Introduction to Arrays

When writing programs, we have to store our data in variables. Until now, we have just declared a variable to hold each piece of information. For example, in the family information program, we needed to store three names, three ages and three favorite colors, so we declared 9 variables of the appropriate types and set about filling them with the data of the program. That’s fine if we know how much data there will be before writing our program and if that amount is reasonably small. However, most of the time, our programs must process either large amounts of data which render the above approach infeasible, or an amount of data which is unknown at the time the program is written. In these cases, we need a way to store and access variable amounts of data. The answer to this problem is arrays.

ArrayBasicsDemo – in class

This will take you through the basics of declaring, creating, initializing and accessing the elements of an array.

Name Age Favorite Color table exercise – in class

This program is basically the same as the one that read in three names, ages and favorite colors except that it prompts the user for how many people to get information. Since you don’t know until run time how many people there will be, you must use arrays to store the information. Here’s the output (some omitted for brevity):

Enter the number of family members:

5

Enter person 1's name:

Dave

Enter Dave's age:

38

Enter Dave's favorite color:

green

Enter person 2's name:

Steph

Enter Steph's age:

38

Enter Steph's favorite color:

blue

. . .

Name Age Favorite Color

===============================================================

Dave 38 green

Steph 38 blue

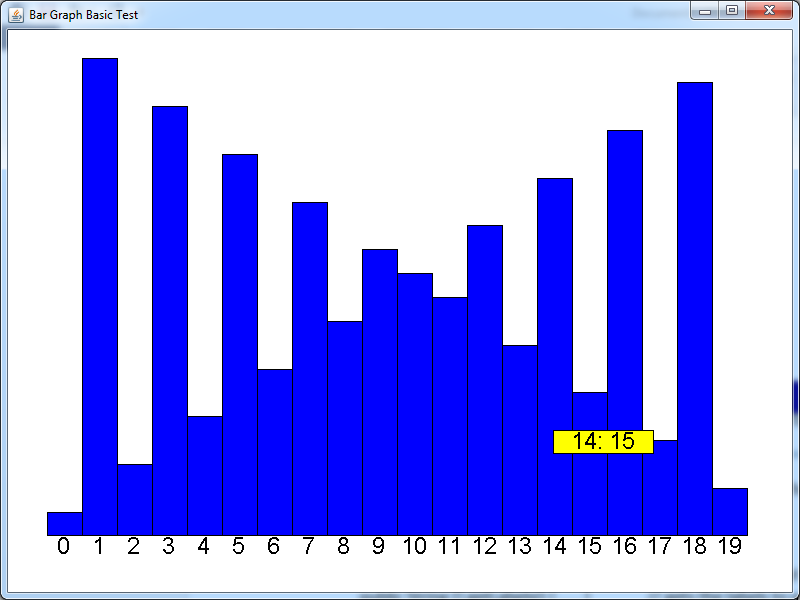
Megan 4 pink

Jon 63 chartreuse

Helen 60 salmon

BarGraphView and BasicBarGraphTest – at home

In this exercise, you will write and test a method which draws a bar graph. The window will look like this:



The paint method for BarGraphView consumes a BarGraph object. The BarGraph object provides two functions which return arrays:

public class BarGraph {

public int [] getBars() { . . . } // gets the heights of the bars

public String [] getLabels() { . . . } // gets the labels to put at the bottom of the graph

}

The paint method should obtain these two arrays from the BarGraph object then draw the bars and the labels below each bar. The total width of the bar graph should be 90% of the width of the window. The total height should be 90% of the height of the window, with the labels taking up 5% and the tallest bar taking up 85% of the height. The other bars should be scaled with respect to the tallest bar according to their heights. The yellow box contains the label of the bar that the mouse is hovering over and the height of the bar, and is located with its ULC at the location of the mouse.

There are two other classes that will help you with this task. The GraphicsUtilityFunctions class provides a couple of useful functions for drawing text to the screen. The Mouse class provides functions to get the current location of the mouse cursor.

public class GraphicsUtilityFunctions {

public static Font getFont( int h ) { . . . } // gets a Font object with height h (in pixels)

// draws a String in a given font centered in a given rectangle

public static void drawStringWithFontInRectangle(

Graphics g, // The graphics object

String s, // the text to be displayed

Font f, // the font to display the text in

int ULCx, int ULCy, // the upper left corner of the bounding box

int width, int height // the width and height of the bounding box

) { . . . }

}

public class Mouse {

public static int getX() { . . . } // returns the x coordinate of the mouse cursor

public static int getY() { . . . } // returns the y coordinate of the mouse cursor

