



Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Programación Avanzada 2MV7

Practica 3

Autor:
Barrios Mendez Jose Alberto

Boleta: 2022640111

Profesor:

Cruz Mora Jose Luis

Ing. Mecatrónica

13 de marzo de 2024

${\bf \acute{I}ndice}$

1.	Objetivo.	3
2.	Introduccion.	3
	Desarrollo.3.1. Computadoras3.2. Mascotas	3 3
	Resultados 4.1. Computadora	
5.	Conclusiones	12

1. Objetivo.

Desarrollar programas utilizando la herencia y polimorfismo.

2. Introduccion.

Para esta practica continuamos con la programacion orienteda a objetos en python, en el desarrollo de esta practica tendremos la implementacion de una tienda de mascotas, la cual esta dividida en categorias, donde aplicaremos la herencia, que es un pilar fundamental de la programacion orientada a objetos. Ademas tendremos una seccion donde desarrollaremos la herencia en clases de computadoras, crearemos objetos distintos como lo son telefonos, pc, laptops.

Estas practicas se desarrollan con el fin de implementar la herencia así que lo trataremos mas a detalle a continuacion:

3. Desarrollo.

3.1. Computadoras.

Para el desarollo del codigo debemos percatarnos que vamos a tener una herencia desde una clase de tipo Interfaz hacia 4 clases diferentes, las cuales son: Computadora portatil, Computadora de Escritorio, Telefono inteligente, Tablet. Donde a pesar que estas clases son diferentes todas van a heredar los atributos del clase abstracta. En la clase abstracta vamos a tener los atributos de Memoria, Procesador, Almacenamiento, GPU. ademas de tener declarados sus respectivos getters y setters de forma abstracta, es decir, solo se declara para que el programa este al tanto de los metodos que las clases que heredan de la clase computadora deveran implementarlos al crear una instancia, o por decirlo de otra forma, obligarlos a declarar todos sus atributos cada que se crea un objeto. vamos a analizar el codigo.Primero creamos la clase Computadora, declaramos el iniciador con los atrivutos antes mencionados, para aplicar el encapsulamiento nosotros declaramos los atrivutos como privados, aunque en python estos no aplican del todo, es bueno considerarlos para llevar acabo una buena practica de programacion. Cabe recordar que en python para crear los atributos privados es de la forma . __variable

```
class Computadora(metaclass=ABCMeta):
    def __init__(self,memoria,procesador,almacenamiento,gpu):
        self._memoria=memoria
        self._procesador=procesador
        self._almacenamiento=almacenamiento
        self._gpu=gpu
```

Figura 1: Creacion de la clase Computadora con sus respectivos atrivutos privados

Posteriormente, considerando que es una clase abstracta, es importante que todos los getters que declaremos dentro de esta clase tambien deben ser abstractos, esto con la finalidad de obligar al usuario a declarar los atributos al iniciar esta clase, ademas de que no se podran crear las instancias directamente de esta clase.

Para crear los getters abstractos soccorremos al decoraror @property y del decorador @abstractmethod. Observese en la figura 2.

Figura 2: Creacion de la clase Computadora con sus respectivos atrivutos privados

Para satisfacer los requerimientos solicitados en la practica, se crean todos los getters de cada atrivuto, de la misma forma como defininos el getter para el atrivuto procesador, ilustrado en la imagen, recordando nuevamente que estos getters no se implementan en la clase abstracta, es la clase a la que heredamos quien se va a encargar de sobreescribir el metodo para que indique la forma de este metodo.

Posteriormente vamos a crear las subclases que heredaran de la clase Computadora, es importante mencionar que para cada subclase es necesario sobreescribir todos los getters y setters, ya que como estas van a crear instancias u objetos, no puden estar definidas de la forma abstracta, cada subclase tendra un atrivuto adicional, ademas de un metodo respectivamente para el atributo, las subclases son.

