January 15, 2025

LEC 04 - Object Oriented Programming Continued

Composition:

Key Concept: Designing classes that interact with other classes

Idea: Use existing classes as types in new classes

- Ex. storing references of a class as an attribute in another class

Composition: A relationship between two classes where instances of one class contain references to instances of the other

- Ex. Class `User` and class `Tweet`

Example:

Say we want to implement class User based on the following spec:

> "A Twitter user has a unique identity, a short bio, and a list of tweets that they created. A user may create a new tweet or follow another user."

- We can have 2 classes: `Tweet`, and `User`
- The User class can have an attribute `tweets: list[Tweet]`, which stores all the tweets that user has made
- The user class stores its identity and bio as attributes
- The Tweet class can store the details of the tweet, such as the creation date, likes, and content

```
class User:
"""A Twitter user.

=== Attributes ===
userid: the userid of this Twitter user.
bio: the bio of this Twitter user.
tweets: a list of the tweets that this user has made.
"""
userid: str
bio: str
tweets: list[Tweet]
```

Data Encapsulation:

General Idea: Attributes and methods of a class are made private to hide implementation details or to protect them from unauthorized use

- Only allowing reading and writing of the attributes / methods using special methods that the class designer can control

January 15, 2025

LEC 04 - Object Oriented Programming Continued

Private: Marking an attribute / method as private signals that the client code should not access it

Conventions:

_name → Should not be used outside of class definition, but still accessible if you try to access it by name
- Ex. Content for a tweet

__name → Inaccessible and invisible
- In theory, since there are still ways to access them

A private attribute / method could be:
- Very complicated
- Subject to several representation invariants
- Seemingly unrelated to the actual purpose of the code
- Changed at any time

Interface vs. Implementation:
- Interface is the part of the code that the client can see and use
- Implementation is the backend that the client does not need to see and use
- Ex. Watch face is the interface, while the inner mechanics are the implementation