CS61A NOTE8 OOP Inheritance

Inheritance

To avoid redefining attributes and methods for similar classes, we can write a single **base class** from which the similar classes **inherit**. For example, we can write a class called **Pet** and define **Dog** as a **subclass** of **Pet**:为了避免为类似的类重新定义属性和方法，我们可以写一个基类，类似的类从基类继承。

纯文本

```
class Pet:#基类

    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!
        self.name = name
        self.owner = owner

    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")

    def talk(self):
        print(self.name)

class Dog(Pet):

    def talk(self):
        super().talk()
        print('This Dog says woof!')
```

Inheritance represents a hierarchical relationship between two or more classes where one class **is a** more specific version of the other: a dog **is a** pet (We use **is a** to describe this sort of relationship in OOP languages, and not to refer to the Python `is` operator).继承表示两个或多个类之间的层次关系，一个类是另一个类更具体的版本：狗是宠物（ OOP 中我们用 is a 描述这种关系，不是指 Python 的 is 操作符）。

Since Dog inherits from Pet, the Dog class will also inherit the Pet class's methods, so we don't have to redefine __inte__ or eat. We do want each Dog to talk in a Dog-specific way, so we can **override** the talk method. 犹豫 Dog 继承于 Pet，Dog 类继承 Pet 的类方法，不必重新定义 **init** 或 eat。希望每只狗都能以狗特有的方式说话，重写说话 method。

We can use super() to refer to the superclass of self, and access any superclass methods as if we were an instance of the superclass. For example, super().talk() in the Dog class will call the talk() method from the Pet class, but passing the Dog instance as the self.使用 super() 来引用 self 的父类，并像访问父类实例一样访问父类 method。例如，狗类中的 super().talk()将调用 pet 类中的 talk()方法，但传递狗的实例作为 self。

Lab8 WWPD

纯文本

```
>>> class A:
...   x, y = 0, 0
...   def __init__(self):
...         return
>>> class B(A):
...   def __init__(self):
...         return
>>> class C(A):
...   def __init__(self):
...         return
>>> print(A.x, B.x, C.x)
0 0 0
>>> B.x = 2
>>> print(A.x, B.x, C.x)
0 2 0
>>> A.x += 1
>>> print(A.x, B.x, C.x)
1 2 1 #A和C都继承A的

>>> obj = C()
>>> obj.y = 1
>>> C.y == obj.y
False #0不等于1
>>> A.y = obj.y
>>> print(A.y, B.y, C.y, obj.y)
1 1 1 1 #去继承A的1
```

Q1: That's inheritance, init?

Let's say we want to create a class `Monarch` that inherits from another class, `Butterfly`. We've partially written an `__init__` method for `Monarch`. For each of the following options, state whether it would correctly complete the method so that every instance of `Monarch` has all of the instance attributes of a `Butterfly` instance. You may assume that a monarch butterfly has the default value of 2 wings.创建 Monarch 继承蝴蝶。每个 Monarch 实例有所有蝴蝶的属性，假设 monarch 有 2 对翅膀

```
class Butterfly():
    def __init__(self, wings=2):
        self.wings = wings

class Monarch(Butterfly):
    def __init__(self):
        ----------------------------------------
        self.colors = ['orange', 'black', 'white']
```

纯文本

```
super.__init__()
```

❌ super必须带parentheses ()

```
super().__init__()
```

✅ 调用格式,记忆：毕竟有super肯定已经有了自己self

```
Butterfly.__init__()
```

❌ 必须明确地把self作为一个参数传入

```
Butterfly.__init__(self)
```

✅ 引用butterfly的init函数，把自己传入，但不用传入wings=2不公有

Some butterflies like the Owl Butterfly have adaptations that allow them to mimic other animals with their wing patterns. Let's write a class for these MimicButterflies. In addition to all of the instance variables of a regular Butterfly instance, these should also have an instance variable mimic_animal describing the name of the animal they mimic. Fill in the blanks in the lines below to create this class.一些蝴蝶，如猫头鹰蝴蝶，有一些适应性，使它们能够用翅膀的图案模仿其他动物。为这些 MimicButterflies 写一个类。除了普通蝴蝶实例的所有变量外，这些蝴蝶还多一个实例变量 mimic_animal 描述它们模仿的动物的名字。

