

Ayudantía Examen

Parte 1



Estructuras de datos

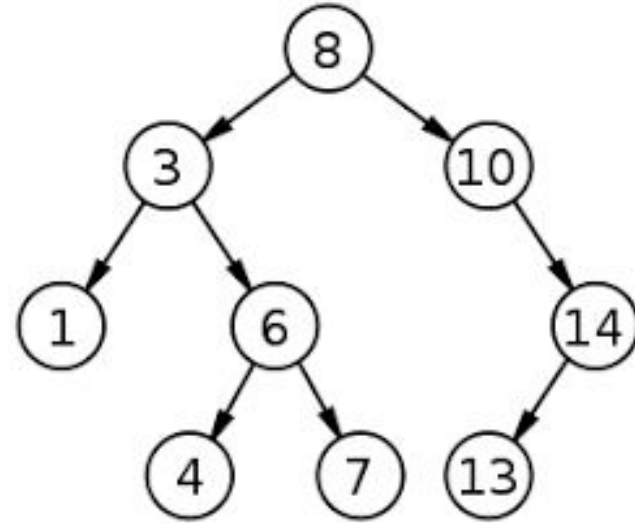
- Listas ligadas
- Árboles binarios
- Grafos
- Búsqueda de nodos
- DFS y BFS

¿Qué recordamos?

- Listas ligadas ?
- Árboles binarios?
- Grafos?

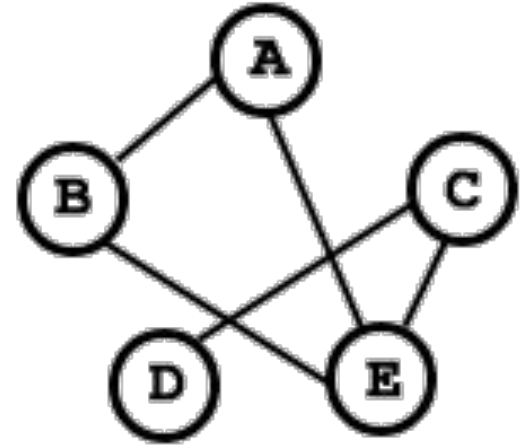
Árboles

Un árbol es un conjunto de nodos que sigue una estructura **jerárquica**. A diferencia de las estructuras basadas en secuencias (lineales) como listas, colas y pilas, en los árboles los elementos están ordenados de acuerdo a una relación *padre-hijo* entre los nodos.



Grafos

Un grafo es un conjunto no vacío de nodos y las relaciones entre estos nodos. En teoría de grafos, a los nodos se les llama **vértices**; a las relaciones entre ellos, **aristas**.



Búsqueda:

- Recursivo vs Iterativo**
- DFS vs BFS**

¡ Resolvamos un ejercicio !

El árbol de prefijos (2017-2)

Paths

- ¿Qué es un *path*?
- *Path* Absoluto
- *Path* Relativo
- ¿Cómo lo hago en *Python*?

¿Qué es un *path*?

- Es la manera como se representa una ubicación en un computador (carpeta, archivo, etc.)
- La forma de escribirlos cambia según los sistemas operativos.

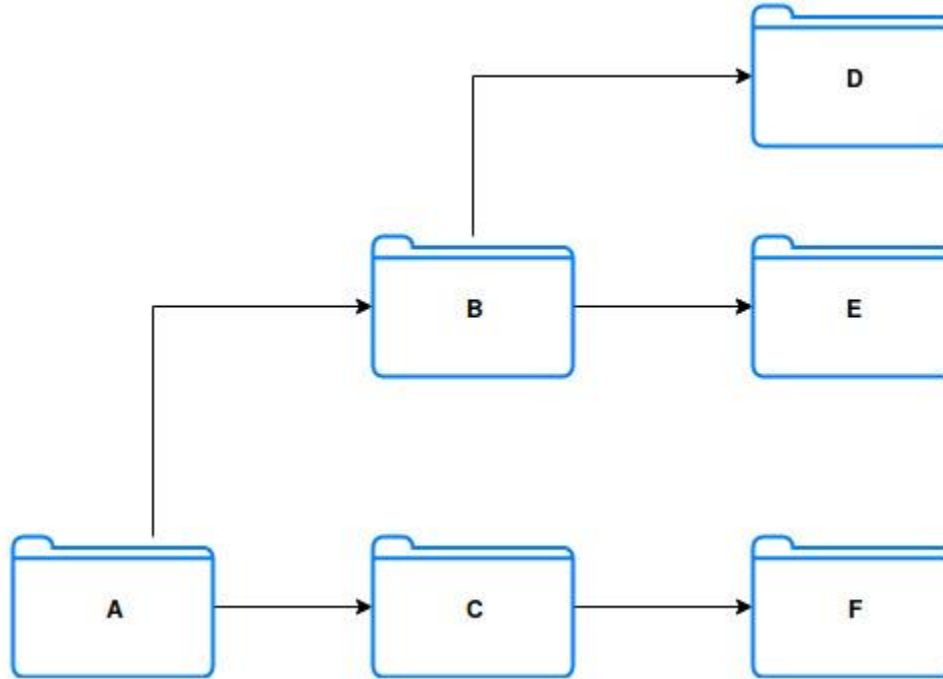
Path Absoluto

- Es el “camino” desde la carpeta raíz del sistema hasta un archivo o carpeta específica.
- Es distinto en todos los computadores, incluso en los distintos usuarios dentro de un sistema operativo.
- Dos *paths absolutos*:
 - C:\Windows\Usuario\Documentos\Examen_Secreto.pdf
 - /home/Usuario/Documentos/Examen_Secreto.pdf

Path Relativo

- Es el “camino” desde la carpeta en la que estamos “parados” hasta un archivo o carpeta específica.
- En distintos computadores si se tiene la misma estructura dentro de la carpeta en la que estamos el *path* será el mismo
- Tomando el ejemplo anterior, si estamos en la carpeta Usuario el *path* relativo sería:
 - Documentos/Examen_Secreto.pdf

Problemos algunos *paths*



¿Cómo lo hago en *Python*?

```
In [1]: import os
```

```
In [2]: # Path absoluto hasta la carpeta en la que me encuentro actualmente  
os.getcwd()
```

```
Out[2]: '/home/benjamin/Desktop/Ayudantia_Examen'
```

```
In [3]: # Todas las carpetas y archivos en dentro de nuestro directorio actual  
os.listdir()
```

```
Out[3]: ['AyudantiaExamen.ipynb', '.ipynb_checkpoints', 'C', 'B', 'A']
```

```
In [4]: # Todas las carpetas y archivos dentro de "A"  
print("Con path relativo: " + str(os.listdir("A")))  
  
absolute_to_A = os.getcwd() + os.path.sep + "A"  
print("Path absoluto: " + absolute_to_A)  
print("Con path absoluto: " + str(os.listdir(absolute_to_A)))
```

```
Con path relativo: ['A3', 'A2', 'A1']  
Path absoluto: /home/benjamin/Desktop/Ayudantia_Examen/A  
Con path absoluto: ['A3', 'A2', 'A1']
```

Strings

- Características principales
- *format* vs. *f-strings*

Características de los *strings*

- Son inmutables, por lo tanto, cada vez que realizamos una operación sobre ellos, se crea uno nuevo.
- Tienen diversos métodos que ayudan a su manipulación, algunos de estos son: **split**, **strip**, **replace**, entre otros.

```
In [17]: test = 'Esto es un string'
test_2 = test.replace('s', '~')
print(test)
print(test_2)
```

```
test[0] = 'A'
```

```
Esto es un string
E~to e~ un ~tring
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-17-d48517be5cbc> in <module>
      4 print(test_2)
      5
----> 6 test[0] = 'A'
```

```
TypeError: 'str' object does not support item assignment
```

format vs. f-strings

```
In [18]: curso = 'IIC2233'
f_str = f'Me encanta {curso}!'
format_str = 'Me encanta {}'.format(curso)
print('Este es un f-string --> ', f_str)
print('Este es un string formateado con format --> ', format_str)
```

Este es un f-string --> Me encanta IIC2233!

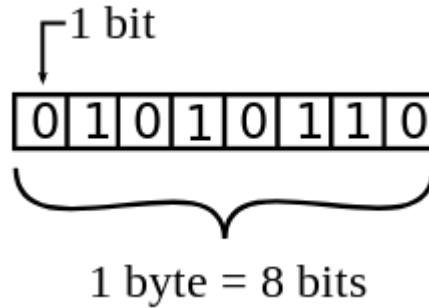
Este es un string formateado con format --> Me encanta IIC2233!

Bytes

- ¿Qué es un bit y un byte?
- Bytearray
- Ejemplos

bits y bytes

- **bit:** Unidad básica de información, solo puede tener dos valores, 0 y 1.
- **byte:** Conjunto de 8 bits, por lo tanto un byte puede tener 256 valores distintos (0 a 255).



```
In [11]: binario = "{0:b}".format(6)
print('Esto --> {} es 6 en binario'.format(binario))

byte = binario.zfill(8)
print('Estos --> {} son 8 bits (un byte) con valor decimal 6'.format(byte))
```

```
Esto --> 110 es 6 en binario
Estos --> 00000110 son 8 bits (un byte) con valor decimal 6
```

```
In [6]: normal_bytes = b'estos son bytes'
print(type(normal_bytes))

normal_bytes[0] = "E"
```

```
<class 'bytes'>
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-6-339c7e0f3324> in <module>
      2 print(type(normal_bytes))
      3
----> 4 normal_bytes[0] = "E"
```

```
TypeError: 'bytes' object does not support item assignment
```

Bytearray

- Es un arreglo de *bytes*, estructura especialmente diseñada para trabajar con estos.
- Tiene una interfaz similar a las listas, es decir, tiene métodos parecidos a esa EDD

```
In [15]: b_array = bytearray(normal_bytes)
          print(type(b_array))

          b_array[0] = ord('E')
          print(b_array)

          <class 'bytearray'>
          bytearray(b'Estos son bytes')
```

Un pequeño ejercicio

Transforma los caracteres del string "wubbalubbadubdub" a ASCII, luego a bytes, invierte los bytes e imprime el string invertido.

```
In [24]: my_str = "wubbalubbadubdub"

ascii_val = [ord(c) for c in my_str]
print('Valores ASCII --> ', ascii_val)

ascii_bytearray = bytearray(ascii_val)
print('Nuestros bytes --> ', ascii_bytearray)

ascii_bytearray = ascii_bytearray[::-1]
decoded_str = ascii_bytearray.decode('ascii')
print('Nuestro string invertido --> ', decoded_str)
```

```
Valores ASCII --> [119, 117, 98, 98, 97, 108, 117, 98, 98, 97, 100, 117, 98, 100, 117, 98]
Nuestros bytes --> bytearray(b'wubbalubbadubdub')
Nuestro string invertido --> budbudabbulabbuw
```

Excepciones y Testing

- ¿Qué es una excepción?
- Tipos de Excepciones
- Manejo de Excepciones
- Testing

¿Qué es una excepción?

- Un evento que ocurre durante la ejecución de un programa e interrumpe el flujo de este.

Tipos de excepciones

- Existen distintos tipos de excepciones, lo que permite personalizar el comportamiento del programa al encontrarse con cada una.

Manejo de Excepciones

- try
- except
- else
- finally

Lanzar Excepciones

- Raise
- Excepciones personalizadas

Testing

- unittest
 - TestCase
 - assert
 - setUp y tearDown

Indique el output

```
def foo(qux):
    def bar(func):
        def baz(*args):
            try:
                print("Morty")
                func(*args)
                return args
            except qux:
                print("Town")
                return args[0]
            else:
                print("Locos")
                return None
            finally:
                print(":)")
        return baz
    return bar

class FunError(ZeroDivisionError): pass

@foo(ZeroDivisionError)
def double_fun(*args):
    raise FunError("Doble diversión!")

try:
    r = double_fun("hola")
except FunError:
    r = "Evil Morty"

print(r)
```

¿Cuándo se pasaría el test?

```
from package import tu_lista
import unittest

class Test(unittest.TestCase):

    def setUp(self):
        self.mi_lista = [3, 1, 4, 1]

    def es_menor(self, a, b):
        return self.assertLess(a, b)

    def test_largo(self):
        self.es_menor(len(self.mi_lista), len(self.mi_lista + tu_lista))
```

Ayudantía Examen

Parte 2



Threading

- ¿Qué es un Thread?
- Principales propiedades
- Desventajas de usar Threads
- Ejercicio(s)

¿Qué es y para qué sirve un Thread?

- Un thread es una secuencia muy pequeña de tareas que puede ser ejecutada por un sistema operativo, es decir, una sección de código.
- Utilizando varios threads, podemos ejecutar varias secciones de código en “paralelo”.
- Ejemplos:
 - Varios enemigos que se mueven simultáneamente en un juego.
 - Escuchar música y trabajar en un documento al mismo tiempo.
 - Las interfaces gráficas, en general, son un thread distinto al programa principal.

Propiedades y sintaxis

- `threading.Thread`
- `target` v/s `herencia`
 - `args, kwargs`
 - `run`
- `run` v/s `start`
- `threading.Timer`
 - `cancel`

Desventajas de usar threads

Sincronización

Soluciones para sincronización y concurrencia

- `thread.is_alive()`
- Daemon Threads
- Join
- Señales
 - wait, set, clear
- Locks
- Deadlocks

```
import threading

N = 100000
T = 2
entradas = [i for i in range(1, N + 1)]

class Vendedor:

    def __init__(self):
        self.siguiente = 0

    def entregar_ticket(self):
        n = self.siguiente
        self.siguiente += 1
        if n >= N:
            return -1
        entradas[n] = 0
        return n

class Cliente(threading.Thread):

    def __init__(self, id_, vendedor):
        super().__init__()
        self.id_ = id_
        self.vendedor = vendedor

    def run(self):
        while self.vendedor.entregar_ticket() != -1:
            pass

if __name__ == "__main__":
    vendedor = Vendedor()
    clientes = [Cliente(i, vendedor) for i in range(T)]
    for i in range(T):
        clientes[i].start()
    for i in range(T):
        clientes[i].join()

    print(sum(entradas))
```

```
import threading
import time

def thread_f(lock1, lock2, name):
    with lock1:
        print('Primera parte {}'.format(name))
        time.sleep(10)
    with lock2:
        print('Segunda parte {}'.format(name))
        time.sleep(5)

a_lock = threading.Lock()
b_lock = threading.Lock()

t1 = threading.Thread(target=thread_f, args=(a_lock, b_lock, "thread - 1"))
t2 = threading.Thread(target=thread_f, args=(b_lock, a_lock, "thread - 2"))

t1.start()
t2.start()
```

```
import threading

def thread_f(lock1, lock2, name):
    for i in range(5):
        with lock1:
            with lock2:
                print(name)

a_lock = threading.Lock()
b_lock = threading.Lock()

t1 = threading.Thread(target=thread_f, args=(a_lock, b_lock, "thread - 1"))
t2 = threading.Thread(target=thread_f, args=(b_lock, a_lock, "thread - 2"))

t1.start()
t2.start()
```

Funcional

```
def foo(n):  
    return reduce(lambda x, y: x + y, map(lambda z: int(z)**int(z), str(n)))
```

Funcional

```
def foo(n):  
    return reduce(lambda x, y: x + y, map(lambda z: int(z)**int(z), str(n)))
```

¿Qué hace la función foo?

Funcional

```
def foo(n):  
    return reduce(lambda x, y: x + y, map(lambda z: int(z)**int(z), str(n)))
```

Indique el output de

```
print(foo(123))
```

Funcional

```
def bar(k, func):  
    return lambda n: reduce(lambda x, y: func(x), range(k), n)
```

¿Qué hace la función bar?

Funcional

```
def foo(n):  
    return reduce(lambda x, y: x + y, map(lambda z: int(z)**int(z), str(n)))  
  
def bar(k, func):  
    return lambda n: reduce(lambda x, y: func(x), range(k), n)
```

Indique el output de

```
print(bar(1, foo)(23))  
print(bar(2, foo)(1234))  
print(bar(5, foo)(3435)) # Bonus 0,5p
```

Funcional

Indique el output de

```
ite1 = [3, 1, 4, 1, 5, 9, 2, 6] # pi
ite2 = [2, 7, 1, 8, 2, 8, 2]    # e
ite3 = [1, 4, 1, 4, 2, 1]       # raíz de 2

resultado = {x[1] * 2 for x in filter(lambda x: x[0] % 2, zip(ite1, ite2, ite3))}
print(resultado)
```

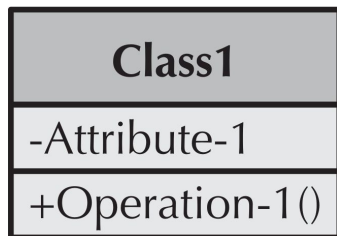
Modelación

UML

- Diagrama de Clases
- Composición
- Agregación
- Herencia
- Multi Herencia
- *Deadly Diamond of Death*

Modelación

Diagrama de clases



Privacidad

Pro tip

Table 3. Visibility Options on UML Class Diagrams

Visibility	Symbol	Accessible to
Public	+	All objects within your system
Protected	#	Instances of the implementing class and its subclasses
Private	-	Instances of the implementing class
Package	~	Instances of classes within the same package

Modelación

Types & Multiplicity

Class2

+ attr1: type
+ attrs2: type[]

Table 6. UML Multiplicity Indicators

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only n (where $n > 1$)
*	Many
0.. n	Zero to n (where $n > 1$)
1.. n	One to n (where $n > 1$)
n .. m	Where n and m both > 1
n ..*	n or more, where $n > 1$

Properties

Pro tip

Class3

+ <<get/set>> prop1: type

Modelación

Siempre hacia la flecha..

Herencia




Hereda de..

Agregación

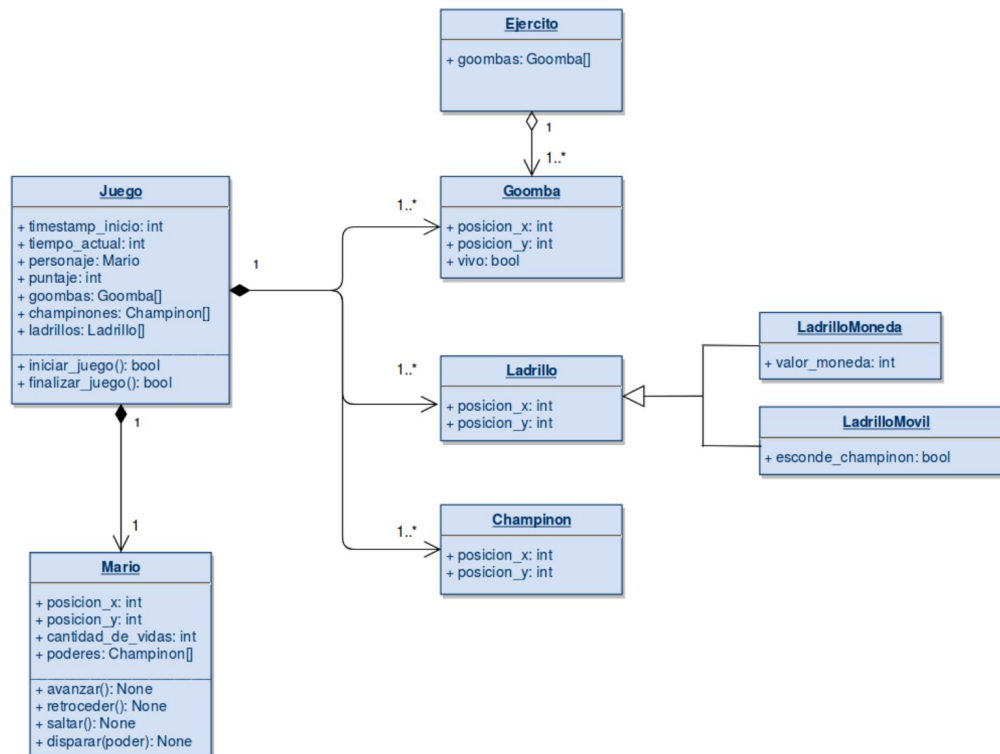
Agregado por..

Composición

Compuesto por..

A generalization: <ul style="list-style-type: none">• Represents a-kind-of relationship between multiple classes.	
An aggregation: <ul style="list-style-type: none">• Represents a logical a-part-of relationship between multiple classes or a class and itself.• Is a special form of an association.	
A composition: <ul style="list-style-type: none">• Represents a physical a-part-of relationship between multiple classes or a class and itself.• Is a special form of an association.	

Ejemplo Clases



Ejercicio Pag 1.

Llegando a fin de semestre, notas que tienes una manía con la programación, tus amigos y familiares te lo dicen, pero tú no estás de acuerdo. Para demostrarles lo contrario decides ir a la clínica *DCClinic*, hacerte todos los exámenes correspondientes, y así demostrar que están equivocados. Sin embargo al llegar y entregar tus datos, te das cuenta, que el software utilizado por la institución es de muy mala calidad. Naturalmente, tu piensas en tus habilidades (no manías) de programación y decides modelar un nuevo software para la DCClinic.

La clínica cuenta con una serie de doctores, todos personas común y corrientes. Todas las personas cuentan con un número entero único para ser identificados (comúnmente conocido como **rut**), un género (masculino, femenino o sin especificar), nombre, apellido y fecha de nacimiento. Naturalmente, desearías que el programa te permita calcular la edad en todo momento, sin tener que hacer el cálculo manualmente.

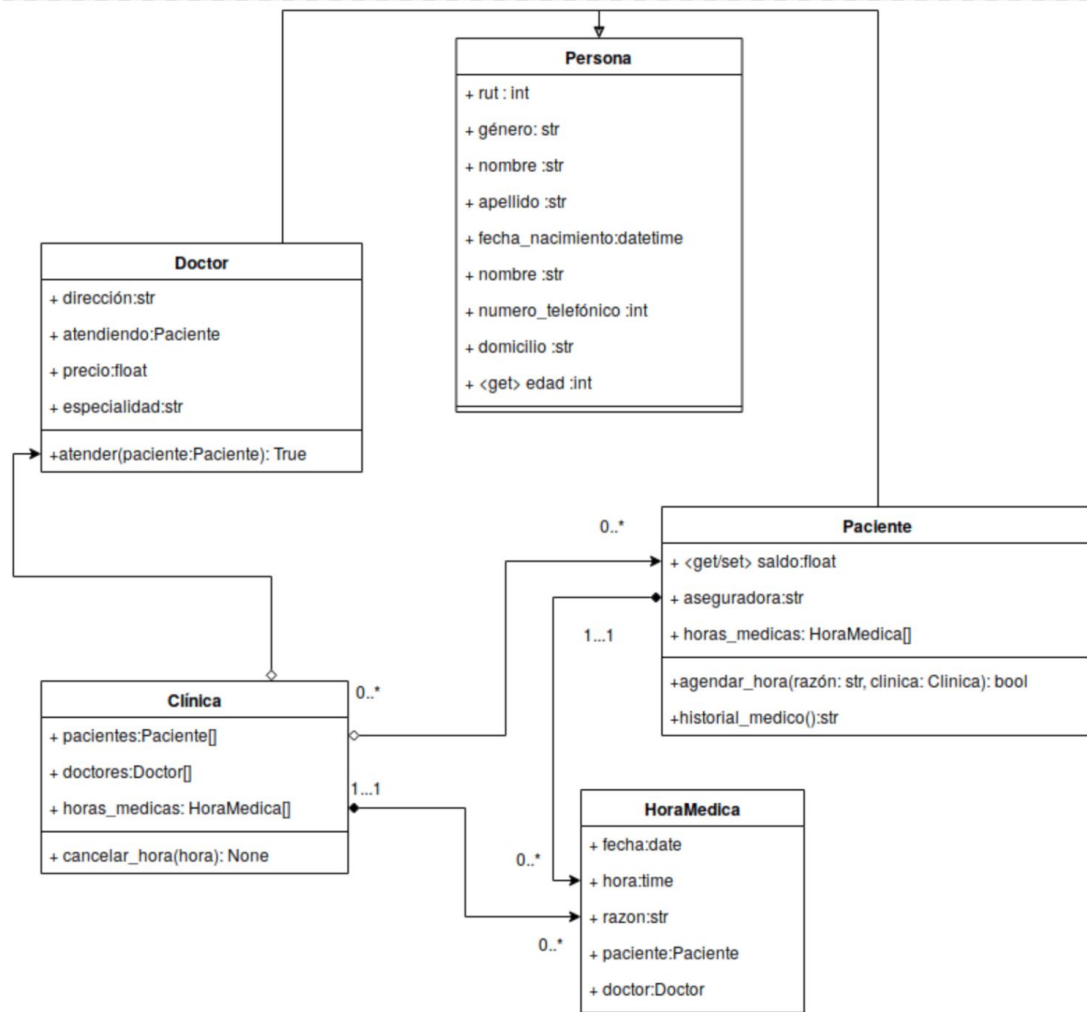
Ejercicio Pag 2.

Los doctores entonces, además de ser personas, cuentan con un número telefónico, dirección de domicilio, especialidad, y precio por realizar visita médica. Estos deben ser capaces de examinar a un paciente, sólo si no está examinando a nadie. Los pacientes por otra parte, cuentan con número telefónico, dirección de domicilio, saldo en cuenta bancaria, y saben el nombre de su aseguradora.

Los pacientes deben ser capaces de poder agendar (a través de la clínica) horas con médicos a una cierta fecha, hora, y deben especificar una razón de apunte de hora médica. Los pacientes deben poder además brindar un historial médico, cuando se les solicite.

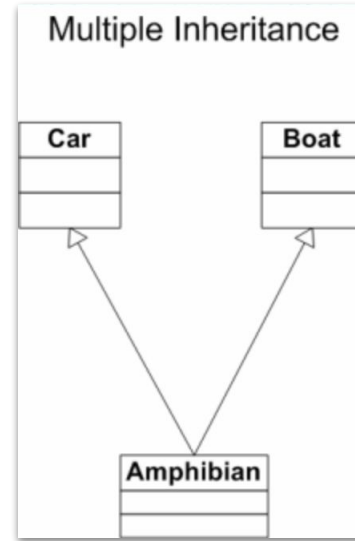
La clínica entonces, debe guardar un registro de pacientes, horas agendadas, y doctores contratados. Además la clínica debe ser capaz de cancelar una hora médica cuando se le solicite.

Resolvamos.

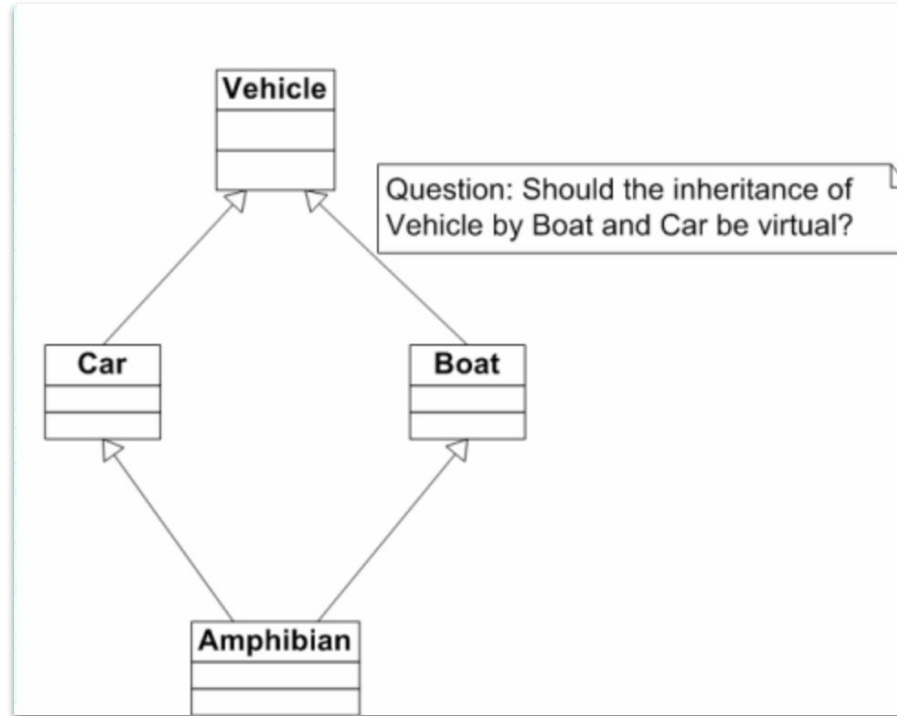


Multi Herencia

Cuando una clase hereda de dos superclases.



Deadly diamond of death



MRO

Indique el output de

```
class A:
    def __init__(self):
        print("initializing A")
        super().__init__()
    def save(self, file):
        print("A saving {}".format(file))

class B:
    def __init__(self):
        print("initializing B")
        super().__init__()
    def save(self, file):
        print("B saving {}".format(file))

class C(A, B): pass

obj_c = C()
obj_c.save("file")
```