Herencia y polimorfismo

Características de un diagrama de clases

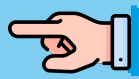
{desafío} latam_

Utilizar el concepto de herencia para la resolución de un problema de baja complejidad acorde al lenguaje Python.

Representar un problema de orientación de objetos mediante un diagrama de clases para su implementación en Python.

{desafío} latam_

- Unidad 1:
  Introducción a la programación orientada a objetos con Python
- Unidad 2:
  Tipos de métodos e interacciones entre objetos
- Unidad 3:
  Herencia y polimorfismo
- Unidad 4:
  Manejo de errores y archivos



Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Bosqueja un diagrama de clases para representar un problema distinguiendo clases, atributos, métodos y relación entre clases.*
- *Implementa un programa utilizando clases, atributos, métodos y relación entre ellos a partir de un diagrama de clases para dar solución a un problema.*

{desafío} latam_

¿Cómo podemos representar un problema de orientación de objetos?

{desafío} latam_

/* Diagrama de clases */

¿Qué es un diagrama de clase?

- Representación gráfica de la estructura de un programa, en él se especifican las clases que componen el programa (con sus respectivos nombres, atributos y métodos) y las relaciones entre ellas.
- Se pueden definir diagramas de clase utilizando el lenguaje unificado de modelado (UML o Unified Modeling Language), el cual es un lenguaje estandarizado de modelado, consistente en un conjunto de diagramas integrados que permiten desarrollar y visualizar artefactos en un proyecto de software.

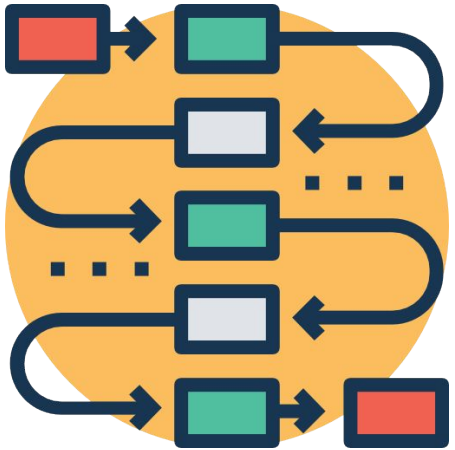
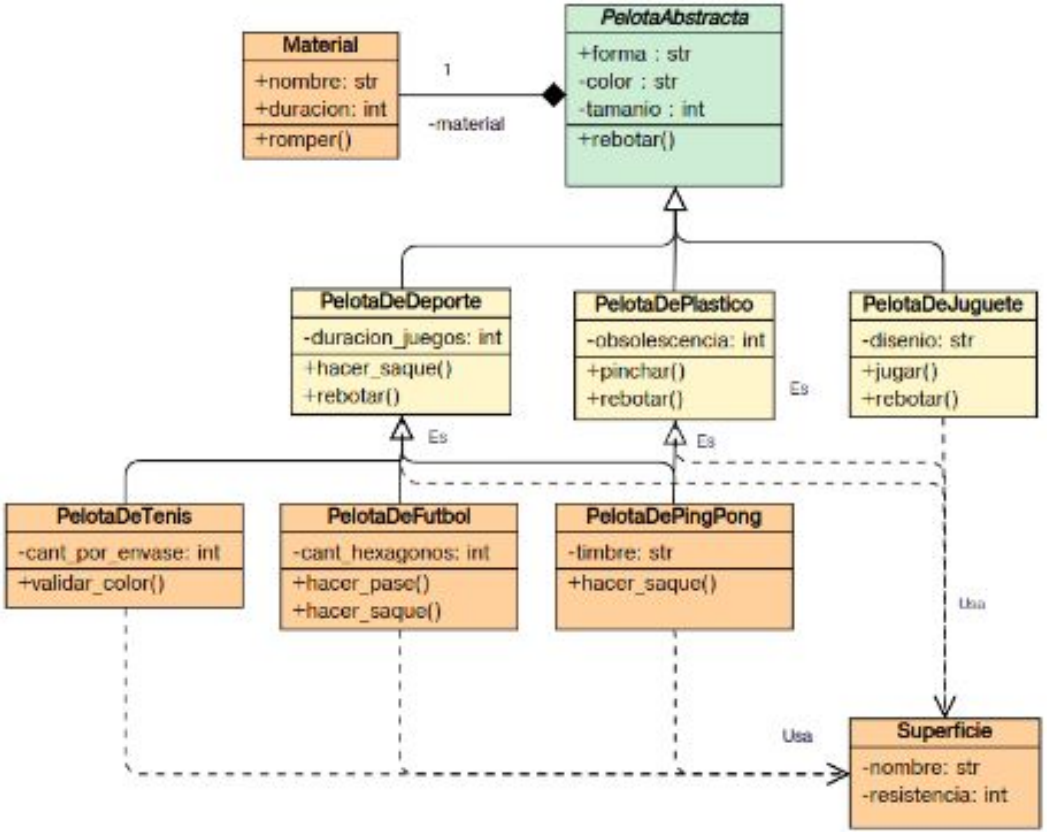


Diagrama de clases

Ejemplo



Existen muchos programas para crear un diagrama de clases en base a cajas de herramientas o similares, sin embargo, no es necesario hacer uso de algún software específico para crear un diagrama de clases.

