University of
South Australia

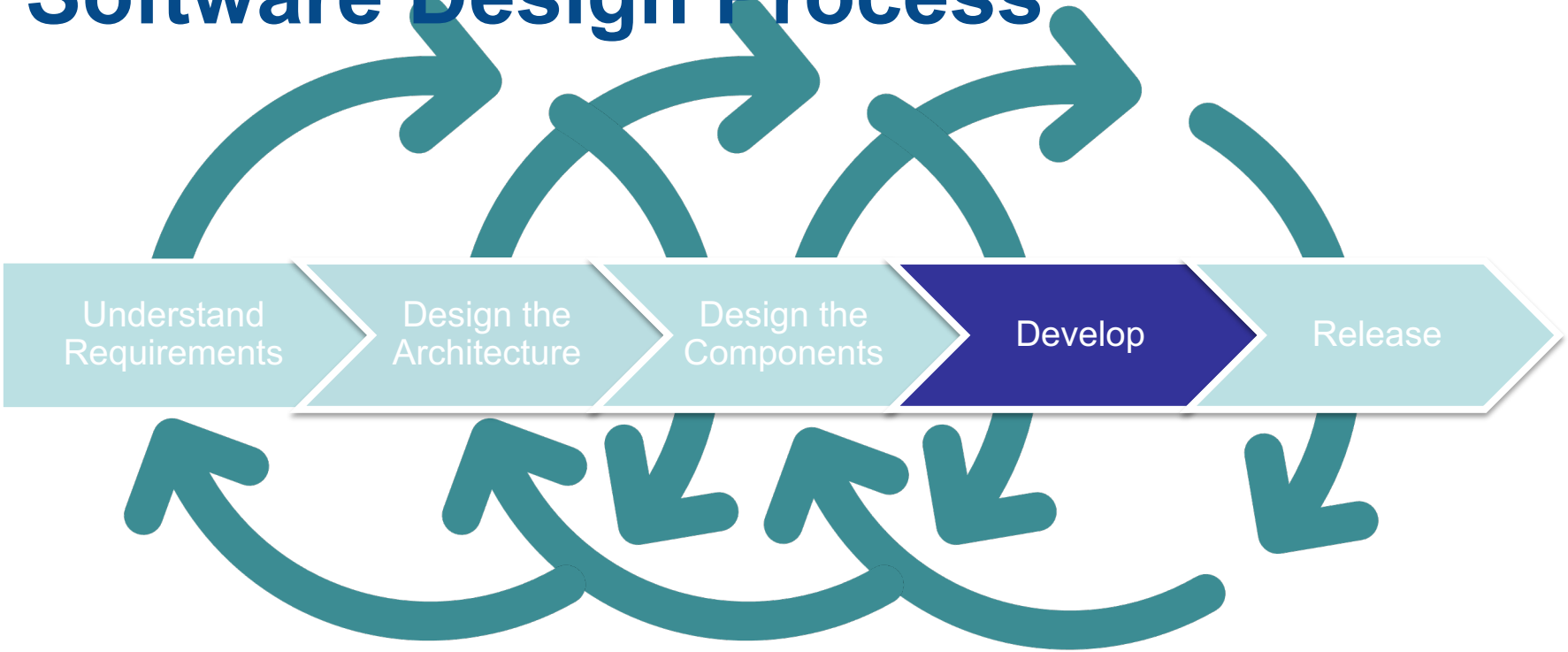
INFS 2044

Practical 1 Answers

Preparation

- Revise Python programming (classes & methods)
- Read the required readings
- Watch the Week 10 Lecture

Software Design Process



Learning Objectives

- Implement a given software design in code (CO4)
- Apply design principles to create clean code (CO4)

Task 1. Install Python

- Install Python 3 and a development environment of your choice.
  We recommend:
  - Anaconda Python
  - Visual Studio Code or Pycharm

- Python may already be installed in the computer pools.

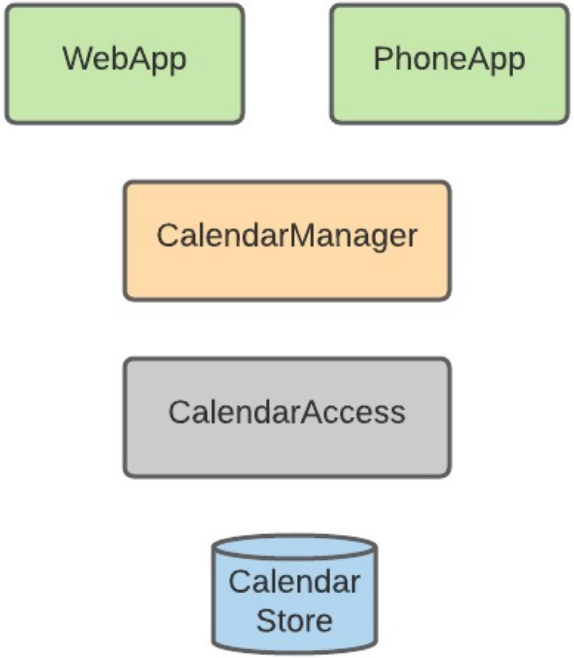
Task 2. Implement a Given Design

- We will revisit the design of the Calendar application from Workshop 4
- You will implement a variant of the calendar application in Python
- **KEEP THE CODE FOR NEXT WEEK**

