# 7.类-面向对象的编程

## 7.1.类的定义

### 7.1.1.类的概述

面向对象更符合人类对客观世界的抽象和理解

一切皆对象  
 一只小狗，一把椅子，一张信用卡，一条巧克力。。。

一切对象，都有自己内在的属性  
 狗狗的品种、椅子的质地、信用卡的额度、巧克力的口味。。。

一切行为，皆是对象的行为  
 狗狗蹲下、椅子移动位置、刷信用卡、巧克力融化了。。。

类是对象的载体

不同年龄、肤色、品质的猫，每一只都是一个对象

他们有一个共同的特征：都是猫

我们可以把一类对象的公共特征抽象出来，创建通用的类

*# 创建类***class** **Cat**():

*"""模拟猫"""*

**def** \_\_init\_\_(self, name):

*"""初始化属性"""*

self.name = name

**def** jump(self):

*"""模拟猫跳跃"""*

print(self.name + " is jumping")

*# 用类创建实例*

my\_cat = Cat("Loser")

your\_cat = Cat("Lucky")

*# 调用属性*

print(my\_cat.name)

print(your\_cat.name)

Loser

Lucky

*# 调用方法*

my\_cat.jump()

your\_cat.jump()

Loser is jumping

Lucky is jumping

### 7.1.2.类的定义

三要素：类名、属性、方法

**类的命名**

要有实际意义

驼峰命名法——组成的单词首字母大写  
 Dog、 CreditCard、 ElectricCar

*# class 类名：*

*"""类前空两行"""*

**class** **Car**():

*"""对该类的简单介绍"""*

**pass**

*"""类后空两行"""*

**类的属性**

*# def \_\_init\_\_(self,要传递的参数) 初始化类的属性*

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**类的方法**

*# 相对于类内部定义的函数*

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**def** get\_main\_information(self): *# self不能省*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}**".format(self.brand, self.model, self.year))

**def** get\_mileage(self):

*"""获取总里程"""*

**return** "行车总里程：**{}**公里".format(self.mileage)

## 7.2.创建实例

### 7.2.1.实例的创建与访问

**实例的创建**

将实例赋值给对象，实例化过程中，传入相应的参数  
 v = 类名（必要的初始化参数）

my\_new\_car = Car("Audi", "A6", 2018)

**访问属性**

print(my\_new\_car.brand)

print(my\_new\_car.model)

print(my\_new\_car.year)

Audi

A6

2018

**调用方法**

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**def** get\_main\_information(self): *# self不能省*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}**".format(self.brand, self.model, self.year))

**def** get\_mileage(self):

*"""获取总里程数"""*

**return** "行车总里程：**{}**公里".format(self.mileage)

实例名.方法名(必要的参数)

my\_new\_car = Car("Audi", "A6", 2018)

my\_new\_car.get\_main\_information()

品牌：Audi 型号：A6 出厂年份：2018

mileage = my\_new\_car.get\_mileage()

print(mileage)

行车总里程：0公里

### 7.2.2.修改属性

**1、直接修改**

先访问，后修改

my\_old\_car = Car("BYD", "宋", 2016)

print(my\_old\_car.mileage)

my\_old\_car.mileage = 12000

print(my\_old\_car.mileage)

0

12000

print(my\_old\_car.get\_mileage())

行车总里程：12000公里

**2、通过方法修改属性**

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**def** get\_main\_information(self): *# self不能省*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}**".format(self.brand, self.model, self.year))

**def** get\_mileage(self):

*"""获取总里程数"""*

**return** "行车总里程：**{}**公里".format(self.mileage)

**def** set\_mileage(self, distance):

*"""设置总里程数"""*

self.mileage = distance

my\_old\_car = Car("BYD", "宋", 2016)

print(my\_old\_car.get\_mileage())

my\_old\_car.set\_mileage(8000)

print(my\_old\_car.get\_mileage())

行车总里程：0公里

行车总里程：8000公里

**3、继续拓展**

禁止设置负里程

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**def** get\_main\_information(self): *# self不能省*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}**".format(self.brand, self.model, self.year))

**def** get\_mileage(self):

*"""获取总里程数"""*

print("行车总里程：**{}**公里".format(self.mileage))

**def** set\_mileage(self, distance):

*"""设置总里程数"""*

**if** distance >= 0:

self.mileage = distance

**else**:

print("里程数不能为负！")

**def** increment\_mileage(self, distance):

*"""总里程数累计"""*

**if** distance >= 0:

self.mileage += distance

**else**:

print("新增里程数不能为负！")

my\_old\_car = Car("BYD", "宋", 2016)

my\_old\_car.get\_mileage()

my\_old\_car.set\_mileage(-8000)

my\_old\_car.get\_mileage()

行车总里程：0公里

里程数不能为负！

行车总里程：0公里

*# 将每次的里程数累加*

my\_old\_car.get\_mileage()

my\_old\_car.set\_mileage(8000)

my\_old\_car.get\_mileage()

my\_old\_car.increment\_mileage(500)

my\_old\_car.get\_mileage()

行车总里程：0公里

行车总里程：8000公里

行车总里程：8500公里

**总结**

类包含的信息量可以是极大的，可以创建无穷多的实例

高度的拟人（物）化，符合人类对客观世界的抽象和理解

## 7.3.类的继承

### 7.3.1.继承概述

看一下人在生物界的分支链

生物—动物界—脊索动物门—哺乳动物纲—灵长目—人科—人属—智人种

公共特征逐渐增加的过程

【问题】

假设二元系统： 人属 = {A人种， B人种， C人种。。。。}  
 为每一个人种构造一个类

方案一： 各自独立，分别构造各自人种的类

方案二：  
 1、将各人种公共特征提取出来，建立人属的类；  
 2、各人种继承上一级（人属）的公共特征，然后添加自身特殊特征，构建各自人种的类。

通常，我们选择方案二，因为他避免了过多的重复劳动

### 7.3.2.简单的继承

**父类**

**class** **Car**():

*"""模拟汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化汽车属性"""* *# 相当于类内部的变量*

self.brand = brand *# 汽车的品牌*

self.model = model *# 汽车的型号*

self.year = year *# 汽车出厂年份*

self.mileage = 0 *# 新车总里程初始化为0*

**def** get\_main\_information(self): *# self不能省*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}**".format(self.brand, self.model, self.year))

**def** get\_mileage(self):

*"""获取总里程数"""*

print("行车总里程：**{}**公里".format(self.mileage))

**def** set\_mileage(self, distance):

*"""设置总里程数"""*

**if** distance >= 0:

self.mileage = distance

**else**:

print("里程数不能为负！")

**def** increment\_mileage(self, distance):

*"""总里程数累计"""*

**if** distance >= 0:

self.mileage += distance

**else**:

print("新增里程数不能为负！")

**子类**

格式：class 子类名（父类名）：

新建一个电动汽车的类

**class** **ElectricCar**(Car):

*"""模拟电动汽车"""*

**def** \_\_init\_\_(self, brand, model, year):

*"""初始化电动汽车属性"""*

super().\_\_init\_\_(brand, model, year) *# 声明继承父类的属性*

自动继承父类的所有方法

my\_electric\_car = ElectricCar("NextWeek", "FF91", 2046)

my\_electric\_car.get\_main\_information()

