CMSC 132 Project #1 Spring 2015

Date due: Thursday, February 12, 10:00:00 p.m.

1 Introduction

In this project you will write some methods that use features of Java that were covered in CMSC 131. You will also become familiar, if you aren't already, with using Eclipse to code in Java, and submitting programming assignments to the CMSC project submission server. Lastly, you will write tests of your own that exercise your methods in ways that the provided public tests discussed below do not. The methods should not be that hard to write, but if you have questions about how to write the project come to the TAs' office hours right away.

This description is lengthy because the first programming assignment needs to explain various details that apply to all the remaining projects. Notice that the actual description of the methods you have to write is less than half of this document.

Read the following three handouts on ELMS (under Pages and then Administrative) carefully:

- The programming assignment submission and grading policy, which describes how programming assignments should be written and will be graded.
- The project style guide, which explains what good programming style is considered to be for this course. Note that a significant part of this project grade will be for style, so not using good style, as defined in this handout, could result in losing a lot of credit.
- A JUnit handout, that explains how to write JUnit tests (like your own student tests), and to run JUnit tests (like your student tests, and the public tests).

Read this entire assignment carefully before starting to code. Also reread it again after you finish the project, to ensure you didn't miss anything important.

Due to the size of the course it is not feasible to provide information or assistance regarding programming assignments via email. You are welcome to ask any questions you may have about this project in person, either during office hours, or during, before, or after discussion section or lecture (if time permits).

As the programming assignment submission and grading policy handout mentioned above discusses, any project can be submitted up to two days late (with a late penalty for each day late). However, this project is due on a Thursday, and there are no course office hours on weekends, so if you don't finish it by the one–day–late deadline you will be on your own trying to figure out any problems on the last day, which will be a Saturday. Consequently you are urged to start on it **right away**.

2 Brief description of problem

The sections after this explain the methods you are to write, how to access the project starting code, submit your code, etc.

This project involves writing some methods to simulate the registration system for a small university. A university has to keep track of different courses being offered, as well as the students who are registered for each one. Courses can be taught in different departments, and each course has a seat count, or a maximum number of students who can be enrolled in it. The problem isn't that interesting, but it does involve using basic concepts from CMSC 131, such as the `ArrayList` class, and objects that contain and use other objects.

The starting code that you will check out from your CVS repository contains a `University` class that you will use to implement a university. You can add whatever fields and write whatever methods you need in the `University` class– given the constraints in Section 6– that will allow you to write the required methods described here. You may also add any other classes to the project, but they **must** be located in the `university` package (or the `tests` package, but only classes related to your student tests (described below) should be put there). Note that one constraint in Section 6 is that you **must** use an `ArrayList` in at least one place for storing data in your program. It can be directly in the `University` class, or it can be in some other class that you add, as long as it's storing some part of the necessary data.

3 Description of methods to be written

Each method you have to write in the `University` class just throws an `UnsupportedOperationException` in the skeleton version of the class provided to you. Remove the exceptions and write the body of each method. All the methods you have to write are public.

Most of the methods have one or more objects as parameters (they are `String`s). You can assume these will never be `null`, so you do not have to check for that situation or handle it in any particular way. You can also assume that any strings passed as arguments to any methods will not be empty strings.

Suggestion– look at some of the public tests (discussed below) before starting to code, so you understand how your methods are going to be used and called.

3.1 `University addCourse(String department, int number, int numSeats)`

This method should add a course to its current object `University`. Its parameters represent the department of the course being added (e.g., "CMSC", "ENGL", etc.), the number of the course being added, and its seat count or enrollment limit. Due to the assumptions given , the method may assume that `department` is neither `null` nor an empty string.

The method should return a reference to its current object, to allow chained calls, for example, a call like `university.addCourse("CMSC", 132, 275).anotherMethod()`, where `anotherMethod()` might be one of the methods described below (or even another call to `addCourse()`). If there is already a course in the current object `University` that has the same number and is being taught in the same department, the method should not have any effect (should not add anything), and should just return a reference to the unmodified current object. Otherwise it should add a course with the department, number, and seat count indicated by its parameters to the current object, and return a reference to the modified current object.

There is no limit to the number of courses that can be added to a university, so your method should be able to be called any number of times on the same `University` object. No two courses in the same department can have the same number, but courses in different departments can have the same number, and there can be different courses in the same department, as long as they have different numbers.

It is not incorrect if the value of `numSeats` is zero or negative in a call to `addCourse()`. It is kind of a useless course in that case, because no students will ever be able to register for it, but the university is allowed to create it. Any attempt to add a student to it later though, by calling `add()`, will always just return false without adding the student.

Note that your code should allow multiple `University` objects to be created and manipulated simultaneously, and their data should not conflict. Each university stores a list of its own courses, and adding a course to one university has no effect on the courses at any other university.

The way that courses are stored in a `University` object is up to you, subject to the constraints in Section 6, so be sure to read that section carefully.

3.2 `boolean cancelCourse(String department, int number)`

This method should remove the course with number `number` being taught in department `department` from its current object `University`. This would be done if the course is canceled for any reason, such as insufficient enrollment, or when the semester ends and the course is over. As described below, courses can have students enrolled in them– after a course is canceled, all of the students it had are not registered for it any longer. The method should return true after removing the course, but if the current object does not have a course in the department `department` with number `number` it should not modify anything, and just return false instead.

When this method is called the university may have other courses in the same department with different numbers, or other courses in different departments with the same number, but those will not be affected. Only the course `number` in department `department` (if it exists) will be removed by this method.

If a course is canceled, another course with that same number in the same department could be re–added to a university later.

3.3 `int numCourses()`

This method should return the number of courses that currently exist in (have been added to) its current object `University`, which should always be a nonnegative number.

3.4 `boolean add(String department, int number, String name)`

This method should add a student to the course with number `number` being taught in the department `department` in the current object `University`. The university that your code is simulating is just a small university, so small that it does not have any students in the university who have the same first name, so this method only needs to have one name to uniquely indicate a student (their first name).

If a student with first name `name` can be added to the indicated course in the current object, the method should add them, and return true. There are several cases in which a student cannot be added, however.

As mentioned, the university that your code is simulating is just a small university, that does not have any students enrolled with identical names. If `add()` is ever called when the course `number` in department `department` already has a student with name `name`, the method should not have any effect, and should just return false (because it means that the same student is already enrolled for the course). It should also return false without changing anything if the current object `University` is not currently storing a course with number `number` in department `department`. The university that will use your program also does not allow students to register for more than five courses, so the method must return false without changing anything if the student with name `name` has already added five other courses (regardless of what department or departments they are in). Lastly, no course can have more students in it than the number of seats that the course was created with when it was added to the university, indicated by the value of the third parameter to `addCourse()`. If course number `number` in department `department` is already full, meaning that it already has that many students enrolled, the method should also return false without changing anything.

Suggestion: as you can see, there are various situations or special cases in which the method should not add the indicated student to the course. It might be easiest to just write the method for the normal situation first, meaning get it to work assuming that the add will always succeed. You might even write the rest of the methods below next, because some of them might be useful in getting the functionality of the special cases to work. Then go back and add code to this method to handle the special cases in the previous paragraph.

3.5 `int numStudentsInCourse(String department, int number)`

This method should return the number of students currently in course number `number` in department `department` stored in its current object `University`. A course that is present in a `University` should always have a nonnegative number of students, but if the current object does not have a course with number `number` in department `department` the method should return -1 instead.

3.6 `boolean isRegisteredForCourse(String department, int number, String name)`

This method should return true if there is a student with name `name` who is currently registered in course number `number` in department `department` stored in the current object `University`. It should return false if there is no course with number `number` in department `department` currently stored in the university, or if there is such a course but there is no student with name `name` currently registered in it.

3.7 `int numCoursesRegisteredFor(String name)`

This method should return the number of courses that the student with name `name` is currently registered for in the current object University. Due to the behavior of the `add()` method above this should always be a number between 0 and 5 (inclusive). If there is no student with name `name` in any courses in the university the method should just return 0.

3.8 `boolean drop(String department, int number, String name)`

This method should remove or drop the student named `name` from course number `number` being taught in department `department` that is stored in the current object `University`. If the current object `University` does not currently have a course with that number in that department, or if there is a course with that number in that department but it doesn't have a student in it who has name `name`, the method should not modify anything and just return false. Otherwise it should remove the student from the course and return true.

Due to the behavior of the `add()` method described above, there should never be two or more students with the same name in any course in a `University` object, so this method will only remove (at most) one student. Removing a student from one course does not affect other students who are also in that course, and it also doesn't remove that student from any other courses that they are registered for.

If course `number` in department `department` is currently full and this method is called to drop a student from it, it opens up a space so another student can be subsequently added. And if the student with name `name` is currently registered for five courses, meaning that they would not be able to add a sixth (due to the behavior of the `add()` method described above), and this method is called to drop the student from one of their courses, they would be able to add some other course after that (or even re–add the same course later).

3.9 `boolean cancelRegistration(String name)`

This method should remove or drop the student with name `name` from **all** of the courses that they are currently taking in the current object `University`. If a student named `name` is in the university when the method is called, meaning they are currently registered for at least one course, the method should remove them from all of their courses and return true. If there is no student with name `name` in any courses in the university when the method is called, the method should not modify anything and just return false.

If a student cancels their registration they can still re–enroll in courses later (just by calling `add()` with their first name). And if a student cancels their registration, it opens up a space in all of the courses that they had previously been registered for.

3.10 Notes

For all methods that have a student name as a parameter that have to check whether a student is registered for a course, case is significant for the name– it must match the name of a student exactly, including capitalization, to find a student in a course. And the department name has to match in case exactly for methods that have to compare names of departments. For example, if a student with name "Sally" is registered for a course with number 132 in the department "CMSC", then it is not a problem to add a student with name "sally" to the same course, because the two students are not considered to be the same one (so the add should succeed). And the student named "Sally" can be registered for "CMSC" 132 and also "Cmsc" 132 at the same time, because the departments "CMSC" and "Cmsc" are not considered to be the same.

4 Checking out the project starting code

You need to import into Eclipse the starting code for the project, which contains the tests (public tests) your project will be run on, as well as skeletons of the classes that you have to write. It's assumed that you have already installed the correct versions of Java and Eclipse, and set up your CVS repository connection in Eclipse, as described in the handout "Eclipse CVS repository setup for CMSC 132" available under "Pages" and then "Administrative" on ELMS. If you haven't yet, you must do that first. After that:

1. Start Eclipse and check out the starting code for the project **proj1** from your CVS repository the same way you checked out the project `example-programming-assignment` in Section 5 of the Eclipse CVS repository setup handout mentioned above.
2. To start working on the project, change to the Java perspective using **Window** → **Open Perspective** → **Other** → **Java (default)**, then clicking on `proj1`. (If **Java** appears near the upper–right of the Eclipse window you can also change to the Java perspective by clicking there.)

3. Click on the plus sign or arrow next to the `proj1` project to expand it, then on the plus sign or arrow next to the `university` package to expand it. Then double–click on the `University.java` class file to open it, where you will write the methods described above.

Note that you only need to use the CVS perspective once for each assignment to be written, although if you use different computers you will need to check out an assignment on each one you use.

5 Submitting programming coursework

To submit a programming assignment you **must** have installed the Course Management Plugin Eclipse plugin (or installed the Eclipse version for Windows with the plugin already present). As explained in the CVS repository setup handout, to submit this project just switch to the Java perspective, right–click on the `proj1` project in the package explorer, and select **Submit Project...** The first time you submit an assignment you'll have to enter your UMCP directory ID and password.

As the programming assignment grading policy on ELMS emphasizes, **do not** use the submit server's web submission procedure to submit a jarfile or zip archive, or a list of files. For this project, doing that could result in losing up to 30 points from your score. If you have trouble submitting assignments using the required procedure described here, you can get help by coming to office hours.

Once you submit, you **must** log in to the CMSC department project submission server, reachable by clicking on `https://submit.cs.umd.edu`. We can only grade what you submit to us, based on how it works on the project submission server, so (as the programming assignment submission and grading policy discussed), you **must** be sure to log into it right after you submit to check your assignment's results there.

5.1 To receive your graded project results

If you want to receive your graded project results back, you **must** do the following. (Yes, this is a silly procedure, and yes, you have to do it. It's just due to the way the ELMS gradebook works.) This process will allow us to upload your graded project to ELMS (as a PDF) when it's ready to be returned to you, where you'll be able to see it, so if you don't do this you will get a grade for the project, but you will not have any way to see the details of your grade, what points you lost, feedback the grading TAs made, etc.

- Go to the course's Projects page in ELMS. Download the small PDF file there that's named `file.pdf`.
- **Rename** that file as `proj1.pdf`. **Note:** its name has to be spelled and capitalized **exactly** as shown, otherwise you will not receive your graded project results.
- Go to the course's ELMS page and click on **Assignments** in the top row of the page, then click this assignment (Project #1). Click on **Submit assignment** on the top near the right of the page. On the page that appears, click on **Browse** under "File Upload", select the file `proj1.pdf` that you renamed, and click on the blue **Submit Assignment** button at the bottom. ELMS will say "Submitting...", and on the next screen you will see "Turned In!" near the right.

Submitting this PDF file doesn't count in your project score. You only need to do this once for each project, so if you resubmit an improved version of your code to the submit server later, it's not necessary to upload this PDF file to ELMS again. Note: **do not** upload your Java project code to ELMS this way– it must be submitted to the submit server using the **Submit Project** menu option in Eclipse, as described above.

6 Project requirements and related issues

6.1 Allowed, required, and prohibited features

1. As mentioned in Section 2, you **must** use an `ArrayList` in at least one place in your university data structure, or you will lose credit when your project is graded. The `ArrayList` can be used directly in the `University` class, or it can be in some other class that you add, as long as it's storing some part of the university, course, or student data. You can use more than one `ArrayList`, and you can also use as many regular Java arrays as you like.

2. To keep things simple for now, you should use **only** the features of Java covered so far this semester and in CMSC 131 in writing this project. If you need to store multiple data items you may use Java arrays, and the `ArrayList` class, but **do not** use any other Java collections (data structures). (Note: don't worry if you don't know any other Java collections yet.)

   This means that the **only** Java library classes that you can use are `ArrayList` and `String`– nothing else. Of course you can use arrays, and any primitive–type variables. (You can use any Java library classes that you like in writing your student tests.)

3. You may add any fields or methods to the `University` class as long as they are **not visible outside of the class**. The only exception is that it's fine to add a public `toString()` method, for debugging purposes.

4. You may add any classes you find necessary to the project (in the `university` or the `tests` packages). Note that the project style guide says that fields and methods should only be made accessible outside of a class when they have to be, so don't provide access to anything outside of your classes unless it's essential.

5. Do **not** add any other packages to the project, or change the package structure in the code as it is initially in your CVS repository, since things won't work right when your project is graded if you do. For this project, doing that could result in losing up to 30 points from your score.

6.2 Tests and testing

1. The JUnit handout on ELMS (under Pages and then Administrative) explains how to write JUnit tests, and to run your program on JUnit tests, like the public tests and your own tests). Be sure to read it unless you're sure you already know this from taking the preceding course here

2. The `tests` package contains the Java source file `PublicTests.java`, which are the public tests for this project. Look carefully at the public tests and see what they are testing– whatever they are not testing is behavior that secret tests might test, so you should write student tests of your own for those cases, to ensure that your program produces correct results for them. You can use the public tests as a guide in writing your own student tests, although some of your tests may be need to be more detailed, longer, or more complex than any of the public tests, to test situations not tested in the public tests.

3. We created an empty class `StudentTests.java` in the `tests` package in the code given to you. You must write your student tests in that class.

4. Even for methods that have been tested in public tests you should still write your own student tests. The public tests are (intentionally) not exhaustive, and do not test methods in all the situations they need to work in. Your code may still have problems, that could result in losing credit on secret tests, even if it passes all the public tests.

6.3 Submitting and score

1. Your grade for this project will be based on:

public tests	40 points
secret tests	30 points
your own student tests	10 points
programming style	20 points

   Note that style is a significant part of this project's score. To know what good style for this course consists of, and be able to avoid losing credit for it, carefully read the project style guide on ELMS.

2. There is no minimum number of student tests you have to write. However, you should attempt to be as thorough as you can in writing student tests. Ideally they would test every method in every case that you can identify that the public tests do not already test. Student tests that exhibit a reasonable attempt to test your code well should receive full credit.

3. **Do not wait** until the last minute to finish and submit your project! It is strongly suggested that you finish and submit it **at least** one day before the deadline, to allow time to reread this assignment and ensure you have not missed anything that could cause you to lose credit. If you did, this will give you time to correct it. This will also give you time to get any unexpected last–minute problems fixed. (Of course, to finish it early you also need to start on the project early.)

   Although you are encouraged to come to the TAs' office hours if you have questions or problems, what we commonly see in large lower–level courses is many students coming to office hours at the last minute before a due date. This is a very big course. If you are one of dozens of students coming to the TAs' office hours the afternoon the project is due, you will almost certainly not be able to receive any help. It is impossible for a TA to help dozens of students in an hour or two, and the TAs are not required to work more hours than they are being paid for. If there is any chance that you may need help in office hours with the project, you need to start working on it **immediately**, so if you have come to office hours you can do so early.

4. **Do not make unnecessary submissions** of the project, or you may lose credit. See the programming assignment submission and grading policies for full details. To ensure that you do not lose credit, do not submit the project more than ten times.

6.4 Miscellaneous

1. If you have a problem with your code that you can't figure out, and need to come to the TAs' office hours, you **must** have written **at least one student test** that illustrates the problem. Write the **smallest** test you can that illustrates the problem.

   If you have a bug requiring office hours assistance you must also be prepared to show the TAs how you tried debugging your code using Eclipse's debugger, and what results you found in the process.

2. During coding you are encouraged to frequently save your work manually to your CVS repository as described in the last section of the handout about setting up your CVS repository.

3. If you ever get a popup message from Eclipse saying that automatic CVS operations are being disabled, you **must** restart Eclipse immediately. Failure to do so could cause problems with your project.

7 Academic integrity

You are allowed to work together or to provide or receive help from anyone else **only** in installing Eclipse, setting up your CVS repository, and checking out the starting code for this project. **Everything else**– namely your implementations of the methods to be written, **as well as** all student tests– must be strictly **your individual work only**, other than possible assistance from the instructional staff during office hours.

Please **carefully read** the academic honesty section of the syllabus. **Any evidence** of impermissible cooperation on programming assignments, use of disallowed materials or resources, or unauthorized use of computer accounts, **will be submitted** to the Student Honor Council, which could result in an XF for the course, or suspension or expulsion from the University. Be sure you understand what you are and what you are not permitted to do in regards to academic integrity when it comes to programming assignments. These policies apply to all students, and the Student Honor Council does not consider lack of knowledge of the policies to be a defense for violating them. More information is in the course syllabus– please review it now.

The academic integrity requirements also apply to any student tests for programming assignments, which must be **your own original work**. Copying the public tests and turning them in as your student tests would be plagiarism, and sharing student tests, or working together to write them, is prohibited.

