COMP 204

Object Oriented Programming (OOP) - Inheritance

Mathieu Blanchette and Yue Li
Inheritance

Motivation: We often need to create classes that are closely related but not identical to an existing class.

Example: We already have created a Bus class with

- attributes: station, capacity, passengers, terminus
- methods: `__init__`, move, unload, load, `__str__`

To represent a bus where passengers have to pay to board, we may want to add new attributes like the price of the ticket and the total amount of money present on the bus.

To represent an express bus that only stops at certain stops, we may want to add attributes about the stops the bus will make, and modify the load/unload methods accordingly.

Note: We want to continue to use all the other attributes and methods defined on the Bus class.
Inheritance

Bad approach: Code Duplication

▶ Create a completely separate PayBus class.
▶ Copy-paste the Bus class code into it.
▶ Add a new attribute `cost_of_ticket` and `cash_onboard`.
▶ **Bad** because:
  ▶ We now have two copies of the Bus code. If we want to make a change to the Bus class (e.g. bug fix, or improvement), we have to remember to make the same change to the PayBus class.
  ▶ Makes program large, difficult to understand.

Good approach: Inheritance

▶ Create a PayBus class that *inherits* the attributes and methods of the Bus class.
Inheritance

**Parent Class**
- Attributes: station, capacity, passengers, terminus
- Methods: move, unload, load
- __str__

**Bus**
- Attributes: station, capacity, passengers, terminus
- Methods: move, unload, load
- __str__

**Inheritance Class**
**PayBus**
- Added Attributes: cost_of_ticket, cash
- Overridden Methods: __init__, load # collect cash, __str__
The Bus *generic* class

```
see bus_generic.py

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):
        # code not shown

    def unload(self):
        # code not shown

    def load(self, waiting_line):
        # code not shown

    def __str__(self):
        # code not shown
```
Creating a subclass from the parent class

Define a subclass PayBus from the Bus class (see paybus0.py):

```python
from bus_generic import Bus

class PayBus(Bus):
    def __init__(self, price=2):
        Bus.__init__(self)
        self.cost_of_ticket = price  # cost of a ticket
        self.cash = 0  # the total cash onboard
```

- The PayBus class is a subclass of Bus because of this line:
  ```
  class PayBus(Bus):
  ```
- PayBus inherits the attributes and methods of the Bus class. Those get initialized by this line:
  ```
  Bus.__init__(self)
  ```
  which calls the `__init__` method of the parent Bus class.
  - Since we call the method directly on the class rather than on an object, `self` needs to be explicitly passed as an argument.
- PayBus extends the Bus class by adding two new attributes: `cost_of_ticket` and `cash`
PayBus class

The PayBus class has 6 attributes:

- station, capacity, passengers, terminus are inherited from the Bus class
- cost_of_ticket (unique to the PayBus class)
- cash (unique to the PayBus class)

Methods:

- All of 4 non-initializer methods are inherited from the Bus class (move, unload, load, __str__)
- Therefore, we can directly use the methods already defined in the Bus class
- We can also override these methods (next)

```python
9  stm_bus = PayBus(price=2)
10  stm_bus.load([3,4,5,2,6,2,3])
11  stm_bus.station = 3
12  stm_bus.cash = 134
13  print(stm_bus)
14  # Bus at station 3 contains passengers [3, 4, 5, 2, 6].
```
Overriding methods from the parent class

Goal: Make new passengers pay price_of_ticket and add cash
Approach: Override the load() method of Bus (paybus.py)

```python
class PayBus(Bus):
    def __init__(self, price=2):
        Bus.__init__(self)
        self.cost_of_ticket = price  # cost of a ticket
        self.cash = 0  # the total cash onboard

    def load(self, waiting_line):
        number_boarding = Bus.load(self, waiting_line)
        self.cash += number_boarding * self.cost_of_ticket
        return number_boarding
```

The new load() method first calls the load method of the parent class. It then updates the cash on the PayBus object.

```python
stm_bus = PayBus(2)
```
Overriding the \_\_str\_\_ method from the generic class

We can also override the \_\_str\_\_ method to make it print information about the amount of cash on board.

class PayBus(Bus):
    def \_\_init\_\_(self, price=2):
        Bus.\_\_init\_\_(self)
        self.cost_of_ticket = price  # cost of a ticket
        self.cash = 0  # the total cash onboard

    def load(self, waiting_line):
        number_boarding = Bus.load(self, waiting_line)
        self.cash += number_boarding * self.cost_of_ticket
        return number_boarding

    def \_\_str\_\_(self):
        return Bus.\_\_str\_\_(self) +
        "\n        Cost of ticket: " +
        str(self.cost_of_ticket) +
        "; Cash collected: " +
        + str(self.cash)
```python
stm_bus = PayBus(2)
stm_bus.load([3,4,5,2,6,2,3])
print("Cash = ", stm_bus.cash)  # Prints Cash = 10

print(stm_bus)
# Bus at station 3 contains passengers [3, 4, 5, 2, 6]
# Cost of ticket: 2; Cash collected: 10

generic_bus = Bus()
stm_bus = PayBus(2)

print(generic_bus)
# Bus at station 0 contains passengers []

print(stm_bus)
# Bus at station 0 contains passengers []
# Cost of ticket: 2; Cash collected: 0

generic_bus.load([4,2,5,3,6,4,2,4])
print(generic_bus)
# Bus at station 0 contains passengers [4, 2, 5, 3, 6]

stm_bus.load([4,2,5,3,6,4,2,4])
print(stm_bus)
# Bus at station 0 contains passengers [4, 2, 5, 3, 6]
# Cost of ticket: 2; Cash collected: 10
```
Multiple classes can inherit from the same generic class

**Parent Class**

**Bus**
- Attributes: station, capacity, passengers, terminus
- Methods: move, unload, load, __str__

**Inheritance Class**

**PayBus**
- Added Attributes: cost_of_ticket, cash
- Overriden Methods: __init__, load # collect cash, __str__

**ExpressBus**
- Added Attributes: stops
- Overriden Methods: __init__, unload, load
- New Methods: load_safe
A class like Bus can have many different subclasses. We will create an ExpressBus subclass (see express_bus.py).

An express bus differs from a normal bus in that it only stops at certain predetermined stop.

Note: We could also have decided that the ExpressBus class is a subclass of the PayBus class, if we needed the functionality of payments.
We now need to override the load and unload methods to allow boarding/unloading only at a station where the bus stops.

class ExpressBus(Bus):
    def __init__(self, my_stops):
        Bus.__init__(self)
        self.stops = my_stops  # list of stations  
        # where the bus will stop

    def unload(self):
        if self.station in self.stops:
            return Bus.unload(self)  # allow unloading
        else:
            return []  # no unloading

    def load(self, waiting_line):
        if self.station in self.stops:  # allow loading
            return Bus.load(self, waiting_line)
        else:  # no loading
            return 0
See the difference between the Bus and ExpressBus classes:

```python
exp = ExpressBus([0,2,4])  # bus will stop only at 0, 2, 4
slow = Bus()
exp.load([5,3,1])
slow.load([5,3,1])
print(exp)  # Bus at station 0 has passengers [5, 3, 1]
print(slow) # Bus at station 0 has passengers [5, 3, 1]

exp.move()
slow.move()
exp.load([4,3])  # Nobody gets loaded onto express bus
slow.load([4,3]) # But passengers can board the slow bus
print(exp)  # Bus at station 1 has passengers [5, 3, 1]
print(slow) # Bus at station 1 has passengers [5, 3, 1, 4, 3]
```
Defining new methods (not overriding existing) for subclass

Subclasses can also have their own methods:

```python
def load_safe(self, waiting_line):
    # allows passengers to board only if
    # their destinations are among the express bus stops
    should_board = [p for p in waiting_line if p in self.stops]
    number_boarding = min(len(should_board),
                           self.capacity - len(self.passengers))
    people_boarding = should_board[0:number_boarding]
    self.passengers.extend(people_boarding)
    return number_boarding
```

load_safe() method only allows boarding for people whose destination is among the stops the express buss will make.

```python
exp = ExpressBus([0,2,4])
exp.load_safe([4,2,3,1,3,2])
print(exp)  # Bus at station 0 has passengers [4, 2, 2]
slow = Bus()
slow.load_safe([4,2,3,1,3,2])
#AttributeError: 'Bus' object has no attribute 'load_safe'
```