Computadora portatil: Para esta subclase fue necesario agregrar un atrivuto protegido llamado tamaño, el cual indica el tamaño de esta misma, los getters y setters quedarian de la forma:

```
class Computadora_Portatil(Computadora):
    def __init__(self, memoria, procesador, almacenamiento, gpu,tamaño):
        super().__init__(memoria, procesador, almacenamiento, gpu)
        self._tamaño=tamaño

@property
def tamaño(self):
    return self._tamaño
    @tamaño.setter
def tamaño(self,nuevo_tamaño):
    self._tamaño=nuevo_tamaño
```

Figura 3: Subclase con el atributo y getters and setters de tamaño

Es importente mencionar que cada subclase sobreescribe cada setter y getter, pero para fines practicos solo mostraremos los que son diferentes entre cada clase

 Computadora Escritorio: Para esta clase solo vamos a definir el atributo y el metodo con sus respectivos getters and setter para el tipo de bocinas que contiene la computadora de escritorio, veamoslo

Figura 4: Subclase con el atributo y getters and setters de bocinas

 Telefono Inteligente: Para esta subclase, lo que va a hacerla diferente va a ser el atrivuto camara, que describiria en un principio la capacidad de esta

```
class TelefonoInteligente(Computadora):

def __init__(self, memoria, procesador, almacenamiento, gpu,camara):

super().__init__(memoria, procesador, almacenamiento, gpu)

self._camara=camara

@property
def camara(self):
    return self._camara

@camara.setter
def camara(self,nueva_camara):
    self._camara=nueva_camara
```

Figura 5: Subclase con el atributo y getters and setters de camara

• Tablet: Para esta subclase la diferencia es que tendremos el atrivuto pantalla

```
class Tablet(Computadora):

def __init__(self, memoria, procesador, almacenamiento, gpu, pantalla):

super().__init__(memoria, procesador, almacenamiento, gpu)

self._pantalla=pantalla

property

def pantalla(self):

return self._pantalla

pantalla._setter

def pantalla(self, nueva_pantalla):

self._pantalla=nueva_pantalla
```

Figura 6: Subclase con el atributo y getters and setters de pantalla

Una vez que hemos definido cada subclase, queda lo mas importante, que es crear los objetos, como sugerencia o instruccion de la practica, nos pide crear al menos 3 objetos por cada subclase, por lo que tendriamos un total de 12 objetos, como estamos utilizando una interfaz o clase abstracta, es necesario que al crea cada objeto, inicializemos todos sus atributos, con

la ayuda de los setters, podemos modificar cada atrivuto posterior a su creacion. veamos en la imagen la creacion de los objetos

```
lap1=Computadora_Portatil(memoria=32,procesador="i3 10th",almacenamiento=512,gpu="Nvidia",tamaño="12 pulagdas")
lap2=Computadora_Portatil(memoria=16,procesador="i7 11th",almacenamiento=1024,gpu="Nvidia 3050",tamaño="15 pulagdas")
lap3=Computadora_Portatil(memoria=64,procesador="ryzen 7560",almacenamiento=1024,gpu="Nvidia 6018",tamaño="14 pulagdas")
pc1=Computadora_Esctritorio(memoria=16,procesador="i9 11th",almacenamiento=1024,gpu="Nvidia 4080",bocinas="bosse")
pc3=Computadora_Esctritorio(memoria=32,procesador="Pentium",almacenamiento=1024,gpu="Nvidia 4080",bocinas="bosse")
pc3=Computadora_Esctritorio(memoria=16,procesador="i9 11th",almacenamiento=1024,gpu="Nvidia 4080",bocinas="bosse")
pc3=Computadora_Esctritorio(memoria=6,procesador="snapdragom 865",almacenamiento=64,gpu="sin datos",camara=24)
cel1=TelefonoInteligente(memoria=6,procesador="snapdragom gen 1",almacenamiento=512,gpu="grafic intel",camara=120)
tab1=Tablet(memoria=6,procesador="A bionic 16",almacenamiento=512,gpu="grafics",pantalla=10.2)
tab2=Tablet(memoria=12,procesador="Kirin 980",almacenamiento=512,gpu="muawei graphics",pantalla=9)
tab1=Tablet(memoria=8,procesador="snapdragon 780",almacenamiento=256,gpu="sin datos",pantalla=8)
```

Figura 7: Creacion de los objetos iniciando sus atributos

3.2. Mascotas.

El objetivo de esta practica es llevar a cabo la herencia en pyhton, la practica nos pide crear varias clases abstractas, comenzando con la clase Mascota, la cual contiene los atributos de: nombre,edad,dueño,tipo. Asi como sus respectivos getters, es importente mencionar que el diagrama de clases nos pide adicionalmente 2 metodos aparte, los cuales son el metodo habla y el metodo toString() que para fines practicos e definido como un metodo str, este metodo se usara para imprimir los datos mas adelante los metodos adicionales son creados de la siguiente forma

```
@property
@abstractmethod
def habla(self):
    pass

@property
@abstractmethod
def __str__(self) -> str:
    pass

@__str__.setter
@abstractmethod
def __str__(self,nuevo_ha) -> str:
pass
```

Figura 8: Creacion de los metodos abstractos habla y str

Posteriormente esta clase abstracta sera la plantilla para crear otras 2 clases abstractas, que seran las clases Domestica y Exotica.

