Project 1: ArrayList Services

A class that provides specialized operations on ArrayLists.

Overview

In this project, you will complete several methods in the `ArrayListServices` class that provide useful operations on ArrayLists. This project is designed as both a Java review/warm up, an opportunity to code list processing algorithms, as an introduction to the use of generic data types, and as an exercise to learn to adhere to a style guide.

Learning Goals:

- Apply knowledge of looping and conditional statements to implement a solution to the stated problem of a method specification.
- Increase your skills by writing list processing code.
- Gain experience working with generically typed code.
- Apply the ArrayList class interface and its methods in your code.
- Gain experience debugging and testing code.
- Gain experience using a style guide.

Project Specification:

Background

Basic list operations such as adding new data, removing data, fetching or accessing data are provided by the ArrayList class, which, as the name suggests, encapsulates an array. While the basic list operations are provided for us, we need to create more specialized operations on lists that frequently occur when dealing with list data in programming. Three of these specialized list operations are the focus of this project.

Generic Typing
In order for our program to be widely useful, we design its methods to work with any kind of data. This is called generic typing. The other benefit of coding with generic types is that it allows us to focus on designing and implementing the algorithm of the list processing task at hand without being concerned about the specifics of the data Objects being processed.

In Java, generic types are denoted by a capital letter variable that stands for any type, and angle brackets, e.g.: <T>. If you look at the Java API documentation for the ArrayList class, you will see that it is a genetically typed class: `Class ArrayList<E>`. Note that the letter used as the generic type variable does not matter. For example, the `ArrayListServices` class uses the letter T instead of E.

Here is some code if you are not familiar with how to create a new ArrayList of generic type T:
```
ArrayList<T> list = new ArrayList<T>();
```
Refer to the text and lecture material for more on generics, or this information.

Gauging Data Equality
It is necessary to have the ability to gauge equality between data objects in order to perform certain list specialized operations. For example, in order to remove duplicate records from a list, we need to be able to detect when two data objects are the same, or equal to each other. Recall that there are two ways to detect equality in Java: use of the equality operator, "==", and use of the `equals` method. When applied to Objects, the equality operator returns true if the Objects are the same; i.e. they have the same memory reference. This ignores the data that is stored in the Object, and is not a suitable technique for our purposes as we want to use class data to gauge equality. Therefore, we use the `equals` method.

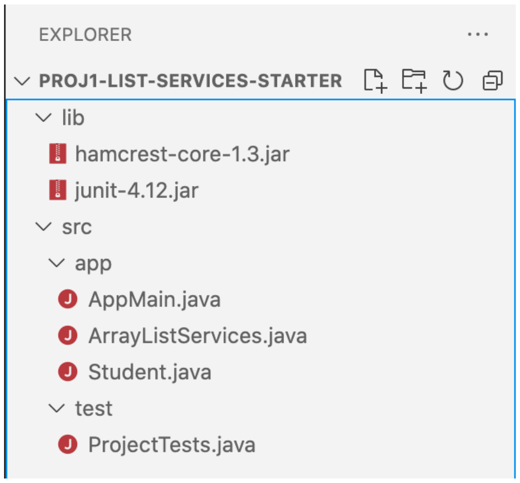
The Java class "Object", the default superclass of all Objects, defines an `equals` method that can be overridden by its subclasses. In this project, we assume that the data stored in the ArrayLists define an appropriate version of the `equals` method. The `equals` method can be customized (overridden) to determine equality between Objects based on the data they contain. Examples of this can be seen in the Java `String` and `Integer` classes. User-defined classes can include their own implementation of the `equals` method, as in the `Student` class included in this project.

Code Structure

This project contains the following important folders:

lib: This is where you can find the two jar files that are used to run the JUnit test framework. You do not need to do anything with these files.
src/app: This is the source folder where all code you are submitting must go. All of your tasks are in the `ArrayListServices` class for this project. The `AppMain` class contains test code that you should use (with the debugger) to develop your code.
src/test: This is the package folder where you can find JUnit unit tests in the `ProjectTests` class. Run these tests after you have developed your code with the help of the `AppMain` class. This file contains a subset of the tests that will be run on your code in the autograder.

