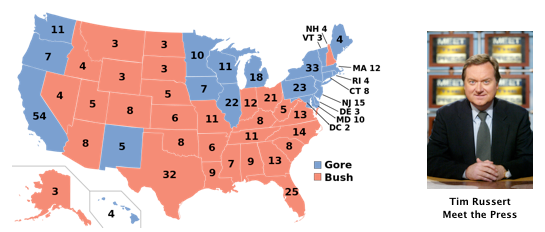
Assignment 6 – (Purple) America

### Overview

This is a variation on a “Nifty” Assignment – the original information is here: <http://nifty.stanford.edu/2014/wayne-purple-america/purple-america.html>

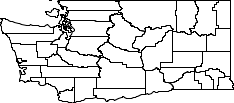
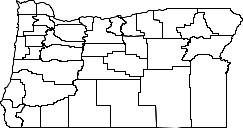
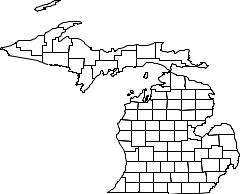
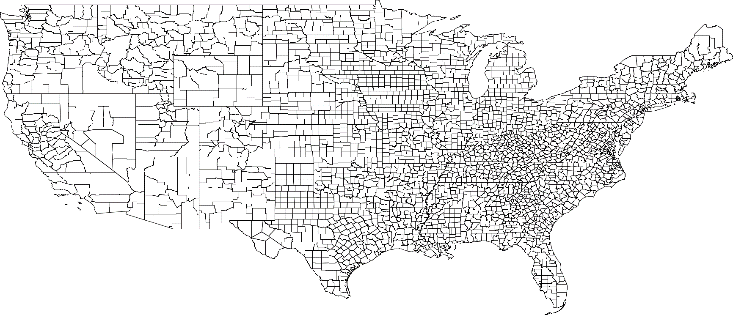
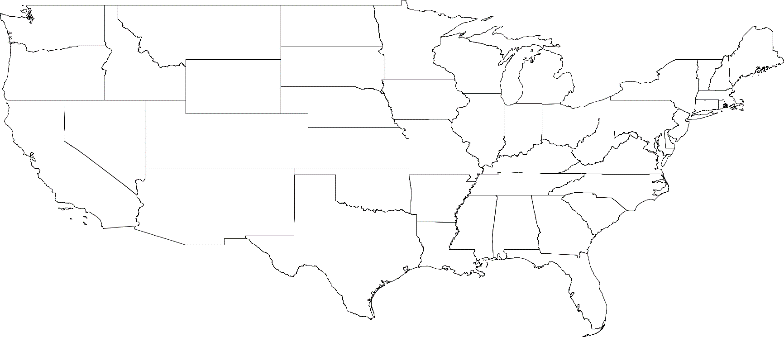
**Historical context.** During coverage of the 2000 presidential election, Tim Russert coined the political terms red states and blue states to refer to states that predominantly vote for the Republican presidential candidate (red) or the Democratic presidential candidate (blue). The news media use red-state blue-state maps, such as the one below, to display election results.



The way he presented this data was rather polarizing. The states were “red” or “blue” based on had more votes in that state and won the electoral college votes. However, if you shaded each state based off a percentage of red or blue based on the percentage of the vote, you got a much different picture:

Purple America: 2000 Presidential Election – From <http://nifty.stanford.edu/2014/wayne-purple-america/PurpleAmerica.pdf>.

The goal of this assignment is to read information from a file and draw it on the screen in two parts. For the first part you will read in the files that contain the detailed polygonal shapes that define the outlines of the states. In the second, challenge, part of the project, you can color these outlines based on election results.

The basic part of program will first print some instructions to the screen, then ask the user which state they would like to see drawn. The user will enter the state as its two letter abbreviation. If the abbreviation entered is valid, it will bring up a window and draw the outline of each of the counties in that state.

This program draws a visualization of

data from a given file. Enter

- the 2 letter abbreviation for each state

- USA for all of the US by state, or

- USA-county for all of the US by county.

What state would you like to see? WA

Do you want to draw something else? y

What state would you like to see? OR

Do you want to draw something else? yuppers

What state would you like to see? MI

Do you want to draw something else? yesireeeeee

What state would you like to see? USA

Do you want to draw something else? YES!

