Inheritance

Lectures adapted from python-course.eu

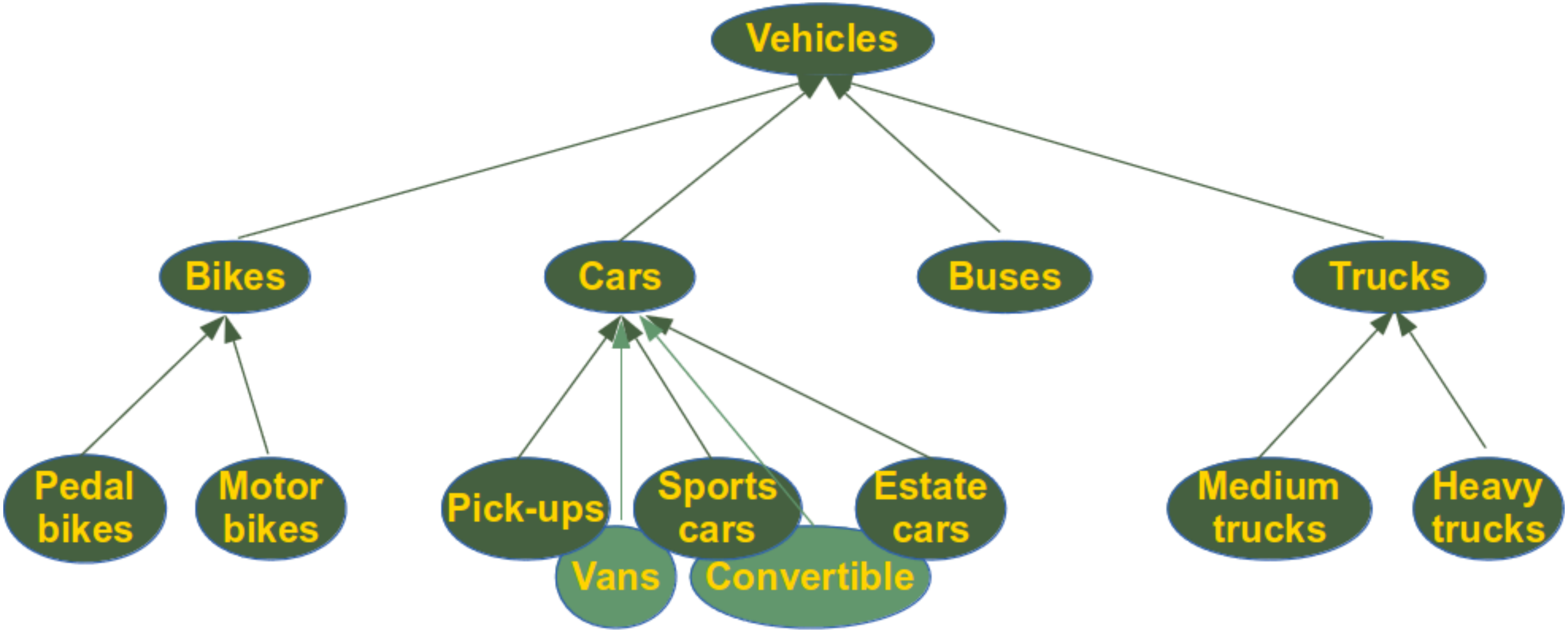
Instructor: Paruj Ratanaworabhan

Introduction

- Inheritance allows programmers to create classes that are built upon existing classes
- A class created through inheritance inherits the attributes and methods of the parent class
    - This is good for supporting code reusability

Definition

- The class from which a class inherits is called the parent or superclass.
- A class which inherits from a superclass is called a subclass
- There exists a hierarchical relationship between classes
  - Similar to relationships or categorizations that we know from real life


Vehicles
Bikes
Cars
Buses
Trucks
Pedal bikes
Motor bikes
Pick-ups
Vans
Sports cars
Convertible
Estate cars
Medium trucks
Heavy trucks

Syntax of Inheritance in Python

The syntax for a subclass definition looks like this:

```
class DerivedClassName(BaseClassName):
    pass
```

Instead of the `pass` statement, there will be methods and attributes like in all other classes

Simple Inheritance Example

```
class Robot:

    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print("Hi, I am " + self.name)

class PhysicianRobot(Robot):
    pass

x = Robot("Marvin")
y = PhysicianRobot("James")

print(x, type(x))
print(y, type(y))

y.say_hi()
```

```
<__main__.Robot object at 0x7fd0080b3ba8> <class '__main__.Robot'>
<__main__.PhysicianRobot object at 0x7fd0080b3b70> <class
'__main__.PhysicianRobot'>
Hi, I am James
```

type and isinstance

```
x = Robot("Marvin")
y = PhysicianRobot("James")

print(isinstance(x, Robot), isinstance(y, Robot))
print(isinstance(x, PhysicianRobot))
print(isinstance(y, PhysicianRobot))

print(type(y) == Robot, type(y) == PhysicianRobot)
```

```
True True
False
True
False True
```

We see that `isinstance` returns `True` if we compare an object either with the class it belongs to or with the superclass.

Whereas the equality operator for `type` only returns `True`, if we compare an object with its own class.

type and isinstance

```
class A:
    pass

class B(A):
    pass

class C(B):
    pass

x = C()
print(isinstance(x, A))
```

What gets printed out?

Overriding

```
class Robot:

    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print("Hi, I am " + self.name)

class PhysicianRobot(Robot):

    def say_hi(self):
        print("Everything will be okay! ")
        print(self.name + " takes care of you!")
```

```
y = PhysicianRobot("James")
y.say_hi()
```

- A method of a parent class gets overridden by simply defining a method with the same name in the child class

Overriding

- A subclass often needs additional methods with additional functionalities that do not exist in the superclass.
- An instance of the PhysicianRobot class will need the method heal so that the physician can do a proper job
- We will also add an attribute health_level to the Robot class, which can take a value between 0 and 1
    - The robots will 'come to live' with a random value between 0 and 1
    - If the health_level of a Robot is below 0.8, it will need a doctor
    - We write a method needs_a_doctor which returns True if the value is below 0.8 and False otherwise
    - The 'healing' in the heal method is done by setting the health_level to a random value between the old health_level and 1; this value is calculated by the uniform function of the random module.

Overriding

```
import random

class Robot:

    def __init__(self, name):
        self.name = name
        self.health_level = random.random()

    def say_hi(self):
        print("Hi, I am " + self.name)

    def needs_a_doctor(self):
        if self.health_level < 0.8:
            return True
        else:
            return False

class PhysicianRobot(Robot):

    def say_hi(self):
        print("Everything will be okay! ")
        print(self.name + " takes care of you!")


    def heal(self, robo):
        robo.health_level = random.uniform(robo.health_level, 1)
        print(robo.name + " has been healed by " + self.name + "!")
```

Overriding

```
doc = PhysicianRobot("Dr. Frankenstein")

rob_list = []
for i in range(5):
    x = Robot("Marvin" + str(i))
    if x.needs_a_doctor():
        print("health_level of " + x.name + " before healing: ", x.health_level)
        doc.heal(x)
        print("health_level of " + x.name + " after healing: ", x.health_level)
    rob_list.append((x.name, x.health_level))

print(rob_list)
```

```
health_level of Marvin0 before healing:  0.5562005305000016
Marvin0 has been healed by Dr. Frankenstein!
health_level of Marvin0 after healing:  0.7807651150204282
health_level of Marvin1 before healing:  0.40571527448692757
Marvin1 has been healed by Dr. Frankenstein!
health_level of Marvin1 after healing:  0.4160992532325318
health_level of Marvin2 before healing:  0.3786957462635925
Marvin2 has been healed by Dr. Frankenstein!
health_level of Marvin2 after healing:  0.5474124864506639
health_level of Marvin3 before healing:  0.6384666796845331
Marvin3 has been healed by Dr. Frankenstein!
health_level of Marvin3 after healing:  0.6986491928780778
health_level of Marvin4 before healing:  0.5983126049766974
Marvin4 has been healed by Dr. Frankenstein!
health_level of Marvin4 after healing:  0.6988801787833587
[('Marvin0', 0.7807651150204282), ('Marvin1', 0.4160992532325318), ('Marvin2', 0.5474124864506639), ('Marvin3', 
0.6986491928780778), ('Marvin4', 0.6988801787833587)]
```

Overriding

- When we override a method, we sometimes want to reuse the method of the parent class and add some new stuffs
- We could use the super function:

```
class PhysicianRobot(Robot):

    def say_hi(self):
        super().say_hi()
        print("and I am a physician!")


doc = PhysicianRobot("Dr. Frankenstein")
doc.say_hi()
```

```
Hi, I am Dr. Frankenstein
and I am a physician!
```

What We Have Learned

- Benefits of inheritance in OO programming
- type versus is instance
- Method overriding