Inheritance & Polymorphism

Learning Outcome

- Explain the concept of inheritance
- Apply superclasses and subclasses
- Apply inheritance in a program
- Explain the concept of polymorphism

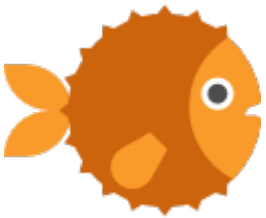
Inheritance

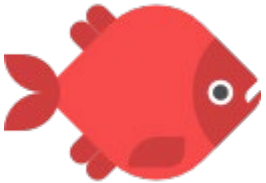
Animal

A horse is an animal

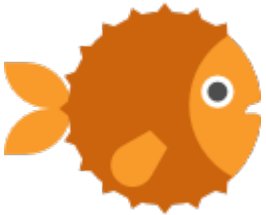
A cow is an animal

A sheep is an animal

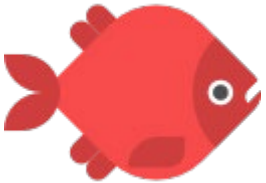




Fish



A puffer fish is a fish



A red fish is a fish

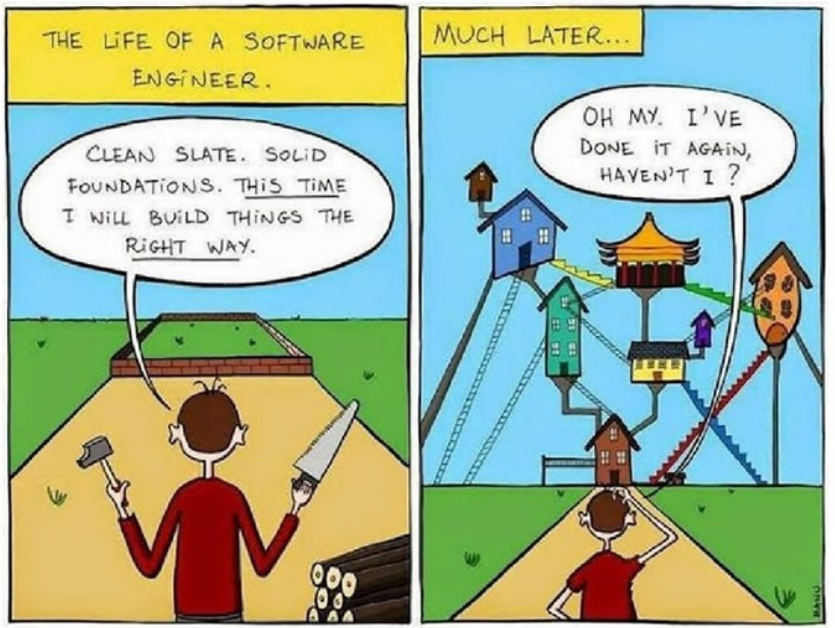
A clown fish is a fish

An angel fish is a fish

Inheritance

Inheritance is ‘Is-A’ relationship

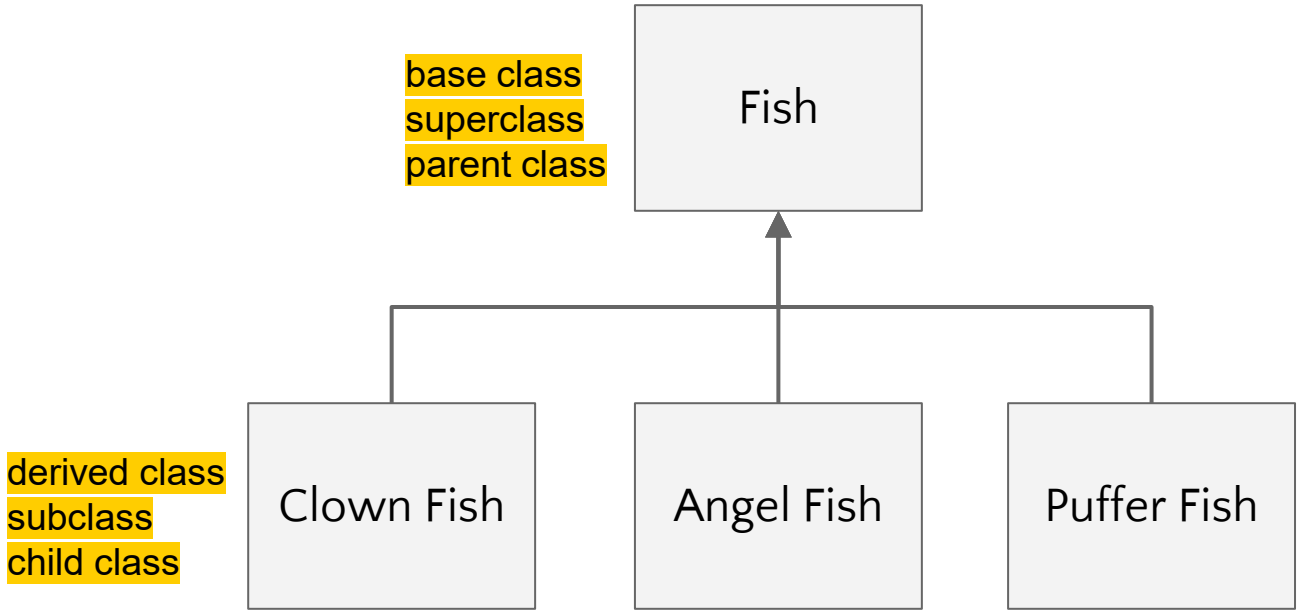
Inheritance enables new classes to receive—or inherit—the properties and methods of existing classes. It override parts with specialized behavior and extend it with additional functionality.



Fish is the superclass (parent class) of Clown Fish, Angel Fish and Puffer Fish.

Clown Fish, Angel Fish and Puffer Fish are subclasses or child classes of Fish.

