

Section 7: Lists

Pair up with anyone who is agreeable to pairing up with you, pick the first driver, and let's get to work!

PART I: The Sequence Class

Your SL should have told you about sequences at the start of section. In this part, you and your partner will create a `Sequence` class that uses an array to hold a sequence of doubles.


1. Open DrJava, and use a web browser to visit the class web page. Find the Section 7 area and the `Sequence.java` shell. Load it into DrJava.
2. `Sequence` is a class with two constructors (provided for you). We've also provided the complete `getNumTerms()`, `extendAtRear(double)`, and `toString()` methods.

Examine these methods, and answer these questions about them:

- (a) The only statement in the no-argument constructor is "`this(4);`". What is the constructor calling, **and** what does 4 represent?
 - (b) `extendAtRear(double)` includes this statement: `sequence[numTerms++] = item;`. Show the statements necessary to perform the same actions as this statement without using the `++` operator.
 - (c) If `toString()` is called when there are two values 8.0 and 9.0 in the sequence, what will be the content of the returned `String` object? (Be sure to plainly indicate whitespace and punctuation symbols.)
3. Your main job: Complete the stubs of these methods:
 - (a) `void doubleCapacity()`: We want a `Sequence` object to automatically grow its array when it is asked to add a value to itself and the current array is full. `doubleCapacity()` needs to:
 - double the value of capacity
 - allocate a new array of that new capacity
 - copy the content from the old array into the new array
 - have the instance variable `sequence` reference the new array
 - (b) `void extendAtFront(double)`: This method adds the given value at the front of the sequence. It needs to:
 - double the capacity of the array if necessary to hold a new value
 - shift the existing values (if any) down the array by one location each
 - place the new item at the front of the array
 - increment `numTerms`

Be sure to examine `extendAtRear(double)`; understanding how it works will help you write `extendAtFront(double)`. It might also help to review the `prepend()` method's code that we covered in class Monday.

4. Make sure your code compiles, then compile and run `Section7.java` (also available from the class web page). You should make it through all of the `Sequence` tests w/o any assertion errors. (`Section7` will have an assertion problem in the `ArithmeticSequence` section of tests. That's OK; we'll deal with that class in Part II.)

 **CHECKPOINT 1** Raise your hand. Your SL will come over and ask about your answers to the questions, and see that your `Sequence` class passes the tests.


PART II: The `ArithmeticSequence` Class

Because an arithmetic sequence has a common difference, we do not need to offer `extendAtFront(double)` and `extendAtRear(double)` methods. Instead, we can offer versions that do not take an argument, and code them to figure out what the preceding or next value will be. But, there are still methods from `Sequence` that we can use, so we'll start by inheriting from it.

1. Load into DrJava the `ArithmeticSequence.java` shell from the class web page.
2. We can reuse `getNumTerms()` and `doubleCapacity()` from `Sequence`, but `ArithmeticSequence` needs a fair amount of new code. First step: Complete the second (three-argument) constructor. It already calls `Sequence`'s one-argument constructor to set the capacity and allocate the array, but you still need to:
 - initialize the `commonDifference` instance variable; you shouldn't have any trouble figuring out where to get its value.
 - make the given seed value the only value in the `sequence` array.
3. Next, complete the stub of `extendAtFront()`. Even though there's no argument in this version, the code will be almost the same as the code you wrote for `Sequence`. Copy that code here, and modify it so that the common difference is used to determine the value to add to the front of the sequence.
4. `extendAtRear()` needs similar treatment: Copy the code from `Sequence`, and modify it so that the correct value is added to the end of the sequence.
5. Two more things to do: Because we are inheriting from `Sequence`, the old versions of `extendAtFront(double)` and `extendAtRear(double)` are available to users of an `ArithmeticSequence` object. But, if we allow those methods to be called, users can add whatever they like to our arithmetic sequence, possibly violating the common difference.

To prevent those inherited methods from being called, we will override them with methods that do one thing: throw an `UnsupportedOperationException`. Write those two one-line methods.

6. Make sure your `ArithmeticSequence` code compiles, then compile and run `Section7.java` again. This time, you should make it through all of the `ArithmeticSequence` tests. You'll get a `GeometricSequence` assertion error; that's OK.
7. One last thing: `Section7.java` has, at the bottom of the `ArithmeticSequence` tests, a commented-out block. Add `///
//` to the front of both of those comment lines, then recompile and rerun `Section7.java`. You should see an `UnsupportedOperationException`, indicating that you accomplished the last step successfully.
8. Remove those `///
//` characters from those two lines, and recompile and rerun `Section7`.

 **CHECKPOINT 2** Raise your hand. Your SL will come over and take a look at your code and the `Section7` output.

(Continued ...)

PART III: The GeometricSequence Class

We could give `GeometricSequence` the same treatment as we gave `ArithmeticSequence`, but there's nothing really new there. Instead, in this part you will finish writing a completely new class that uses an `ArrayList` object to hold the sequence, instead of using an array.

1. Create a new file in DrJava; yes, you'll be writing `GeometricSequence.java` yourselves, from scratch.
2. Before we get started, take a look at the `ArrayList` methods in the Java API. You don't see a `toString()` method, but an `ArrayList` object will answer when you ask it to run `toString()`. From where is it getting that `toString()` method? Make note of your answer; your SL will want to know.
3. Here's what your `GeometricSequence` class needs:
 - instance variables for the sequence (an `ArrayList` of `Double`) and the common ratio.
 - two constructors:
 - a two-argument constructor that accepts the seed and the ratio (in that order) and calls the three-argument constructor to create a `GeometricSequence` object with an initial capacity of 4.
 - a three-argument constructor (the third argument is the initial capacity of the `ArrayList`) that does what `ArithmeticSequence`'s three-argument constructor did, but with an `ArrayList` instead of an array, and a common ratio instead of a common difference.
 - the `getNumTerms()` method (which will just ask the `ArrayList` for that information, and return it)
 - the `extendAtFront()` method (again, no arguments) that uses the common ratio to add the appropriate value to the start of the sequence. Hopefully, you remember what your SL said at the start of section about adding to the front of an `ArrayList` object. If not, read `ArrayList`'s API entry.
 - the `extendAtRear()` method (no arguments). This will be similar to `extendAtFront()`, but even easier.
 - the `toString()` method. This is *really* easy, thanks to what you learned in Step 2, above.


That's what you need to know; get to it!

4. Make sure your `GeometricSequence` code compiles, then recompile and rerun Section7. If everything works, you should pass all of the tests and see the 'congratulations' message at the end of the output.

 **CHECKPOINT 3** Raise your hand. Your SL will come over and verify that your class works.

PART IV: Clean Up!

1. Log out of your computer.
2. Pick up your papers, writing implements, cell phones, trash, etc.
3. Push in your chair(s).

 **CHECKPOINT 4** Raise your hand. Your SL will come over and remark on how the building's insects won't need to waste their time looking for decaying food at your workspace.

You're free to go! But, if you have time, we recommend that you use it to work some more on the programming assignment (individually, of course), if you're not already done.