Software para crear diagramas de clases

Estos son algunas herramientas en las que podemos realizar diagramas de clases:

- StarUML
- Creately
- Lucidchart
- Miro
- Draw.io

Cabe destacar que algunas de estas aplicaciones tienen ciertas limitaciones en su plan gratuito, si desean acceder a funcionalidades avanzadas se debe actualizar a un plan de pago.

¿Para qué sirven?

- Representar el panorama general del programa, enseñando sus elementos principales (clases, atributos, operaciones y relaciones), en él, se representa la jerarquía de las clases involucradas en el programa, siendo capaz de representar herencia, composición y colaboración.
- Al diseñar el programa utilizando el diagrama de clases, se puede determinar dónde se deben aplicar estas relaciones; por lo tanto, el proceso de diseñar el diagrama de clases ayuda a construir el código, y a entender el propósito del programa antes de construirlo.
- Teniendo un diagrama de clases ya construido, y sabiendo cómo interpretarlo, un programador puede también construir un programa a partir de este.

Notación

Clases

Se representan por un rectángulo separado en tres secciones horizontales, que se utilizan para escribir el nombre de la clase, sus atributos, y sus operaciones (métodos) respectivamente.

Notación

Relaciones

La multiplicidad indica cuántos objetos de una clase que se relacionan con otra se puede tener (se usa en colaboración y composición) y puede ser representada como:

- **Rango:** Indica un rango de valores posibles, indicando el mínimo, luego dos puntos, y luego el máximo. Ejemplo: 0..1.
- **Rango con cota:** Indica un valor mínimo, luego dos puntos, y luego un asterisco, indicando que no existe límite. Ejemplo: 0..*.
- **Valor:** Cantidad única posible. Ejemplo: 1.
- **Serie de valores:** Únicos valores posibles. Entre paréntesis, separados por comas. Ejemplo: (2, 5, 7).

Niveles de acceso

de atributos y operaciones

- **+**: Acceso público
- **-** : Acceso privado (solo dentro de la clase)
- **~**: Acceso solo dentro del mismo paquete
- **#**: Acceso protegido (solo dentro de la clase y sus subclases o clases hijas)

/* Construcción de un diagrama de clases */

Atributos en un diagrama de clases

Se detallan en la sección media del rectángulo que representa la clase, considerando lo siguiente:

- Antes del nombre del atributo, se indica el nivel de acceso, según la notación vista previamente. Ejemplo: +color.
- Al lado del nombre del atributo, después de :, se escribe el tipo de dato. Ejemplo: +nombre: str.
- Si el atributo es de clase, va subrayado. Ejemplo: +forma: str.
- También se puede indicar el valor de inicio del atributo, en cuyo caso se omite el tipo de dato, y se indica mediante = el valor por defecto. Ejemplo: +color = "amarillo".

Métodos en un diagrama de clases

Se detallan en la sección final del rectángulo que representa la clase, sin embargo, se debe tener en cuenta que en esta sección lo que se escribe son las **operaciones** (las operaciones corresponden a aquellos comportamientos que caracterizan a la clase).

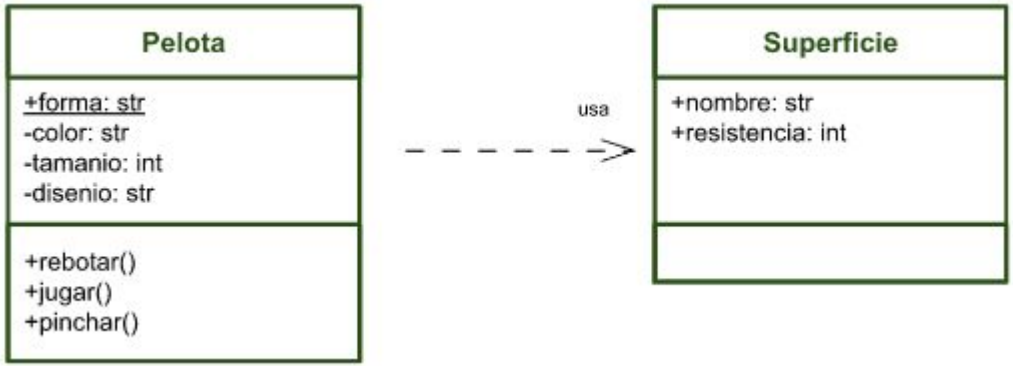
Por tanto:

- No se debe escribir métodos que sean getters, setters, ni métodos privados.
- Se debe indicar el nivel de acceso antes del nombre del método.
- Opcionalmente, se puede agregar el tipo de dato de retorno del método (de la misma forma que el tipo de dato de los atributos). Si se trata de un método abstracto, se debe escribir en cursiva.