Figure 1 gives the overview of the folders and files given to you in the starter code.



```
EXPLORER
PROJ1-LIST-SERVICES-STARTER
  lib
    hamcrest-core-1.3.jar
    junit-4.12.jar
  src
    app
      AppMain.java
      ArrayListServices.java
      Student.java
    test
      ProjectTests.java
```

Fig 1: Code structure for the project.

Your main task is to complete the implementation of the three methods in the ***`ArrayListServices`*** class. Note that the generic type T should provide an implementation of the `equals` method (see Gauging Data Equality in the Background section above).

Tasks and TODOs

The project TODOs denote where you need to make changes in the starter code given to you. We do this using comments with the word TODO written in the comment. Most IDEs typically recognize TODO comments and provide an interface for navigating each TODO. VSCode has the ***Todo Tree*** extension which you can install if you haven't already done so.

Do not modify any method signatures or class variables in the `ArrayListServices` class. You may add any private variables or methods as you wish. You may modify any code elsewhere in the project, such as `AppMain`.

As stated above, all of the TODOs are in the `ArrayListServices` class. There are three TODOs in that file, each asks you to implement a method according to the specification in the comments and as stated in the **Tasks** section below.

Tasks

There are three TODOs that you will implement, each corresponding to a method in the `ArrayListServices` class.

You may write any private methods you wish to support your code, though that is not likely to be necessary for this project. It is recommended that you call other methods that you have written, if they accomplish a useful task in support of the method you are currently writing. That avoids code duplication and makes your code easier to read and understand, not to mention test and debug.

1- The ***`deduplicate`*** method.
This method takes an ArrayList as a parameter and returns a new ArrayList that does not contain any duplicate data. For example, if this list was passed in: [A, B, D, A, A, E, B], the method would return a list containing: [A, B, D, E]. The ordering of the data does not matter.

Your method can assume that the parameter is not null, but it may be an empty ArrayList, in which case your method returns a new, empty ArrayList. Also note that the ArrayList that is returned must be a new ArrayList which is not the same as the ArrayList passed in. In other words, the parameter must not be changed by your method code.

2- The ***`getSetIntersection`*** method.
This method takes two ArrayLists as parameters and returns a new ArrayList that contains the intersection of the data in the ArrayLists passed in. The intersection contains the elements that occur in both lists. For example, if these lists were passed in: [A, B, D, A, A, E, B], [B, E, C], the method would return a list containing: [B, E]. The ordering of the data does not matter. Note that the result **does not contain any duplicates**.

Your method can assume that the parameters are not null, but one or both may be an empty ArrayList, in which case your method returns a new, empty ArrayList. Also note that the ArrayList that is returned must be a new ArrayList which is not the same as the ArrayList passed in. In other words, the parameter must not be changed by your method code.

3- The ***`getSetDifference`*** method.
This method takes two ArrayLists as parameters and returns a new ArrayList that contains the set difference of the data in the ArrayLists passed in. The set difference contains the elements that occur only in one or the other list, but not in both. For example, if these lists were passed in: [A, B, D, A, A, E, B], [B, E, C], the method would return a list containing: [A, D, C]. The ordering of the data does not matter. Note that the result **does not contain any duplicates**.

Your method can assume that the parameters are not null, but one or both may be an empty ArrayList. In the case where one list is empty, your method returns a new ArrayList that contains all of the elements on the other list- with no duplicates. In the case where both lists are empty, your method returns a new, empty ArrayList. Also note that the ArrayList that is returned must be a new ArrayList which is not the same as the ArrayList passed in. In other words, the parameter must not be changed by your method code.

