C3

Creative Project 3: OOP It!

Specification

1️⃣ Learning Objectives

- Implement a well-designed Java class to meet a given specification
- Implement correctly encapsulated objects with instance methods and constructors
- Rewrite a previously-implemented program to using Object-Oriented design principles
- Follow prescribed conventions for code quality, documentation, and readability

2️⃣ Program Behavior

You have come a long way since the beginning of the quarter, and have implemented a lot of really cool programs! More recently, we've made the transition to focusing on *object-oriented programming* (or OOP), and benefits of *abstraction* or *separation of concerns* with this difference between a *client* and *implementer* view.

In this assignment, your first task will be to take one of the assignments you've completed and to reimplement it using object-oriented design principles (or "OOP it!").

3️⃣ OOP It!

You should choose **one** assignment to "OOP" out of the following options:

- C0 (Todo List Manager)
- P1 (Music Playlist)

- P2 (Absurdle)

Note that this assignment will build off of your implementations for these assignments. While you can choose any of the assignments listed above, we **strongly recommend** choosing one that you were able to fully and successfully implement.

To "OOP" one of these assignments, you will extract the basic functionality to an object that works in conjunction with a client program to reproduce the **same external behavior as in the original assignments**.

Your overall task on this assignment is to split up the code from your original implementation into a client class and an object class.

Your client program (`_____Main.java`) should handle all of the user interaction (prompting and reading for what the user wants to do next) and should act as a client of your object class (`TodoList`, `MusicPlaylist`, or `Absurdle`), creating an instance of the class and updating it through its instance methods.

C0: Todo List Manager

Object class: `TodoList.java`

Client program: `TodoListMain.java`

Your `TodoList` object should represent a TODO list and should have *instance methods* for all the functionality related to managing a TODO list (including but not limited to: adding an item, marking an item as done, printing out its items, etc.).

If you choose this assignment, you should OOP the original TodoListManager (not your extension).

P1: Music Playlist

Object class: `MusicPlaylist.java`

Client program: `MusicPlaylistMain.java`

Your `MusicPlaylist` object should represent a playlist of songs and should have *instance methods* for all the functionality related to managing a playlist of songs (including but not limited to: adding a song, playing a song, deleting from history, etc.).

P2: Absurdle

Object class: `Absurdle.java`

Client program: `AbsurdleMain.java`

Your `Absurdle` object should contain all of the main functionality involved in playing a game of Absurdle (including but not limited to: making a guess, reporting how many guesses the user has made, etc.).

4️⃣ Testing

The second part of your assignment is to test your new implementation. You should write a series of JUnit Tests in the `____Tests.java` file associated with your chosen assignment to test the code that you've written. Your tests should be thorough and test the correctness **every `public` method you have written for your object**.

Note that it is tricky to test the *console output* of methods using JUnit tests. It is possible, but is more complicated than testing the *state* or returned values from methods.

Because of this, **you do not need to test the console output of your object class's instance methods**. If a method's sole purpose is to produce console output (e.g., printing the history of the music playlist) you do not need to test that method specifically. However, if that method changes the state of your object, you should test out the change of state.

Note that this is specifically referring to *console* output - if you have methods that produce **file output**, those should still be tested!

In order to test the *state* of your object in cases such as this, you might want to include some *accessor* or *getter* methods so that your tests can access your state and check it!

You may want to refer back to the JUnit PCM and In-Class Materials as you work on your own tests to remind yourself of JUnit syntax and to see examples of how to structure tests.

Your program is expected to pass the tests you write in the `____Tests.java` file.

The **"Run"** button on each coding slide will run your client program, while the **"Check"** button will run your JUnit tests.

5️⃣ Implementation Details

As described above, your *client program* should handle all of the user interaction. In particular, **your object class should not use a `Scanner` attached to `System.in` at all**.

Development Strategy

We recommending OOPing your chosen assignment by following these steps:

1. Review the assignment's specification to recall the program's expected behavior.
2. Copy and paste your submission for the assignment into the client program (`___Main.java`) in the provided file, and read through it to consider what functionality should belong in the object class vs. the client program.
    - Recall that all user interaction should be done through the **client program**, not your object class.
3. Create the skeleton of your object class in the provided file.
4. Write your constructor(s).
    - How should your object be initialized? Does it make sense to have more than one constructor?
    - Use your constructor(s) to create an instance of your object in the client program.
5. One-by-one, address each functionality you want to move into your object class:
    - Write an *instance method* to capture the functionality.
        - Think carefully about what parameters your instance methods should accept. If they need to access data that is now a *field*, it probably doesn't need to be passed as a parameter!
    - Call that newly-written instance method in the *client program*.

- Remove the copy/pasted code that previously implemented this functionality.

6. Review your finished client program and object class and remove any code that is no longer being used.

We've included an example of taking a program from earlier in the quarter and "OOP"ing it so you can get an idea of how we'd like you to split up the functionality between your object class and client program. See the next slide "Example of OOP'ing It"!

6️⃣ Assignment Requirements

Note that you will be submitting 3 files for this assignment:

- Your object class (`TodoList.java`, `MusicPlaylist.java`, or `Absurdle.java`)
- Your client program (`_____Main.java`)
- Your JUnit tests (`_____Tests.java`)

Object Design

Your object class should have

- At least one constructor defined for the class
- At least three `public` instance methods

It is up to you to determine what state and behavior methods your object should implement. All fields that you create should be properly encapsulated and all of your object's fields should be absolutely necessary for solving the problem (i.e., your class should have no extraneous fields).

Testing

Your JUnit tests should be thorough and test the correctness **every `public` method you have written for your object**. In particular, you should have **at least one call to `assertEquals` (or an equivalent method) to test every `public` method.**

General Requirements

- You should use interface types where appropriate and properly use type parameters when specifying the types of data structures in your solution.
- You should comment your code following the Commenting Guide. You should write comments with basic info (a header comment at the top of your file), a class comment for every class (except tests), and a comment for every method other than `main` (see caveat below about test methods).
    - Make sure to avoid including *implementation details* in your comments. As you saw in class when we talked about Third Party Libraries, your method comments and class comments serve as your class's **documentation**!
    - In particular, for your object class, a *client* should be able to understand how to use your object effectively by only reading your class and method comments, but your comments should maintain *abstraction* by avoiding implementation details.

    - **Comments on test methods are not required** so long as your test method names are descriptive of what's being tested. See the example solution from JUnit in class materials for what descriptive test names might look like.
    - You also **do not need a class comment for your test file**.
- As a general rule, you have learned everything necessary to implement a solution to an assignment by the time it is released. If you are tempted to use any concepts or tools that have *not* been covered in class, check the **Forbidden Features** section of the Code Quality Guide first to make sure they do not appear there!