Colaboración y composición

en un diagrama de clases

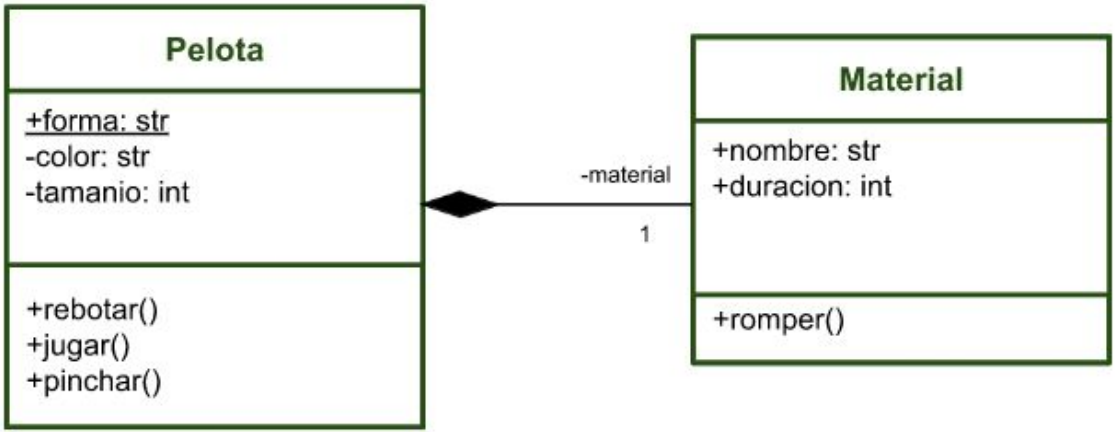
Para representar **colaboración**, se debe dibujar una flecha punteada entre las clases, con una punta negra apuntando hacia la clase que se utiliza (se instancia de otra clase) en la colaboración.



Colaboración y composición

en un diagrama de clases

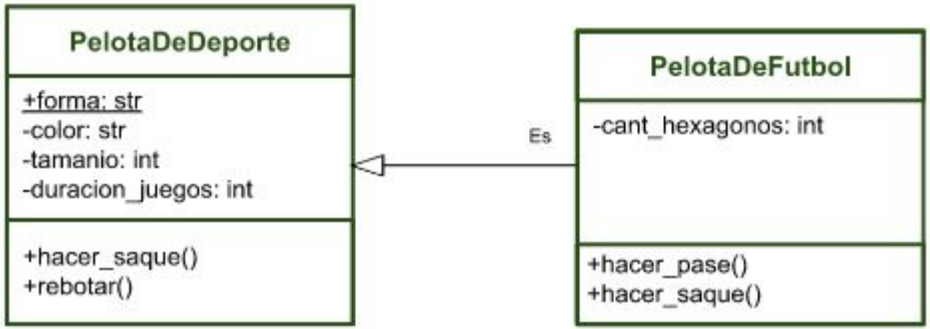
Para representar **composición**, se debe dibujar una línea sólida con un rombo negro sólido en el extremo de la clase compuesta.



Colaboración y composición

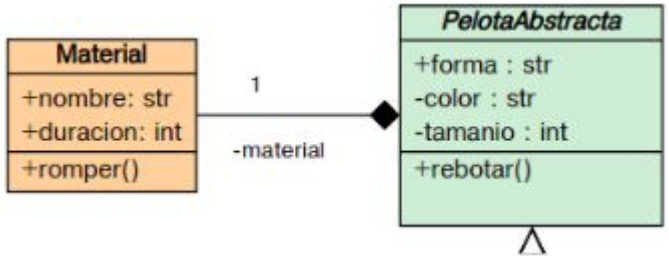
en un diagrama de clases

Para representar **herencia**, se utiliza una flecha sólida con el extremo de punta apuntando hacia la clase padre, donde la punta tiene forma de triángulo y es blanca. Las clases hijas no deben volver a escribir los nombres de los atributos y operaciones heredados, solo se vuelven a escribir las operaciones heredadas que son sobrescritas.



Leyendo un diagrama de clases

"La clase abstracta PelotaAbstracta es un compuesto que tiene 1 componente Material. La clase PelotaAbstracta tiene el atributo de clase forma (público), los atributos de instancia (privados) color, tamanio y forma, y el método abstracto rebotar (público). La clase Material tiene los atributos de instancia (públicos) nombre y duración, y el método (público) romper".

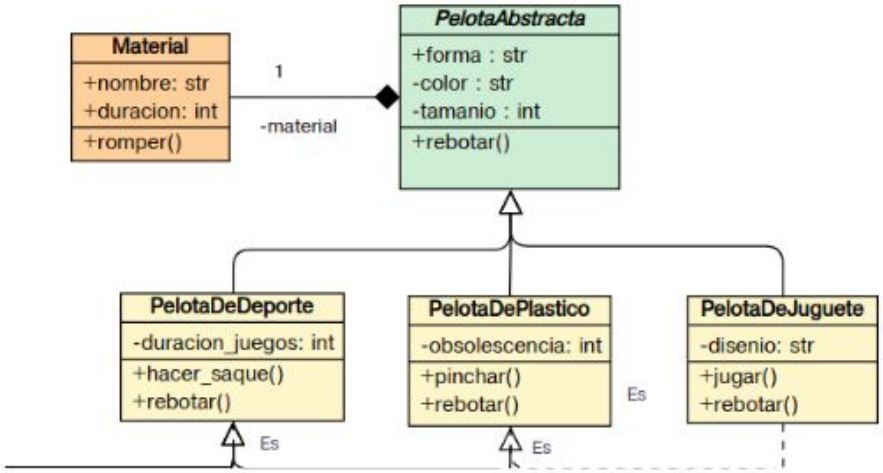


Leyendo un diagrama de clases

"Las clases PelotaDeDeporte, PelotaDePlastico y PelotaDeJuguete son subclases de la clase base PelotaAbstracta, heredando sus atributos. Las tres sobrescriben el método rebotar."