品牌：NextWeek 型号：FF91 出厂年份：2046

### 7.3.3.给子类添加属性和方法

**class** **ElectricCar**(Car):

*"""模拟电动汽车"""*

**def** \_\_init\_\_(self, brand, model, year, bettery\_size):

*"""初始化电动汽车属性"""*

super().\_\_init\_\_(brand, model, year) *# 声明继承父类的属性*

self.bettery\_size = bettery\_size *# 电池容量*

self.electric\_quantity = bettery\_size *# 电池剩余电量*

self.electric2distance\_ratio = 5 *# 电量距离换算系数 5公里/kW.h*

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio *# 剩余可行驶里程*

**def** get\_electric\_quantit(self):

*"""查看当前电池电量"""*

print("当前电池剩余电量：**{}** kW.h".format(self.electric\_quantity))

**def** set\_electric\_quantity(self, electric\_quantity):

*"""设置电池剩余电量，重新计算电量可支撑行驶里程"""*

**if** electric\_quantity >= 0 **and** electric\_quantity <= self.bettery\_size:

self.electric\_quantity = electric\_quantity

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio

**else**:

print("电量未设置在合理范围！")

**def** get\_remainder\_range(self):

*"""查看剩余可行驶里程"""*

print("当前电量还可以继续驾驶 **{}** 公里".format(self.remainder\_range))

my\_electric\_car = ElectricCar("NextWeek", "FF91", 2046, 70)

my\_electric\_car.get\_electric\_quantit() *# 获取当前电池电量*

my\_electric\_car.get\_remainder\_range() *# 获取当前剩余可行驶里程*

当前电池剩余电量：70 kW.h

当前电量还可以继续驾驶 350 公里

my\_electric\_car.set\_electric\_quantity(50) *# 重设电池电量*

my\_electric\_car.get\_electric\_quantit() *# 获取当前电池电量*

my\_electric\_car.get\_remainder\_range() *# 获取当前剩余可行驶里程*

当前电池剩余电量：50 kW.h

当前电量还可以继续驾驶 250 公里

### 7.3.4.重写父类的方法——多态

**class** **ElectricCar**(Car):

*"""模拟电动汽车"""*

**def** \_\_init\_\_(self, brand, model, year, bettery\_size):

*"""初始化电动汽车属性"""*

super().\_\_init\_\_(brand, model, year) *# 声明继承父类的属性*

self.bettery\_size = bettery\_size *# 电池容量*

self.electric\_quantity = bettery\_size *# 电池剩余电量*

self.electric2distance\_ratio = 5 *# 电量距离换算系数 5公里/kW.h*

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio *# 剩余可行驶里程*

**def** get\_main\_information(self): *# 重写父类方法*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}** 续航里程：**{}** 公里"

.format(self.brand, self.model, self.year, self.bettery\_size\*self.electric2distance\_ratio))

**def** get\_electric\_quantit(self):

*"""查看当前电池电量，重新计算电量可支撑行驶里程"""*

print("当前电池剩余电量：**{}** kW.h".format(self.electric\_quantity))

**def** set\_electric\_quantity(self, electric\_quantity):

*"""设置电池剩余电量"""*

**if** electric\_quantity >= 0 **and** electric\_quantity <= self.bettery\_size:

self.electric\_quantity = electric\_quantity

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio

**else**:

print("电量未设置在合理范围！")

**def** get\_remainder\_range(self):

*"""查看剩余可行驶里程"""*

print("当前电量还可以继续驾驶 **{}** 公里".format(self.remainder\_range))

my\_electric\_car = ElectricCar("NextWeek", "FF91", 2046, 70)

my\_electric\_car.get\_main\_information()

品牌：NextWeek 型号：FF91 出厂年份：2046 续航里程：350 公里

### 7.3.5.类中的引用对象

把电池抽象成一个对象  
逻辑更加清晰

**class** **Bettery**():

*"""模拟电动汽车的电池"""*

**def** \_\_init\_\_(self, bettery\_size = 70):

self.bettery\_size = bettery\_size *# 电池容量*

self.electric\_quantity = bettery\_size *# 电池剩余电量*

self.electric2distance\_ratio = 5 *# 电量距离换算系数 5公里/kW.h*

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio *# 剩余可行驶里程*

**def** get\_electric\_quantit(self):

*"""查看当前电池电量"""*

print("当前电池剩余电量：**{}** kW.h".format(self.electric\_quantity))

