封装和继承

封装

概念： 将现实世界事务的属性和行为在类中描述为成员变量和成员方法，即为封装.

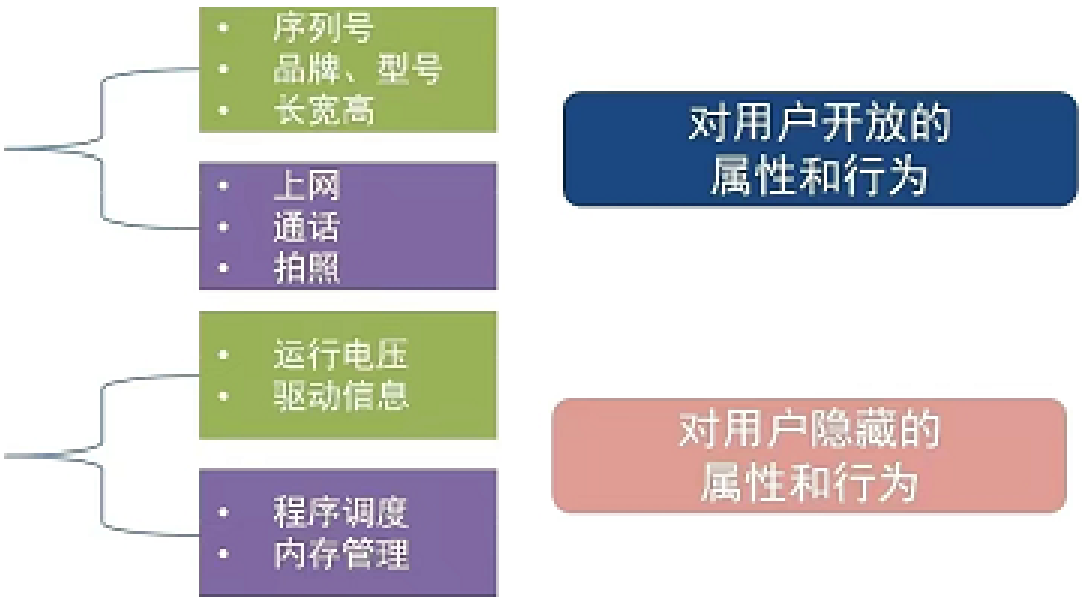
```
# 模板类 (这是我们封装好的一个类)
class Phone():
    def __init__(self,brand,model,price,battery_life):
        self.brand = brand      # brand: 手机品牌（字符串）
        self.model = model    # model: 手机型号（字符串）
        self.price = price    # price: 手机价格（浮点数）
        self.battery_life = battery_life    # battery_life: 手机电池寿命（整数）

    def daDianHua(self,pnumber):
        print(f"即将给号码为{pnumber}的机主打电话")
    def faDuanXin(self,pnumber,text):
        print(f"正在给{pnumber}发送{text}")
"""
怎么写类的构造方法？
1、定义构造函数
2、在构造函数的参数列表中，写上该类的属性
3、在构造函数中写上这样格式的代码： self.属性名 = 属性名

缩进很重要，Table键可以缩进，shift + Table 可以取消缩进
"""
```

