COMP 204

Object Oriented Programming (OOP) - Examples

Yue Li
based on material from Mathieu Blanchette

MIBS Tea Time Advertisement

SCIENTISTS PRESENT WILL BE:

Dr. Tony Mittermaier: Chemical Biology and Chemical Physics - Protein structural mobility

Dr. Jackie Vogel: Cellular Biology and Biophysics - Spindle alignment in cell division of budding yeast

Dr. Arnold Hayer: Cellular Mechanisms - Coordination and signalling among migrating cells and cytoskeletal dynamics during collective cell migration

Dr. Mathieu Blanchette: Bioinformatics - Computational tools to study genomic evolution and gene expression regulation

Dr. Justin Marleau: Postdoctorate student at Dr. Guichard's lab, Biomathematics and Ecological Dynamics

MIBS

COMP204 Midterm grade

Class Statistics | User Statistics

View By: User | Apply

Midterm Exam Class Statistics

Number of submitted grades: 65 / 68

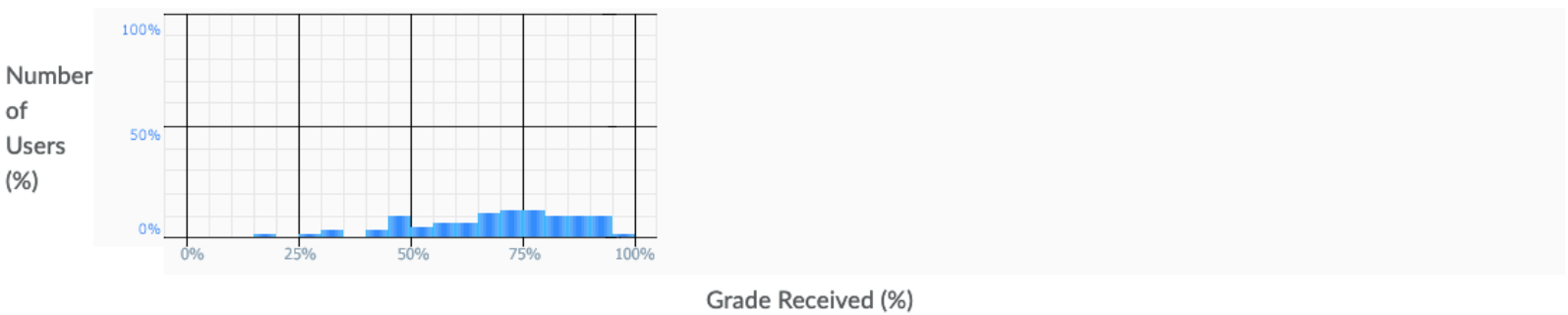
Minimum:		16.7 %
Maximum:		96 %
Average:		68.5 %

Mode: 93 %

Median: 72 %

Standard Deviation: 17.9 %

Grade Distribution



Outline

Object-Oriented Programming Vocabulary (recap)

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.
- **attribute**: One of the named data items that makes up an instance.
- **method**: A function that is defined inside a class definition and is invoked on instances of that class.
- **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.

Object-Oriented Programming Vocabulary (recap)

