多态和多态性

- 多态
  - 多态指的是一类事物有多种形态；
  - 例如：动物：狗、猫、鱼 文件：文本文件、可执行文件
- 多态性
  - 向不同的对象发送同一个指令，不同的对象在接收时会产生不同的行为（方法）
  - 也就是说，每个对象可以用自己的方式去响应这个指令。
  - 指令：用刀
  - 对象：外科医生（做手术）、混混（砍人）、屠夫（杀动物）

```python
# 动物
class Animal:
    def talk(self):
        pass

class Cat:
    def talk(self):
        print("miaomiao")

class Dog:
    def talk(self):
        print("wangwang")

class Pig:
    def talk(self):
        print("aoaoaoao")
```

综上所述，多态性：一个接口，多实现

多态性的好处：

- 增加了程序的灵活性
- 增加了程序的可扩展性

封装

- 第一层：类就是一个袋子，里面可以放属性和方法。
- 第二层：类中定义私有的属性和方法，只有类的内部能够使用，外部无法访问。
  - 在python中用双下划线开头的方式将属性和方法隐藏（设置成私有）

```python
class A:
    num = 100
    __x = 200  # 隐藏属性

    def __foo(self):  # 方法也能隐藏
        print("run foo")

    def fun(self):
        print(self.__x)  # 在类的内部可以访问私有属性

    def bar(self):
        self.__foo()


a = A()

print(a._A__x)

# 在类的外部访问属性
# print(a.num)
# print(a.__x)  # __x 是私有的
# a.fun()
# a.__foo()  # __foo 是一个私有方法，外部无法访问
# a.bar()
# print(A.__dict__)

# 私有化就是在类的定义阶段，对属性名或者方法名 进行变形
# 变形的规则：_类名__属性名、_类名__方法名
```

封装的意义

- 封装数据属性：明确区分内外，控制外部对隐藏的属性的操作行为

```python
class People:
    def __init__(self, name, age):
        self.__name = name  # 把传进来的两个数据属性封装起来
        self.__age = age

    def __str__(self):
        return "我是{},年龄是{}".format(self.__name, self.__age)
```

```
    # 操作姓名和年龄
    def set_info(self, name, age):
        # 在这个间接操作的过程中，我们可以做一些判断
        if not isinstance(name, str):
            print("姓名必须是字符串")
            return
        if not isinstance(age, int):
            print("年龄必须是整型")
            return
        self.__name = name
        self.__age = age

p1 = People("郑俊杰", 18)
print(p1)
# p1.__name = "黄之程"
# print(p1)

p1.set_info("祝伟", 20)
print(p1)
```

- 封装方法：隔离复杂度

```
"""
取款是功能,而这个功能有很多功能组成:
插卡、密码认证、输入金额、打印账单、取钱
#对使用者来说,只需要知道取款这个功能即可,
其余功能我们都可以隐藏起来,很明显这么做
隔离了复杂度,同时也提升了安全性
"""

class ATM:
    # 对于用户来说，没必要关心这些流程
    # 我只取钱，我只要调用一个取款的方法就好了
    def __card(self):
        pass

    def __auth(self):
        pass

```

```
    def __input(self):
        pass

    def __print_bill(self):
        pass

    def __take_money(self):
        pass

    def withdraw(self):
        self.__card()
        self.__auth()
        self.__input()
        self.__print_bill()
        self.__take_money()

# 实例化一个对象（取款的事件）
a = ATM()
a.withdraw()
# 对于使用者来说，类内部的流程隐藏了
```