Assignment 5: Software Design

Background

A fellow classmate, George P. Burdell, is looking for a new job in the US after graduation. As it can be complicated to compare job offers with benefits, in different locations, and other aspects beyond salary, he would like an app to help with this process and has asked for your assistance in creating a simple, single-user job offer comparison app. As a first step, he would like you to create an initial design for the app, expressed in UML, based on a set of requirements he provided. **This deliverable thus consists of (1) a UML design document and (2) a design description document**.

Important: Please note that, although this assignment is only worth a small percentage of your overall grade, it is an important one because it will be used as the basis for your first group deliverable, which consists of sharing and discussing your design with your teammates. Therefore, doing a poor job in this assignment will likely penalize your group performance and ultimately hurt your collaboration grade.

Requirements

1. When the app is started, the user is presented with the main menu, which allows the user to (1) enter or edit current job details, (2) enter job offers, (3) adjust the comparison settings, or (4) compare job offers (disabled if no job offers were entered yet[1]).
2. When choosing to *enter current job details,* a user will:
    a. Be shown a user interface to enter (if it is the first time) or edit all of the details of their current job, which consist of:
        i. Title
        ii. Company
        iii. Location (entered as city and state)
        iv. Cost of living in the location (expressed as an index)
        v. Possibility to work remotely (expressed as the number of days a week one could work remotely, between 1 and 5)
        vi. Yearly salary
        vii. Yearly bonus
        viii. Retirement benefits (as percentage matched)
        ix. Leave time (vacation days and holiday and/or sick leave, as a single overall number of days)
    b. Be able to either save the job details or cancel and exit without saving, returning in both cases to the main menu.

[1] To be precise, this functionality will be enabled if there are either (1) at least two job offers, in case there is no current job, or (2) at least one job offer, in case there is a current job.

3. When choosing to *enter job offers,* a user will:
    a. Be shown a user interface to enter all of the details of the offer, which are the same ones listed above for the current job.
    b. Be able to either save the job offer details or cancel.
    c. Be able to (1) enter another offer, (2) return to the main menu, or (3) compare the offer with the current job details (if present).
4. When *adjusting the comparison settings,* the user can assign integer *weights* to:
    a. Possibility to work remotely
    b. Yearly salary
    c. Yearly bonus
    d. Retirement benefits
    e. Leave time

    If no weights are assigned, all factors are considered equal.
5. When choosing to *compare job offers,* a user will:
    a. Be shown a list of job offers, displayed as Title and Company, ranked from best to worst (see below for details), and including the current job (if present), clearly indicated.
    b. Select two jobs to compare and trigger the comparison.
    c. Be shown a table comparing the two jobs, displaying, for each job:
        i. Title
        ii. Company
        iii. Location
        iv. Yearly salary adjusted for cost of living
        v. Yearly bonus adjusted for cost of living
        vi. Retirement benefits (as percentage matched)
        vii. Leave time
    d. Be offered to perform another comparison or go back to the main menu.
6. When ranking jobs, a job's score is computed as the **weighted** sum of:

    AYS + AYB + (RBP * AYS) + (LT * AYS / 260) - ((260 - 52 * RWT) * (AYS / 260) / 8)

    where:
    AYS = yearly salary adjusted for cost of living
    AYB = yearly bonus adjusted for cost of living
    RBP = retirement benefits percentage
    LT = leave time
    RWT = remote work days per week
    The rationale for the RWT subformula is:
    a. value of an employee hour = (AYS / 260) / 8
    b. commute hours per year (assuming a 1-hour/day commute) = 1 * (260 - 52 * RWT)
    c. therefore **commute-time cost = (260 - 52 * RWT) * (AYS / 260) / 8**

For example, if the weights are 2 for the yearly salary, 2 for the retirement benefits, and 1 for all other factors, the score would be computed as:

2/7 * AYS + 1/7 * AYB + 2/7 * (RBP * AYS) + 1/7 * (LT * AYS / 260) - 1/7 * ((260 - 52 * RWT) * (AYS / 260) / 8)

7. The user interface must be intuitive and responsive.
8. For simplicity, you may assume there is a *single system* running the app (no communication or saving between devices is necessary).

Details

To create your design, you should follow the same approach that we present in the P3L2 lesson. That is, analyze the requirements (provided above) to identify and refine **(1) classes, (2) attributes, (3) operations, and (4) relationships** in your design. Just to be completely clear, **your task is to design the system, not to implement it**. At this stage, the design should be implementation neutral, and not involve constructs specific to a language or framework. For example, we are not looking for Android Activity classes.

