


Week 2 Module 3 – Using the StorageResource Class

Video 1. Separation of Concerns

Beyond Printing?

- 1 Set `startIndex` to 0
- 2 Repeat the following steps
 - 3 Find the next gene after `startIndex`
 - 4 If no gene was found, leave this loop
 - 5 Do **SOMETHING** with that gene
 - 6 Set `startIndex` to just past the end of the gene
- What kind of “something”?
 - Check condition before printing?
 - Count? (maybe with some condition)?
 - Save to file?
 - Build web page with them? ...




Duke UNIVERSITY

Now, you can iterate over all the genes in the DNA string and print them out. If you wanted to iterate over the genes in the DNA string and do something else to them, the algorithm would be pretty similar. In fact, for whatever you want to do with each gene, it would look pretty much like this. The line in blue is the only thing you would change to do whatever you might want to. Maybe printing only those genes that meet some condition, counting genes, saving them to a file, building a web page with all of them or anything else you can think of.

Many Different Algorithms: Copy/Paste?


- Could just copy/paste/edit?
 - Works, but...
- Generally a bad idea!
 - Error-prone
 - Tedious
 - Indicates bad programming choices



So if you wanted to do these other things, you might copy the algorithm you have pasted. Paste it into a new method and edit that line to make small changes. This approach works, but it's generally a bad idea. Why? Well, for one thing, copy and paste is error prone. You might forget to change some things that you need to. Even worse, if you find a bug in your original implementation after you have made copies, you need to go fix every copy you made. It is also tedious. You have to go find the method, copy and paste it and change it. This may not be so bad, if you want one variation. But if you want to do five different things, then it's pretty boring. And finally, it indicates bad programming design choices. Whenever you find yourself wanting to copy and paste, there's almost always a better approach.

Separation of Concerns

- 1 Set `startIndex` to 0
- 2 Repeat the following steps
 - 3 Find the next gene after `startIndex`
 - 4 If no gene was found, leave this loop
 - 5 **Print the gene**
 - 6 Set `startIndex` to just past the end of the gene



Let's take a moment to see something that we could improve about this algorithm. Why it would make a lot of work if we leave it as is and copy, paste, change and then we'll understand the motivation for how to make to fix it. This is our algorithm to print all the genes in a string.

Separation of Concerns



Print All Genes
From a String



Separation of Concerns

- 1 Set startIndex to 0
- 2 Repeat the following steps
 - 3 Find the next gene after startIndex
 - 4 If no gene was found, leave this loop
 - 5 **If the gene has a high CG ratio, print it**
- 6 Set startIndex to just past the end of the gene

Print All Genes
From a String

Separation of Concerns

Print All Genes
From a String
Print High CG Genes
From a String

Separation of Concerns

- 1 Set startIndex to 0
- 2 Repeat the following steps
 - 3 Find the next gene after startIndex
 - 4 If no gene was found, leave this loop
 - 5 **Print , then the gene, then **
- 6 Set startIndex to just past the end of the gene

Print All Genes
From a String
Print High CG Genes
From a String

Separation of Concerns

Print All Genes
From a String
Print High CG Genes
From a String

Print Genes in HTML
From a String

Separation of Concerns

Print All Genes
From a String
Print High CG Genes
From a String

Print Genes in HTML
From a String
Save Genes To File
From a String

Separation of Concerns

- 1 Set startIndex to 0
- 2 Repeat the following steps
 - 3 Find the next gene after startIndex
 - 4 If no gene was found, leave this loop
 - 5 **Whatever else you want to do with the gene...**
- 6 Set startIndex to just past the end of the gene

Print All Genes
From a String
Print High CG Genes
From a String

Print Genes in HTML
From a String
Save Genes To File
From a String

Count Genes with CGA
From a String

We're going to condense it down to a small description at the bottom and then copy and paste it and change line five to print only the genes with a high CG ratio, and then we'll condense that down to a short description.


Now we're going to copy, paste and edit to make several other algorithms to do various things with the genes from our DNA string like printing genes in HTML ,writing genes to an output file, counting genes with the codon CGA or whatever else you want to do.

All these algorithms are the same, except for the details of what they do to the DNA string. So at first, copy and pasting does not seem like a big deal.

Separation of Concerns

Print All Genes From a File	Print Genes in HTML From a File	Count Genes with CGA From a File
Print High CG Genes From a File	Save Genes To File From a File	(whatever else) From a File

Print All Genes From a String	Print Genes in HTML From a String	Count Genes with CGA From a String
Print High CG Genes From a String	Save Genes To File From a String	(whatever else) From a String




Later, we end up with some other DNA data, which list all the genes in a file, one gene per line. We need to do the same sorts of operations on this data too or the algorithm will be slightly different. It will have a for each line in the file loop, then do the same operation for the genes. With our copy and paste approach, we now need to write and test six algorithms. They're pretty similar to each other, so it may not be so hard, but it's tedious error prone work.

