Homework 1C (CS 17/111/112/19): Lists and Mutation

Due: Thursday, February 9th, 2023 at 11:59pm ET

> ***Note:*** In order to complete this assignment, you must have already completed HW1B. In this homework, we'll be building on top of classes that we've created in the previous homeworks, and you will need to copy your class implementations from previous assignments.

Collaboration Policy: *You may collaborate as much as you want on this assignment* (this is more flexible than the normal course policy). The goal is to get everyone up to speed on functional programming. You are required to turn this in, but it will be weighted lightly in final grades. That said, we strongly encourage you to actively try writing these on your own while collaborating, as assignments after we come back together will assume you can do these sorts of problems on your own.

Learning Objectives

- To practice working with Java's built-in lists
- To practice structuring code around access modifiers
- To practice using Java types to control the ability to perform certain computations

Setup

Using this GitHub Classroom link, accept the assignment and clone on your machine. (This assignment should be a continuation of homework 1B, you can use your code from homework 1A, and homework 1B as a start).

If you're having trouble with IntelliJ, our First Time Setup Guide includes a common bugs/FAQ section.

Please make sure to read our Gradescope submission guide before handing in your assignment!

Style Expectations

For all future assignments, including this one, we expect your code to conform to the CS200 Java Style Guide.

Important: When comparing fields between classes, it is good object-oriented code organization to not use getters and setters. This means that your classes should not have methods where the only purpose is returning an inputted field ("getting" a value") or changing an inputted field's value ("setting" a value).

Problems

In the last assignment, we created a `GradeReport` class. Now, it's time to set up the gradebook for an entire class, as well as the ability to manage (and update) grades. We also want to support `Faculty` teaching multiple courses.

More Courses

Task 1: Edit your `Faculty` class so that the field that stores what they are teaching is a `LinkedList` of `Course` rather than a single `Course`. You'll have to update the constructor and perhaps other methods that worked on the previous field type as well.

Reminder that to use this, you will need the following import statement at the top of your `Faculty.java`:

```
01|  import java.util.LinkedList;
```

If you need to check whether a specific `Course` object is in a faculty member's list of courses, use the `LinkedList` method `contains` which takes an object and returns a boolean.

Creating a Gradebook

Task 2: Add a `gradebook` field to the `Course` class. It should be of type `LinkedList<GradeReport>`. This field should be initialized to an empty list within the `Course` constructor.

Don't forget to import `java.util.LinkedList` in `Course.java`!

Adding Grades

Task 3: Add a method named `storeGrade` to the `Course` class. The method should take in a `Student` and a valid grade (`LetterGrade` or `SNCGrade` from assignment 1B). The method should result in updating the `Course`'s `gradebook` to include a `GradeReport` for the corresponding student. You may assume that the student doesn't already have a grade in the gradebook.

Looking up GradeReports

Task 4: Create a method `findGradeReport` in the `Course` class that takes the name of a student and returns the `GradeReport` of the named student in the course. The method should return `null` if the student is not in the course. Mark this method as `private` (replacing `public` with `private` at the start of the method header.)

Updating Grades

Task 5: Add a method named `updateGrade` to the `Course` class. The method should take in the name of a student (String) and a valid grade. The method should result in the new grade being stored in the gradebook for the given student. If the student does not have a grade, throw a `RuntimeException` with the message `"no grade for student"`.

Think about how to use methods you've previously written as helpers.

Note: Java has different kinds of exceptions for different kinds of errors. For now, we will use a `RuntimeException` for any error related to the logic of the program or system being represented (as opposed to `IllegalArgumentException` which we use when an input is not from an expected set of values).

```
throw new RuntimeException(<insert error message>);
```

Hint:

Finding Students in a Gradebook

Task 6: Add a method called `gradeLookup` to the `Course` class. The method should take a `Faculty` or `TA` object and a student name as input.

- If `gradebook` has a `GradeReport` for the given student name, the method should return a `String` representation of the grade.
- If the given `Faculty` or `TA` is not teaching the course, throw a `RuntimeException` with the message "`permission denied`".
- If no such grade is found, a `RuntimeException` with the message "`student <NAME> not found`" should be thrown (where "`<NAME>`" is replaced with the student name being searched for).

In cases where both exceptions would apply, throw the `permission denied` exception.

Think about how to reuse methods that you've already written.

*Part of the point of the question is for you to think about how to take an argument that can be `Faculty` or `TA`, but not a general `Student`. (We've covered everything you need in order to do this. **You may NOT use `instanceOf`**).*

Identifying Students in Trouble

Task 7: Add a method `reportsWithC` to the `Course` class. The method should return a list of `GradeReport` from the course `gradebook` in which the grade is the letter `C`.

*The point of this question is for you to practice using good object-oriented code organization to check attributes of objects. **You should NOT use getters to do this** (by this, we mean that your code should not pull a grade out of a gradebook to compare it to "C").*

Note: In order to write this, you will need to be able to check whether two grades are the same. Lab 2 gave a quick introduction to writing `equals` methods, but IntelliJ will also generate `equals` methods for you automatically. If you start typing `public boolean equals` in a class, IntelliJ will give you a popup that asks whether it should generate "equals and hashcode methods". Click through doing that (just ignore or delete the generated `hashcode` method).