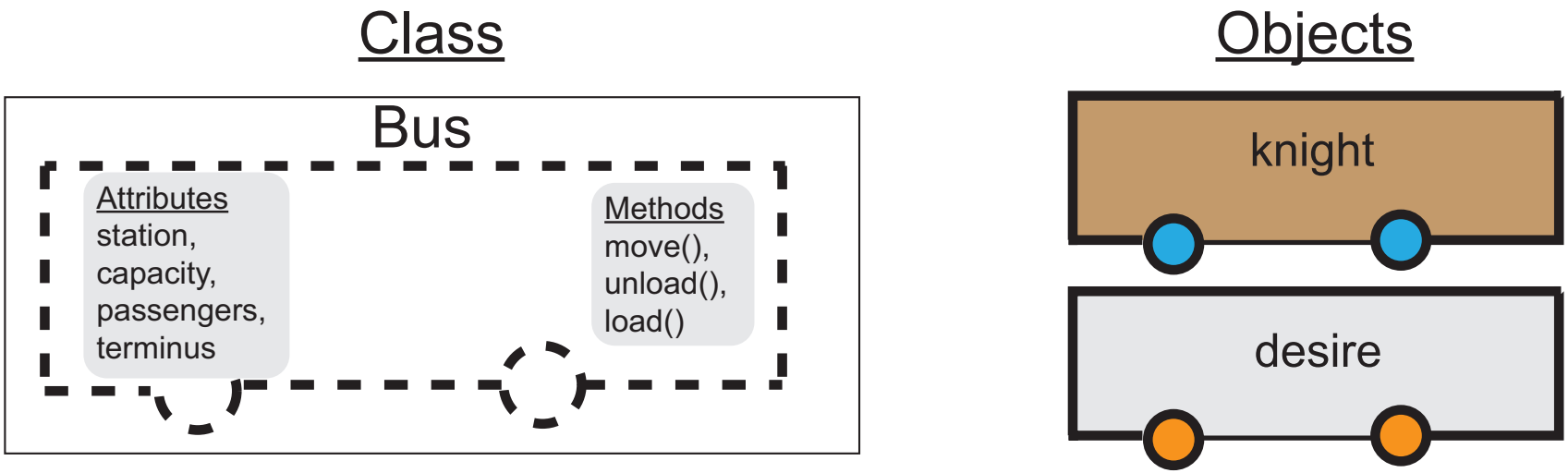
From http://interactivepython.org/courselib/static/thinkcspy/ClassesBasics/Glossary.html

- **object**: A compound data type that is often used to model a thing or concept in the real world. It bundles together the data and the operations that are relevant for that kind of data. Instance and object are used interchangeably.
- **instance**: An object whose type is of some class. Instance and object are used interchangeably.
- **to instantiate**: To create an 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.

Outline

Bus Class and Bus Objects

- To make use a class, we need to create objects of that class.
- An **object** is an instantiation of a class that contains all the data for a particular example of that class.



```python
class Bus:
    def __init__(self):
        self.station = 0          # the position of the bus
        self.capacity = 5         # the capacity of the bus
        self.passengers = []      # the content of the bus
        self.terminus = 5         # The last station

    def move(self):
        if self.station < self.terminus:
            self.station+=1

    def unload(self):
        unloaded = [dest for dest in self.passengers if
        ↪  dest==self.station]
        self.passengers = [dest for dest in
        ↪  self.passengers if dest!=self.station]
        return len(unloaded)

    def load(self, waiting_line):
        nb_board = min(len(waiting_line), \
                       self.capacity-len(self.passengers))
        people_boarding = waiting_line[0:nb_board]
        self.passengers.extend(people_boarding)
        return nb_board
```

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 the print function). This is achieved by defining the method __str__(self):

```
1  class Bus:
2      def __init__(self):
3          self.station = 0
4          self.passengers = []
5
6      def __str__(self):
7          """
8          Args: self
9          Returns: String describing bus
10         """
11         return "Bus at station "+str(self.station) + \
12                " contains passengers " +
                   ↪  str(self.passengers)
13
14 my_bus = Bus()
15 print(my_bus)  # will execute __str__() on my_bus to get a
   ↪  String, which then gets printed.
```

The OOP design makes the program a more readable

All the code that pertains to the bus behavior is in the Bus class.

See busSim_object_oriented.py

```
90  for time in range(0,simulation_duration):
91
92      # how many people are still waiting?
93      for station, waiting in waiting_at_stop.items():
94          nb_waiting_over_time[station][time]=len(waiting)
95
96      # move the buses up by one station
97      for bus in buses:
98          bus.move()
99
100     # bring new bus to station 0 at start_frequency
101     if time % start_frequency == 0 :
102         new_bus = Bus()
103         buses.append(new_bus)
104
105     # let people disembark if they are at their station
106     for bus in buses:
107         nb_disembarked = bus.unload()
108         nb_arrivals_over_time[bus.station][time]=nb_disembarked
109
110     # let people embark, until the bus is full
111     for bus in buses:
112         nb_boarded = bus.load(waiting_at_stop[bus.station])
113         del waiting_at_stop[bus.station][0:nb_boarded]
```

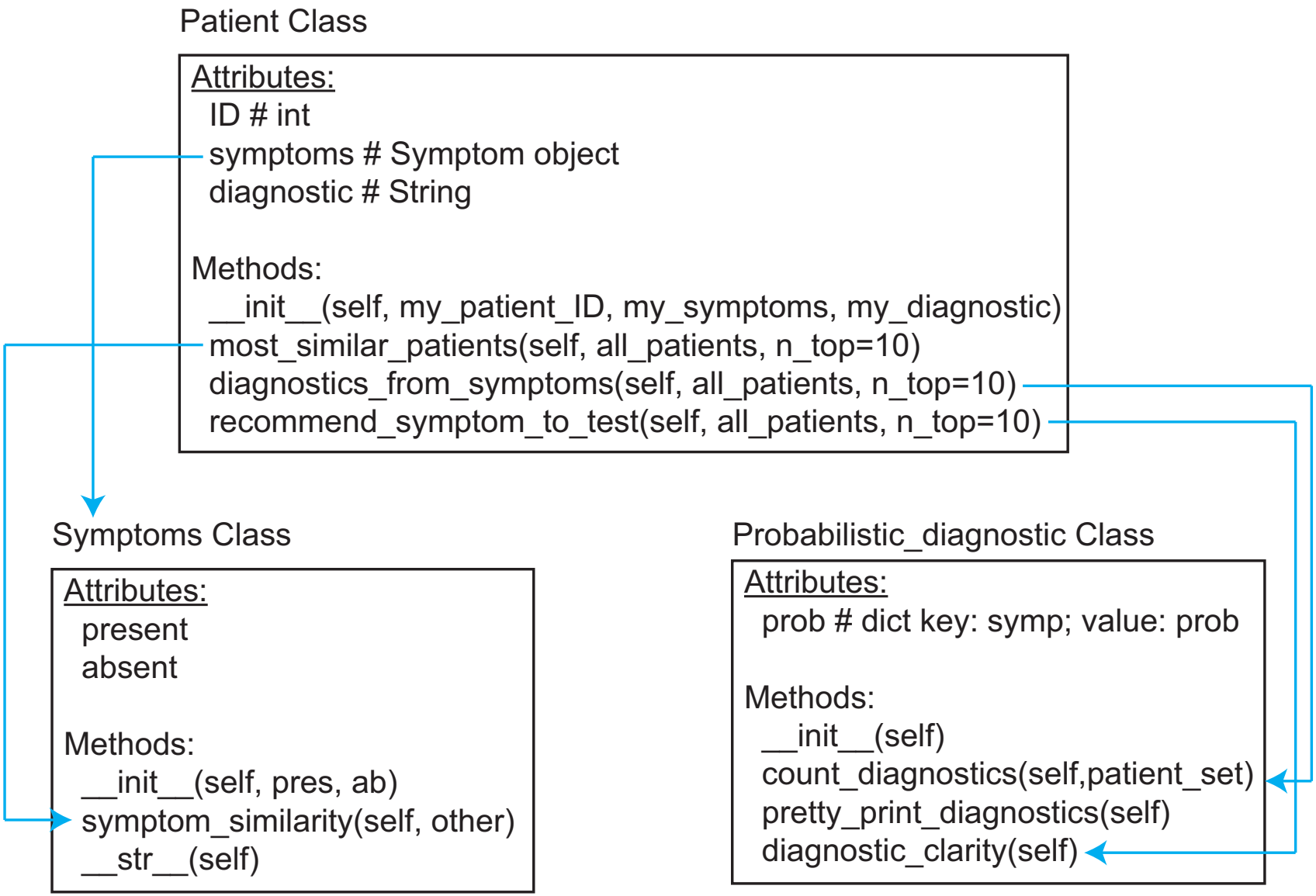
Outline

Outline

An OOP diagnostic program (similar but not equivalent to our A3)

- Encapsulation: Define separate classes for separate concepts:
  - `Symptoms`
  - `Patient`
  - `Probabilistic_diagnostics`
- Each class will be stored in a different Python file (also called a module):
  - symptoms.py
  - patient.py
  - probabilistic_diagnostic.py.
- A module can import code (classes, functions, etc.) from another module.
- This allows big programs to be broken down into smaller, more digestible chunks.
- Makes easier understanding, developing, and debugging large programs

OOP design of the medical diagnostic program



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`

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)`

Note: The Patient class needs to know about the `Symptoms` and `Probabilistic_diagnostic` classes. See patient.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`

Tester code

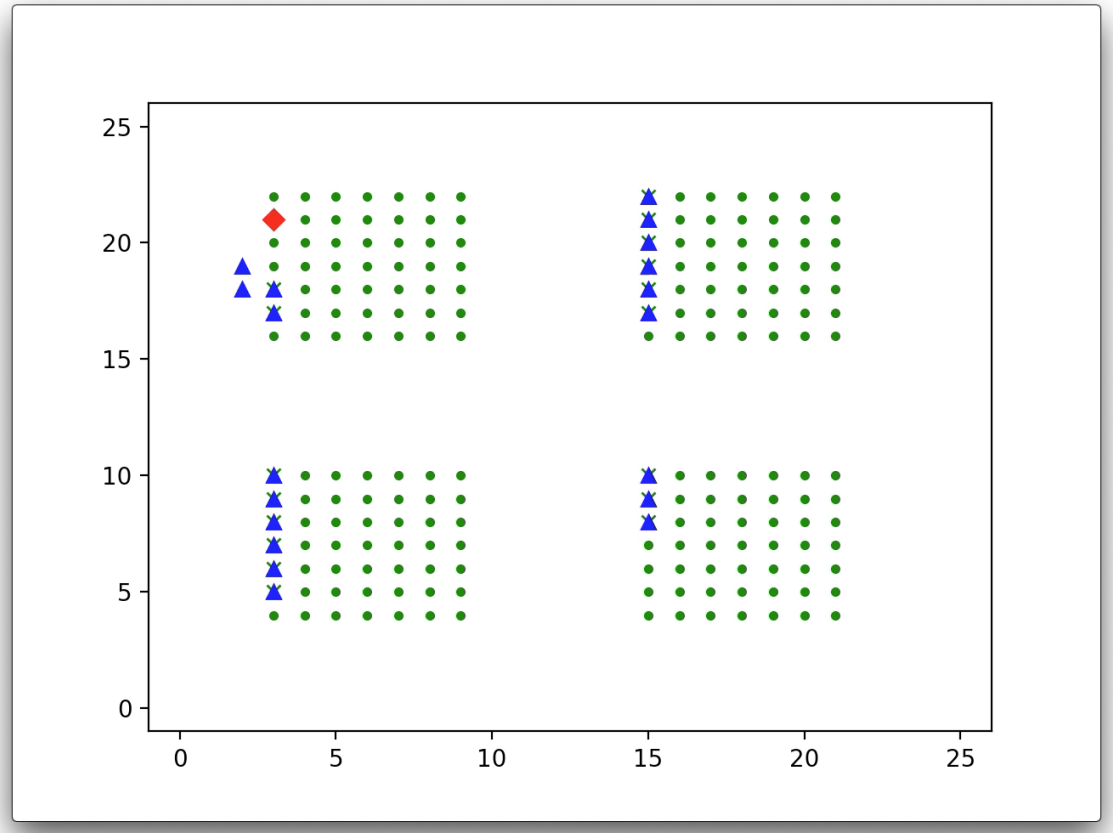
The test code that puts everything together is in a separate file: medical_diagnostic_tester.py.

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
```

Outline

An OOP simulation program for ecosystem (A4 preview)

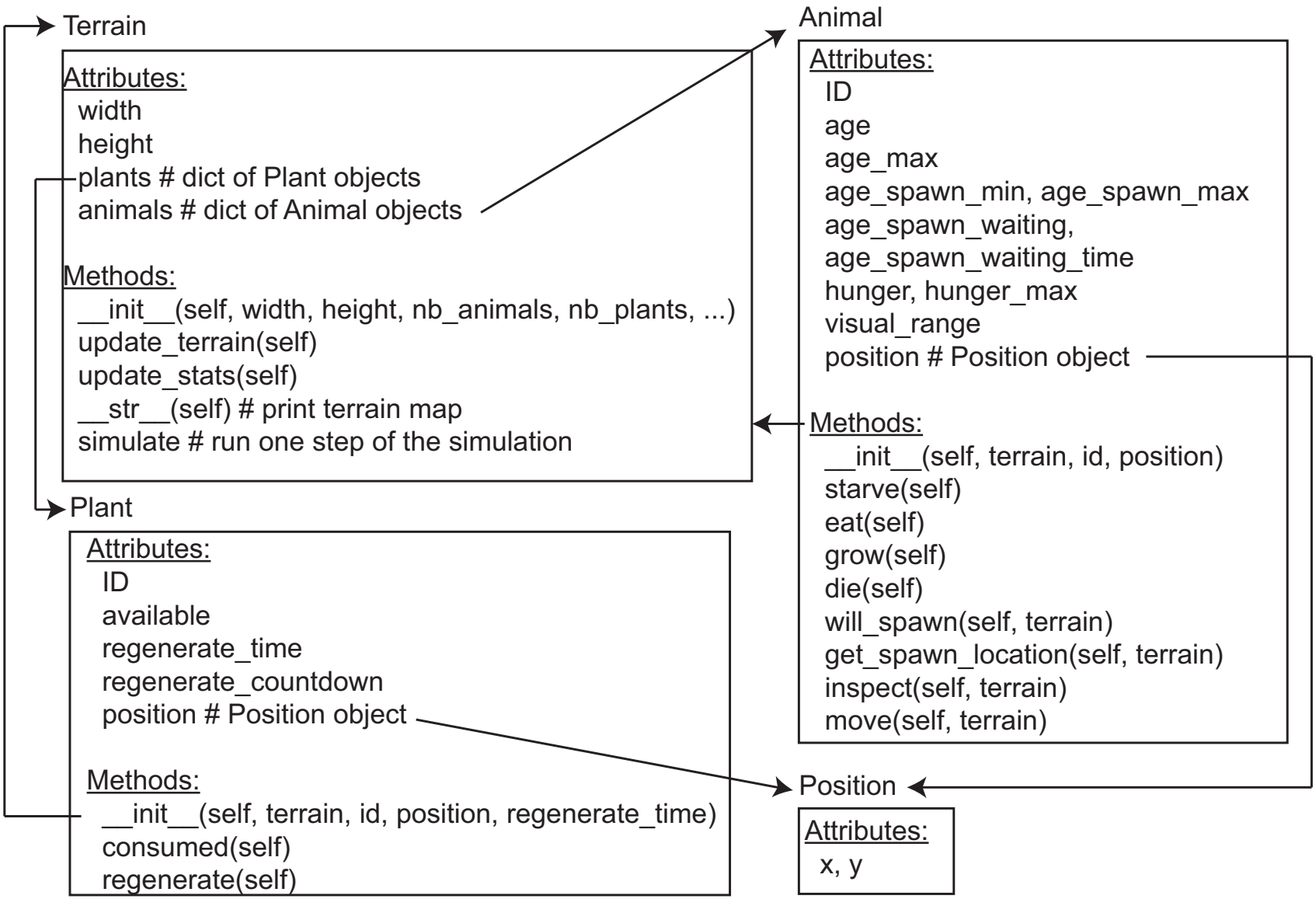


See the movie ecosim.mp4 file

Ecosim classes

- Animal
  - Attributes: id, age, age_max, age_spawn_min, age_spawn_max, spawn_waiting, spawn_waiting_time, hunger, hunger_max, visual_range,position
  - Methods: __init__, starve, eat, grow, die, will_spawn, inspect, move
- Plant
  - Attributes: id, available, regenerate_time, regenerate_countdown,position
  - Methods: __init__, consumed, regenerate
- Position
  - Attributes: x, y
- Terrain
  - Attributes: width, height, plants, animals
  - Methods: __init__, update_terrain, update_stats, __str__, simulate

Ecosim OOP overall design



Question about the design

How can we make the pray and predator behave differently while sharing other attributes and methods under the `Animal` class?

Next lecture: Class Inheritance