UCSC

UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

Taller de programación II - IN1071C

Diseño Orientado a Objetos

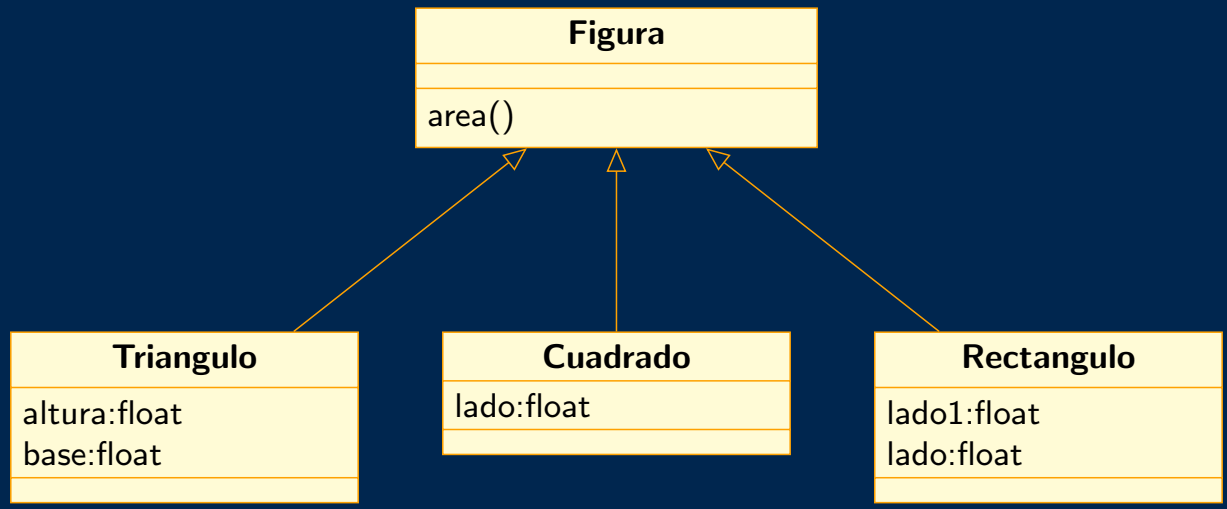
Diego Maldonado

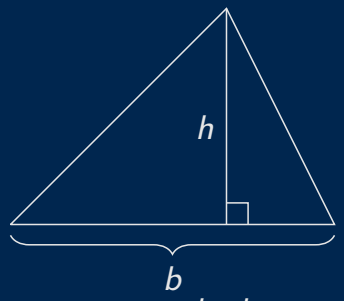
Facultad de Ingeniería
Departamento de Ingeniería Informática

5 de diciembre de 2022

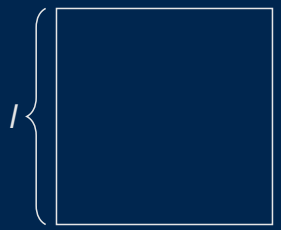
Contenido

En DOO la abstracción se ve en la herencia y polimorfismo, los padres son siempre más abstractos que sus hijos.

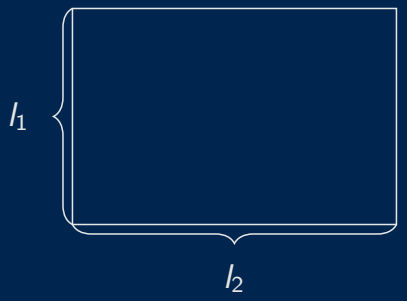




Área$= \frac{b \cdot h}{2}$



Área$= l^2$



Área$= l_1 \cdot l_2$

Definición

Una clase abstracta es una clase no representa a objetos directamente del sistema que se está modelando.

Ejemplo

En el caso anterior, `Figura` es una clase abstracta, pues no es posible definirle un área. Sin embargo, las clases `Triangulo`, `Cuadrado` y `Rectangulo` heredan el método `área` y si lo pueden usar.