"La clase PelotaDeDeporte tiene, además de los heredados, el atributo de instancia duracion_juegos (privado) y el método hacer_saque (público). La clase PelotaDePlastico tiene, además de los heredados, el atributo de instancia obsolescencia (privado), y el método pinchar (público)."

"La clase PelotaDeJuguete hereda de la clase PelotaDePlastico, y tiene (además de los atributos heredados de PelotaAbstracta y de PelotaDePlastico) el atributo de instancia disenio (privado) y el método jugar (público)"



{desafío} latam_

Leyendo un diagrama de clases

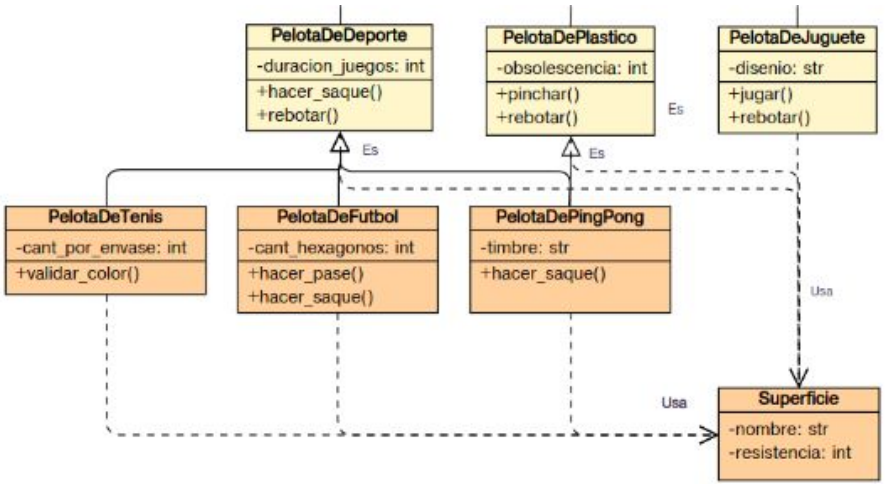
"Las clases PelotaDeDeporte, PelotaDePlastico y PelotaDeJuguete usan la clase Superficie (la clase Superficie colabora con ellas)."

"Las clases PelotaDeTenis, PelotaDeFutbol y PelotaDePingPong son una PelotaDeDeporte (heredan de ella), siendo además la clase PelotaDePingPong una PelotaDePlastico (hereda de ella). Estas tres últimas también usan la clase Superficie en una colaboración."

"La clase PelotaDeTenis, además de los atributos heredados (tanto de PelotaDeDeporte como de PelotaAbstracta), tiene el atributo de instancia (privado) cant_por_envase, el método (público) validar_color y el método rebotar (público, sobrescrito de PelotaDeDeporte)."

"La clase PelotaDeFutbol, además de los atributos heredados (tanto de PelotaDeDeporte como de PelotaAbstracta) tiene el atributo de instancia (privado) cant_hexagonos, y los métodos (públicos) hacer_pase, y hacer_saque (sobrescrito de PelotaDeDeporte)."

"La clase PelotaDePingPong tiene, además de los atributos heredados (tanto de PelotaDeDeporte, como de PelotaDePlastico y de PelotaAbstracta), el atributo de instancia timbre (privado), y el método hacer_saque (sobrescrito de PelotaDeDeporte)".



{desafío} latam_

Ejercicio guiado

"Escribiendo un diagrama de clases"

{desafío} latam_

Escribiendo un diagrama de clases

A continuación, veremos la construcción de un diagrama de clases paso a paso, a partir de un contexto de una problemática real. En este caso, se trata de una red social.

Forma parte del equipo de trabajo de una empresa de creación de software.
En esta ocasión, se le ha encargado a la empresa la construcción de una red social online, y a ti te han solicitado crear el diagrama de clases del prototipo de la aplicación que involucra a la entidad Usuario.

Para ello, debes considerar la siguiente información:

Escribiendo un diagrama de clases

Para crear un usuario

- Se debe solicitar un correo electrónico y una contraseña. Existen reglas de validación para ambos campos, por lo que su asignación (tanto en la creación como cambios de estos valores que se realicen posteriormente) debe ser controlada. Considera que este mecanismo de validación es propio de los usuarios (no debe estar disponible hacia otras entidades).
- El usuario inicialmente tiene una lista de amigos (otros usuarios) que está vacía. Tiene también una foto de perfil (con una imagen por defecto) y un álbum de fotos (lista de fotos) vacío. Considera que estos tres valores requieren de procesos específicos para ser modificados (no se les puede asignar cualquier valor libremente).

Escribiendo un diagrama de clases

El usuario tiene amigos

- Los amigos del usuario son otros usuarios. Un usuario puede tener ningún o muchos amigos, sin un límite superior.

El usuario usa las fotos para reaccionar

- Puede utilizar una foto de otro usuario (no necesariamente amigo) para añadirle una reacción. Al hacerlo, la foto recibida aumentará en 1 su cantidad de reacciones.