Child class (subclass) inherits from the parent class (superclass).



Why Inheritance?

Reusability

Child class can reuse the methods defined in the parent class without rewriting the same.

Extensibility

Extend the parent class logic so that meaningful implementation of the parent class method can be designed in the child class.

Base class, Fish contains 2 public data attributes first_name and last_name.

Derived class also will inherit the 2 data attributes from base class, Fish.

Class definition of base class, Fish

```
class Fish:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Class definition of derived class, ClownFish

```
class ClownFish(Fish):
    def lives_in_sea(self):
        print("The clown fish lives in the sea")
```

Class definition of base class, Fish

```
class Fish:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Class definition of derived class, ClownFish

```
class ClownFish(Fish):
    def lives_in_sea(self):
        print("The clown fish lives in the sea")
```

Assume both class definitions in Fish.py

```
#Test program in Test.py

import Fish as f

fish = f.Fish('Parent', 'Fish')
print(fish.first_name)

clown = f.ClownFish('Child', 'Fish')
print(clown.first_name)
print(clown.last_name)
```

```
Parent
Child
Fish
```

ClownFish inherits data attributes from Fish the parent class

Class definition of base class, Fish

```
class Fish:
    def __init__(self, first_name, last_name):
        self.__first_name = first_name
        self.__last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Data attributes are encapsulated

Access through accessor method for the private data attribute

Access through the inherited accessor methods for the private data attribute

Class definition of derived class, ClownFish

```
class ClownFish(Fish):
    def lives_in_sea(self):
        print("The clown fish lives in the sea")
```

Assume both class definitions in Fish.py

```
#Test program in Test.py

import Fish as f

fish = f.Fish('Parent', 'Fish')
print(fish.first_name)

clown = f.ClownFish('Child', 'Fish')
print(clown.first_name)
print(clown.last_name)
```

AttributeError: 'Fish' object has no attribute 'first_name'

ClownFish inherits data attributes from Fish the parent class

Base class, Fish contains the __init__, swim and swim_backwards methods.

Derived class ClownFish contains the lives_in_sea method. At the same time, it inherits the 3 methods from base class, Fish.

Class definition of base class, Fish

```
class Fish:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Class definition of derived class, ClownFish

```
class ClownFish(Fish):
    def lives_in_sea(self):
        print("The clown fish lives in the sea")
```

Class definition of base class, Fish

```
class Fish:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Class definition of derived class, ClownFish

```
class ClownFish(Fish):
    def lives_in_sea(self):
        print("The clown fish lives in the sea")
```

Assume both class definitions in Fish.py

```
#Test program in Test.py

import Fish as f

fish = f.Fish('Parent', 'Fish')
fish.swim()

clown = f.ClownFish('Child', 'Fish')
clown.swim()
clown.swim_backwards()
clown.lives_in_sea()
```

```
The fish is swimming
The fish is swimming
The fish can swim backwards
The clown fish lives in the sea
```

ClownFish inherits methods from Fish the parent class

Car dealership example

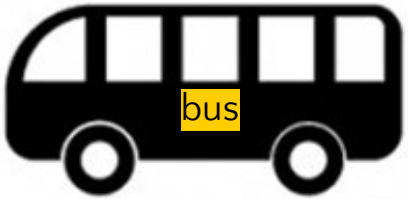
make

price

Vehicle

model

mileage



Load capacity

Passenger capacity

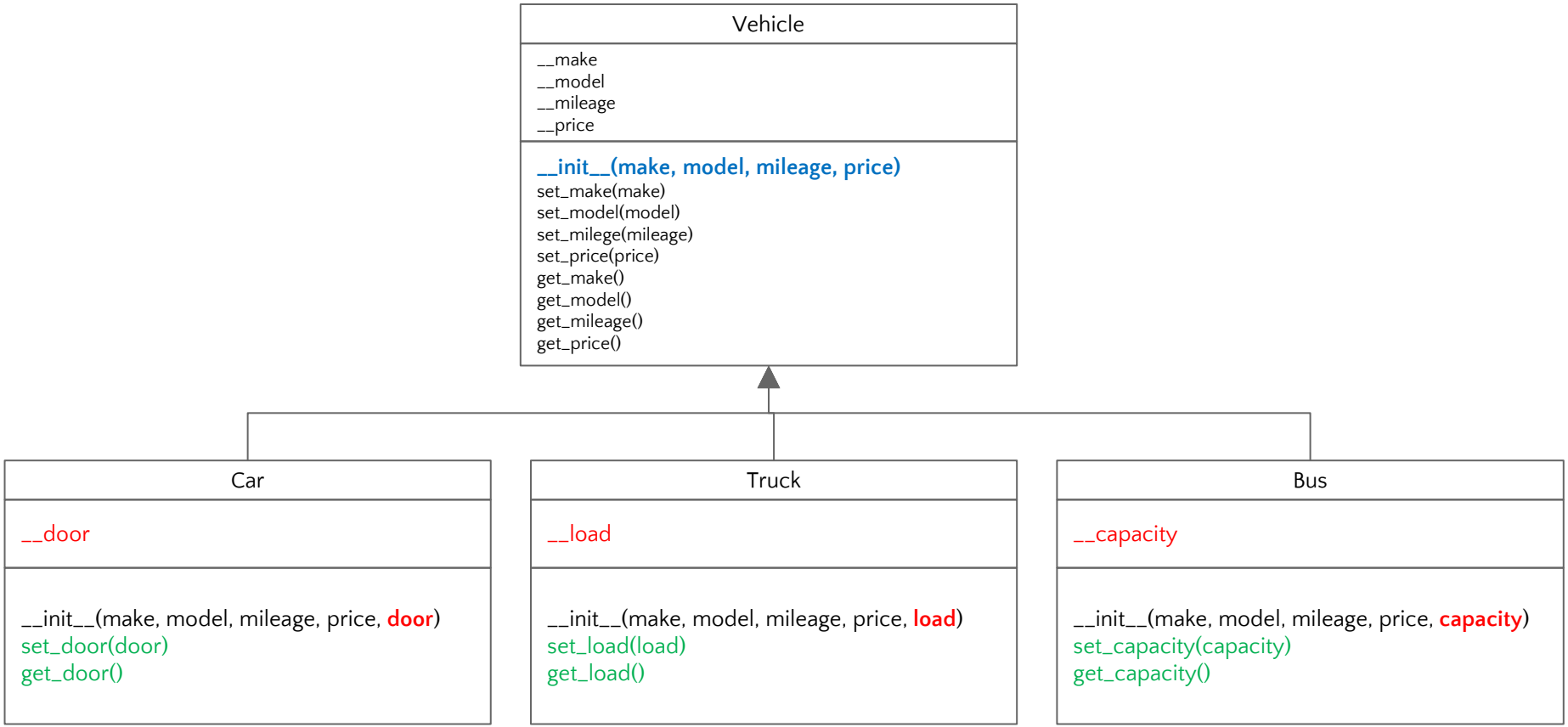
Number of doors

A truck is a vehicle
A bus is a vehicle
A car is a vehicle

Vehicle class

Vehicle
__make __model __mileage __price
__init__(make, model, mileage, price) set_make(make) set_model(model) set_milege(mileage) set_price(price) get_make() get_model() get_milege() get_price()

UML diagram for Inheritance



Class definition of base class, Vehicle

```
class Vehicle:
    def __init__(self, make, model, mileage, price):
        self.__make = make
        self.__model = model
        self.__mileage = mileage
        self.__price = price
# mutators and accessors
```

Class definition of derived class, Car

```
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, door):
        # Call the superclass's __init__ method, and pass the required
        # arguments, note that you need to pass self as an argument
        Vehicle.__init__(self, make, model, mileage, price)
        # initialize the __door attribute
        self.__door = door
# mutators and accessors
```

Points to note for subclass

Class declaration

class Car(Vehicle)

Car inherits from the Vehicle class

Car class is the subclass, and the Vehicle class is the superclass

Initializer __init__

__init__(self, make, model, mileage, price, door)

Must call the Vehicle.__init__ initializer to pass in the 4 data attributes of Vehicle class.

Class definition of derived class, Car

```
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, door):
        # Call the superclass's __init__ method, and pass the required
        # arguments, note that you need to pass self as an argument
        Vehicle.__init__(self, make, model, mileage, price)
        # initialize the __door attribute
        self.__door = door
```

How it works

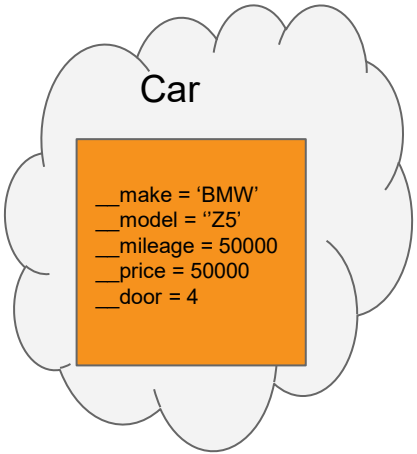
Class definition of derived class, Car

```
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, door):
        Vehicle.__init__(self, make, model, mileage, price)
        # initialize the __door attribute
        self.__door = door

bmw = Car('BMW', 'Z5', 50000, 50000, 4)
```

An object is created in memory from the Car class

The __init__ initializer is called, and the 4 base class attributes are set. The __door attribute is initialized to the passed in argument value

A Car object will exist with its 5 attributes set to some value



Calling method in base class

Vehicle.__init__()
Call the method in base class using class name
Must include the self parameter

super().__init__()
Call the method in base class using super() builtin function
Do not include the self parameter

To call the superclass method use:

Vehicle.__init__()

Class definition of base class, Vehicle

```
class Vehicle:
    def __init__(self, make, model, mileage, price):
        self.__make = make
        self.__model = model
        self.__mileage = mileage
        self.__price = price
# mutators and accessors
```

Class definition of derived class, Car

```
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, door):
        # Call the superclass's __init__ method, and pass the required
        # arguments, note that you need to pass self as an argument
        Vehicle.__init__(self, make, model, mileage, price)
        # initialize the __door attribute
        self.__door = door
# mutators and accessors
```

To call the superclass method use:

super().__init__()

Class definition of base class, Vehicle

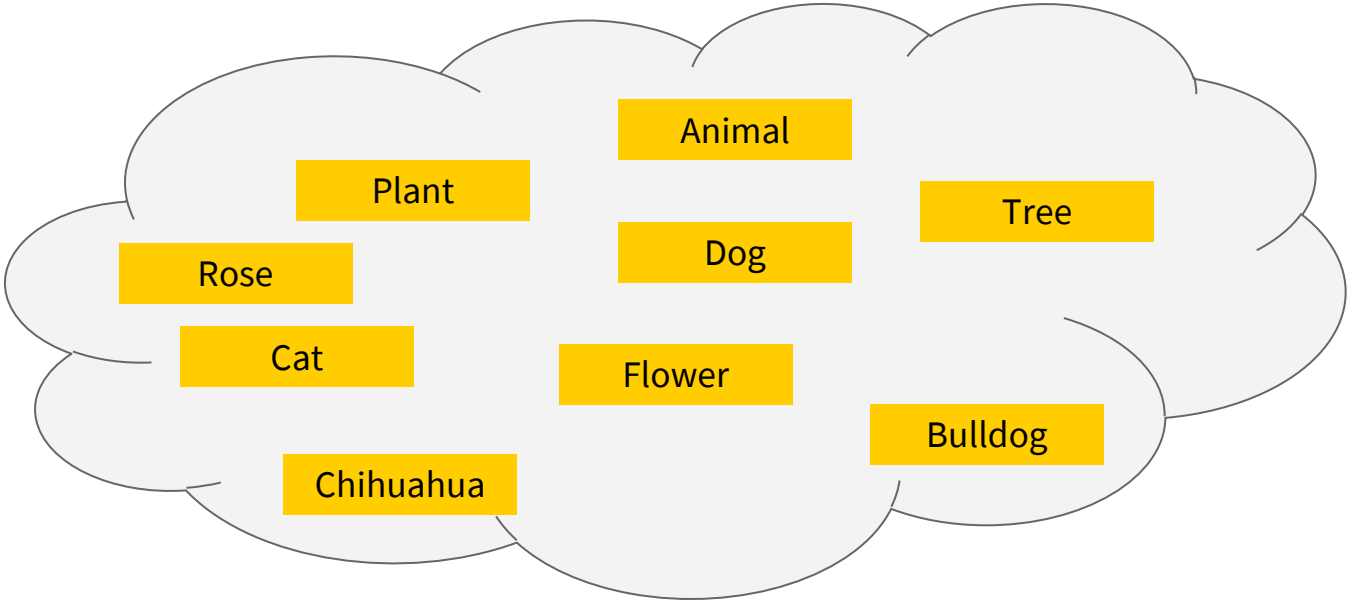
```
class Vehicle:
    def __init__(self, make, model, mileage, price):
        self.__make = make
        self.__model = model
        self.__mileage = mileage
        self.__price = price
# mutators and accessors
```

Class definition of derived class, Car