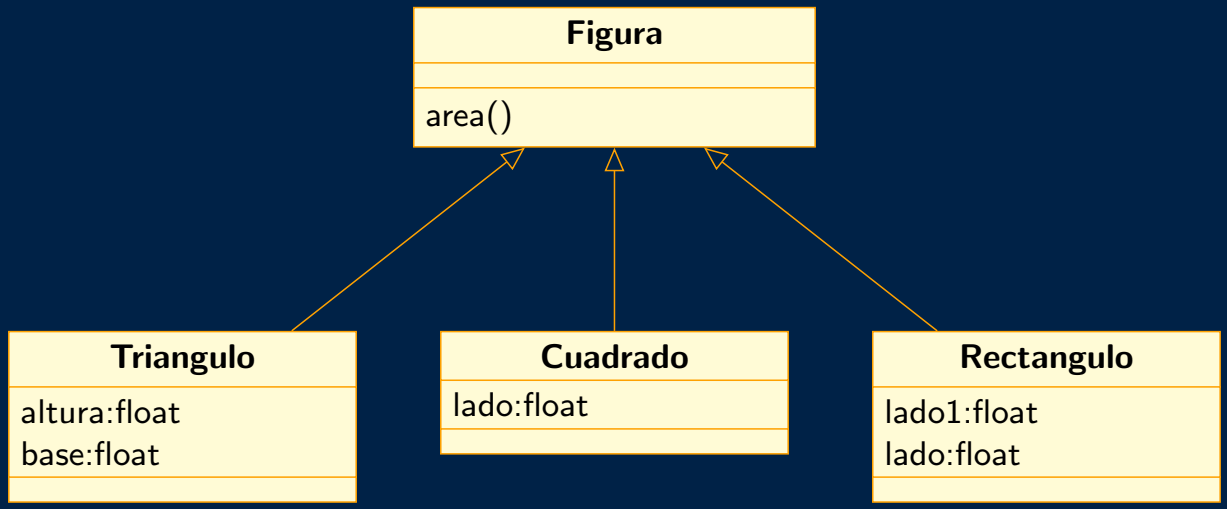
En `Python` se implementa utilizado la biblioteca estándar `abc` (Abstract Base Classes) de la siguiente forma:

```
1  from abc import ABCMeta, abstractmethod
2
3  class ClaseAbstracta(metaclass=ABCMeta):
4    @abstractmethod   # indica que el metodo es abstracto
5    def metodoAbstracto(self):
6      pass
7
```

Nota

El objetivo es forzar a que las clases hijas de la clase abstracta definan los métodos abstractos.

Ejercicio



1. Implemente el UML anterior en `Python`.
2. Implemente la suma de Figuras, que entrega como resultado la suma de sus áreas.
3. Sume un triángulo con un cuadrado.

La encapsulamiento en POO es la acción de restringir la mayor cantidad de atributos y métodos posibles de los objetos.
Este ocultamiento puede tener tres alcances: Privado, Protegido y Público

- Privado ($-$). Sólo puede tener acceso a un atributo privado la clase en la que está definido. En forma similar, un método privado sólo puede ser llamado por la clase en la que está definida. Las subclases u otras clases no pueden tener acceso a los atributos y operaciones privados.
- Protegido ($\#$). La clase en la que están definidos y cualquier descendiente de la clase pueden tener acceso a un atributo u método protegidos.
- Público ($+$). Cualquier clase puede tener acceso a un atributo u método público.

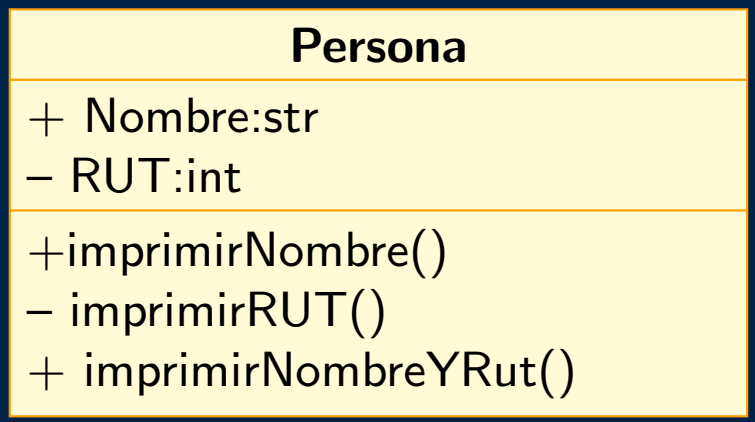
En UML se denota con los símbolos $-$, $\#$ y $+$ antes del nombre o método respectivo.

En Python todo es público, por lo que estudiaremos como hacerlos privados, basta añadir dos guiones bajos antes del nombrel del método o atributo (__).

```
1  class Clase():
2    def __init__(self,a,b):
3      self.atributoPublico    = a
4      self.__atributoPrivado = b
5    def metodoPublico(self):
6      pass
7    def __metodoPrivado(self):
8      pass
9
```

Ejercicio

Implemente en Python la siguiente clase:

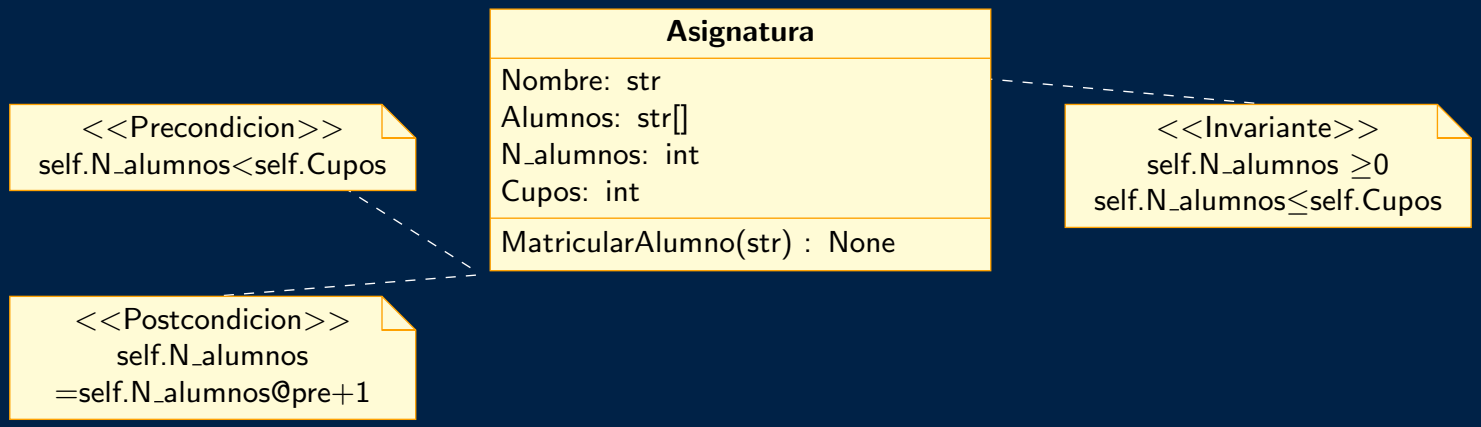


Los contratos son restricciones sobre una clase que permiten que el llamador y el llamado compartan las mismas suposiciones acerca de la clase. Un contrato especifica restricciones que debe satisfacer el llamador antes de usar la clase, así como las restricciones que asegura cumplir el llamado cuando se le usa. Los contratos incluyen tres tipos de restricciones:

- Un invariante es un predicado que siempre es cierto para todas las instancias de una clase. Los invariantes son restricciones asociadas con clases. Los invariantes se usan para especificar restricciones de consistencia entre atributos de clase.
- Una precondición es un predicado que debe ser cierto antes de que se llame a un método. Las precondiciones están asociadas con un método específico. Las precondiciones se usan para especificar restricciones que debe satisfacer el llamador antes de llamar a un método.
- Una poscondición es un predicado que debe ser cierto después de que se llama a una método. Las poscondiciones están asociadas con una método específico. Las poscondiciones se usan para especificar restricciones que el objeto debe asegurar después de la invocación del método.

En UML las restricciones se expresan usando OCL. OCL es un lenguaje que permite que se especifiquen de manera formal las restricciones en elementos únicos del modelo o en grupos de elementos del modelo.
Una restricción puede mostrarse como una nota asociada al elemento UML restringido mediante una relación de dependencia. Una restricción puede expresarse en lenguaje natural o en un lenguaje formal, como OCL.

Ejemplo



Dado que el diseño por contrato requiere el cumplimiento de ciertas condiciones, debemos detectar cuando el contrato no se está cumpliendo. Para esto, Python tiene una primera herramienta: las Aserciones[1].
La sintaxis es:

```
1     assert booleano, string
2
```

Si el booleano es falso entrega el mensaje de error, en otro caso no hace nada.

Nota

Las aserciones son una forma simple de verificar el contrato, pero no se recomienda usar salvo para la fase de desarrollo. Hay métodos más adecuados para manejar errores.

Ejemplo

Para validar si un texto es un correo válido (contiene el carácter '@') se puede utilizar la siguiente aserción:

```
1     correo = 'dmaldonado@ucsc.cl'
2     assert '@' in correo, 'Correo no valido'
3
```

[1] Sinónimo: Aseveraciones, afirmaciones

Ejercicio

Construiremos la clase Grafo en `Python`. Para esto:

1. Determine las condiciones que debe satisfacer el conjunto de vértices y aristas.
2. Cree métodos para verificar estas condiciones.
3. Cree los métodos añadir vértice y añadir arista.

Una solución que nos podría tentar al momento de manejar errores, sería entregar alguna salida que indique error (-1, False, "Hubo un error"). El inconveniente de esto es que los contratos exigen que las salidas sean de un cierto tipo.

```
1  def division(a,b):
2      if b==0:
3          print("Cuidado!!! No se puede dividir por 0")
4      else:
5          return a/b
6
7  5+division(3,0)
```

Entrega:

```
Cuidado!!! No se puede dividir por 0
---------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [1], line 7
                4     else:
                5         return a/b
----> 7 5+division(3,0)

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

Ejercicio

¿Sería mejor cambiar el mensaje de error por una salida?
¿Cuál es el contrato de la función "+"?

En Python, la palabra clave utilizada para generar una excepción es raise, y los objetos desencadenados deben heredar obligatoriamente de la clase Exception.

```
1  class MiExcepcion(Exception):
2    pass
3  def test():
4    print("Antes de producirse")
5    raise MiExcepcion()
6    print("Despues de producirse")
```

Entrega:

```
Antes de producirse
---------------------------------------------------------------------
MiExcepcion                               Traceback (most recent call last)
Cell In [1], line 7
                5       raise MiExcepcion()
                6       print("Despues de producirse")
----> 7 test()
Cell In [1], line 5, in test()
                3 def test():
                4       print("Antes de producirse")
----> 5         raise MiExcepcion()
                6       print("Despues de producirse")
MiExcepcion:
```

Aquí raise detiene la ejecución del programa y propaga un error del tipo MiExcepcion.

Entrega:

```
1  def test3():
2      raise MiExcepcion()
3  def test2():
4      test3()
5  def test1():
6      test2()
7  def test0():
8      test1()
9  test0()
```

```
---------------------------------------
MiExcepcion                Traceback (most recent
    call last)
Cell In [6], line 9
                7 def test0():
                8     test1()
----> 9 test0()

Cell In [6], line 8, in test0()
                7 def test0():
----> 8     test1()

Cell In [6], line 6, in test1()
                5 def test1():
----> 6     test2()

Cell In [6], line 4, in test2()
                3 def test2():
----> 4     test3()

Cell In [6], line 2, in test3()
                1 def test3():
----> 2     raise MiExcepcion()

MiExcepcion:
```

Nota

Los llamados a las funciones se almacenan en la Pila de llamadas y y define el orden en que aparecen los errores.

Para manejar los errores se usan los comandos `try`, que "intenta" ejecutar un código y `except`, que en caso de ocurrir una excepción ejecuta otro código.

```
1  def test3():
2      raise MiExcepcion()
3  def test2():
4      test3()
5  def test1():
6      try:
7          test2()
8      except:
9          print("Excepcion capturada")
10 def test0():
11     test1()
12 test0()
```

Entrega:

```
Excepcion capturada
```

Nota

No se generaron errores de Python.

Es posible especificar el tipo de error que queremos capturar[1]:

```
1  try:
2      raise KeyError()
3  except KeyError:
4      print("Capturado KeyError")
5
6  try:
7      raise NotImplementedError()
8  except KeyError:
9      print("Capturado KeyError")
```

Entrega:

```
Capturado KeyError
---------------------------------
NotImplementedError   Traceback (
    most recent call last)
Cell In [8], line 7
    4      print("Capturado
          KeyError")
    6 try:
----> 7      raise
     NotImplementedError()
    8 except KeyError:
    9      print("Capturado
          KeyError")

NotImplementedError:
```

Nota

Con `except:` al final de una captura de excepciones se capturan las excepciones de cualquier tipo.

[1] En `https://docs.python.org/es/3/library/exceptions.html` se encuentran los tipos de excepciones de Python

El comando `finally`: ejecuta el código que le sigue haya capturado una excepción o no. Es útil para cerra documentos abiertos y limpiar datos.

```
1 try:
2   archivo = open("test.txt", 'r')
3   archivo.read()
4 finally:
5   archivo.close()
```

Ejercicio

1. Genere las clases audio con atributo privado string `direccion` y listaDeReproduccion con atributo privado lista, una lista de audios. Además cree métodos para entregar estos atributos.
2. Cree los errores ArchivoNoAudio, NoEsMP3.
3. Cree métodos para ingresas direcciones a los Audios y Audios a las listaDeReproduccion.
4. Debe aparecer el error NoEsMP3 si no se ingresa un string terminado en ".mp3" en Audio.
5. Debe aparecer el error ArchivoNoAudio si no se ingresa una lista de Audios en listaDeReproduccion.
6. Cree el método AgregarAudioDesdeDir en listaDeReproduccion. Debe recibir como entrada un texto y si no se puede crear un Audio con ese texto debe entregar un mensaje que diga "Archivo con formato incorrecto".
7. Cree el método Reproducir en listaDeReproduccion. Debe intentar leer el texto en cada archivo .mp3 e imprimirlo si existe, en caso de fallar debe cerrar los archivos.

Use `isinstance()` para verificar si las variables son del tipo de datos correctos.

El enmascaramiento de errores es una mala práctica en programación, en la cual se esconden errores en vez de desplegarlos. En general, consiste en capturar una excepción y no tomar ninguna medida al respecto.

Ejemplo

```
1 try:
2   ecuacion_complicada()
3 except ZeroDivisionError:
4   pass
```

Aquí se oculta que hubo una operación que no está bien definida.

Nota

Las excepciones están para señalar errores, inconsistencias o valores que van más allá del alcance de la definición de los métodos (fuera del contrato). Lanzar una excepción equivale a indicarle al programa que hay un problema en los datos que se deben manejar y que el riesgo de errores es alto.

Para propagar una excepción, debe usar `raise` sin argumento dentro de una cláusula.
Una forma correcta de manejar la división por cero es la siguiente:

```
1 try:
2   1/0
3 except ZeroDivisionError:
4   print("Se ha realizado una division por cero")
5   raise
6
```

Entrega:

```
Se ha realizado una division por cero
---------------------------------------------------------------
ZeroDivisionError                Traceback (most recent call last)
Cell In [6], line 2
                1 try:
----> 2         1/0
                3 except ZeroDivisionError:
                4       print("Se ha realizado una division por
                      cero")

ZeroDivisionError: division by zero
```

- Las excepciones personalizadas se crean como clases hijas de la clase `Exception` o de sus hijas.
- Pueden recibir argumentos para enriquecer la recolección de información que generó el error.
- También se puede sobrecargar `__str__` para desplegar mensajes de error.

Ejemplo

```
1  class PotenciaZeroException(Exception):
2      def __init__(self, value):
3          # Este atributo permite tener una informacion adicional.
4          self.value = value
5      def __str__(self):
6      # Mensaje de error que  mostrara si la excepcion se captura.
7          return "Error: valor {} elevado a 0".format(self.value)
8  def potencia(a, b):
9      if b == 0:
10         raise PotenciaZeroException(a)
11     return a ** b
12 try:
13     print(potencia(5, 2))
14     print(potencia(5, 0))
15 except PotenciaZeroException as e:
16     print(e)
```

Nota

En este caso, puede reemplazar “Exception” por “ArithmeticError”.

Las enumeraciones son formas en las que podemos definir una serie de constantes relacionadas entre si en un modelo. Una forma simple de hacer esto en Python es mediante atributos de clase.

Ejemplo

```
1  class semana():
2      lu=0
3      ma=1
4      mi=2
5      ju=3
6      vi=4
7      sa=5
8      do=6
9  hoy = semana.lu
10 print(f'Hoy es miercoles? \nResuesta: {semana.mi==hoy}')
```

Nota

Note que el llamado es a la clase y no a un objeto particular.

Este tipo de implementación no es recomendable, pues Python tiene una biblioteca para este propósito: enum.

Ejemplo

```
1  from enum import Enum
2  class semanaEnum(Enum):
3      lu=0
4      ma=1
5      mi=2
6      ju=3
7      vi=4
8      sa=5
9      do=6
10 meses = Enum(
11   value='meses',
12   names=('ENE FEB MAR ABR MAY JUN JUL AGO SEP OCT NOV DIC'))
```

Nota

Note que el llamado es a la clase y no a un objeto particular.

Este tipo de implementación no es recomendable, pues Python tiene una biblioteca para este propósito: enum.

Ejercicio (en grupos, los del proyecto)

Modele una baraja de cartas inglesas, una mano y un contador de puntos:

1. Genere las clases enumeradas `pinta`, `numero` y `reverso`.
2. Genere la clase enumerada `carta`, con su pinta, número, reverso y puntos.
3. Cree la clase `mano`, debe tener un método para contar los puntos de la mano.

Evaluaciones:

Test 2:	12/12/2022
Certamen 2:	19/12/2022
Informe:	23/12/2022
Presentación:	26/12/2022
Certamen 3:	28/12/2022

- Debe implementar las dos primeras partidas del juego, es decir, dos trios y un trio y una escala.
- Deben poder jugar dos jugadores.
- La interacción de los jugadores con el sistema debe ser por la consola de comandos (terminal).
- Las comunicación con el cliente debe ser incluyendo al menos un escenario asociado al motivo de la comunicación.
- El punto 3.5 "Modelo de objetos" del DAR constara de: 3.5.1 Diagrama de Clases, 3.5.2 Diagrama de casos de uso y 3.5.3 Diagrama de secuencia.
- El desarrollo del software debe considerar **TODOS** los contenidos vistos en clases. Lea nuevamente el materia disponible en EV@ para estar seguro de considerar todos los aspectos de la obtención de requerimientos, análisis y desarrollo vistos en el curso.
- El desarrollo debe cumplir con las expectativas del cliente. Es deber de ustedes averiguarlas.
- El cliente no estará disponible para consultas asociadas al proyecto a partir del 21 de diciembre después de clases.
- Debe incluir un anexo con un listing del código realizado.
- Deben actualizar el DAR con las correcciones realizadas.
- Recuerden, primero hacer los diagramas de clases, casos de uso y secuencia, después programar.

Evaluación:

- Informe
  - Cumplimiento y correctitud del DAR.
- Presentación
  - Introducción
  - Sistema propuesto
  - Consideraciones finales
  - Presentación y claridad de la exposición
  - Calidad de las respuestas
  - Tiempo (No más de 15 minutos)
- Programa
  - Cumplimiento de los requerimientos
  - Robustez
  - Coherencia con el DAR, en particular con los diagramas

Nota: promedio(Informe, Presentación, Programa)