Partial Volatility List

- Client Applications
  - Web app
  - Phone app
- Storage Backend
  - In-memory
  - Relational database
  - Document-oriented database

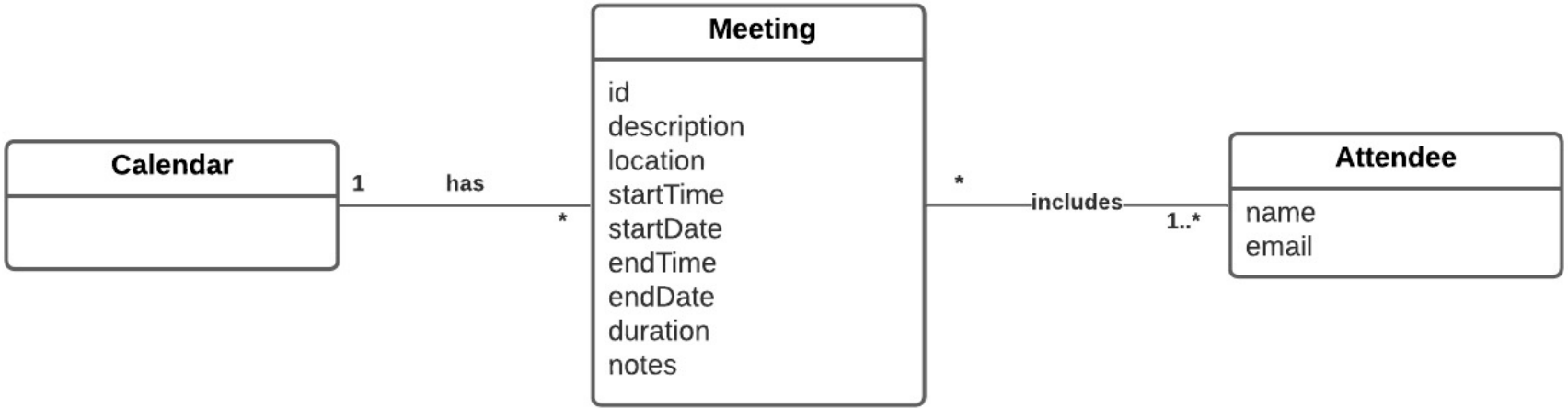
Component Decomposition



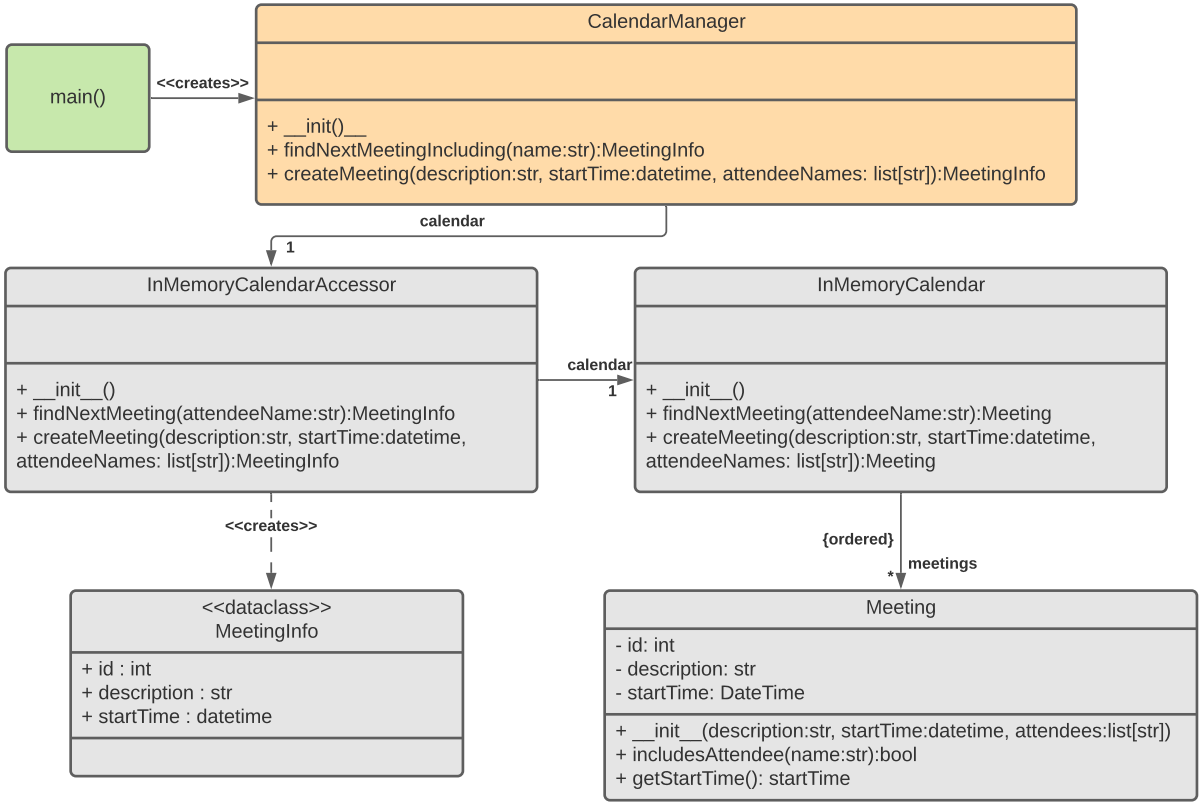
Component Responsibilities

- WebApp, PhoneApp
  - Display information and take commands from user
- CalendarManager
  - Provide an interface to the applications
- CalendarAccessor
  - Access the data store
- CalendarStore
  - Store the data of calendar, meetings, contacts

Domain Model



Partial Implementation Design



Understand the Design

- Look at the design on the previous slide and try to understand:
  - The purpose of each class
  - How each class relates to the component decomposition and the domain model
  - How objects collaborate at runtime to
    - » Create meetings
    - » Find meetings including a given attendee
  - Understand the purpose of the *MeetingInfo* structure

Information Hiding at Boundary

- InMemoryCalendarAccessor is a boundary class
- InMemoryCalendar and Meeting are private implementation classes
  - The Manager should not depend on them
- MeetingInfo decouples the Manager from the private implementation classes

This Design is Incomplete

- You will need to make some design decisions
  - Which object creates and stores the attendee information of a meeting
  - Which object creates the InMemoryCalendar and its Accessor
  - Which object creates Meetings
  - Which object generates unique identifiers for meetings
- Try to adhere to clean code principles
  - Small methods with a clear purpose, meaningful names, loose coupling among classes

Design Decisions 1

- Which object creates and stores the attendee information of a meeting?

- Attendee information shall be stored in a dedicated class (Contact), since attendee information has multiple attributes.
- The Meeting object should create the Contact objects, since the Meeting aggregates these objects.

Design Decisions 2

