Programacion 2

Dependencia entre clases, igualdad y equivalencia superficial

Repaso relaciones de asociación

En la Programación Orientada a Objetos (POO), las relaciones entre clases son fundamentales para modelar sistemas. Las dos relaciones más comunes son asociación y dependencia.

Asociación

indica que dos clases están relacionadas de forma continua y una clase "tiene" una referencia a la otra. Esto significa que los objetos de una clase contienen o utilizan objetos de otra clase. La asociación puede ser de uno a uno o uno a muchos.

características:

- **Duración prolongada**: El objeto "asociado" puede existir durante toda la vida del objeto principal.
- **Referencias directas**: Un objeto "tiene" una referencia a otro objeto en forma de atributo.

Dependencia entre clases

La dependencia entre clases se produce cuando una clase declara una **variable local** o un **parámetro**, o **retorna un resultado** de otra clase.

Decimos que la relación entre objetos es del tipo **usaUn**.

Notemos que un caso particular de dependencia se presenta entre la clase tester con la clase que va a ser verificada.

La relación de dependencia indica que un objeto necesita usar otro objeto en algún momento, pero no lo almacena permanentemente como una referencia interna. Es decir, una clase usa a otra clase de manera temporal o puntual.

Características:

- **Duración corta**: El objeto dependiente no se almacena permanentemente, sino que solo se usa en un método o función.
- **Temporal**: La clase "depende" de la otra para realizar una acción en particular, pero no mantiene una referencia a ella una vez que esa acción ha sido ejecutada.

Caso de estudio: Juego de rol simple

Imaginemos un juego de rol básico en el que los personajes puedan luchar entre sí utilizando diferentes tipos de armas. Cada **personaje** tendrá características únicas como su nombre, vida, ataque y defensa. Además, los personajes podrán equiparse con diversas **armas**, cada una con sus propios valores de daño y tipo. En el juego también hay **cajas sorpresa**, que mejoran o empeoran en distinta medida las características del personaje que la abre, es decir, pueden sumar o restar vida, ataque y defensa.

Caso de estudio: Juego de rol simple

En este juego de rol, cada Personaje se identifica por su nombre, y tiene las siguiente características:

- Nombre: Identifica al personaje.
- Vida: Representa la salud del personaje (comienza en 100 y es el valor máximo).
- Ataque: Indica el poder de ataque del personaje (mínimo 5, máximo 50).
- Defensa: Reduce el daño recibido en combate (mínimo 0, máximo 45).

Caso de estudio: Juego de rol simple

Los personajes pueden equiparse con diferentes Armas. Cada arma tiene sus propias estadísticas:

- Nombre: El nombre del arma (por ejemplo, espada, arco, hacha).
- Daño: Representa cuánto daño inflige el arma.
- Tipo: Tipo de arma (por ejemplo, cuerpo a cuerpo, a distancia).

Cada personaje sólo puede tener un arma para atacar.

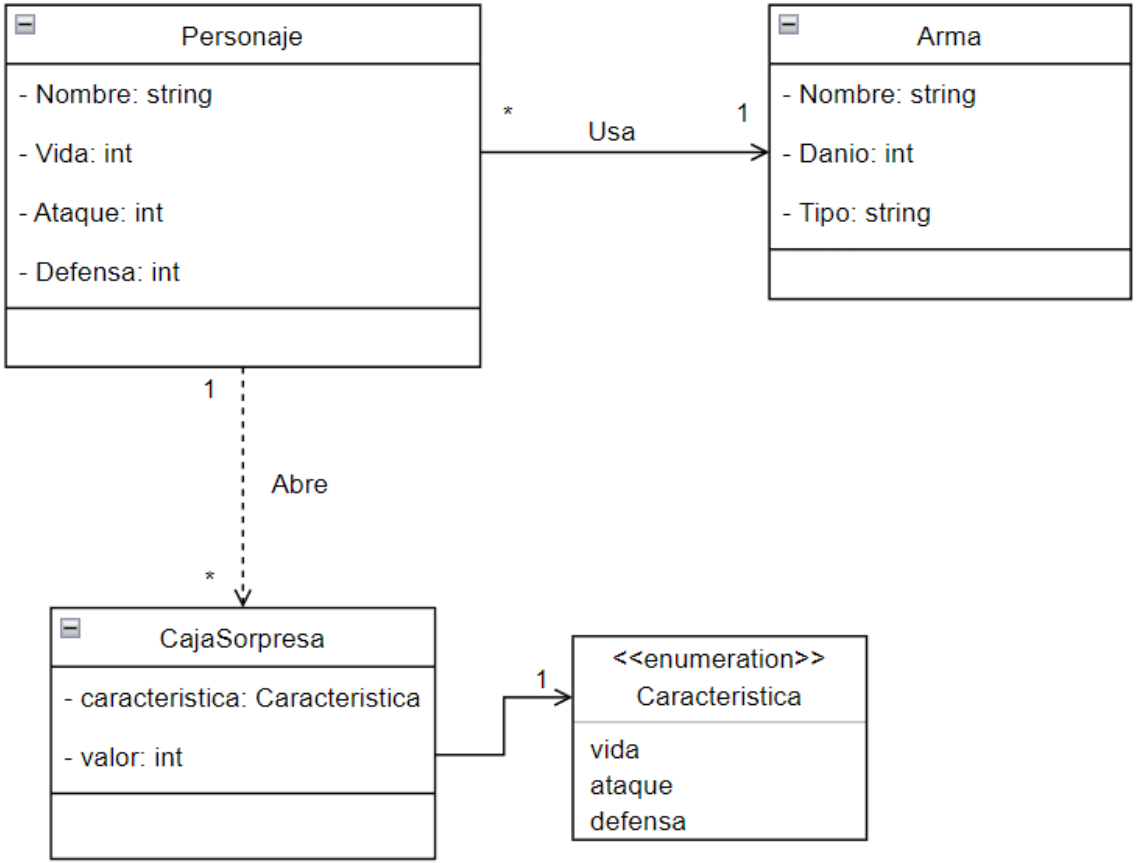
Caso de estudio: Juego de rol simple

En el juego hay Cajas Sorpresa que los personajes pueden abrir. Estas cajas modifican los atributos del personaje de forma aleatoria.

Cada caja otorga un valor aleatorio entre -10 y +20 a uno de los siguientes atributos: vida, ataque o defensa.

La caja puede mejorar o empeorar el estado del personaje.

Caso de estudio: Juego de rol simple - Diagrama de Clases



UML - Enumeration

¿Que es? Una enumeración (o enum) es un tipo de dato que se utiliza para definir un conjunto finito de valores posibles. Cada valor dentro de una enumeración es una constante, que representa una opción dentro de ese conjunto limitado.

¿Para qué se usa?Las enumeraciones se usan para limitar los valores que una variable o atributo puede tomar. Esto mejora la claridad, legibilidad y seguridad del código, ya que se evita que se introduzcan valores fuera de los permitidos.

¿Cuando se usa? Se utiliza una enumeración cuando se tiene un atributo que sólo puede tomar un conjunto específico de valores predefinidos. Por ejemplo:

- Días de la semana: Lunes, Martes, Miércoles, etc.
- Estados de un sistema: Encendido, Apagado, Suspendido.
- Colores: Rojo, Verde, Azul.

¿Como se representa en UML? Una enumeración se representa de la siguiente manera:

1. Se usa una caja similar a una clase, con el estereotipo <<enumeration>> encima.
2. Dentro de la caja, se listan todos los valores posibles de la enumeración.
3. Si un atributo de una clase puede tomar un valor de esa enumeración, el tipo del atributo será el nombre de la enumeración.