Escribiendo un diagrama de clases

El usuario tiene fotos (de perfil o en el álbum)

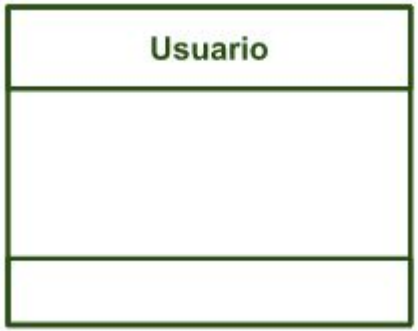
- Considera que una foto no puede existir por sí misma, sino que siempre estará asociada a un usuario (dentro del álbum o como foto de perfil).
- Para crear una foto, se debe solicitar una imagen (una ruta web a un archivo), un ancho y un alto (ambos en píxeles). La foto se crea además con una cantidad de 0 reacciones. Si bien aún no se establecen reglas para modificar estos valores, se cree que en el futuro podría haber limitaciones, por lo que lo debe tener en consideración.
- La foto de perfil es un tipo de foto especial, que al momento de ser creada establece una ruta pre definida para la imagen (imagen por defecto, considere la ruta 'extras/user.png'), y que tiene 400px de ancho y alto. También se crea con 0 reacciones. Además, la foto de perfil tiene un atributo recorte, cuyo valor inicial es una lista con cuatro tuplas con dos valores cada una, para guardar las dimensiones originales de recorte (tamaño original). Este valor se puede modificar siguiendo ciertas reglas respecto de los valores que se puede asignar a cada tupla.

Escribiendo un diagrama de clases

Solución

Paso 1: Creación de la clase Usuario

a. Crear rectángulo que representa la clase

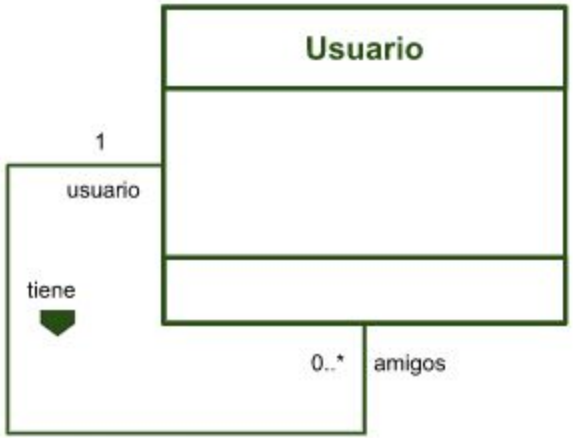


Escribiendo un diagrama de clases

Solución

Paso 1: Creación de la clase Usuario

b. Crear relaciones

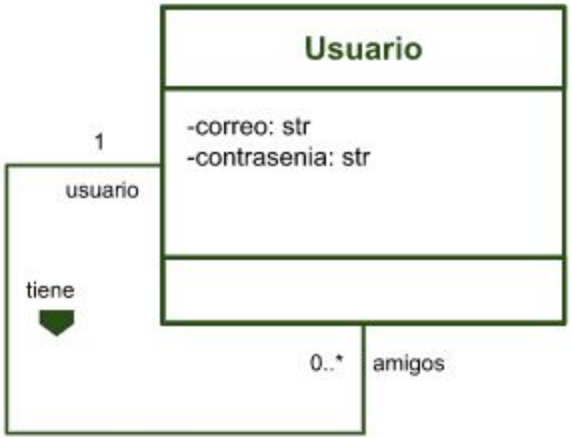


Escribiendo un diagrama de clases

Solución

Paso 1: Creación de la clase Usuario

c. Creación de atributos



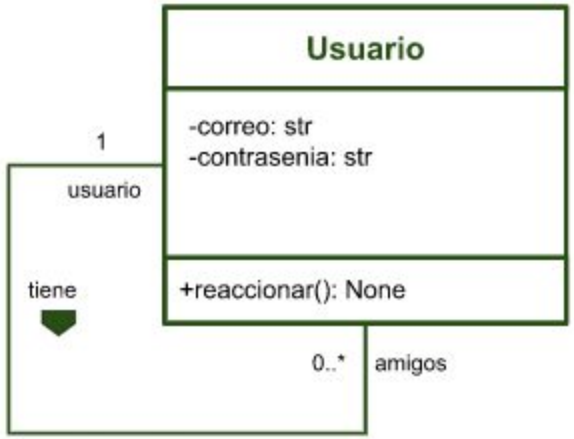
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 1: Creación de la clase Usuario

b. Creación de operaciones



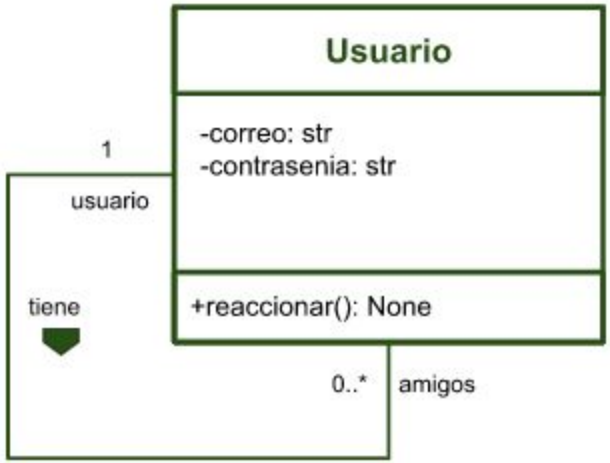
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 2: Creación de la clase Foto

a. Crear rectángulo que representa la clase

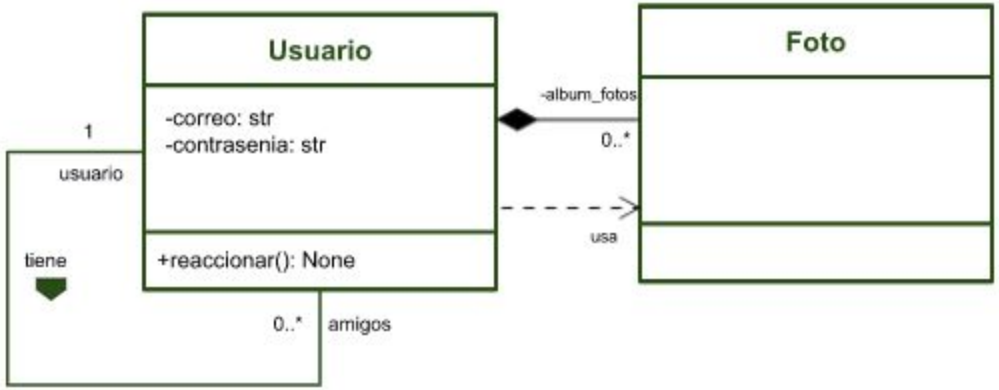


Escribiendo un diagrama de clases

Solución

Paso 2: Creación de la clase Foto

b. Crear relaciones



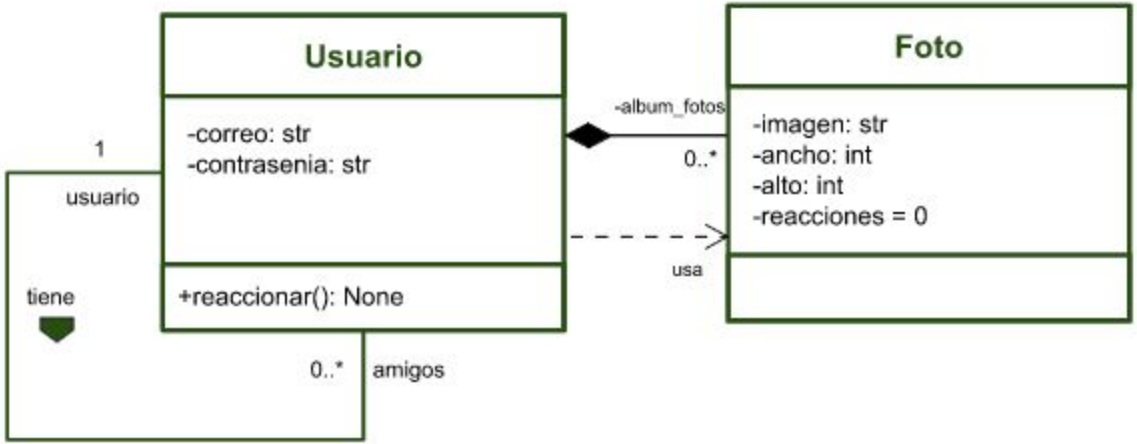
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 2: Creación de la clase Foto

c. Creación de atributos



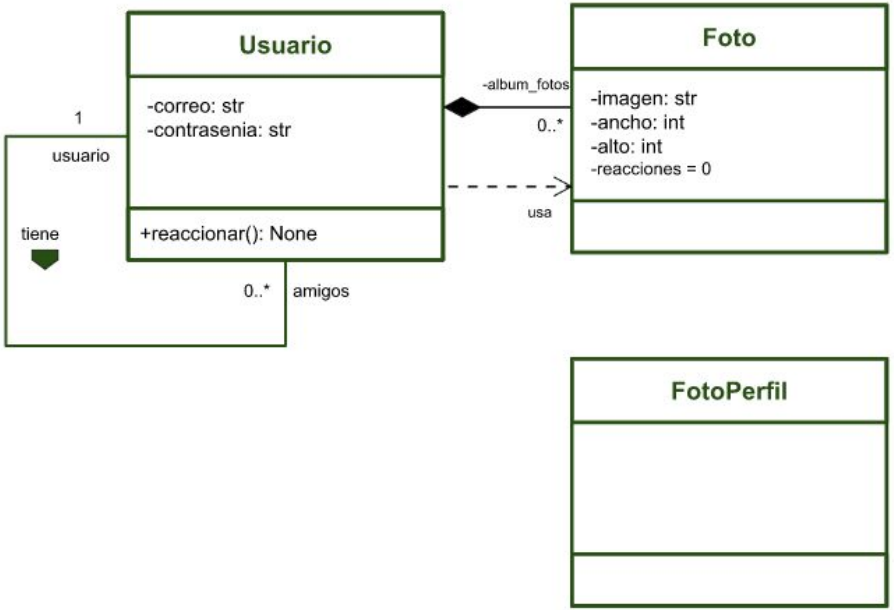
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 3: Creación de la clase FotoPerfil