Please note that not every requirement must be fully and directly represented in your design. For instance, at this level of detail, you do not have to show any purely GUI-specific classes. Similarly, any database support layer may be left out, if it is purely doing persistence tasks (simple CRUD only) on **data and collections already fully represented in the design**.

Please also note that we are expecting to see, in your design, a class that represents the entry point to the system, and that ties the various pieces together.

Your design should be expressed using a UML class diagram, and the level of detail of the design should be analogous to the level of detail we used throughout the **whole** P3L2 lesson. That is, do not limit your design to only the elements shown in the final video, where some of the elements shown earlier (e.g., methods) are elided for space). Specifically, **you must provide enough details for the design to be self-contained and for us to be able to assess whether the design suitably realizes all system requirements**.

To help to make your design self-contained, you must also provide a “design description” document, in which you concisely describe, **for each of the requirements listed above**, either (1) how that requirement is realized in your design, or (2) why it does not directly affect the design and is therefore not shown. To produce this document, you should **copy the list of requirements and add your explanation for each one of them**. For example, using some partial requirements for a cash register app:

…

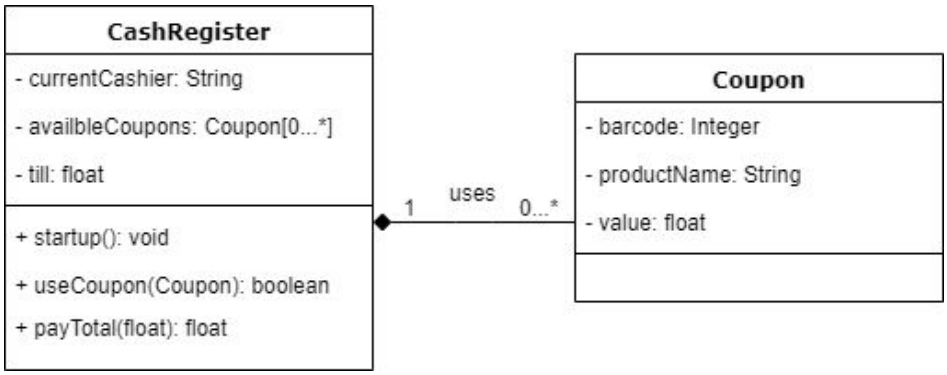
2. After starting the cash register, the cashier enters her name, and the total amount of money available in the till.

`To realize this requirement, I added a 'currentCashier' to the register class to track the signed-in cashier, and a float 'till' to represent the money in the till. These values are entered by the startup() method, after prompts are handled within the GUI.`

…<additional requirements reflected in example UML diagram>...

16. The User Interface (UI) must be intuitive and responsive.

`This is not represented in my design, as it will be handled entirely within the GUI implementation.`



The explanation in the design description should be clear enough to allow us to follow the rationale behind your design and **how it will fulfill each specified requirement, including any that are not directly depicted in your class diagram.** You can also provide in the document additional information about your design, such as assumptions or rationale for some design decisions. Use the document to review your design and ensure you have included everything necessary for your app to fill the list of requirements.

Submission Instructions

To submit your assignment, you should do the following:

1. Create a directory called `Assignment5` in the usual **personal GitHub repository we assigned to you**. This is an **individual assignment**; do **not** use your new team repositories.
2. Save your UML class diagram in the `Assignment5` directory as a PDF file named "`design.pdf`". You can use any UML tool of your choice to create your UML design, but please **do not hand draw the design**.
   Important: Make sure to open your PDF after generating it and double-check it, as we have had a number of cases of students not realizing that the conversion to PDF had not worked as expected.
3. Save the "design description" document in the same directory, in markdown format, and named "**`design-description.md`**".
4. Commit and push your file(s) to your remote repository.
5. On gradescope, submit a file called `submission.txt` that contains, in two separate lines (1) your GT username and (2) the commit ID for your submission. For example, the content of file `submission`.txt for George P. Burdell could look something like the following:

   ```
   gpburdell1
   81b2f59
   ```

   Important: As soon as you submit, your assignment will be checked by making sure that you have the required directory structure and documents. However, we cannot automatically check your design, so **a positive response from Gradescope only indicates that we will be able to grade your assignment, and the grade you will receive is only a placeholder.**

   You can resubmit as many times as you want before the deadline, so you have a chance to address issues with your assignments if Gradescope finds any problem with your submission.

PLAGIARISM

As for all other assignments, we check for plagiarism. Unfortunately, there are plagiarism cases every semester. Please keep in mind that the tools we use to identify cases of plagiarism have access to the same online resources that are available to you.