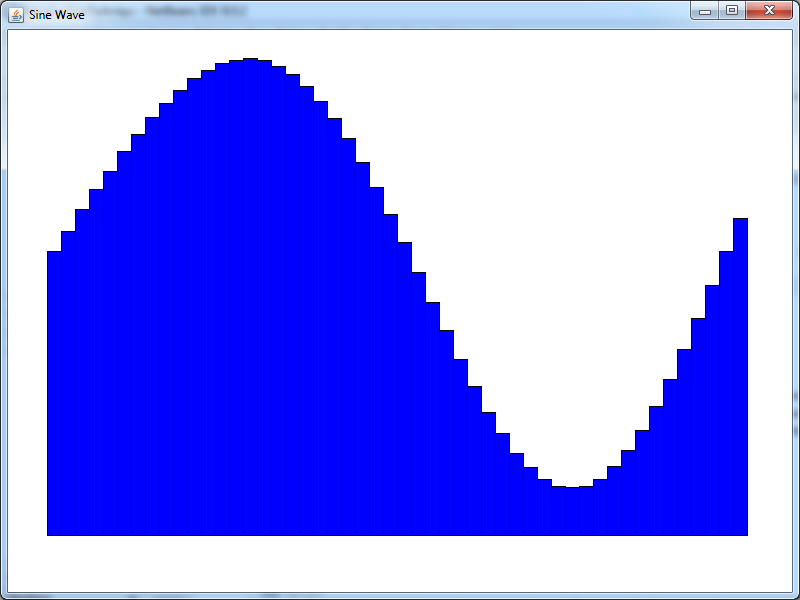
}

More detailed instructions are in the paint function comments.

To test the paint function, complete the function runBasicBarGraphTest. You need to create two arrays, one of ints and one of Strings. Detailed instructions are in the comments.

SinBarGraphController – at home

In this exercise, you’ll implement several methods to create this view:



(when done properly, the wave will be animated and oscillate smoothly).

First, you’ll need to implement the handleTimePassage method in class SinBarGraphController. This method consumes a BarGraph object and a time parameter dt which indicates how many milliseconds have passed since the last update. The variable t keeps track of the current time. Each time handleTimePassage is called, dt is added to t. (Don’t worry about this, it’s taken care of). Your job is to obtain the bars array from the BarGraph object and update it so that the ith bar is:



Where n is the number of bars. The Math class provides a sin function and the constant PI:

public class Math {

public static double sin( double theta ) { . . . }

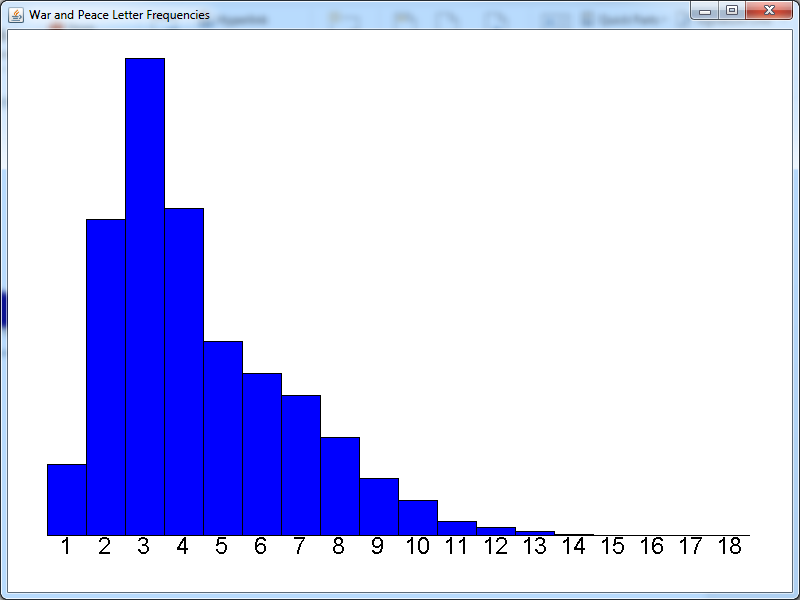
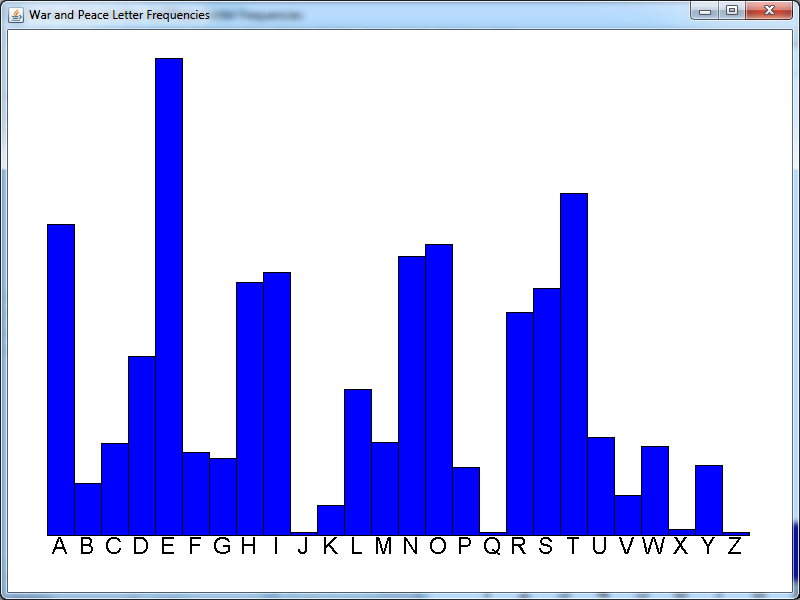
public static final double PI = 3.14159 . . . ;

