COMP 204

Object Oriented Programming (OOP) - Part II

Mathieu Blanchette

Object-Oriented Programming Vocabulary

From `http://interactivepython.org/courselib/static/thinkcspy/ClassesBasics/Glossary.html`

- **class**: A user-defined compound type. A class can also be thought of as a template for the objects that are instances of it.
- **object (aka instance)**: A bundle of data (attributes) built from a particular class.
- **attribute**: One of the named data items that makes up an object.
- **method**: A function that is defined inside a class definition and is invoked on instances of that class.

Object-Oriented Programming Vocabulary

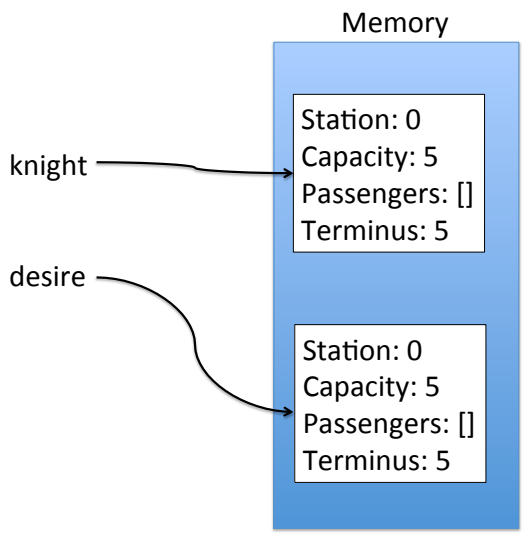
From `http://interactivepython.org/courselib/static/thinkcspy/ClassesBasics/Glossary.html`

- **initializer (or constructor) method**: A special method in Python (called __init__) that is invoked automatically to set a newly-created object's attributes to their initial state.
- **to instantiate**: To create an object (or instance) of a class, and to run its initializer.
- **object-oriented programming**: A powerful style of programming in which data and the operations that manipulate it are organized into classes and methods.
- **object-oriented language**: A language that provides features, such as user-defined classes and inheritance, that facilitate object-oriented programming.

Defining and instantiating a class (recap)

```
1  class Bus:
2      def __init__(self):
3          self.station = 0          # the position of the bus
4          self.capacity = 5         # the capacity of the bus
5          self.passengers = []      # the content of the bus
6          self.terminus = 5         # The last station
7  # end of Bus class definition
8
9  knight = Bus()  # Create an object of class Bus
10 desire = Bus()  # Create a second object of class Bus
```

Each object has its own set of attributes. The station, capacity, passengers, and terminus of knight and desire are different from each other.



The __str__(self) method

It is often useful to define how an object of given class should be converted to a string (e.g. for printed). This is achieved by defining the method __str__(self):

```
    def __str__(self):
        """
        Args: Self,
        Returns: String describing bus
        """
        return "Bus at station "+str(self.station) + \
               " contains passengers " + str(self.passengers
    )
```

Then:
my_bus = Bus()
print(my_bus) # will execute __str__() on my_bus to get a String, which then gets printed.

Putting it all together

See busSim_object_oriented.py

Notice how much simpler the simulation loop becomes!

