Modify the program to accept uppercase letters and convert them to lowercase (hint: checkout the toLowerCase method of the Character class).
2. Modify the program to display an error if the user enters in a character that is not a letter.
3. Modify the program to display an error if the user fails to enter any character for their input.
4. Modify the program to not allow the user to guess the same letter more than once.
5. Modify the program so that the user is able to play the game until they desire to quit. Display a count of the wins and losses.

You are welcome to use additional static methods if you feel they would be helpful on these challenge tasks. Remember the first line of a method, which is called the heading, is in this form:

```
public static <return type> <methodName> (<comma separated list of parameters>)
```

Where the *return type* is the Java type of the return value (if no value is returned, use void for the return type), *methodName* is an identifier you choose, and the parameters are pairs of type name followed by parameter name (separated by commas if you use more than one parameter). Refer to the two methods in our program for examples.

**Additional Learning Materials**

When you have mastered everything in this lab, then you are welcome to learn from additional learning resources available on the web and beyond this course: https://cs200-www.cs.wisc.edu/wp/learn-to-program-resources/

*Note: Due to programs and zyBooks being individual work, it is Not appropriate to work on them during the Team Lab.*

Lab designed by Jim Williams, Hangman exercise developed by Jim Skrentny.