- Which object creates the InMemoryCalendar and its Accessor?

- CalendarManager creates the Accessor object
- Accessor creates the InMemoryCalendar object

- We will revisit these decisions in Prac 2

Design Decisions 3

- Which object creates Meetings?
- InMemoryCalendar creates Meeting objects, since the calendar aggregates the list of Meetings.

Design Decisions 4

- Which object generates unique identifiers for meetings?
- Identifiers are created by a separate UniqueIDGenerator object.
  - This responsibility is not intrinsic to the Meeting class
  - It is generic and can be reused in other contexts

Task 2 Brief

- Implement the design shown on the previous diagram in Python
  - Consider only the in-memory calendar store
  - Implement a main() program that directly invokes the Manager. Ignore the Client Applications.
  - Adhere to the classes and method signatures given in the design. You may need to introduce additional classes and methods.

Task 2 Subtasks

Implement in the following order:

1. main()
2. Test scenario
3. Manager component
4. In-memory calendar store and corresponding accessor

At each step, the parts of the program that you have already written will call parts that have not yet been written. Create placeholder classes that define the methods in the to-be-written part of the program. You may insert print() statements to verify that these methods are indeed called when the program is run. You will replace the placeholders with the correct code in a later step.

Task 2.1 main()

- Write a main() function that creates the CalendarManager
- Create a placeholder CalendarManager class
- Verify that the program runs

Task 2.2 Test Scenario

- Add to main() statements that
  - Create three or more meetings with several attendees
  - Find the next meeting including one of the attendees
  - You can hard-code the meetings and attendees for now
  - Add placeholder methods to the Manager as needed
  - Run the program to ensure it executes without errors

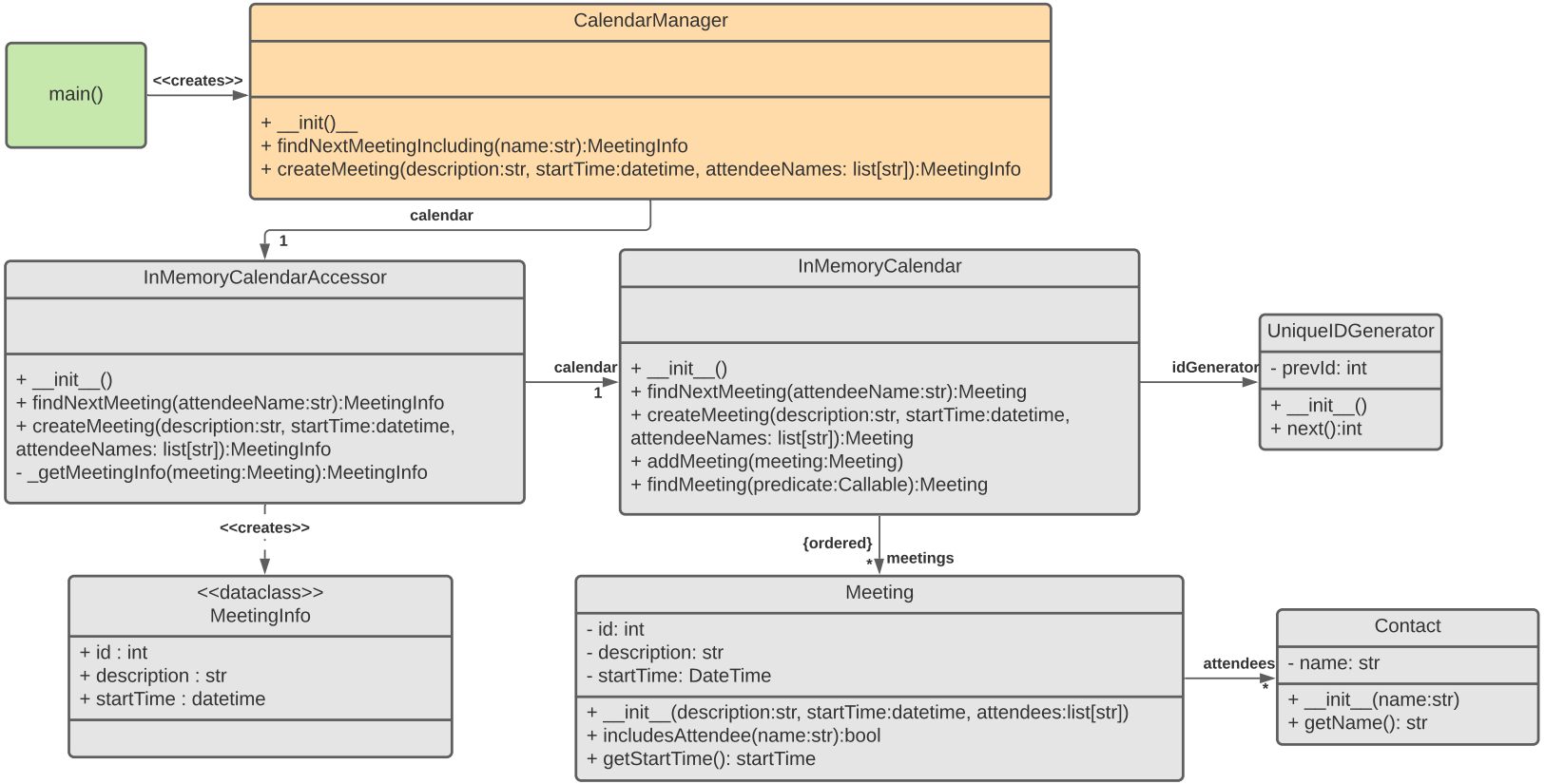
Task 2.3 Manager Component

- Implement the functions you have created in the Manager component
  - Introduce classes and placeholder methods in the resource component
  - Run the program to ensure it executes without errors

Task 2.4 Resource Component

- Implement the calendar resource access component and the in-memory calendar store
  - The in-memory store holds all calendar data in the form of objects in memory. There is no backing to a database etc.
  - Complete the implementation of the classes in these components.
  - Run the program to ensure it executes without errors.
  - Now the program should be complete. Verify that the program correctly creates the meetings and finds the correct meeting with including the attendee that you have coded in the main() test scenario.

Final Implementation Design



Implementation Decisions

- Private method _getMeetingInfo(…) has been introduced in the Accessor, as this function may be used in several public functions in the accessor once it grows.
- addMeeting(…) was introduced in the Calendar to keep the responsibility for keeping the list of meetings sorted separate from creating the meeting object
- findMeeting(…) was introduced as this method is a useful generalisation of finding a meeting by a given criterion that may be useful later. findNextMeeting(…) is implemented using findMeeting().

Task 3. SAVE THE CODE

- Save the code for next week.
- We will continue to improve this code in the next Practicals
  - Write automated tests
  - Improve naming and comments
  - Refactor the code

You Should Know

- Program in Python
- Translate a design expressed as UML diagrams into code
- Structure methods and classes according to design principles

Activities this Week

- Complete Quiz 6

University of
South Australia