Object Oriented Programming using Python (I)

Lecture(6)

Class Inheritance

Prepared by: Ahmed Eskander Mezher

University of Information Technology and Communications

College of Business Informatics

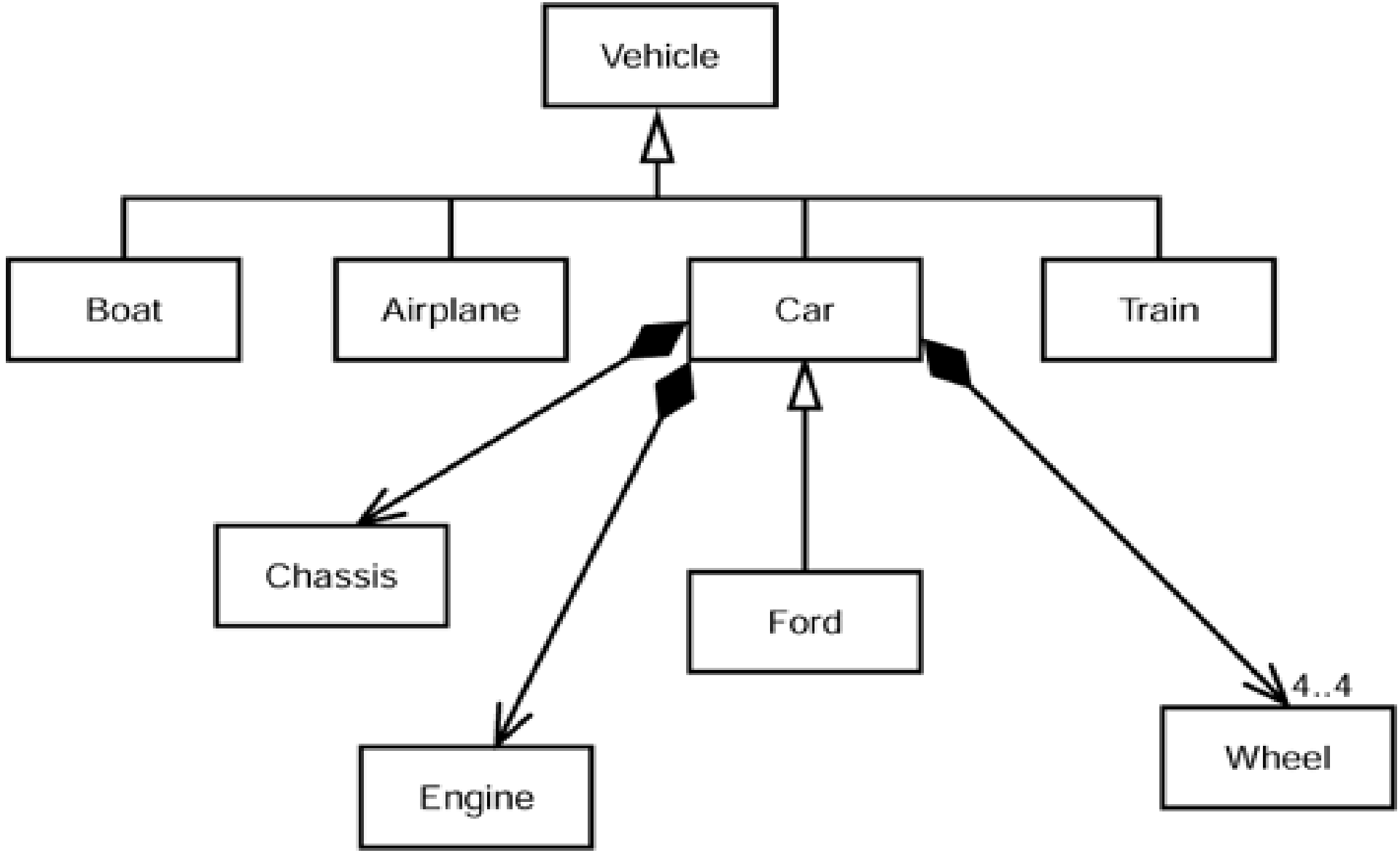
Class Inheritance

What is Inheritance?

Inheritance is a powerful feature in object oriented programming.