纯文本

```
class MimicButterfly(Butterfly):
    def __init__(self, mimic_animal):
        super().__init__() #子类init跟父类init没有共有的所以()
        self.mimic_animal = mimic_animal
```

Rosemary for Remembrance (FA17 MT2, Q1) Part 1 (solution)

```
class Plant:
    k = 1
    kind = "green"
    def __init__(self):
        self.k = Plant.k
        Plant.k = self.k + 1
        if self.k > 3:
            Plant.name = lambda t:"tree"
            Plant.k = 6
    def name(self):
        return kind
    def __repr__(self):
        s = self.name() + " "
        return s + str(self.k)

class Flower(Plant):
    kind = "pretty"
    def __repr__(self):
        s = self.smell() + " "
        return s + Plant.__repr__(self)
    def smell(self):
        return "bad"
```

```
class Rose(Flower):
    def name(self):
        return "rose"
    def smell(self):
        return "nice"
class Garden:
    def __init__(self,kind):
        self.name = kind
        self.smell = kind().smell
    def smell(self):
        return self.name.kind
f1 = Flower()
f2 = Flower()
```

f1.name()	Error
f1.k	1
Plant().k	3
Rose.k	4

25

1.有()是想执行的，但没传 self 自己，用的也不是 super

2.继承 plant 的 k 一开始被赋予 1

3.每构造一个 plant 对象，类属性 plant.k+1，构造过 f1 f2 两个对象，第三次对象属性 k 为 3

4.rose 再次调用 plant 的 k，第三次了，为 4

Rosemary for Remembrance (FA17 MT2, Q1) Part 2 (solution)

```
class Plant:
    k = 1
    kind = "green"
    def __init__(self):
        self.k = Plant.k
        Plant.k = self.k + 1
        if self.k > 3:
            Plant.name = lambda t:"tree"
            Plant.k = 6
    def name(self):
        return kind
    def __repr__(self):
        s = self.name() + " "
        return s + str(self.k)

class Flower(Plant):
    kind = "pretty"
    def __repr__(self):
        s = self.smell() + " "
        return s + Plant.__repr__(self)
    def smell(self):
        return "bad"
```

```
class Rose(Flower):
    def name(self):
        return "rose"
    def smell(self):
        return "nice"
class Garden:
    def __init__(self,kind):
        self.name = kind
        self.smell = kind().smell
    def smell(self):
        return self.name.kind
f1 = Flower()
f2 = Flower()
```

Plant()	tree 4
Rose()	nice rose 6
Garden(Flower).smell()	'bad'
Garden(Flower).name()	bad tree 6

27

Q2: Shapes

Fill out the skeleton below for a set of classes used to describe geometric shapes. Each class has an `area` and a `perimeter` method, but the implementation of those methods is slightly different. Please override the base `Shape` class's methods where necessary so that we can accurately calculate the perimeters and areas of our shapes with ease.在下面框架中填写描述几何图形的类。每个类都有面积和周长 method，method 实现略有不同。请在必要时覆盖基类 Shape 的方法，这样就能准确地计算出周长和面积。