```
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, door):
        # Call the superclass's __init__ method, and pass the required
        # arguments, note that you need to pass self as an argument
        super().__init__(make, model, mileage, price)
        # initialize the __door attribute
        self.__door = door
# mutators and accessors
```

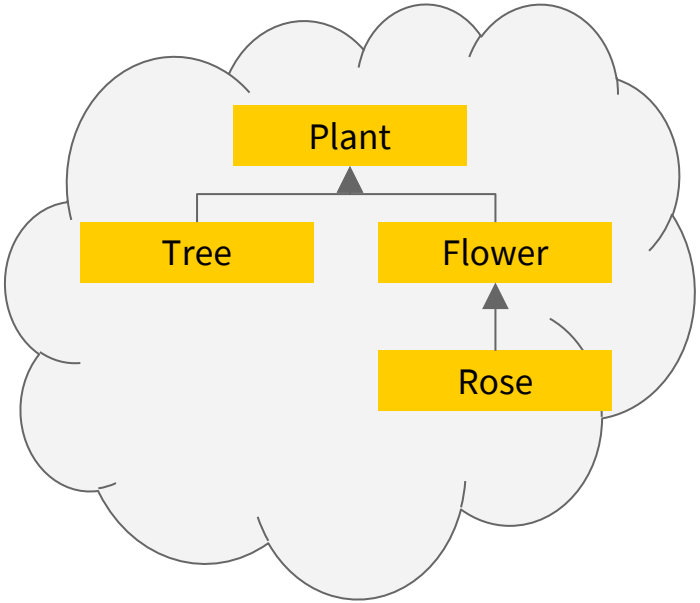
Check point #1

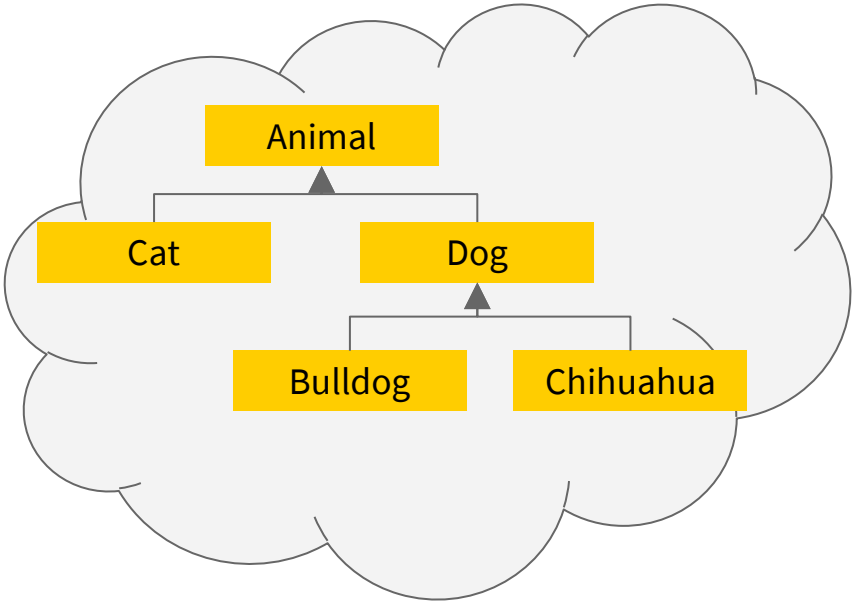
Identify the possible base classes and their derived classes



Check point #1

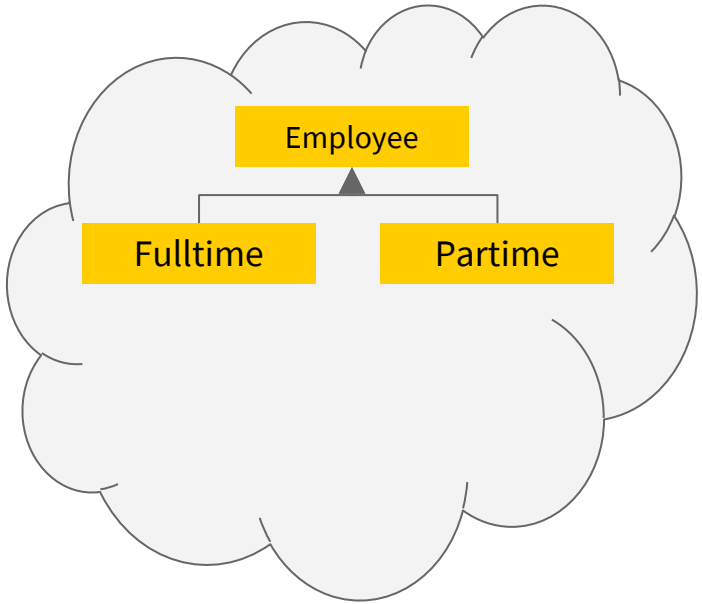
From general to specific in the inheritance hierarchy

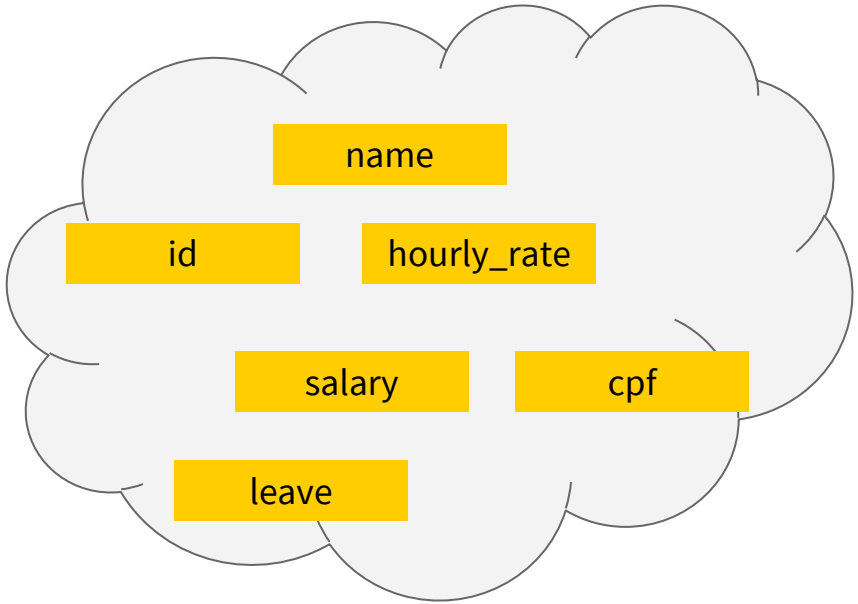




Check point #2

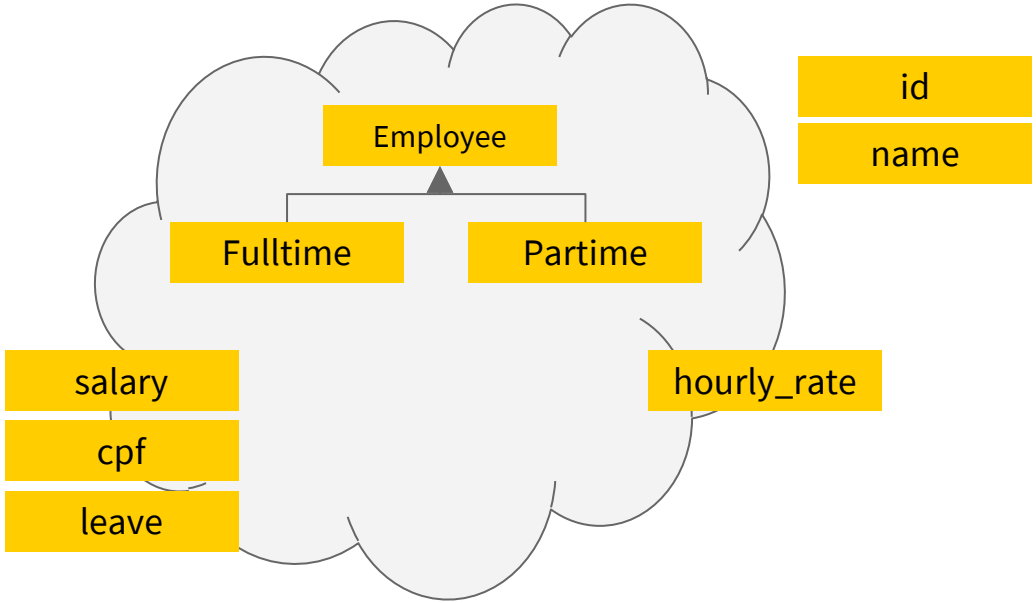
Match the attributes





Check point #2

Try to fit as many attributes to the base class