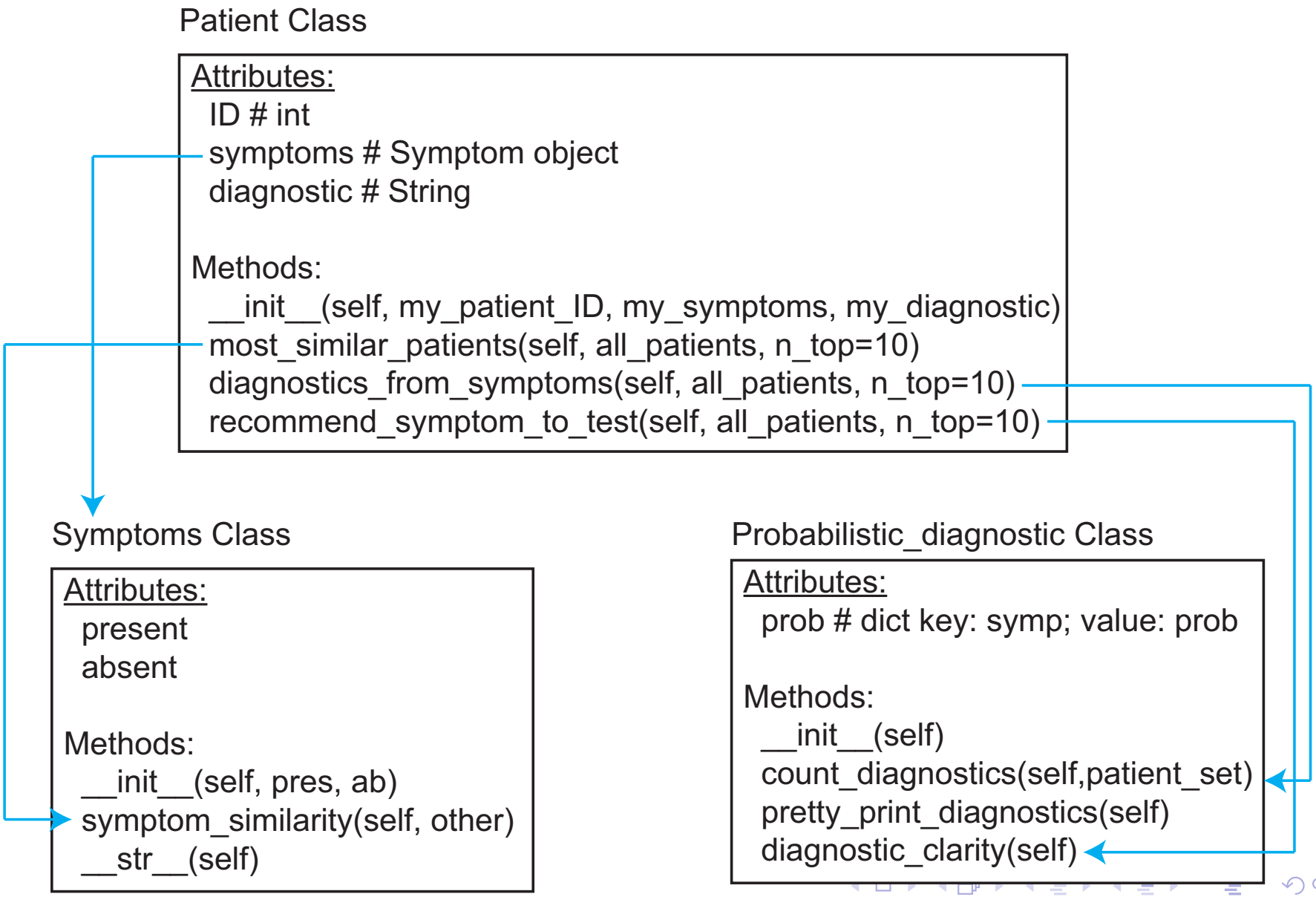
Advantage: All the code that pertains to the bus behavior is in the Bus class. The programmer of the simulation loop does not need to know all the details of the Bus class. It only needs to know how to use its methods properly.

Revisiting our medical_diagnostic program

Our program was a bit complicated because data and code pertaining to different concepts are intermingled.

- *Symptoms*
  - Data: Symptoms present and absent were store in a tuple. Programmer needs to remember that the first element of the tuple corresponds to the symptoms that are present, and the second to the symptoms that are absent.
  - Code: symptom_similarity function
- *Patients*
  - Data: patients' symptoms and diagnostics were stored in separate dictionaries: all_patients_symptoms, all_patients_diagnostics
  - Code: most_similar_patients(), diagnostics_from_symptoms(), recommend_symptom_to_test()
- *Probabilistic diagnostics*
  - Data: dictionary of diseases with associated probabilities.
  - Code: count_diagnostics(), pretty_print_diagnostics(), diagnostic_clarity():

Class diagram



Symptoms class

- Attributes:
  - present: Set of symptoms (Strings) that are present
  - absent: Set of symptoms (Strings) that are absent
- Methods:
  - __init__(self,pres,abs)
  - symptom_similarity(self, other)
  - __str__(self)

See symptoms.py

Probabilistic_diagnostic class

- Attributes:
  - prob: Dictionary of diagnostic probabilities
  - symptoms: Object of class Symptoms
  - diagnostic: String
- Methods:
  - __init__(self)
  - count_diagnostics(self,patient_set):
  - pretty_print_diagnostics(self):
  - diagnostic_clarity(self):

See probabilistic_diagnostic.py

Patient class

- Attributes:
  - ID: Integer
  - symptoms: Object of class Symptoms
  - diagnostic: String
- Methods:
  - __init__(self, my_patient_ID, my_symptoms, my_diagnostic)
  - most_similar_patients(self, all_patients, n_top=10)
  - diagnostics_from_symptoms(self, all_patients, n_top=10)
  - recommend_symptom_to_test(self, all_patients, n_top=10)
  - __str__(self)

See patient.py

Note: The Patient class needs to know about the Symptoms and Probabilistic_diagnostic classes. So:

```
1 # import the class Symptoms from file symptoms.py
2 from symptoms import Symptoms
3 # import the class Probabilistic_diagnostic from file
      probabilistic_diagnostic.py
4 from probabilistic_diagnostic import
      Probabilistic_diagnostic
```

Tester code

Our code that puts everything together is in a separate file: medical_diagnostic_tester.py.

It needs to import the three other modules:

```
1 from symptoms import Symptoms
2 from patient import Patient
3 from probabilistic_diagnostic import
      Probabilistic_diagnostic
```