Inheritance, Representation, Efficiency

Aditya Iyer

Section Outline

Total Time: 50 mins

Mini Lecture: 10 mins

Question 1: 10 mins

Question 2: 10 mins

Question 1: 10 mins (Representation)

Question 1: 10 mins (Efficiency)

Inheritance

Why do we need inheritance?

- Often times, real world objects have many similar properties. For example, let us look at modes of transportation. In this example, we'll consider a plane, a car, a motorcycle and a boat.

```
#class variables (car)
num_of_wheels
num_of_engines

#instance variables
company_name
model_name
mileage

#then we also have a few methods
```

```
#class variables (motorcycle)
num_of_wheels
num_of_engines

#instance variables
company_name
model_name
mileage

#then we also have a few methods
```

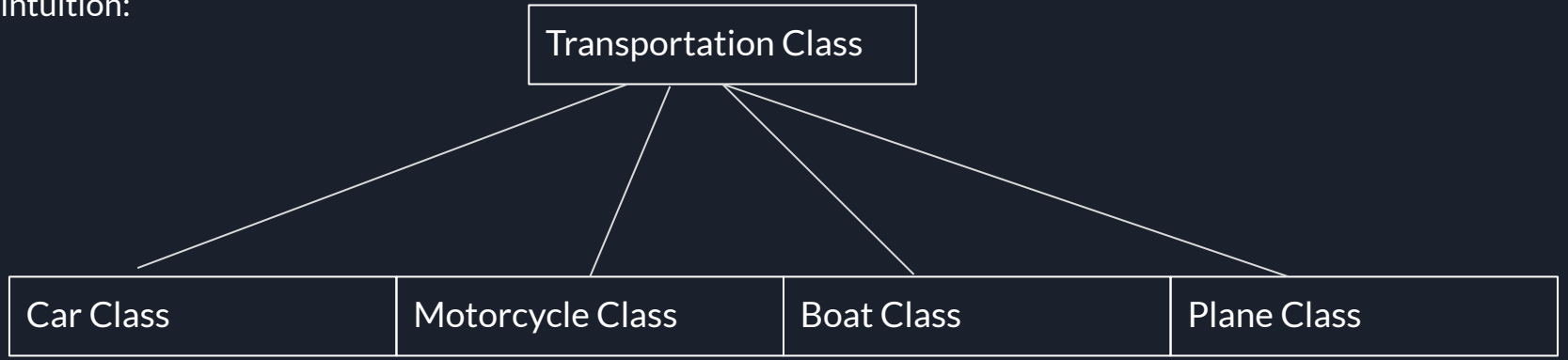
```
#class variables (boat)
num_of_wheels
num_of_engines

#instance variables
company_name
model_name
mileage

#then we also have a few methods
```

```
#class variables (plane)
num_of_wheels
num_of_engines

#instance variables
company_name
model_name
mileage

#then we also have a few methods
```

Do you see the similarities?

What now?

To avoid redundancy, we can have a super class and make the other classes inherit properties from this super class. Here is a visual representation my instructor showed me that helped me form an intuition:



Properties of the Super Class

- It contains methods common to all subclasses
- Contains class variables that are common to all subclasses. This is good to avoid redundancy.

Example of Superclass:

```
class Transportation:
    num_wheels = 4
    num_engines = 1

    def __init__(self, manufacturer, model,mileage):
        self.manufacturer = manufacturer
        self.model = model
        self.mileage = mileage

    def drive(#some param):
        #something here

    def park(#some pwram):
        #something here
```

To access the superclass from the subclass, pass it in. Eg:
class Car (Transportation):

Overriding

Subclasses can override class variables and also create new ones:

```
class Boat(Transportation):
    num_wheels = 0
    num_engines = 1
    medium = "water"
```

Overriding

- If a subclass overrides a method, then the new method definition (in the subclass) is the one that is used.
- Say you're working with an instance of the subclass (an object of type subclass) and if you want to access the method in the super class then you can do:

  super.method_name() #you could also use the superclass name instead of super

- Note, a class may inherit from multiple super classes.

Representation (Credits: Pamela Fox)

__str__ : this returns a human readable string representation of an object.

__repr__: returns a string that evaluates to an object with the same value.

```
from fractions import Fraction

one_third = 1/3
one_half = Fraction(1, 2)
```

```
>>> float.__str__(one_third)
'0.3333333333333333'
>>> Fraction.__str__(one_half)
'1/2'
```

```
from fractions import Fraction

one_half = Fraction(1, 2)
```

```
>>> Fraction.__repr__(one_half)
'Fraction(1, 2)'
>>> eval(Fraction.__repr__(one_half))
Fraction(1, 2)
```

I'd highly recommend taking a look at these slides created by Pamela Fox linked here.

Efficiency

Orders of Growth:

- Exponential Growth: $\Theta(b^n)$
- Quadratic Growth: $\Theta(n^2)$
- Linear Growth: $\Theta(n)$
- Logarithmic Growth: $\Theta(\log n)$
- Constant Time: $\Theta(1)$

Due to the section time constraint, this is in no way comprehensive. If you're interested in learning more about efficiency, linked here are the lectures by Professor Josh Hug. However, you should note that the level of material in those are way higher than what you need to know for CS61A.

Efficiency Examples 1

```
def func1():
    for i in range (5):
        for j in range(6):
            #do something, this takes constant time
```

Efficiency Examples 1 Solution

```
def func1():
    for i in range (5):
        for j in range(6):
            #do something, this takes constant time
```

Solution: $\Theta(n^2)$

Efficiency Examples 2

```
def func2():
    arr = []
    for i in range(10):
        arr.append(i)
```

Efficiency Examples 2 Solution

```
def func2():
    arr = []
    for i in range(10):
        arr.append(i)
```

Solution: Θ(n)

Efficiency Examples 3

```
def func3(n):
    return n*(n-1)/2
```

Efficiency Examples 3 Solution

```
def func3(n):
    return n*(n-1)/2
```

Solution: Θ(1)

Efficiency Examples 4

```
def func3(n):
    return n*(n-1)/2
```

Efficiency Examples 4

```
def func3(n):
    return n*(n-1)/2
```

Solution: Θ(log(n))

Exam Level Problem

6. (3.0 points) Mapping Time and Space

(a) The goal of the `maplink_to_list` function below is to map a linked list `lnk` into a Python list, applying a provided function `f` to each value in the list along the way. The function is fully written and passes all its doctests.

```
def maplink_to_list1(f, lnk):
    """Returns a Python list that contains f(x) for each x in Link LNK.
    >>> square = lambda x: x * x
    >>> maplink_to_list1(square, Link(3, Link(4, Link(5))))
    [9, 16, 25]
    """
    new_lst = []
    while lnk is not Link.empty:
        new_lst.append(f(lnk.first))
        lnk = lnk.rest
    return new_lst
```

i. (1.0 pt) What is the order of growth of `maplink_to_list1` in respect to the size of the input linked list `lnk`?

- ◯ Constant
- ◯ Logarithmic
- ◯ Linear
- ◯ Quadratic

(b) The next function, `maplink_to_list2`, serves the same purpose but is implemented slightly differently. This alternative implementation also passes all the doctests.

```
def maplink_to_list2(f, lnk):
    """Returns a Python list that contains f(x) for each x in Link LNK.
    >>> square = lambda x: x * x
    >>> maplink_to_list2(square, Link(3, Link(4, Link(5))))
    [9, 16, 25]
    """
    def map_link(f, lnk):
        if lnk is Link.empty:
            return Link.empty
        return Link(f(lnk.first), map_link(f, lnk.rest))

    mapped_lnk = map_link(f, lnk)
    new_lst = []
    while mapped_lnk is not Link.empty:
        new_lst.append(mapped_lnk.first)
        mapped_lnk = mapped_lnk.rest
    return new_lst
```

i. (1.0 pt) What is the order of growth of the alternative function, `maplink_to_list2`, in respect to the size of the input linked list `lnk`?

- ◯ Constant
- ◯ Logarithmic
- ◯ Linear
- ◯ Quadratic
- ◯ Exponential

(c) (1.0 pt) Which of the functions requires more **space** to run?

- ◯ `maplink_to_list1`
- ◯ `maplink_to_list2`
- ◯ They both require the same amount of space.

Exam Level Problem

i. (1.0 pt) What is the order of growth of `maplink_to_list1` in respect to the size of the input linked list `lnk`?

- ○ Constant
- ○ Logarithmic
- ● Linear
- ○ Quadratic
- ○ Exponential

(c) (1.0 pt) Which of the functions requires more **space** to run?

- ○ `maplink_to_list1`
- ● `maplink_to_list2`
- ○ They both require the same amount of space.

(b) The next function, `maplink_to_list2`, serves the same purpose but is implemented slightly differently. This alternative implementation also passes all the doctests.

```
def maplink_to_list2(f, lnk):
    """Returns a Python list that contains f(x) for each x in Link LNK.
    >>> square = lambda x: x * x
    >>> maplink_to_list2(square, Link(3, Link(4, Link(5))))
    [9, 16, 25]
    """
    def map_link(f, lnk):
        if lnk is Link.empty:
            return Link.empty
        return Link(f(lnk.first), map_link(f, lnk.rest))

    mapped_lnk = map_link(f, lnk)
    new_lst = []
    while mapped_lnk is not Link.empty:
        new_lst.append(mapped_lnk.first)
        mapped_lnk = mapped_lnk.rest
    return new_lst
```

i. (1.0 pt) What is the order of growth of the alternative function, `maplink_to_list2`, in respect to the size of the input linked list `lnk`?

- ○ Constant
- ○ Logarithmic
- ● Linear
- ○ Quadratic
- ○ Exponential

Anonymous Feedback Form