6 – Object-Orientated Programming

Friday, September 21, 2018 8:57 AM

OOP involves writing code around classes and objects.

Why is the paradigm used?

What's the point?
https://stackoverflow.com/questions/24270/whats-the-point-of-oop

Key Concepts

Some of the concepts used in OOP are:

- Class – an extensible template to create objects
- Object – an instance of a class, containing variables, procedures and data structures
- Instantiation – the process of creating a class from an object
- Encapsulation – the process of making object data only accessible via methods (private)
- Inheritance – importing methods and variables from one class into another
- Aggregation – holding a reference to a different object within another
- Composition – creating an instance of a different object within another
- Polymorphism – classes have different functionality but a common interface (e.g. integers and floats can both be added, subtracted etc but are different classes and you could create your own `number` classes that also implemented these functionalities). Functions or methods handle data differently depending on their types or classes.
- Overriding – when superclass methods are re-defined by sub-classes

See https://github.com/joeiddon/python_chat for Python language specifics.

Principals

There are three object-oriented design principles:
- encapsulate what varies - make code localised rather than interlinked (e.g. through messy globals), so if a project requirement changes, the code change is not spread around
- favour composition over inheritance - composition can give you access to just some methods of classes rather than all. E.g. a bird may need the fly ability of a plane, so it makes sense to extract the fly ability out of the plane as a class/interface/both and make it a member of both classes
- program to interfaces, not implementation - this hides the things you do not need to know about a class, so using it is simpler. Think of library documentation.

(An interface is a contract defining the methods that must be defined for each implementation. They are not available in Python but are in, for instance, Java.)

Class Diagrams

- Arrow = inheritance, arrow pointing to parent
- Empty diamond arrow = aggregation
- Filled diamond arrow = composition
- Variables must give their type and their visibility is described by these specifiers:
  * a hyphen (-) = private (just that class)
  * a plus (+) = public (all classes)

* a hashtag (#) = protected (that class and its children)

Methods

Methods can be: **abstract** meaning they are defined but contain no implementation so subclasses *must* override them; **virtual** meaning they *can* be overridden but don't need to be as they are already implemented and **static** meaning they *can't* be overridden and can be called on the class directly without instantiating objects (@staticmethod decorator in Python).