Relación de dependencia

Personaje
<<atributos de clase>> - max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int **<<atributos de instancia>>** - nombre : String - vida: entero - ataque: entero - defensa: entero - arma: Arma
<<constructor>> + Personaje (nombre: string, ataque: entero, defensa: entero) **<<consultas>>** **<<comandos>>** + atacar(otroPersonaje: Personaje) + recibirAtaque(valorAtaque: entero) **+ abrirCaja(caja: CajaSorpresa)**

CajaSorpresa
<<atributos de instancia>> - característica: string - valor: entero
<<constructor>> + CajaSorpresa() **<<consultas>>** obtenerCaracteristica():string obtenerValor(): int **<<comandos>>**

Entre Personaje y CajaSorpresa existe una relación de dependencia.

Personaje **Usa** una CajaSorpresa

Clase Personaje

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)

Personaje(nombre, ataque, defensa)
inicializa el personaje con vida completa, sin arma.

atacar(otroPersonaje: Personaje)
Si el personaje está vivo, envía el mensaje recibirAtaque con su valor de ataque al objeto otroPersonaje

recibirAtaque(valorAtaque: entero)
actualiza la vida del personaje restandole el valor de ataque y considerando su valor de defensa

abrirCaja actualiza los valores de vida, ataque y defensa del personaje según el contenido de la caja.
Requiere que luego de abrir la caja ésta sea eliminada

La clase que usa CajaSorpresa asume la responsabilidad de eliminar la caja luego de usarla.

Clase Caja Sorpresa

CajaSorpresa
<<atributos de instancia>> - característica: Caracteristica - valor: entero
<<constructor>> + CajaSorpresa() **<<consultas>>** obtenerCaracteristica():string obtenerValor(): int **<<comandos>>**

<<enumeration>> **Caracteristica**
vida ataque defensa

CajaSorpresa() inicializa de manera aleatoria el contenido de sus atributos internos.

Caso de estudio: Juego de rol simple

<<enumeration>>
Caracteristica

vida
ataque
defensa

CajaSorpresa

<<atributos de instancia>>
- característica: string
- valor: entero

<<constructor>>
+ CajaSorpresa()

<<consultas>>
obtenerCaracteristica():string
obtenerValor(): int

```python
import random
from enum import Enum

class Caracteristica(Enum):
    VIDA = "vida"
    ATAQUE = "ataque"
    DEFENSA = "defensa"

class CajaSorpresa:
    def __init__(self):
        """Inicializa una caja sorpresa con valores aleatorios de vida, ataque y defensa."""
        self.__caracteristica = random.choice(list(Caracteristica))
        self.__valor = random.randint(-10,20)

    def obtenerCaracteristica(self)->Caracteristica:
        """Devuelve la característica de la caja sorpresa."""
        return self.__caracteristica

    def obtenerValor(self)->int:
        """Devuelve el valor de la caja sorpresa."""
        return self.__valor
```

Caso de estudio: Juego de rol simple

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)

```python
class Personaje:
    # Atributos de clase
    MAX_VIDA = 100
    MAX_ATAQUE = 50
    MAX_DEFENSA = 45
    MIN_VIDA = 0
    MIN_ATAQUE = 5
    MIN_DEFENSA = 0

    def __init__(self, nombre:str, ataque:int, defensa:int):
        """
        Inicializa un nuevo personaje.

        Parámetros:
        - nombre: El nombre del personaje.
        - ataque: La cantidad de ataque del personaje.
        - defensa: La cantidad de defensa del personaje.
        """
        if not isinstance(nombre, str) or nombre == "" or nombre.isspace():
            raise ValueError("El nombre debe ser un string válido.")
        if not isinstance(ataque, int) or ataque < Personaje.MIN_ATAQUE or ataque > Personaje.MAX_ATAQUE:
            raise ValueError(f"El ataque debe ser un número entero entre {Personaje.MIN_ATAQUE} y {Personaje.MAX_ATAQUE}.")
        if not isinstance(defensa, int) or defensa < Personaje.MIN_DEFENSA or defensa > Personaje.MAX_DEFENSA:
            raise ValueError(f"La defensa debe ser un número entero entre {Personaje.MIN_DEFENSA} y
{Personaje.MAX_DEFENSA}.")
        self.__nombre=nombre
        self.__vida = Personaje.MAX_VIDA
        self.__ataque=ataque
        self.__defensa=defensa
        self.__arma=None
```

Caso de estudio: Juego de rol simple

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)

```
    #consultas triviales
    def obtenerNombre(self)->str:
        """Devuelve el nombre del personaje."""
        return self.__nombre

    def obtenerVida(self)->int:
        """Devuelve la vida del personaje."""
        return self.__vida

    def obtenerAtaque(self)->int:
        """Devuelve el ataque del personaje."""
        return self.__ataque

    def obtenerDefensa(self)->int:
        """Devuelve la defensa del personaje."""
        return self.__defensa

    def obtenerArma(self)->Arma:
        """Devuelve el arma del personaje."""
        return self.__arma

    def __str__(self)->str:
        """Devuelve una representación de string del personaje."""
        return f"Nombre: {self.__nombre}, Vida: {self.__vida}, Ataque: {self.__ataque}, Defensa:
{self.__defensa}, Arma: {self.__arma if self.__arma!=None else 'Ninguna'} "

    #consultas
    def estaVivo(self):
        """Devuelve True si el personaje está vivo."""
        return self.__vida>0
```

Caso de estudio: Juego de rol simple

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)

```
def establecerVida(self, vida:int):
    """Establece la vida del personaje. Retorna ValueError si no recibe un número
entero."""
    if isinstance(vida, int):
        if vida>=Personaje.MIN_VIDA and vida<=Personaje.MAX_VIDA:
            self.__vida=vida
    else:
        raise ValueError("El valor de vida debe ser un número entero.")

def establecerAtaque(self, ataque:int):
    """Establece el ataque del personaje. Retorna ValueError si no recibe un número
entero."""
    if isinstance(ataque, int):
        if ataque>=Personaje.MIN_ATAQUE and ataque<=Personaje.MAX_ATAQUE:
            self.__ataque=ataque
    else:
        raise ValueError("El valor de ataque debe ser un número entero positivo.")
```

Caso de estudio: Juego de rol simple

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)

```
def establecerDefensa(self, defensa:int):
    """Establece la defensa del personaje. Retorna ValueError si no recibe un número entero."""
    if isinstance(defensa, int):
        if defensa>=Personaje.MIN_DEFENSA and defensa<=Personaje.MAX_DEFENSA:
            self.__defensa=defensa
    else:
        raise ValueError("El valor de defensa debe ser un número entero positivo.")

def establecerArma(self, arma:Arma):
    """Establece el arma del personaje. Retorna ValueError si no recibe un arma."""
    if isinstance(arma, Arma):
        self.__arma = arma
    else:
        raise ValueError("El arma debe ser un objeto de la clase Arma.")
```

Caso de estudio: Juego de rol simple

```
#comandos
def atacar(self, otro_personaje:"Personaje"):
    """
    Ataca a otro personaje.
    Requiere que otro_personaje esté ligado a un objeto Personaje (no sea None), sino retorna ValueError.
    """
    if isinstance(otro_personaje, Personaje):
        if self.estaVivo():
            if self.__arma!=None:
                # Si el personaje tiene un arma, se suma el daño del arma al ataque
                otro_personaje.recibirAtaque(self.__ataque+self.__arma.obtenerDanio())
            else:
                otro_personaje.recibirAtaque(self.__ataque)
    else:
        raise ValueError("El personaje a atacar debe ser un objeto de la clase Personaje.")
```

Caso de estudio: Juego de rol simple

```
    def recibirAtaque(self, valorAtaque:int):
        """Recibe un ataque y ajusta la vida del personaje. Retorna ValueError si no recibe un número entero."""
        if isinstance(valorAtaque, int):
            if Personaje.MIN_ATAQUE <= valorAtaque <= Personaje.MAX_ATAQUE:
                if self.estaVivo():
                    if self.__defensa < valorAtaque:
                        # Si la defensa del personaje es menor al ataque recibido,
                        # se resta la diferencia a la vida del personaje
                        self.__vida -= (valorAtaque-self.__defensa)
        else:
            raise ValueError("El valor de ataque debe ser un número entero positivo.")
```

Caso de estudio: Juego de rol simple

```
    def abrirCaja(self, caja: CajaSorpresa):
        """Abre una caja sorpresa y ajusta los atributos del personaje."""
        if isinstance(caja, CajaSorpresa) and self.estaVivo():
            if caja.obtenerCaracteristica() == Caracteristica.VIDA:
                if self.__vida + caja.obtenerValor() > Personaje.MAX_VIDA:
                    self.__vida = Personaje.MAX_VIDA
                elif self.__vida + caja.obtenerValor() < Personaje.MIN_VIDA:
                    self.__vida = Personaje.MIN_VIDA
                else:
                    self.__vida += caja.obtenerValor()

            elif caja.obtenerCaracteristica() == Caracteristica.ATAQUE:
                if self.__ataque + caja.obtenerValor() > Personaje.MAX_ATAQUE:
                    self.__ataque = Personaje.MAX_ATAQUE
                elif self.__ataque + caja.obtenerValor() < Personaje.MIN_ATAQUE:
                    self.__ataque = Personaje.MIN_ATAQUE
                else:
                    self.__ataque += caja.obtenerValor()

            elif caja.obtenerCaracteristica() == Caracteristica.DEFENSA:
                if self.__defensa + caja.obtenerValor() > Personaje.MAX_DEFENSA:
                    self.__defensa = Personaje.MAX_DEFENSA
                elif self.__defensa + caja.obtenerValor() < Personaje.MIN_DEFENSA:
                    self.__defensa = Personaje.MIN_DEFENSA
                else:
                    self.__defensa += caja.obtenerValor()
```

Relación de asociación

Personaje
<<atributos de clase>> - max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int **<<atributos de instancia>>** - nombre : String - vida: entero - ataque: entero - defensa: entero - **arma: Arma**
<<constructor>> + Personaje (nombre: string, ataque: entero, defensa: entero) **<<consultas>>** **<<comandos>>** + atacar(otroPersonaje: Personaje) + recibirAtaque(valorAtaque: entero) **+** abrirCaja(caja: CajaSorpresa)

Arma
<<atributos de instancia>> - nombre : String - danio: entero - tipo: entero
<<constructor>> + Arma(nombre: string, tipo: string, danio: entero) **<<consultas>>** + obtenerNombre(): string + obtenerTipo(): string + obtenerDanio(): entero + toString(): string

Entre Personaje y Arma existe una relación de asociación.

Personaje **TieneUn** Arma

Caso de estudio: Juego de rol simple

Arma

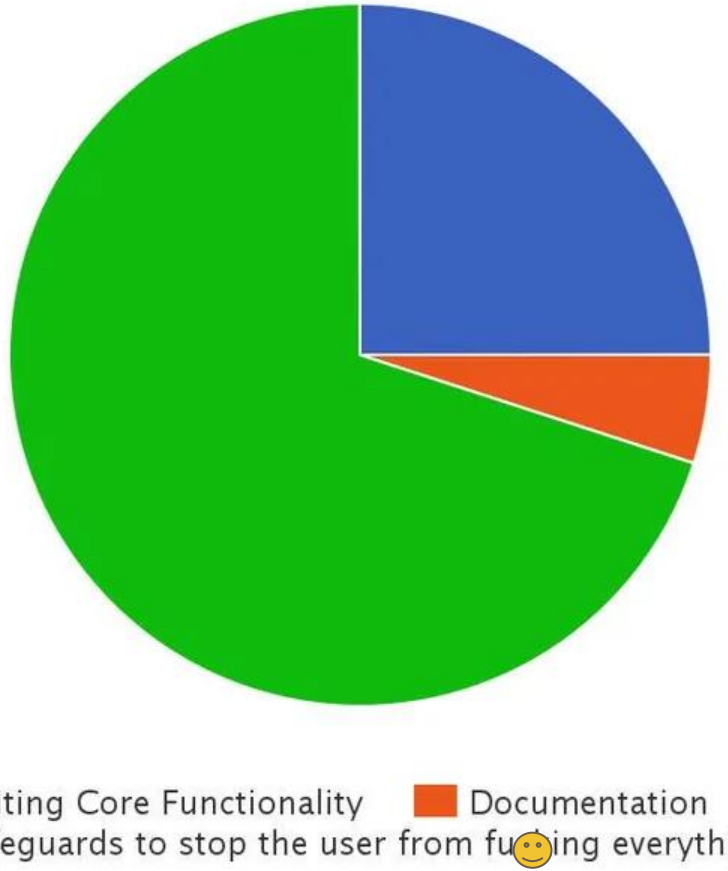
<<atributos de instancia>>
- nombre : String
- danio: entero
- tipo: entero

<<constructor>>
+ Arma(nombre: string, tipo: string, danio: entero)

<<consultas>>
+ obtenerNombre(): string
+ obtenerTipo(): string
+ obtenerDanio(): entero
+ toString(): string

```python
class Arma:
    def __init__(self, nombre:str, tipo:str, danio:int):
        if not isinstance(nombre, str) or nombre == "" or nombre.isspace():
            raise ValueError("El nombre debe ser un string válido.")
        if not isinstance(tipo, str) or tipo == "" or tipo.isspace():
            raise ValueError("El tipo debe ser un string válido.")
        if not isinstance(danio, int) or danio < 0:
            raise ValueError("El daño debe ser un número entero positivo.")
        self.__nombre = nombre
        self.__tipo = tipo
        self.__danio = danio

    def obtenerNombre(self):
        return self.__nombre

    def obtenerTipo(self):
        return self.__tipo

    def obtenerDanio(self):
        return self.__danio

    def __str__(self):
        return f"{self.__nombre}: {self.__tipo} ({self.__danio} de daño)"
```

Time Spent Programming



meta-chart.com

Clase cliente y clase proveedora

Las clases Arma y CajaSorpresa **brindan servicios** que la clase Personaje **usa**. Se dice entonces que las clases Arma y CajaSorpresa son **proveedoras** y la clase Personaje es su **cliente**.

La clase Personaje puede implementarse conociendo **qué** hacen las clases Arma y CajaSorpresa, pero no interesa **cómo** lo hace.

La clase Arma puede implementarse sin saber que va a ser usada por la clase Personaje. (ídem para la clase CajaSorpresa)

Cada clase debe conocer los servicios que brindan sus clases proveedoras, pero no necesita conocer quienes son sus clientes.

Cada clase va a ser verificada por separado y luego en conjunto con las demás clases relacionadas.

Clase tester - CajaSorpresa

En las responsabilidades establecidas por el diseñador se indicó que luego de abrir una caja ésta debía eliminarse (abrirCaja requiere que luego la caja sea eliminada)

```python
from CajaSorpresa import CajaSorpresa
class TesterCajaSorpresa:
    @staticmethod
    def test():
        caja1 = CajaSorpresa()
        caja2 = CajaSorpresa()
        caja3 = CajaSorpresa()
        caja4 = CajaSorpresa()
        caja5 = CajaSorpresa()
        print(f"Característica: {caja1.obtenerCaracteristica()}, Valor: {caja1.obtenerValor()}")
        print(f"Característica: {caja2.obtenerCaracteristica()}, Valor: {caja2.obtenerValor()}")
        print(f"Característica: {caja3.obtenerCaracteristica()}, Valor: {caja3.obtenerValor()}")
        print(f"Característica: {caja4.obtenerCaracteristica()}, Valor: {caja4.obtenerValor()}")
        print(f"Característica: {caja5.obtenerCaracteristica()}, Valor: {caja5.obtenerValor()}")
        del caja1, caja2, caja3, caja4, caja5

if __name__ == "__main__":
    TesterCajaSorpresa.test()
```

del no destruye inmediatamente el objeto en memoria, simplemente elimina la referencia que la variable tiene a ese objeto. El recolector de basura de Python se encargará de eliminar el objeto de la memoria cuando no haya más referencias a él.

Clase tester - Arma

```python
from Arma import Arma

class TesterArma:
    @staticmethod
    def test():
        arma1 = Arma("Espada", "Corte", 10)
        arma2 = Arma("Arco", "Perforación", 8)
        arma3 = Arma("Bastón", "Contundente", 5)
        print(arma1)
        print(arma2)
        print(arma3)
        print(f"{arma1.obtenerNombre()} tiene {arma1.obtenerDanio()} de daño")
        print(f"{arma2.obtenerNombre()} tiene {arma2.obtenerDanio()} de daño")
        print(f"{arma3.obtenerNombre()} tiene {arma3.obtenerDanio()} de daño")

if __name__ == "__main__":
    TesterArma.test()
```

Clase tester - Personaje

```
from Arma import Arma
from CajaSorpresa import CajaSorpresa
from Personaje import Personaje

class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        print(personaje1)
        print(personaje2)
        print(personaje3)
        print(separador)
        print(f"{personaje1.obtenerNombre()} tiene {personaje1.obtenerAtaque()} de ataque y {personaje1.obtenerDefensa()} de
defensa")
        print(f"{personaje2.obtenerNombre()} tiene {personaje2.obtenerAtaque()} de ataque y {personaje2.obtenerDefensa()} de
defensa")
        print(f"{personaje3.obtenerNombre()} tiene {personaje3.obtenerAtaque()} de ataque y {personaje3.obtenerDefensa()} de
defensa")
        print(separador)
```

Clase tester - Personaje

```
#ataques entre personajes sin armas
personaje1.atacar(personaje2)
print(f"{personaje1.obtenerNombre()} atacó a {personaje2.obtenerNombre()} y le queda {personaje2.obtenerVida()} de vida")

personaje2.atacar(personaje1)
print(f"{personaje2.obtenerNombre()} atacó a {personaje1.obtenerNombre()} y le queda {personaje1.obtenerVida()} de vida")

personaje3.atacar(personaje1)
print(f"{personaje3.obtenerNombre()} atacó a {personaje1.obtenerNombre()} y le queda {personaje1.obtenerVida()} de vida")

personaje1.atacar(personaje3)
print(f"{personaje1.obtenerNombre()} atacó a {personaje3.obtenerNombre()} y le queda {personaje3.obtenerVida()} de vida")

personaje2.atacar(personaje3)
print(f"{personaje2.obtenerNombre()} atacó a {personaje3.obtenerNombre()} y le queda {personaje3.obtenerVida()} de vida")

personaje3.atacar(personaje2)
print(f"{personaje3.obtenerNombre()} atacó a {personaje2.obtenerNombre()} y le queda {personaje2.obtenerVida()} de vida")

print(separador)
```

Clase tester - Personaje

```
arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
arma2 = Arma("Arco", "Ataque a distancia", 8)
arma3 = Arma("Bastón", "Cuerpo a cuerpo", 5)
personaje1.establecerArma(arma1)
personaje2.establecerArma(arma2)
personaje3.establecerArma(arma3)
print(f"{personaje1.obtenerNombre()} tiene un arma de {personaje1.obtenerArma().obtenerTipo()}")
print(f"{personaje2.obtenerNombre()} tiene un arma de {personaje2.obtenerArma().obtenerTipo()}")
print(f"{personaje3.obtenerNombre()} tiene un arma de {personaje3.obtenerArma().obtenerTipo()}")
print(separador)
```

Clase tester - Personaje

```
#ataques entre personajes con armas
personaje1.atacar(personaje2)
print(f"{personaje1.obtenerNombre()} atacó a {personaje2.obtenerNombre()} y le queda {personaje2.obtenerVida()} de vida")

personaje2.atacar(personaje1)
print(f"{personaje2.obtenerNombre()} atacó a {personaje1.obtenerNombre()} y le queda {personaje1.obtenerVida()} de vida")

personaje3.atacar(personaje1)
print(f"{personaje3.obtenerNombre()} atacó a {personaje1.obtenerNombre()} y le queda {personaje1.obtenerVida()} de vida")

personaje1.atacar(personaje3)
print(f"{personaje1.obtenerNombre()} atacó a {personaje3.obtenerNombre()} y le queda {personaje3.obtenerVida()} de vida")

personaje2.atacar(personaje3)
print(f"{personaje2.obtenerNombre()} atacó a {personaje3.obtenerNombre()} y le queda {personaje3.obtenerVida()} de vida")

personaje3.atacar(personaje2)
print(f"{personaje3.obtenerNombre()} atacó a {personaje2.obtenerNombre()} y le queda {personaje2.obtenerVida()} de vida")

print(separador)
```

Clase tester - Personaje

```
    #uso de cajas sorpresas
caja_sorpresa1=CajaSorpresa()
caja_sorpresa2=CajaSorpresa()
caja_sorpresa3=CajaSorpresa()
caja_sorpresa4=CajaSorpresa()

print(personaje1)
print(f"La caja sorpresa 1 tiene {caja_sorpresa1.obtenerValor()} de {caja_sorpresa1.obtenerCaracteristica().value}")
personaje1.abrirCaja(caja_sorpresa1)
del caja_sorpresa1
print(personaje1)
print(separador)
```

Personaje **usa** una CajaSorpresa (dependencia --> relación **UsaUn**)

En las responsabilidades establecidas por el diseñador para la clase Personaje se indicó que luego de abrir una caja ésta debía eliminarse (abrirCaja requiere que luego la caja sea eliminada)

Clase tester - Personaje

```
print(personaje2)
print(f"La caja sorpresa 2 tiene {caja_sorpresa2.obtenerValor()} de {caja_sorpresa2.obtenerCaracteristica().value}")
personaje2.abrirCaja(caja_sorpresa2)
del caja_sorpresa2
print(personaje2)
print(separador)
print(personaje3)
print(f"La caja sorpresa 3 tiene {caja_sorpresa3.obtenerValor()} de {caja_sorpresa3.obtenerCaracteristica().value}")
personaje3.abrirCaja(caja_sorpresa3)
del caja_sorpresa3
print(personaje3)
print(separador)
print(personaje1)
print(f"La caja sorpresa 4 tiene {caja_sorpresa4.obtenerValor()} de {caja_sorpresa4.obtenerCaracteristica().value}")
personaje1.abrirCaja(caja_sorpresa4)
del caja_sorpresa4
print(personaje1)
```

En las responsabilidades establecidas por el diseñador para la clase Personaje se indicó que luego de abrir una caja ésta debía eliminarse (abrirCaja requiere que luego la caja sea eliminada)

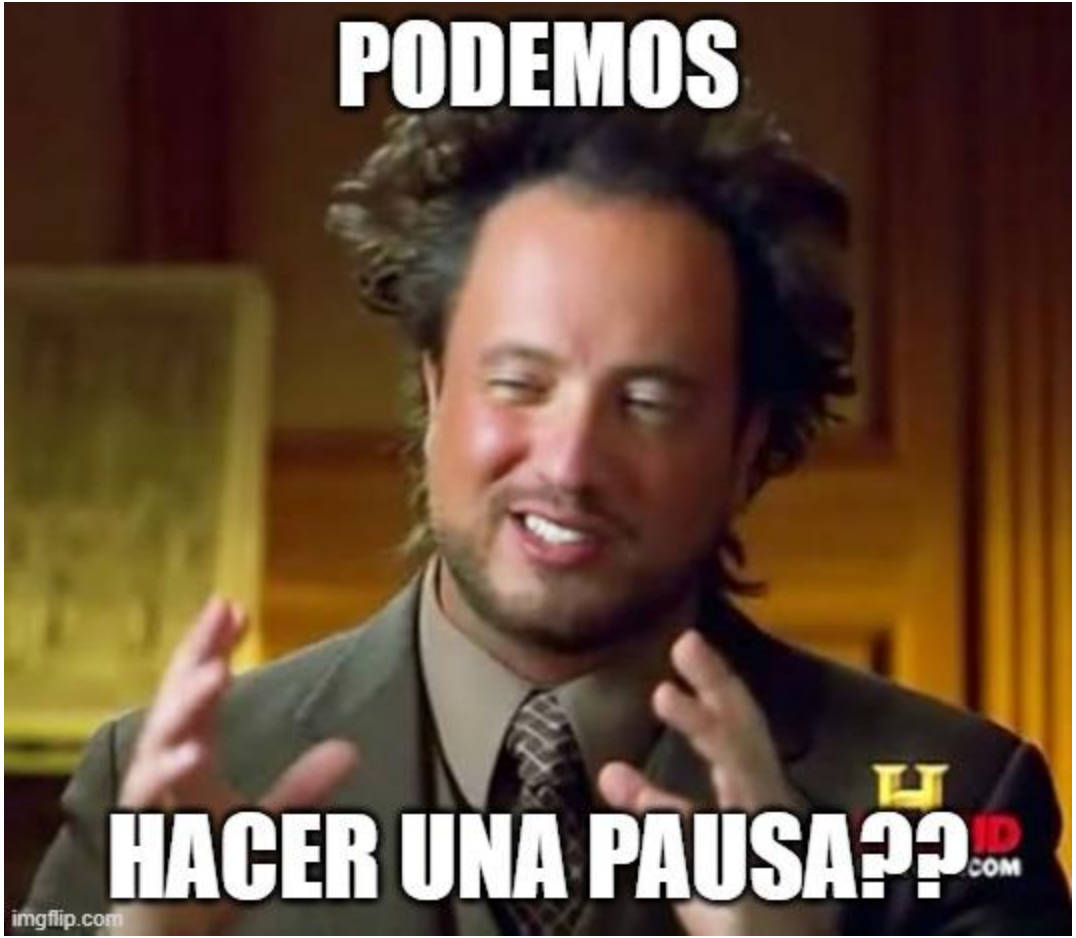
Clase tester - Personaje

Resultado de la ejecución:

```
Nombre: El Mago Loco, Vida: 100, Ataque: 25, Defensa: 20, Arma: Ninguna
Nombre: La Princesa Valiente, Vida: 100, Ataque: 30, Defensa: 22, Arma: Ninguna
Nombre: El Guerrero Cobarde, Vida: 100, Ataque: 25, Defensa: 15, Arma: Ninguna
------------------------------------------------------------------------
El Mago Loco tiene 25 de ataque y 20 de defensa
La Princesa Valiente tiene 30 de ataque y 22 de defensa
El Guerrero Cobarde tiene 25 de ataque y 15 de defensa
------------------------------------------------------------------------
El Mago Loco atacó a La Princesa Valiente y le queda 97 de vida
La Princesa Valiente atacó a El Mago Loco y le queda 90 de vida
El Guerrero Cobarde atacó a El Mago Loco y le queda 85 de vida
El Mago Loco atacó a El Guerrero Cobarde y le queda 90 de vida
La Princesa Valiente atacó a El Guerrero Cobarde y le queda 75 de vida
El Guerrero Cobarde atacó a La Princesa Valiente y le queda 94 de vida
------------------------------------------------------------------------
El Mago Loco tiene un arma de Corte cuerpo a cuerpo
La Princesa Valiente tiene un arma de Ataque a distancia
El Guerrero Cobarde tiene un arma de Cuerpo a cuerpo
------------------------------------------------------------------------
```

Clase tester - Personaje

```
El Mago Loco atacó a La Princesa Valiente y le queda 81 de vida
La Princesa Valiente atacó a El Mago Loco y le queda 67 de vida
El Guerrero Cobarde atacó a El Mago Loco y le queda 57 de vida
El Mago Loco atacó a El Guerrero Cobarde y le queda 55 de vida
La Princesa Valiente atacó a El Guerrero Cobarde y le queda 32 de vida
El Guerrero Cobarde atacó a La Princesa Valiente y le queda 73 de vida
----------------------------------------------------------------------
Nombre: El Mago Loco, Vida: 57, Ataque: 25, Defensa: 20, Arma: Espada - Corte cuerpo a cuerpo (+10 de daño)
La caja sorpresa 1 tiene 9 de ataque
Nombre: El Mago Loco, Vida: 57, Ataque: 34, Defensa: 20, Arma: Espada - Corte cuerpo a cuerpo (+10 de daño)
----------------------------------------------------------------------
Nombre: La Princesa Valiente, Vida: 73, Ataque: 30, Defensa: 22, Arma: Arco - Ataque a distancia (+8 de daño)
La caja sorpresa 2 tiene 5 de vida
Nombre: La Princesa Valiente, Vida: 78, Ataque: 30, Defensa: 22, Arma: Arco - Ataque a distancia (+8 de daño)
----------------------------------------------------------------------
Nombre: El Guerrero Cobarde, Vida: 32, Ataque: 25, Defensa: 15, Arma: Bastón - Cuerpo a cuerpo (+5 de daño)
La caja sorpresa 3 tiene 13 de vida
Nombre: El Guerrero Cobarde, Vida: 45, Ataque: 25, Defensa: 15, Arma: Bastón - Cuerpo a cuerpo (+5 de daño)
----------------------------------------------------------------------
Nombre: El Mago Loco, Vida: 57, Ataque: 34, Defensa: 20, Arma: Espada - Corte cuerpo a cuerpo (+10 de daño)
La caja sorpresa 4 tiene 15 de defensa
Nombre: El Mago Loco, Vida: 57, Ataque: 34, Defensa: 35, Arma: Espada - Corte cuerpo a cuerpo (+10 de daño)
```


PODEMOS
HACER UNA PAUSA??
imgflip.com

Modificación del diseño

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean
+ clonar(): Personaje
+ esIgual(otroPersonaje: Personaje): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)
+ copiarValores(otroPersonaje: Personaje)

clonar() devuelve un nuevo objeto Personaje equivalente al personaje que recibió el mensaje.

esIgual(otroPersonaje: Personaje): boolean devuelve un True si el objeto que recibe el mensaje tiene el mismo estado interno que el objeto recibido por parámetro

copiarValores(otroPersonaje: Personaje) actualiza el estado interno del objeto asignándole los valores del objeto recibido por parámetro.

Cambios en el diseño

Personaje

<<atributos de clase>>
- max_vida, max_ataque, max_defensa, min_vida, min_ataque, min_defensa: int

<<atributos de instancia>>
- nombre : String
- vida: entero
- ataque: entero
- defensa: entero
- arma: Arma

<<constructor>>
+ Personaje (nombre: string, ataque: entero, defensa: entero)

<<consultas>>
+ estaVivo(): boolean
+ clonar(): Personaje
+ esIgual(otroPersonaje: Personaje): boolean

<<comandos>>
+ atacar(otroPersonaje: Personaje)
+ recibirAtaque(valorAtaque: entero)
+ abrirCaja(caja: CajaSorpresa)
+ copiarValores(otroPersonaje: Personaje)

```python
def clonar(self)->"Personaje":
    """Devuelve un clon del personaje."""
    clon = Personaje(self.__nombre, self.__ataque, self.__defensa)
    clon.establecerVida(self.__vida)
    clon.establecerArma(self.__arma)
    return clon

def esIgual(self, otro:"Personaje")->bool:
    """Devuelve True si el personaje es igual a otro, False en caso contrario. Retorna
ValueError si no recibe un Personaje."""
    if isinstance(otro, Personaje):
        return self.__nombre==otro.obtenerNombre() and self.__vida==otro.obtenerVida() and
self.__ataque==otro.obtenerAtaque() and self.__defensa==otro.obtenerDefensa() and
self.__arma==otro.obtenerArma()
    else:
        raise ValueError("El personaje a comparar debe ser un objeto de la clase Personaje.")

def copiarValores(self, otro:"Personaje"):
    """Copia los valores de otro personaje. Retorna ValueError si no recibe un Personaje."""
    if isinstance(otro, Personaje):
        self.__nombre = otro.obtenerNombre()
        self.__vida = otro.obtenerVida()
        self.__ataque = otro.obtenerAtaque()
        self.__defensa = otro.obtenerDefensa()
        self.__arma = otro.obtenerArma()
    else:
        raise ValueError("El personaje a copiar debe ser un objeto de la clase Personaje.")
```

Cambios en la clase tester

```
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

        #cambios en clase tester por cambios en el diseño
        #se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

Diagrama de objetos

```python
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

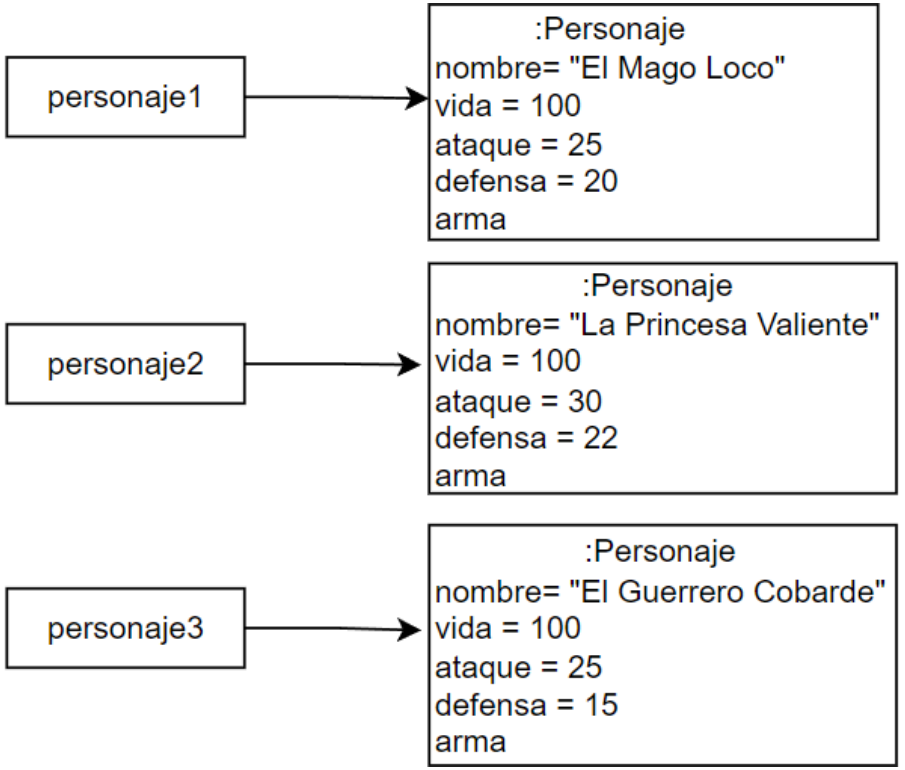


Diagrama de objetos

```
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

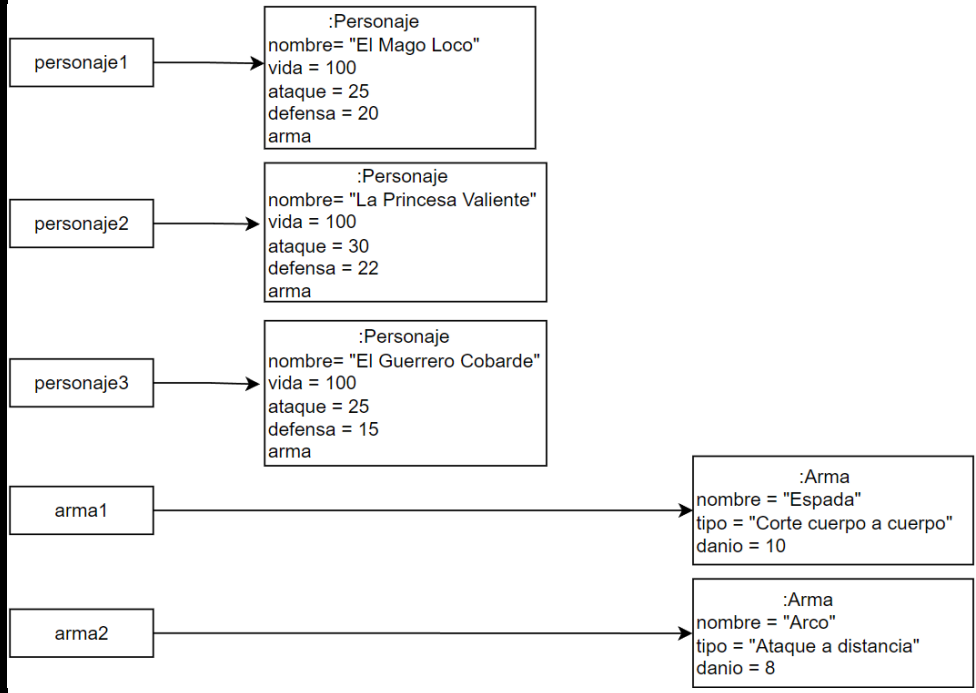


Diagrama de objetos

```
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

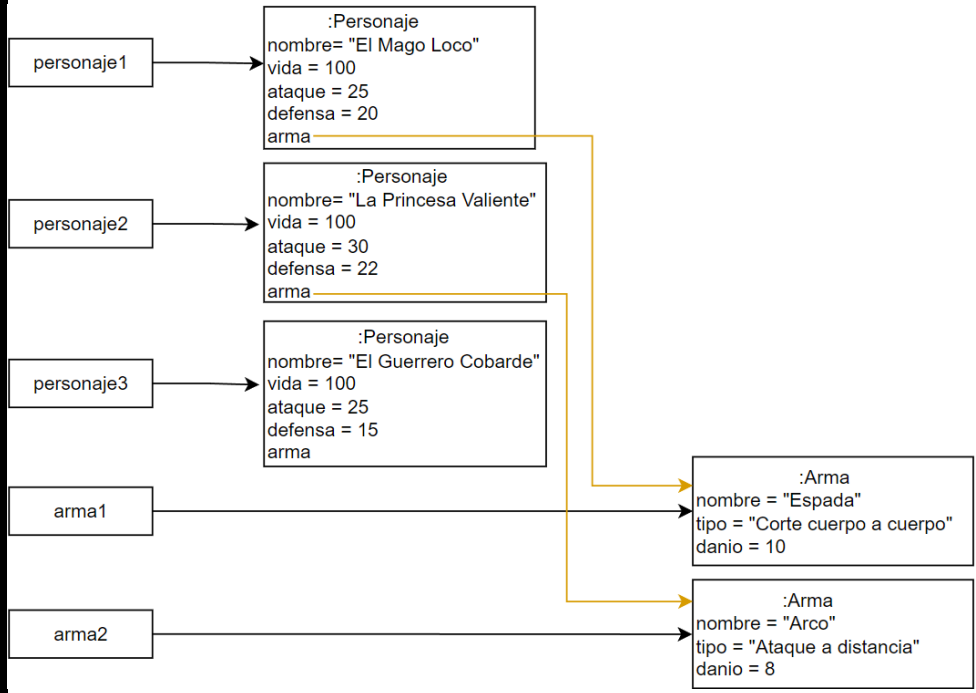


Diagrama de objetos

```python
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

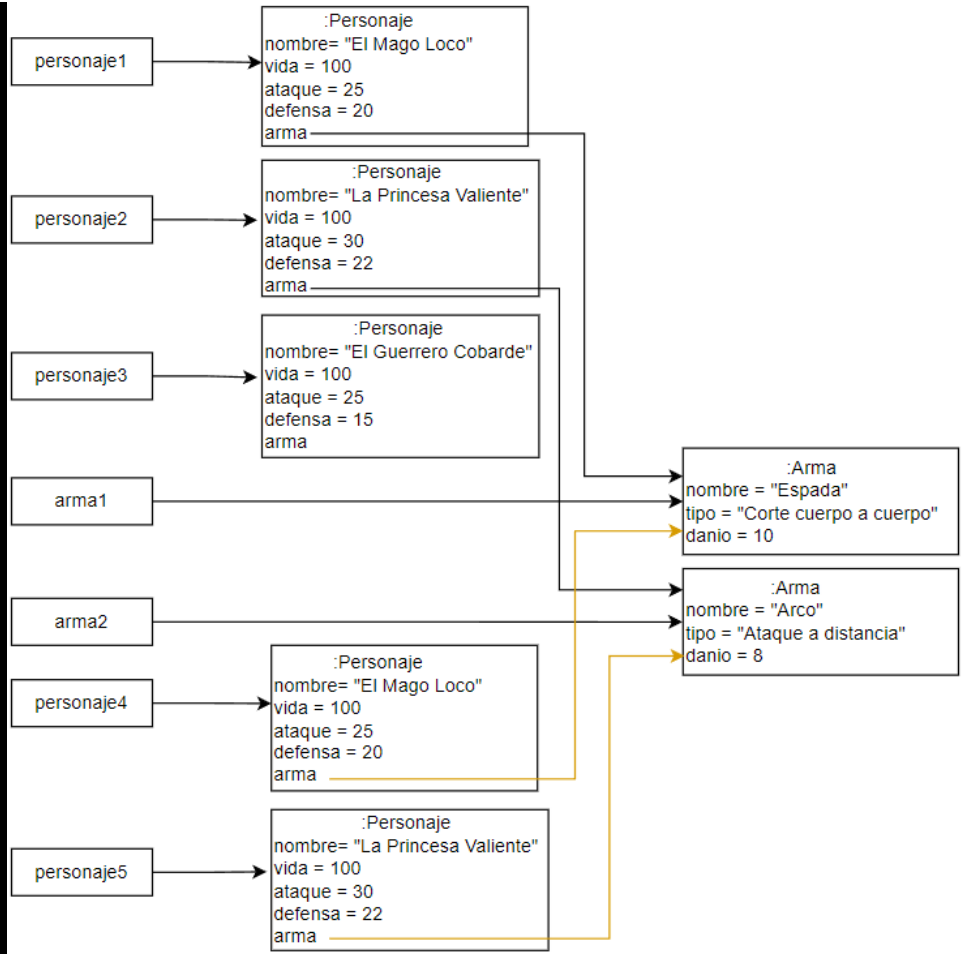


Diagrama de objetos

```
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```

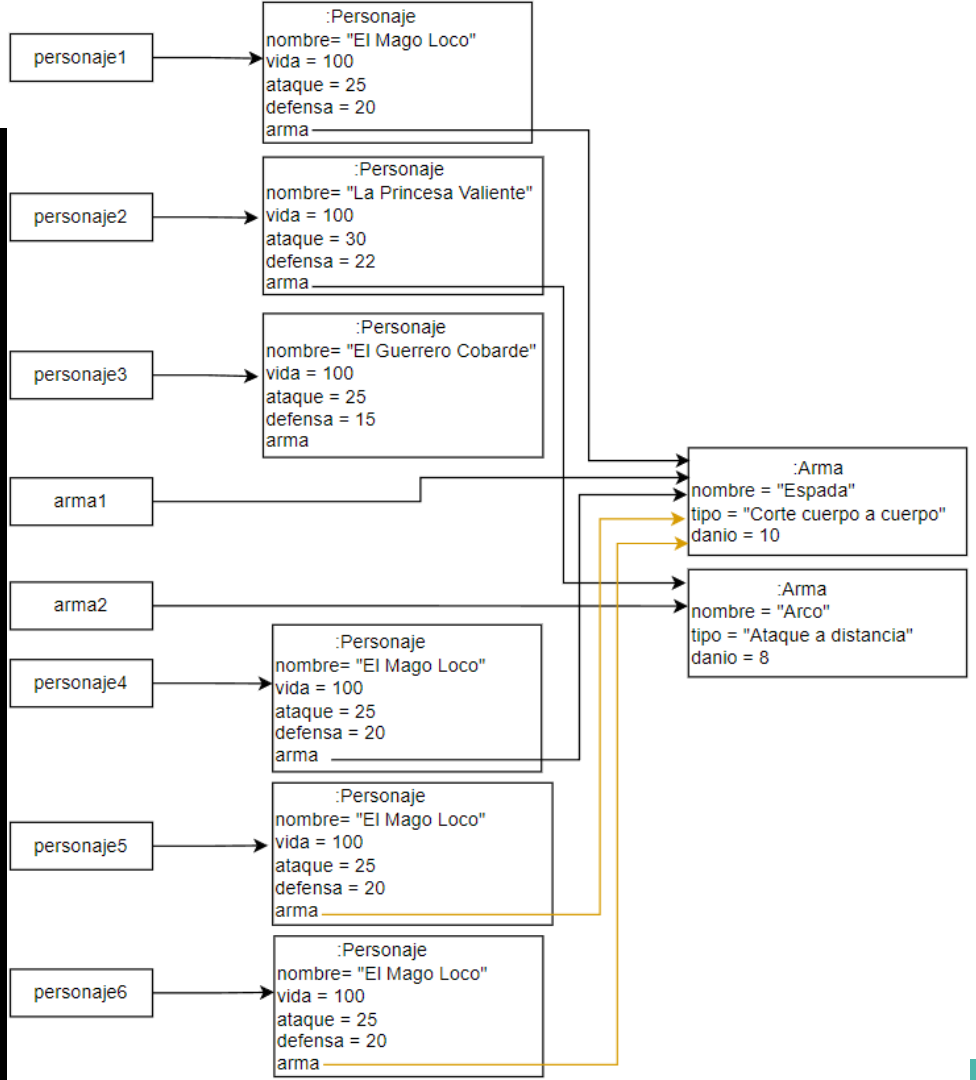
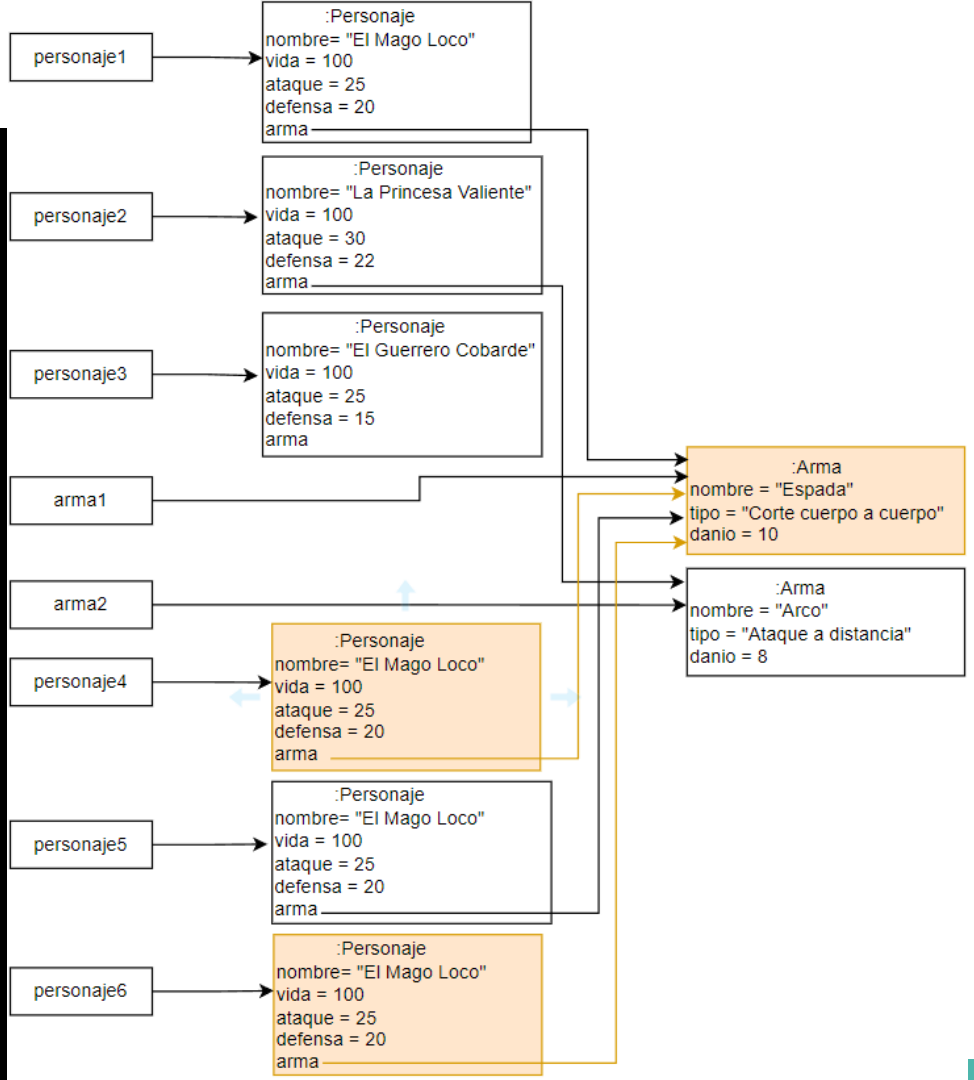


Diagrama de objetos

```
[ ... imports ... ]
class TesterPersonaje:
    @staticmethod
    def test():
        separador = "-"*70
        personaje1 = Personaje("El Mago Loco", 25, 20)
        personaje2 = Personaje("La Princesa Valiente", 30, 22)
        personaje3 = Personaje("El Guerrero Cobarde", 25, 15)
        [ ... ]
        arma1 = Arma("Espada", "Corte cuerpo a cuerpo", 10)
        arma2 = Arma("Arco", "Ataque a distancia", 8)
        [ ... ]
        personaje1.establecerArma(arma1)
        personaje2.establecerArma(arma2)

#cambios en clase tester por cambios en el diseño
#se agregaron los metodos clonar(), esIgual() y copiarValores()
        personaje4 = personaje1.clonar()
        print(personaje4)
        personaje5 = personaje2.clonar()
        personaje5.copiarValores(personaje4)
        personaje6=personaje1.clonar()
        esIgual = personaje6.esIgual(personaje4)
```



```python
def copiarValores(self, otro:"Personaje"):
    """Copia los valores de otro personaje. Retorna ValueError si no recibe un Personaje."""
    if isinstance(otro, Personaje):
        self.__nombre = otro.obtenerNombre()
        self.__vida = otro.obtenerVida()
        self.__ataque = otro.obtenerAtaque()
        self.__defensa = otro.obtenerDefensa()
        self.__arma = otro.obtenerArma()
    else:
        raise ValueError("El personaje a copiar debe ser un objeto de la clase Personaje.")
```

Copia superficial

El comando **copiarValores** asigna al personaje que recibe el mensaje el mismo nombre, la misma cantidad de vida, ataque y defensa que el personaje recibido como parámetro y **lo asocia** también a la misma arma.

```python
def clonar(self)->"Personaje":
    """Devuelve un clon del personaje."""
    clon = Personaje(self.__nombre, self.__ataque, self.__defensa)
    clon.establecerVida(self.__vida)
    clon.establecerArma(self.__arma)
    return clon
```

Clon superficial

El comando **clonar** crea un nuevo personaje inicializándolo con los valores de su estado interno (mismo nombre, ataque y defensa), y antes de retornar el nuevo personaje le establece su valor de vida y su misma arma.

```
def esIgual(self, otro:"Personaje")->bool:
        """Devuelve True si el personaje es igual a otro, False en caso contrario. Retorna
ValueError si no recibe un Personaje."""
         if isinstance(otro, Personaje):
            return self.__nombre==otro.obtenerNombre() and self.__vida==otro.obtenerVida()
and self.__ataque==otro.obtenerAtaque() and self.__defensa==otro.obtenerDefensa() and
self.__arma==otro.obtenerArma()
        else:
            raise ValueError("El personaje a comparar debe ser un objeto de la clase
Personaje.")
```

Comparación superficial

El comando **esIgual** evalúa los valores y referencias del estado interno del objeto recibido por parámetro comparándolas con los valores y referencias del propio estado interno.

El operador relacional == compara la identidad del arma del personaje que recibió el mensaje con la identidad del arma del personaje recibido por parámetro

Copia y Clon superficial

Es una copia de un objeto en la que sólo se duplican las referencias a los objetos contenidos, en lugar de duplicar los propios objetos. Es decir, si el objeto original contiene referencias a otros objetos, la copia superficial no crea nuevos objetos, sino que copia esas referencias.

Esto implica que cualquier modificación en los objetos referenciados por el clon se refleja en el objeto original, ya que ambos apuntan a los mismos datos internos.

En otras palabras, funciona así:

- **Objetos simples**: Si el objeto original contiene tipos de datos primitivos (como números, cadenas, o booleanos), una copia superficial funcionará como una copia real, ya que estos valores se duplican directamente.
- **Objetos compuestos**: Si el objeto contiene referencias a otros objetos (como listas, diccionarios, u otros objetos definidos por el usuario), tanto el objeto original como la copia superficial apuntarán a las mismas instancias de estos objetos. Por lo tanto, si se modifica un objeto compuesto dentro de la copia, el cambio también será visible en el objeto original, ya que ambos comparten la misma referencia.

Para practicar

A un organizador le interesa contar con un sistema para gestionar un torneo donde varios autos compiten entre sí. El torneo se desarrolla en una fecha y una pista en particular, y mantiene una tabla de posiciones de acuerdo al orden de llegada. La pista tiene un nombre que la identifica y una distancia de recorrido en kilómetros.

Los pilotos se inscriben en el torneo con el auto que usarán.

Cada auto tiene un piloto asignado, que es quien lo conduce, y una potencia específica medida en caballos de fuerza. El sistema necesita llevar el control de los autos y sus pilotos para realizar un seguimiento de quienes están compitiendo y cuán potente es cada auto.

Cada auto tiene un piloto asociado, que se identifica por su nombre, apellido, número de inscripción y experiencia en carreras (medida en años).

El auto, por su parte, tiene una marca, un peso, una velocidad máxima y un valor que representa la potencia de su motor medido en caballos de fuerza.

Realiza el diagrama UML y luego la implementación en Python.