**def** set\_electric\_quantity(self, electric\_quantity):

*"""设置电池剩余电量，计重新算电量可支撑行驶里程"""*

**if** electric\_quantity >= 0 **and** electric\_quantity <= self.bettery\_size:

self.electric\_quantity = electric\_quantity

self.remainder\_range = self.electric\_quantity\*self.electric2distance\_ratio

**else**:

print("电量未设置在合理范围！")

**def** get\_remainder\_range(self):

*"""查看剩余可行驶里程"""*

print("当前电量还可以继续驾驶 **{}** 公里".format(self.remainder\_range))

**class** **ElectricCar**(Car):

*"""模拟电动汽车"""*

**def** \_\_init\_\_(self, brand, model, year, bettery\_size):

*"""初始化电动汽车属性"""*

super().\_\_init\_\_(brand, model, year) *# 声明继承父类的属性*

self.bettery = Bettery(bettery\_size) *# 电池*

**def** get\_main\_information(self): *# 重写父类方法*

*"""获取汽车主要信息"""*

print("品牌：**{}** 型号：**{}** 出厂年份：**{}** 续航里程：**{}** 公里"

.format(self.brand, self.model, self.year,

self.bettery.bettery\_size\*self.bettery.electric2distance\_ratio))

my\_electric\_car = ElectricCar("NextWeek", "FF91", 2046, 70)

my\_electric\_car.get\_main\_information() *# 获取车辆主要信息*

品牌：NextWeek 型号：FF91 出厂年份：2046 续航里程：350 公里

my\_electric\_car.bettery.get\_electric\_quantit() *# 获取当前电池电量*

当前电池剩余电量：70 kW.h

my\_electric\_car.bettery.set\_electric\_quantity(50) *# 重设电池电量*

my\_electric\_car.bettery.get\_electric\_quantit() *# 获取当前电池电量*

当前电池剩余电量：50 kW.h

my\_electric\_car.bettery.get\_remainder\_range() *# 获取当前剩余可行驶里程*

当前电量还可以继续驾驶 250 公里

## 7.4.作业练习

类的定义与调用：

1、构造一个银联信用卡的类，

\*包含如下属性：

顾客姓名

信用卡授信额度

当前额度

单次刷卡金额上限

\*包含如下方法：

分别获得上述属性的方法；

对授信额度进行修改的方法；

对单次刷卡金额上限修改的方法；

实现刷卡方法，传入一个刷卡金额，先判断是否超过单次刷卡金额上限以及当前额度是否够用，如合理，则执行刷卡，将当前额度减去刷卡金额。

\*创建信用卡实例并执行调用相应的属性和方法

类的继承：

2、通过继承银联信用卡的类，构造中国银行信用卡的类：

实现对银联信用卡类的继承；

新增加属性：中国银行信用卡积分、优惠店铺列表；

重写刷卡方法：

传入消费店铺名称和消费金额，如果店铺名称在优惠店铺列表中，则刷卡金额打95折；

每消费10元，信用卡积分增加1分；

保留父类刷卡方法的其他功能。

新增如下方法：

获得用户积分的方法；

设置优惠店铺列表的方法。

答案：

1.

**class** **UnionPayCreditCard**():

*'''构造一个银联信用卡的类'''*

**def** \_\_init\_\_(self, name, amount, current\_amount, single\_limit):

*''' 顾客姓名 信用卡授信额度 当前额度 单次刷卡金额上限 '''*

self.name = name

self.amount = amount

self.current\_amount = current\_amount

self.single\_limit = single\_limit

*'''对于每个属性分别设置getter'''*

**def** get\_name(self):

**return** self.name

**def** get\_amount(self):

**return** self.amount

**def** get\_current\_amount(self):

**return** self.current\_amount

**def** get\_single\_limit(self):

**return** self.single\_limit

**def** modify\_amount(self, new\_amount):

*'''对授信额度进行修改的方法'''*

self.amount = new\_amount

print("你的消费额度修改为**{}**元".format(self.amount))

**def** modify\_single\_amount(self, new\_single\_amount):

*'''对单次刷卡金额上限修改的方法'''*

self.single\_limit = new\_single\_amount

print("你的消费上限修改为**{}**元".format(self.single\_limit))

**def** consume(self, cost):

*'''实现刷卡方法，传入一个刷卡金额， 先判断是否超过单次刷卡金额上限以及 当前额度是否够用，如合理，则执行刷 卡，将当前额度减去刷卡金额'''*

**if** cost < self.single\_limit **and** cost < self.amount:

self.amount -= cost

print("你本次消费**{}**元，剩余额度为**{}**元".format(cost, self.amount))

**else**:

print("你的消费金额无法支付，请检查你的信用额度！")

**return** self.amount

**def** start():

card = UnionPayCreditCard("Tony", 10000, 5000, 1000)

name = card.get\_name()

amount = card.get\_amount()

current\_amount = card.get\_current\_amount()

single\_limit = card.get\_single\_limit()

print("name=**{}**, amount=**{}**, current\_amount=**{}**, single\_limit=**{}**".format( name, amount, current\_amount, single\_limit))

card.modify\_amount(20000)

card.modify\_single\_amount(5000)

card.consume(2000)

card.consume(100000)

**if** \_\_name\_\_ == "\_\_main\_\_":

start()

name=Tony, amount=10000, current\_amount=5000, single\_limit=1000

你的消费额度修改为20000元

你的消费上限修改为5000元

你本次消费2000元，剩余额度为18000元

你的消费金额无法支付，请检查你的信用额度！

2.

**class** **BankOfChinaCreditCard**(UnionPayCreditCard):

**def** \_\_init\_\_(self, name, amount, current\_amount, single\_limit):

super().\_\_init\_\_(name, amount, current\_amount, single\_limit)

self.integral = 0

self.discount\_shop = ["Huawei旗舰店", "小米旗舰店", "Apple旗舰店"]

**def** consume(self, shop\_name, cost):

*'''重写刷卡方法'''*

**if** cost < self.single\_limit **and** cost < self.amount **and** shop\_name **in** self.discount\_shop:

cost \*= 0.95

self.integral = cost // 10

print("你本次消费**{}**元, 优惠**{}**元，获得积分**{}**, 已有积分**{}**".format(cost, 0.05 \* cost, cost // 10, self.integral))

**def** get\_integral(self):

*'''获得用户积分的方法'''*

print("你的积分为**{}**".format(self.integral))

**return** self.integral

**def** set\_discount\_shop(self, shop\_name):

*'''设置优惠店铺列表的方法'''*

self.discount\_shop.append(shop\_name)

**return** self.discount\_shop

**def** start():

card = BankOfChinaCreditCard("Tony", 10000, 5000, 1000)

name = card.get\_name()

amount = card.get\_amount()

current\_amount = card.get\_current\_amount()

single\_limit = card.get\_single\_limit()

integral = card.get\_integral()

print("name=**{}**, amount=**{}**, current\_amount=**{}**, single\_limit=**{}**, integral=**{}**".format( name, amount, current\_amount, single\_limit, integral))

card.modify\_amount(20000)

card.modify\_single\_amount(5000)

card.consume( "小米旗舰店", 2000)

card.consume("Apple旗舰店", 100000)

**if** \_\_name\_\_ == "\_\_main\_\_":

start()

你的积分为0

name=Tony, amount=10000, current\_amount=5000, single\_limit=1000, integral=0

你的消费额度修改为20000元

你的消费上限修改为5000元

你本次消费1900.0元, 优惠95.0元，获得积分190.0, 已有积分190.0