Homework Assignment 2

Due by 10:50am on Tuesday, October 31, 2017.

What to do

(a) Solve the problems described below.

(b) Follow the submission instructions to prepare your printed and electronic versions of the assignment. Remember, if you do not follow the submission instructions, your assignment will not be graded and you will receive a grade of 0.

(c) Submit your assignment as described below.

How to submit

You need to hand in your printed version before class on the due date. You need to submit your electronic version on Moodle before class on the due date.

What to submit

Printed version: Your printed assignment **must** be stapled and organized in the following way:

(a) Cover page (you can download it here). You can also find it on Moodle under "Homework Assignments".

(b) The output from the final version of the class `GuitarTester`.

(c) The final version of `GuitarTester.java`.

(d) The final version of all other classes you created.

Electronic version: All files except for code must be submitted as a PDF file. You can submit multiple PDF files if needed. Do not submit text files, Microsoft Word files, etc. PDF files can be prepared using `pdflatex`, `MS Word`, or other common word-processing tools. You can also "print" to PDF using a PDF printer. It does *not* work to take a plain text file and simply change the file name extension to `.pdf`. It is helpful if the file name contains your name, for example `ewa-syta-hw1.pdf`. Please submit all your files using the Moodle *Assignments* tool.

Please see the instructor if you have questions about submitting your assignment.

Assignment Objectives

The goal for this assignment is to ensure that you understand and follow the three steps of writing great software, as discussed in Lecture 10. To do so, you will improve the program your wrote for Rick's Guitar Inventory by fixing any functional issues there might be and you will refactor your code to ensure a flexible, maintainable and reusable design.

First, review Lecture 10 to understand the three steps of writing great software and how those steps were applied to the *sample* Guitar Inventory solution. For this assignment, you will improve and refactor your own solution. The goal is to produce a design that has the same qualities (flexibility, maintainability, and reusability) and not to simply implement a solution based on the UML diagrams given in Lecture 10. In fact, those UML diagrams are incomplete and reflect arbitrary design choices which might not fit your solution. Again, you are welcome to use the UML diagrams and explanations from Lecture 10 to guide your fixes and refactoring, however, you are not to simply copy it.

You need to document your functional improvements (create a list of functional fixes you made) and your refactoring (create a list of refactoring changes and briefly explain them). You specifically need to explain your use of exceptions (see Section 2).

Assignment overview:

- Part 1: Instructions to provide definitions and examples of OO concepts.
- Part 2: Instructions for functional improvements of your program.
- Part 3: Instructions for refactoring your code.
- Part 4: Instructions for testing.
- Part 5: Instructions for the UML Class Diagram.
- Part 6: Instructions for code style and documentation.

Part 1: Object-Oriented Concepts

So far, we have discussed several object-oriented concepts:

(a) Abstraction
(b) Encapsulation
(c) Inheritance
(d) Polymorphism
(e) Composition
(f) Delegation

For each of these concepts, *i)* give a **definition** using your own words (do *not* use a definition from any other source) and *ii)* if appropriate, give a specific **example** from your code for this assignment of how you used this concept in practice. Do not simply copy and paste code, include a proper explanation and only include code is necessary. Please use the following format.

Concept name

- Definition: your definition goes here...
- Example: your example goes here...

Part 2: Functional Improvements to the Guitar Inventory

Search Functionality

The search functionality is the most important feature in Rick's program. You need to ensure that it works correctly. The following list contains the minimum correctness requirements.

- Use keywords based on the properties encapsulated in `GuitarSpecification`.
- Searches should case-insensitive.
- Searches allow to omit some or all keywords.
- Searches should return all matching results.

Integrity of Entries

Your design should prevent adding `Guitar` entries with erroneous data to the inventory.[1] You should use `enums` for appropriate guitar properties to ensure that only allowed values for guitar properties are used. Further, you should make use of `IllegalArgumentException` when appropriate. You can read more about it here. Remember to document your choices in using `IllegalArgumentException`.

Part 3: Refactoring the Guitar Inventory

The goal here is to ensure that your code follows proper OO programming principles. You should refactor your code, that is, redesign your code and design without changing the behavior of your program. If you used inheritance, you should review your design and seriously reconsider your choice. Further, you should split your `Guitar` object into two objects.

- An object that holds properties that reflect how a guitar is viewed in the context of Rick's inventory.
- An object that holds properties to describe guitar specifications and that could be used to define Rick's customer's preferences for a guitar.

Once this step is done, you should use these two objects appropriately in your design. Review all of your classes and make sure that you take advantage of this design change. Specifically, you should:

- Review the `Guitar` constructor.
- Review the `add()` method.
- Review the `search()` method and delegate the comparison of two guitar specifications to the appropriate class instead of directly making the comparisons. Notice that you still need the `search()` method, you will simply delegate the comparison of guitar specifications and then use the result in `search()`.

Part 4: Testing

In this version of your program, you will not have a user interface. Instead, in `GuitarInventory.java`, you will design a series of tests to verify that the core functionalities (search, add, remove, and modify) of your program work correctly. Each test should be its own separate method (with appropriate helper methods when needed). Again, your tests will not obtain input from the keyboard, you should manually create `Guitar` objects for your tests.

Your tests should be designed to **convince** me that your program works as intended. Keep your tests comprehensive but succinct and focused. Pay attention to formatting of your output!

[1] After you refactor your code, please ensue that `null` objects are not passed as guitar specifications.

Part 5: UML Class Diagram

You should create a proper UML class diagram for your design. Please make sure to include constructors, multiplicities, and relationship labels. You should use appropriate connectors to indicate specific types of relationships between classes. Please use a proper drawing tool to prepare your diagram. Lucidchart is one of many such tools. You can create a trial account on Lucidchart and then upgrade it to a free and unlimited educational account. In Lucidchart, you can export your UML class diagram as a PDF or image.

Part 6: Code Style and Documentation

You need to follow the Google Java Style Guide. You can find it here. You may follow your own preferences for indentation: either 2 or 4 spaces. You should also follow the clean code principles as discussed in class. I strongly recommend using IntelliJ IDEA, a great IDE for Java. It has built-in tools that help to keep your code clean.

Document your code using the provided cover page. Specifically, document any issues with your code and your design choices.