在我们之前的学习中，我们封装的属性和行为都是能够公开访问的属性和行为。那当有`属性`和`行为`只能在类中被使用，不能被外部访问的需求的时候怎么办呢？

以手机为例：



手机中有对用户开放的属性和行为，也有对用户隐藏的属性和行为。 既然现实事务有不公开的属性和行为,那么作为现实事务在程序中的映射,也应该支持。**类中提供了私有成员和私有方法的形式来支持。**

```python
class Phone():
    def __init__(self,brand,model,price,battery_life,_rSY='弱私有',__qSY='强私有',__is5G=False):
        # 公有属性
        self.brand = brand      # brand: 手机品牌（字符串）
        self.model = model     # model: 手机型号（字符串）
        self.price = price    # price: 手机价格（浮点数）
        self.battery_life = battery_life    # battery_life: 手机电池寿命（整数）
        # 私有属性
        self._rSY = _rSY    # 弱私有，只是通过名字的方式去提醒程序员，这个是私有的，可以直接强制访问
        self.__qSY = __qSY    # 强私有，通过更改名字的方式，限制对象无法访问
        # 私有属性：当前网络状态
        self.__is5G = __is5G

        # 私有方法出现是为了满足一个需求：有些属性只能在类中被使用，不能被外部访问的需求。
        # 【1、可以让程序员知道哪些是他们需要负责的属性，哪些是不需要负责的。2、可以保护私有属性和方法不被外部污染。】
        # 私有化可以定义不直接对用户开放的属性和行为，既有私有成员的功能，又不会被用户直接使用到。

    # 公有方法
    def daDianHua(self,pnumber):
        # 所以私有属性都可以在内部被self直接调用
        if self.__is5G :
            print(f"即将给号码为{pnumber}的机主打电话")
        else:
            print("即正在4G通讯中")
    def faDuanXin(self,pnumber,text):
         # 所以私有方法都可以在内部被self直接调用
        if self.__jiancha():
            print(f"正在给{pnumber}发送{text}")
        else:
            print("不好意思，请修下手机")
    # 私有方法
    def _rsy(self):
        print("我是弱私有")
    def __qsy(self):
        print("我是强私有")
    # 发短信之前要检查手机硬件软件是否合规  ，这个检查手机硬件软件的方法，不需要外部访问，没必要，它是为共有方法做服务的
    def __jiancha(self):
        # ... 很多很多检查业务代码
        return False


xiaoMi = Phone('小米','civi',3999,60)      # 创建一个具体的手机对象，并且在创建的同时设置好手机的属性值
xiaoMi.daDianHua(110)
xiaoMi.faDuanXin(110,'救救我！')
# print(xiaoMi.brand)    # 访问公有变量
# print(xiaoMi._rSY)    # 弱私有可以强制访问
# # print(xiaoMi.__qSY)     # 强私有不可以直接访问，除非根据Python的私有机制，找到它的别称
# xiaoMi._rsy()    # 访问弱私有方法
# xiaoMi.__qsy()     # 会报错，因为不能强制访问，除非找到改后的名字
```

```
# xiaoMi._Phone__qsy()    #可以运行，因为找到了改后的名字
```

注意：python中的私有变量可以说是一种约定俗成和遵守python命名规则的一种语法，没有保护机制来防止外部的对象对其访问。，因此，在Python编程中应尽量避免使用定义以下划线开头的变量!

继承

```
# 思考:
# 手机需要出新版本(新增指纹识别登录的功能） ， 如果你是设计师,你会如何选择?
# 1.每一代新款手机,都从0开始出设计图 (从头写一个新的类)
# 2,基于老款的设计图,修修改改. (基于已有的Phone类代码进行修改)
```

概念：从父类中继承(复制)来成员变量和成员方法(不含私有)

继承有两种：

单继承：

语法：

```
class 类名(父类1, 父类2, ......, 父类N):
    类内容体
```

```
# 手机模板类（第一代手机设计稿）
class Phone():
    def __init__(self,id,name,price):
        self.id = id #手机编号
        self.name = name
        self.price = price    # 价格
    def call(self):
        print("打电话")
    def msg(self):
        print("发信息")

# 单继承，只继承一个对象
# 手机模板类（第2代手机设计稿）
class Phone_plus(Phone):       #  2代继承1代，是在2代创建的时候，在类名（）中写上1代的名字
    pass       #  pass 用于补全语法，没有任何意思，比如说现在就是声明一个继承了一代手机设计稿的 空类

# Phone666 是 Phone 的儿子，它拥有 Phone 的所有功能（除了私有化之外）
xiaoMi = Phone_plus('civi001','小米',3980)    # 创建了一个对象
xiaoMi.call()
xiaoMi.msg()     # 操作对象干活，调用发短信方法
```

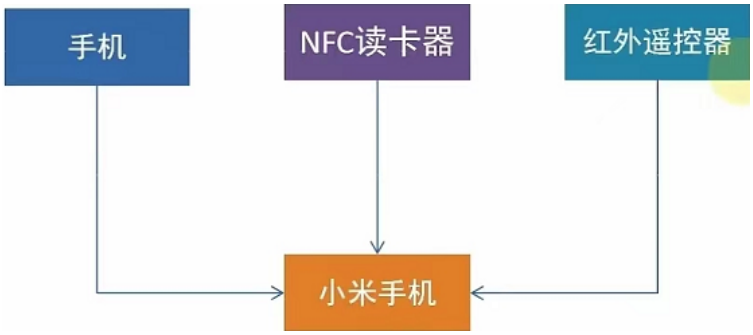
多继承

语法：

```
class 类名(父类1, 父类2, ......, 父类N):
    类内容体
```

注意： *多继承中，如果父类有同名方法或者属性，默认以继承顺序为优先级。即先继承的被保留，后继承的被覆盖。*

多继承理解：




```
# 多继承：  一个儿子可以有多个父亲（一个子类可以继承很多个父类的东西）
class Phone():
    def __init__(self,id,name,price):
        self.id = id #手机编号
        self.name = name
        self.price = price    # 价格
    def call(self):
        print("打电话")
    def msg(self):
        print("发信息")
    def h(self):
        print("1代手机遥控功能")
# NFC读卡器
class Nfc():
    def read_Nfc(self):
        print("读取nfc信息")
    def write_Nfc(self):
        print("写入NFC信息")
# 红外线
class Hong():
    def h(self):
        print("开始遥控")

#  设计一个升级版的手机,拥有正常手机，NFC，红外线功能
class Plus(Hong,Nfc,Phone):
    pass    # 语法补全

oppo = Plus(1001,'OPPO',3980)    # 创建升级版新手机
#测试父类方法名重复时，子类调用优先级
oppo.h()
# 测试这个手机升级到底有没有成功？
oppo.read_Nfc()
oppo.call()
oppo.h()
```

重写

概念：子类继承父类的成员属性和方法后，如果对其不满意，那么可以进行复写。即：在子类中重新定义同名的属性方法即可。

注意： 一旦复写父类成员，那么类对象调用成员的时候，就会调用复写后的新成员。

如果需要使用被复写的父类成员，需要特殊的调用方式。（*只能在子类内部调用父类的同名成员，子类的实体类对象调用默认是调用子类复写*）

```python
# 多继承： 一个儿子可以有多个父亲（一个子类可以继承很多个父类的东西）
class Phone():
    # 父类构造方法
    def __init__(self,id,name,price):
        self.id = id #手机编号
        self.name = name
        self.price = price    # 价格
    def call(self):
        print("打电话")
    def msg(self):
        print("发信息")
    def h(self):
        print("1代手机遥控功能")

# NFC读卡器
class Nfc():
    def read_Nfc(self):
        print("读取nfc信息")
    def write_Nfc(self):
        print("写入NFC信息")

# 红外线
class Hong():
    def h(self):
        print("开始遥控")

#  设计一个升级版的手机,在拥有正常手机，NFC，红外线功能之外，还能6G上网（方法），录入指纹信息（属性）
class Plus(Hong,Nfc,Phone):
    pass
    # 子类构造方法在写的过程中，实际上是会覆盖掉它继承的父类方法(因为名字相同)
    # 为了解决覆盖之后，父类构造方法无法使用的问题，
    # 我们需要在子类构造方法中，去调用父类构造方法，并把需要的值传过去，记得传self
    # 方法重写
    def __init__(self,id,name,price,zhiWen):
        # Phone.__init__(self,id,name,price)   # 需要传参 self
        super().__init__(id,name,price)     # 无须传承 self
        self.zhiWen = zhiWen
    # 重写父类方法
    def call(self,pnumber):
        print(f"正在使用5G通话功能，与{pnumber}进行通话")
    # 自己创建新的方法
    def kaiQi6G(self):
        print("即将开启6G上网")
        print("正在开启6G上网！")

oppo = Plus(1001,'OPP0',3980,"不渝的指纹")     # 创建升级版新手机
#测试父类方法名重复时，子类调用优先级
oppo.h()
# 测试这个手机升级到底有没有成功？
oppo.read_Nfc()
oppo.call(110)  # 调用的是新的打电话功能，也就是我们重写后的代码
oppo.h()
oppo.kaiQi6G()
```

课后作业：

1、将课堂代码敲一遍。

2、封装、继承、重写 分别单独用代码举例（代码需求自己定，只需要用到这三个概念即可）。