FCPS
Java
Packets

Unit Four – Data Storage:
Arrays, Files, Matrices

July 2018

Developed by Shane Torbert
edited by Marion Billington
under the direction of Gerry Berry
Thomas Jefferson High School for Science and Technology
Fairfax County Public Schools
Fairfax, Virginia

Contributing Authors
The author is grateful for additional contributions from Marion Billington, Charles Brewer, Margie Cross, Cathy Eagen, Anne Little, John Mitchell, John Myers, Steve Rose, John Totten, Ankur Shah, and Greg W. Price.

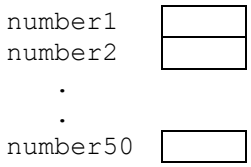
The students' supporting web site can be found at http://academics.tjhsst.edu/compsci/
The teacher's (free) FCPS Computer Science CD is available from Stephen Rose (srose@fcps.edu)

Java Instruction Plan—Unit Four

Discussion
Arrays

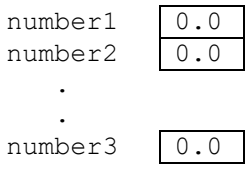
Suppose we wanted to store fifty values in fifty variables. We could just create different variables using different names.

```
double number1;
double number2;
      :
double number50;
```

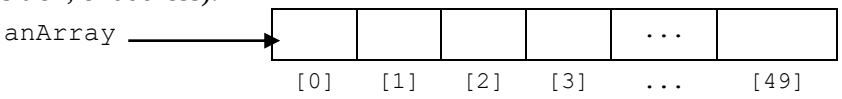


But even with copy-and-paste, this is terribly inconvenient. We would need fifty commands to initialize each of our variables to zero.

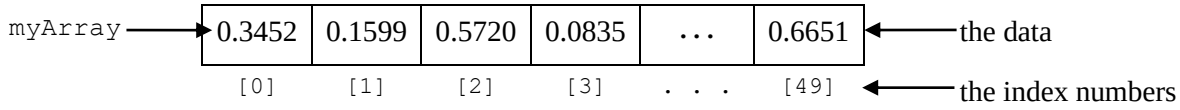
```
number1 = 0.0;
number2 = 0.0;
      :
number50 = 0.0;
```



This situation begs for an *array*, which is a collection of variables under one name, where the variables are accessed by *index* numbers. As an analogy, arrays are like lockers and the locker numbers are like index numbers. You access each locker, or *cell*, by knowing its number. In our example there are fifty cells. Each cell is numbered, starting from zero. The integer numbering each cell, in the square brackets, is that cell's *index* (or its location, or position, or address).




```
double[] myArray = new double[50];      //declares array filled with 0.0
for(int k = 0; k < 50; k++)             //the for-loop visits each cell
   myArray[k] = Math.random() * 10;     //and assigns random numbers
```



In Java, arrays are objects requiring the keyword `new`. The data in an array is accessed through the array's name, which is a reference, and the index number in square brackets. Arrays are usually processed by for-loops.

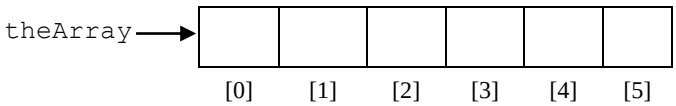
Be careful! Arrays are zero-indexed so the first cell is cell number zero and the fiftieth cell is cell number forty-nine. You cannot change this fact so you must either adjust your thinking or accept frustration.

The name of an array does not have to be `myArray`, the type of data stored in an array does not have to be `double`, and the size of an array does not have to be `50`. Here are more examples of array declarations:

```
int[] scores = new int[18];
char[] alphabet = new char[26];
String[] words = new String[NUMITEMS];    //NUMITEMS must have been initialized
```

You access individual items of data by specifying the index of the cell. Show the effects of these commands:

```
int[] theArray = new int[6];
theArray[4] = 5;
theArray[1] = 3;
theArray[4] = theArray[1];
theArray[1] = theArray[1] * 2;
System.out.println("" + theArray[1]);
```



Lab00
Hello Array

Objective
Introduction to Arrays. Introduction to Scanner class.

Background
A nice tutorial on arrays is available at http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Specification
Download the Unit4 units. In the Lab00 folder, enter the source code shown below, then compile and run.

```
 5   import java.util.Scanner;
 6   public class Driver00
 7   {
 8     public static final int NUMITEMS = 10;
 9     public static void main(String[] args)
10     {
11       double[] array = new double[NUMITEMS];
12       Scanner keyboard = new Scanner(System.in);
13       for(int x = 0; x < NUMITEMS; x++)
14       {
15          System.out.print("#" + (x+1) + ":  ");
16          array[x] = keyboard.nextDouble();
17       }
18       System.out.println("The numbers you typed in, backwards:");
19       for(int x = 0; x < NUMITEMS; x++)
20          System.out.println("\t" + array[NUMITEMS - x - 1]);
21     }
```

Line 5: The Scanner class, like the JOptionPane class in Unit 3, is used for *keyboard input*. The Scanner class, unlike JOptionPane, does not use GUIs. The traditional term for non-GUI input and output is *console I/O*.

Line 8: NUMITEMS cannot be changed; it stays the same for the whole program; its keyword is **final**.

Line 11: As usual, **new** instantiates an object, this time an array with 10 cells, filled with 0.0, ready to store decimal numbers. How many objects are instantiated in this program? _____

Line 12: Instantiate a Scanner object. The `System.in` *returns input from the keyboard*.

Lines 13-18: The for-loop counts from 0 to 9 by ones. Line 15 is the *prompt*. The array[x] in Line 16 visits each cell. `nextDouble()` is a Scanner method that reads the input (as a string) and returns the string converted to a double. What method do you think would change the string into an integer? _____

Lines 20-21: The for-loop counts from 0 to 9 by ones. The array[ ] visits each cell. The formula NUMITEMS – x – 1 calculates the index numbers we want. How does the formula work?

Extension
Insert two lines after line #20. Use a different for-loop and print the array backwards in a different way.

Sample Run
`Unit4Lab00.jar`

Exercises
Lab00

1) Draw a picture of what this declaration accomplishes:
```
int[] scores = new int[5];
```

2) Write the code to fill the array above with 100's.

3) Draw a picture of what this declaration accomplishes:
```
char[] answers = new char[4];
```

4) Write the code to fill each cell in the array above with an `'A'`.

5) Write the code to print each element of the `answers` array in the console I/O window.

6) Given an array of size NUMITEMS containing doubles, write the code to multiply each element by 10.0.

7) Given an integer array of size N which is empty, write the code to insert N number of random integers, between 5000 and 10000, inclusive.

Complete this scavenger hunt using Sun's on-line Java API.

8) What package is Scanner in? _____

9) How many constructors does Scanner define? _____

10) What are the differences between `next()`, `nextInt()`, and `nextDouble()`?

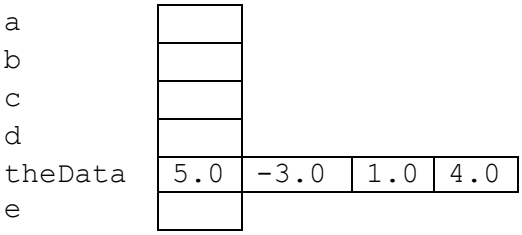
11) `hasNext()` returns a Boolean. Under what conditions might it be useful to use `hasNext()`?

12) Which do you prefer, Scanner or JOptionPane? _____ Why?

Discussion
Min and Max

The Math class defines methods `min` and `max` for both the primitive types `int` and `double`. The `Math.min` method takes two arguments and returns the smaller argument. Fill in the boxes:

```
a = Math.min(-2, -8);
b = Math.max(37.8, 1.0475);
c = Math.max(a, b);
d = Math.min(a, Math.min(b, c));
e = Math.max(theData[2], d);
```



Note carefully the code for finding the minimum of three values.

You should be aware that `min(int, int)` is a different method from `min(double, double)`. We say that the min method is *overloaded* because the same name, `min`, has different types of arguments.

```
public static int min(int arg1, int arg2)
{
   . . .
}
public static double min(double arg1, double arg2)
{
   . . .
}
```

The compiler decides which `min` method to use based on what type of arguments you pass. (Because Java does this automatically, you might not think it a big problem, but it is a big problem. Wait until you take a compiler course!) If you call `Math.min(-2, -8)` the computer will choose the correct `min` method and return an integer value, namely, -8 .

If both arguments are `int`s, the correct `min` method exists and will work. If both arguments are `double`s, the correct `min` method exists and will work. What if one argument is an `int` and the other is a `double`? In that case, the Java programmers decided that it would be convenient for Java to automatically *cast* an `int` to a `double`. Thus, in `min(-8, 37.8)`, the -8 automatically becomes -8.0 and `min(double, double)` is called, which returns -8.0 .

You should know that Java never automatically casts a `double` value to an `int`. The `double` is "too big" to fit into an `int`. To store a `double` value as an `int` you must *cast* explicitly, which you have seen before with `Math.random`. For instance:

```
int roll = (int)(Math.random() * 6 + 1);
int b    = (int)Math.max(37.8, 1.0475);
int b    =  Math.max((int)37.8, (int)1.0475);
```

In the second and third examples, the “answer” 37.8 will be *truncated* and `b` will get the integer value 37

In general, pay attention to which variables in the program are doubles and which are integers. Whether you see `int x;` or `double x;` the code is trying to tell you important information. If you aren't careful, some error messages you may see include "possible loss of precision" and "type mismatch."

Lab01
Sum, Avg, Min, Max

Objective
Processing data in an array.

Background
In Driver00 and in Driver01 we declared a constant in good Java style:

```
public static final int NUMITEMS = 10;
```

You should know what all the keywords mean. **`public`** means that objects outside the class can access it. **`static`** means that the constant belongs to the class, i.e., there is one copy of it, no matter how many objects are instantiated. **`final`** means that this value cannot be changed, i.e., a subsequent command such as NUMITEMS = 15 would be illegal. **`int`** allocates the correct amount of memory to store one item of integer data. As a matter of convention, constant values are written in ALLCAPS, which makes it easier for us to quickly read our own code. The whole line is placed before the methods, in the largest scope possible, to allow access throughout the class.

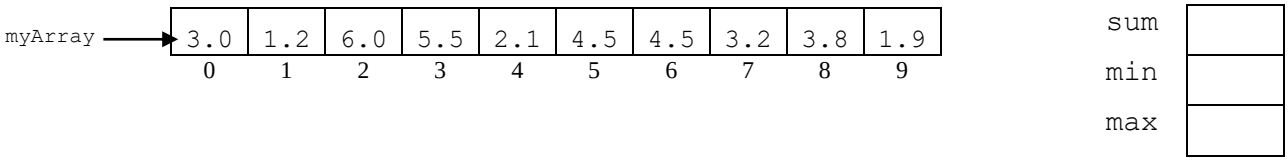
Once a Java array is created, we cannot change its length.

Arrays in Java "know" their own length. We can use the public variable `myArray.length` to return the length of the array. All the for-loops in the previous labs could have been written (and should have been written) using `myArray.length` instead of NUMITEMS:

```
for(int k=0; k < myArray.length; k++)
      System.out.println( myArray[k] );
```

For historical reasons, `length` is a variable, not a method of the array class, and does not use parentheses.

When you process an array, you typically want to do something to each cell of the array. As you have seen, a for-loop and an index number are typically used for this purpose. For example, to find the sum of the cells in the array, you would want to visit each cell and add it to the running sum. To find the minimum, you would want to visit each cell, compare it to the old minimum, and update if needed. The same algorithm finds the maximum.