Separation of Concerns

Print All Genes From a File	Print Genes in HTML From a File	Count Genes with CGA From a File
Print High CG Genes From a File	Save Genes To File From a File	(whatever else) From a File

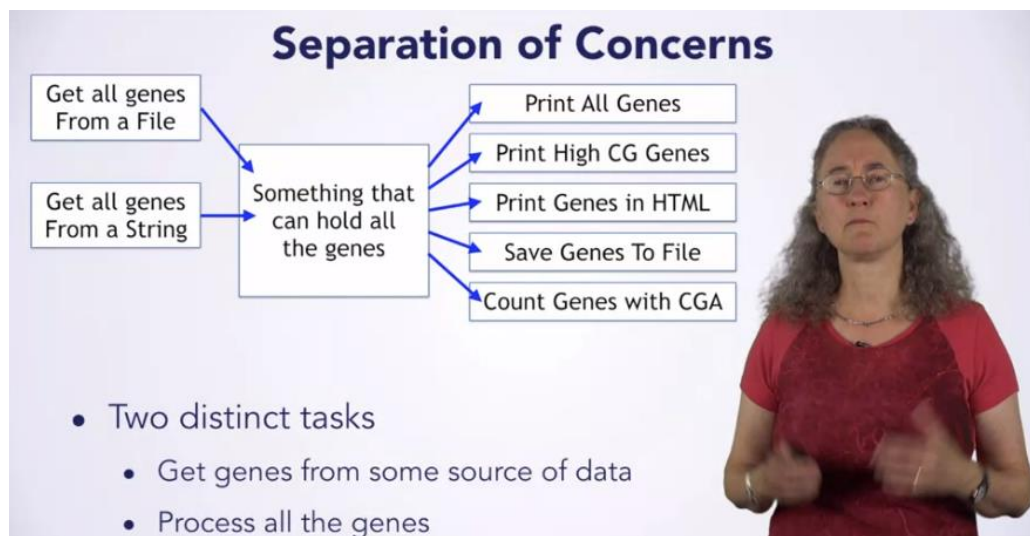
Print All Genes From a ...	Print Genes in HTML From a ...	Count Genes with CGA From a ...
Print High CG Genes From a ...	Save Genes To File From a ...	(whatever else) From a ...

Print All Genes From a String	Print Genes in HTML From a String	Count Genes with CGA From a String
Print High CG Genes From a String	Save Genes To File From a String	(whatever else) From a String



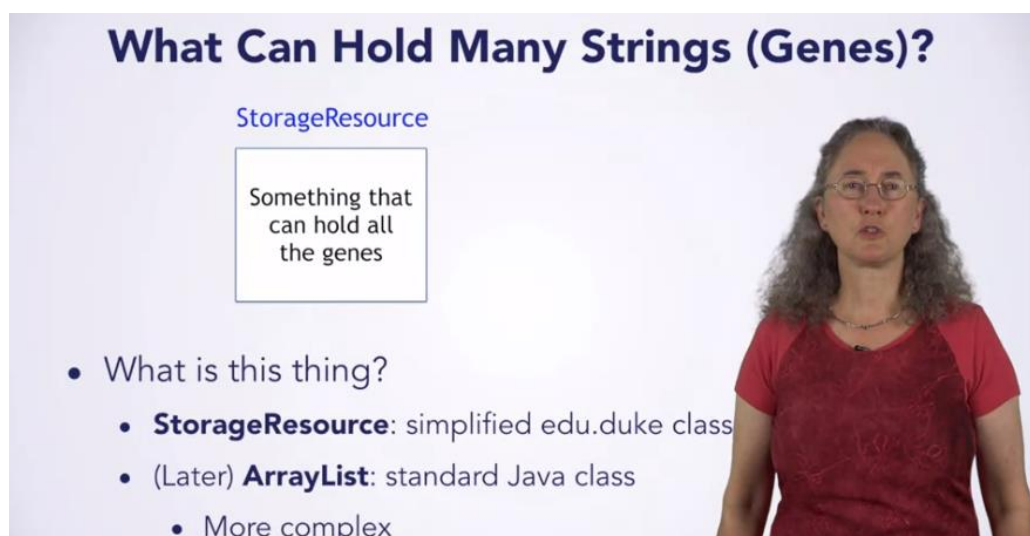
Then if we end of with some other source of data, we're going to have to go make all six algorithms again for that data source. Likewise, if we end up with a new operation that we need to do, we're going to have to write three copies of it, one for each data source. Ick, what a mess. What we would really

like to do is redesign our algorithm to use separation of concerns.



