CSE 12 Programming Assignment 2

Lists

This assignment is open to collaboration.

You should finish reading the whole writeup before you start to code.

This assignment will exercise your understanding of array and linked lists.

This PA is due on ** **Tuesday, April 12 at 10:00pm** **.

CSE Mantra: *Start early, start often!*

You will notice throughout the quarter that the PAs get harder and harder. By starting the quarter off with good habits, you can prepare yourself for future PAs that might take more time than the earlier ones.

Getting the Code

You can download the starter code at https://github.com/ucsd-cse12-sp22/cse12-pa2-Lists. If you are not familiar with Github, here are two easy ways to get your code.

1. Download as a ZIP folder

   After going to the Github repository, you should see a green button that says *Code*. Click on that button. Then click on *Download ZIP*. This should download all the files as a ZIP folder. You can then unzip/extract the zip bundle and move it to wherever you would like to work.

2. Using git clone (requires terminal/command line)

   After going to the Github repository, you should see a green button that says *Code*. Click on that button. Select *HTTPS* and copy the link. In terminal/command line, navigate to whatever folder/directory you would like to work. Type the command `git clone _` where the _ is replaced with the link you copied. This should clone the repository on your computer and you can then edit the files on whatever IDE you see fit.

If you are unsure or have questions about how to get the starter code, feel free to make a Piazza post or ask a tutor for help.

Code Layout

- `MyList.java` – you *cannot* edit this file
- `MyTransformer.java` – you *cannot* edit this file
- `MyChooser.java` – you *cannot* edit this file
- `ArrayGL.java` – you will edit this file
- `LinkedGL.java` – you will edit this file
- `TestLists.java` – you will edit this file
- `Choosers.java` – you will edit this file
- `Transformers.java` – you will edit this file

Assignment Overview

Part 1: Implementation of MyList (42 points)

The `MyList` interface is implemented in `ArrayGL.java` and in `LinkedGL.java`:

```
public interface MyList<E>{
    E[] toArray();
    void transformAll(MyTransformer mt);
    void chooseAll(MyChooser mc);
    boolean isEmpty();
}
```

The constructor for the lists will take a generic array (`E[]`) as a parameter. We provide you with the constructor for `ArrayGL.java`. It is your task to complete the constructor for `LinkedGL.java`.

In part 1 and part 2, you will implement and thoroughly test the following methods:

- `LinkedGL(E[] contents)`
- `isEmpty` for both types of list
- `toArray` for both types of list
- `transformAll` for both types of list
- `chooseAll` for both types of list
- Implementations of the `MyTransformer` and `MyChooser` interfaces

The related interfaces `MyTransformer` and `MyChooser` are defined as:

```
public interface MyTransformer<E>{
    E transformElement(E e);
}

public interface MyChooser<E>{
    boolean chooseElement(E e);
}
```

`public LinkedGL(E[] contents)`

Constructor that creates a new `LinkedGL` with its elements from `contents` in the same order. For example, the following constructor call:

```
Integer[] input = {1, 2, 3};
LinkedGL<Integer> list = new LinkedGL<Integer>(input);
```

should create a new Linked Integer list with contents `{1, 2, 3}`.

`public E[] toArray()`

Returns the contents of the list as a new array, with *shallow copy* of the elements in the same order they appear in the list. The length of the array produced must be the same as the size of the list. To notice, you cannot initialize a generic array, like `E[] array = new E[];`. **Hint**: consider typecaste or check out lecture materials.

`public boolean isEmpty()`

Returns `true` if the list has no elements, `false` otherwise.

`public void transformAll(MyTransformer mt)`

Changes the contents of the list according to the provided `MyTransformer`. `mt` is a concrete class that implements `MyTransformer`. (We will illustrate how to implement `MyTransformer` in part 2.) It has a method called `transformElement(E e)`, which takes an element as an argument and returns the transformed element. Therefore, you should apply `transformElement` method of `mt` class to each element in the list, getting the transformed element and replacing the orginal one. For example, consider the provided `UpperCaseTransformer` that implements `MyChooser`, which transforms a string into uppercase. If we construct a list like:

```
String[] contents = {"a", "b", "c"};
ArrayGL<String> agl = new ArrayGL<String>(contents);
agl.transformAll(new UpperCaseTransformer());
```

then we should expect the contents of the list after to be `{"A", "B", "C"}`.

When the element is `null`, it should not be transformed. For example,

```
String[] contents = {"a", "b", null};
ArrayGL<String> agl = new ArrayGL<String>(contents);
agl.transformAll(new UpperCaseTransformer());
```

then we should expect the contents of the list after to be `{"A", "B", null}`.

`public void chooseAll(MyChooser mc)`

Changes the list to contain only elements selected by the `MyChooser`. `mc` is a concrete class that implements `MyChooser`. (We will illustrate how to implement `MyChooser` in part 2.) It has a method called `chooseElement(E e)`, which takes an element as an argument and check if this element should be put into the new list. It return `true`, if we should choose this element. Otherwise, it returns `false`. Therefore, you should apply `chooseElement` method of `mc` class to each element in the list, verifying if the element should be chosen and keeping the element if true. The elements should remain in the same order after `chooseAll` is called. For example, consider the provided `LongWordChooser` that implements `MyChooser`, which chooses a string whose length is more than 5. If we construct a list like:

```
String[] contents = {"longword", "longerword", "short"};
ArrayGL<String> agl = new ArrayGL(contents);
agl.chooseAll(new LongWordChooser());
```

then we should expect the contents of the list after to be `{"longword", "longerword"}`.

When element is `null`, it should not be chosen. For example,

```
String[] contents = {"longword", null, "short"};
ArrayGL<String> agl = new ArrayGL(contents);
agl.chooseAll(new LongWordChooser());
```

then we should expect the contents of the list after to be `{"longword"}`.

Part 2: Implementations of `MyChooser` and `MyTransformer` (4 points)

You must add (at least) 2 implementations of **each** of these interfaces. In other words, you should create at least 2 classes which implement `MyChooser` in `Choosers.java` and implement `MyTransformer` in `Transformers.java`. You have free choice in what you implement for these, and they will be graded manually. We have provided examples below:

```
// inside Choosers.java
class LongWordChooser implements MyChooser<String> {
    @Override
    public boolean chooseElement(String s) {
        return s.length() > 5;
    }
}

// inside Transformers.java
class UpperCaseTransformer implements MyTransformer<String> {
    @Override
    public String transformElement(String s) {
        return s.toUpperCase();
    }
}
```

“Choosers” should overwrite `public boolean chooseElement(E e)` of `MyChooser`. When you put a “chooser” into `chooseAll(MyChooser mc)`’s paramter, like `agl.chooseAll(new LongWordChooser())`, you can access`chooseElement` method by `mc.chooseElement(element)`. Similarly, “transformers” should overwrite `E transformElement(E e)`and you can access `transformElement` by `mt.transformElement(element)`.

Note: you should not use `toLowerCase()` in `Transformers.java` and >, <, or = in `Choosers.java`

Part 3: Gradescope Assignment (4 points)

You will also answer question on Gradescope regarding the assignment. Answer the two questions written below. **Make sure to submit directly to the Gradescope assignment: "Programming Assignment 2 - questions"**

1. Describe a mistake you made in your implementation, and how you fixed it. (Don't worry even if you think you implementation is incomplete when answering the question. Mention about mistakes you made to get to your current point).
2. Was it easier to implement `toArray`, `transformAll`, and `chooseAll` on one of `ArrayGL` or `LinkedGL`? Why? (150 words or less)

Getting Started

After you get the code, you will notice that the class bodies for both `ArrayGL` and `LinkedGL` are quite empty. Indeed, trying to compile the program by `javac -cp ../lib/junit-4.12.jar:../lib/hamcrest-core-1.3.jar:. *.java` will result in errors like:

```
ArrayGL.java:1: error: ArrayGL is not abstract and does not override abstract method i
public class ArrayGL<E> implements MyList<E> {
       ^
LinkedGL.java:1: error: LinkedGL is not abstract and does not override abstract method
public class LinkedGL<E> implements MyList<E> {
       ^
2 errors
```

The first thing you should do is get a basic implementation of all the required methods in place. For example, you might fill in `isEmpty` with a method that always returns `false`:

```
public boolean isEmpty() {
    return false;
}
```

This clearly returns the wrong value, but it will allow you to compile the files and start running the tester. Make sure the method returns the correct type. Do this first!

After doing this, you can start working on the individual methods. You may want to start with implementing the methods for `ArrayGL`, since the constructor is already provided.

Run the tests as you go. Even if you hanve't finished implementing other methods, as long as the files compiles, you can test for the method you're currently working on.

As you get more comfortable with the PA, move back and forth between the implementations as you see fit to make progress.

Testing

The thoroughness and correctness of your tests will be graded automatically. Correctnes will be assessed by running your tests against our reference implementation and checking if the results are as expected. Thoroughness will be assessed by running your tests against each buggy implementation and checking if the results are different than on the reference implementation.

You will be able to see the correctness assessment in Gradescope to confirm that your tests match our expected behavior.

What is wheat and chaff?

- Wheat and chaff are terms to describe good and bad code. **Be sure to understand these two terms as you will find that they will be used in later programming assignments as well!** Wheat refers to functional code while chaff refers to buggy code. Throughout this course we will do the following in order to score your written tests:
  - running your written tests against a wheat implementation to see if your tests are correct.
  - running your tests against a series of chaff implementations to make sure your tests are thorough and able to catch potential bugs.

The following table shows the test case breakdown along with some descriptions to help you write your own tests. Note, **this is not a comprehensive list of tests** and you will have to think of some of your own test cases. This list is simply a guide to help you implement your own.

Test Cases	Description	Points
chaff implementations	The following are examples of bad implementations where your tests will be expected to catch the bugs, look at the names to help get an idea of what the bug could be. For tricker bugs, further explanations are given. chaffAlwaysChoosesFirstArrayGL - in ArrayGL, chooseAll() always chooses the first element chaffIsEmptyReturnsTrueIfSizeGreaterThan0ArrayGL chaffReturnNewArrayArrayGL - in ArrayGL, toArray() does not create a new array chaffIsEmptyReturnsFalseSizeGreaterThan3 chaffChooseAllFailsIfLastNotChosenLinkedGL - in LinkedGL, chooseAll() will cause an exception if the last node is not chosen chaffDoWhileToArrayLinkedGL - in LinkedGL, a do-while loop is used in toArray() chaffDoWhileTransformArrayGL chaffFixedSizeConstructorLinkedGL chaffIncorrectTransformBoundsLinkedGL - in LinkedGL, transformAll() does not loop through the entire list chaffIncludeNullToArrayArrayGL - in ArrayGL, extra nulls are being copied to the end of the new array	10
wheat implementation	`TestLists.java` will be used against a correct implementation. This will check if the tests written are correct and do not flag any errors for the wheat implementation.	5
Constructor	Correctly populates the instance variables for the object. - Pass an empty array - Pass an array with multiple values	3
isEmpty	Correctly checks to see if the ArrayGL/LinkedGL is empty. - The list is empty, returns True - the list is not empty with multiple values, returns False - the list is not empty with only one value, returns False	6 (3 for ArrayGL, 3 for LinkedGL)
toArray	Correctly returns an array of the values that were in the ArrayGL/LinkedGL. - the list is empty - the list has a large size - creates a new array to return	6 (3 for ArrayGL, 3 for LinkedGL)
choose	Correctly chooses the desired elements. - the list is empty - choose all of the elements in the list - list with only two elements, choose second element - choose first and last in list	6 (3 for ArrayGL, 3 for LinkedGL)
transform	Correctly transforms the elements in the list. - the list is empty - the list has two items - the list has a large size	6 (3 for ArrayGL, 3 for LinkedGL)

Sanity Check

When you submit, you will see a `Sanity Check`. This is a subset of the tests we will be running on your code to help you see if you are on the right track. **Passing all of these does not necessarily mean you will get full credit!!!** You need to write your own tests to make sure you have the correct functionality for all of the required methods.

Constraints

Your implementation is subject to the following constraints; violating them will result in substantial deductions or a 0:

- You cannot use `ArrayGL` to implement `LinkedGL` or vice versa
- You cannot add, remove, or change fields in `ArrayGL` or `LinkedGL`
- Your implementations of `ArrayGL` and `LinkedGL` cannot use the built-in Java collections classes (including `ArrayList` and `LinkedList`)

- In your tests in `TestLists.java`, you can only use the `makeList` method to construct lists, and you can only call methods declared on the `MyList` interface to manipulate the lists you create. This makes sure we can automatically grade your submission.

You *are* allowed and encouraged:

- to write any helper methods or classes you need or want
- to use methods you've already written within a class to help implement others

You are free to use all of the following resources:

- Code from this PA writeup
- Code from lecture
- Code from discussion
- Code posted on the course web site and linked resources
- Code from your past PAs
- Code that was public on Piazza before the PA was released
- Code or ideas from the official Java documentation

We encourage you to make heavy use of these resources! Much of these are linked from the schedule on the course web page.

Answers for FAQ

1. Even though we only offer Choosers/Transformers examples of String, your implementation of transformerAll/chooseAll can transform/choose any object types, including String, Integer, or your own object class. Therefore, you should consider write your own Choosers and Transformers of multiple object types and test them. For example, consider such a transformer: class IntegerTransformer implements MyTransformer.
2. When you apply Choosers of String to an Integer list, there will be a java.lang.ClassCastException (you should try it). However, you do not need to worry about how to handle the error. When we are testing, we will apply the correct type.
3. We don't test null input in LinkedGL's or ArrayGL's constructors.
4. It's ok to have warnings.
5. In the writeup, it says "you should not use <, ==, or > in Choosers.java" and toLowerCase in Transformers.java because if those are allowed, students can just copy the code from LongWordChooser and UpperCaseTransformer and change < into == or > and toUpperCase into toLowerCase, which does not count as "create your own choosers and transformers." Therefore, if you want to use <, ==, > or toLowerCase, make sure you are not just taking a shortcut.
6. Magic numbers are allowed in your Choosers.java and Transformers.java source files since the sample ones contain them.
7. We don't test null inputs to chooseAll/transformAll.
8. You may write JUnit tests that test your own choosers and transformers.
9. About toArray, return a "new" array which contains the shallow copy(reference) of each element.
10. Null element does not count as empty. For example, if I create a list with {null, null, "A"}, its size should be 3 instead of 1.
11. You are not allowed to use system calls to copy array. You should copy an array though a loop.
12. chaffFixedSizeConstructorLinkedGL() is an implementation whose constructor has a fixed maximum size. Meaning if the content's length is larger than this fixed size, it won't populate the list with all of the contents.
13. You can use Arrays class in your tests

Asking for Help

Feel free to ask the staff about anything from lecture. For this PA, you may find it especially helpful to go over the worksheets and code from the lecture on Array Lists and Linked Lists. The staff will may also provide guidance and clarification on this PA.

If you have any policy questions, please ask!

Style

On this assignment, we will give you feedback on style but not deduct points for problems with style. In future assignments, we will deduct style points if you have:

- Lines in code files that are longer than 100 characters
- Inconsistent indentations
- Unit tests with generic names like “test1”
- Helper methods with generic names like “method1”

Submission Checklist

Checklist:

- `LinkedGL` constructor with `E[]` parameter
- `isEmpty` for `ArrayGL` and for `LinkedGL`
- `toArray` for `ArrayGL` and for `LinkedGL`
- `transformAll` for `ArrayGL` and for `LinkedGL`
- `chooseAll` for `ArrayGL` and for `LinkedGL`
- A correct and thorough set of tests
- 2 implementations of `MyChooser` (in addition to `LongWordChooser`)
- 2 implementations of `MyTransformer` (in addition to `UpperCaseTransformer`)
- 2 assignment questions

Submitting

Part 1 & 2

On the Gradescope assignment **Programming Assignment 2 - code** please submit the following files:

```
* ArrayGL.java
* Choosers.java
* LinkedGL.java
* MyChooser.java
* MyList.java
* MyTransformer.java
* TestLists.java
* Transformers.java
```

Part 3

Please submit your answers to the questions from part 3 on the Gradescope assignment **Programming Assignment 2 - questions**. You may submit as many times as you like till the deadline.

Scoring (50 points total)

- **Coding Style** (0 points)
- **Correctness** (42 points)
  - Does your code compile? If not, you will get 0 points.
  - Does it pass all of the provided unit tests?
- **Implementations of MyChooser/MyTransformer** (4 points)
- **Gradescope Assignment** (4 points)

Modern Java: Lambda Expressions

This section is not required for completing the PA, but is really cool and can help you write more tests more quickly.

Read this blog post, up to the part about Block Lambda Expressions:

https://medium.freecodecamp.org/learn-these-4-things-and-working-with-lambda-expressions-b0ab36e0fffc

Note that `MyChooser` and `MyTransformer` are *functional interfaces*. This means that another way to write this example:

```
String[] contents = {"a", "b", "c"};
ArrayGL<String> agl = new ArrayGL<String>(contents);
agl.transformAll(new UpperCaseTransformer());
```

is to instead write:

```
String[] contents = {"a", "b", "c"};
ArrayGL<String> agl = new ArrayGL<String>(contents);
agl.transformAll(s -> s.toUpperCase());
```

This doesn’t require writing the `UpperCaseTransformer` class at all!

Lambda expressions were added in Java version 8, and allow for us to write methods and classes in the concise style shown above if we design with functional interfaces in mind.