You can find more resources for this course at: <http://www.dukelearntoprogram.com/course2/index.php>

This website of programming resources will be essential to your success in this course. The link above for this course is where you will go to:

* Download our custom version of the BlueJ environment;
* Find project resources, such as example code from the lecture videos;
* Download images and data files for the programming exercises; and
* See documentation for the edu.duke library, and other Javadoc.

**Course Programming Resources**

The **BlueJ Environment** download, documentation, and programming resources are available at:

<http://www.dukelearntoprogram.com/course2/index.php>

(Note that if you have already downloaded a version of BlueJ, you will need to uninstall and download this one to get the edu.duke library.)

BlueJ has provided a [playlist of videos](https://www.youtube.com/watch?v=CPUaTT0Xoo4&list=PLYPWr4ErjcnzWB95MVvlKArO6PIfv1fHd) explaining some key Java terminology and how to use BlueJ.

**Download BlueJ**

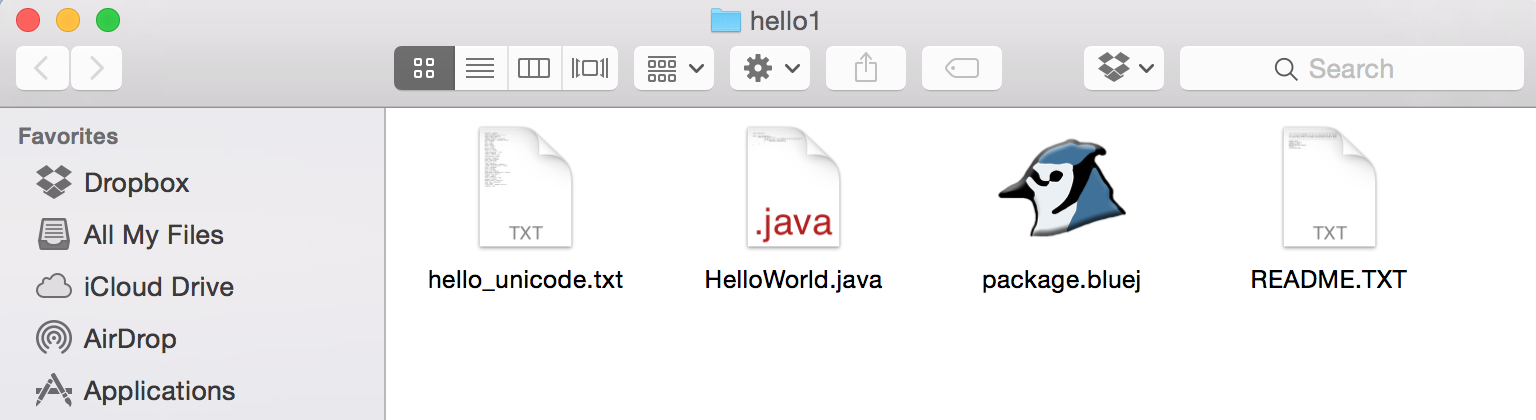
From the above linked DukeLearnToProgram page, click on "Download BlueJ Environment" (<http://www.dukelearntoprogram.com/downloads/bluej.php>). Follow the relevant instructions for your device's operating system (Windows, Mac, or Linux). Note that on a Mac, you may want to move the BlueJ application to your "Applications" folder.

Also, for Mac users, if you get an alert when you try to open BlueJ for the first time that says, "BlueJ” can’t be opened because the identity of the developer cannot be confirmed," simply two-finger or right click on the BlueJ application icon and click "Open". You will still receive the same alert message, but now there will be a button in the alert that will allow you to open BlueJ. BlueJ should open normally without the alert after you open the application this first time.

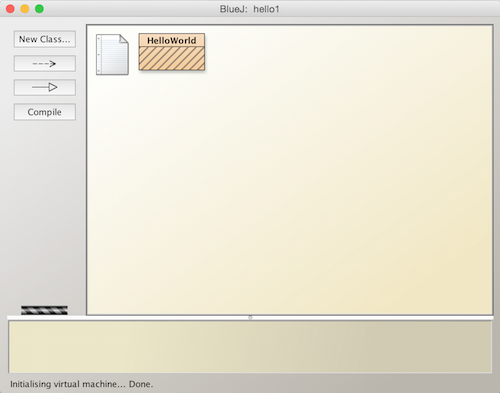
If you intend to follow along with the *"Hello!" around the World* example showcased in the next video, download the project containing the example code from the DukeLearnToProgram Project Resources (<http://www.dukelearntoprogram.com/course2/files.php>, download **"BlueJ Project: HelloWorld class using FileResource**").

**Opening BlueJ Project in Mac**

The HelloWorld project contains these files in a zip folder:

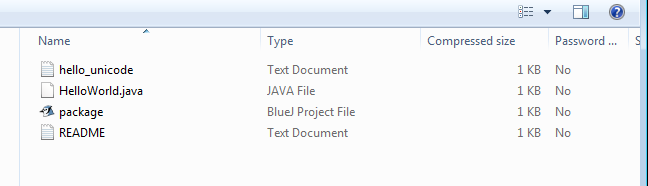


After downloading those files, open the BlueJ application, and from the menu bar, select Project -> Open Project. Select the project folder you have just downloaded, which should now have a BlueJ logo on the folder icon. Click "Open," and you are ready to compile and run the example program along with the next videos!

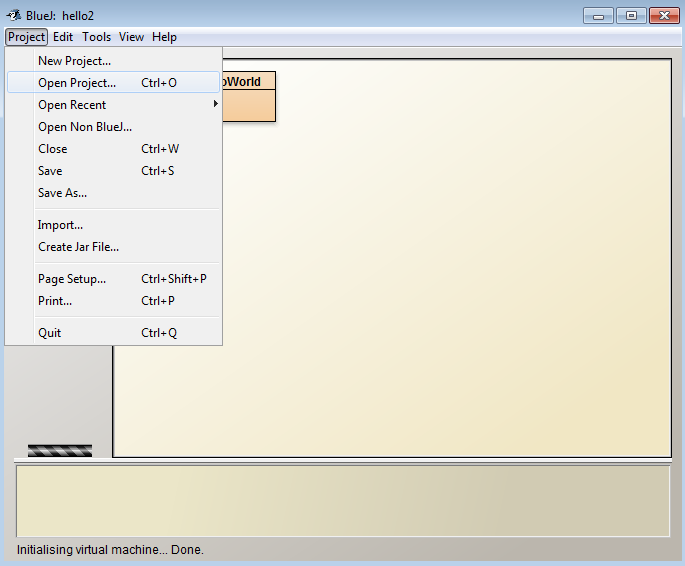


**Opening BlueJ Project in Windows**

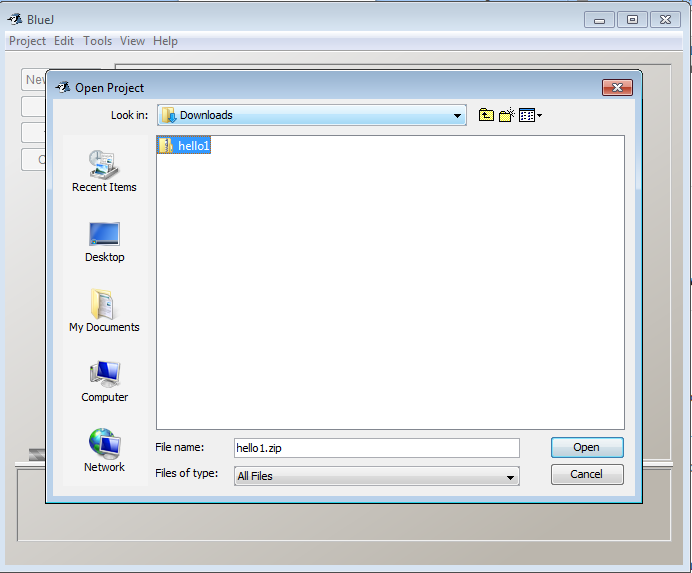
The HelloWorld project contains these files in a zip folder:



After downloading the zip folder, open the BlueJ application, and from the menu bar, select Project -> Open Project.



