Assignment 4: Santa’s Software Helper

Summary

In this assignment, each student will develop an object-oriented program that enables a user to work with specific sets of data. This program will be developed using Python, leveraging what has been learnt in class.

Program Description

The program manages Santa’s data needs by integrating different data sources. The program manages a list of letters, each with an id, information about the child who authored the letter, a list of requested toys, and a flag of whether the child has been naughty or nice. Specific methods are designed to: open letter data, export a children list, import naughty/nice data, export toy manufacturing data, and save letter data.

Submission Checklist

Submit the following in Slate:

1. A completed Object-Oriented Design diagram/map (UML), saved as a PDF.
2. Visual Studio Code solution containing all the source files and git repository, compressed in a ZIP file.

Requirements

Object-Oriented Design (15%)

Create a UML that represents the design of your program. You must add required the required classes in addition to any additional classes used in your program.

You may use any program to design your UML (although FigJam is strongly recommended).

Project Setup (5%)

- Create a Python program with source files in one directory/folder.

- Be sure to use version controlling with a **git repository**. **Commit changes throughout the development of the program. Be sure to start committing changes from the start!**
- The program should be developed, and able to run, using Visual Studio Code.

Program Structure (40%)

It is important that your code is clear, concise, and adheres to standards and naming conventions discussed during class.

Your code must meet all the following conditions:

- Must have no syntax errors in your submission.
- Can only import the `random` and `math` modules (in addition to any module you create).
- **Use only techniques and concepts demonstrated in the course.**
- Should have meaningful names for variables and functions as well as follow Python naming conventions.
- Each custom class should be written in their own files and located in the program's directory/folder.
- **Use functions/methods to make code readable, organized, and modular (e.g. defining functions/methods that do specific things).**
- Classes should use attributes that are protected, implement initializers that are appropriate, and make use of accessor and mutator methods to reinforce encapsulation.
- Avoid using `break` to end a while loop.
- **Do not modify or circumvent any of the required components specified in the provided UML.**
- **Catch and handle all exceptions, including entering a wrong option from a provided choice.**

In your code, provide meaningful and relevant comments on design decisions made throughout the program. This includes comments for:

- Branching and looping.
- Functions and methods (placed at definition).
- Classes
- Header information for each file, describing the purpose of the file/module.
- Header information of the title, course, and author in `program.py`
- Where needed to explain complex code or specific decisions.

Use an object-oriented approach to programming. All statements **MUST** be in classes. The only exceptions to this are:

- Any `import` statements at the start of a module.
- In `program.py`, where a `Program` object is instantiated, and its `run()` method is executed.

The following classes, in addition to other classes you create, must be part of your program:

- `Program`:
  - Can have a `run()` method used to start and execute the program.
  - Has a list of `Letter` objects.
  - Each of the required options is initiated through a specific method.
- `Letter`:
  - Represents a letter from a child, requesting one or more toys.
  - Each letter has an ID, unique to the object (used for correlating with imported data).
  - Instance field variables reflect the fields in the provided `Letters.json` data file.
  - Each letter has an instance field variable named `_approved` that is `True` for a child has been nice, `False` if not, and None if yet to be determined.
  - Has a list of Toy objects requested in the letter.
- `Toy`:
  - Represents a toy.
  - Instance field variables describe the attributes of a toy and reflect the fields associated with a toy object in `Letters.json`.

User Interaction (40%)

No user interface is required. Instead, a user will use your code by running methods that accomplish specific tasks. Your program should have the following methods:

1. Open letter data: `openLetterData()`
   - Opens and reads a JSON file representing information collected from letters written to Santa.
   - Data is loaded as `Letter` objects that are stored as a list in a `Program` object. Requested toys are loaded as a list of Toy objects associated with each letter.

2. Export children list: `exportChildrenList()`
    - Exports a CSV file named `ChildrenListTemplate.csv` to be used by the elves determining who has been naughty and who has been nice.
    - The CSV file has the following fields: “Letter ID”, “Full Name”, “Nice”.
    - Exports data from `Letter` objects stored in the program.
    - The “Nice” field is left empty.
3. Import naught/nice data: `importChildrenData()`
    - Imports a CSV file named `ChildrenList.csv` provided by the elves determining who has been naughty and who has been nice.
    - Updates `Letter` objects with where “Letter ID” is the same the `Letter` object’s ID.
    - Updates the `_approved` attribute with either `True` or `False`.
    - Once the data is imported, the `Letters.json` file is updated using the `saveLetterData()` method.
4. Export toy manufacturing data: `exportToyManufacturingData()`
    - Exports a CSV file named `RequestedToys.csv` to be used by elves making toys.
    - The CSV file has the following fields: “Name”, “Category”, and “Description”.
    - Exports data from `Letter` objects stored in the program.
5. Save letter data: `saveLetterData()`
    - Writes updates to the `Letters.json` file.

Each time any of these methods are executed, a file names `ProgramLog.txt` is updated. The method executed, as well as the data and time of executing the method, should be logged.

Sample data is posted alongside this description in Slate.

Notes

1. The assignment shall be submitted by the specified due date. Late submissions will be penalized with 10% per day for up to 3 calendar days after which the assignment cannot be submitted anymore. Assignments are not accepted after the final deadline.
2. This assignment shall be completed individually.
3. Advanced AI tools are not permitted in any aspect of the assignment (e.g. artificial intelligence or machine learning tools such as ChatGPT). The assignment is to be completed without substantive assistance from others, including automated tools.

Remember that completing the assignment by yourself will ensure your success on the midterm and final exam. See the Academic Honesty guidelines at Sheridan.

4. **Reminder: if you want to include code found elsewhere, or not sure about how you can use a resource or technique, please consult with your professor first.**
5. Submitting the assignment is done using the SLATE Assignment folder. DO NOT email your submission.
6. Try to have fun when working on this assignment!