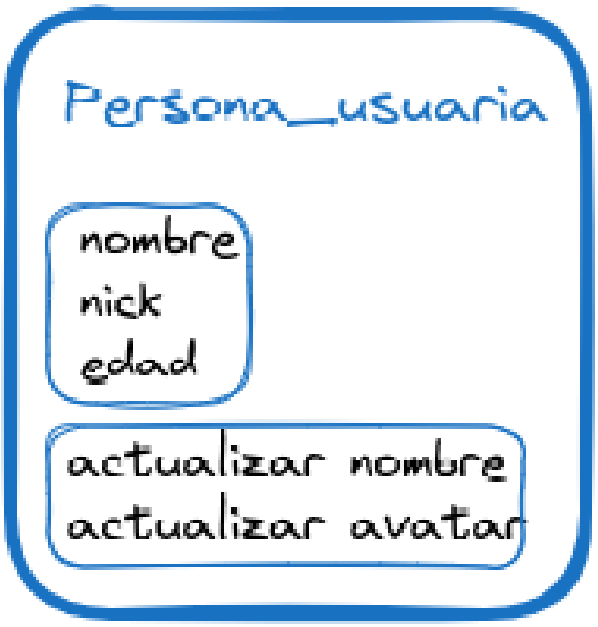
Seminario de Lenguajes - Python

Cursada 2024

Aspectos básicos de POO (Cont.)

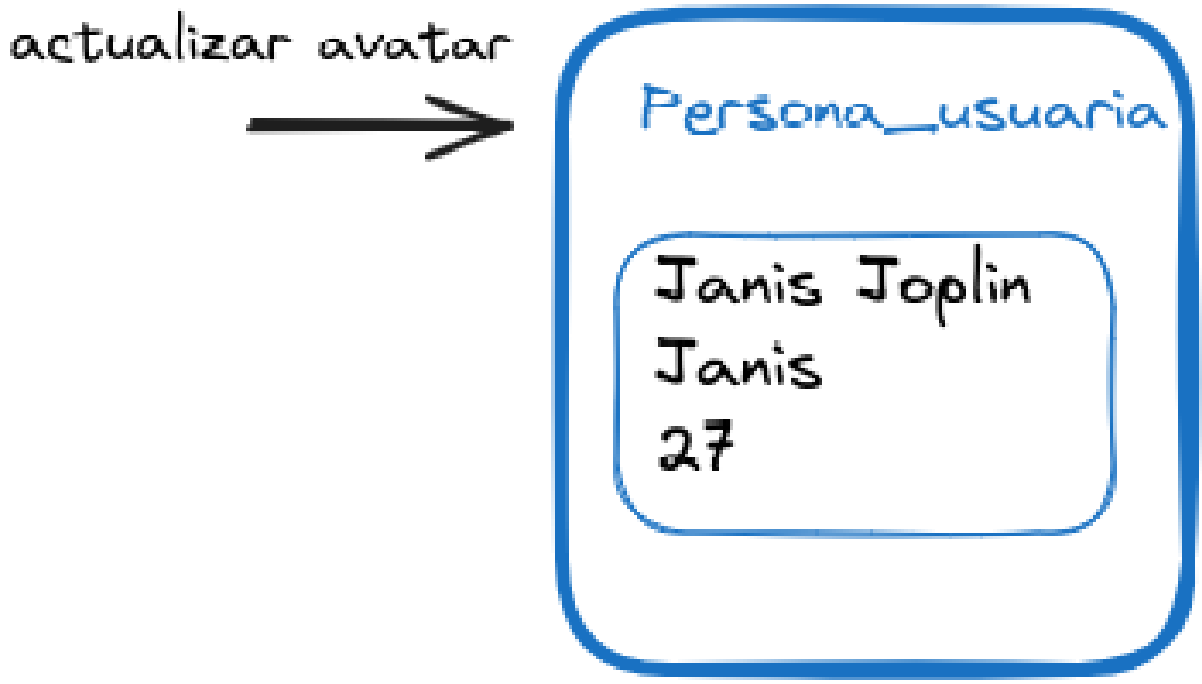
Repasemos algunos conceptos vistos previamente