纯文本

```
import math
pi = math.pi

class Shape: #基类，一些name传入print定义
    """All geometric shapes will inherit from this Shape class."""
    def __init__(self, name):
        self.name = name

    def area(self):
        """Returns the area of a shape"""
        print("Override this method in ", type(self))
```

```
    def perimeter(self):
        """Returns the perimeter of a shape"""
        print("Override this function in ", type(self))

class Circle(Shape):   #父类是Shape
    """A circle is characterized by its radii"""
    def __init__(self, name, radius):
        super().__init__(name) #基类init和子类init共有的除了self以外的要重新传入
        self.radius = radius #新的重新定义

    def perimeter(self):
        """Returns the perimeter of a circle (2πr)"""
        return 2*pi*self.radius

    def area(self):
        """Returns the area of a circle (πr^2)"""
        return pi*self.radius**2

class RegPolygon(Shape): #一父类是Shape
    """A regular polygon is defined as a shape whose angles and side length
    This means the perimeter is easy to calculate. The area can also be don
    def __init__(self, name, num_sides, side_length):name直接()传入,其余重新定
        super().__init__(name) #父类是shape，传入name
        self.num_sides = num_sides #两个不共有的传入
        self.side_length = side_length

    def perimeter(self): #只有周长可以在这里定义很简单边数*边长
        """Returns the perimeter of a regular polygon (the number of sides
        return self.num_sides*self.side_length

class Square(RegPolygon): #父类是RegPolygon
    def __init__(self, name, side_length):
        super().__init__(name, 4, side_length) #子父类init共有name,num_sides,

    def area(self):
        """Returns the area of a square (squared side length)"""
        return self.side_length**2
    #已继承不用定义周长了

class Triangle(RegPolygon): #父类是RegPolygon
    """An equilateral triangle"""
    def __init__(self, name, side_length):
        super().__init__(name, 3, side_length)
```

```
    def area(self): #self是永远会在这里传入的
        """Returns the area of an equilateral triangle is (squared side len
        constant = math.sqrt(3)/4 #可以直接sqrt(3)/4吗
        return constant*self.side_length**2
```

Q3: Cat

Below is a skeleton for the `Cat` class, which inherits from the `Pet` class we saw in the Inheritance introduction. To complete the implementation, override the `__init__` and `talk` methods and add a new `lose_life` method.下面是一个猫类的框架，从 Pet 类中继承。为了完成这个实现，写__init__ 和 talk method，并添加一个新的 lose_life method。

纯文本

```
class Pet:#基类

    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!后面变False
        self.name = name
        self.owner = owner

    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")

    def talk(self):
        print(self.name)

class Cat(Pet):

    def __init__(self, name, owner, lives=9):#类似全局变量
        super().__init__(name, owner) #共有的name owner要传入
        self.lives = lives  #不共有的lives传入，后面-1-1-1

    def talk(self):
        """Print out a cat's greeting.

        >>> Cat('Thomas', 'Tammy').talk() #name字符串 owner
        Thomas says meow!
        """
        print(self.name + ' says meow!') #+后面加''
```

```
    def lose_life(self):#将猫的生命值减少1。到0时is_alive为False，打印...
        """Decrements a cat's life by 1. When lives reaches zero,
        is_alive becomes False. If this is called after lives has
        reached zero, print 'This cat has no more lives to lose.'
        """
        if self.lives > 0:
            self.lives -= 1
            if self.lives == 0:
                self.is_alive = False
        else:
            print("This cat has no more lives to lose.")

    def revive(self): #if（死了）-else（打印）
        """Revives a cat from the dead. The cat should now have
        9 lives and is_alive should be true.一切都重新传入 Can only be called
        on a cat that is dead. If the cat isn't dead, print
        'This cat still has lives to lose.'
        """
        if not self.is_alive:
            self.__init__(self.name, self.owner) 找到lives=9那一行
            #self提前后面不写，lives=9默认不用重新传，这只猫自己重新初始化。不是supe
        else:
            print('This cat still has lives to lose.')
```

Q4: NoisyCat

More cats! Fill in this implementation of a class called `NoisyCat`, which is just like a normal `Cat`. However, `NoisyCat` talks a lot: in fact, it talks twice as much as a regular `Cat`! If you'd like to test your code, feel free to copy over your solution to the `Cat` class above.更多的猫! NoisyCat 的类的实现：像普通的猫。然而 NoisyCat 说的话语是普通猫的两倍

纯文本

```
class NoisyCat(Cat): # Fill me in!
    """A Cat that repeats things twice."""
    def __init__(self, name, owner, lives=9):
        # Is this method necessary? Why or why not?
        super().__init__(name, owner, lives)#上行和cat共有的name,owner,lives
        # No, this method is not necessary because NoisyCat already inherit

    def talk(self):
        """Talks twice as much as a regular cat.
```

```
        >>> NoisyCat('Magic', 'James').talk() #name是Magic
        Magic says meow!
        Magic says meow!
        """
        super().talk()
        super().talk() #不仅可以继承init里的，还可以继承函数
```

Lab HW6 Q1 Mint 铸币厂

A mint is a place where coins are made. In this question, you'll implement a `Mint` class that can output a `Coin` with the correct year and worth.