Our initial algorithm does two tasks. One is getting all the genes from some source of data and the other is printing them or whatever else we want to do to each of them. We would like to split these up.

By having the algorithms that find the genes, put them into some structure that can hold a list of all of the genes. Then having the algorithms that print the genes, count the genes or whatever else we want to do to the genes. **They should operate on that list.** Now if you need to add some new source of data, you just write a way to get its genes into our list and it automatically works with every processing algorithm you already wrote. Likewise, if you need to write a new processing algorithm, it automatically works with every source of data that you already wrote. **No copying and pasting is ever needed.**



So, what is this thing that can hold all the genes your algorithms find? We're going to start by using a


class from the edu.duke package called `StorageResource`, which is a simplified way of doing this. Later on, once you have learned a few more concepts, you will transition to using the standard `Java.util.ArrayList` class, which has similar functionality, but it's a lot more complex. Thank you.

Video 2. StorageResource Class

What Is StorageResource? How To Use It?

```
StorageResource sr = new StorageResource();  
sr.add("Hello");  
sr.add("World");  
for ( String s : sr.data() ) {  
    System.out.println(s);  
}
```

- Holds a collection of Strings
 - Can `.add(someString)` to put into it
 - Can iterate over it with `.data()`
 - Some other methods too (documentation!)



The slide features the Duke University logo in the bottom left corner. A man with glasses and a light blue button-down shirt stands to the right of the slide, gesturing with his hands.

Okay, now you've learned that you want to store your data in a list, so you can separate concerns. You learned you'll start out with our `edu.duke.StorageResource` class, which is a simplified way of doing this. What exactly is a `StorageResource` and how do you use it? It's a **class that holds a collection of strings**. You can call `.add` to put a string into the `StorageResource`.

You can also use `.data` to get an iterable, so you can iterate over all the strings that have been put into the `StorageResource` you've created. There are a few more methods and you can read about them in the Dukelearntoprogram.com website where there is a document page for the `StorageResource` class. Let's look at an example of using it.

First, we are going to declare a variable `sr` of type `StorageResource`. Once we've declared the variable `sr` of type `StorageResource`, and initialized it to a new `StorageResource`, we'll see that it's empty. Then we might add a string such as, `Hello`. Then we might add another string such as, `World`.

And then we could iterate over all the strings in the `StorageResource` by writing a for-each loop and using `sr.data`.

What Is StorageResource? How To Use It?

```
StorageResource sr = new StorageResource();  
sr.add("Hello");  
sr.add("World");  
for ( String s : sr.data() ) {  
    System.out.println(s);  
}
```

sr

Output

sr

s

Hello
World

Output

sr

s

Hello
World

Output

Hello
World

Let's see what happens when we step through this code. The first line declares a variable, so we'll make a box for it labeled sr. And then calling new, creates a new StorageResource. It's going to be an empty list of strings which the sr variable will refer to. We'll add the string, Hello, to the storage resources list of strings. Similarly, sr.add World, will add, World, to the list. Now, we're at the for-each loop. We're iterating over sr.data. So, we'll make a variable we're going to use to iterate. And have it refer to the first item in the StorageResource object we created. By referring to this first item in the StorageResource object, we'll be able to print it. Here it refers to the first item in the StorageResource, the string Hello. Inside the body of the loop, we print out s. So, we print that. Then, we go back to the beginning of the loop and refer to the next string in the list, in this case, World. So, we print, World, as we enter the loop. Once we've reached the end of the loop, we go back to the beginning of the loop and see that there are no more strings in the list. So we will go past the loop and we are done iterating over the elements of sr.

If you want to learn more about other methods in StorageResource or you forgot about the details of the ones we discussed here. You can find the documentation for this class on Dukelearntoprogram.com.

Gene Finding: Put in StorageResource

- 1 Set `startIndex` to 0
- 2 Repeat the following steps
 - 3 Find the next gene after `startIndex`
 - 4 If no gene was found, leave this loop
 - 5 Print the gene
 - 6 Set `startIndex` to just past the end of the gene



Gene Finding: Put in StorageResource

- 1 Create an empty `StorageResource`, call it `geneList`
- 2 Set `startIndex` to 0
- 3 Repeat the following steps
 - 4 Find the next gene after `startIndex`
 - 5 If no gene was found, leave this loop
 - 6 Add that gene to `geneList`
 - 7 Set `startIndex` to just past the end of the gene
- 8 Your answer is `geneList`



Finally, let's see what our algorithm would look like to find all the genes and put them into a `StorageResource`. It's pretty much what you had before, as we show here. Except, we've made three changes.

First, at the beginning of the algorithm, we make an empty `StorageResource` to put the strings we find the genes in.

After making the empty resource, we do something to each gene we found. We add it to the `StorageResource`.

Finally at the end, we give an answer. We use the `StorageResource` with all the genes in it. Rather than printing them out, this method will return a value to whatever code called it. So, the caller can use the `StorageResource` and the data inside it for whatever purpose they want. That could be to print it, or it could be to process that data further. Have fun with `StorageResource`.

Imagine you have a method named `getData` that returns a `StorageResource`. Which of the following snippets of code calls `getData` and prints every `String` in the `StorageResource` it returns?

- ☐

```
1 resource = getData();  
2 for (String s : resource.data()) {  
3     System.out.println(s);  
4 }
```
- ☒

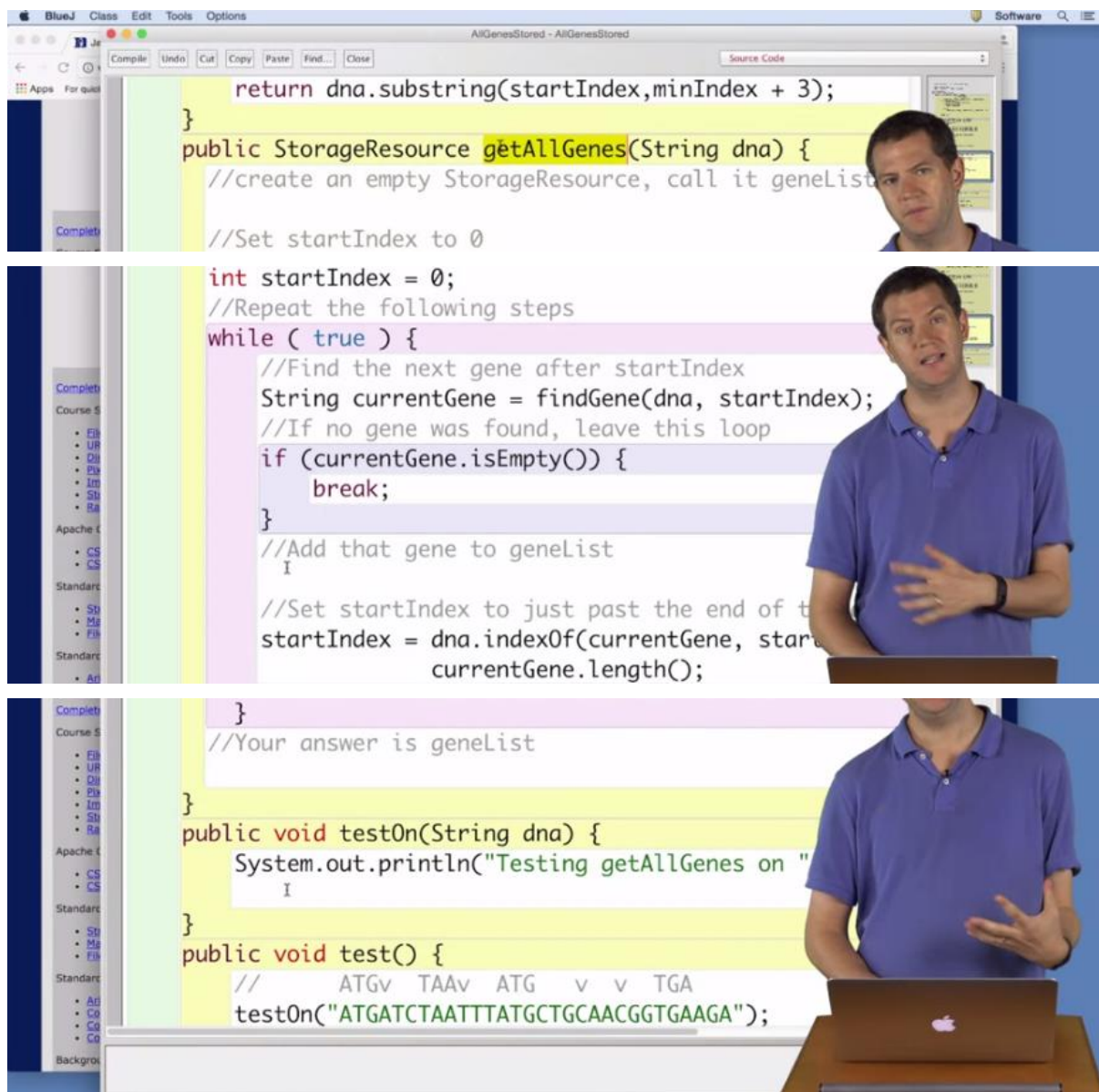
```
1 StorageResource resource = getData();  
2 for (String s : resource.data()) {  
3     System.out.println(s);  
4 }
```

