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

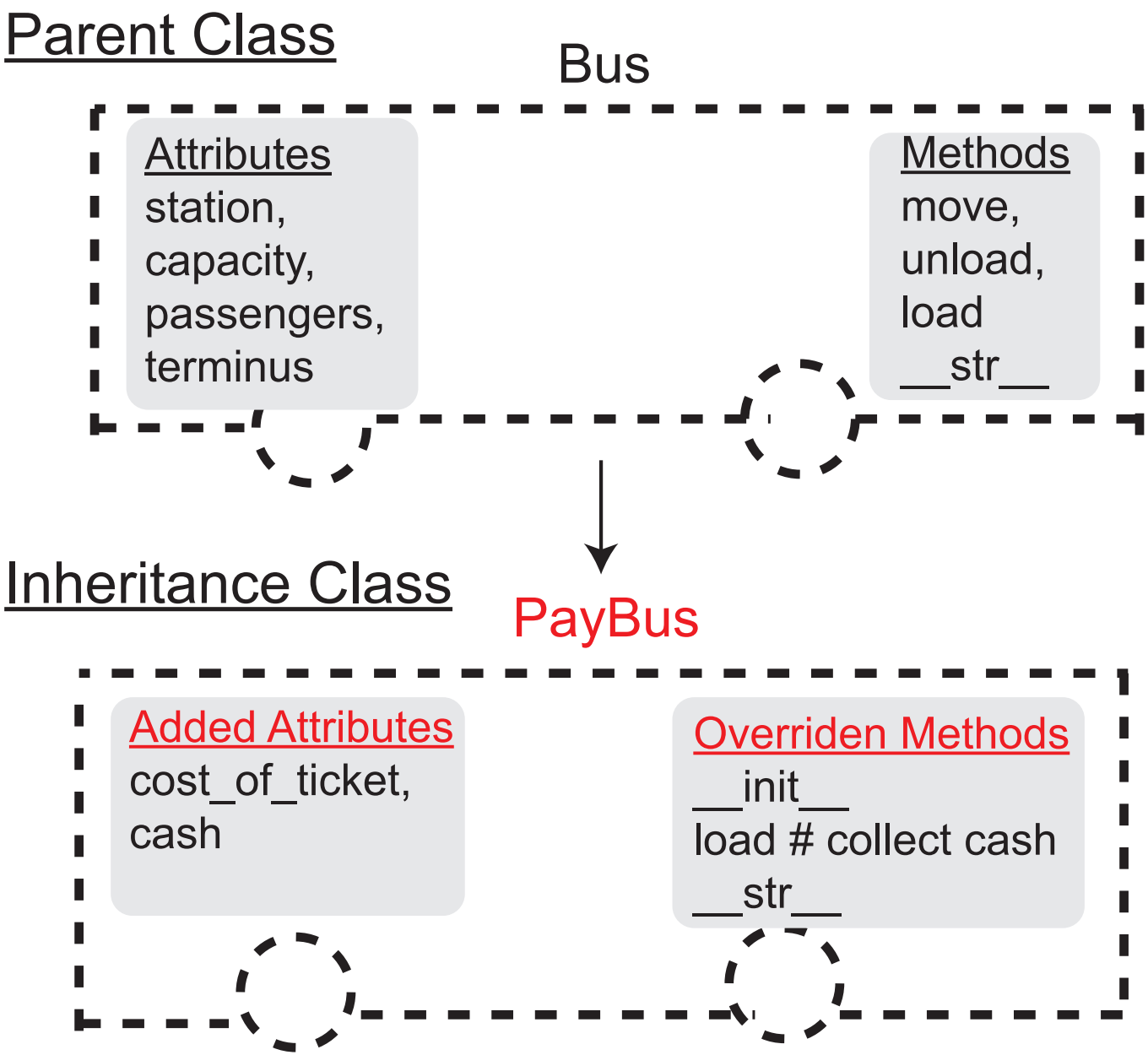
Bad approach: Code Duplication

- 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



The Bus *generic* class

see bus_generic.py

```
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
8      def move(self):
9          # code not shown
10
11     def unload(self):
12         # code not shown
13
14     def load(self, waiting_line):
15         # code not shown
16
17     def __str__(self):
18         # code not shown
```

Creating a subclass from the parent class

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

```
1  from bus_generic import Bus
2
3  class PayBus(Bus):
4      def __init__(self, price=2):
5          Bus.__init__(self)
6          self.cost_of_ticket = price # cost of a ticket
7          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)

```
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`)

```
3  class PayBus(Bus):
4      def __init__(self, price=2):
5          Bus.__init__(self)
6          self.cost_of_ticket = price # cost of a ticket
7          self.cash = 0               # the total cash onboard
8
9      def load(self, waiting_line):
10         number_boarding = Bus.load(self, waiting_line)
11         self.cash += number_boarding * self.cost_of_ticket
12         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.

```
20
21
22 stm_bus = PayBus(2)
```

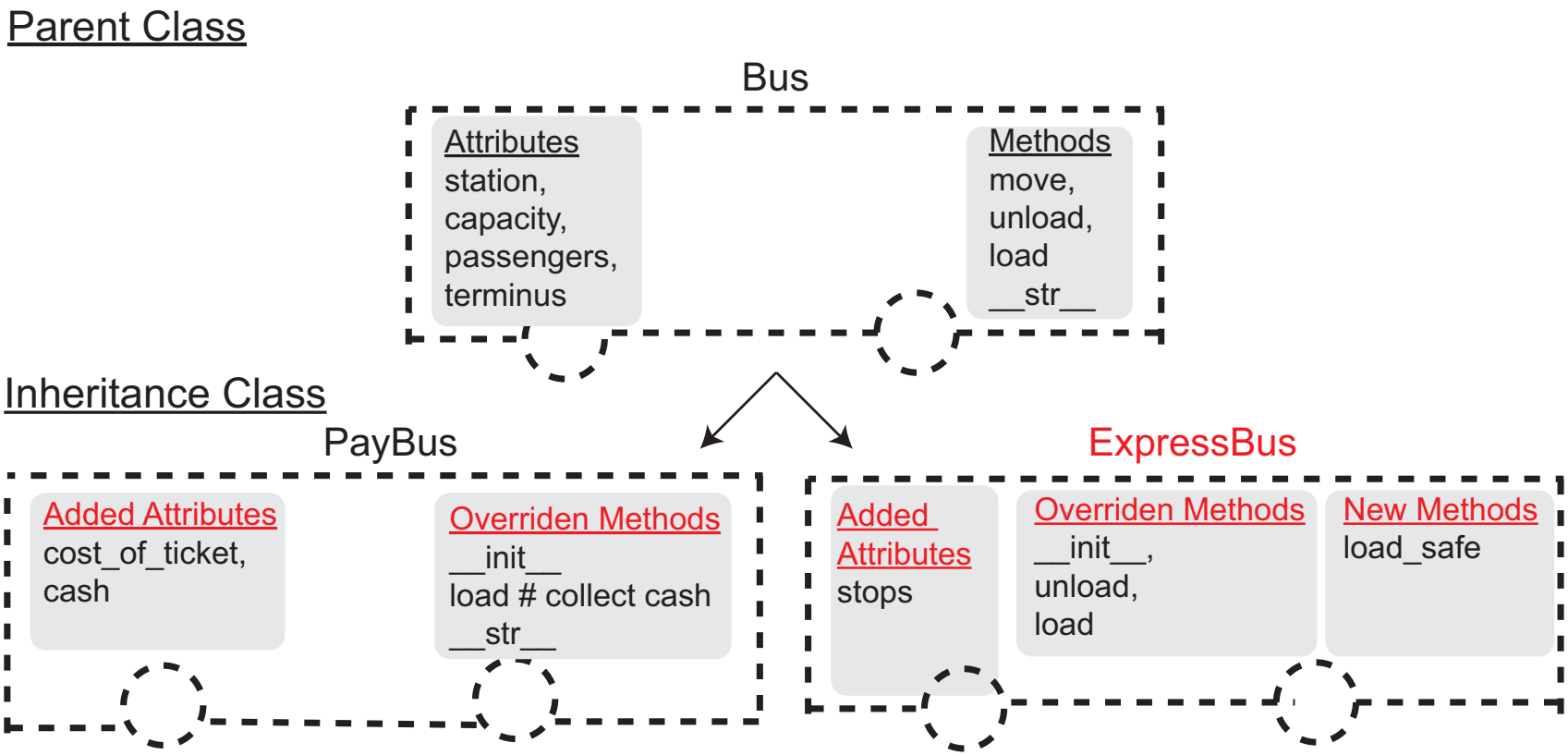
Overriding the `__str__` method from the generic class

We can also override the `__str__` method to make it print information about the amount of cash on board.

```
3  class PayBus(Bus):
4      def __init__(self, price=2):
5          Bus.__init__(self)
6          self.cost_of_ticket = price # cost of a ticket
7          self.cash = 0               # the total cash onboard
8
9      def load(self, waiting_line):
10         number_boarding = Bus.load(self, waiting_line)
11         self.cash += number_boarding * self.cost_of_ticket
12         return number_boarding
13
14     def __str__(self):
15         return Bus.__str__(self)+\
16                "\nCost of ticket: " +\
17                str(self.cost_of_ticket) +\
18                "; Cash collected: "\
19                + str(self.cash)
```

```
22  stm_bus = PayBus(2)
23  stm_bus.load([3,4,5,2,6,2,3])
24  print("Cash = ",stm_bus.cash) # Prints Cash = 10
25
26  print(stm_bus)
27  #Bus at station 3 contains passengers [3, 4, 5, 2, 6]
28  #Cost of ticket: 2; Cash collected: 10
29
30  generic_bus = Bus()
31  stm_bus = PayBus(2)
32
33  print(generic_bus)
34  #Bus at station 0 contains passengers []
35
36  print(stm_bus)
37  #Bus at station 0 contains passengers []
38  #Cost of ticket: 2; Cash collected: 0
39
40  generic_bus.load([4,2,5,3,6,4,2,4])
41  print(generic_bus)
42  # Bus at station 0 contains passengers [4, 2, 5, 3, 6]
43
44  stm_bus.load([4,2,5,3,6,4,2,4])
45  print(stm_bus)
46  #Bus at station 0 contains passengers [4, 2, 5, 3, 6]
47  #Cost of ticket: 2; Cash collected: 10
```

Multiple classes can inherit from the same generic class



ExpressBus class

```
1  from bus_generic import Bus
2
3  class ExpressBus(Bus):
4      def __init__(self, my_stops):
5          Bus.__init__(self)
6          self.stops = my_stops # list of stations
```

- 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.

ExpressBus class

We now need to override the load and unload methods to allow boarding/unloading only at a station where the bus stops.

```
3  class ExpressBus(Bus):
4      def __init__(self, my_stops):
5          Bus.__init__(self)
6          self.stops = my_stops # list of stations
7                                # where the bus will stop
8
9      def unload(self):
10         if self.station in self.stops:
11             return Bus.unload(self) # allow unloading
12         else:
13             return []               # no unloading
14
15     def load(self, waiting_line):
16         if self.station in self.stops:  # allow loading
17             return Bus.load(self, waiting_line)
18         else:                           # no loading
19             return 0
```

ExpressBus class

See the difference between the Bus and ExpressBus classes:

```
33 exp = ExpressBus([0,2,4]) # bus will stop only at 0, 2, 4
34 slow = Bus()
35 exp.load([5,3,1])
36 slow.load([5,3,1])
37 print(exp) # Bus at station 0 has passengers [5, 3, 1]
38 print(slow) # Bus at station 0 has passengers [5, 3, 1]
39
40 exp.move()
41 slow.move()
42 exp.load([4,3])  # Nobody gets loaded onto express bus
43 slow.load([4,3])  # But passengers can board the slow bus
44 print(exp)  # Bus at station 1 has passengers [5, 3, 1]
45 print(slow) # Bus at station 1 has passengers [5, 3, 1, 4,
   ↪  3]
46
```

Defining new methods (not overriding existing) for subclass

Subclasses can also have their own methods:

```
21  def load_safe(self, waiting_line):
22      # allows passengers to board only if
23      # their destinations are among the express bus stops
24      should_board = [p for p in waiting_line \
25                      if p in self.stops]
26      number_boarding = min(len(should_board),\
27                    self.capacity -len(self.passengers))
28      people_boarding = should_board[0:number_boarding]
29      self.passengers.extend(people_boarding)
30      return number_boarding
```

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

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