- Each `Mint` *instance* has a `year` stamp. The `update` method sets the `year` stamp of the instance to the `present_year` *class attribute* of the `Mint` *class*. 铸币厂实例有年份。更新方法输入年份（铸币厂类下当前年类的实例）
- The `create` method takes a subclass of `Coin` (*not* an instance!), then creates and returns an *instance* of that class stamped with the `mint`'s year (which may be different from `Mint.present_year` if it has not been updated.) 创建方法有硬币子类而非实例，创建并返回实例，盖上铸币厂年份章，这个年份章可能与 Mint.present_year 不同
- A `Coin`'s `worth` method returns the `cents` value of the coin plus one extra cent for each year of age *beyond 50*. A coin's age can be determined by subtracting the coin's year from the `present_year` class attribute of the `Mint` class. 价值加当前年-硬币类年份超过 50 的部分

纯文本

```
class Mint:
    """A mint creates coins by stamping on years.

    The update method sets the mint's stamp to Mint.present_year.!!!

    >>> mint = Mint()
    >>> mint.year
    2022
    >>> dime = mint.create(Dime)
    >>> dime.year
    2022
    >>> Mint.present_year = 2102  # 新 Time passes ！！！2102-2022差80年
```

```
    >>> nickel = mint.create(Nickel)
    >>> nickel.year     # The mint has not updated its stamp yet
    2022
    >>> nickel.worth()  # 5 cents + (80 - 50 years)
    35
    >>> mint.update()   # The mint's year is updated to 2102
    >>> Mint.present_year = 2177     # 新 More time passes ！！！2177-2102差
    >>> mint.create(Dime).worth()    # 10 cents + (75 - 50 years)
    35
    >>> Mint().create(Dime).worth()  # 新 A new mint has the current year
    10
    >>> dime.worth()     # 10 cents + (155 - 50 years) 2177-2022=155年
    115
    >>> Dime.cents = 20  # 新 Upgrade all dimes!
    >>> dime.worth()     # 20 cents + (155 - 50 years)
    125
    """
    present_year = 2022

    def __init__(self):
        self.update()

    def create(self, coin): #takes a subclass of Coin (not an instance!)
        "*** YOUR CODE HERE ***"
        #不是实例不用self.coin=coin
        return coin(self.year)
        #returns an instance of that class stamped with the mint's year
        # different from Mint.present_year if it has not been updated

    def update(self):
        "*** YOUR CODE HERE ***"
        #update method sets the year stamp of the instance to the present_y
        self.year=Mint.present_year

class Coin:
    cents = None  # will be provided by subclasses, but not by Coin itself

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

    def worth(self):
        "*** YOUR CODE HERE ***"
        #worth method returns the cents value of the coin plus one extra ce
        #A coin's age can be determined by subtracting the coin's year from
```

```
        return self.cents+max(0,Mint.present_year-self.year-50)

class Nickel(Coin):
    cents = 5

class Dime(Coin):
    cents = 10
```

HW6 Election

Let's implement a game called Election. In this game, two players compete to try and earn the most votes. Both players start with 0 votes and 100 popularity. 两个竞争者竞争更多选票

The two players alternate turns, and the first player starts. Each turn, the current player chooses an action. There are two types of actions:

- The player can debate, and either gain or lose 50 popularity. If the player has popularity `p1` and the other player has popularity `p2`, then the probability that the player gains 50 popularity is `max(0.1, p1 / (p1 + p2))` Note that the `max` causes the probability to never be lower than 0.1. 辩论，得 50 声望以一定概率，失 50 声望以一定概率
- The player can give a speech. If the player has popularity `p1` and the other player has popularity `p2`, then the player gains `p1 // 10` votes and popularity and the other player loses `p2 // 10` popularity.演讲，我得一定投票和声望，对方失一定声望

The game ends when a player reaches 50 votes, or after a total of 10 turns have been played (each player has taken 5 turns). Whoever has more votes at the end of the game is the winner!停止，若到 50 投票或 10 轮各 5 轮，有更多投票的是赢家

Q3: Player

First, let's implement the `Player` class. Fill in the `debate` and `speech` methods, that take in another `Player` `other`, and implement the correct behavior as detailed above. Here are two additional things to keep in mind:

- In the `debate` method, you should call the provided `random` function, which returns a random float between 0 and 1. The player should gain 50 popularity if the random number is smaller than the probability described above, and lose 50 popularity otherwise.调用随机函数，用 if
- Neither players' popularity should ever become negative. If this happens, set it equal to 0 instead. 注意概率总大于 0，提醒取 max(0,)

纯文本

```
# Phase 1: The Player Class
class Player:
    """
    >>> random = make_test_random()
    >>> p1 = Player('Hill')
    >>> p2 = Player('Don')
    >>> p1.popularity
    100
    >>> p1.debate(p2)  # random() should return 0.0
    >>> p1.popularity
    150
    >>> p2.popularity
    100
    >>> p2.votes
    0
    >>> p2.speech(p1)
    >>> p2.votes
    10
    >>> p2.popularity
    110
    >>> p1.popularity
    135

    """

    def __init__(self, name):
        self.name = name
        self.votes = 0
        self.popularity = 100

    def debate(self, other):
        prob=max(0.1, self.popularity / (self.popularity + other.popularity
        if random()<prob:  #注意random()
            self.popularity+=50
```

```
        else:
            self.popularity = max(0, self.popularity - 50)

    def speech(self, other):
        self.votes +=self.popularity // 10
        self.popularity += self.popularity // 10
        other.popularity -= other.popularity // 10

    def choose(self, other): #己方，对方
        return self.speech
```

Q4: Game

Now, implement the `Game` class. Fill in the `play` method, which should alternate between the two players, starting with `p1`, and have each player take one turn at a time. The `choose` method in the `Player` class returns the method, either `debate` or `speech`, that should be called to perform the action.

`play` method 选择两个玩家，从 p1 开始，轮换。其中 `choose` method 用来选，会被调用

In addition, fill in the `winner` method, which should return the player with more votes, or `None` if the players are tied.可能返回三种情况

纯文本

```
# Phase 2: The Game Class
class Game:
    """
    >>> p1, p2 = Player('Hill'), Player('Don')
    >>> g = Game(p1, p2)
    >>> winner = g.play() 是g玩家传入吗
    >>> p1 is winner #提示play之后得到p1，则play函数return self.winner()
    True

    """

    def __init__(self, player1, player2):
        self.p1 = player1
        self.p2 = player2
        self.turn = 0 #提示

    def play(self):
        while not self.game_over():
            self.turn += 1
```

```
            if self.turn % 2==1:
                return self.p1
            else:
                return self.p2
        return self.winner()   #有()？执行得到结果

    def game_over(self):
        return max(self.p1.votes, self.p2.votes) >= 50 or self.turn >= 10

    def winner(self):
        if self.p1.votes > self.p2.votes:
            return self.p1
        elif self.p2.votes > self.p1.votes:
            return self.p2
        else:
            return None
```

Q5: New Players 继承 + 修改新 choose

The `choose` method in the `Player` class is boring, because it always returns the `speech` method. Let's implement two new classes that inherit from `Player`, but have more interesting `choose` methods. 继承别的 player，但改改 choose 方法

Implement the `choose` method in the `AggressivePlayer` class, which returns the `debate` method if the player's popularity is less than or equal to `other`'s popularity, and `speech` otherwise. 若。。。返回 debate

Also implement the `choose` method in the `CautiousPlayer` class, which returns the `debate` method if the player's popularity is 0, and `speech` otherwise.若。。。返回 debate

纯文本

```
# Phase 3: New Players
class AggressivePlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = AggressivePlayer('Don'), Player('Hill')
    >>> g = Game(p1, p2)
    >>> winner = g.play()
    >>> p1 is winner
    True
    """
```

```
    def choose(self, other):
        if self.popularity <= other.popularity:
            return self.debate
        else:
            return self.speech


class CautiousPlayer(Player):
    """
    >>> random = make_test_random()
    >>> p1, p2 = CautiousPlayer('Hill'), AggressivePlayer('Don')
    >>> p1.popularity = 0
    >>> p1.choose(p2) == p1.debate
    True
    >>> p1.popularity = 1
    >>> p1.choose(p2) == p1.debate
    False
    """
    def choose(self, other):
        if self.popularity == 0:
            return self.debate
        else:
            return self.speech
```

lab8 Accout

model a bank account that can handle interactions such as depositing funds or gaining interest on current funds. In the following questions, we will be building off of the `Account` class. Here's our current definition of the class:

纯文本

```
class Account:
    """An account has a balance and a holder.
    >>> a = Account('John')
    >>> a.deposit(10)
    10
    >>> a.balance
    10
    >>> a.interest
    0.02
```

```
    >>> a.time_to_retire(10.25) # 10 -> 10.2 -> 10.404
    2
    >>> a.balance               # balance should not change
    10
    >>> a.time_to_retire(11)    # 10 -> 10.2 -> ... -> 11.040808032
    5
    >>> a.time_to_retire(100)
    117
    """
    max_withdrawal = 10
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        if amount > self.max_withdrawal:
            return "Can't withdraw that amount"
        self.balance = self.balance - amount
        return self.balance
```

lab8 Q3: Retirement

Add a `time_to_retire` method to the `Account` class. This method takes in an `amount` and returns how many years the holder would need to wait in order for the current `balance` to grow to at least `amount`, assuming that the bank adds `balance` times the `interest` rate to the total balance at the end of every year.返回到达 amount 的最短时间，年底计

纯文本

```
class Account:
    """An account has a balance and a holder.
    >>> a = Account('John')
    >>> a.deposit(10)
    10
```

```
    >>> a.balance
    10
    >>> a.interest
    0.02
    >>> a.time_to_retire(10.25) # 10 -> 10.2 -> 10.404
    2
    >>> a.balance # balance should not change另设cash
    10
    >>> a.time_to_retire(11)    # 10 -> 10.2 -> ... -> 11.040808032
    5
    >>> a.time_to_retire(100)
    117
    """
    max_withdrawal = 10
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0
        self.holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        if amount > self.max_withdrawal:
            return "Can't withdraw that amount"
        self.balance = self.balance - amount
        return self.balance

    def time_to_retire(self, amount):
        """Return the number of years until balance would grow to amount.""
        assert self.balance > 0 and amount > 0 and self.interest > 0
        "*** YOUR CODE HERE ***"
        year=0
        cash=self.balance #balance不能变
        while cash<amount:
            cash*=1+self.interest
            year+=1
        return year
```

Q4: FreeChecking

Implement the `FreeChecking` class, which is like the `Account` class from lecture except that it charges a withdraw fee after 2 withdrawals. If a withdrawal is unsuccessful, it still counts towards the number of free withdrawals remaining, but no fee for the withdrawal will be charged.哪怕不成功也算在免费取里面，不过没有额外 fee

Hint: Don't forget that `FreeChecking` inherits from `Account`! Check the Inheritance section in Topics for a refresher.

```
纯文本
```

```
class FreeChecking(Account):
    """A bank account that charges for withdrawals, but the first two are f
    >>> ch = FreeChecking('Jack')
    >>> ch.balance = 20
    >>> ch.withdraw(100)  # First one's free. Still counts as a free withdr
    'Insufficient funds'
    >>> ch.withdraw(3)    # Second withdrawal is also free
    17
    >>> ch.balance
    17
    >>> ch.withdraw(3)    # Ok, two free withdrawals is enough
    13
    >>> ch.withdraw(3)
    9
    >>> ch2 = FreeChecking('John')
    >>> ch2.balance = 10
    >>> ch2.withdraw(3) # No fee
    7
    >>> ch.withdraw(3)  # ch still charges a fee
    5
    >>> ch.withdraw(5)  # Not enough to cover fee + withdraw
    'Insufficient funds'
    """
    withdraw_fee = 1  #多扣1元
    free_withdrawals = 2  #2次免费机会

    def __init__(self,account_holder):
        super().__init__(account_holder) #除self以外
        self.withdrawal_amount=0

    def withdraw(self,amount):
```

```
        self.withdrawal_amount+=1
        fee=0
        if self.withdrawal_amount>self.free_withdrawals:
            fee=self.withdraw_fee
        return super().withdraw(amount+fee) #除了self以w外,应用super的withdra
```