Specification
Create Unit4\Lab01\Driver01.java. You may save Driver00.java as Driver01.java and modify the file. You should organize your code in three separate tasks: fill, process, and display. Fill the array from console I/O with ten decimal values. Process the array to find the sum of the data. Then calculate the average of the data. Then process the array to find the smallest value and the largest value. Display these four results to standard output using `System.out.print`.

```
Sum: 55.0
Avg: 5.5
Min: 1.0
Max: 10.0
```

Sample Run
`Unit4Lab01.jar`

Exercises
Lab01

Answer these questions.

1)
```
public static final int NUMITEMS = 100;
double[] myAr = new double[NUMITEMS];
```

Explain exactly what the two declarations above accomplish. Draw a picture of the array. Include indexes.

2) Write the code to visit each cell in the array above, and fill it with random doubles between -50 and 50.

3) Write the code to *traverse* the array, and find the smallest number in the array above.

4) Write the code that searches the array and counts the number of negative numbers.

5) Write the code to calculate the sum of all the numbers in the array above.

6) Use the result from #5 to calculate the average of the numbers in the array.

7) Traverse the array above, calculating the sum of the negative numbers and the sum of the positive numbers.

Lab02
Fahrenheit to Celsius

Objective
Parallel arrays, input and output, numeric calculations.

Background
This lab will *prompt the user* to enter ten Fahrenheit temperatures. It will *convert* the temperatures to Celsius. Finally, it will *display* the temperatures *in columns*.

The mathematical formula to convert from Fahrenheit to Celsius is $C \leftarrow \frac{5}{9}(F - 32)$

In Unit 3, Lab03, we talked about some properties of integer division and modulus division. The example there was 37/8 and 37%8. In the formula above, Java sees that 5 and 9 are both integers and therefore does integer division. What is 5/9 ? _____. You will have to find a way to force your Java code to do decimal division, so that 5/9 evaluates to 0.555555556. There are several ways to do this. Write as many as you can:

```
Fahrenheit      | Celsius
----------------|-----------------
212.0           | 100.0
32.0            | 0.0
1.0         | -17.22222222222222
2.0         | -16.666666666666668
3.0         | -16.11111111111111
4.0         | -15.555555555555555
5.0         | -15.0
6.0         | -14.444444444444445
7.0         | -13.88888888888889
8.0         | -13.333333333333334
278.8           | 137.11111111111111
451.9           | 233.27777777777777
936.8           | 502.6666666666667
116.3           | 46.833333333333336
156.8           | 69.33333333333333
```

When using `System.out.println`, the `"\t"` escape sequence inserts a tab. A tab helps to line up the columns in the table. Sometimes, the tabs don't line up quite correctly, as shown above.

You should organize your code in three separate tasks: fill the array, process the data, and print the table.

Specification
Create filename Unit4\Lab02\Driver02.java. Create two arrays of 10 cells, one for Fahrenheit temperatures and one for Celsius temperatures. Use a Scanner to read ten Fahrenheit temperatures from the user. Convert all the Fahrenheit temperatures to Celsius. Then display all the values arranged in two columns. Label the columns.

Sample Run
`Unit4Lab02.jar`

Extension
Change NUMITEMS to 15. Let the user enter 10 numbers, as above. Then use another loop to generate five more Fahrenheit random `doubles` between [0, 1000), which you know how to do. This time, display those numbers with a precision of one decimal place. There are two ways to do that, either `DecimalFormat` or `(int)`. See page Four-18.

Extension 2
Display all the values formatted to one decimal place.

Sample Run
`Unit4Lab02ext.jar`

Exercises
Lab02

1) What will be the output produced by the following code?

```
int[] a = new int[10];
for (int i = 0; i < a.length; i++)
  a[i] = 2 * i;
for (int i = 0; i < a.length; i++)
  System.out.print(a[i] + " ");
```

2) What will be the output produced by the following code?

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
for (int index = vowels.length - 1; index > 0; index--)
  System.out.println( vowels[index] );
```

3) Given the following declaration,

```
double [] apple = {12.2, -7.3, 14.2, 11.3};
```

write the code to print each number, its half, and its double.

4) Given the following declaration,

```
double [] banana = {12.2, -7.3, 14.2, 11.3};
```

write the code to find the average of the array. Then output the numbers in the array as well as how much each number differs from the average (the residual).

5) Write the code to fill an array with 20 values of the type double, read in from the keyboard.

Discussion
Reading Text Files

In Labs00 and 01, you entered data from the keyboard or console using the Scanner class and its method `nextDouble`. This works fine for entering small amounts of data. However, entering large amounts of data from the keyboard is not convenient. For large data sets it is better to *read from a text file*. That is, the data comes from a file previously stored on your hard drive. Here is Driver00, re-written to read the data from a text file. You should compare and contrast this code with the code on page Four-4. Reading from a text file requires four new statements. The prompt is no longer needed. The changes from Driver00 are underlined.

```
1  import java.io.*;                    //for File
2  import java.util.*;                  //for Scanner
3  public class Driver00
4  {
5    public static final int NUMITEMS = 10;
6    public static void main(String[] args) throws Exception
7    {
8        double[] array = new double[NUMITEMS];
9        Scanner infile = new Scanner( new File("filename.txt") );
10       for(int x = 0; x < NUMITEMS; x++)
            {//System.out.print("#" + (x+1) + ":  ");  //delete the prompt
11          array[x] = infile.nextDouble();
            }
12       infile.close();
13       System.out.println("The numbers in the file, backwards:");
14       for(int x = 0; x < NUMITEMS; x++)
15           System.out.print("\t" + array[NUMITEMS - x - 1]);
```

Lines 1 and 2: The package `java.io` contains the File class, while `java.util` contains the Scanner class.

Line 6: The compiler requires `throws Exception` because of the `new File` object on Line 9. We discuss exceptions on page 14.

Line 9: The Scanner object accepts a File object, whose argument, in quotation marks, is the name of the text file. Isn't it nice that we don't have to know exactly how these objects work!

Line 11: If you instantiate the Scanner object as in Line 9, then `infile.nextDouble()` takes care of the details of reading the text file. It even converts (if it can) the string data into double data. Cool! If the string can't be converted, the method will *throw an exception*, which stops the program. We discuss exceptions later.

Line 12: At this point, the data from the text file has been stored in the array. Close the scanner object.

Remember! Data in a text file is always stored as a string. It is the programmer's responsibility to know whether to change those strings into doubles or integers, or to leave them as strings. In the next labs, sometimes you will use `infile.nextDouble()`, sometimes `infile.nextInt()`, and sometimes `infile.next()`.

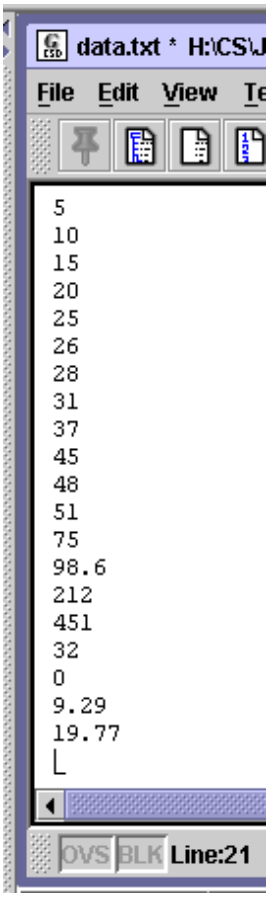
The `infile.next()` method reads one *string*—a sequence of characters up to whitespace—at a time from the file. If you put `infile.next()` in a loop, then it reads string after string, one by one, straight though until the end is reached. If there is an error in reading, the program crashes. You can't read the same string twice. Neither can you read some, then go back and start over. If you need to access the same data twice, you can either use two scanners, or read the data into an array. That is why these labs usually read data into arrays.

Be careful! The name of a file as given on Line 9 is almost never `"filename.txt"`—that is a placeholder. Similarly, the name of the Scanner object on Line 9 does not have to be `infile`.

Lab03
Text Files

Objective
To read data from a text file. To read data from multiple files.

Background
The contents of "data.txt" are shown here to the right. Rather than have the user type in all these values from the keyboard, your Driver03 program will *read input from the text file* itself. As we did before in Lab02, we will use a Scanner object, but in Lab03 the Scanner object will take its input from a file on the hard drive, rather than from the keyboard.



Otherwise, the Lab03 problem is the same as the Lab02 problem, namely, converting an array of Fahrenheit temperatures into an array Celsius temperatures. As you can imagine, this will make the code for your Driver03.java look a lot like the code from your Driver02.java.

Specification
Use Filename Unit4\Lab03\data.txt. You do not have to open this file. Your program will open and process the data in this file.

Open Filename Unit4\Lab03\Driver03.java. As in Lab02, convert a series of Fahrenheit temperatures into Celsius temperatures. This time, read the Fahrenheit data from "data.txt" into an array. Then process the array, turning all the Fahrenheit values into Celsius values. Express all calculations to two decimal places. Display the results in table form.

Sample Run
```
Unit4Lab03.jar
```

Extension 1
There is one item, the name of the text file, that is often *read from the keyboard* even if the program ultimately will *read from a text file*. Use either a JOptionPane object or a Scanner object to *prompt the user* to type in the name of the file from which to read. Create text files "data2.txt" and "data3.txt". For now, you will have to make sure to put exactly twenty items in each file because your program uses a `for`-loop to read exactly twenty items.

Extension 2
If you don't know beforehand how many items there are, you can still process the file by using a `while`-loop like that shown in Lab04 Exercises, #4. Re-write Lab03 using a `while`-loop. Test your program on data files that have 1, 2, 5, or 17 different Fahrenheit temperatures.

Exercises
Lab03

Answer these questions about file input.

1a) For every line, explain what's happening.

```
1. Scanner infile = new Scanner( new File("diskfile.txt") );
2. String s = infile.next();
3. double d = infile.nextDouble();
4. int n = infile.nextInt();
5. System.out.print("You read: " + s + ", " + d + ", " + n);
```

1b) Trace the code above. If `diskfile.txt` contains

```
Shane
23.4
18
```

what is the output?

1c) Trace the code above. If `diskfile.txt` contains

```
18 23.4 Shane
```

what happens?

1d) Trace the code above. If `diskfile.txt` contains

```
Shane
23.4
```

what happens?

2) Suppose you had to read from a text file a person's Social Security Number, two semester averages (which could be decimals), and a letter final grade. Write the code to do that.

3) What is the return type of the method `showInputDialog()`? _____

4) What is the return type of the method `next()`?_____

5) What class is the method `showInputDialog()` defined in? _____

6) What class is the method `nextDouble()` defined in? _____

7) What is the return type of `nextDouble()`?_____

8) What package must you import to use `Scanner`?_____

Discussion
Exceptions

In Unit 1 you learned about different kinds of errors, namely, syntax errors, run-time errors, and logic errors. The programmer is completely responsible for syntax and logic errors, but sometimes is not responsible for run-time errors. Run-time errors could be caused by the user, for example, if the user typed the wrong password, or misspelled the file name. Still, the programmer would like to be able to handle these kinds of error gracefully, say by offering the user another chance. To allow the programmer to attempt to recover gracefully, Java *throws an Exception*. Usually an exception is thrown in one part of the program and caught in another part of the program. It can be like a game of hot potato. First someone throws the hot potato to another, who throws it to another, and so on until someone catches it and deals with it. If no one catches it, it eventually lands back at "the system," which abruptly halts the program. The program crashes, which is neither graceful nor user-friendly.

In Lab03, when the method `nextDouble` encountered an input error, it threw an IOException to the `main`. In that lab, we decided not to deal with it there, but to pass it up to the system. We told the compiler we wanted to toss the error to the system by using `throws Exception` in the header:

```
public static void main(String[] args) throws Exception
```

If we want to be more graceful, we can *catch the exception* before it is passed on to the system. Java uses the `try/catch` block to do that. The basic structure is

```
try
{
    //run some code which might throw an exception
}
   catch (Exception e)
   {
        //if thrown, try to deal gracefully with the exception
   }
//the rest of the program
```

Because all sorts of things can go wrong, Java provides many sorts of exceptions. Go to the API and look up the Exception class. Then click on IOException and then on FileNotFoundException, which is the exception that we will be throwing in the next lab. Running off the end of an array will throw an ArrayIndexOutOfBoundsException. Incorrectly addressing an object throws a NullPointerException. Improperly casting throws a ClassCastException. Every time your program crashes, you are actually throwing a new RuntimeException, and so on.

Besides the exceptions Java has provided, you can create your own, such as `DivideByZero`, used like this:

```
try
{
   //code to prompt the user to enter a denominator
   if (denominator == 0)
            throw new DivideByZero();
}
   catch (DivideByZero e)
   {
      System.out.println("Sorry, you cannot divide by zero!");
      System.exit(0);
   }

//the rest of the program
```

Lab04
Text Files and Try/Catch

Objective
Read the first line of a text file to set the size of an array. Use `try/catch` blocks.

Background
As you know, it is easy for a user to mistype a file name. If that happens, the File constructor will search the hard drive in vain and throw a FileNotFoundException. If your program uses `new File`, you need to tell the computer how to deal with the exception. Unlike in Lab03, the `try/catch` code below catches the exception, prints a message, and ends the program.

```
public static void main(String[] args)
{
   Scanner infile = null;
   try
   {
      String filename = JOptionPane.showInputDialog("Enter filename");
      infile = new Scanner( new File(filename) );
   }
   catch(FileNotFoundException e)
   {
     JOptionPane.showMessageDialog(null, "Error: File not found.");
     System.exit(0);
   }
```

When you use a try-catch, you don't need **`throws Exception`** here.

Commands, such as `new File`, that could throw exceptions should be in the `try` block. The attempt to recover should be in the `catch` block, which is only executed if needed. Note that the `catch` block shown above does not give the user a second chance.

Specification
Use Unit4\Lab04\data.txt. This is the data file. The first line contains an integer `6525`, which tells you how big to make the array.

```
6525
390.0202356936466
948.1569277972972
638.3505188754447
574.3344213810659
        .
        .
```

Open Unit4\Lab04\Driver04.java. First, prompt the user to enter the filename, using the try/catch structure. Then read the integer, create an array of that size, and read the data into the array. (Note that, in this lab, the line `public static final int NUMITEMS = 6525;` does not exist because `NUMITEMS` will change for each data file.) Finally, as in Lab01, process the array to calculate the sum, average, max, and min.

Sample Run
`Unit4Lab04.jar`

Extensions
1. Compile and run `MakeDataFile.java`. Verify that the set of data in `"data.txt"` is different. Run `Driver04.java` again. How does the same code handle different numbers of items in the file?

2. Put a loop around the try/catch block. Keep prompting the user to enter a filename until an existing infile is found.

3. Modify the program again so that the user is given only three chances to enter a good filename. If the three chances are used up, display an appropriate message and terminate the program.

Lab04
Exercises #1

Below are the beginnings of three programming assignments. Read them carefully for all the information.

1. Your professor assigns you to write part of a program that processes a set of exam scores. She says there are exactly 181 exams. The scores are in a text file "exam01.txt", one score per line. Write the code fragment to read the exam scores into an array.

2. Up till now, your professor has kept exam scores on paper. She assigns you to write a program so that someone can enter the exam scores into an array, from the keyboard. The number of scores has to be entered from the keyboard as well. She wants you to use good, friendly prompts.

3. Your professor teaches several courses. Each set of exam scores is saved to disk under different filenames. Each set of exam scores has an integer in the first line, which tells the number of exam scores in the file. Write the code fragment that prompts the user for the file name, reads the first line, and creates an array of the correct size.

4. It turns out that, to calculate the sum, average, and max, you don't need to store the data in an array. In this case you can process the data from the text file on the fly, without knowing in advance how much data you have. The Scanner class has two methods `hasNext` and `nextDouble` to help. Fill in the blanks:

```
Scanner infile = new Scanner( new File("data.txt") );
double sum = ____;
double max = Double.MIN_VALUE;
int count = ____;
while( infile.hasNext() )
{
    double x =  infile.nextDouble();
    sum = sum + _____;
     max = Math.max(_____________,  ____);
     count++;
}
System.out.println("Sum:  " + _______);
System.out.println("Average:  " + ____________________________________);
System.out.println("Max:  " + _______);
```

Lab04
Exercises #2

Study the code of `MakeDataFile`*, shown below. In Extension 1 of Lab04, you used* `MakeDataFile` *to create a different* `data.txt`*. How does* `MakeDataFile` *work? Answer the questions below.*

```
 4  import java.io.*;
 5  public class MakeDataFile
 6  {
 7     public static void main(String[] args) throws Exception
 8     {
 9       PrintWriter outfile = new PrintWriter(new FileWriter("data.txt"));
10
11       int numitems = (int)(Math.random() * 5000 + 5000);
12       outfile.println(numitems);
13
14       for(int x = 0; x < numitems; x++)
15          outfile.println(Math.random() * 1000);
16       outfile.close();
17    }
19  }
```

1) What does Line 9 do?

2) Why does Line 7 need `throws Exception`?

3) How can you throw the exception in a different way?

4) What does Line 11 do?

5) What does Line 12 do?

6) What do Lines 14 and 15 do?

7) What does Line 16 do?

8) In the code above, `outfile` is a reference to a _____object.

9) In the code above, the object referenced by `outfile` was instantiated out of class _____

10) If a "data.txt" file already exists on the hard drive, does **`new`** `FileWriter("data.txt")` overwrite that file or not?

11) In your own words, briefly describe what `MakeDataFile` does.

Discussion
Decimal Formatting

Say you want to *truncate* a double type value (e.g. 1.2345678) to two decimal places (to 1.23). One option is:

```
double x = 1.2345678;
double num = (int)(x * 100) / 100.0;
```

If x equals 1.2345678, then x * 100 is 123.45678, which when cast to an integer yields 123. Dividing this by 100.0 results in the desired 1.23. Make sure to use 100.0 as shown because 123 / 100 is just 1. (Why?) To *round* positive numbers you must add 0.5 before casting. To round negative numbers, add -0.5 before casting.

Truncating actually changes the number. If you don't want the number to be changed, but only to be displayed differently, use the DecimalFormat class from the `java.text.*` package. The desired formatting pattern is passed as a string "0.00" to the DecimalFormat constructor. A call to the `format` method, passing a double argument, converts the double type value to its string representation in the desired formatting pattern.

```
DecimalFormat d2 = new DecimalFormat("0.00");
String s = d2.format(x);
```

Be careful! The return value of the `format` method is a String, not a number. A string.

Discussion
`do-while` loops

You have known about the `while` loop since Unit 1.

```
while(CONDITION)
{
   COMMANDS;
}
```

The `do-while` loop is a similar structure.

```
do
{
   COMMANDS;
}while(CONDITION);
```

The difference between these two control structures is that the `while`-loop might not ever execute its commands at all. If the `while`-loop condition is false initially then the loop is skipped.

On the other hand, a `do-while` loop will always execute its commands at least one time. It keeps on looping while the condition is `true`. (Or: until the condition is `false`.) This behavior can be very useful, as we will see in the next lab.

The `while`–loop is a *precondition loop* and the `do-while` is a *postcondition loop*. Both are *indefinite* loops, in contrast to a `for`-loop, which is a *definite* loop.

Here is a reminder: when you know in advance how many loops will run, use a `for`-loop. On the other hand, when you don't know in advance how many loops will run, use either a `while` or a `do-while` loop.

Lab05
Luck of Many Rolls

Objective
`do. . .while` loop. Dice class. Frequency table for data.

Background
This lab conducts three different experiments that roll two dice. **1)** How many rolls does it take to roll boxcars (double 6's)? **2)** How many rolls does it take to roll doubles? **3)** If we roll 100 times, how many 2's, 3's, 4's, etc. do we get?

What private variables would it make sense for a Dice class to posses?

What tools does the Dice class give us? You look at its API, shown at the right.

- `doubles` returns a boolean, which is true/false. What do you think `doubles` tests for?
- `roll` is void. What do you think `roll` does?
- `toString` returns a string of the *private data* in the dice object, i.e., what the dice show after you roll them. This `toString` *overrides* the `toString` inherited from Object, which we will discuss in Lab07.
- `total` returns an integer. What do you think `total` does?

public class **Dice**
extends Object

public class **Dice**
extends Object

Constructor Summary
Dice()

Method Summary	
boolean	**doubles**()
void	**roll**()
String	**toString**()
int	**total**()

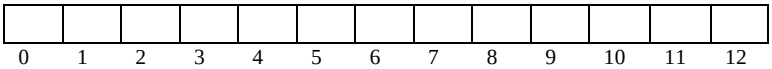
The code for **Experiment 1** is shown here to the right. Note Line 7. On Line 11 a Dice object is created. Objects have methods and private data. On Line 12 a count variable is initialized to zero. The count is incremented (line 16) each time we roll the dice. We roll the dice until we get double-6's (Line 17). We use a `do-while` loop here because we know that at least one roll is required.

```
5  public class Driver07
6  {
7   public static final int TRIALS = 100;
8
9   public static void main(String[] args)
10  {
11   Dice d = new Dice();
12   int count = 0;
13   do
14   {
15    d.roll();
16    count = count + 1;
17   }while(d.total() != 12);
18   System.out.println("It took " + count + " rolls to
```

Specification
Open Unit4\Lab05\Dice.java. Complete the implementation of the class.

Open Unit4\Lab05\Driver05.java. **Experiment 2:** Add code to count how many rolls it takes to get any doubles, not necessarily boxcars. When you finally roll doubles, report the values of the doubles.

Experiment 3: add code to make a *frequency table* showing the results for 100 rolls of the dice. In a frequency table, the number you rolled is also the index number in the array. You'll need to make an array (from 0 to 12 is helpful here) and store the number of 2's, 3's, 4's, etc., that were rolled.



Experiment 3 extension: count and report the number of doubles rolled in those 100 rolls.
Sample Run `Unit4Lab05.jar`

Exercises
Lab05

1) There are three kinds of loops. What are they? _____ _____ _____

2a) What is the output of this loop?

```
int limit = 6;
for (int count = 1; count<=limit; count++)
      System.out.println(count + " squared is " + count * count);
```

2b) Rewrite the loop as a while-loop.

2c) Rewrite the loop in 2a as a do-while loop.

3) In one phrase, what does this loop accomplish?