}

When you are done filling out this array, you need to complete function runSinBarGraph. You’ll create the initial bars array (fill it with 1s) and the labels array (for this exercise, the labels should just be empty strings).

War and Peace Letter Frequency and Word Length Frequency – at home

In these exercises, you’ll create bar graphs of the letter and word length frequencies in War and Peace. Here are the final results:



In the function runWarAndPeaceLetterFrequency you will:

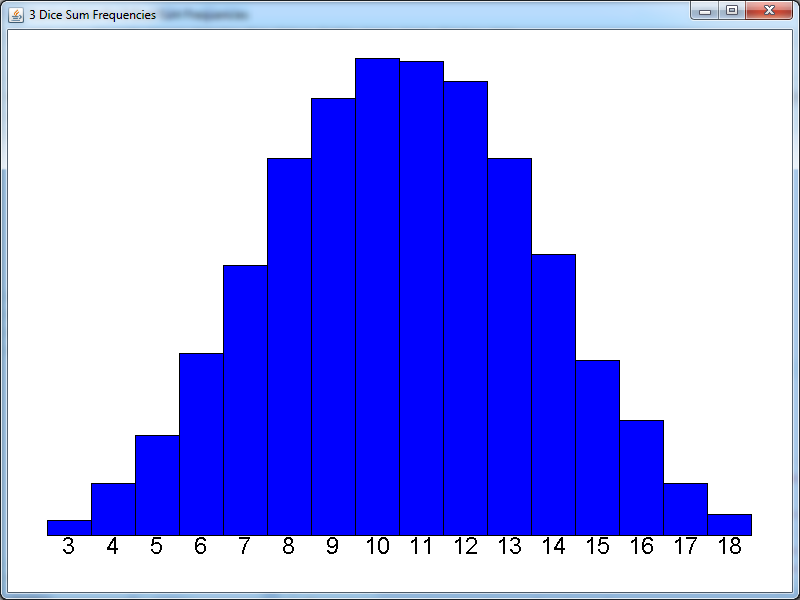
1. Create an array of ints. Each entry in the array will hold the count of a different letter.
2. Create a Scanner to read from the file “War and Peace.txt”
3. Read words from the file as long as the scanner has more data.
4. For each word, scan through the characters of the word. If the letter is an A (or a), increase the bar in index 0 of your bars array by 1, if the letter is a B increase bar 1 etc.
5. Create an array of labels.

In the function runWarAndPeaceWordLengthFrequency you will:

1. Create an Scanner to read from the file again
2. Read all the words, keeping track of the longest word.
3. Create an array which will hold the frequencies of all the different word lengths
4. Re-create the scanner from the file so that you can re-read the words
5. For each word, bump up the appropriate bar in your array.
6. Create an array of labels.

Dice Distribution – at home

In this exercise, you will simulate rolling 3 dice thousands of times and keeping track of how often each sum (from 3 to 18) is achieved.

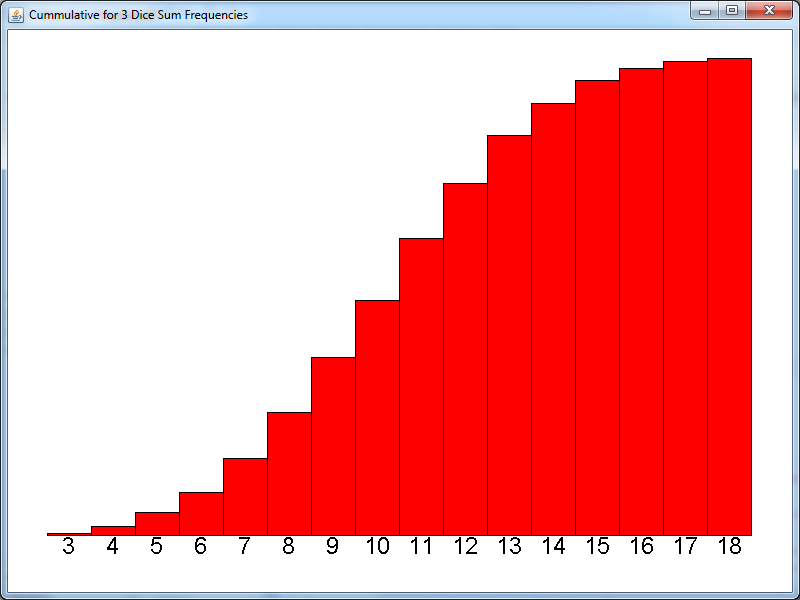
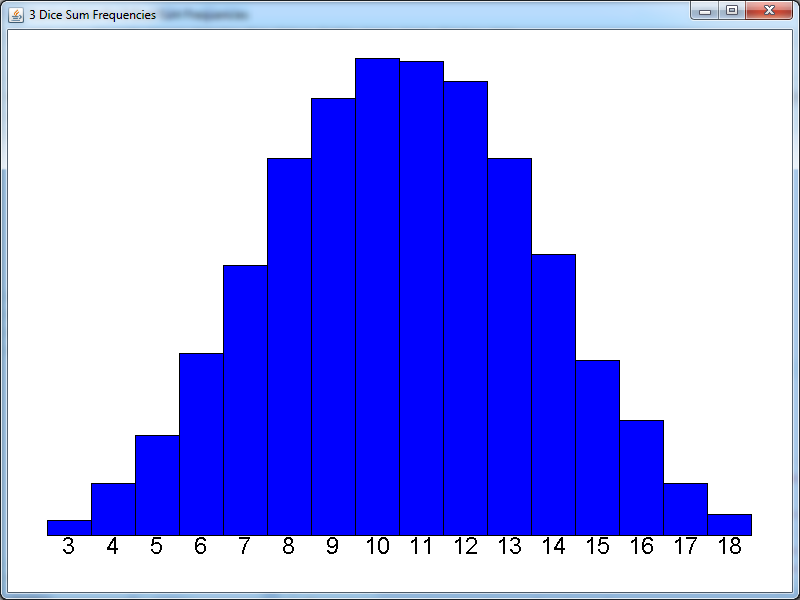


In function runDiceDistribution you will:

1. Create an array with 16 ints to store the bars.
2. In a loop that runs 10000 times, use the random function to generate 3 random numbers between 1 and 6 inclusive.
3. Add these three numbers together.
4. Bump up the bar which corresponds to the sum.
5. Create an array of labels.

BarGraphCummulativeView

In the last few exercises, you created various bar graph arrays. In this exercise, you will create a cumulative bar graph. In a typical frequency plot, each bar keeps track of the frequency of a particular event happening. In a cumulative plot, each bar, from left to right, keeps track of the cumulative totals of the bars in a typical frequency plot. For example, in the dice rolling exercise, the frequency plot looks like this:



The first three bars are 43, 166, and 279. In the cumulative plot (shown to the right), the ith bar contains the sum of the first i bars in the frequency plot. That is, the first bar is 43. The second bar is 209 (43 + 166). The third bar is 488 (43 + 166 + 279) etc.

In class BarGraphCummulativeView, the paint method consumes a BarGraph. Your job is to create a new BarGraph which is the cumulative graph for the given BarGraph. You should:

1. Get the bars from the given BarGraph
2. Create a new array of bars
3. Set the ith bar equal to the sum of the first i bars from the given graph.
4. Create a new BarGraph from the new bars and the labels from the given BarGraph
5. Create a new BarGraphView and call its paint method with the new BarGraph.

To test this out, uncomment the lines in each runXXX function that say “uncomment this line when you have finished the cumulative exercise”.