a. Crear rectángulo que representa la clase



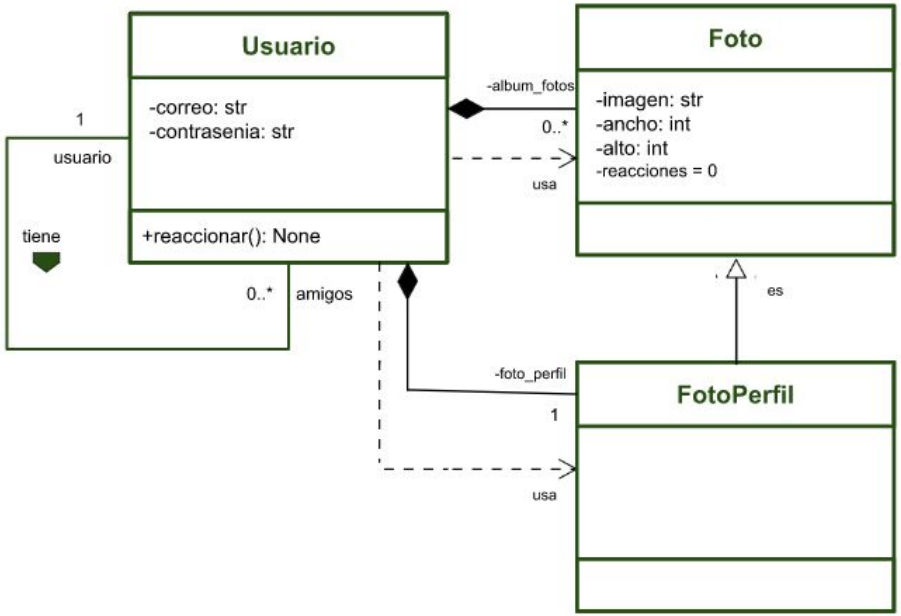
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 3: Creación de la clase FotoPerfil

b. Crear relaciones



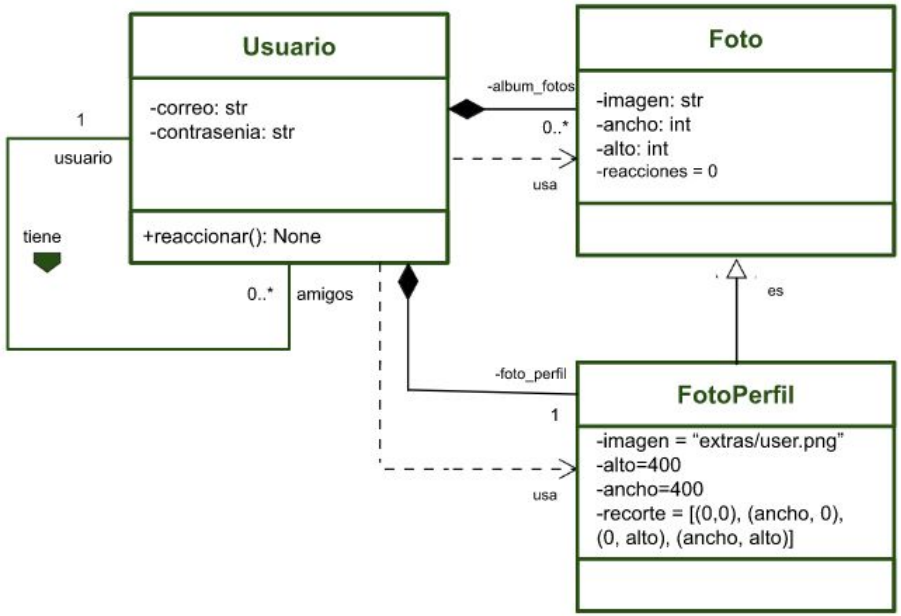
{desafío} latam_

Escribiendo un diagrama de clases

Solución

Paso 3: Creación de la clase FotoPerfil

c. Creación de atributos



{desafío} latam_

Ejercicio guiado

"Código a partir de un diagrama de clases"

{desafío} latam_

Código a partir de un diagrama de clases

Este ejercicio toma como base el diagrama de clases construido en el ejercicio guiado anterior.

Se debe definir todas las propiedades que se considere necesarias, aunque no es necesario implementar lógica adicional dentro de los métodos getter y setter, más allá de retornar el valor del atributo en el caso de los getter, y de asignar un nuevo valor (entregado por parámetro) a los atributos en el caso de los setter.

Código a partir de un diagrama de clases

Solución

Paso 1

Lo primero es identificar qué clases no requieren de la definición de otras para constituirse. En este caso, solo la clase Foto cumple con esa condición, ya que se puede ver en el diagrama que la clase Usuario usa otras clases, y la clase FotoPerfil hereda de otra. Por lo tanto, como primer paso, en un archivo foto.py, se define la clase Foto.

```
#archivo foto.py
class Foto():
```

Código a partir de un diagrama de clases

Solución

Paso 2

A continuación, dentro de la clase Foto, se define su constructor (no se indica en el diagrama que exista atributos de clase). Según el diagrama, solo reacciones tiene un valor de inicio, por lo que el constructor debe solicitar por parámetro los valores de los otros atributos. El diagrama también indica que los atributos son privados, por lo que se debe aplicar encapsulamiento de ellos.

```
def __init__(self, imagen: str, ancho: int, alto: int) -> None:
    self.__imagen = imagen
    self.__ancho = ancho
    self.__alto = alto
    self.__reacciones = 0
```

Código a partir de un diagrama de clases

Solución

Paso 3

Luego, en la misma clase, se define las propiedades con getters y setters para todos los atributos (implementación básica). Estos métodos no se encontraban explícitos dentro del diagrama de clases, pero se deduce por la indicación de privacidad y porque hay una clase que le hereda (y otra que la usa) que deben ser definidos.

```
@property
    def imagen(self) -> str:
        return self.__imagen
    @imagen.setter
    def imagen(self, imagen: str) -> None:
        self.__imagen = imagen
    @property
    def ancho(self) -> int:
        return self.__ancho
    @ancho.setter
    def ancho(self, ancho: int) -> None:
        self.__ancho = ancho
    @property
    def alto(self) -> int:
        return self.__alto
    @alto.setter
    def alto(self, alto: int) -> None:
        self.__alto = alto
    @property
    def reacciones(self) -> int:
        return self.__reacciones
    @reacciones.setter
    def reacciones(self, reacciones: int) -> None:
        self.__reacciones = reacciones
```

Código a partir de un diagrama de clases

Solución

Paso 4

La clase Foto no declara operaciones en el diagrama de clases, por lo que con el paso anterior se termina su construcción. Luego se debe identificar qué clases no dependen de otras (que no sean Foto) para su definición. Solo la clase FotoPerfil cumple con esta condición, por lo que en el mismo archivo, a continuación de la clase Foto, se debe definir la clase FotoPerfil (heredando de Foto).

```
class FotoPerfil(Foto):
```

Código a partir de un diagrama de clases

Solución

Paso 5

Luego, dentro de la clase Foto, se debe definir su constructor, ya que no se declara atributos de clase en el diagrama de clases. El constructor debe sobrescribirse, ya que los valores de imagen, ancho y alto no se deben solicitar por parámetro, sino que se asignan con los valores de inicio indicados en el diagrama de clases.

```
def __init__(self) -> None:
    super().__init__(
        "extras/user.png",
        400,
        400
    )
```

Código a partir de un diagrama de clases

Solución

Paso 6

Además, también dentro del constructor, se debe agregar el atributo recorte, con el valor indicado en el diagrama de clases.

```
self.__recorte = [
    (0, 0),
    (self.ancho, 0),
    (0, self.alto),
    (self.ancho, self.alto)
]
```

Código a partir de un diagrama de clases

Solución

Paso 7

La clase FotoPerfil no declara operaciones, y hereda todos los métodos de las propiedades desde su clase padre, por lo que con el paso anterior se termina la construcción de la clase FotoPerfil. Por lo tanto, lo siguiente a realizar es definir la única clase restante, Usuario. Esta clase se define en un archivo usuario.py, por lo que se debe importar en ella las clases FotoPerfil y Foto desde foto (el diagrama indica que son componentes de Usuario, por lo que se sabe que se utilizarán).

```
from foto import Foto, FotoPerfil
from typing import Union, List

class Usuario():
```

Código a partir de un diagrama de clases

Solución

Paso 8

La clase Usuario no declara atributos de clase, por lo que el siguiente paso es definir su constructor. En él, se deben declarar los atributos de instancia privados indicados en el diagrama de clases, solicitando además su valor por parámetro.

```
def __init__(self, correo: str, contraseña: str) -> None:
    self.__correo = correo
    self.__contraseña = contraseña
```

Código a partir de un diagrama de clases

Solución

Paso 9

A continuación, dentro del constructor, se debe además crear los atributos foto_perfil y album_fotos. Para el primer caso, el valor asignado corresponde a una instancia de la clase FotoPerfil. Para el segundo caso se asigna una lista vacía.

```
        self.__album_fotos = []
        self.__foto_perfil = FotoPerfil()
```

Código a partir de un diagrama de clases

Solución

Paso 10

Luego, si bien el diagrama no lo indica, por buena práctica se definen las propiedades con getters y setters para correo y contraseña.

```
@property
    def correo(self) -> str:
        return self.__correo
    @correo.setter
    def correo(self, correo: str) -> None:
        self.__correo = correo
    @property
    def contraseña(self) -> str:
        return self.__contraseña
    @contraseña.setter
    def contraseña(self, contraseña: str) -> None:
        self.__contraseña = contraseña
```

Código a partir de un diagrama de clases

Solución

Paso 11

En el caso de album_fotos y foto_perfil, se define sus getters, pero en lugar de setters se utiliza métodos, ya que los métodos setter de las propiedades solo permiten 1 parámetro adicional a la instancia (existe otras formas de manejar esto, pero en este caso se decidió simplemente definir métodos propios)

```
@property
def album_fotos(self) -> List[Foto]:
    return self.__album_fotos
def agregar_fotos_al_album(self, imagen: str, ancho: int, alto: int) -> None:
    self.__album_fotos.append(Foto(imagen, ancho, alto))
@property
def foto_perfil(self) -> Foto:
    return self.__foto_perfil
def actualizar_foto_perfil(self, imagen: str, ancho: int, alto: int) -> None:
    self.__foto_perfil.imagen = imagen
    self.__foto_perfil.ancho = ancho
    self.__foto_perfil.alto = alto
```

Código a partir de un diagrama de clases

Solución

Paso 12

Finalmente, se define el método declarado en las operaciones de la clase Usuario. El diagrama de clases no detalla la implementación; en este caso se incluye cómo sería la implementación del método (se debe añadir un import de Union), pero basándose solo en el diagrama de clases solo se podría definir su nombre.

```
    def reaccionar(self, foto: Union[Foto, FotoPerfil]) -> None:
        foto.reacciones += 1
```

¿Qué utilidad tiene diseñar un diagrama de clases?

{desafío} latam_

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales