CPS209 Project #1

Choose your own adventure

Preamble:

Our first CPS209 project will have a similar style to the CPS109 project that most of you completed last semester. For those who didn't, the motivation is that computing can be used to solve a wide variety of problems across all kinds of applications. You probably have your own ideas on how the skills you have learned so far in CPS209 can be used to solve your own problems. Thus, for this project, *you* will come up with a problem description and then solve that problem using Java. Importantly, your Java solution will demonstrate many of the object-oriented concepts we've learned so far.

Note that the definition of "problem" is very loose. It can take almost any form (some sort of game is a common choice), as long as the project requirements are met.

OOP Requirements:

As CPS209 is a course about object-oriented programming, your program must demonstrate your understanding of object-oriented concepts. At minimum, you must demonstrate each of the following in your solution:

- **Inheritance:** Your program should consist of at least two classes, one of which inherits from the other. The subclass should meaningfully exhibit a "kind-of" relationship with its superclass.
- **Accessors and Mutators:** You should practice good OOP principles by keeping instance variables private and using public setter and getter methods instead.
- **Constructors:** Your classes should implement their own constructors, and you must demonstrate constructor overloading at least once.
- **Methods:** Every class should implement at least two meaningful methods, not including setters and getters. Static methods also count.
- **Overriding:** You should meaningfully override the toString() and equals() methods in at least one of your classes (but not necessarily the same class).
- **Interfaces:** One of your classes should implement at least one interface. An easy choice would be to implement Comparable (or Comparator).
- **Collections:** Your program should use at least one data structure from the Java Collection hierarchy in a meaningful way. This could be something as simple as an ArrayList. Regular old static arrays do not count.

Non-OOP Requirements:

- **Program size:** Your solution should require at least 100 lines of meaningful code. Do not write inefficient code or needlessly expand your syntax for the sake of hitting 100 lines. If you're having trouble hitting the 100-line requirement, consider expanding the scope of your problem.
- **Comments:** You don't have to comment every line of code, but you should include a block comment before each method giving a brief description of what that method does and why it exists. Using the Javadoc format is suggested but not required.
- **Main method:** Your solution should include a separate class that contains a main() method and nothing more. Name this class ProjectOneTester and place it in a file called ProjectOneTester.java. You main() method, when executed, should demonstrate the basic functionality of your program and demonstrate how your classes work/behave. In other words, when our TAs run your main method, it should be clear to them what your program does.

A note on the word *meaningful*: "Meaningful" implies that this construct plays some useful role in your program. Simple declaring an ArrayList in one of your classes and then not using it is not meaningful and will not satisfy the Collections requirement. Overriding toString() and then simply returning "Hello" arbitrarily is not meaningful, and will not satisfy the toString() requirement. There is some subjectivity here, but as long as you're not trying to game or fake the requirement, you're probably fine

What to do:

1) Come up with a problem you wish to solve and describe it in a paragraph or two. Your description should be technical and precise – no vague handwaving. The clearer the problem description, the easier it will be to implement. This should also include a brief description of each class included in your program.

2) Your program will span several Java files. Separate classes should be in their own files, and the name of each file should match the name of the class it contains. Nested/local classes are of course exempt from this.

3) Solve your problem, satisfying all the requirements outlined above.

4) Paste your program description from part 1) into a block comment directly above your main() method.

Marking Scheme:

Out of 40 marks:

3 marks	Clear and concrete problem description
3 marks	Methods are (meaningfully) commented
5 marks	100 lines of (meaningful) code
5 mark	“Kind-of” inheritance relationship is demonstrated
3 marks	Accessors and mutators are present, instance variables are private
3 marks	Constructors are present
6 marks	toString() and equals() are overridden at least once
4 marks	An interface is implemented at least once
3 marks	A Java collection is used in a meaningful way.
5 marks	Your main() method adequately demonstrates the behavior of your program.

What to Submit:

Submit all your Java files on D2L under Project #1. Please do NOT submit a zip archive, or any other archive. Submit your Java files and only your Java files. Ensure your program works when all files are in the same directory, as this is how we will test your code. In addition, if you are using an IDE other than VSCode, ensure there are no “package” statements left in your submitted code (ie. “package mypack;”). Again, this might only happen if you are using something other than VSCode.

Plagiarism Disclaimer:

You are to work **alone** when writing your code. Students may not copy problem descriptions, and if two students independently come up with the same or similar problems, their programs must be sufficiently unique. You can discuss general ideas with your classmates, but you cannot copy code or develop code together nor take code from the web (this includes asking ChatGPT, CoPilot, or other Generative AI systems to solve your problem for you).

The Department of Computer Science takes the act of plagiarism very seriously. Those caught plagiarizing (both originators and copiers) will be sanctioned. Please see TMU’s Policy 60 for possible penalties and consequences. If you are unsure what constitutes plagiarism, please see your instructor.