Select the zip folder you have just downloaded. Click "Open," and you are ready to compile and run the example program along with the next videos!



# **Perimeter Assignment Introduction/Code Review**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

In the next few readings, you're going to use what you've learned in the past few lessons to create a working Java program that can calculate the perimeter of a shape, from some given points. Before we get started with that, however, we thought it would be best to go through the sample code that you'll be using as the foundation for your Java Program!

# Sample Code Explanation and First Steps

**Note: You must have downloaded BlueJ on or after September 7, 2017, from** [**http://www.dukelearntoprogram.com/**](http://www.dukelearntoprogram.com/) **in order to access the Shape and Point Java classes used in the following programming assignments.**

In these exercises, you will use the Shape and Point classes to answer questions about a Shape that is made up of a collection of points from the x-y plane, as shown in this lesson. The shape is defined by drawing a line between two adjacent points, for every pair of adjacent points, and also a line between the first and last point. Be sure to consult the documentation on DukeLearnToProgram to understand how the Shape, Point, DirectoryResource and FileResource classes work:<http://www.dukelearntoprogram.com/course2/doc/javadoc/index.html?course=2>[.](http://www.dukelearntoprogram.com/course2/doc/)

## Calculating information about shapes

In this assignment, you will complete the PerimeterAssignmentRunner class to calculate lots of interesting facts about shapes. This class has been started for you in the BlueJ project called PerimeterAssignmentRunner (go to:<http://www.dukelearntoprogram.com/course2/files.php> and download the **Quiz - Calculating the Perimeter of a Shape** BlueJ project). This project also contains several data files. In addition, you will need to look at the documentation for the Shape class and the Point class.

The PerimeterAssignmentRunner class already includes the following complete methods:

1. The getPerimeter method has one parameter s of type Shape. Given a shape, this method returns the perimeter of the shape.
2. The testPerimeter method has no return value, hence its return type is void. This method is used to select a data file by using the FileResource class, create a shape based on the points from that data file, and then calculate the perimeter of the shape and output its value.
3. The triangle method has no return value and creates a triangle that you can use to test the methods you will create in this assignment.
4. The printFileNames method, which we will discuss in a future lesson.
5. The main method.

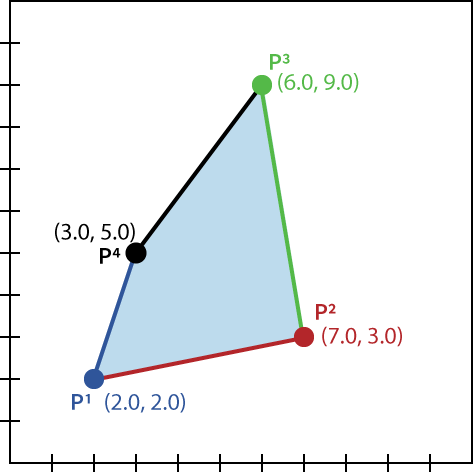
For this assignment, you will add or modify several methods in the PerimeterAssignmentRunner class.

## A few ground rules for Shapes

For the sake of familiarizing ourselves with the abstract concept of a Shape, we're going to take a look at an example. We can think of a shape as a collection of points that are connected together by lines, so for our example, we're going to use a shape with four random points. Here is an example Shape that has 4 points in this order: (2.0, 2.0) (7.0, 3.0) (6.0, 9.0) (3.0, 5.0)

By defining the shape with the points in this order, that means the shape has these lines: a line from (2.0, 2.0) to (7.0, 3.0), a line from (7.0, 3.0) to (6.0, 9.0), a line from (6.0, 9.0) to (3.0, 5.0), and finally, there is a line from the last point (3.0, 5.0) to the first point (2.0, 2.0).

Here's what our shape would look like:



## An Explanation of the complete getPerimeter method:

Here is the idea of how the getPerimeter method works with our example. Remember the Shape is defined as these four points: (2.0, 2.0) (7.0, 3.0) (6.0, 9.0) (3.0, 5.0) To calculate the perimeter, we would loop over all the points in Shape, and for each point calculate the length of the line formed with the previous point. We're just using the distance formula for these calculations, but we'll actually be able to get our program to calculate that for us in the code without having to remember the distance formula itself. Starting with the first point (2.0, 2.0) we would need the last point in the Shape with it, (3.0, 5.0) and calculate its length as 3.16 . For point (7.0, 3.0), calculate its length with (2.0, 2.0) which is 5.09. For point (6.0, 9.0), calculate its length with (7.0, 3.0) which is 6.08. For point (3.0, 5.0), calculate its length with (6.0, 9.0) which is 5.

We return the sum of these, which is 3.16 + 5.09 + 6.08 + 5, which is 19.33.

Now let's see how the code for getPerimeter implements this idea. The getPerimeter method reads as follows:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

public double getPerimeter (Shape s) {

        // Start with totalPerim = 0

        double totalPerim = 0.0;

        // Start wth prevPt = the last point

        Point prevPt = s.getLastPoint();

        // For each point currPt in the shape,

        for (Point currPt : s.getPoints()) {

            // Find distance from prevPt point to currPt

            double currDist = prevPt.distance(currPt);

            // Update totalPerim by currDist

            totalPerim = totalPerim + currDist;

            // Update prevPt to be currPt

            prevPt = currPt;

        }

        // totalPerim is the answer

        return totalPerim;

}





So immediately, we see some other classes and methods being called here. The “Point” class is something that is being used here and it was loaded in the duke **package** at the top of the project, with the line import edu.duke\*; . To find out more about this class, you can reference [the documentation on the Duke site](https://www.dukelearntoprogram.com/course2/doc/javadoc/index.html?course=2). Within that page, you’ll have to select the edu.duke package, and then select the “Point” class. This documentation shows that this class can make use of the “distance” method, which outputs a **double**-type value. You will also see the “distance” method being called in the above “getPerimeter” method. To run through the code sample above, we proceed as follows:

1. This method “getPerimeter” first has a parameter, **s**, which is of the type Shape.
2. We then create a double-type variable named totalPerim and initialize it as 0. This variable needs to be double-type, as the distance between two points is not always an integer, and subsequently the perimeter will not always be an integer.
3. Next, we create a new Point object named prevPt. This object is initialized as the output of the getLastPoint method from the object **s**. As it is listed in the documentation for the Shape Class, getLastPoint is a Point-type method that returns to the last point added to a selected object/shape (checking the documentation on all these methods is essential).
4. We next enter a “for each” loop, which initializes the Point-type variable currPt for each entry in the **iterable** getPoints. In the documentation for the Shape class, we see that the iterable getPoints allows access to a shape one point at a time.
5. Next, we create a double-type variable currDist, and initialize it by referencing another method, distance. This variable will be initialized as the distance from prevPt to currPt.
6. We then update the variable totalPerim (which was previously initialized at 0.0) to be equal to totalPerim + currDist. We do this, so that we can keep a running total. Each time this loop runs (which, as an iterable of the getPoints method, will be as many times as there are points in the Shape Object, **s**) totalPerim will be with the currDist between the next two points.
7. When the loop iterates through every point in **s**, we exit, and return the double-type variable totalPerim as the output of the “getPerimeter” Method