```
int sum = 0, count = 0, grade;
grade = Integer.parseInt(JOptionPane.showInputDialog("Enter a grade:"));
while (grade > -1)
   {
   count++;
   sum += grade;
   grade=Integer.parseInt(JOptionPane.showInputDialog("Enter a grade:"));
   }
System.out.println("The average is " + (double)sum/count);
```

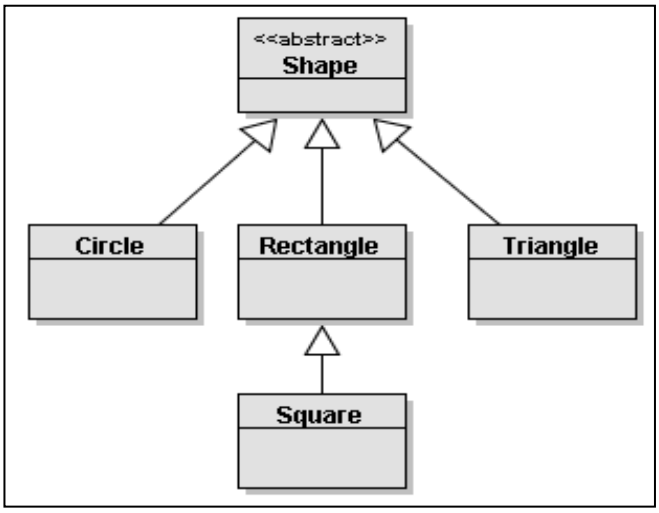
3b) Rewrite the loop above as a do-while loop.

4) Since a while-loop tests the boolean condition at the _____ of the body, the body sometimes _____.

5) Since a do-while loop tests the boolean condition at the _____ of the body, the body _____ executes at least once.

Discussion
Shape Hierarchy

The shape classes represent geometric shapes numerically, not graphically. Note that the root of this hierarchy is an abstract class. Why does it make sense that Shape is "abstract"?




```
public abstract class Shape
{
  public abstract double findArea();
}
```

Which classes in the hierarchy are concrete? What makes them concrete?

Any time you encounter a new class, you need to ask yourself, "what does this class know about itself?"

What do all Shapes know how to do?

To find out what Circles know, look at its API. What private data does a circle know about itself? What methods does this Circle class know how to do? What method *must* be implemented in the Circle class? Why?

Let's go on to Rectangle. What private data should a rectangle know about itself? What methods would be useful in rectangles?

Similarly, plan the Triangle class, which models equilateral triangles.

Now plan the Square class. Note carefully that Square extends Rectangle. Why does that make sense? What are the consequences of that inheritance? If you look at Unit 4's API for Square (not Java's API for Square!), note carefully that Square does **not** use a private field `mySide`; it somehow accesses Rectangle's private fields `myWidth` and `myHeight`.

In implementing the methods, recall that the Math class contains a static constant `Math.PI` and a static method `Math.sqrt`.

Note that Lab06 does not involve an array. It is Lab07 that will make arrays of these objects. Your job in Lab06 is to make the instance methods work. Possibly a look at the Bug class from Unit 2, Lab05 will help you with the keywords and syntax for constructors, the get and set methods, and the return methods.

public class **Circle**
extends Shape

Field Summary

private double	**myRadius**

Constructor Summary

Circle(double x)
Constructs a circle with initial radius specified by x.

Method Summary

double	**findArea**() Calculates and returns the circle's area.
double	**findCircumference**() Calculates and returns the circle's circumference.
double	**getRadius**() Returns the circle's radius
void	**setRadius**(double x) Sets the radius to the input number.

Lab06
Shapes and Areas

Objective
Abstract classes, concrete classes, output to a text file.

Background
Driver06 is a tester program. It instantiates four objects and tests their instance methods. In the code below, a circle object is instantiated that has a radius of 75. The `println` methods label the output nicely. Then, by using `c.getRadius()`, it "sends a message" to the object. The object "knows" how to respond to the message. The driver program is the client and the object is the server.

```
Circle c = new Circle(75);
System.out.println("Circle");
System.out.println("------");
System.out.println("Radius: " + c.getRadius());
System.out.println("Area: " + c.findArea());
System.out.println("Circumference: " + c.findCircumference());
```

Your job is to make the instance methods work. Perhaps a look at the Bug class from Unit 2, Lab05 will help you with the keywords and syntax for constructors, the get and set methods, and the return methods.

In this lab, the output will not go to the monitor, but to a text file called "`output.txt`". (We did this before with `MakeDataFile` in Lab04, Exercises #2.) The relevant line is:

```
PrintWriter toFile = new PrintWriter(new FileWriter("output.txt"));
```

When your program runs nothing will appear on the monitor. All the output is saved on the disk. To view the results, open "`output.txt`" *after* your program is finished executing.

Specification
Open Unit4\Lab06\Shape.java. This abstract class is the superclass of all the other subclasses in this lab. You cannot instantiate an abstract class, e.g. `Shape shape = new Shape();` is illegal.

Open Unit4\Lab06\Circle.java. Notice the private data field. Your job is to complete the constructor, the get and set methods, and the instance methods.

Create Unit4\Lab06\Rectangle.java. Implement the rectangle class. What private data should a rectangle know? What methods should a rectangle be able to do?

Create Unit4\Lab06\Square.java. Implement the Square class, which extends Rectangle. Square inherits public methods from Rectangle. Square's constructor will use `super`. Square is somehow able to use `myLength` and `myHeight`, which are private fields in Rectangle!

Create Unit4\Lab06\Triangle.java. Implement the equilateral triangle class. What private data should an equilateral triangle know about itself? Every Algebra II student should know how to calculate the area of an equilateral triangle.

Open Unit4\Lab06\Driver06.java. Complete the implementation of `main` to test the four concrete shape classes.

After your application runs, open Unit4\Lab06\output.txt. Look at the output.

Lab06 Extension
Producing an API

Your teachers have often directed you to look at the API because the API specifies what each class and method does. Even professional programmers refer to the API all the time. Because the API is such an important document, Java has made the javadoc program to produce it for each class you write. Suppose you write the `Foo` class. The command `javadoc Foo` will create a file called `Foo.html` which contains the comments, the brown tabs, the horizontal lines, and the light blue background. You can copy and paste this `Foo.html` file directly into your website, as happened here https://academics.tjhsst.edu/compsci/CSweb/Unit4/api/index.html

All javadoc documents follow the same conventions. Study the example below for `Circle.java`. Each javadoc comment begins with `/**` and ends with `*/` Descriptive comments precede the class and each method. Javadoc comments can include @ tags, such as @author, @version, @param, and @return, which specify information about author, version, arguments, and return types.

```
/*************************************************************
* A Circle is a Shape that maintains information about its radius. A
* Circle knows how to return its radius, set its radius, calculate and
* return its area, and calculate and return its circumference.
* @author Torbert, e-mail: mr@torbert.com
* @version 7.14.2003
*************************************************************/
   public class Circle extends Shape
   {
      private double myRadius;
   /*************************************************************
   * Constructs a circle with initial radius specified by x.
   * @param x    initial radius
   *************************************************************/
     public Circle(double x)
     {
        myRadius = x;
     }
   /*************************************************************
   * Returns the circle's radius
   * @return radius
   *************************************************************/
      public double getRadius()
     {
        return myRadius;
     }
   /*************************************************************
   * Sets the radius to the input number.
   * @param x     assigns x to myRadius
   *************************************************************/
       public void setRadius(double x)
      {
         myRadius = x;
```

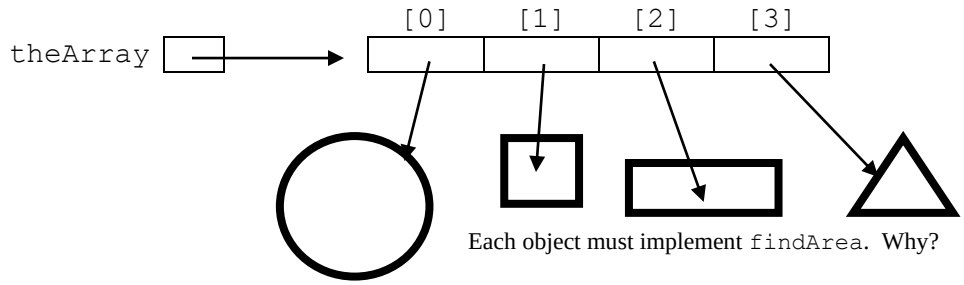
Assignment

Add appropriate javadoc comments to your `Rectangle.java` file. Run the javadoc program on that to produce a well-commented Rectangle API. Ask your teacher how to run javadoc on your system.

Discussion
Polymorphism

Any given array can only store one type of data, e.g., `int` or `double` or Shape. However, you can store different subclass objects in an array of references to the superclass—and even if the superclass is an abstract class. In other words, an array of Shapes can reference objects instantiated from the concrete subclasses of Shape. Like this:

```
Shape[] theArray = new Shape[4];    //instantiate an array of references
theArray[0] = new Circle(75);
theArray[1] = new Square(50.0);
theArray[2] = new Rectangle(30., 60.);
theArray[3] = new Triangle(77);
```



Does this array store references to Shapes? Or does it store references to instances of circles, squares, rectangles, and triangles?

Polymorphism, which works through a superclass reference to a subclass object, is put into use with the code:

```
for(int k = 0; k < theArray.length; k++)
   theArray[k].findArea();
```

Which of the four `findArea` methods get executed? That depends on the type of object stored in each cell. Each object knows its own `findArea` method. Remember that Java uses dynamic method binding so the method actually used at runtime is determined by the type of the object and not by the type of the reference. It's a good thing, too, because in this case the references in the array all point to Shapes, and Shape doesn't even have a body for the `findArea` method. Each reference points to a Shape, the superclass of the actual object, and the object implements `findArea`, which gets executed. That is how Java produces polymorphic behavior.

We showed an example of filling an array of shapes in a fixed manner above. Alternatively we could have used this short-cut:

```
Shape[] theArray = {  new Circle(75),
                      new Square(50.0),
                      new Rectangle(30., 60.),
                      new Triangle(77)
                   };
```

You have actually seen this shortcut once before in Unit 2, where we drew a polygon which required an array of x-coordinates. We instantiated that array with `int[] xPoints = {50, 150, 200};`

In Lab07 you have to fill a randomly-sized array of shapes with randomly-chosen shapes.

Lab07
Array of Shapes

Objective
Array of abstract objects. Polymorphic behavior. String representation of objects.

Background
There are two steps in constructing an array of objects. First you must construct the array itself, then you must construct each individual object. Since arrays in Java are treated as objects you must use the keyword `new` twice in this process: once for the array and once for the contents of the array.

For example:
```
Foo[] fooAry = new Foo[100];                  //create an empty array
for(int k = 0; k < fooAry.length; k++)        //fill it
      fooAry[k] = new Foo();                  //         with objects
```

Be careful! The array is of type Foo[], the contents of the array are of type Foo, and the constructor is Foo().

The method `toString` can be called on any object, because the Object class, from which every Java class extends, contains a default implementation for the method `toString`. Given any Java object `obj`, `obj.toString()` returns the name of the object's class, the @ symbol, and the *hexadecimal address* where the object is stored inside your computer. For example, for any object `obj` of class `Foo`, then `System.out.println(obj.toString())` might show its *string representation* as `"Foo@45a877"`. Here Java allows a shortcut, so that `System.out.println(obj)` automatically calls the `toString` method and will show the same string representation.

In most programs, the default string representation is not very useful. Usually, the programmer *overrides* the default `toString` so that it returns the object's private data. You did this, for example, in Lab05 when the Dice's `toString` returned the values of the dice, nicely formatted with parentheses and a comma.

Specification
Create Unit4\Lab07\Driver07.java. Instantiate an array of some largish, random size. Fill each cell of the array by instantiating different objects (either circle, rectangle, triangle, or square) at random. Do not merely alternate among the four in a predictable fashion! Also make sure to create the sizes of the individual objects at random, i.e. two circles should have different, randomly-generated radii. After the array is filled, output the area of each object and the string representation of each object to the file `output.txt`.

Copy Shape, Circle, Rectangle, Square, and Triangle from the Unit4\Lab06 folder.

After your application runs, open Unit4\Lab07\output.txt and view the output.

Sample Run
`Unit4Lab07.jar`