Style Guide:

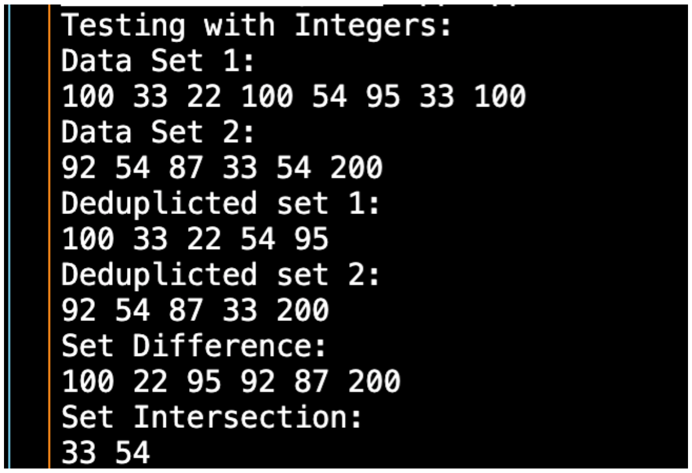
Follow these style guidelines that will be manually graded in the project.

1. Declare variables and initialize where appropriate and practical.
2. Use descriptive variable and method names.
3. Use proper indenting and bracketing style.
4. Use blank lines between methods and spaces for operators.
5. Use camelCase for variable/parameter and method names.
6. Remove all unnecessary lines of code.
7. Use comments for methods to describe purpose, inputs, and returns of methods where applicable.
8. Use comments for variables: comments describing the meaning/purpose of non-trivial and non-obvious variables.

Testing

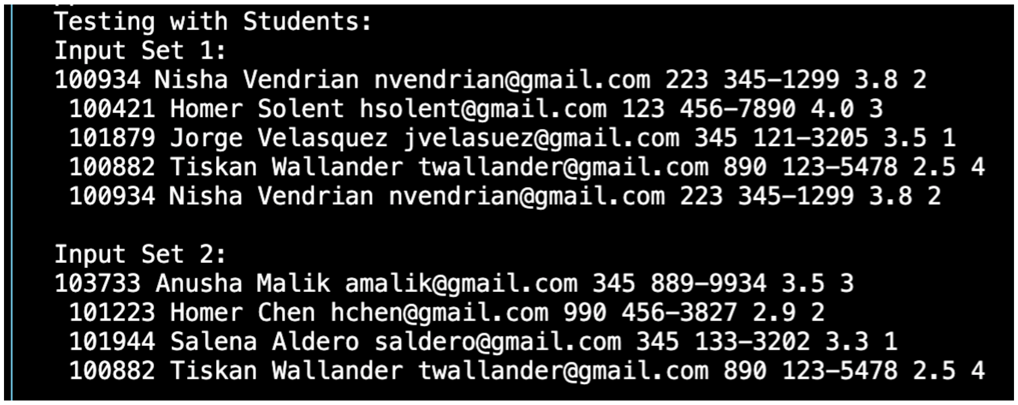
Develop your code one TODO at a time. Use the `AppMain` class to test and debug your code. You may need to comment out parts of this code as necessary. Note that the main method throws an exception when you initially run it.

Here is the correct output when you run `intTest()` in `AppMain` class.

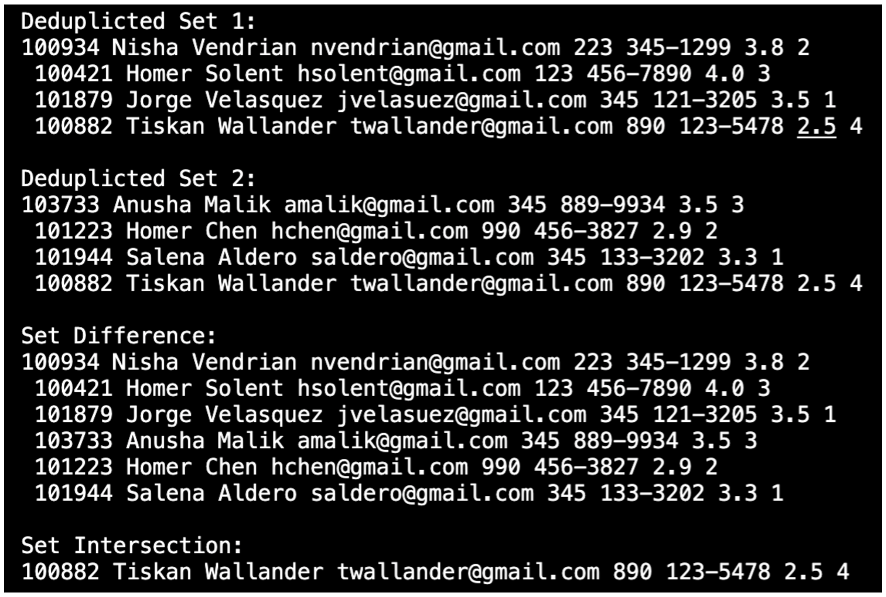


```
Testing with Integers:
Data Set 1:
100 33 22 100 54 95 33 100
Data Set 2:
92 54 87 33 54 200
Deduplicted set 1:
100 33 22 54 95
Deduplicted set 2:
92 54 87 33 200
Set Difference:
100 22 95 92 87 200
Set Intersection:
33 54
```

Here is the correct output when you run `studentTest()` in `AppMain` class.



```
Testing with Students:
Input Set 1:
100934 Nisha Vendrian nvendrian@gmail.com 223 345-1299 3.8 2
 100421 Homer Solent hsolent@gmail.com 123 456-7890 4.0 3
 101879 Jorge Velasquez jvelasuez@gmail.com 345 121-3205 3.5 1
 100882 Tiskan Wallander twallander@gmail.com 890 123-5478 2.5 4
 100934 Nisha Vendrian nvendrian@gmail.com 223 345-1299 3.8 2

Input Set 2:
103733 Anusha Malik amalik@gmail.com 345 889-9934 3.5 3
 101223 Homer Chen hchen@gmail.com 990 456-3827 2.9 2
 101944 Salena Aldero saldero@gmail.com 345 133-3202 3.3 1
 100882 Tiskan Wallander twallander@gmail.com 890 123-5478 2.5 4
```



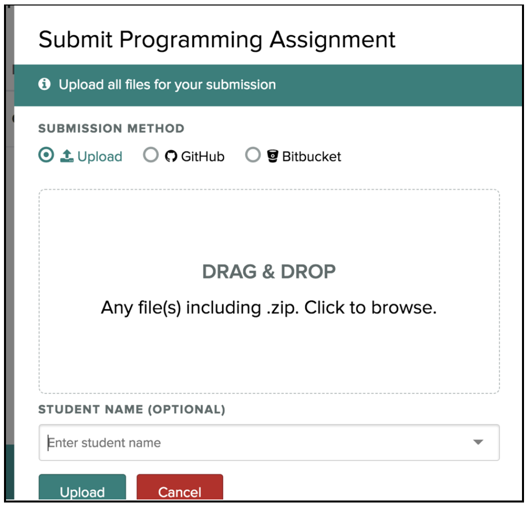
```
Deduplicted Set 1:
100934 Nisha Vendrian nvendrian@gmail.com 223 345-1299 3.8 2
 100421 Homer Solent hsolent@gmail.com 123 456-7890 4.0 3
 101879 Jorge Velasquez jvelasuez@gmail.com 345 121-3205 3.5 1
 100882 Tiskan Wallander twallander@gmail.com 890 123-5478 2.5 4

Deduplicted Set 2:
103733 Anusha Malik amalik@gmail.com 345 889-9934 3.5 3
 101223 Homer Chen hchen@gmail.com 990 456-3827 2.9 2
 101944 Salena Aldero saldero@gmail.com 345 133-3202 3.3 1
 100882 Tiskan Wallander twallander@gmail.com 890 123-5478 2.5 4

Set Difference:
100934 Nisha Vendrian nvendrian@gmail.com 223 345-1299 3.8 2
 100421 Homer Solent hsolent@gmail.com 123 456-7890 4.0 3
 101879 Jorge Velasquez jvelasuez@gmail.com 345 121-3205 3.5 1
 103733 Anusha Malik amalik@gmail.com 345 889-9934 3.5 3
 101223 Homer Chen hchen@gmail.com 990 456-3827 2.9 2
 101944 Salena Aldero saldero@gmail.com 345 133-3202 3.3 1

Set Intersection:
100882 Tiskan Wallander twallander@gmail.com 890 123-5478 2.5 4
```

We provide some unit tests as well. Run `ProjectTests.java` (in the test folder). These should be run after you have completed all of the code. Use the debugger to step through your code when troubleshooting. This is a key skill for all programmers to be comfortable with. ***Remember: Do not modify any existing instance variables or method signatures.***

Export and Submit

Step 1: Export your project
Within VSCode click on the "*View* > *Command Palette*…" menu option. Then type into the Command Palette: "*Archive Folder*" and hit enter. This will produce a zip file of your project folder. You can then upload that zip file to the corresponding project assignment in Gradescope. You can add the Archive extension to VSCode if you don't have it.