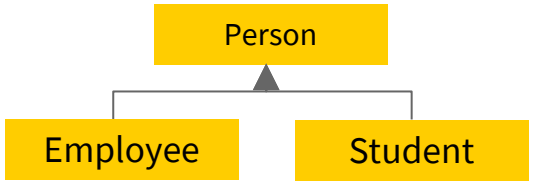


Check point #3

Given the sample output. Identify the superclasses, subclasses and respective data attributes and methods

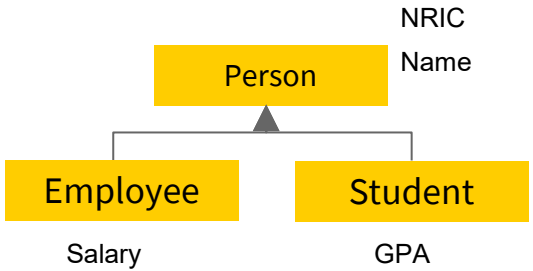
```
Enter NRIC: S1234567A
Enter Name: Daniel Khoo
Student or Employee? (S or E): E
Enter Salary: 2500.50
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S9876543W
Enter Name: Jeremy Lim
Student or Employee? (S or E): s
Enter GPA:3.78
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S1256783H
Enter Name: May Tan Ai Ling
Student or Employee? (S or E): e
Enter Salary: 4300
Do you want to continue to enter another Student or Employee? (Y or N): N
======== Student ==========
Name: Jeremy Lim NRIC: S9876543W
======== Employee ==========
Name: Daniel Khoo NRIC: S1234567A
Name: May Tan Ai Ling NRIC: S1256783H
```

Check point #3