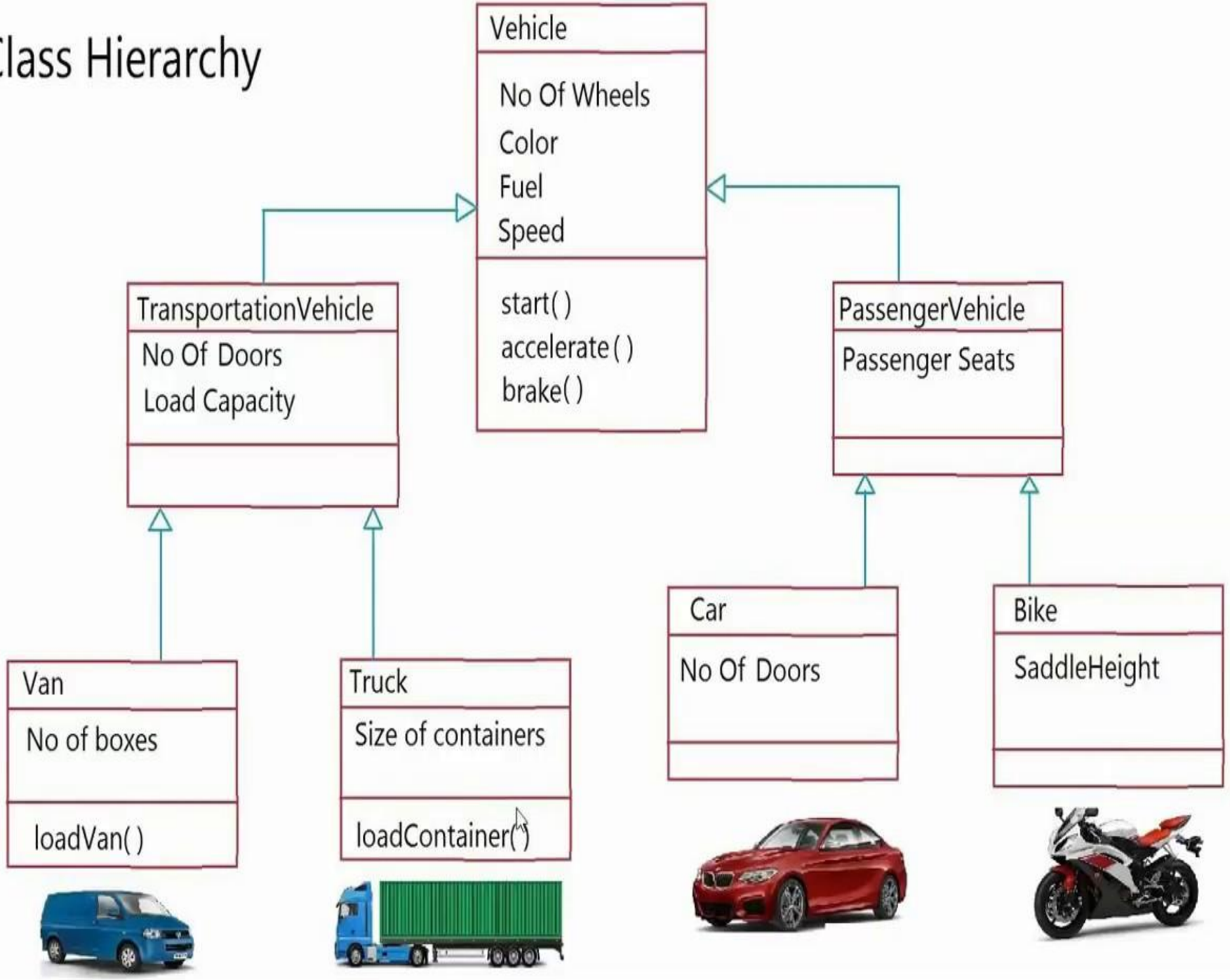
It refers to defining a new class with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

- Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.
- The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.



class diagram for the transportation package depicts inheritance and class associations.

