桂林电子科技大学

**实验3 面向对象编程**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **面向对象编程** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计科专业 | | |
| 学 号 | 2100300234 | | | 姓名 | | 谢志锋 | | |
| 实验日期 | 2023 | 年 | 4 | | 月 | | 2 | 日 |
|  |  | | | | | | | |

## 一、实验目的

1. 掌握面向对象程序设计的继承性和多态性
2. 掌握装饰器
3. 掌握LFU类的设计与实现

## 二、实验内容

1. 定义时钟类模拟数字时钟走字
2. 定义类描述平面上的点
3. 工资结算系统
4. 设计LFU缓存类

## 三、实验环境

在Educoder平台进行实验

## 四、实验要求

根据每个实训的每个关卡要求完成代码提交和测评

## 五、实验步骤

第一题：

初始化数字时钟类，用初始化方法，将初始的hour,minute,second都通过类构建时传入，然后编写run方法，每60秒分进1，每60分时进1，当时为12时下一个进一归位1，然后定义显示方法，显示时间

1. from time import sleep
2. class Clock(object):
3. """数字时钟"""
4. def \_\_init\_\_(self, hour=0, minute=0, second=0):
5. """初始化方法
6. :param hour: 时
7. :param minute: 分
8. :param second: 秒
9. """
10. self.\_hour = hour
11. self.\_minute = minute
12. self.\_second = second
13. def run(self):
14. """走字"""
15. self.\_second += 1
16. if self.\_second == 60:
17. self.\_second = 0
18. self.\_minute += 1
19. if self.\_minute == 60:
20. self.\_minute = 0
21. self.\_hour += 1
22. if self.\_hour == 24:
23. self.\_hour = 0
24. def show(self):
25. """显示时间"""
26. return '%02d:%02d:%02d' % \
27. (self.\_hour, self.\_minute, self.\_second)
28. def main():
29. #h为时，m为分，s为秒
30. h,m,s = input().split(',')
31. h = int(h)
32. m = int(m)
33. s = int(s)
34. #        请在此处添加代码       #
35. # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#
36. clock = Clock(h,m,s)
37. for i in range(60):
38. print(clock.show())
39. clock.run()
40. # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#
41. if \_\_name\_\_ == '\_\_main\_\_':
42. main()

## 

第二题：

定义初始化方法，传入初始坐标，定义移动方法，移动新坐标直接传给x,y，定义移动到方法，x为x自身加上增量，y同理，定义计算距离方法，返回所计算出的距离。

from math import sqrt

class Point(object):

    def \_\_init\_\_(self, x=0, y=0):

        """初始化方法

        :param x: 横坐标

        :param y: 纵坐标

        """

        self.x = x

        self.y = y

    def move\_to(self, x, y):

        """移动到指定位置

        :param x: 新的横坐标

        "param y: 新的纵坐标

        :return : 无返回值

        """

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.x=x

        self.y=y

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    def move\_by(self, dx, dy):

        """移动指定的增量

        :param dx: 横坐标的增量

        "param dy: 纵坐标的增量

        :return : 无返回值

        """

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.x=self.x+dx

        self.y=self.y+dy

        return

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    def distance\_to(self, other):

        """计算与另一个点的距离

        :param other: 另一个点,坐标为(other.x,other.y)

        ：return ：返回两点之间的距离

        """

        #        请在此处添加代码       #

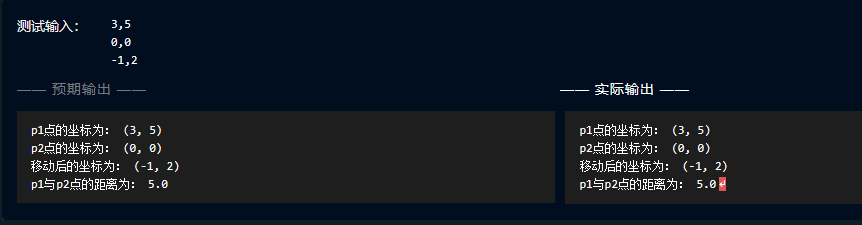
        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return ((other.x - self.x)\*\*2 + (other.y - self.y)\*\*2)\*\*0.5

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    def \_\_str\_\_(self):

        return '(%s, %s)' % (str(self.x), str(self.y))



第三题：

创建员工类，初始化方法传入姓名，用@property定义name方法返回姓名，定义程序员类，初始化方法传入pworking\_hour，和pname，用@property定义working\_hour方法返回工作时长，定义得到的工资，其值等于150\*pworking\_hour，其他部分大部分流程和操作都是差不多的，我就不累述了。

"""

某公司有三种类型的员工 分别是部门经理、程序员和销售员

需要设计一个工资结算系统 根据提供的员工信息来计算月薪

部门经理的月薪是每月固定15000元

程序员的月薪按本月工作时间计算 每小时150元

销售员的月薪是1200元的底薪加上销售额5%的提成

"""

from abc import ABCMeta, abstractmethod

class Employee(object, metaclass=ABCMeta):