Step 2: Submit the zip file to Gradescope
Log into Gradescope, select the assignment, and submit the zip file for grading.


Submit Programming Assignment

Upload all files for your submission

SUBMISSION METHOD

[✓ Upload] GitHub Bitbucket

DRAG & DROP

Any file(s) including .zip. Click to browse.

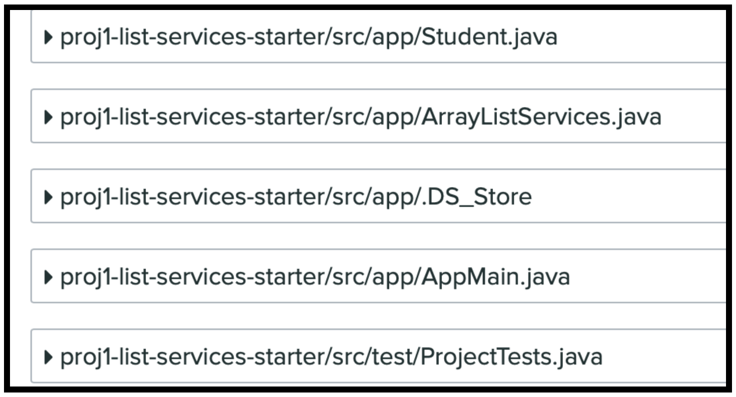
STUDENT NAME (OPTIONAL)

Enter student name

Fig 2: Gradescope submission window for the project.

Important

Your top level folder must be named: **`proj1-list-services-starter`**
See the figure below that shows files in Gradescope after **`proj1-list-services-starter.zip`** has been uploaded.


proj1-list-services-starter/src/app/Student.java

proj1-list-services-starter/src/app/ArrayListServices.java

proj1-list-services-starter/src/app/.DS_Store

proj1-list-services-starter/src/app/AppMain.java

proj1-list-services-starter/src/test/ProjectTests.java

Fig 3: Gradescope code window after the zip file is uploaded.

If your zip file is not in the correct form, the autograder will not process your code and you will not receive any credit. The autograder will not run (often stating an error occurred) if your code does not compile or if your zip file is not correctly made.

- The autograder will not run successfully if you do submit a **correctly formatted zip file**- it has to have the same **names and directory structure as described above**.
- The autograder will also not run if your code **does not compile**, or if you i**mport libraries** that were not specifically allowed in the instructions.

Note that we will not grade code manually, only for style. If your code does not compile or if your zip file is not correctly structured you will not receive any credit for the project. Do not import any libraries or change the starter code method signatures.

There are usually more tests in the autograder than provided with the starter code. If your code passes all provided tests, it is a good indication that your code will pass all of the autograder tests. If that is not the case, you are likely not considering some of the “edge” cases in your algorithm. Re-examine your code, use the debugger to troubleshoot. Pose specific questions on CampusWire for some outside help. Remember that these projects take longer than you estimate. Starting early means you have more time to get help if you are stuck.

Grading Rubric:

Rubric	Points
Autograded	
Style/Aesthetic @1 each Declare variables and initialize where appropriate and practical. Use descriptive variable names. Use proper indenting. Use consistent bracketing style. Use blank lines between methods. Use camelCase for variable/parameter names. Remove all unnecessary lines of code.	/7
Comments: Use comments for methods to describe purpose, inputs, and returns of methods where applicable. @3 Use comments for variables: comments describing the meaning/purpose of non-trivial and non-obvious variables. @2	/5

Remember, you can re-submit the assignment to gradescope as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does.