Hint: The Equals methods for `LetterGrade` and `SNCGrade` ->

```
01|     @Override
02|      public boolean equals(Object o) {
03|          if (this == o) return true;
04|          if (o == null || this.getClass() != o.getClass()) return
   false;
05|          LetterGrade that = (LetterGrade) o;
06|          return this.grade == that.grade;
07|      }
```

```
01|    @Override
02|      public boolean equals(Object o) {
03|          if (this == o) return true;
04|          if (o == null || this.getClass() != o.getClass()) return
   false;
05|          SNCGrade sncGrade = (SNCGrade) o;
06|          return this.pass == sncGrade.pass && this.passDistinction
   == sncGrade.passDistinction;
07|      }
```

Protecting Grade Information

Task 8: Grade information is sensitive, and hence should be kept private. Other information may also be sensitive. Mark all of the fields in `GradeReport`, `Student`, `Faculty`, and `Course` as `private`. If this breaks some of your existing code, create additional methods as needed to restore your code to working.

Hint:

Note: You will learn about `private` annotations in lab this week. In a nutshell, marking a field `private` means that the field cannot simply be accessed from outside the class by writing `object.field`. Instead, you have to provide a method in the field's class that will perform the desired computation (you should not just hand the field out through a simple "getter" method, for those who already know that term).

Testing

Note: When writing your test suite, you should test only the methods we have asked you to write in this handout. Do NOT reference in your tests fields or methods that are not required by the assignment, or the autograder may fail.

Put your testing in the file `Homework1CTest.java`. You will submit this file to the **Homework 1C (17/111/112/19): Implementation** assignment on Gradescope.

Task 9: Write a thorough set of tests for `gradeLookup`. You should develop a set of tests that check the expected interactions among `storeGrade`, `updateGrade`, `gradeLookup`, and `reportsWithC`. What do we mean? One might expect that `reportsWithC` (for example) could return a different output after a call to `updateGrade`. Good test suites therefore test the interactions between methods when the changes made by one should affect (or not!) the output that comes from the other. We're asking you to develop a set of such interaction tests for these four functions.

Note: **You cannot test `private` methods in `Homework1CTest.java`** (because the test class cannot "see" private methods). You may want to test that method before you make it private, but do not include tests for private methods in your `Homework1CTest.java` file.

Pro Tip: If you find that you need to set up the same courses, students, etc for multiple tests, you can write a single method to set up your data and tell JUnit to run it before each test method. Here's an example:

```
01|  public class Homework1CTest {
02|    Course c1, c2;
03|
04|    @Before
05|    public void setup() {
06|      this.c1 = new Course("VISA", 100, 1)
07|      this.c2 = new Course("HIST", 310, 1)
08|    }
09|
10|    @Test
11|    public void testDept() {
12|      Assert.assertEquals(c1.department,"VISA");
13|    }
14|  }
```

We create the variables for data as fields outside the methods. The `@Before` annotation tells JUnit to run that method before running every test. This way, your tests can focus just on the conditions to check, and you can set up the data just one time. This also restores the values of c1, etc before each test, in case you have a test that needs to modify your data.

Handing In

Begin by cloning the GitHub repository found by clicking the above GitHub Classroom link. The following files should be in the `sol` directory. You can move the files you wrote in 1B into this directory.

- **`Course.java`** containing public class `Course`

- **`Faculty.java`** containing public class `Faculty`
- **`Student.java`** containing public class `Student`
- **`Person.java`** containing public abstract class `Person`
- **`TA.java`** containing public class `TA`
- **`GradeReport.java`** containing public class `GradeReport`
- **`IGrade.java`** containing public interface `IGrade`
- **`Letter.java`** containing public enum `Letter`
- **`LetterGrade.java`** containing public class `LetterGrade`
- **`SNC.java`** containing public enum `SNC`
- **`SNCGrade.java`** containing public class
- **`Homework1CTest.java`** containing public class `Homework1CTest`
- **`TestRunner.java`** containing public class `TestRunner`
- **`AutograderCompatibility.java`** containing public class `AutograderCompatibility`

After completing this assignment, you should be ready to turn in (at least) the following files to the **Homework 1C-17/111/112/19: Implementation** assignment on Gradescope (make sure to exclude the **`AutograderCompatibility.java`** file from your submission):

- **`Course.java`** containing public class `Course`
- **`Faculty.java`** containing public class `Faculty`
- **`Student.java`** containing public class `Student`
- **`Person.java`** containing public abstract class `Person`
- **`TA.java`** containing public class `TA`
- **`GradeReport.java`** containing public class `GradeReport`
- **`IGrade.java`** containing public interface `IGrade`
- **`Letter.java`** containing public enum `Letter`
- **`LetterGrade.java`** containing public class `LetterGrade`
- **`SNC.java`** containing public enum `SNC`
- **`SNCGrade.java`** containing public class `SNCGrade`
- **`Homework1CTest.java`** containing public class `Homework1CTest`

Note: You may also include other classes, interfaces, and abstract classes.

Once you have handed in your homework, you should receive an email, more or less immediately, confirming that fact. If you don't receive this email, try handing in again, or ask the TAs what went wrong.

Note: There should be a class in the stencil code named `AutograderCompatibility`. Using this class is required to ensure that your submission is working correctly with the autograder. You will be penalized if your code does not work with the autograder.

If Gradescope gives you the message *"Could not run your code (0.0/0.0),"* uncomment the main method of `AutograderCompatibility` and check that it compiles and runs. If the Gradescope autograder still doesn't work, come to hours or post on Ed for help.

FAQ

For examples of tests, see the Homework 1A FAQ. If you have other questions, feel free to post on Ed or come to hours.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS200 document by filling out our anonymous feedback form!