    """员工"""

    def \_\_init\_\_(self, name):

        """

        初始化方法

        :param name: 姓名

        """

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.pname = name;

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @property

    def name(self):

        '''返回姓名'''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return self.pname;

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @abstractmethod

    def get\_salary(self):

        """

        获得月薪

        :return: 月薪

        """

        pass

class Manager(Employee):

    """部门经理"""

    def get\_salary(self):

        return 15000.0

class Programmer(Employee):

    """程序员"""

    def \_\_init\_\_(self, name, working\_hour=0):

        """

              初始化方法

              :param name: 姓名

              :param working\_hour：工作时长

              """

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.pworking\_hour = working\_hour

        self.pname = name

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @property

    def working\_hour(self):

        '''返回工作时长'''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return self.pworking\_hour

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @working\_hour.setter

    def working\_hour(self, working\_hour):

        '''

        设置工作时长

        :param working\_hour:工作时长

        '''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.pworking\_hour = working\_hour

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    def get\_salary(self):

        '''返回程序员所得工资'''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return 150.0 \* self.pworking\_hour

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

class Salesman(Employee):

    """销售员"""

    def \_\_init\_\_(self, name, sales=0):

        """

                   初始化方法

                   :param name: 姓名

                   :param sales：销售额

                   """

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.pname = name

        self.psales =  sales

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @property

    def sales(self):

        '''返回销售额'''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return self.psales

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    @sales.setter

    def sales(self, sales):

        '''

        设置销售额

        :param sales:销售额

        '''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        self.psales = sales

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

    def get\_salary(self):

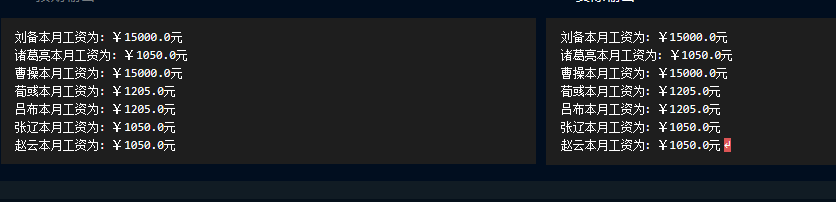
        '''返回销售员所得工资'''

        #        请在此处添加代码       #

        # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

        return 1200 + self.psales \* 0.05

        # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#



第四题：

根据题目中所提示信息，定义get方法，当key不存在时直接返回-1；当key存在时， \_keyMap[key]所对于的value值，定义put方法，如果key已经存在，修改对应的value，并increase，将元素从访问频率key的链表中移除，放到key+1的链表，如果key的链表为空，则从\_\_keyMap中移除key，如果key不存在，创建新元素，设置访问频率为1，然后创建所对应的链表。

from operator import methodcaller

class Node(object):

    """

    双链表中的链表节点对象

    """

    def \_\_init\_\_(self, key=None, value=None, freq=0):

        """

        Args:

            key:对应输入的key

            value:对应输入的value

            freq:被访问的频率

            pre:指向前一个节点的指针

            next:指向后一个节点的指针

        """

        self.key = key

        self.value = value

        self.freq = freq

        self.pre = None

        self.next = None

class LinkedList(object):

    """

    自定义的双向链表

    """

    def \_\_init\_\_(self):

        """

        Args:

            \_\_head:双向链表的头结点

            \_\_tail:双向链表的尾节点

        """

        self.\_\_head = Node()

        self.\_\_tail = Node()

        self.\_\_head.next = self.\_\_tail

        self.\_\_tail.pre = self.\_\_head

    def insertFirst(self, node):

        """

        将指定的节点插入到链表的第一个位置

        Args:

            node:将要插入的节点

        """

        node.next = self.\_\_head.next

        self.\_\_head.next.pre = node

        self.\_\_head.next = node

        node.pre = self.\_\_head

    def delete(self, node):

        """

        从链表中删除指定的节点

        Args:

            node:将要删除的节点

        """

        if self.\_\_head.next == self.\_\_tail:

            return

        node.pre.next = node.next

        node.next.pre = node.pre

        node.next = None

        node.pre = None

    def getLast(self):

        """

        从链表中获取最后一个节点

        Returns:

            双向链表中的最后一个节点，如果是空链表则返回None

        """

        if self.\_\_head.next == self.\_\_tail:

            return None

        return self.\_\_tail.pre

    def isEmpty(self):

        """

        判断链表是否为空，除了head和tail没有其他节点即为空链表

        Returns:

            链表不空返回True，否则返回False

        """

        return self.\_\_head.next == self.\_\_tail

class LFUCache(object):

    """

    自定义的LFU缓存

    """

    def \_\_init\_\_(self, capacity):

        """

        Args:

            \_\_capacity:缓存的最大容量

            \_\_keyMap: key->Node 这种结构的字典

            \_\_freqMap:freq->LinkedList 这种结构的字典

            \_\_minFreq:记录缓存中最低频率

        """

        self.\_\_capacity = capacity

        self.\_\_keyMap = dict()

        self.\_\_freqMap = dict()

        self.\_\_minFreq = 0

    def get(self, key):

        """

        获取一个元素，如果key不存在则返回-1，否则返回对应的value

        同时更新被访问元素的频率

        Args:

            key:要查找的关键字

        Returns:

            如果没找到则返回-1，否则返回对应的value

        """

        #你的代码在这里#

        if key not in self.\_\_keyMap:

            return -1

        node = self.\_\_keyMap[key]

        self.\_\_increment(node)

        return node.value

    def put(self, key, value):

        """

        插入指定的key和value，如果key存在则更新value，同时更新频率

        如果key不存并且缓存满了，则删除频率最低的元素，并插入新元素

        否则，直接插入新元素

        Args:

            key:要插入的关键字

            value:要插入的值

        """

        #你的代码在这里#

        if key in self.\_\_keyMap:

            node = self.\_\_keyMap[key]

            node.value = value

            self.\_\_increment(node)

        else:

            if self.\_\_capacity == 0:

                return

            if len(self.\_\_keyMap)==self.\_\_capacity:

                self.\_\_removeMinFreqElement()

            node = Node(key,value,1)

            self.\_\_increment(node,True)

            self.\_\_keyMap[key] = node

    def \_\_increment(self, node, is\_new\_node=False):

        """

        更新节点的访问频率

        Args:

            node:要更新的节点

            is\_new\_node:是否是新节点，新插入的节点和非新插入节点更新逻辑不同

        """

        if is\_new\_node:

            self.\_\_minFreq = 1

            self.\_\_setDefaultLinkedList(node)

        else:

            self.\_\_deleteNode(node)

            node.freq += 1

            self.\_\_setDefaultLinkedList(node)

            if self.\_\_minFreq not in self.\_\_freqMap:

                self.\_\_minFreq += 1

    def \_\_setDefaultLinkedList(self, node):

        """

        根据节点的频率，插入到对应的LinkedList中，如果LinkedList不存在则创建

        Args:

            node:将要插入到LinkedList的节点

        """

        if node.freq not in self.\_\_freqMap:

            self.\_\_freqMap[node.freq] = LinkedList()

        linkedList = self.\_\_freqMap[node.freq]

        linkedList.insertFirst(node)

    def \_\_deleteNode(self, node):

        """

        删除指定的节点，如果节点删除后，对应的双链表为空，则从\_\_freqMap中删除这个链表

        Args:

            node:将要删除的节点

        """

        if node.freq not in self.\_\_freqMap:

            return

        linkedList = self.\_\_freqMap[node.freq]

        freq = node.freq

        linkedList.delete(node)

        if linkedList.isEmpty():

            del self.\_\_freqMap[freq]

    def \_\_removeMinFreqElement(self):

        """

        删除频率最低的元素，从\_\_freqMap和\_\_keyMap中都要删除这个节点，

        如果节点删除后对应的链表为空，则要从\_\_freqMap中删除这个链表

        """

        linkedList = self.\_\_freqMap[self.\_\_minFreq]

        node = linkedList.getLast()

        linkedList.delete(node)

        del self.\_\_keyMap[node.key]

        if linkedList.isEmpty():

            del self.\_\_freqMap[node.freq]

if \_\_name\_\_ == '\_\_main\_\_':

    operation = eval(input())

    data = eval(input())

    cache = eval("{}({})".format(operation.pop(0), data.pop(0)[0]))

    output = []

    for i, j in zip(operation, data):

        if i == 'put':

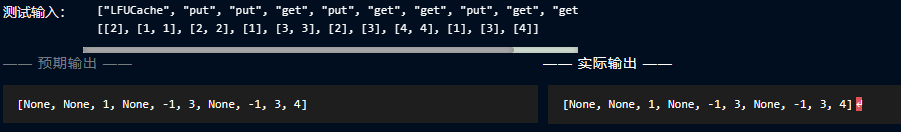
            methodcaller('put', j[0], j[1])(cache)

            output.append(None)

        elif i == 'get':

            output.append(methodcaller('get', j[0])(cache))

    print(output)



## 六、问题记录和实验总结（必写）

在这一次实验中，我掌握了python面向对象的方法，与其他语言的方法相比，python的初始化方法，set和get方法都感觉相差不大，但是我发现python自带的@property极大的简便了获取也就是get方法的定义，而这也是python独特与其他语言的不同之处，python的面向对象是python语言的重要组成部分，python的面向对象方法的掌握，对于我未来的前沿专业知识学习和自身创新创业能力的培养以及学术能力的塑造都有着重要意义。