```
Shapes
------
area = 6666.2649    Circle@45a877
area = 6153.6139    Rectangle@ad3ba4
area = 81.9818      Square@6b97fd
area = 24285.3185   Circle@1e63e3d
area = 1152.9965    Triangle@18fe7c
area = 50.6994      Rectangle@d3aee3
< etc. >
```

Extensions:
1) Replace `array[x].findArea()` with `array[x].getLength()`. Explain why it won't compile.

2) Is it possible to print each Rectangle's length? Y/N Explain.

Exercises
Lab07

```
public abstract Animal
{
   public abstract String speak();
}
public Dog extends Animal
{
   public String speak()
   {
      return "bow-wow";
   }
}
public Cat extends Animal
{
   public String speak()
   {
      return "meow";
   }
}
```

```
public Parakeet extends Animal
{
   ____________  __________  ____________
   {
      _________  _________
   }
}
public NoisyDog extends Dog
{
   public String speak()
   {
      return super.speak() + "-" + super.speak();
   }
}
```

1. Draw the inheritance hierarchy for the classes above.

2. Given: `Animal[] myPets = { new Cat(), new Cat(), new Parakeet(), new Dog, new NoisyDog() };`
   a. Draw the array declared above.

   b. Write a loop that goes through the array and commands each animal to speak.

   c. Write the output of that for-loop:

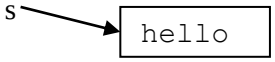
   d. Which method shows polymorphic behavior? _____ Why?

3. Are these statements legal?
   a. `Animal[] yourPets = new Animal[3];` Y/N
   b. `Animal[] felines = new Cat[300];` Y/N
   c. `Cat[] myCats = new Cat[3];` Y/N
   d. `Cat[] myPets = new Animal[5];` Y/N
   e. `Cat[] felines = new Dog[2];` Y/N
   f. `Dog[] dogPound = new NoisyDog[10];` Y/N

4. Which instantiation(s) in #3 show a superclass reference to subclass objects?

Discussion
Strings

A "string" to a computer scientist is a sequence of characters. (Characters are single letters, symbols, or punctuation marks.) Some examples of strings are "ABC", "1.23", "123", "$", and even "". In Java, strings are implemented as objects, instances of the String class, but in many ways strings act unlike other objects. For example, you can instantiate a string without using the keyword `new`.



`String s = "hello";` is short for `String s = new String("hello");`

The only overloaded operators for objects in Java is the "+" sign, used for string *concatenation*. "Concatenate" means to paste together. The three commands shown below produce the string "Concatenation".

```
String str = "Concat";
str = str + "enat";
str += "ion";
```

If you want to compare strings for equality, do not use ==. Instead, use the `equals` method. Whenever you compare objects, you must use `equals`. Any object can call `equals` because there is a default implementation in Object. The concrete class, in this case String, overrides that `equals` by defining an `equals` that is appropriate to strings, i.e., the letters match.

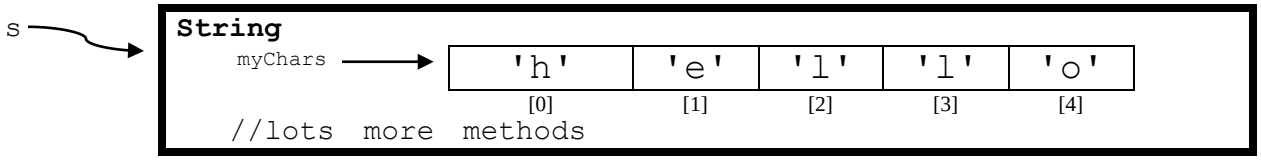
In the case above, `str.equals("Concatenation")` returns `true`, and `str.equals(s)` returns `false`. Unfortunately for you, `str == s` is legal and it compiles. (It compares the hexadecimal addresses of the two objects, which in this case returns `false`, because the objects are stored in different hex addresses.)

The case of the letters matter, so that the string "Hello" and the string "hello" are not the same. `s.equals("hello")` returns `true` but `s.equals("Hello")` returns `false`.

You could also compare them the other way around, because string literals are objects themselves. For instance, `"hello".equals(s)` returns `true` but `"Hello".equals(s)` returns `false`.

Strings are *immutable*, meaning that once a string object is created it can never be changed, which is equivalent to saying that strings do not have set methods. In the concatenation example above, it looks like there is only one string, but actually there are three different immutable string objects that exist at one time or another: "Concat", "Concatenat", and "Concatenation". The `str` reference changes to point to three different objects.

As we said above, strings are sequences of characters. Characters are single letters, symbols, or punctuation marks, and are stored in primitive type `char` variables. Java denotes a `char` value with single quotation marks, such as `'a'`, `'>'`, or `'.'`. Every String object has a private field holding an array of characters. You should picture String `s` as a reference pointing to a String object that stores an array of characters, like this:

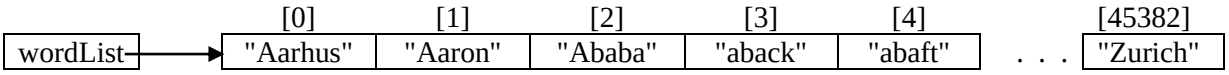


The reason programmers go to the trouble of making classes is not only to store data but also to provide methods to work on that data. Java's String class has many methods which are all, of course, listed in the String API. Be sure to look up `equals`, `equalsIgnoreCase`, and `compareTo`.

Lab08
Dictionary

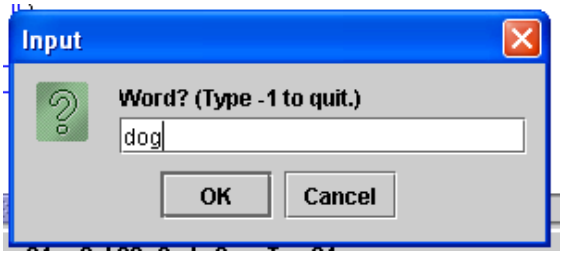
Objective
To read in, and to search, an array of strings.

Background
As in previous labs, we will be reading data from the hard drive into an array, processing the array, and outputting the results. In this lab, the data is strings, as opposed to integers or doubles. A string (in computer science) is a sequence of characters up to white space. A String in Java is an object that stores and manipulates strings. After reading in the string data, using the techniques in Lab04, your array would look like:



In this lab we are going to repeatedly prompt the user to enter a word. One way to loop repeatedly is to use an infinite loop that keeps prompting the user until the user wants to stop, in this case by typing "-1".

```
while(true)
{
  String myWord = JOptionPane.showInputDialog("Word? (Type -1 to quit:)");
  if(myWord.equals("-1"))
     break;
  //your code goes here.
}
```



This would normally loop without ending, since the condition of the while-loop is hard coded to be `true`. However, when the user types in "–1" (without quotation marks) then the `break` statement ends the infinite loop.

Specification
Use Unit4\Lab08\words.txt. Open this file and look at it. The first item in `words.txt` is an integer indicating how many words are in the file. Driver08 will open this file, read the integer, and then read the list of words into an array. You did something similar in Lab04.

Create Unit4\Lab08\Driver08.java. Let's make a simple spell checker. Read the contents of `words.txt` into an array of strings. (The first item in `words.txt` is an integer indicating how many words are in the file.) Inside an infinite loop, prompt the user to enter a string. Search the array for the string and report whether it is in the array. If the word is there, also display the word's index (its position) in the array. Continue this spell-checking process until the user wants to quit by typing "-1" (without the quotation marks). Then print "goodbye" and exit the program.

Sample Run
`Unit4Lab08.jar`

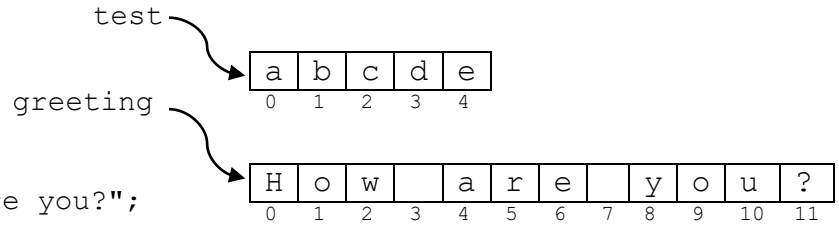
```
Yes, "dog" is a word, #13380.
Sorry, "God" is not a word.
Yes, "god" is a word, #18516.
Good-bye.
```

Extension
Look at Java's API for Strings. Change your code so that it tests strings for equality while ignoring capital letters. After that change, the spell-checker will confirm that "God", "GOD", and "goD" are all words in the dictionary.

Exercises
Lab08

Given the strings:

```
String test = "abcde";
String greeting = "How are you?";
```



What is the output of the following code fragments?

1. `System.out.println(greeting + " " + test);` _____

2. `System.out.println("ab\ncd\te");` _____

3. `System.out.println(""+test.length());` _____

4. `System.out.println(test.charAt(2));` _____

5. `System.out.println(test.indexOf("c"));` _____

6. `System.out.println(greeting.substring(4));` _____

7. `System.out.println(greeting.substring(0, 4));` _____

8. `System.out.println(test.substring(0,1));` _____

9. `System.out.println(test.substring(2,3));` _____

10.
```
String s1 = greeting.substring(0,8);
String s2 = s1.substring(4);
String s3 = greeting.substring(4,7);
System.out.println(s2 + s3);
```
_____

11. `System.out.println(test.toUpperCase());` _____

12. `System.out.println(test.equals("abcd"));` _____

13. `System.out.println(test.equals("ABCDE"));` _____

14. `System.out.println(test.equalsIgnoreCase("ABCDE"));` _____

15. What type of argument is required by the String method `charAt()`? _____

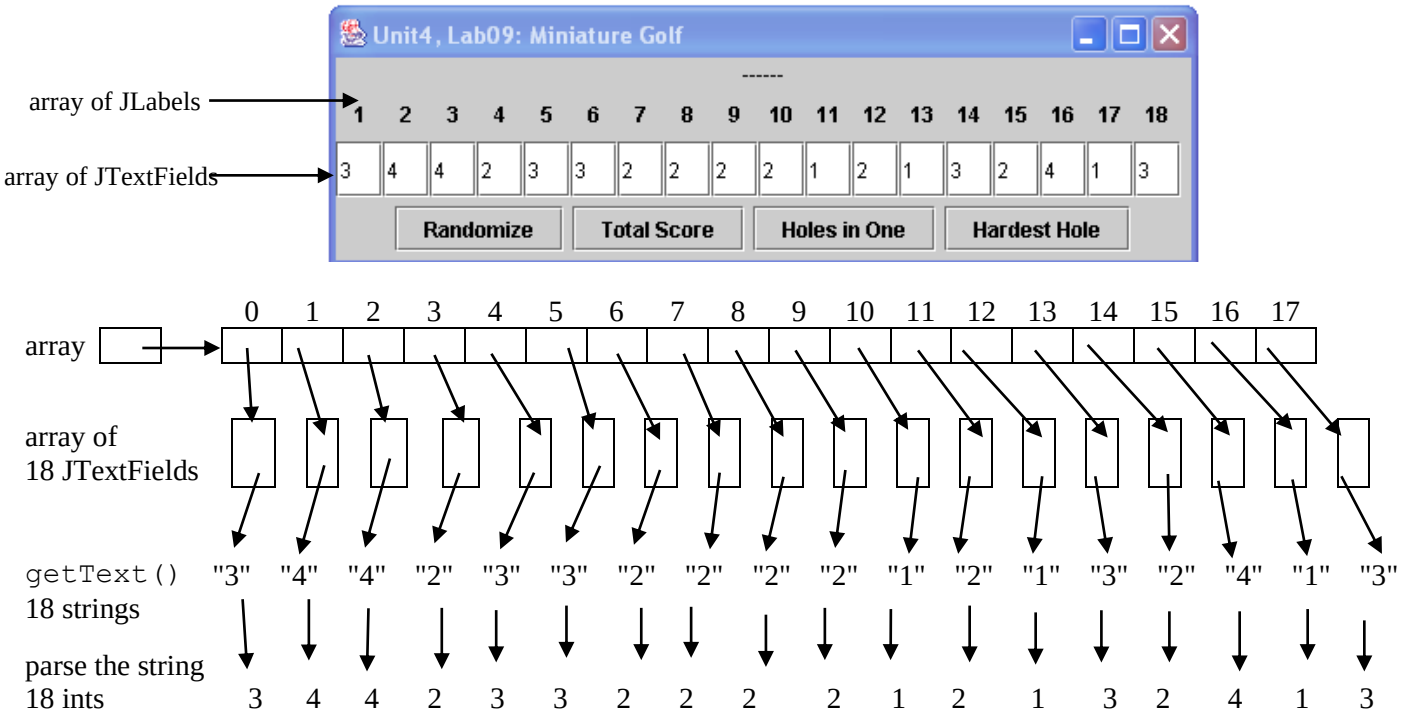
16. What is the return type of the String method `charAt()`? _____

17. How do you change a `char` into a String? _____

18. How do you change a single-character String into a `char`? _____

Discussion
An Array of Text Fields

An array of text fields in a JPanel provides a good visual reinforcement of the structure of an array.



You must understand that each cell in the array does not store the numbers directly, as they did in previous labs. Instead, the numbers are stored in JTextFields, each of which is an object. To access the private data in the JTextField, you'll need to use its `getText` and `setText` methods. (Look at the JTextField API, or recall the Unit 3 labs.) You should recall that `getText` returns a String while `setText` requires a String argument. Yet we will be wanting to process the data as integers, adding and averaging and so on. That means that you will always be converting strings to integers and integers to strings. How did you do that before?

As usual, creating an array of objects is a two-step process (lines 18 to 21).

Adding 18 JTextFields to the GUI is a third step (line 22). These JTextFields are empty at this point.

The fourth step is to put data into the 18 JTextFields. That is your job.

```
17
18      input = new JTextField[18];
19      for(int x = 0; x < input.length; x++)
20      {
21          input[x] = new JTextField();
22          add(input[x]);
23      }
```

The last steps are to process the data in the JTextFields. You'll be writing the code that executes when the buttons are clicked.

Lab09
Miniature Golf

Objective
Work with an array of JTextFields.

Background
This lab models a scorecard for a round of miniature golf. It also processes the array to calculate three different statistics: the total for the round, the numbers of holes-in-one (aces), and the hole with the highest score.

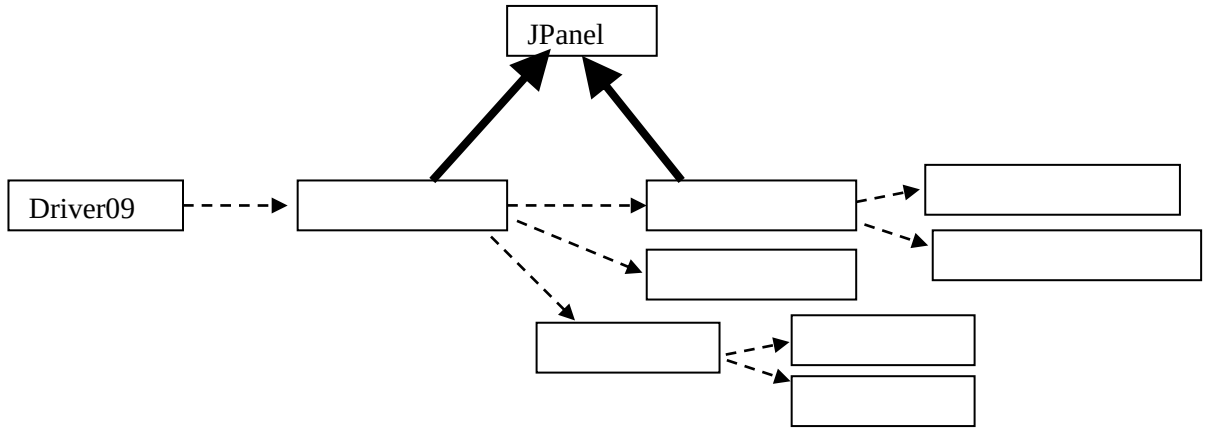
Specification
Open and look at Unit4\Lab09\Driver09.java. This is our standard GUI driver. You don't have to write any code. Notice that Driver09 hasa Panel09, which isa JPanel.

Open Unit4\Lab09\Panel09.java. What layout is in Panel09? Where is the code for the Scorecard object? You have to call the private factory method to make the buttons. You have to write code for three of the four listeners. These three listeners each call a method in Scorecard and display the results in the label in the NORTH.

Open Unit4\Lab09\ScoreCard09.java. What layout is in Scorecard09? You have to add 18 JLabels. You have to add an array of 18 JTextFields (see the previous page). ScoreCard09 also implements four methods that execute when Panel09's buttons are clicked. Complete the implementation of the four methods: `randomize`, `findTotal`, `findAces`, and `findHardestHole`. The `findHardestHole` method displays the number of the hole with the highest score. If several "hardest holes" have the same score, show the first one.

Sample Run
`Unit4Lab09.jar`

This lab, with panels and buttons, is structured like the labs in Unit 3. Complete the UML diagram for Lab09. What do the dashed lines represent? What do the thick solid arrows represent?



Lab10
Binary Reversal

Objective
Manipulate an array of labels.

Background
At the lowest level, computers calculate in binary, i.e., in zeros and ones. Common binary arithmetic operations include Shift and Rotate. The binary sequence shown here represents 210 in our normal base-10 system. Recall that the right-most digit has the smallest place value. The left-most digit has the largest place value.

`randomize` sets each digit independently to either a zero or a one at random. Do not flip a coin and then set all the digits to zero or all the digits to one.

`reverse` swaps the first and last digits, swaps the second and seventh digits, swaps the third and sixth digits, and swaps the fourth and fifth digits. There are several ways to do this. One way uses a temporary storage array.

`shift` makes each digit equal to its neighbor on its right, effectively shifting all the digits one place to the **left**. The left-most digit's value is discarded and the right-most digit's value becomes zero.

`rotate` is exactly like `shift` except the digits wrap around: the left-most digit is saved, and eventually becomes the right-most digit.

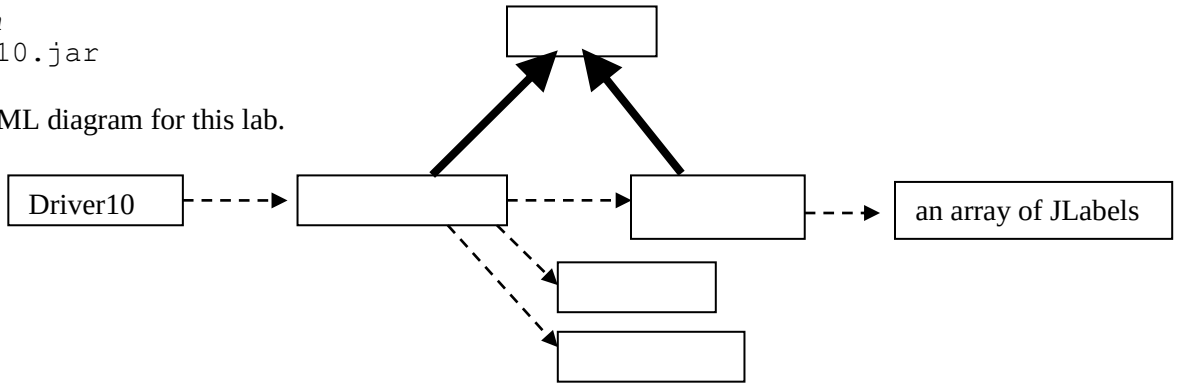
Specification
Open Unit4\Lab10\Driver10.java. Driver10 is our usual driver, instantiating and calling Panel10.

Open Unit4\Lab10\Panel10.java. Panel10 uses a borderLayout. Panel10 hasa Display10 in its center. The listeners in Panel10 send messages to the methods owned by an object of the classDisplay10.

Open Unit4\Lab10\Display10.java. Display10 uses an array of eight JLabels. Complete the implementation of the four methods `randomize`, `reverse`, `shift`, and `rotate`. In this lab, you don't have to convert the text of these labels to numbers, you just have to move the text around.

Sample Run
`Unit4Lab10.jar`

Fill in the UML diagram for this lab.



Discussion
Binary Number System

Our number system is a *place value* system. (Roman numerals is an example of a number system that does not use place value.) The base of our number system is 10, probably because we have ten fingers. Each *digit* is multiplied by its *place value*, then added together to form the actual number, like this:

```
937.5 = 9 * 100 + 3 * 10 + 7 * 1 + 5 * 0.1
```

The unit's place is always immediately left of the decimal point. The unit's place is the "middle" of the number. The digit in the unit's place is not multiplied by any place value. The digits to the left of the unit's place are multiplied by increasing powers of the base of the number system. The digits to the right of the unit's place are multiplied by decreasing powers of the base of the number system. A digit's value is determined by the product of itself and the base of the number system raised to the power corresponding to the digit's place.

Digit:	9	3	7	5	
Base to the Power:	10^2	10^1	10^0	10^{-1}	
Place Value:	100	10	1	0.1	
Digit's Value:	900	30	7	0.5	= 937.5

The same procedure holds in number systems with other bases, only with increasing and decreasing powers of the base for that number system instead of ten. The digits used in each base's number system are always less than the base itself.

```
1100 0101 = 1*128 + 1*64 + 0*32 + 0*16 + 0*8 + 1*4 + 0*2 + 1*1
```

Again we have increasing powers of the base, but now the base is two. Base 2 is the *binary number system* and is used in all our digital electronics. We can convert to base 10 by summing the products of the digit and place value. Since there are only two digits (zero and one) and since zeros destroy products and ones don't affect them, we can write:

```
1100 0101 = 128 + 64 + 4 + 1
```

So $1100\ 0101_2 = 197_{10}$. The subscript indicates the base of the number system. In binary we often separate out the digits of a number in groups of four, similar to using commas for large numbers in base ten. Each binary digit is called a *bit*. Eight bits are called a *byte*.

When working with binary numbers it helps to know the powers of two. It is useful to memorize that $2^8 = 256$.

Digit:	1	1	0	0	0	1	0	1	
Base to the Power:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Place Value:	128	64	32	16	8	4	2	1	
Digit's Value:	128	64	0	0	0	4	0	1	= 197

Computer programmers also have to know base-8 (octal) and base-16 (hexadecimal) number systems.

Lab10extension
Binary to Decimal

Objective
Convert binary to decimal numbers.

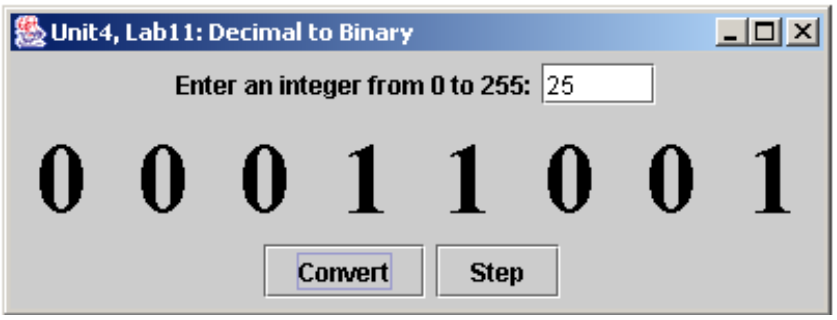
Specification
This lab is an extension of Lab10. Return to Unit4\Lab10\Display10.java. Modify Display10.java by adding an independent JLabel to the gridLayout. Write a private void method `convert` that converts the binary number to a decimal number. Every time the binary number changes, call `convert` and set the text on the label.

The `convert` method should make a temporary array of integers. Then it processes the array with a loop. Starting at the right, multiply the digit times 2-raised-to-a-power. Add that to the running total. Each time around the loop, increase the power by 1.

Sample Run `Unit4Lab10ext.jar`

Lab11
Decimal to Binary

Specification
In Lab11, the user enters an integer in the text box. Then you convert that integer number to binary. There are several algorithms, but here is a good one: modulus the integer by 2 and set the result in the rightmost label. Update the integer by integer division by 2. Repeat 8 times.



The Convert button outputs the binary representation of the number in the textfield using the algorithm above. The Step button increases both the textfield number and the binary number by 1. However, if the eight binary digits have an *overflow*, at 1111 1111 plus 1, then it "wraps around" to 0.

Specification
Open Unit4\Lab11\Panel11.java. The south sub-panel is done for you. You will make a north sub-panel with a label and a textfield. You will instantiate a display and put it in the center. Listener1 is done, but you must do Listener2, which increments the integer (but wraps around to 0 upon an overflow) and also calls `setValue`.

Open Unit4\Lab11\Display11.java. Implement the methods `setValue` and `setBlank`. The method `setBlank` is called twice, once to start things off, and again if the user enters bad data.

Open Unit4\Lab11\Driver11.java. Compile and run.

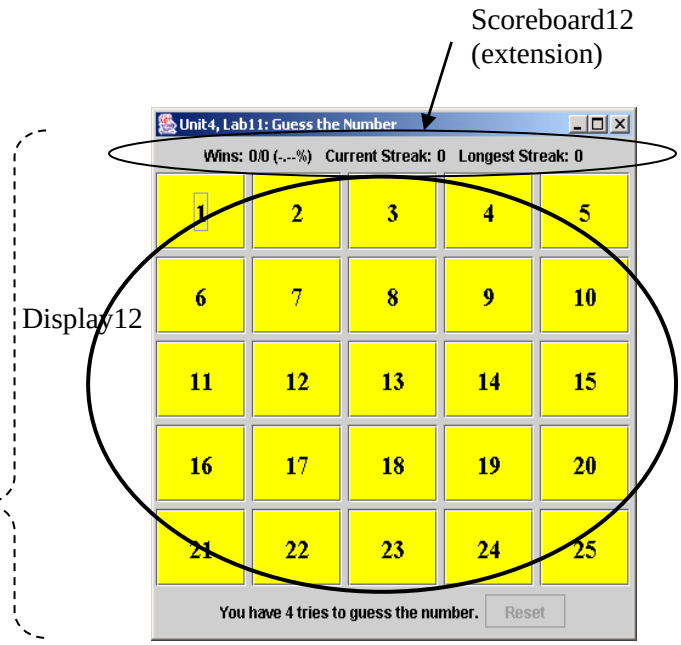
Sample Run
`Unit4Lab11.jar`

Lab12
Guess the Number

Objective
To program a game that uses an array of buttons.

Background
The game is played as follows:

1. Click on a number from 1 to 25.
2. The application will tell you one of three things:
   a. Congratulations, you picked the right number.
   b. Sorry, your guess was too high.
   c. Sorry, your guess was too low.
3. Repeat until either:
   a. the player picks the target or
   b. the player misses 4 times.



Be careful! The array of buttons is displayed in a grid layout. This causes the array cells to wrap when they get to the end of each row. Do not be deceived. The array is linear. It has only one dimension. The rows and columns you see on the screen are only significant to the GUI display, not the logic of the game.

Specification
Open Unit4\Lab12\Panel12.java. Panel12 isa JPanel and hasa Display12. Panel12 has one listener and makes the decisions. If the guess is too low, too high, or on target, Panel12 calls the appropriate method, and passes the appropriate argument, in the display object. Panel12 also reports the number of tries left. Finally, Panel 12 has a reset button.

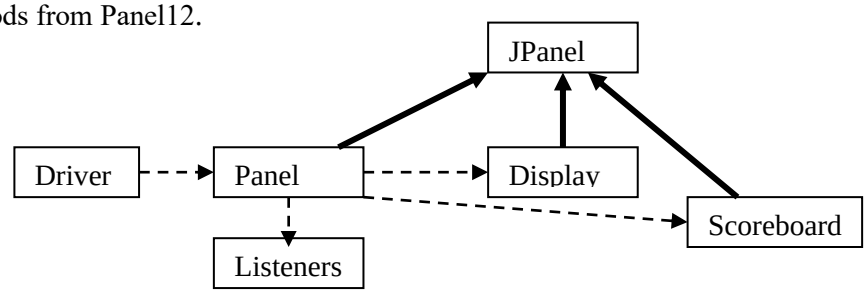
Open Unit4\Lab12\Display12.java. Display12 isa JPanel with 25 buttons in one array. Display12 changes the background colors, and enables the buttons, depending on what Panel12 communicates to it. (Note to Mac users: you must do `setOpaque(true)` on each button to get the colors to work.)

Open Unit4\Lab12\Driver12.java. Compile and run.

Sample Run
`Unit4Lab12.jar`. Play the demo. Who is in charge of the listener? Who is in charge of the buttons?

Extension
Let the user play the game as many times as he or she wants. Keep track of the wins, the games played, the percentage won, the streak of wins, and the longest streak. Add a scoreboard object to the northern region of Panel12. Call the scoreboard's methods from Panel12.

Here is the (partial) class diagram. Not shown are about 15 other classes including JButton, Color, Font, JLabel, String, 3 layout managers, JFrame, DecimalFormat, etc.



Discussion
Two-Dimensional Arrays (Matrices)

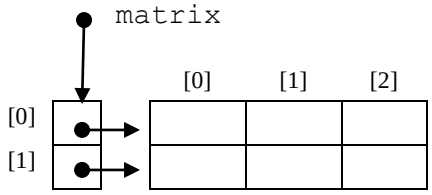
The arrays you have worked with so far have been one-dimensional arrays. To review:

```
double[] array1 = new double[10];
boolean[] array2 = {true, true, false, true, false, false, false};
String[] array3 = {"dog", "cat", "eat", "pie"};
Foo[] array4 = new Foo[100];
for(int k = 0; k < array4.length; k++)
   array4[k] = new Foo();
```

The square brackets `[]` indicate that you are creating an array of doubles, booleans, Strings, or Foos rather than a single double, boolean, String, or Foo. The array itself is an object and is instantiated with the `new` keyword, unless you use the short-cut { } notation. The indices of the array are always numbered with integers starting from zero. The public field `length` returns the number of cells in the array. The square brackets are used to access the contents of individual cells. Creating an array of objects is a two-step process: first create the array, then create the objects to fill the array. The same array cannot store objects of different types, unless the objects are subclasses of the declared type. The declared type might even be an abstract type (like Shape) or an interface.

A two-dimensional array is a *matrix*. In Java, a matrix is an array of arrays. For instance:

```
double[][] matrix = new double[2][3];
```



This declaration creates a matrix named `matrix` with two rows and three columns. Of course the name of a two-dimensional array does not have to be `matrix`. Always indicate the rows first, then the columns.

To access any one cell you must specify both the row and the column. Some examples:

```
matrix[0][1] = 3.7;

for(int r = 0; r < matrix.length; r++)      //assigns 1.0 to each
   matrix[r][0] = 1.0;                      //cell in col 0.

for(int c = 0; c < matrix[0].length; c++)   //assigns 2.0 to each
   matrix[1][c] = 2.0;                      //cell in row 1

for(int r = 0; r < matrix.length; r++)         //sets all values
   for(int c = 0; c < matrix[0].length; c++)   //in the matrix
      matrix[r][c] = 0.0;                      //to 0.0
```

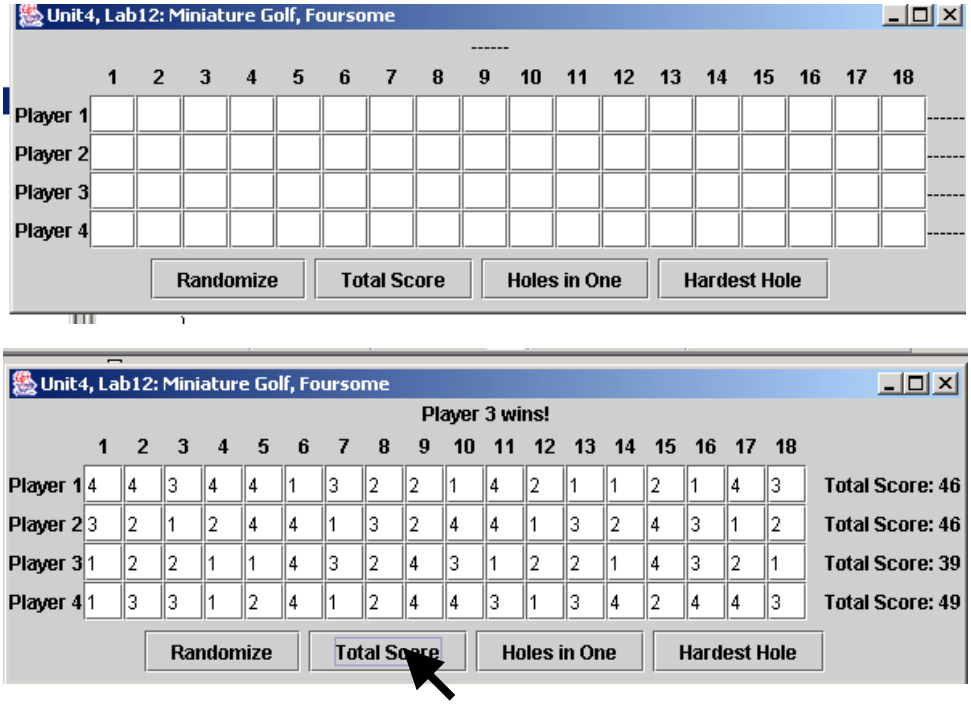
The number of rows is indicated by `matrix.length`. In the example above, `matrix.length` returns 2. The number of columns is `matrix[0].length`, which returns 3 in the example above. To fill one column, fix the column value and loop over the rows. To fill one row, fix the row value and loop over the columns. To fill the entire matrix, loop over both the rows and the columns.

Be careful! Always specify the row first and then the column. You can think of RC Cola, which stands for Royal Crown Cola everywhere else but in Computer Science it means **R**ow first, then **C**olumn.

Lab13
Miniature Golf, Foursome

Objective
To process 2-dimensional arrays.

Background
You solved the miniature golf problem in Lab09 for one player only, and used a one-dimensional array to keep track of his or her score.



In Lab13 you have a foursome. A foursome needs a two-dimensional array to keep track of all four players' scores over eighteen holes of golf. Player one gets all the scores in row zero, player two gets all the scores in row one, and so on.

In golf, the winning player is the one with the lowest total score.

Be careful! Both the rows and the columns of a matrix are zero indexed. The upper-left corner of the matrix is position (0, 0).

Specification
As usual, we have a driver, a panel, and a display (called the scorecard).

Open and look at Unit4\Lab13\Driver13.java. The driver hasa Panel13.

Open and look at Unit4\Lab13\Panel13.java. Panel13 implements the buttons, the listeners, and all the labels. Panel13 also hasa ScoreCard.

Open Unit4\Lab13\ScoreCard13.java. Complete the implementation of five methods:

- `randomize`, which loops over both rows and columns and generates a random integer for each.
- `findTotal(int row)`, which finds the sum of the scores in the specified row (in the argument).
- `findAces(int row)`, which counts the number of 1's in the specified row.
- `findHardestHole(int row)`, which returns the index number of the maximum value in the specified row. Hint: for this row, find the maximum value and store its index.
- `findHardestHole()`, which returns the index number of the column which has the maximum sum. Hint: for each column, sum the rows. Keep track of the maximum sum so far and store its index.

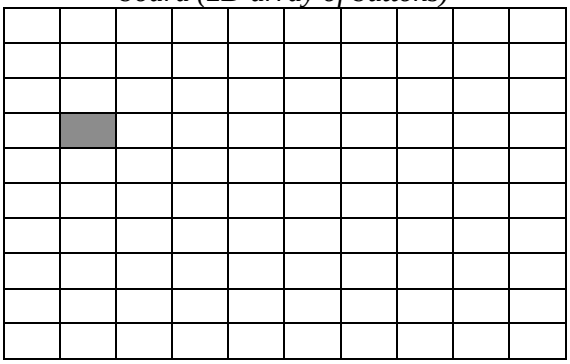
Sample Run
`Unit4Lab13.jar`

Lab14
Battleship

Objective
To use two matrices.

Background
This version has two matrices, one that displays the buttons and an invisible one that stores the location of the battleship. If you click on the location below, should it turn red (a hit), or white (a miss)?

board (2D array of buttons)



matrix (2D array of integers)

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Specification
Open Unit4\Lab14\Battleship.java. Complete the implementation of the three methods below. (People coding on Mac computers need to set each button to opaque.)

- `placeShip`: Simulate flipping a coin to decide whether the ship will be oriented vertically or horizontally. To flip a coin use `int coin = (int)(Math.random() * 2);`. Then pick a random position on the board to represent the starting point for the ship. If the ship is vertical then this random point represents the top of the ship; if horizontal then the left-most part of the ship. The ship is four blocks long so make sure you don't go out of bounds. Store the ship in the integer matrix by changing the zeros to ones. This will be used in Handler1's `actionPerformed` to tell whether a particular shot was a hit or a miss.

- `actionPerformed(Handler1)`: This is where most of the action happens. Since the user has pressed a button, subtract one from the number of torpedoes. The handler object accepts two arguments that tell it the button's location on the board. The handler object uses its private `int myRow` and `myCol` and the integer matrix to determine whether this shot was a hit or a miss. Whatever the outcome, the background on the appropriate button on the board needs to change color. The user wins by getting four hits on the ship. The user loses by running out of torpedoes.

- `actionPerformed(Handler2)`: When the user presses the Reset button, put everything back to its original state. Get ready to play the game again!

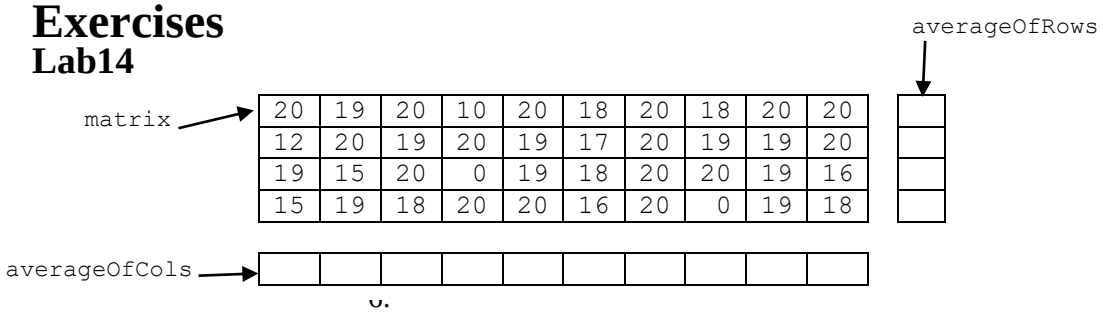
Open Unit4\Lab14\BattleshipDriver.java. Compile and run.

Sample Run
`Unit4Lab14.jar`. Play the demo and ask yourself which object is in charge of doing what.

Extensions
1. Make a BattleshipBoolean that uses a matrix of booleans, not of `int`s. Get the application to work.
2. How hard would it be to add a scoreboard? If you encapsulated Lab12's scoreboard, it should be fairly easy.

Exercises

Lab14



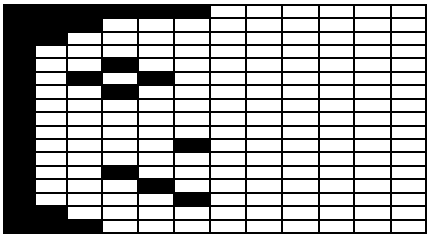
1. Given a 2-D array of integers, write code that calculates the average of each row and puts each average in the vertical array of doubles.

2. Given a matrix of integers, write code that calculates the average of each column and puts each average in the horizontal array of doubles.

2	1	1	1	1
3	2	1	1	1
3	3	2	1	1
3	3	3	2	1
3	3	3	3	2

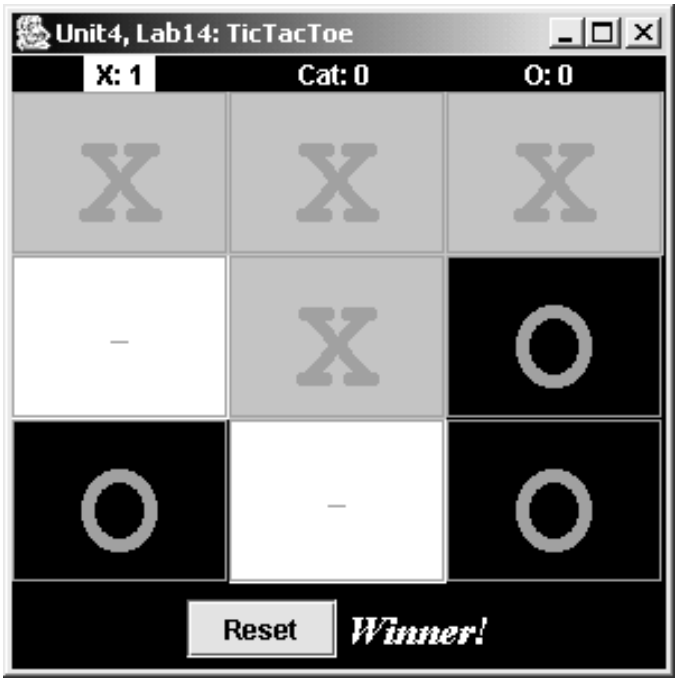
3. There is a pattern in this square matrix. What is the pattern? Write the code to instantiate and fill any square matrix with the same pattern.

4. Here is the left half of a black-and-white 2-D array of pixels. Write the code so that the right side is a mirror image of the left side. Algorithm: for each colored pixel on the left side, get that color and set it in the corresponding position on the right side.



Lab15
Tic-Tac-Toe

Objective
To use a matrix.



Background
The game is played as follows:

1. Pick a location on a 3 x 3 board
2. One of three things will happen:
   A. Congratulations, you have won the game.
   B. Sorry, the game is over in a tie.
   C. The game is not over.
3. Alternate turns with the other player.
4. Pick a location that has not been picked.

The game is played until one of two things happens: either one player gets three of their pieces in a row (win) or no one gets three in a row and we have clicked the buttons 9 times (tie). Players can win with three in a row horizontally, vertically, or diagonally. Let the users play the game as many times as they want and keep track of each player's total wins and the ties (cat's game).

Specification
Open Unit4\Lab15\Panel15.java. Panel15 hasa Gameboard15 and a Scoreboard15. Panel15 has a listener that asks the gameboard if there is a win or a tie. The panel then sends messages to the gameboard and the scoreboard, as appropriate. The panel also has code for the Reset button and the "Winner!" label.

Open Unit4\Lab15\Gameboard15.java. The gameboard has 9 tic-tac-toe buttons. Each button has two listeners, one for the panel and one for itself! It has a boolean `winner` method to check for a winner, a count-up-to-9 boolean method `filled`, and a `freeze` method, which dis-enables all the buttons at the end of a game. Gameboard also has the `reset` method.

Open Unit4\Lab15\Scoreboard15.java. The scoreboard counts and displays the wins for X and O and the ties (the cat's games.) Notice also that the scoreboard above has a yellow background behind the X, showing that it is X's turn. Therefore, the scoreboard has a `toggle` method that moves that yellow background to the current player.

Open Unit4\Lab15\Driver15.java. Compile and run.

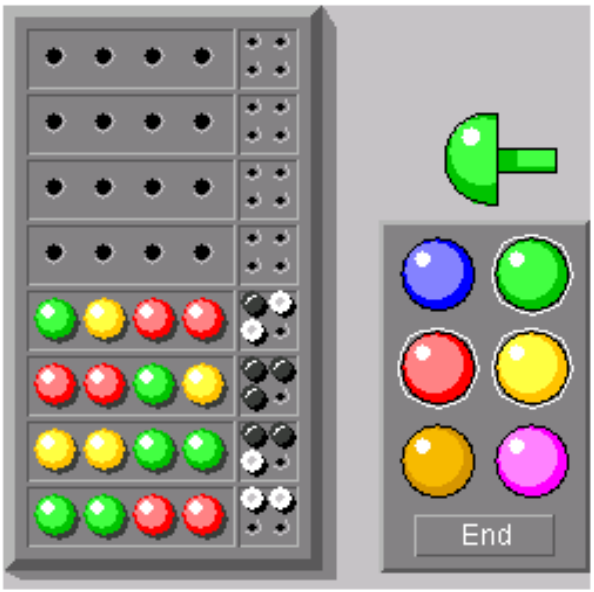
Extension
Have the person play against the computer. Let the computer always play the 0. Make the computer play an intelligent tic-tac-toe game.

Sample Run
`Unit4Lab15.jar`

Lab16
Mastermind

Objective
To use arrays.

Background
The game is played as follows:



1. The computer selects a random sequence of four colors. Order matters, but colors may be repeated. The possible colors are red, blue, green, yellow, orange, and purple.

2. The player guesses a sequence of four colors.

3. The computer informs the player as to how many colors were guessed correctly in the correct location (black pegs) and how many colors were guessed correctly but in the wrong location (white pegs). If a color is repeated in the answer and a player includes that color in part of his or her guess, the color may be counted only once in the information provided by the computer.

4. The player then makes another guess.

The game is played until one of two things happens: either the player guesses the answer pattern (win) or the player does not guess the answer pattern and runs out of guesses (lose). Players get 8 guesses before the game ends. Let the player play the game as many times as he or she wants and keep track of the wins and losses over time.

Specification
Make a Mastermind game.

Sample Run
`Unit4Lab16.jar`