Course Project Description

In this project, each group needs to apply the knowledge learned in the course to develop a command-line-based personal information manager (PIM). Appendix B gives more information about the PIM.

Group Forming: Form your groups of 3 to 4 students, from the same or different class sessions, on Blackboard before or on 10 October 2023. Afterward, students with no group will be randomly grouped, and requests for group change are only approved if written agreements from all members of the involved groups are provided before or on 14 October 2023.

Object-oriented vs. Procedural-oriented: You are suggested to design and implement the PIM in an object-oriented way because OO techniques often help produce software systems with better quality in terms of, e.g., code cohesion and coupling, understandability, and maintainability. That, however, does not mean you will be penalized for choosing the other way. We will consider the capability and limitations of each design and programming paradigm when grading, so if you feel more comfortable working with the procedure-oriented paradigm, or if you don't have enough time to learn the OO techniques for this project, feel free to adopt the procedure-oriented way. Particularly, you can program the PIM in either Java or Python. If you want to use another programming language, consult your instructor first.

Deliverables: Each group needs to submit the following deliverables, compressed into a single ZIP archive, before or on 25 November 2023 (Saturday). Unless explicitly stated otherwise, all the required documents in the deliverables should be in DOC, DOCX, or PDF format.

1. **Software requirements specification (SRS). (6 points)**
   You may decide on the specifics regarding how users will use the PIM, but all the user stories given in Appendix B should be supported, and the requirements should be valid, consistent, complete, realistic, and verifiable.
   a. *Document structure*: See slides “The Structure of A Requirements Specification (1) & (2)” from “Lecture 04 Requirements Engineering” for the overall structure of a requirements specification; Note that chapters “System models”, “System evolution”, “Appendix”, and “Index” are not needed in your requirements specification.
   b. *Contents*: All the important functional and non-functional requirements should be included.
      - See slides “Natural Language Specification” and “Example Natural Language Specifications” from the same lecture for guidelines on, and examples of, writing natural language requirements.
      - To facilitate unit testing (see Deliverable 4), you should put all the code that belongs to the model of the PIM into a separate package named “model”. You may refer to the Model-View-Controller (MVC) pattern [a] to help you understand what constitutes the model of the PIM. We will discuss the MVC pattern in Lecture 6 Architectural Design.
      - It is not recommended to add obviously additional features, e.g., a GUI or the support for online use, to the PIM. Your efforts to design and implement the additional features will not get you extra credit.

2. **Design document. (5 points)**
   The deliverable should contain 1) a diagram to describe the PIM’s architecture, 2) diagrams to show both the structure of and the relationship among the major code components in the PIM, and 3) a diagram to outline the process of an example use of the PIM, where a user first searches for some personal information records (PIRs) and then updates one of the returned PIRs (See Appendix B for more information about the example use of the PIM).
   a. Architecture: Explain which generic architectural pattern you picked for the PIM, why you chose this pattern, and how the components of the pattern were instantiated in the context of the PIM. (“Architectural Design” will be discussed in Lecture 6.)

[a] https://en.wikipedia.org/wiki/Model-view-controller

b. *Structure of and relationship among major code components*: If you adopt object-oriented techniques, explain the classes in your design and their fields and public/protected methods; If you adopt procedural-oriented techniques, explain the user-defined data types and functions in your design. For each method/function, the explanation should include information about the return type, the method/function name, the argument names and types, and possible exception declarations. Explain how the major code components are related to each other in the PIM.

c. *Example use*: Demonstrate the collaboration among the code components using a sequence or activity diagram.

d. *Necessary textual explanations*: Each diagram should be accompanied by necessary textual explanations to improve understanding.

3. **Implementation. (6 points)**
The deliverable should include 1) the complete source code for the PIM, 2) a developer manual explaining how to compile the code and build the project, 3) a user manual describing how to use the PIM, 4) a short video (no more than 4 minutes) of the PIM in use, and 5) a short report on requirements coverage achieved by your implementation.
    a. *Source code:* The code implementing the model of the PIM should be placed in a separate package named "model", and the other major code components should also be easily identifiable in the source code. Your implementation should only use Java/Python standard libraries.
    b. *Developer manual*: Here, you may assume the readers know how to properly set up a Java/Python development environment, and you don't have to explain that in the manual. You only need to prepare the manual for one platform, e.g., Windows, macOS, or Ubuntu. You need to explain which version of JDK/Python was used for the development, which IDE should be used to open your project, which commands should be used to compile/build your project, and/or how the PIM can be started in the debugging mode.
    c. *User manual*: In the user manual, you need to explain things like what types of PIRs the program can manage, what are the restrictions associated with different types of PIRs, which input commands are supported, what their formats are, what happens if an input command is invalid, and how to interpret the output of the PIM. Everything a user needs to know to effectively use the PIM should be explained in the user manual.
    d. *Video*: The video should be in MP4 format. Given the limited length of the video, you only need to demonstrate the most important features of the PIM. A feature can be important for the user because it is a basic operation or it greatly influences the user's experience, and it can be important for you, as the developer, because, e.g., you are proud of the design behind the feature. You don't need to verbally explain everything in the video since most interactions between the PIM and its users should be straightforward, but feel free to add narratives and/or texts to the video to describe the things that are less obvious and/or worth extra attention.
    e. *Requirements coverage report*: Report in a separate spreadsheet, i.e., an XLS or XLSX file, which user stories from Appendix B and which requirements in the SRS have been implemented. Put the file in the root folder of your source code.

4. **Unit tests for the PIM model. (4 points)**
Each test should clearly state the functionalities it exercises (e.g., using comments) and the expected results (e.g., using assertions).
    a. *Unit testing framework*: All the unit tests should be automatically executable and should pass when executed against your PIM implementation. You are strongly recommended to prepare the tests using unit testing frameworks. Two example unit tests, in JUnit for Java and unittest for Python, respectively, are given in Figure 1 for your reference, but feel free to use other unit testing frameworks if you so desire. Please refer to the corresponding documents and/or tutorials for information about writing unit tests based on different unit testing frameworks.

```java
// JUnit for Java
...
public class JUnitProgram {
  @Test
  public void test_JUnit() {
    String str1="testcase ";
    assertEquals("testcase ", str1);
  }
}
```

```python
# unittest for Python
import unittest
class Testing(unittest.TestCase):
    def test_string(self):
        a = 'some'
        b = 'some'
        self.assertEqual(a, b)
if __name__ == '__main__':
    unittest.main()
```

Figure 1. Example unit tests in Junit for Java and unittest for Python, respectively.

b. *Test coverage*: Add a separate TXT file to report the line coverage achieved by your tests *on the PIM's model* and place the file in the root directory of your test files. *Note again that you only need to write unit tests for the model of the PIM.*

5. **Presentation slides and recording (4 points)**
   The project presentation will be delivered in the last week of the semester. Each group has 4.5 minutes for the presentation and 1 minute for Q&A. Details about the organization of the presentations will be announced later.
   a. *Presentation contents*:
      - Three system requirements regarding 1) the creation of a PIR, 2) the definition of a search criterion, and 3) the search for specific PIRs, respectively.
      - The overall design of the PIM, i.e., the combination of Deliverables 2.a and 2.b.
      - One important lesson learned from the project regarding requirements engineering, API design, or unit testing.
   b. *Presentation slides and recording:* The slides should be in PDF format, and the recording should be in MP4 format.

Appendix A. Grading Criteria

1 is for the lowest quality, coverage, etc.; 5 is for the highest.

Software requirements specification	1	2	3	4	5
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Requirements quality (in terms of validity, consistency, completeness, realism, and verifiability)	☐	☐	☐	☐	☐
Design document					
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Design quality (in terms of modularity, efficiency, extendibility, justifiability, etc.)	☐	☐	☐	☐	☐
Diagram quality (in terms of completeness, well-formedness, understandability, etc.)	☐	☐	☐	☐	☐
Implementation					
Deliverable completeness	☐	☐	☐	☐	☐
Document quality (in terms of contents, structure, and writing)	☐	☐	☐	☐	☐
Code quality (in terms of reliability, readability, etc.)	☐	☐	☐	☐	☐
Requirements coverage	☐	☐	☐	☐	☐
Unit tests for the PIM model					
Deliverable completeness	☐	☐	☐	☐	☐
Test quality (in terms of readability and granularity)	☐	☐	☐	☐	☐
Test coverage	☐	☐	☐	☐	☐
Presentation slides and recording					
Deliverable completeness	☐	☐	☐	☐	☐
Presentation quality (in terms of contents, length, and delivery)	☐	☐	☐	☐	☐
The presentation is properly timed.	☐	☐	☐	☐	☐

Individual grading. In the project, each group member is expected to actively participate and make fair contributions. In general, group members will receive the same grade for what their group produces at the end of the project. Individual grading, however, could be arranged if any member believes "one grade for all" is unfair and files a request to the instructor in writing (e.g., via email). In individual grading, the grade received by each group member will be based on his/her own contributions to the project, and each member will be asked to provide evidence for his/her contributions to facilitate the grading.

Use of GenAI tools. If you use GenAI tools in preparing any of the deliverables, you need to clearly state in a separate document 1) which deliverable(s) and which part(s) of the deliverable(s) were prepared with the help of GenAI tools, 2) how you used the GenAI tools, and 3) what you did to verify/revise/improve the results produced by the tools. Without such a document in your submission, we will consider all the deliverables your original work. Using GenAI tools in the project is permitted, but you should give proper acknowledgement to all the content from those tools that you included in your deliverables. Failure to reference externally sourced, non-original work can be considered plagiarism. You may refer to https://libguides.lb.polyu.edu.hk/academic-integrity/GenAI_and_Plagiarism for more guidance on GenAI and plagiarism.

Appendix B. User Stories for The Personal Information Manager (PIM)

US1. As a user, I want to create different types of personal information records (PIRs) in the PIM so that all the information I care about can be managed in a single location.

US2. As a user, I want to create new plain texts as PIRs so that I can use the PIM to take quick notes.

US3. As a user, I want to create new tasks with the corresponding descriptions and deadlines as PIRs so that I can use the PIM to manage my to-dos.

US4. As a user, I want to create new events with the corresponding descriptions, starting times, and alarms as PIRs so that I can use the PIM to manage my schedule.

US5. As a user, I want to create new contacts with the corresponding names, addresses, and mobile numbers as PIRs so that I can use the PIM to manage my contacts.

US6. As a user, I want to modify the data in existing PIRs so that I can keep the PIRs up to date.

US7. As a user, I want to search for PIRs based on criteria concerning their types and the data stored in their fields. A criterion may check whether a piece of text (stored in a note, a description, a name, an address, or a mobile number) contains a string, whether a time (stored in a deadline, a starting time, or an alarm) is before (<), after (>), or equal to (=) another given point in time, or whether a condition combining multiple other conditions via logical connectors and (&&), or (||), and negation (!) is satisfied.

US8. As a user, I want to print out detailed information about a specific PIR or all PIRs.

US9. As a user, I want to delete a specified PIR.

US10. As a user, I want to store the PIRs in a file with the extension name ".pim" so that I can access them using the PIM in the future.

US11. As a user, I want to load the PIRs from a file with the extension name ".pim" so that I can continue working with the PIRs I stored earlier.