Classes and Object-Oriented Programs

Chapter 10

What is an Object?

- Collection of data and methods
- The methods act on that data
- ‘Abstracts’ a concept
- In Python – defines a new **type**

- Example:
  - List
    - List.count()

What is an Abstract Data Type?

- Set of **objects** and **operations** on those objects
- Bound together so we can pass an object from one part of a program to another
- The specifications of those operations (functions) define the **interface** or **abstraction barrier**
  - This is where black box testing comes into play
- Supports key programming methodologies
  - **Decomposition** – program structure
  - **Abstraction** – implementation hiding

Creating an Abstract Data Type in Python

```
class IntSet(object):
    """ An IntSet is a set of integers"""
    #Information about the implementation
    #Value of the set is represented by a list of ints,
    #using variable self.vals
    #Each int in the set occurs in self.vals exactly once

    def __init__(self):
        """ Creates an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """ Assumes e is an integer.
            Inserts e into the set if not already a member"""
        if e not in self.vals:
            self.vals.append(e)
```

Some ‘classy’ terminology

- Everything indented under the **class** keyword is considered the **class definition**
- A **class definition** creates an object of type **type**
- The class object contains a set of objects of type **instancemethod**
  - For now these will always be functions
- Each function within the class definition is referred to as a **method** of that class
- The attributes (parameters) of a method are referred to as **method attributes**
- Data associated with a class (e.g. self.vals) are referred to as **data attributes**

Using a class (more terms)

- When we create an object of a class it is called **instantiating**
  - `int_set = IntSet()`
- When we access a method of an instance those are **attribute references**
  - `int_set.member`
  - `int_set.insert`
- A **class variable** is a variable that retains the same value for all instances of a class
  - More to come
- Access data attributes through **getters** and **setters**

Special “magic” methods

- `__init__()`
- `__str__()`
- `__del__()`

- **`__hash__()`**
- **`__eq__()`**

- `__lt, le, gt, ge, ne__()`
- `__add, sub, mul, div, ..._()`

- `__iter__()`, `__next__()`

What is a Representation Invariant?

- Which values of **data attributes** represent valid representations of class instances

- In IntSet we want only unique integers
  - The integers are assumed in our implementation
  - The members of the class enforce the **representation invariant**

  - The ‘RI’ is not enforced by the Python interpreter

Creating a class structure

- How can we classify various means of transportation
- What types of transportation do we want to classify?
  - Let’s say we are a cargo company wanting to move goods from one place to another
    - We want to pickup and deliver anytime, anywhere, on any continent
- What do they have in common
- What makes each distinct
- Can we group them, which are like another

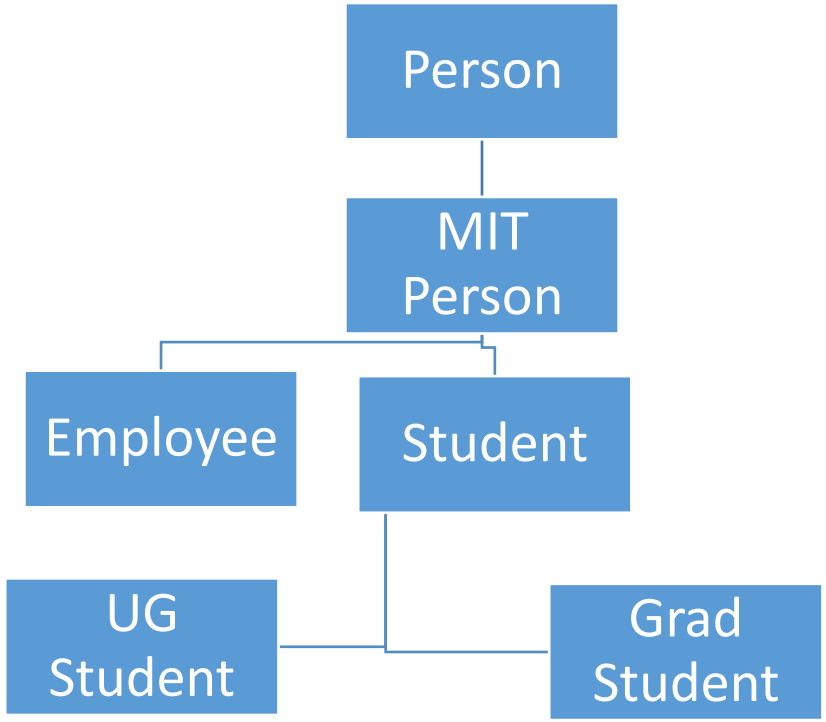
A Person but what type of Person?

- Guttag wants to model all students, faculty and staff at MIT
- What do all persons have in common?
  - Name
  - Birth Date
  - Birth Place
  - Parents
  - Gender
  - Ethnicity
  - Address
- Which are important to our problem

Superclass, subclass and inheritance

- A **subclass inherits** from its **superclass(es)**
  - Methods
  - Data Attributes
- A **subclass** can add new attributes
- A **subclass** can **override** a method of its **superclass**
- All actions performed against a **superclass** should ‘work’ against its **subtypes**
- The Python method `isinstance()` will return `True` for any type within a class hierarchy

Now for some inheritance



Encapsulation and Data Hiding

- These are two key concepts to **Object Oriented Programming**
- **Encapsulation** bundling together related data and methods
  - In the case of Python (and most OOP languages) into classes
- **Information hiding** means that clients (or users) of a class do not know about the implementation of the class but access advertised attributes based on the **class specification**
  - Python does not prevent you from directly accessing data in a class even when they don't have a getter
  - Python 3 supports the convention that `__variable` represents a private variable and cannot be accessed outside the class

Generators

- Allows client programs to iterate across a collection without having a copy of the whole collection

```
 def getStudents(self):
        """Return the students in the grade book one
at a time"""
        if not self.isSorted:
            self.students.sort()
            self.isSorted = True
        for s in self.students:
            yield s
```

Abstract Base Classes

- Sometimes you may want to create a class that only defines behavior of an object but does not become a part of the class hierarchy
- These are **abstract base classes** or **abc**s
- You can also create **abstractmethods**
- Can also be used in **multiple inheritance**

Mortgage Example

- Consider three types of common mortgages (home loans)
  - Fixed Term
    - Interest rate does not change over the term of the loan
  - Fixed with Points
    - Interest rate is ‘bought down’ with points
    - Basically paying interest in advance
  - Balloon Mortgage
    - Early mortgage period at a low ‘teaser’ rate
    - Balance of the mortgage at a higher rate
  - *Variable Rate*
    - *Interest rate adjusted periodically based on the market*