What state would you like to see? USA-county

Do you want to draw something else? nope

If the user enters something other than a correct two letter abbreviation for a state, the program will print an error and reprompt for the state again.

What state would you like to see? Utah

File not found. Try again.

What state would you like to see? UT

Do you want to draw something else? n

NOTE: that if the user enters the abbreviation as two lower case letters, this will work fine on a Windows machine. This will NOT work on a Mac or Unix based machine. Windows file systems are not case sensitive, but Mac/Unix are. This will be considered correct behavior for the purposes of this project.

What state would you like to see? mi  
Do you want to draw something else? n

If the user requests USA, the entire continental United States will be shown with the outlines of the states, and USA-county will show the continental US outlined by county.

You will be supplied geographic data (sourced from the [U.S. Census](http://www.census.gov/tiger/boundary)) that describes the boundary of each state and county in the United States. These files will be in the PurpleAmerica.zip file located on the course website that when extracted will be in a data subdirectory.

* The first two lines consists of four real numbers, representing the bounding box of the region. The first line contains two numbers are the minimum longitude and latitude values; the second line contains two numbers are the maximum longitude and latitude values.
* The next line is an integer that specifies the number of subregions.
* There is one block for each subregion (with a blank line separates blocks):
  + The first line of a block is a string that is the name of the subregion.
  + The second line of a block is a string that is the name of the region.
  + The third line of a block is an integer N that specifies the number of points in the polygon describing the subregion.
  + The remaining N lines of the block describe the polygonal boundary, given as N pairs of real numbers, representing the longitude and latitude coordinates.

Examples of three text files are shown in the table below. We note that the number of subregions in USA.txt is not 50 for two reasons: first, we do not include either Alaska or Hawaii; second, we include an entry for each polygonal subregion—some states (such as Michigan, Florida, and California) comprise several polygonal subregions.

|  |  |  |
| --- | --- | --- |
| **USA.txt** | **NJ.txt** | **USA-county.txt** |
| -124.731216 24.544102  -66.980385 49.384365 104  Alabama USA 498  -88.200027 34.995548  -88.202919 35.007942  -87.984886 35.005848 ...  -88.153519 34.921185  -88.176064 34.962433  -88.187088 34.974182  ...  Wyoming USA 68 -111.048203 44.474144 -111.054558 44.666336 -111.054420 45.001392  ... -111.043846 43.315800 -111.044724 43.501213 -111.046272 43.983456 | -75.560143 38.928589  -73.894402 41.357330 21  Atlantic NJ 127  -74.877563 39.608414  -74.736694 39.729721  -74.676102 39.691162  ...  -74.857353 39.420528  -74.856087 39.424465  -74.985443 39.514725  ...  Warren NJ 121  -75.120819 40.968208  -75.122986 40.970055  -75.131744 40.969185  ...  -75.095901 40.924057  -75.112061 40.948017  -75.118141 40.952927 | % **more** [**USA-county.txt**](http://nifty.stanford.edu/2014/wayne-purple-america/data/USA-county.txt) -124.731216 24.544102  -66.980385 49.384365 3206  Autauga AL 118  -86.916969 32.664028  -86.816589 32.659988  -86.713409 32.661602  ...  -86.916809 32.649662  -86.917458 32.653877  -86.921387 32.655415  ...  Weston WY 11 -105.078743 44.176205 -104.375000 44.181641 -104.054001 44.180401  ... -105.081238 43.592144 -105.078255 43.827049 -105.080872 43.826954 |

### Learning Objectives

* Practice previous concepts - procedural decomposition, variables, datatypes, etc, no magic numbers, good naming, loops, scoping, parameters, return values, using objects, fencepost loops, cumulative sum/product or min/max loops, roundoff errors, if/else statements/nested if/else statements, printf/format), Java doc with pre and post conditions where needed, while/do-while loops, Random objects, boolean operations, assertions
* Demonstrate the ability to read from a file
* Demonstrate the ability to write to a file.
* Demonstrate the correct use of try/catch blocks
* Demonstrate understanding of line and token based processing
* Continue to demonstrate programming for robust user interfaces

### External Requirements

Part of your program's score will come from its "external correctness." External correctness measures whether the output matches exactly what is expected. We are very picky about the output matching exactly. Every character and space must match, as well as the states drawn on the screen Use the **output comparison tool** to ensure that your text output matches exactly, and use the DrawingPanel’s **File->Compare To File…** menu option to compare the visual output. Programs that do not compile will receive no external correctness points. The specific requirements for external correctness are detailed below.

* You are to exactly reproduce the appropriate output based on the user’s input.   
  There are three main sections to the output:
  + The introduction
  + The request for a state abbreviation
  + Showing the outline of the state
* Your program should output the correct results for any user input, valid or invalid.
* Your program should NOT throw any exceptions, particularly InputMismatchException, or FileIOExceptions

### Internal Requirements

Internal correctness means that your program uses the programming elements and structure that are detailed in the list of requirements below.

* Write a Java class named Outline in a file named Outline.java. If you choose to do the challenge to color the states according to the votes, turn that file as a Java class called Purple in a file named Purple.java. Use exactly this file name, including identical capitalization.
* Include a header – a comment at the top of each .java file that has some basic information and a description of the program in your own words.
* Continue to use static methods so that no substantial groups of identical statements appear in your code.  In other words, avoid redundant code.
* Structure your program so that the methods match the structure of the output itself.
* Your main method should coordinate your other methods. It should not do a huge share of the work.
* You should create one (and only one) Scanner object to get input from the user. This Scanner object can be passed from one method to another, if necessary. You should NOT use JOptionPane, even if you have in the past.
* Please be sure to write all import statements as import java.<whatever>.\*; for use with GradeIt!
* Your program must correctly use parameters and return values.
* Your program must correctly use flow of control structures such as if/else if/else statements, for loops and while loops as well as try/catch statements.
* You must define a **class constant** for the scaling factor for drawing the states or US on the screen. This should be set to 30 for the version you turn in, but it should be able to be changed to any number greater than 1, and the program would work perfectly using that new number. Test your program by changing your constant and running it again to make sure that everything uses the new value.
* You must define a **String class constant** which contains the directory name where the data files are located in. On your machine, you should have all your data files in a data directory, and this class constant should be set to “data\\”. This is required so we can get your programs to function on GradeIt.
* You may assume that the user will type a one-word string as the response to this “Do you want to play again?” question. In other words, use next() (do NOT use nextLine()) to get that input from the user.
* Structure your solution using static methods that accept parameters and return values where appropriate. **For full credit, you must have at least the following two methods other than main in your program** (you may, and likely should, have other methods as well):
  + A method to get a Scanner on the coordinate input file
    - This method should be robust, if a bad filename is passed in it should not throw an exception
  + A method to draw the outline of the region
    - This method will parse through the file and get the coordinates that will create the polygonal shapes of the region.
* You are to reuse Inputter.java if you take on the Purple challenge section of this assignment.
* The PurpleAmerica.zip file will also contain four files you must use as part of this assignment (and a sample for helping you): DrawingPanel.java, DrawingPanelPlus.java, Polygon2D.java, FileData.java and DrawSample.java. We will also include Javadoc for DrawingPanelPlus, Polygon2D and FileData.java.
* DrawingPanelPlus is a drawing library (that happens to be drawing on the DrawingPanel) where all the points must be between 0 and 1 and the coordinate (0, 0) is in the lower left hand corner of the window as we would expect. There are methods that you will need to use to set the X and Y scale of the window. Check the DrawingPanelPlus Javadoc for more information on these methods. Hint: you will need the bounding box of the region in order to do this.
* DrawSample.java has some examples of how to use DrawingPanelPlus. Use this file to experiment with the DrawingPanelPlus to see how drawing and scaling works in this environment.
* For simplicity, draw the point with longitude x and latitude y at location (x, y) in the plane. Use the bounding box of the region to determine the part of the plane to display in the drawing window and rescale the coordinates accordingly.
* You may not use any programming constructs that are not in chapters 1 through 6 of the textbook. In other words, you may not use arrays, recursion or developing your own classes and objects.

### Style Requirements

* Properly indent your code
* Localize variables whenever possible
* Write proper JavaDoc comments for all your class constants and variables, methods with their parameters and return values
* Tricky, non-obvious, and important code must be explained with in-line comments.
* Give meaningful names to methods and variables and follow Java's naming standards (ClassNames, methodAndVariableNames, and CONSTANT\_NAMES).
* Limit the lengths of all lines in your program to fewer than 100 characters.

### Development Recommendations

* Completing the chapter 6 worksheets are crucial, particularly in order to complete Milestone 1.
* You will need to download the input files and associate class files from the web site and save them in the same directory as your program
* A common bug is calling next, nextLine, etc. on the wrong Scanner, so examine those calls carefully.
* Write a small test program to get use to the scaling of the DrawingPanelPlus.

### Milestone 1

This is going to be a little different because we're short on time because of the holiday. By 5pm on the due date listed on the Haiku, I want you to upload the start of your program to GradeIt

* Create the program file and a main method.
* Write three methods
  + Write a method that takes a String parameter (the filename). This method opens the file and attaches a Scanner to that file. Use try/catch to handle the exception. Return the Scanner.
  + Write a method that takes a String parameter (the filename). This method opens a PrintStream on the File created from that filename. Use try/catch to handle the exception. Return the PrintStream.
  + Create a third method that takes both the input Scanner and output PrintStream. This method should read in the input file (line by line) and write it out to the output file, line by line. Bonus: have it split each line using line based processing and write out each token.
* Call these three methods from main, ensuring that your input file name and output file names are different. (You don't want to overwrite your input file)
* You may hard code the location of the files (in other words you don't have to have code to ask the user what file to open). You may use a relative path and can assume your input file and output file are in the same directory as your Java class file.
* You do not need to build in robustness (i.e. reprompting the user until they enter a good file name.)
* Test plan! In a comment block at the bottom of your program I want you to have a test plan that describes the inputs and expected outputs for this program. Remember, the input is not just coming from the user! What if you have a badly formed file? Example test plan will look like this (sample part of the test plan from assignment 5)

Input What will happen

User enters a letter Program will tell the user to

enter a number and reprompt

User enters a number Program will reprompt for a number

out of range in range.

User enters a number Program will respond "higher or

in range lower" depending on the

computer’s random number.

### Milestone 2

For Milestone 2 complete the GuessingGame.java program (and if you choose to do the challenge, also the modification: Purple.java, see the challenge section for more information), meeting all internal and external requirements and turn it into the GradeIt link provided on the course website.

### Grading

You will be graded on your Milestone 1, the program’s "external correctness" (whether the program compiles and produces exactly the expected output), its "internal correctness" (whether your source code follows the stylistic guidelines in this document) as well as its style and documentation.

**Milestone 1:**

1 point -  "Design" as a .java file that reads in a file and splits each line (using line based processing) and writes it back out to a file.

1 point – “Design” includes a test plan as a comment at the bottom of your program file.

**Milestone 2:**

10 points for external correctness

5 points for internal design and efficiency

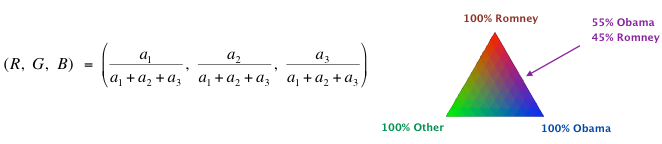
3 points for style and documentation

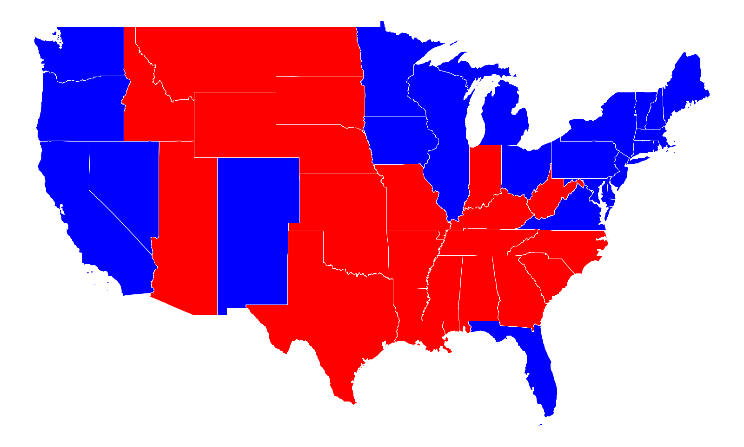
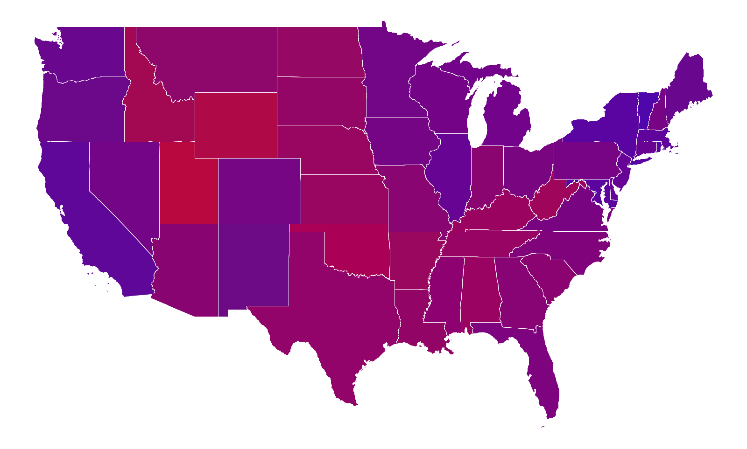
### Challenges

Note: Only attempt the challenges if you are absolutely sure your program matches the expected output 100%. If you do attempt a challenge, please make sure you **comment out, or** **programmatically “turn off”** the additional work so your output matches the expected output exactly.

**Purple:**

In 2000, Bob Vanderbei created a [Purple America](http://en.wikipedia.org/wiki/Purple_America) map, in which each region is colored in a shade of red, green, and blue, according to the proportion of votes received by each candidate. Specifically, if the Republican, Independent, and Democratic candidates receive a1, a2, and a3 votes, respectively, then we draw the subregion using the following formula:





In this challenge you are to write a program Purple.java that starts out the same as Outline.java not only asks for the state, but also asks for the election year from 1960-2012. The program then not only opens up the data file containing the region information, but a second data file that contains the voting information. The file name is in the form of <State Abbreviation><year>.txt. The program will then draw two versions of the region – one where the color is determined solely as a function of the winner, and the other will be based on a more refined visualization reveals that the United States is not as polarized by geography as suggested in the strictly “binary” visualizations.

This program draws a visualization of

data from a given file. Enter

- the 2 letter abbreviation for each state

- USA for all of the US by state, or

- USA-county for all of the US by county.

What state would you like to see? WA

For what year (1960 to 2012) do you want to see data: 2000

Displaying 2000 US Presidential Election data

Red = Bush

Blue = Gore

Green = Nader

Displaying Adams data

Red = 3440

Blue = 1406

Green = 85

Displaying Asotin data

Red = 4909

Blue = 2736

Green = 215

**Other Challenges:**

* Write a program to screen scrape the election return data from [Dave Leip's Atlas of U.S. Presidential Elections](http://www.uselectionatlas.org/). Pay careful attention to name clashes between Dave Leip's site and the U.S. Census (e.g., LaSalle vs. La Salle, Kings County vs. Brooklyn).
* Modify your program to include Hawaii and Alaska.
* Use a [map projection](http://en.wikipedia.org/wiki/Map_projection) (such as Mercator, azimuthal, Albers, or Gall-Peters) to transform longitude and latitude coordinates into points in the plane.
* Explore a different color palette (with 5-7 color categories) for coloring the subregions. Here is one [example](http://cdn.filmschoolrejects.com/images/What-America-Looks-Like-2012-Election-Map-640x454.jpg).
* Write the state name in the appropriate place. For large states, draw it in the centroid of the polygon describing the state.
* Create an interactive GUI which displays the election returns for a county as the user hovers over it. You will probably need to add a method to your polygon data type to determine whether a point is inside the polygon.
* Visualize the gradient or change in votes from one election to the next.
* Visualize a different data set by county, e.g., poverty rate, access to Internet, and average price of health care. Or collect data for elections in another country.