```
Enter NRIC: S1234567A
Enter Name: Daniel Khoo
Student or Employee? (S or E): E
Enter Salary: 2500.50
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S9876543W
Enter Name: Jeremy Lim
Student or Employee? (S or E): s
Enter GPA:3.78
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S1256783H
Enter Name: May Tan Ai Ling
Student or Employee? (S or E): e
Enter Salary: 4300
Do you want to continue to enter another Student or Employee? (Y or N): N
======== Student ==========
Name: Jeremy Lim NRIC: S9876543W
======== Employee ==========
Name: Daniel Khoo NRIC: S1234567A
Name: May Tan Ai Ling NRIC: S1256783H
```

Check point #3




```
Enter NRIC: S1234567A
Enter Name: Daniel Khoo
Student or Employee? (S or E): E
Enter Salary: 2500.50
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S9876543W
Enter Name: Jeremy Lim
Student or Employee? (S or E): s
Enter GPA:3.78
Do you want to continue to enter another Student or Employee? (Y or N): y
Enter NRIC: S1256783H
Enter Name: May Tan Ai Ling
Student or Employee? (S or E): e
Enter Salary: 4300
Do you want to continue to enter another Student or Employee? (Y or N): N
======== Student ==========
Name: Jeremy Lim NRIC: S9876543W
======== Employee ==========
Name: Daniel Khoo NRIC: S1234567A
Name: May Tan Ai Ling NRIC: S1256783H
```

Check point #3

Superclass/Base Class

```
class Person:

    def __init__(self, nric, name):
        self.__nric = nric
        self.__name = name

    def set_name(self, name):
        self.__name = name
    def get_nric(self):
        return self.__nric
    def get_name(self):
        return self.__name
    def __str__(self):
        s = 'Name: {} NRIC: {} '.format(self.__name, self.__nric)
        return s
```

Check point #3

Subclasses / Derived class

```
import Person

class Student(Person.Person):
    def __init__(self, nric, name, gpa):
        Person.Person.__init__(self, nric, name)
        self.__gpa = gpa

    def set_gpa(self, gpa):
        self.__gpa = gpa

    def get_gpa(self):
        return self.__gpa
```

```
import Person

class Employee(Person.Person):
    def __init__(self, nric, name, salary):
        Person.Person.__init__(self, nric, name)
        self.__salary = salary
    def set_salary(self, salary):
        self.__salary = salary
    def get_salary(self):
        return self.__salary
```

Check point #3 Test Program

```
import Person as p
import Employee as e
import Student as s

status = 'Y'
studList = []
empList = []

while status.upper() == 'Y':
    nric = input('Enter NRIC:')
    name = input('Enter Name:')
    category = input('Student or Employee? (S or E):')
    if category.upper() == 'S':
        gpa = float(input('Enter GPA:'))
        stud = s.Student(nric, name, gpa)
        studList.append(stud)
    else:
        salary = float(input('Enter Salary:'))
        emp = e.Employee(nric, name, salary)
        empList.append(emp)
    status = input('Do you want to continue to enter another Student or Employee (Y or N)')

print("==========Student==========")
for i in studList:
    print(i)
print("==========Employee==========")
for i in empList:
    print(i)
```

Check point #4

How to display GPA or Salary?

Use the accessor method to print GPA or Salary

```
Enter NRIC:S1234567A
Enter Name:Daniel Khoo
Student or Employee? (S or E):E
Enter Salary:2500.50
Do you want to continue to enter another Student or Employee (Y or N)y
Enter NRIC:S9876543W
Enter Name:Jeremy Lim
Student or Employee? (S or E):s
Enter GPA:3.78
Do you want to continue to enter another Student or Employee (Y or N)y
Enter NRIC:S1256783H
Enter Name:May Tan Ai Ling
Student or Employee? (S or E):e
Enter Salary:4300
Do you want to continue to enter another Student or Employee (Y or N)n
==========Student=============
Name: Jeremy Lim NRIC: S9876543W
GPA  3.78
==========Employee=============
Name: Daniel Khoo NRIC: S1234567A
Salary  2500.5
Name: May Tan Ai Ling NRIC: S1256783H
Salary  4300.0
```

Check point #1

Test Program

Use the accessor method to print GPA or Salary

```
import Person as p
import Employee as e
import Student as s

status = 'Y'
studList = []
empList = []

while status.upper() == 'Y':
    nric = input('Enter NRIC:')
    name = input('Enter Name:')
    category = input('Student or Employee? (S or E):')
    if category.upper() == 'S':
        gpa = float(input('Enter GPA:'))
        stud = s.Student(nric, name, gpa)
        studList.append(stud)
    else:
        salary = float(input('Enter Salary:'))
        emp = e.Employee(nric, name, salary)
        empList.append(emp)
    status = input('Do you want to continue to enter another Student or Employee (Y or N)')