Class Hierarchy



Syntax

➢Derived class inherits features from the base class, adding new features to it. This results into **re-usability of code**

Python Inheritance Syntax

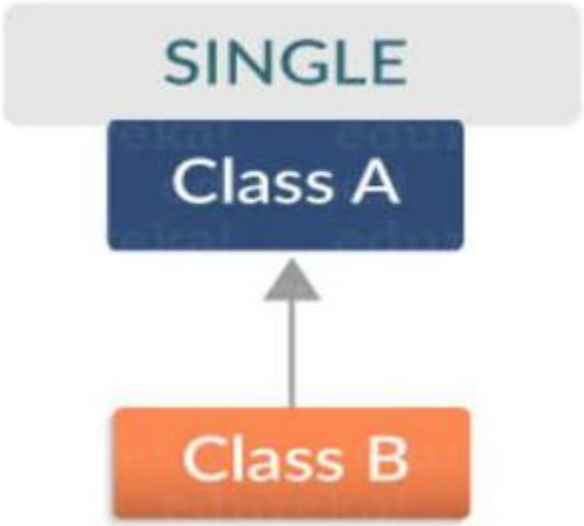
```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

TYPES OF INHERITANCE

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hybrid Inheritance

SINGLE INHERITANCE

- In which there is one base class and one derived class



Example

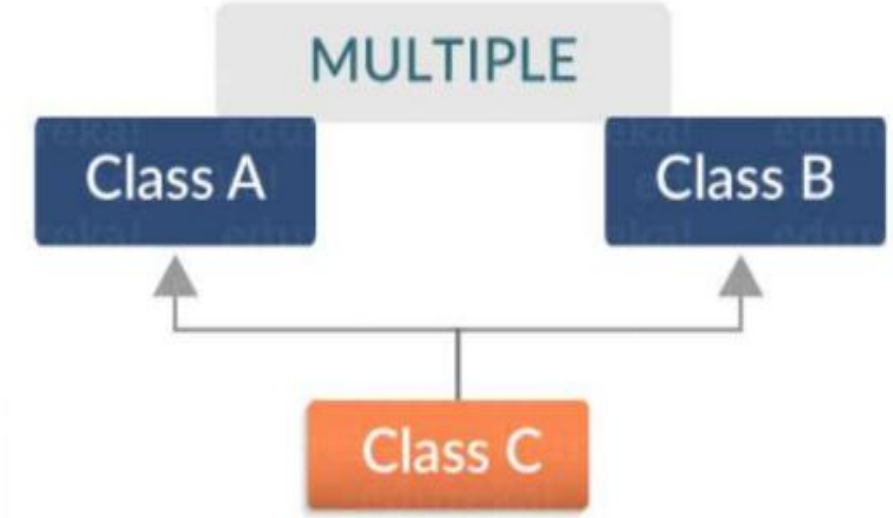
```
#!/usr/bin/python

class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print "Calling parent constructor"

   def parentMethod(self):
      print 'Calling parent method'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
   def __init__(self):
      print "Calling child constructor"

   def childMethod(self):
      print 'Calling child method'

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

When the code is executed, it produces the following result

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

MULTIPLE INHERITANCE

- Multiple inheritance is possible in python
- A class can be derived from more then one base classes. The syntax for multiple inheritance is similar to single inheritance
- Here is an example of multiple inheritance



inher2.py - C:\Users\Rula\AppData\Local\Programs\Python\Python37-32\inher2.py (3.7.0)

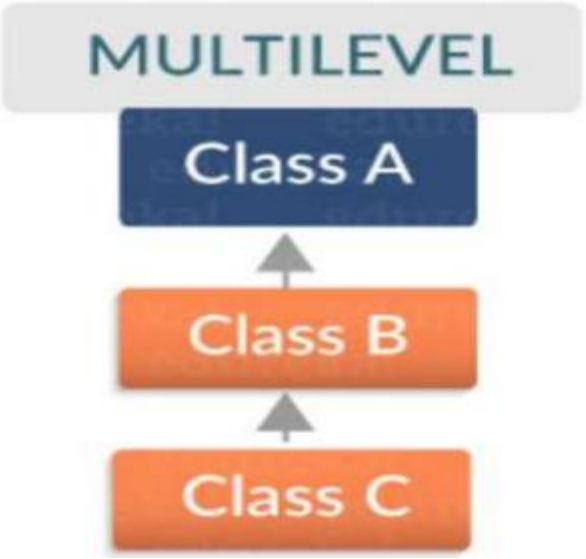
File Edit Format Run Options Window Help

```
class first:
    def sum(self,a,b):
        c=a+b
        return c
class second:
    def sub(self,x,y):

        z=x-y
        return (z)
class third(first,second):
    def display(self):
        return(self.sub(5,4))

obj=third()
print(obj.sum(5,5))
print(obj.display())
```

```
>>>
= RESTART: C:/Users/
10
1
>>>
```

MULTILEVEL INHERITANCE

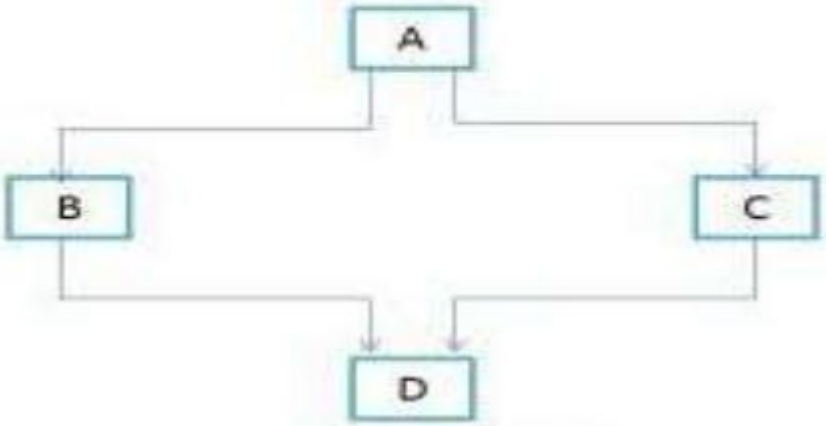
- Multilevel inheritance is also possible in Python like other Object Oriented programming languages. We can inherit a derived class from another derived class, this process is known as multilevel inheritance. In Python, multilevel inheritance can be done at any depth.



```
class first():
    def m1(self,a,b):
        c=a+b;
        return c;
class second(first):
    def m2(self):
        print('M2 method is called')
class third(second):
    def m3(self):
        print('M3 method is called')


thirdobj=third()
print(thirdobj.m1(20,25))
```

```
Python 3.6.5 (v3.6.5:f5
4)] on win32
Type "copyright", "cred
>>>
========== RESTART:
45
>>>
```

HYBRID INHERITANCE



Hybrid Inheritance

```
class a:
    def m(self):
        print("message from class a")
class b(a):
    def m(self):
        print("message from class b")
class c(a):
    def m(self):
        print("message from class c")
class d(b,c):
    def m(self):
        print("message from class d")
        a.m(self)
        b.m(self)
        c.m(self)


obj1=d()
obj1.m()
```

```
>>>
 RESTART: C:\Users\Rula\i
nce.py
message from class d
message from class a
message from class b
message from class c
>>>
```

```
# python-inheritance.py
class CompanyMember:
        '''Represents Company  Member.'''
        def __init__(self, name, designation, age):
                self.name = name
                self.designation = designation
                self.age = age
        def tell(self):
                '''Details of an employee.'''
                print('Name: ', self.name,'\nDesignation : ',self.designation, '\nAge : ',self.a


class FactoryStaff(CompanyMember):
        '''Represents a Factory Staff.'''
        def __init__(self, name, designation, age, overtime_allow):
                CompanyMember.__init__(self, name, designation, age)
                self.overtime_allow = overtime_allow
                CompanyMember.tell(self)
                print('Overtime Allowance : ',self.overtime_allow)
```

```
class OfficeStaff(CompanyMember):
        '''Represents a Office Staff.'''
        def __init__(self, name, designation, age, travelling_allow):
                CompanyMember.__init__(self, name, designation, age)
                self.marks = travelling_allow
                CompanyMember.tell(self)
                print('Traveling Allowance : ',self.marks)


o1=OfficeStaff('ali','manager',54,'n')
o2=FactoryStaff('saad','Maintenance technician',35,'y')
```

Q-Write python program that represent the following UML

Here's a UML class diagram for an inheritance hierarchy that keeps track of people in a department store: