Universidad Peruana Cayetano Heredia
Facultad de Ciencias e Ingeniería
Departamento de Ciencias Exactas

Programación Avanzada 2025 – Ingeniería Biomédica

Lab. 6.1. Programación Orientada a Objetos. Conceptos Básicos

Septiembre 24, 2025

Cree una carpeta (folder) en el disco D, nómbrela con su apellido paterno seguido de su código. Ejemplo: LOPEZ12345

Programación Orientada a Objetos (POO)

Es un paradigma de programación que permite desarrollar programas teniendo en cuenta la forma que las personas conciben los objetos que nos rodean en el mundo real. Las personas piensan en forma natural sobre las cosas como objetos que tienen atributos y comportamientos, y éstos determinan la forma como interactuamos con ellos. La interacción se lleva a cabo en la mayoría de los casos en forma abstracta.

Interacción Abstracta

Si queremos cambiar el canal de televisión desde nuestros asientos, usamos el control remoto. El control remoto es un **objeto** con un número de **atributos** y **comportamientos** ocultos dentro de él. Sin entender los atributos ocultos (microchip, cableado, etc.), aun podemos saber y esperar que presionando un botón se ejecutará una particular función. En este caso la interacción con el control remoto es *en forma abstracta*. En esto reside la belleza de la POO – el foco está en cómo los objetos se comportan, no en el código requerido para decirles cómo comportarse.

Clases y Objetos

La *Programación Orientada a Objetos* (POO) *encapsula* los datos (*atributos*) y funciones (*comportamientos*) en componentes llamados *clases*. Los datos y funciones de una clase están íntimamente ligados. Una clase es como un plano. Usando un plano, un constructor puede construir una casa. Usando una clase, un programador puede crear un *objeto* (también llamado una *instancia*). Un plano puede ser usado varias veces para construir varias casas. Una clase puede ser usada varias veces para crear varios objetos de la misma clase. Las clases tienen una propiedad llamada *ocultamiento de información*. Esto significa que, aunque los objetos pueden saber cómo comunicarse el uno al otro a través de *interfaces* bien definidas, un objeto normalmente no está permitido saber cómo está implementado otro objeto – los detalles de implementación están ocultos dentro del objeto mismo.

En programación orientada a objetos, las funciones implementadas dentro de los objetos, se suelen llamar **métodos**.

Ejemplo. Diseñar una clase para describir un artículo con los siguientes atributos: código, cantidad y precio.

Instrumentación

Nombre de la clase: articulo

Atributos:	cod	(código del artículo)
	cant	(cantidad actual)
	pre	(precio)
Métodos	cantidad	(muestra la cantidad actual)
	precio	(muestra el precio)
	vender	(reduce la cantidad actual)
	comprar	(incrementa la cantidad actual)

La clase articulo

```
class Articulo:
    def __init__(self,cd,ct,pr):
        self.cod=cd
        self.cant=ct
        self.pre=pr

    def cantidad(self):
        print('Cantidad actual: ',self.cant)

    def precio(self):
        print('Precio:',self.pre)

    def vender(self,x):
        if x<=self.cant:
            self.cant=self.cant-x
        else:
            print('Cantidad insuficiente')

    def comprar(self,x):
        self.cant=self.cant+x
```

1. Guarde la clase Articulo en un archivo llamado `E1.py`
2. ¿Qué es una clase?, ¿qué contiene una clase?
3. ¿Cómo se define una clase?
4. Identifique el estado y comportamiento de los objetos `Articulo`
5. ¿qué es el método `__ init__` ?, ¿se puede obviar su uso?, ¿cuántos constructores puede tener una clase?
6. Cada método tiene un parámetro llamado `self`, ¿cuál es su uso?, ¿se usa cuando se invoca a los métodos?

7. Escriba un programa para probar la clase `Articulo`, guárdelo en un archivo llamado `E1.test.py`
   Cree dos objetos `Articulo`, para cada uno de ellos, imprima la cantidad y precio. Pruebe los métodos `vender()` y `comprar()`, verifique cómo cambió la cantidad.

8. Explique los procesos de **referenciación**, **instanciación** e **inicialización** de un objeto

9. Elabore una nueva versión de la clase `Articulo` en la que los métodos `cantidad`, `precio`, `vender` y `comprar` se implementen en el programa de prueba y no en la clase `Articulo`. Nombre a la nueva clase **`Articulo1`**. Escriba el programa de prueba correspondiente. Cree dos objetos `Articulo1` ingresando los datos del teclado y muestre el contenido de los campos `cant` y `pre` de dos maneras, invocando los métodos `cantidad` y `precio`, y sin invocar los métodos mencionados, accediendo directamente a los campos de los objetos.

10. Analice la diferencia de los objetos `Aticulo` y `Articulo1`. ¿Cuál de las implementaciones sería la más recomendable?, explique su respuesta.

11. Elabore una nueva versión de la clase `Articulo`, de tal manera que los métodos `cantidad()` y `precio()` no impriman directamente en pantalla, más bien deben retornar las variables de instancia `cant` y `pre`. Nombre a la nueva clase `Articulo2`.

12. Escriba un programa para probar la clase `Articulo2`. Genere N objetos `Articulo2` con valores aleatorios y determine el artículo o los artículos de mayor precio. Muestre un listado de los campos de cada objeto.

13. Elabore, en el cuaderno, el diagrama de clases UML para la clase del ejercicio N° 12 y los diagramas UML para los objetos de la mencionada clase.

Ejercicio a resolver en casa:

14. Defina una clase `Empleado` que represente a un empleado de una empresa. Cada empleado tiene los siguientes atributos, código, nombre, genero, ciudad y salario.
    a) Defina un constructor que no reciba parámetros, sin embargo, inicialice las variables de instancia solicitando datos del teclado.
    b) Defina un método de instancia que permita visualizar todos los atributos del objeto.
    c) Defina un método de instancia que permita modificar todos los atributos del objeto.
    d) Agregue a la clase una *variable de clase*, aquel que es accesible por todos los objetos de la clase, es decir, es compartido por todas las instancias de la clase. A esta variable se puede acceder tanto desde fuera de la clase como desde dentro de la clase. Esta variable aumentará en 1 cada vez que se crea una instancia de la clase.

15. Escriba un programa para probar la clase `Empleado`. Cree al menos dos instancias de la clase Empleado. Determinar el empleado con el mayor salario.

16. Elabore el diagrama de clases UML para la clase del ejercicio N° 14.

Guarde todos vuestros programas en una carpeta con el nombre su **Apellido** paterno seguido de vuestro **DNI**, luego comprima esta carpeta. Envíe este archivo a Katherine Navarro katherine.navarro@upch.pe especificando como asunto **Lab6.1**.