CS 172 - Computer Programming II

Introduction to Object Oriented Programming

Objectives

- Basic Object-Oriented Analysis (OOA), Object-Oriented Design (OOD), and Programming (OOP) Concepts
  - Understand what object-oriented analysis and design is
  - Understand the relationship between objects and classes
  - Attributes and behaviors of objects
- Using Objects
- Modules and Packages

Intro to Object Oriented Design

- In many applications, bundling together variables to form an "object" makes sense.
- For instance, if our application requires keeping track of many cars, it may make sense to define the concept of a car
  - A car contains a make, model, color, year, mileage, etc.

OOA, OOD and OOP

- In addition, we may want to define what *actions* can be performed on those objects
  - Change color
  - Update mileage
  - Speed up
- The process of taking a problem and identifying the objects needed and the interactions between the objects is called *object-oriented analysis*
- The outcome of the analysis is a set of requirements.

OOA, OOD and OOP

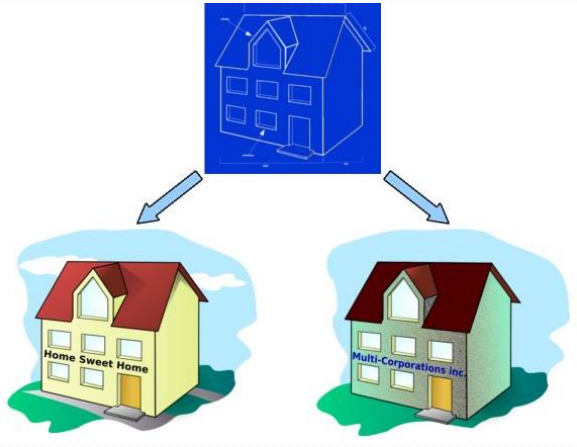
- The process of converting the requirements into an implementation specification is called *object-oriented design*.
- A lot goes into the design process
  - What variables should there be?
  - What actions should there be?
- *Object-oriented Programming* (OOP) is a programming technique that takes advantage of the concept of objects.
  - Converts the OOD into a working program

So… what's an object?

- An object is
  - A collection of data and
  - A set of actions that can be performed on the object
- In OOP terminology, we usually call
  - The collection of data that defines a type of object, its *attributes* (or *members* or *properties*)
  - the collection of actions, its *methods*

What is a class?

- A *class* defines object properties, and it also describes object behavior
- Classes describe objects
  - All objects of the same class have the same attributes and methods
  - Like a blueprint for creating an object
  - Like a cookie-cutter



Instantiating a class

- If we want to create an instance of a type of object, we call it *instantiating a class*
  - Or creating an instance of a class
- Typically, to create an instance of a class, we call a special function that is just the name of the class.
- This function returns a new instance of the class (an object) that we can store in a variable.
- As we'll see a bit later, this process can also involve an implicit call to a special method called `__init__` to which we can pass parameters to help initialize the attributes of the object.
  - This method is often referred to as the *constructor*
  - So, we are constructing objects

Using Objects

- Let's start off by revisiting using existing classes.
- The first thing we need to do is create an instance of a class.
- Then we can call its methods!
- We've already used some classes that are provided for us by the developers of Python *(built-ins)*:
  - `string`
  - `file`
  - `list`

Python `list` class

- To demonstrate this process, let's look at Python's built-in **`list`** class
- To create a new **`list`**, we can do:

```
myList = list()
```

- Often there are several ways to call this function, with a varying number of parameters:

```
myList = list('123')
myList = list([0, 1, 'two'])
```

- Note: `myList = [0, 1, 'two']` is just shorthand/syntactic sugar.

Calling Methods

- Once we have an instance of a class, we can call methods on it.
- We call a method on an object using the name of the object variable, followed by the dot operator, followed by the method's name, and its parameters (if needed).
  - This is called the dot-notation
- For example:

```
myList.append(4)

howMany = myList.count(4)

myList.sort()
```

Static Attributes and Methods

- Sometimes we want an attribute or method to be independent of a particular instance (object).
- We can think of it as being "shared" with all instances of the class.
- These attributes and methods are called static
  - not to be confused with instance attributes and instance methods, which require an object.
- To access a static attribute or method, you just use the class name, followed by the dot operator, followed by the attribute or method name.
- One such class that has many static attributes and methods is the `math` class.

Static Attributes and Methods

- For example, let's get access to the π static attribute
  - `math.pi    #shared/static attribute`
- Static methods can be useful
  - Some actions pertain to a class, not an instance of it
    - That is, they do not depend on any instance data.
    - Although they can be used by non-static methods.
- Example:
  - `math.sqrt(25.0)`

Using Custom Classes

- Are all built-in classes automatically included in all Python scripts?
  - Of course not! That would be wasteful.
  - Just a few are (like `list`).
- What if we want to use a class that isn't automatically included?
  - For example, although the `math` class is provided for us, it isn't automatically included when you start Python, since only some programs may need it.
- We could either
  - Add that class code to our script
  - Store the class code in a different file and `import` it.
    - This is the preferred approach

Modules

- Code that is stored in an external file, which can be used in another script, is referred to as a module
- We refer to a module by the file name without its extension.
- To use a module, we must import it.
- Let's imagine we have a class called `Database` that's within a file called `database.py`
  - For now, let's assume it's in the same directory as your main script.
- We could import the entire module as

```
import database
```

- Then, to access the `Database` class, we could type:

```
db = database.Database()
```

Modules

- Or we could just import the class from the module:

```
from database import Database
```

- And now we can make use of that class!

```
db = Database()
```

- To avoid name conflicts, we can import as some other name:

```
from database import Database as DB
```

- We can also import several features/classes from a module:

```
from database import Database, Query
from math import pow, sqrt
x = pow(2, 3) # the same as x = math.pow(2, 3)
from math import * # imports everything in the math module
x = pow(2, 3)
```

Packages

- As our projects grow, we'll likely include more and more modules.
  - It might be a pain to have to include each module
- A package is a collection of modules in a folder
  - The name of the package is the name of the folder
- Then we can just import this folder/package
  - As long as there is a "special" file in it to indicate this directory is part of a package
  - This file must be called `__init__.py` and is typically just empty

Packages

- Imagine an e-commerce project that wants to make use of a lot of modules:
  - database
  - products
  - square payments
  - stripe payments
- We could organize these hierarchically in directories
  - Each of which needs that special `__init__.py` file

```
parent_directory/
    main.py
    ecommerce/
        __init__.py
        database.py
        products.py
        payments/
            __init__.py
            square.py
            stripe.py
```

Packages

- Imagine that `Product` is a class defined in the `products` module.
- Now we can import from parts of packages:

```
from ecommerce import products
prod = products.Product()
from ecommerce import Product
prod = Product()
```

```
parent_directory/
    main.py
    ecommerce/
        __init__.py
        database.py
        products.py
        payments/
            __init__.py
            square.py
            stripe.py
```

CS 172

- In this course, we'll focus on OOD and OOP
- We want to look at interesting problems and identify the use of OOP
  - After all, not all problems call for OOP
  - Then use OOD to come up with the class designs.
- Along the way, we'll leverage key concepts of OOP/OOD like interfaces, composition, and inheritance for some fun and useful examples

Lab 1 – Using an Existing Class

- You are provided a class called `Spellchecker`.
- This class has several methods:
  - A constructor
    - takes a text file's name as a parameter. This file is basically a collection of valid words
  - `check(a)` boolean
    - Returns `True` if `a` is in the collection, `False` otherwise.
- Your script will
  - Read from a text file
  - Create an instance of `Spellchecker` based on a provided file that contains a list of valid words.
  - Use this `Spellchecker` object to find all the incorrectly spelled words in the text file.