Domestica: Para esta clase se nos pide agregar un nuevo atributo que es el de nivel de ternura, con sus respectivo set y get, ya que este valor si puede ser modificado.

```
class Domestica(Mascota):
    def __init__(self, nombre, edad, dueño, tipo,ternura) -> None:
        super().__init__(nombre, edad, dueño, tipo)
        self.__ternura=ternura

        @property
        @abstractmethod
        def factor_ternura(self):
            pass

        @factor_ternura.setter
        @abstractmethod
        def factor_ternura(self,nuevo_factor_ternura):
            pass
```

Figura 9: Creacion de la clase abstracta Domestica con el metodo ternura

Por su contraparte tenemos la clase abstracta, a comparacion con la clase Domestica este va a tener el metodo Peligro.

Figura 10: Creacion de la clase abstracta Exotica con el metodo peligro

De la clase Domestica se generan 2 subclases que son Perro y Gato, para estas clases se deben reescribir los metodos, ya que a partir de esta clase si se generan objetos. Para ejemplificar mostraremos los metodos habla y str de la clase Gato

```
def __str__(self) -> str:
    return f"Gato: {self._Mascota__nombre}, Edad: {self._Mascota__edad}, Dueño: {self._Mascota__dueño}, ...
    ripo: {self._Mascota__tipo}, Ternura: {self._Domestica__ternura}"

@property
def habla(self):
    return 'MIAU MIAU'
```

Figura 11: sobre escritura de los metodos habla y str de la clase Gato

Para la herencia de la clase Exotica vamos a encotrar que este genera clases con el atributo peligro, las subclases que heredan esto son: vivora, tigre, dinosaurios. Veamos un ejemplo de la sobre escritura de los metodos habla y str de un dinosaurio

```
@property
def habla(self):
    return 'Hola soy un brontosaurio grrrrr'

def __str__(self) -> str:
    return f"{self.habla}. Nombre: {self._Mascota__nombre}, Edad: {self._Mascota__edad},

Dueño: {self._Mascota__dueño}, Tipo: {self._Mascota__tipo}, Peligro: {self._Exotica__peligro}"
```

Figura 12: sobre escritura de los metodos habla y str de la clase Brontosaurio

Una vez que hemos creado todas las clases con ayuda de la herencia, dejamos este archivo para exportarlo como un modulo posteriormente, en un nuevo programa creamos la estructura para el menu dentro al momento de ejecutarlo y este sea de una forma mas legible y con ayuda de funciones para poder validar los datos de entrada, nos es posible crear los objetos, aunado a un ciclo while donde se encuentra la parte principal del codigo:

```
print(imprimir menu())
          opcion = obtener_opcion("\nIngrese una opcion: ", 3)
          os.system("cls")
116 ⊞
          if opcion == 1: ···
          elif opcion == 2:
              print('-'*60)
              print("MASCOTAS ADQUIRIDAS")
              print('-'*60)
              for i in range(len(lista)):
                  print(lista[i])
              input("Presione cualquier tecla para regresar al menú principal.")
              os.system("cls")
          elif opcion == 3:
              print("Hasta la proxima!!! ")
152
```

Figura 13: codigo para añadir mascotas como desee el usuario

en el codigo se puede observar que llamanos la funcion menu para mostrarlo en pantalla, preguntamos sobre la opcion que el usuario desea hacer, en caso de ser 1, se desplega otro menu para mostrar las mascotas, en el caso 2, se muestran las mascotas adquiridas, en la opcion 3 es para salir del programa

4. Resultados

4.1. Computadora

Al momento de ejecutar el codigo nos va a mostrar todos los objetos con sus características, tal como se muestra en la siguiente figura.

```
Computadoras portátiles:
Laptop 1: memoria=32, procesador=i3 10th, almacenamiento=512, gpu=Nvidia, tamaño=12 pulagdas
Laptop 2: memoria=16, procesador=i7 11th, almacenamiento=1024, gpu=Nvidia 3050, tamaño=15 pulagdas
Laptop 3: memoria=64, procesador=ryzen 7560, almacenamiento=256, gpu=Nvidia 6018, tamaño=14 pulagdas
Computadoras de escritorio:
PC 1: memoria=16, procesador=i9 11th, almacenamiento=1024, gpu=Nvidia 4080, bocinas=bosse
PC 2: memoria=32, procesador=Pentium, almacenamiento=256, gpu=none, bocinas=akg
PC 3: memoria=16, procesador=i9 11th, almacenamiento=1024, gpu=Nvidia 4080, bocinas=bosse
Teléfonos inteligentes:
Celular 1: memoria=6, procesador=snapdragom 865, almacenamiento=64, gpu=sin datos, camara=24
Celular 2: memoria=8, procesador=snapdragom gen 1, almacenamiento=512, gpu=grafic intel, camara=108
Celular 3: memoria=4, procesador=A bionic 14, almacenamiento=64, gpu=sin datos, camara=12
Tablet 1: memoria=6, procesador=A bionic 16, almacenamiento=512, gpu=apple grafics, pantalla=10.2
Tablet 2: memoria=12, procesador=Kirin 980, almacenamiento=512, gpu=huawei graphics, pantalla=9
Tablet 3: memoria=8, procesador=snapdragon 780, almacenamiento=256, gpu=sin datos, pantalla=8
PS E:\Programacion\Programacion_Avanzada_2MV7\Practica3\Barrios_Mendez_JoseAlberto
```

Figura 14: Ejecucion del programa numero 1

Es importante mencionar que al usar los setters es posible modificar los valores de cada atributo, esto se puede llevar a cabo de la siguiente forma lap1.memoria=67, lo cual lo convierte en una forma mas sencilla de poder sobreescribir lo que en un principio se definio al momento de crear el objeto

4.2. Mascotas

Al ejecutar el programa nos arroja lo siguiente

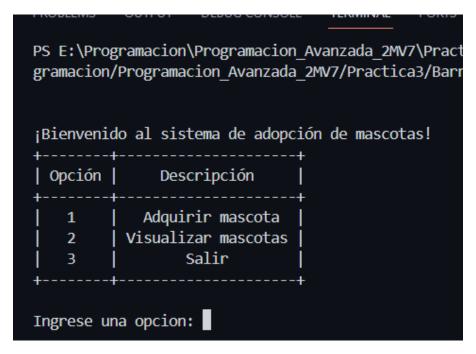


Figura 15: Resultado al ejecutar el programa

vamos a empezar con añadir una mascota, que es la opcion 1:

A continuación, las mascotas disponibles: +	
Domésticas 1 Perro	
2 Gato	
-	
Exóticas 3 Vívora	
4 Tigre	
5 Dinosaurio (Brontosaurio)	
6 Dinosaurio (Raptor)	
7 Dinosaurio (TRex)	

Figura 16: Segundo menu para elegir una mascota

Al seleccionar una opcion por ejemplo Gato (opcion 4) el programa procede a solicitar los datos:

10

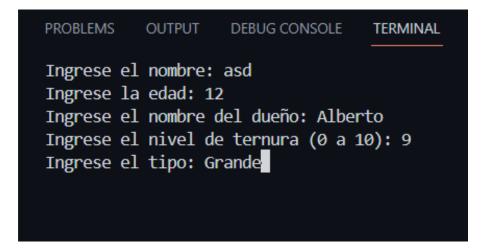


Figura 17: Ingreso de los datos de la mascota seleccionada

Al terminar de introducir los datos, nos arroja lo siguiente:

```
Mascota agregada con exito

+-----+

| Opción | Descripción |

+-----+

| 1 | Adquirir mascota |

| 2 | Visualizar mascotas |

| 3 | Salir

+-----+

Ingrese una opcion:
```

Figura 18: Mascota añadida y vuelta al menu

Para mostrar las mascotas adquiridas es suficiente con seleccionar la opcion numero 2:

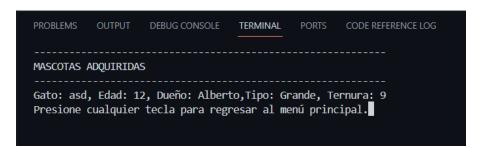


Figura 19: Visualizacion de las mascotas adquiridas

5. Conclusiones.

Al realizar esta practica, de forma independiente a lo extensa que resulto, aunque la programación orientada a objetos puede no siempre conducir a una reducción directa en la cantidad de líneas de código escritas, sus beneficios en términos de mantenimiento, reutilización, flexibilidad y comprensión del código son invaluables. Al adoptar los principios de la programacion orientada a objetos y aplicarlos de manera efectiva en nuestros proyectos, podemos construir sistemas más robustos, flexibles y fáciles de mantener, lo que finalmente conduce a un desarrollo de software más eficiente y exitoso.