print("==========Student==========")
for i in studList:
    print(i)
    print("GPA ", i.get_gpa())
print("==========Employee==========")
for i in empList:
    print(i)
    print("Salary ", i.get_salary())
```

Activity

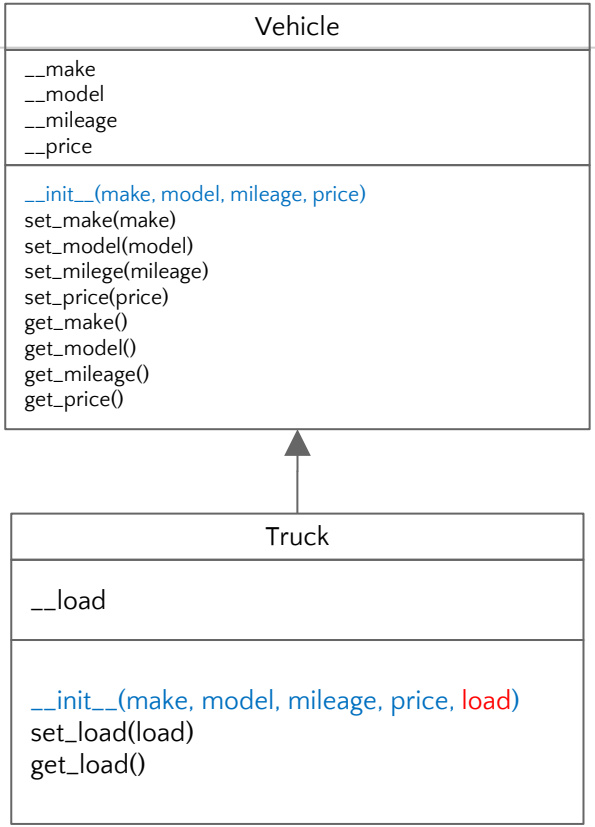
Practical Question 1

Polymorphism

Method Overriding

Method Overriding

__init__() method was defined in both classes. When this happens, the method in the subclass overrides that in the superclass. When overriding a superclass method, we tend to extend the definition rather than replace it.



Check point #1 Use Method Overriding

```
Enter NRIC:S1234567A
Enter Name:Daniel Khoo
Student or Employee? (S or E):E
Enter Salary:2500.50
Do you want to continue to enter another Student or Employee (Y or N)y
Enter NRIC:S9876543W
Enter Name:Jeremy Lim
Student or Employee? (S or E):s
Enter GPA:3.78
Do you want to continue to enter another Student or Employee (Y or N)y
Enter NRIC:S1256783H
Enter Name:May Tan Ai Ling
Student or Employee? (S or E):e
Enter Salary:4300
Do you want to continue to enter another Student or Employee (Y or N)n
==========Student=============
Name: Jeremy Lim NRIC: S9876543W GPA : 3.78
==========Employee=============
Name: Daniel Khoo NRIC: S1234567A Salary(s$): 2500.5
Name: May Tan Ai Ling NRIC: S1256783H Salary(s$): 4300.0
```

Check point #1

Use Method Overriding

```
class Person:
def __init__(self, nric, name):
    self.__name = name
    self.__nric = nric
def __str__(self):
    s = 'Name: {} NRIC: {}'.format(self.__name, self.__nric)
    return s
```

Calling methods in base class

```
import Person

class Student(Person.Person):
    def __init__(self, nric, name, gpa):
        Person.Person.__init__(self, nric, name)
        self.__gpa = gpa
    def __str__(self):
        s = super().__str__()
        s = s + ' GPA: {}'.format(self.__gpa)
        return s
```

```
import Person

class Employee(Person.Person):
    def __init__(self, nric, name, salary):
        Person.Person.__init__(self, nric, name)
        self.__salary = salary
    def __str__(self):
        s = super().__str__()
        s = s + ' Salary(s$): {}'.format(self.__salary)
        return s
```

What is the output?

Vehicle is moving
Car is moving

Class definition of base class, Vehicle and Car

```
class Vehicle:
    def __init__(self, model):
        self.__model = model
# mutators and accessors
    def move(self):
        print('Vehicle is moving')

class Car(Vehicle):
    def __init__(self, model, door):
        super().__init__(model)
        self.__door = door
# mutators and accessors
    def move(self):
        super().move()
        print('Car is moving')
```

Calling Overloading methods

```
car = Car('BMW', 4)
car.move()
```

What is the output?

Runtime Error! Need to set model attribute in base class

Class definition of base class, Vehicle and Car

```
class Vehicle:
    def __init__(self, model):
        self.__model = model
    def get_model(self):
        return self.__model

class Car(Vehicle):
    def __init__(self, model, door):
        self.__door = door
    def get_door(self):
        return self.__door

car = Car('BMW', 4)
print(car.get_model())
```

super().__init__(model)

or

Vehicle.__init__(self, model)

What is the output?

Runtime Error!
Model attribute is private, cannot be accessed directly!

Class definition of base class, Vehicle and Car

```
class Vehicle:
    def __init__(self, model):
        self.__model =model
    def get_model(self):
        return self.__model

class Car(Vehicle):
    def __init__(self, model, door):
        super().__model = model
        self.__door = door
    def get_door(self):
        return self.__door

car = Car('BMW', 4)
print(car.get_model())
```

What is the output?

BMW

Class definition of base class, Vehicle and Car

```
class Vehicle:
    def __init__(self, model):
        self.__model =model
    def get_model(self):
        return self.__model
    def set_model(self, model):
        self.__model = model

class Car(Vehicle):
    def __init__(self, model, door):
        super().set_model(model)
        self.__door = door
    def get_door(self):
        return self.__door

car = Car('BMW', 4)
print(car.get_model())
```

What is the output?

BMW
Note the self parameter if accessed base method using class name

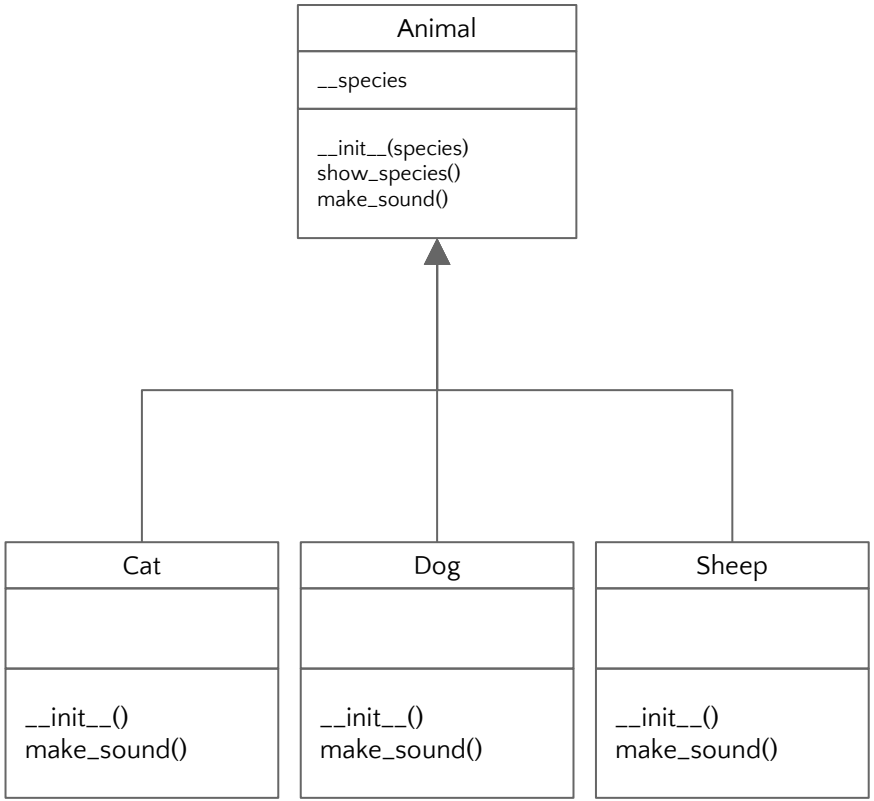
Class definition of base class, Vehicle and Car

```
class Vehicle:
    def __init__(self, model):
        self.__model =model
    def get_model(self):
        return self.__model
    def set_model(self, model):
        self.__model = model

class Car(Vehicle):
    def __init__(self, model, door):
        Vehicle.set_model(self, model)
        self.__door = door
    def get_door(self):
        return self.__door

car = Car('BMW', 4)
print(car.get_model())
```

Polymorphism

Allows subclass to have methods with the same names as methods in their superclasses.

Gives the ability of a program to call the correct method depending on the type of object that is used to call it.



Class definition of base class, Animal

```
class Animal:
    def __init__(self, species):
        self.__species = species
    def show_species(self):
        print('I am a', self.__species)
    def make_sound(self):
        print('Grrrr')
```

Class definition of derived class, Cat

```
class Cat(Animal):
    def __init__(self):
        super().__init__('Cat')
    def make_sound(self):
        print('Meow!')
```

What is the output?

```
I am a Unknown
Grrrr
I am a Cat
Meow!
I am a Dog
Woof!
```

Function that call the overloaded methods

```
def show_animal_info(creature):
    creature.show_species()
    creature.make_sound()

animal = Animal('Unknown')
cat = Cat()
dog = Dog()

show_animal_info(animal)
show_animal_info(cat)
show_animal_info(dog)
```

What is the output?

Runtime Error!
Because string does not have the show_species and make_sound method!

Function that call the overloaded methods

```
def show_animal_info(creature):
    creature.show_species()
    creature.make_sound()

show_animal_info('I am a string!')
```

Use the isinstance method to determine if an object is an instance of a class

This is not an animal!

Function that call the overloaded methods

```
def show_animal_info(creature):
    if isinstance(creature, Animal):
        creature.show_species()
        creature.make_sound()
    else:
        print('This is not an animal!')

show_animal_info('I am a string!')
```

Check Point #2

```
class Account:
    def __init__(self, type):
        self.__type = type
    def get_type(self):
        return self.__type

class SavingAccount(Account):
    def __init__(self):
        super().__init__('Saving')
        self.__interest_rate = 0.05
    def show_interest_rate(self):
        print(super().get_type(), 'account interest rate :', self.__interest_rate)

class CurrentAccount(Account):
    def __init__(self):
        super().__init__('Current')
        self._interest_rate = 0.01
    def show_interest_rate(self):
        print(super().get_type(), 'account interest rate :', self.__interest_rate)
```

What is/are the name of the superclass?

Account

What is/are the names of the subclass?

SavingAccount, CurrentAccoount

What is/are the overriden method?

__init__()

Check Point #2

```
class Account:
    def __init__(self, type):
        self.__type = type
    def get_type(self):
        return self.__type

class SavingAccount(Account):
    def __init__(self):
        super().__init__('Saving')
        self.__interest_rate = 0.05
    def show_interest_rate(self):
        print(super().get_type(), 'account interest rate :', self.__interest_rate)

class CurrentAccount(Account):
    def __init__(self):
        super().__init__('Current')
        self._interest_rate = 0.01
    def show_interest_rate(self):
        print(super().get_type(), 'account interest rate :', self.__interest_rate)

c = SavingAccount()
c.show_interest_rate()
```

What is the output?

Saving account interest rate : 0.05

Check Point #3

```
class Food:
    def __init__(self, name, food_type='meat'):
        self.__food_type = food_type
        self.__name = name
    def message(self):
        print(self.__name, 'is a', self.__food_type)

class Beef(Food):
    def __init__(self):
        super().__init__('beef')

class Grape(Food):
    def __init__(self):
        super().__init__('grape', 'fruit')

f = Food('Mutton')
f.message()
g = Grape()
g.message()
b = Beef()
b.message()
```

Are you able to explain the differences?

How does it impact on object creation?

By default, the food_type is "meat".

What is the output?

Mutton is a meat
grape is a fruit
beef is a meat

Activity

Practical Question 3

True or False

A derived class can access only some of the data attributes and methods of its superclass.

It is possible to call a superclass's __init__ method from a subclass's __init__ method.

Superclass can also access the data attributes of its subclass

Only the __init__ method can be overridden.

You can use the isinstance function to determine whether an object is an instance of a class.

Summary

- Explain the concept of inheritance
- Apply superclasses and subclasses
- Apply inheritance in a program
- Explain the concept of polymorphism