COMP1005 Week 6 Cheat Sheet

Lisa Luff

9/22/2020

Contents

Object-Orientation

- In object-oriented programming, behaviour (methods) and data (attributes) are bundled together
- Benefits:
  - OO protects data from being used incorrectly
  - Increase code reuse - fewer errors if you write less, which will be the case with re-use
  - Makes code easier to read and maintain
  - objects "know" how to respond to requests
  - Relates to how objects function in the real world

Classes

- Classes are ways to define objects
- They specify the state and behaviour an object can take:
  - State - What the object is
    - Attributes or member fields
  - Behaviour - What the object does
    - Methods or functions
- A class provides encapsulation
  - Communication with the rest of the software system is clearly defined
    - This is done through methods
  - It's obligations to the software system are clearly defined
    - What services the class offers (data and methods)
  - Implementation details should be hidden from the user
    - Makes use of the "information hiding" principle
    - Don't need to know how it does these things
    - Stops us "accidentally" changing things and creating errors
- Class specification must include:
  - Method names
  - Exact data representation required
  - Method implementation
- Classes vs. objects
  - An object is a specific instance of a class
  - The class definition will provide a template for the specific object to be based on
  - An object gives details for a specific instance
    - Eg. The class would be "cat", and the object would be "Cat"
- Class roles:
  - Every class has a specific role in mind
  - The total set of functional requirements for a software system is broken down into a set of tasks
  - Collections of tasks are grouped together and mapped to roles
  - Roles are mapped to specific tasks

data → function = task → method = collection of tasks → class = role

- A software application will:
  - Identify required classes
  - Assign specific classes
  - Determine relationship between classes
  - Each responsibility should be handled by that class and no other

Comparing to Non-OO Design

- In a top-down procedural approach:
  - We design an algorithm with a main module using step-wise refinement to determine the processing steps
  - Some steps get refined into sub modules and the process repeats until the design is refined enough to code
- OO Design:
  - Before the algorithm is designed -
    - Classes are identified
    - Each class assigned roles/responsibilities
    - The required sub modules are designed
    - Each class is thoroughly tested via a test harness
  - Then the main algorithm and required sub modules are designed using the classes

Nouns and Verbs

- We use the nouns and verbs approach to class naming:
  - Nouns are the class names - Describe a "thing"
  - Verbs are the sub modules within classes - Describes an action

Object Communication

- Or message passing
- It's when an object of one class, calls an object of another class
- The public methods must provide the functionality required for the class to fulfill its role
- Five categories of methods in a class:
  1. The constructors - Create an object
  2. The accessor methods (interrogative methods) - get info
  3. The mutator methods (informative methods) - change info
  4. Doing methods (imperative methods) - use info
  5. [Private] methods - the user does not see

Classes in Python

- Declare the components of each class in the following order:
  - Declarations for class constants
  - Declarations for class variables/fields
    - Global variables (used for all methods of the class)
  - Declarations of instance variables
    - Local variables (local to each object)
  - Declarations of the constructors (___init___)
  - Accessor methods
  - Mutator methods
  - Doing methods ("public")
  - Internal methods ("private")
- **Note**: Everything in Python is "public", so we can only treat methods and data as private

How to use

- To create a class:

class *Noun*():

myclass = *Noun*

- *globalvariable* = *value*
- def ___init___(*self*, *instancevariable*):
-- self.*instancevariable* = *instancevariable*

- def *verb*(self):
-- *do thing to instancevariable*

- To use a class:
  from *program* import *Noun* (OR import * for multiple classes)

object1 = *Noun*(‘*instancevariable*’)

object1.verb(‘*thing*’)

print(“Data:”, *object1.instancevariable*)

- **Note:** self is the naming convention for any instance variable
  - This is to make it clear whether this is an instances cheese attribute or a local variable
- **Note:** Should also include a printing function, because if you try to print directly you can end up with binary