## An Explanation of the complete testPerimeter and main Methods:

1

2

3

4

5

6

public void testPerimeter () {

        FileResource fr = new FileResource();

        Shape s = new Shape(fr);

        double length = getPerimeter(s);

        System.out.println("perimeter = " + length);

}





First of all, we see that the testPerimeter method has no type, and so it gets the designation void. Proceeding from there, the code proceeds as follows

1. We initialize a FileResource object named fr, using the **new** keyword and the FileResource class (this was imported with the edu.duke package).
2. From the duke documentation on the FileResource class, we are shown that when called without a parameter, the FileResource class will open a dialogue box and allow us to select a file. This file will be stored as the variable fr.
3. Next, we initialize a Shape-type object named **s** (this is the object the code above referenced so many times), as a new object and call the Shape class, using the variable fr (the FileResource Object we just created) as the parameter for this shape object. This means that we will use the contents of fr (the file we selected in a dialogue box) to create a shape object.
4. We create a double-type variable named length, and initialize it as the output of the getPerimeter method, with the parameter of s (the shape object we just created). This will pass the Shape **s** to the getPerimeter method (detailed above), and initialize the variable length as the output of that method. If you recall, getPerimeter accepts a shape-type object as a parameter, and outputs a double-type variable.
5. We then issue a system  output command to print the string “perimeter = “ and the variable “length”.

And for the final bit of this sample file code that we’re examining in this exercise, the main method. This is the area that our program will execute by default after we compile and run it. For that reason, it’s important to take a look at anything in the main method.

1

2

3

4

public static void main (String[] args) {

      PerimeterAssignmentRunner pr = new PerimeterAssignmentRunner();

      pr.testPerimeter();

    }





Reading through this code we see that it simply creates a **new** PerimeterAssignmentRunner-type object, pr, and then calls the method testPerimeter using the object pr, which we outlined above.

## Running through the program as it is currently written

So as a brief summary, if we were to compile and run this program, it would progress as follows:

1. Execution starts in the main method, creates a new PerimeterAssignmentRunner-type object
2. Using the new PerimeterAssignmentRunner-type object, call the method testPerimeter
3. The testPerimeter method creates a new FileResource object, fr,  prompting us to select a file via a dialogue box
4. We use this FileResource object to create a new Shape object, s.
5. Create a variable called length, and initialize it to be “the result of calling the getPerimeter method on s”, the shape object we just created.
6. The getPerimeter method creates a new double-type variable called totalPerim, and creates/initializes the point-type variable prevPt by calling the getLastPoint method on our shape, s.
7. We enter a “for each” loop by calling the getPoints method on our shape, s, and initializing the variable currPt to to be the current point in the iterable. This will loop for as many points as there are in shape s
8. We proceed through the “for each” loop, adding the distance between the last two points (currDist) to totalPerim as we go, and updating currPt at the end of each loop.
9. When all the loops are done, getPerimeter returns the variable totalPerim to the testPerimeter method, thereby initializing the variable length.
10. testPerimeter then uses the system output command to print a string, and the variable length, finishing the testPerimeter method and returning to the main method.
11. The main method finishes executing and terminates.

# Conclusion

There’s a great deal of information to encode here, but there’s a few major takeaways in terms of programming ideas that are very important.

1. We were able to use a “for each” loop with an iterable method to easily loop through every entry in a text file (this is very important and you’ll be asked to do this on your own in future activities).
2. We initialized a variable outside of a loop and then updated its value throughout a loop to keep track of a running tally (we used this to keep track of a total perimeter of a shape, but it could just as easily be used to track the number of iterations throughout a loop, or a number of other functions).
3. We saw how you can use multiple methods and classes that are loaded into a project through an import , and how to call methods inside and outside of the main method.

We definitely won’t go through future example code in this level of detail, but we felt it was necessary to further explain the example files given, and to jumpstart your imagination about how you might think about writing code on your own. In the next few exercises, you’ll need to utilize everything you’ve learned in the lecture videos, and what you’ve experienced firsthand in this example code. When you feel comfortable with the information in this reading, proceed on to the next activity!

# **Perimeter Assignment: Part One**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

# Introduction

In this assignment, you will complete the PerimeterAssignmentRunner class to calculate lots of interesting facts about shapes. This class has been started for you in the BlueJ project called PerimeterAssignmentRunner (**this is the same project file that we were looking at in the previous reading, so feel free to open the one you downloaded for the last reading**). This project also contains several data files. In addition, you will need to look at the documentation for the Shape class and the Point class, which we went over in the last reading.

Our goals for this exercise are to:

**1a. Complete writing the method getNumPoints** that has one parameter s that is of type Shape. This method returns an integer that is the number of points in Shape s.

**1b. Add code in the method testPerimeter** to call getNumPoints and to print the result.

**2a. Complete writing the method getAverageLength** that has one parameter s that is of type Shape. This method returns a number of type double that is the calculated average of all the sides’ lengths in the Shape S.

**2b. Add code in the method testPerimeter** to call the method getAverageLength and to print out the result.

## Discussion

### Complete the method getNumPoints

For this section, we need to complete the code for the method getNumPoints. We are instructed that the method has one parameter, s, that is a Shape-type object. This makes sense with the other code we’ve seen in this project, as the only shape-type object we’ve created is, in fact, named s. To understand how to show what parameter a method has, check out the code review we did in the last reading. (**Hint:** the getPerimeter method also has a parameter named s that is of type shape, but the getPerimeter method returns a double-type variable, whereas we want getNumPoints to return an int-type variable).

After passing the parameter along, we’ll need to write some language that will allow for this method to return an int that is the number of points in Shape s. This immediately tells us that at some point, we will need to declare an int-type variable to return at the end of this method (review the video about variables and types if you’re having trouble with variable declaration). This variable will be a counter for the number of points within a given shape, s, which means that it will need to be increased for each point in the shape, s. This tells us quite a bit about how we might translate that action into java syntax, namely that we’ll be using some sort of loop. For more information about writing code that iterates over an entire object, see the video “Seven Steps in Action: Translating to Code” and look into the code review for this exercise (Hint: the getPerimeter method has both a for each loop, and a variable that updates after each iteration).

### Add Code in the Method testPerimeter to output the result

In this case, we’re simply looking to write some code that will output the result of the getNumPoints method. This tells us that within the testPerimeter method we’ll need to add a variable to correspond to the output of the getNumPoints method. The prompt told us that this method is supposed to output an int-type variable, which means that whatever variable we declare here will also need to be int-type. We’ll then need to initialize it to the output of the getNumPoints method for the shape s. After initialization, we’ll just need to add some code that will cause the system to output some text and our variable (**Hint:** the testPerimeter method already has examples of all of these processes).

At this point, it’s worth compiling your code and running it on some of the example text files included with the BlueJ project file. For the sake of this example, we’ll only use the example1.txt. When you run your program on example1.txt, it should tell you that the shape has four points. And if we open the text file itself, we can see that the shape does indeed have four points, so that checks out. You should run it on the other text files as well, to verify that it does check for the number of points in the shape, and does not just always output 4 points.

### Complete the method getAverageLength

Before we can think about what code we would need to write to figure out the average side length for the shape s, we need to understand exactly what we’re trying to calculate, and then translate that to java syntax. In this case, a couple of things that help us, we know that the way to calculate the average side length is the (sum of all sides lengths) / (number of sides). Additionally, we know that the number of sides is equal to the number of points in a two dimensional shape. We already know how to code a method that can calculate the perimeter of a shape, and we know how to code a method that can count the number of points in a shape, so now it’s just a matter of writing a method that can do both and return a double-type variable that is the quotient of those two quantities.

One thing to keep in mind, however, is **types**. It’s important to remember that when you’ll be doing operations on more than one variable, you may need to make sure they’re the same type, or that you’re casting your variables correctly for the operation you’re attempting. For more information on types and casting, see the “Types” video. For example, the getPerimeter method uses double-type variables, and the getNumPoints method uses int-type variables.

### Add Code to testPerim to output the result of getAverageLength

You’ll do this almost exactly the same you did for the getNumPoints method.

At this point, it’s worth testing your program again, and when you compile and run it on example1.txt, it should tell you that “the average length of a side in your shape is 4.0”.

This is where we’ll leave off with this exercise, but you’ll want to **make sure you save all of your project files**, as we’ll be using the exact same project files for the next few activities.

# **Perimeter Assignment: Part Two**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

## Introduction

In this assignment, you will complete the PerimeterAssignmentRunner class to calculate lots of interesting facts about shapes. This class has been started for you in the BlueJ project called PerimeterAssignmentRunner (**this is the same project file that you were using in the last exercise, so feel free to open it up again and continue working**). The goals for this exercise are as follows:

**1a. Complete writing the method getLargestSide** that has one parameter s that is of type Shape. This method returns a number of type double that is the longest side in the Shape S.

**1b. Add code in the method testPerimeter** to call the method getLargestSide and to print out the result.

**2a. Complete writing the method getLargestX** that has one parameter s that is of type Shape. This method returns a number of type double that is the largest x value over all the points in the Shape s.

**2b. Add code in the method testPerimeter** to call the method getLargestX and to print out the result. Note if you were to select the file example1.txt, then the largest x value should be 4.0.

## Discussion

### Complete the getLargestSide method and output the results

For this section, we’ve got to complete the getLargestSide method, which we know has a parameter s, of type shape. We learned in the last assignment and readings about how to show what kinds of parameters a method has, so make sure to reference that if you’re having trouble remembering how to get started. On paper, we’re simply trying to compare the length of all of the sides of shape s, and report the largest side. This tells us that we’ll need to be able to find the length of a side of a shape, but thankfully we already know how to do that from the other methods we’ve written and the Shape Class documentation (**Hint:** the length of a side of a shape is the distance between two points of the shape).

Additionally, we know that we need to compare all of the sides of the shape, which implies we will need to iterate through each side of the shape to find its length. An important thing to remember when you’re thinking about programming, is what your objective requirements are. In this case, we’re only trying to find what the longest side is, so we only need to know if any given side is the longest side. We know how to calculate the length of a side, we know how to iterate through each point in a shape, we know how to keep a running tally while iterating, and we know how to compare values (if this is giving you trouble, revisit the video on conditionals). From here, it’s simply putting it all together.

As always, when you’re done writing your code in the getLargestSide method, remember to add the code to call the getLargestSide method in the testPerimeter method and add the code to have the system print the output (the same way we did in the last exercise). As always, this is a great time to compile and test your code, and if you run this code using example1.txt as your shape file, then the longest side should be 5.0.

### Complete the getLargestX method and output the results

Once again, for this section we know how to show what type of parameter a method has, and in this case we know that the getLargestX method will be outputting a double-type variable, so that’s a great place to start. From here, we know we’ll need to find the largest X value of any given point in shape S, so we’ll need to iterate through the points of the shape s and compare their x components (**Hint:** If you’re having trouble finding the x component of a point, make sure to check the Point class documentation in the course site). From here out, we know how to iterate through all the points in a shape, compare their values, and keep track of the largest value.

As always in these activities, call your getLargestX method in the testPerimeter method, and have the system output the results. If you run this section of code with example1.txt, the Largest X will be 4.0.

This is where we’ll leave off with this exercise, but you’ll want to **make sure you save all of your project files**, as we’ll be using the exact same project files for the next few activities.

# **Programming Exercise: Finding a Gene and Web Links**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

A PDF of the programming exercise can be found in the **Resources** tab.

For files related to this assignment, visit the DukeLearnToProgram Project Resources page for this course: <http://www.dukelearntoprogram.com/course2/files.php>. Also linked in the **Resources** tab.

You can also find the frequently asked questions page for this course’s assignments on DukeLearnToProgram: <http://www.dukelearntoprogram.com/course2/faq.php>. Also linked in the **Resources** tab.

## Part 1: Finding a Gene - Using the Simplified Algorithm

This assignment is to write the code from the lesson from scratch by following the steps below. This will help you see if you really understood how to put the code together, and might identify a part that you did not fully understand. If you get stuck, then you can go back and watch the coding videos that go with this lesson again. We recommend you try this with many of the future Java coding examples before starting programming exercises.

Specifically, you should do the following:

1. Create a new Java project named StringsFirstAssignments. You can put all the classes for this programming exercise in this project.

2. Create a new Java Class named Part1. The following methods go in this class.

3. . Write the method findSimpleGene that has one String parameter dna, representing a string of DNA. This method does the following:

* Finds the index position of the start codon “ATG”. If there is no “ATG”, return the empty string.
* Finds the index position of the first stop codon “TAA” appearing after the “ATG” that was found. If there is no such “TAA”, return the empty string.
* If the length of the substring between the “ATG” and “TAA” is a multiple of 3, then return the substring that starts with that “ATG” and ends with that “TAA”.

4. Write the void method testSimpleGene that has no parameters. You should create five DNA strings. The strings should have specific test cases, such as: DNA with no “ATG”, DNA with no “TAA”, DNA with no “ATG” or “TAA”, DNA with ATG, TAA and the substring between them is a multiple of 3 (a gene), and DNA with ATG, TAA and the substring between them is not a multiple of 3. For each DNA string you should:

* Print the DNA string.
* See if there is a gene by calling findSimpleGene with this string as the parameter. If a gene exists following our algorithm above, then print the gene, otherwise print the empty string.

## Part 2: Finding a Gene - Using the Simplified Algorithm Reorganized

This assignment will determine if a DNA strand has a gene in it by using the simplified algorithm from the lesson, but organizing the code in a slightly different way. You will modify the method findSimpleGene to have three parameters, one for the DNA string, one for the start codon and one for the stop codon.

Specifically, you should do the following:

1. Create a new Java Class named Part2 in the StringsFirstAssignments project.

2. Copy and paste the two methods findSimpleGene and testSimpleGene from the Part1 class into the Part2 class.

3. The method findSimpleGene has one parameter for the DNA string named dna. Modify findSimpleGene to add two additional parameters, one named startCodon for the start codon and one named stopCodon for the stop codon. What additional changes do you need to make for the program to compile? After making all changes, run your program to check that you get the same output as before.

4. Modify the findSimpleGene method to work with DNA strings that are either all uppercase letters such as “ATGGGTTAAGTC” or all lowercase letters such as “gatgctataat”. Calling findSimpleGene with “ATGGGTTAAGTC” should return the answer with uppercase letters, the gene “ATGGGTTAA”, and calling findSimpleGene with “gatgctataat” should return the answer with lowercase letters, the gene “atgctataa”. HINT: there are two string methods toUpperCase() and toLowerCase(). If dna is the string “ATGTAA” then dna.toLowerCase() results in the string “atgtaa”.

## Part 3: Problem Solving with Strings

This assignment will give you additional practice using String methods. You will write two methods to solve some problems using strings and a third method to test these two methods.

Specifically, you should do the following:

1. Create a new Java Class named Part3 in the StringsFirstAssignments project. Put the following methods in this class.

2. Write the method named twoOccurrences that has two String parameters named stringa and stringb. This method returns true if stringa appears at least twice in stringb, otherwise it returns false. For example, the call twoOccurrences(“by”, “A story by Abby Long”) returns true as there are two occurrences of “by”, the call twoOccurrences(“a”, “banana”) returns true as there are three occurrences of “a” so “a” occurs at least twice, and the call twoOccurrences(“atg”, “ctgtatgta”) returns false as there is only one occurence of “atg”.

3. Write the void method named testing that has no parameters. This method should call twoOccurrences on several pairs of strings and print the strings and the result of calling twoOccurrences (true or false) for each pair. Be sure to test examples that should result in true and examples that should result in false.

4. Write the method named lastPart that has two String parameters named stringa and stringb. This method finds the first occurrence of stringa in stringb, and returns the part of stringb that follows stringa. If stringa does not occur in stringb, then return stringb. For example, the call lastPart(“an”, “banana”) returns the string “ana”, the part of the string after the first “an”. The call lastPart(“zoo”, “forest”) returns the string “forest” since “zoo” does not appear in that word.

5. Add code to the method testing to call the method lastPart with several pairs of strings. For each call print the strings passed in and the result. For example, the output for the two calls above might be:

* The part of the string after an in banana is ana.
* The part of the string after zoo in forest is forest.

## Part 4: Finding Web Links

Write a program that reads the lines from the file at this URL location, <http://www.dukelearntoprogram.com/course2/data/manylinks.html>, and prints each URL on the page that is a link to youtube.com. Assume that a link to youtube.com has no spaces in it and would be in the format (where [stuff] represents characters that are not verbatim): “http:[stuff]youtube.com[stuff]”

Here are suggestions to get started.

1. Create a new Java Class named Part4 in the StringsFirstAssignments project and put your code in that class.

2. Use URLResource to read the file at <http://www.dukelearntoprogram.com/course2/data/manylinks.html> word by word.

3. For each word, check to see if “youtube.com” is in it. If it is, find the double quote to the left and right of the occurrence of “youtube.com” to identify the beginning and end of the URL. Note, the double quotation mark is a special character in Java. To look for a double quote, look for (\”), since the backslash (\) character indicates we want the literal quotation marks (“) and not the Java character. The string you search for would be written “\”” for one double quotation mark.

4. In addition to the String method indexOf(x, num), you might want to consider using the String method lastIndexOf(s, num) that can be used with two parameters s and num. The parameter s is the string or character to look for, and num is the last position in the string to look for it. This method returns the last match from the start of the string up to the num position, so it is a good option for finding the opening quotation mark of a string searching backward from “youtube.com.” More information on String methods can be found in the Java documentation for Strings: <http://docs.oracle.com/javase/7/docs/api/java/lang/String.html>.

Caution: The word Youtube could appear in different cases such as YouTube, youtube, or YOUTUBE. You can find the URLs more easily by converting the string to lowercase. However, you will need the original string (with uppercase and lowercase letters) to view the YouTube URL to answer a quiz question because YouTube links are case sensitive. The link <https://www.youtube.com/watch?v=ji5_MqicxSo> is different than the link <https://www.youtube.com/watch?v=ji5_mqicxso>, where all the letters are lowercase.

# **Programming Exercise: Parsing Export Data**

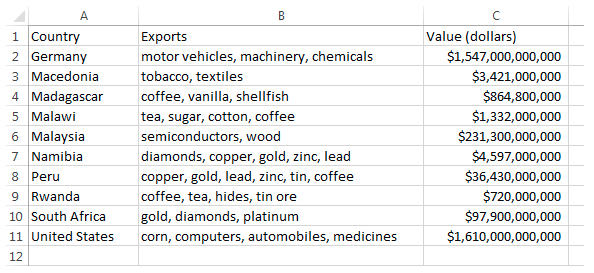
[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

([Click here for PDF version.](https://d396qusza40orc.cloudfront.net/phoenixassets/duke-java-programming/ProgrammingExercise-ParsingExportData.pdf))

### Assignment

The CSV file **exportdata.csv** has information on the export products of countries; you can download a .zip folder with this and other export data files [here](http://www.dukelearntoprogram.com/course2/data/exports.zip). In particular it has three column headers labeled **Country**, **Exports**, and **Value (dollars)**. The **Country** column represents a country from the world, the **Exports** column is a list of export items for a country, and the **Value (dollars)** column is the dollar amount in millions of their exports in the format of a dollar sign, followed by an integer number with a comma separator every three digits from the right. An example of such a value might be “$400,000,000”.

The CSV file **exports\_small.csv** is a smaller version of the file above with the same columns that you may find helpful in testing your program. We show a picture of it here.



**Write the following program.** Be sure to see the sample program in this lesson's videos.

1. Write a method named **tester** that will create your CSVParser and call each of the methods below in parts 2, 3, 4, and 5. You would start your code with:

1

2

FileResource fr = new FileResource();

CSVParser parser = fr.getCSVParser();





Each time you want to use the parser with another method, you will need to reset the parser by calling fr.getCSVParser() again to get a new parser.

1

parser = fr.getCSVParser();





2. Write a method named **countryInfo** that has two parameters, **parser** is a **CSVParser** and **country** is a String. This method returns a string of information about the country or returns “NOT FOUND” if there is no information about the country. The format of the string returned is the country, followed by “: “, followed by a list of the countries’ exports, followed by “: “, followed by the countries export value. For example, using the file **exports\_small.csv** and the country Germany, the program returns the string:

1

Germany: motor vehicles, machinery, chemicals: $1,547,000,000,000





3. Write a void method named **listExportersTwoProducts** that has three parameters, **parser** is a **CSVParser**, **exportItem1** is a String and **exportItem2** is a String. This method prints the names of all the countries that have both **exportItem1** and **exportItem2** as export items. For example, using the file **exports\_small.csv**, this method called with the items “gold” and “diamonds” would print the countries

1

2

Namibia

South Africa





4. Write a method named **numberOfExporters**, which has two parameters, **parser** is a **CSVParser**, and **exportItem** is a String. This method returns the number of countries that export **exportItem**. For example, using the file **exports\_small.csv**, this method called with the item “gold” would return 3.

5. Write a void method named bigExporters that has two parameters, parser is a CSVParser, and amount is a String in the format of a dollar sign, followed by an integer number with a comma separator every three digits from the right. An example of such a string might be “$400,000,000”. This method prints the names of countries and their Value amount for all countries whose Value (dollars) string is longer than the amount string. You do not need to parse either string value as an integer, just compare the lengths of the strings. For example, if bigExporters is called with the file exports\_small.csv and amount with the string $999,999,999, then this method would print eight countries and their export values shown here:

1

2

3

4

5

6

7

8

Germany $1,547,000,000,000

Macedonia $3,421,000,000

Malawi $1,332,000,000

Malaysia $231,300,000,000

Namibia $4,597,000,000

Peru $36,430,000,000

South Africa $97,900,000,000

United States $1,610,000,000,000





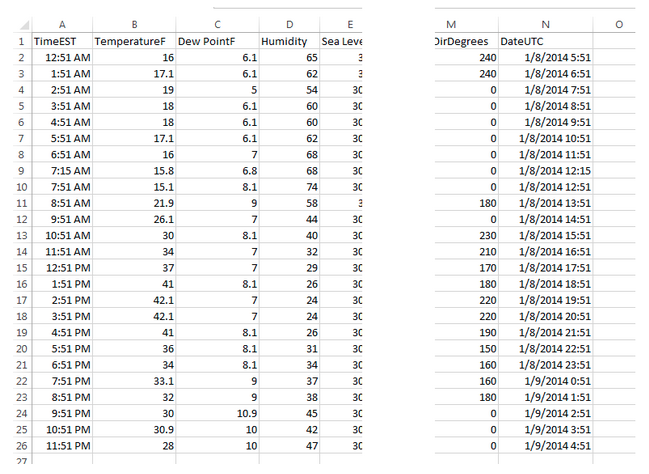
Link to FAQ page for this course: <http://www.dukelearntoprogram.com/course2/faq.php>

# **Programming Exercise: Parsing Weather Data**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

### Assignment

You will write a program to find the coldest day of the year and other interesting facts about the temperature and humidity in a day. To test your program, you will use the **nc\_weather** data folder that has a folder for each year; you can download a .zip folder with these files by clicking [here](http://www.dukelearntoprogram.com/course2/data/nc_weather.zip). In the **year** folder there is a CSV file for every day of the year; each file has the following information. For example, in the 2014 folder, we show parts of the file **weather-2014-01-08.csv**, the weather data from January 8, 2014.



You will write a program with several methods and tester methods to test each method you write. You should start with the methods from the lesson to find the hottest temperature in a day (and thus in a file) and the hottest temperature in many files and their tester methods. You can use these to write similar methods to find the coldest temperatures.

**Specifically you should write the following methods.**

1. Write a method named **coldestHourInFile** that has one parameter, a CSVParser named **parser**. This method returns the **CSVRecord** with the coldest temperature in the file and thus all the information about the coldest temperature, such as the hour of the coldest temperature. You should also write a void method named **testColdestHourInFile()** to test this method and print out information about that coldest temperature, such as the time of its occurrence.

NOTE: Sometimes there was not a valid reading at a specific hour, so the temperature field says -9999. You should ignore these bogus temperature values when calculating the lowest temperature.

2. Write the method **fileWithColdestTemperature** that has no parameters. This method should return a string that is the name of the file from selected files that has the coldest temperature. You should also write a void method named **testFileWithColdestTemperature()** to test this method. Note that after determining the filename, you could call the method **coldestHourInFile** to determine the coldest temperature on that day. When **fileWithColdestTemperature** runs and selects the files for January 1–3 in 2014, the method should print out

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

Coldest day was in file weather-2014-01-03.csv

Coldest temperature on that day was 21.9

All the Temperatures on the coldest day were:

2014-01-03 05:51:00: 41.0

2014-01-03 06:51:00: 39.0

2014-01-03 07:51:00: 35.1

2014-01-03 08:51:00: 30.9

2014-01-03 09:51:00: 28.0

2014-01-03 10:51:00: 25.0

2014-01-03 11:51:00: 24.1

2014-01-03 12:51:00: 23.0

2014-01-03 13:51:00: 25.0

2014-01-03 14:51:00: 26.1

2014-01-03 15:51:00: 28.0

2014-01-03 16:51:00: 30.0

2014-01-03 17:51:00: 30.9

2014-01-03 18:51:00: 33.1

2014-01-03 19:51:00: 33.1

2014-01-03 20:51:00: 33.1

2014-01-03 21:51:00: 30.9

2014-01-03 22:51:00: 28.9

2014-01-03 23:51:00: 28.9

2014-01-04 00:51:00: 26.1

2014-01-04 01:51:00: 24.1

2014-01-04 02:51:00: 24.1

2014-01-04 03:51:00: 23.0

2014-01-04 04:51:00: 21.9





3. Write a method named **lowestHumidityInFile** that has one parameter, a CSVParser named **parser**. This method returns the CSVRecord that has the lowest humidity. If there is a tie, then return the first such record that was found.

Note that sometimes there is not a number in the Humidity column but instead there is the string “N/A”. This only happens very rarely. You should check to make sure the value you get is not “N/A” before converting it to a number.

Also note that the header for the time is either TimeEST or TimeEDT, depending on the time of year. You will instead use the DateUTC field at the right end of the data file to get both the date and time of a temperature reading.

You should also write a void method named **testLowestHumidityInFile()** to test this method that starts with these lines:

1

2

3

FileResource fr = new FileResource();

CSVParser parser = fr.getCSVParser();

CSVRecord csv = lowestHumidityInFile(parser);





and then prints the lowest humidity AND the time the lowest humidity occurred. For example, for the file **weather-2014-01-20.csv**, the output should be:

1

Lowest Humidity was 24 at 2014-01-20 19:51:00





NOTE: If you look at the data for January 20, 2014, you will note that the Humidity was also 24 at 3:51pm, but you are supposed to return the first such record that was found.

4. Write the method **lowestHumidityInManyFiles** that has no parameters. This method returns a CSVRecord that has the lowest humidity over all the files. If there is a tie, then return the first such record that was found. You should also write a void method named **testLowestHumidityInManyFiles()** to test this method and to print the lowest humidity AND the time the lowest humidity occurred. Be sure to test this method on two files so you can check if it is working correctly. If you run this program and select the files for January 19, 2014 and January 20, 2014, you should get

1

Lowest Humidity was 24 at 2014-01-20 19:51:00





5. Write the method **averageTemperatureInFile** that has one parameter, a CSVParser named **parser**. This method returns a double that represents the average temperature in the file. You should also write a void method named **testAverageTemperatureInFile()** to test this method. When this method runs and selects the file for January 20, 2014, the method should print out

1

Average temperature in file is 44.93333333333334





6. Write the method **averageTemperatureWithHighHumidityInFile** that has two parameters, a CSVParser named **parser** and an integer named **value**. This method returns a double that represents the average temperature of only those temperatures when the humidity was greater than or equal to value. You should also write a void method named **testAverageTemperatureWithHighHumidityInFile()** to test this method. When this method runs checking for humidity greater than or equal to 80 and selects the file for January 20, 2014, the method should print out

1

No temperatures with that humidity





If you run the method checking for humidity greater than or equal to 80 and select the file March 20, 2014, a wetter day, the method should print out

1

Average Temp when high Humidity is 41.78666666666667





Link to FAQ page for this course: <http://www.dukelearntoprogram.com/course2/faq.php>

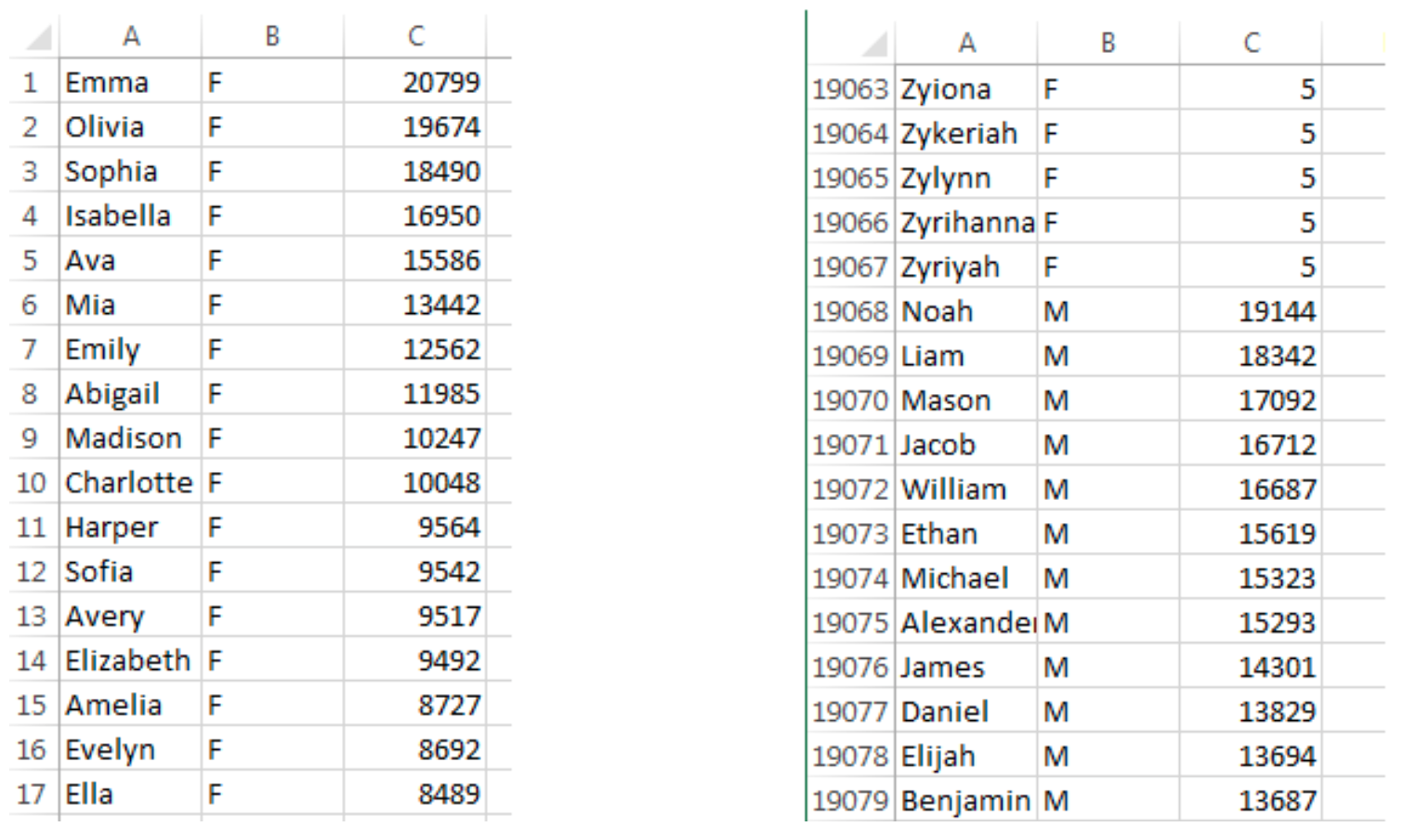
# **MiniProject Exercise Guide**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

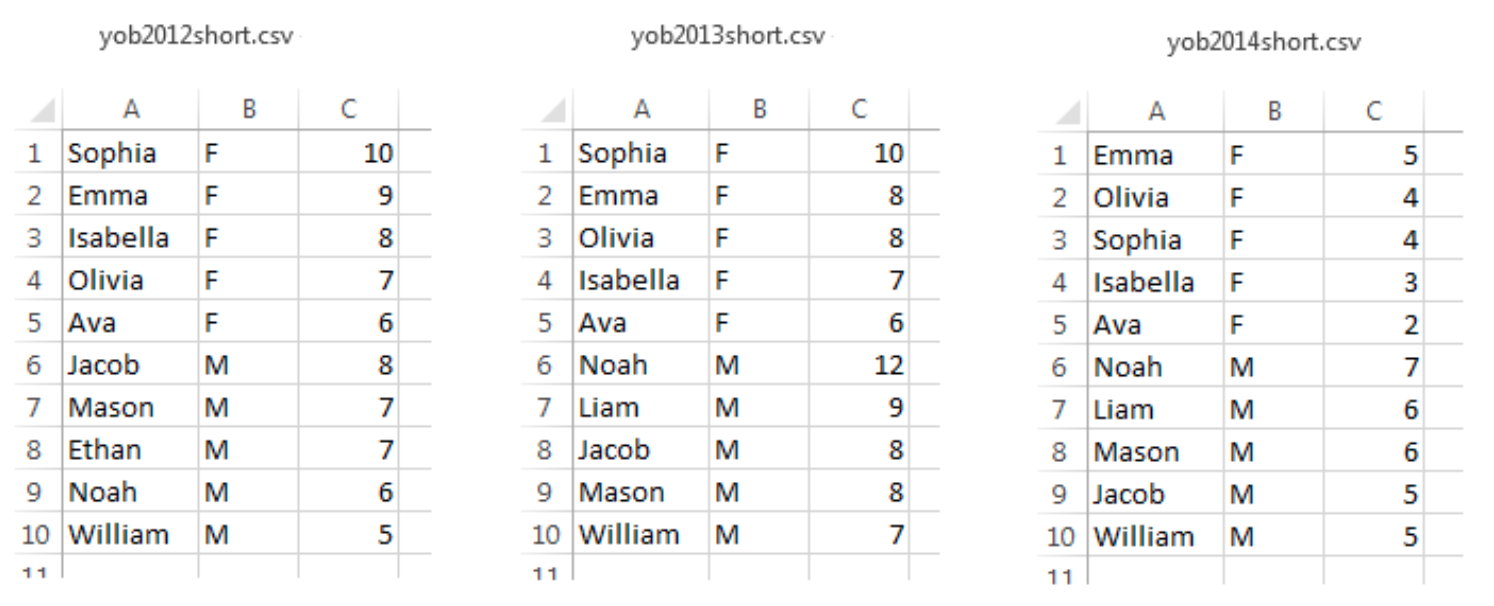
## Assignment

For this assignment, we are providing data on baby names from the United States and you will answer questions about this data. The data files give information on the first names of people born in a particular year. We have data from 1880 through 2014 on both boys and girls names. You can download a .zip folder of all baby name data by clicking [here](http://www.dukelearntoprogram.com/course2/data/us_babynames.zip).

For example, below are two pictures of parts of the file "yob2014.csv", for names of babies born in 2014. On the left is the top of the file which shows that Emma is the most popular name, the F is for female and the 20799 indicates the number of Emma’s born in 2014. The file lists all the girls first, and they are listed in order based on the number of births, from largest numbers to smallest numbers. On the right is another snapshot of the file, showing the end of the girls and the start of the boys. You can see that Noah was the most popular boys name in 2014, the M indicates a male, and 19144 is the number of boys named Noah that year. Also notice there are no headers in this file. The video showed you how to access the fields in a .csv file when there are no headers.



Since these data files are quite large, we will also provide three small files with fake data for testing. We show the three files called "yob2012short.csv", "yob2013short.csv" and "yob2014short.csv" below in that order. The three files have only ten lines each, 5 girls and 5 boys names, and smaller numbers to work with. You can download a .zip folder of the shortened baby name data by clicking [here](http://www.dukelearntoprogram.com/course2/data/us_babynames_small.zip).



You will write a program with several methods and tester methods to test each method you write. You should start with understanding the methods shown in the video. Specifically you should write the following methods.

* Modify the method **totalBirths** (shown in the video for this project) to also print the number of girls names , the number of boys names and the total names in the file.
* Write the method named **getRank** that has three parameters: an integer named **year**, a string named **name**, and a string named **gender** (F for female and M for male). This method returns the rank of the name in the file for the given gender, where rank 1 is the name with the largest number of births. If the name is not in the file, then -1 is returned. For example, in the file "yob2012short.csv", given the name Mason, the year 2012 and the gender ‘M’, the number returned is 2, as Mason is the boys name with the second highest number of births. Given the name Mason, the year 2012 and the gender ‘F’, the number returned is -1 as Mason does not appear with an F in that file.
* Write the method named **getName** that has three parameters: an integer named **year**, an integer named **rank**, and a string named **gender** (F for female and M for male). This method returns the name of the person in the file at this rank, for the given gender, where rank 1 is the name with the largest number of births. If the rank does not exist in the file, then “NO NAME” is returned.
* What would your name be if you were born in a different year? Write the void method named **whatIsNameInYear** that has four parameters: a string **name**, an integer named **year** representing the year that name was born, an integer named **newYear** and a string named **gender** (F for female and M for male). This method determines what **name** would have been named if they were born in a different year, based on the same popularity. That is, you should determine the rank of name in the year they were born, and then print the name born in **newYear** that is at the same rank and same gender. For example, using the files "yob2012short.csv" and "yob2014short.csv", notice that in 2012 Isabella is the third most popular girl's name. If Isabella was born in 2014 instead, she would have been named Sophia, the third most popular girl's name that year. The output might look like this:

1

Isabella born in 2012 would be Sophia if she was born in 2014.





* Write the method **yearOfHighestRank** that has two parameters: a string **name**, and a string named **gender** (F for female and M for male). This method selects a range of files to process and returns an integer, the year with the highest rank for the name and gender. If the name and gender are not in any of the selected files, it should return -1. For example, calling **yearOfHighestRank** with name Mason and gender ‘M’ and selecting the three test files above results in returning the year 2012. That is because Mason was ranked the 2nd most popular name in 2012, ranked 4th in 2013 and ranked 3rd in 2014. His highest ranking was in 2012.
* Write the method **getAverageRank** that has two parameters: a string **name**, and a string named **gender** (F for female and M for male). This method selects a range of files to process and returns a double representing the average rank of the name and gender over the selected files. It should return -1.0 if the name is not ranked in any of the selected files. For example calling getAverageRank with name Mason and gender ‘M’ and selecting the three test files above results in returning 3.0, as he is rank 2 in the year 2012, rank 4 in 2013 and rank 3 in 2014. As another example, calling **getAverageRank** with name Jacob and gender ‘M’ and selecting the three test files above results in returning 2.66.
* Write the method **getTotalBirthsRankedHigher** that has three parameters: an integer named **year**, a string named **name**, and a string named **gender** (F for female and M for male). This method returns an integer, the total number of births of those names with the same gender and same year who are ranked higher than **name**. For example, if **getTotalBirthsRankedHigher** accesses the "yob2012short.csv" file with name set to “Ethan”, gender set to “M”, and year set to 2012, then this method should return 15, since Jacob has 8 births and Mason has 7 births, and those are the only two ranked higher than Ethan.

Link to FAQ page for this course: <http://www.dukelearntoprogram.com/course2/faq.php>

# **Extend Your Program**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

Here are some optional ideas to extend your program even further:

* **Test edge cases.** 'Edge cases' refer to special situations where a program might break down. For example, consider searching for names that don't exist at all on the list of baby names provided, or only appear in some years but not in others. What happens when you try to find the most popular decade for such a name?
* **Use a different set of data.** Does your home country record a similar set of data? Try finding a similar but different set of data and modifying your program to work with it. What changes do you have to make with the new data set? What are the similarities? How might those similarities and differences affect how you would write a program that used data from every country in the world?
* **Explore different statistics.** If you wanted to know the median rank for a name over a period of multiple years, rather than the most popular year in that span, how would you write your program? What about finding a list of all the names that were used for fewer than 10 children in a particular year? What about finding the most popular name and year in the entire data set from a short list of your friends and family names? What are your own ideas for discovering interesting facts from this data set?
* **Adapt your program to a new problem.** This project focused on reading data from CSV files, which is a common data storage format. Try adapting your baby name program to do something new. For example, you might be a teacher with a gradebook of student test results; you could use your program to find the average scores for each test. Or you might run a business and have accounting records; you could find your most profitable month over the last two years of operation.

Whatever you do to extend your program and solve new problems, share it with us and your peers in the forums! Happy programming!

# **Programming Exercise: Batch Grayscale and Image Inversion**

[**Practice with Lab Sandbox**](https://www.coursera.org/learn/java-programming/lab-sandbox)

([Click here for PDF version.](https://d396qusza40orc.cloudfront.net/phoenixassets/duke-java-programming/ProgrammingExercise-BatchGrayscale.pdf))

### Assignment 1: Batch Grayscale

You have learned how to convert an image to grayscale, and how to select and process several images to convert them to grayscale and display them. You also learned how to copy an image and save it with a different filename. Now put this all together in one program that batch processes several images, and creates and saves new images (with new filenames) that are grayscale versions of each image.

More specifically,

* Your program should let the user select multiple image files
* For each image, create a new image that is a grayscale version of the original image
* For each image, save the grayscale image in a new file with the same filename as the original image, but with the word “gray-” in front of the filename. For example, if the original file was named lion.png, the new image would be a grayscale image and be named gray-lion.png.

Hint: Start with the Batch Grayscale program that processes many images, and add in code to save those files with new names.

### Assignment 2: Image Inversion

Write a program to create new images that are photographic negatives (or inverted images) of selected images and save these new images with filenames that are related to the original images, such as adding “inverted-” in front of the old filename. In inverting an image, a pixel’s red, blue, and green components are modified to be the exact opposite within the 0 to 255 range. That is, if a pixel’s red, blue, and green values are (34, 198, 240), then that same pixel in the inverted image would have the red, blue and green values of (221, 57, 15). Note that 255 - 34 is 221, 255 - 198 is 57, and 255 - 240 is 15.

For example, see the original and inverted images of Robert.



Here is another example. The second image below is the inverted image of the first image.



To tackle this problem:

1. First think about how to create a new image that is the inverse of another image.
2. Start by solving an example by hand, write down the steps, find patterns and then test the algorithm
3. Then create a new BlueJ class called **BatchInversions**. In this class create a method named **makeInversion** that has one parameter, an image, and returns a new image that is the inverse of the original image.
4. Then add another method called **selectAndConvert** to handle the batch processing of files. This method allows the user to select several files and display the images. Test this method to make sure it works.
5. Then modify the **selectAndConvert** method to save each inverted image as a file with a new filename. Use the old filename and append to the front of it “inverted-”