Correct

Continue

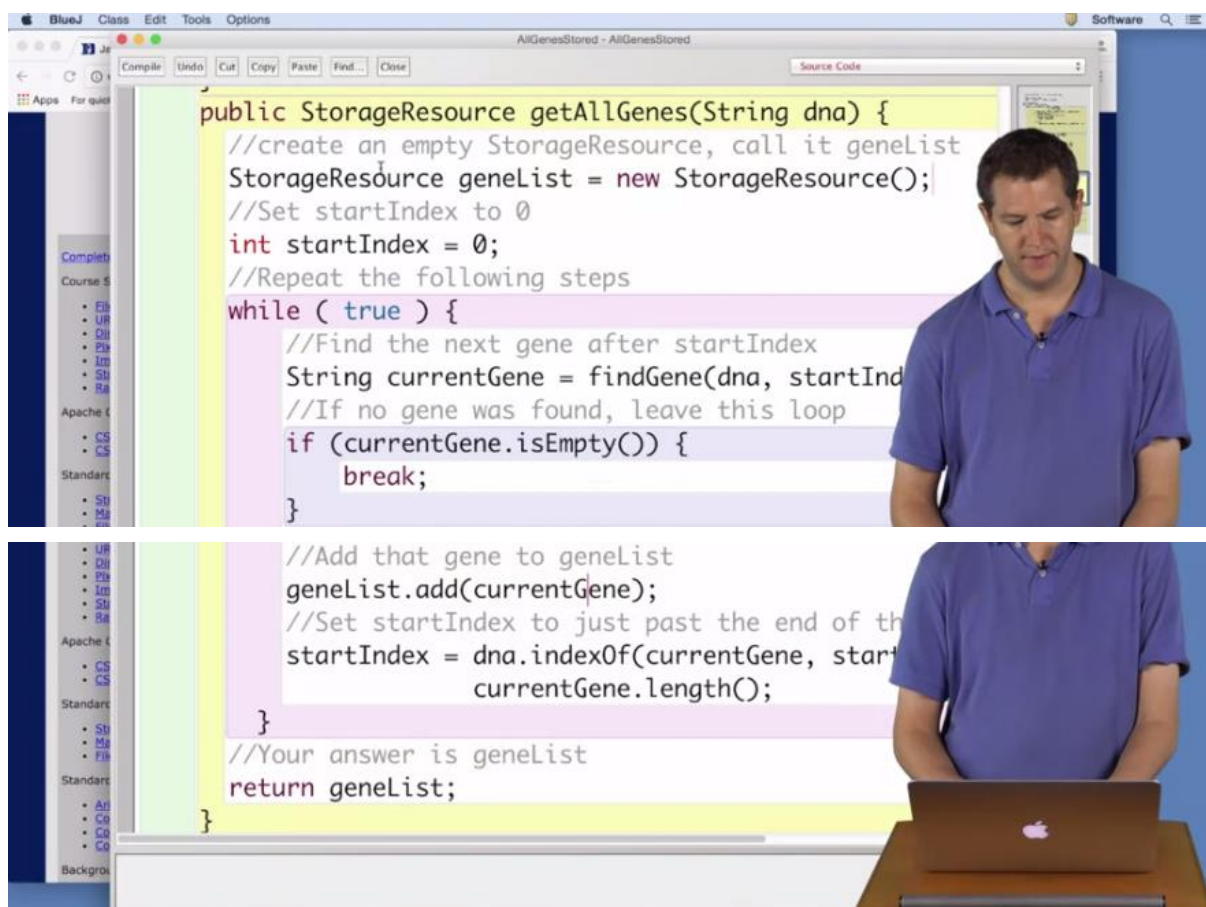
Video 3. Coding StorageResource Class

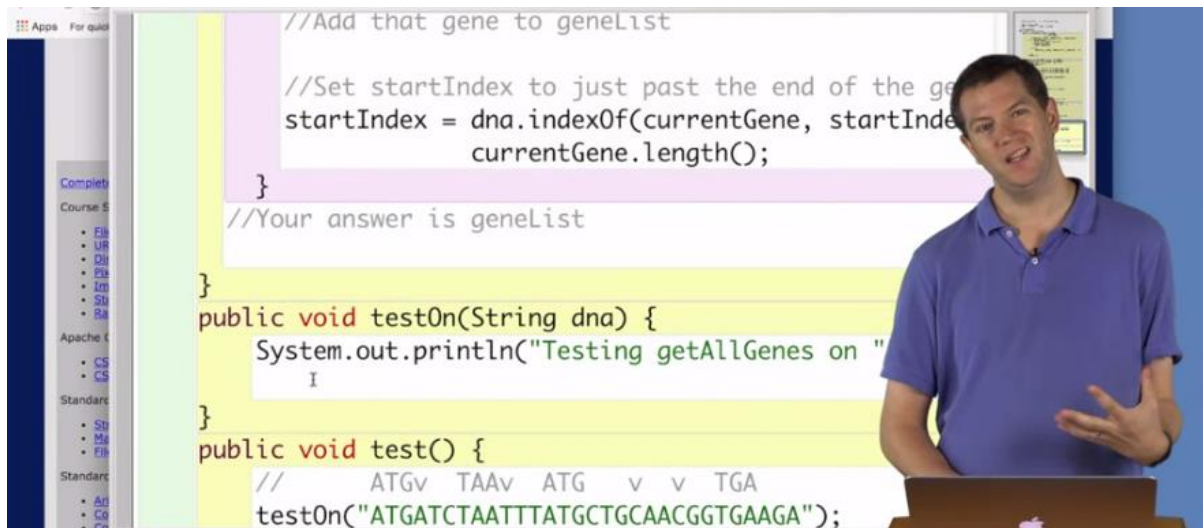
All right, so you've learned that what you would like to do now is separate 'finding the genes' from 'printing the genes'. And the way we are going to do this is we're going to take all the genes you find and put them in a storage resource. Then you can iterate over that storage resource and print them out. Or if you wanted to do other things with the genes once you've found them, filter them, calculate properties about them, whatever. You could iterate over the storage resource and do those without reduplicating all of your code to have another thing that finds them all.



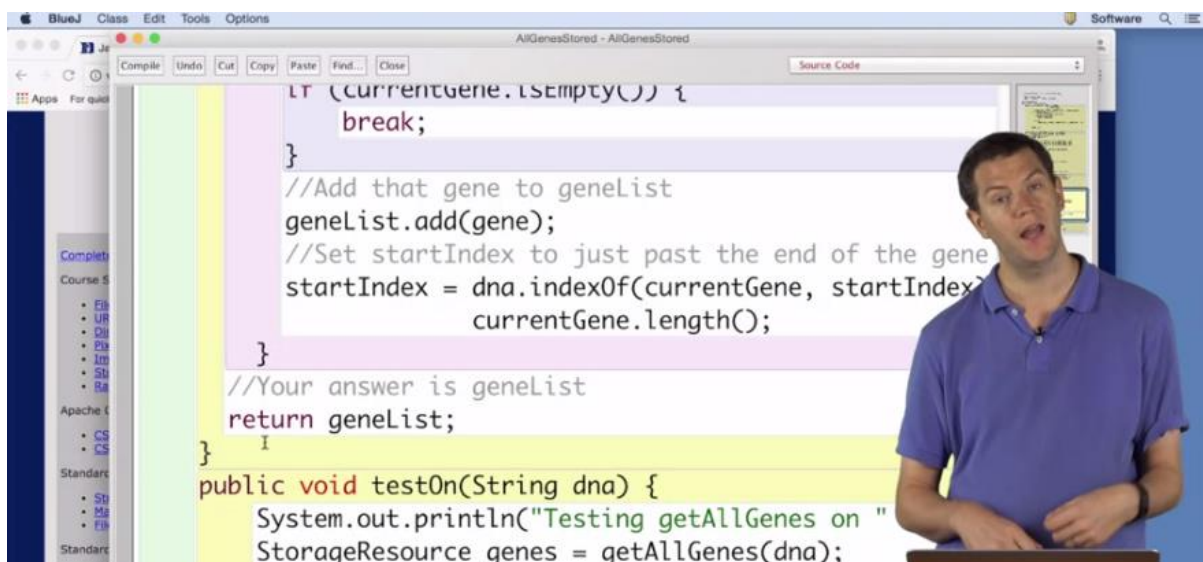
So I've taken our code that prints all the genes which we developed in a previous video and I've made a couple small changes to it. So I changed it from being called printAllGenes to getAllGenes since we're

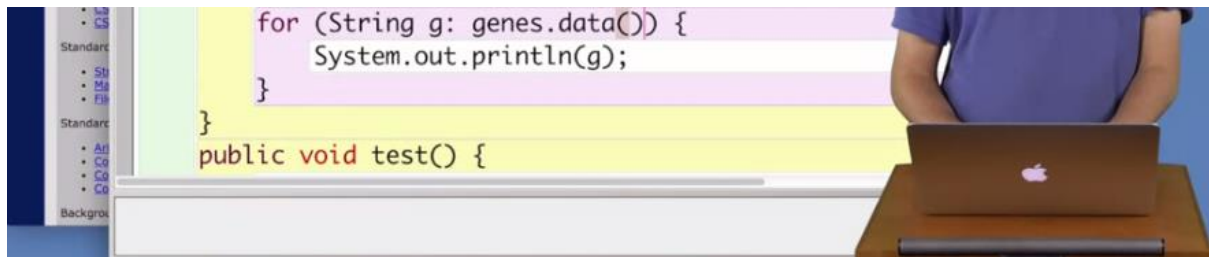
going to get them instead of print them. It returns a StorageResource now because that's what it's going to give back as its answer, one of these things that holds all of these strings. I've altered the comments here with the steps to the ones that we developed in the previous video where we came up with the algorithm. And where these have been the same as before, which is most of the steps, I've left that code. For one of these, where we were previously printing it and are now adding it to a list. I deleted the old code, the System.out.println, which we don't want. But otherwise this is pretty much the same. And then, in our test methods, I've removed that code, because we're going to have to do something a little bit different.



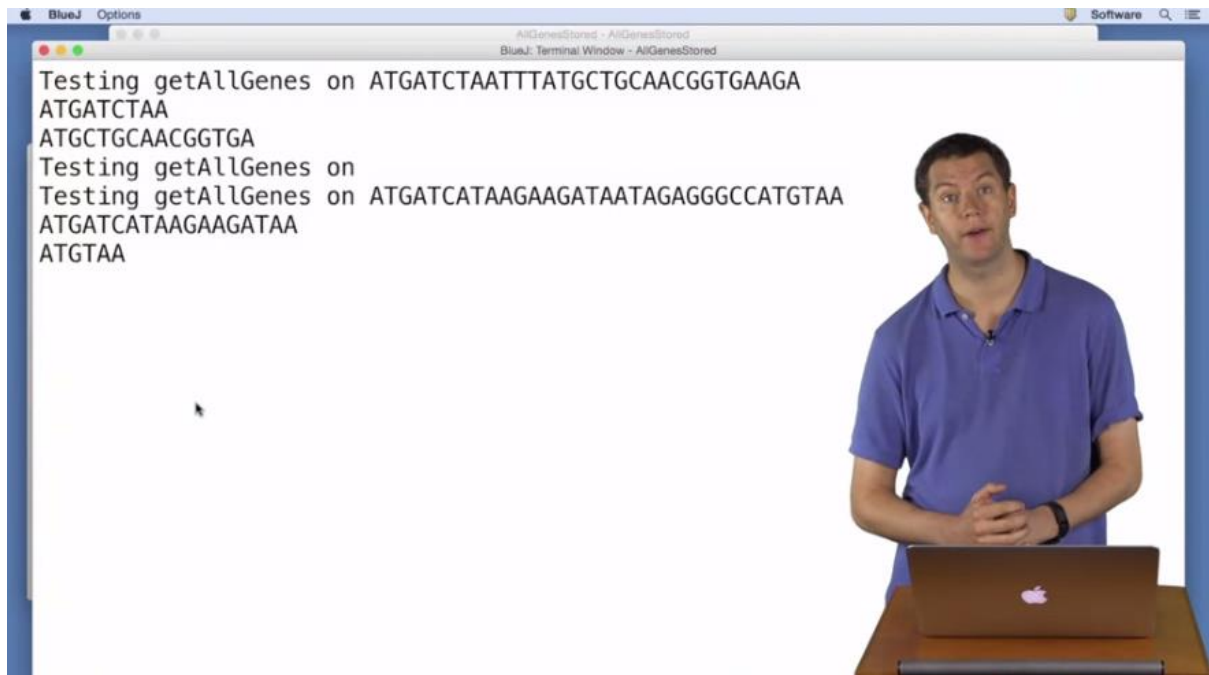


So let's start by turning these steps into code. Now the first thing we want to do is create an empty storage resource and call it geneList. So geneList is going to be a new variable. We need to declare that its type is StorageResource. geneList is its name. And it's going to be an empty StorageResource, a new one with nothing in it. It's going to be a new StorageResource. And then these steps are already translated to code until we get down here to where we want to take that gene and add it to our gene list. So that's just going to be geneList.add(gene);. And then at the end here, our answer is geneList. As you've seen many times by now, when your method or function knows its answer, it returns that answer to whoever called it. Okay?





So now I want to write my method that's going to test this. And we're going to test it by printing things out. So the behavior of our code is going to be just what we did before but it's going to be more useful. More reusable, we can put it to other purposes than just printing out these genes. So the first thing I want to do is call this method, `getAllGenes`, that we just finished writing. And store its result in a `StorageResource` variable. Store, which I'm not sure I can spell. We'll just call this `genes = getAllGenes` on that DNA. Now we want to iterate over the things in it. So I have here the Duke Learn to Program documentation because I'm not very familiar with `StorageResource`. It's not one of the standard Java classes. It's kind of a nice simplification for you. If I didn't remember how to iterate over all of these things I could look at this and say okay, add, we just used that. `Size`. `.data` returns an iterable. Those are the things that we can write for each loops. It even gives us this little example here. So what we're going to do is for All of the strings, all of the genes which are strings in these genes we just got back, we want to print that particular gene out. I'm going to hit Compile. Whoops. Up here this was called `currentGene`. Java is fussing at me because it has no variable there called just `gene`. Fix that.



And here, [SOUND], whoops, I looked up the method and then I forgot to call it. That was good, easy

to fix. I'm going to get rid of this documentation. I'm going to come over here, and I'm going to make a new one of these. And then I'm going to run it. And it ran just fine and gave me the same results as before. But our code is now more reusable. We could put it to other purposes more easily.

Reading: Programming Exercise – Finding Many Genes

Part 1

This assignment is to write the code from the lesson to use a `StorageResource` to store the genes you find instead of printing them out. 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.

Specifically, you should do the following:

1. Create a new Java project named `StringsThirdAssignments`. You can put all the classes for this programming exercise in this project.
2. Create a new Java Class named `Part1`. Copy and paste the code from your `Part1` class in your `StringsSecondAssignments` project into this class.
3. Make a copy of the `printAllGenes` method called `getAllGenes`. Instead of printing the genes found, this method should create and return a `StorageResource` containing the genes found. Remember to import the `edu.duke` libraries otherwise you will get an error message cannot find the class `StorageResource`.
4. Make sure you test your `getAllGenes` method.

Part 2

Write the method `cgRatio` that has one `String` parameter `dna`, and returns the ratio of C's and G's in `dna` as a fraction of the entire strand of DNA. For example if the `String` were "ATGCCATAG," then `cgRatio` would return `4/9` or `.4444444`.

Hint: `9/2` uses integer division because you are dividing an integer by an integer and thus Java thinks you want the result to be an integer. If you want the result to be a decimal number, then make sure you convert one of the integers to a decimal number by changing it to a float. For example, (float) `9/2` is interpreted by Java as `9.0/2` and if one of the numbers is a decimal, then Java assumes you want the result to be a decimal number. Thus (float) `9/2` is `4.5`.

Write a method `countCTG` that has one `String` parameter `dna`, and returns the number of times the codon CTG appears in `dna`.

Part 3

Write the void method `processGenes` that has one parameter `sr`, which is a `StorageResource` of strings. This method processes all the strings in `sr` to find out information about them. Specifically, it should:

print all the Strings in `sr` that are longer than 9 characters

print the number of Strings in `sr` that are longer than 9 characters

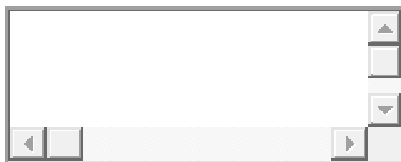
print the Strings in `sr` whose C-G-ratio is higher than 0.35

print the number of strings in `sr` whose C-G-ratio is higher than 0.35

print the length of the longest gene in `sr`

Write a method `testProcessGenes`. This method will call your `processGenes` method on different test cases. Think of five DNA strings to use as test cases. These should include: one DNA string that has some genes longer than 9 characters, one DNA string that has no genes longer than 9 characters, one DNA string that has some genes whose C-G-ratio is higher than 0.35, and one DNA string that has some genes whose C-G-ratio is lower than 0.35. Write code in `testProcessGenes` to call `processGenes` five times with `StorageResources` made from each of your five DNA string test cases.

We have some real DNA for you to test your code on. Download the file `brca1line.fa` from the DukeLearnToProgram Project Resources page. Make sure you save it in your BlueJ project so that your code can access it. You can use a `FileResource` to open the file and the `FileResource` method `asString` to convert the contents of the file to a single string so that you can use it like the other DNA strings you have been using. Here is an example:



1

2

```
FileResource fr = new FileResource("brca1line.fa");
```

```
String dna = fr.asString();
```

Modify your processGenes method so that it prints all the Strings that are longer than 60 characters and prints the number of Strings that are longer than 60 characters (you do not need to make changes to the rest of the method).

Modify the method testProcessGenes to call processGenes with a StorageResource of the genes found in the file brca1line.fa.