> Un objeto es una colección de datos con un comportamiento asociado en una única entidad

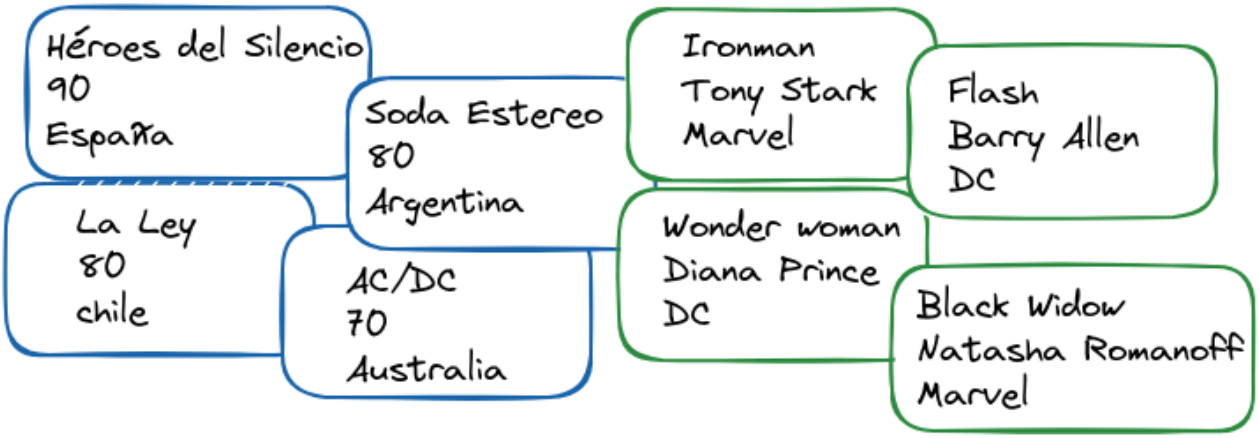


POO: conceptos básicos

- En POO un programa puede verse como un **conjunto de objetos** que interactúan entre ellos **enviándose mensajes**.
- Estos mensajes están asociados al **comportamiento** del objeto (conjunto de **métodos**).



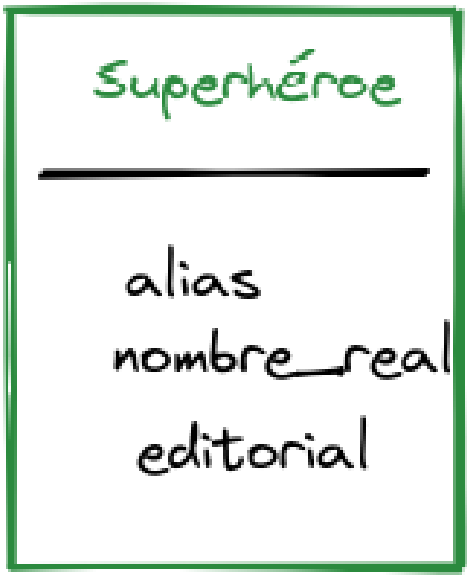
El mundo de los objetos



- No todos los objetos son iguales, ni tienen el mismo comportamiento.
- Así **agrupamos** a los objetos de acuerdo a **características comunes**.

Objetos y clases

> Una clase describe los atributos de objetos (**variables de instancia**) y las acciones (**métodos**) que pueden hacer o ejecutar dichos objetos.



La clase Band

In [1]:
```
class Band():
    """ Define la entidad que representa a una banda   ..   """
    all_genres = set()

    def __init__(self, name, genres="rock"):
        self.name = name
        self.genres = genres
        self._members = []
        Band.all_genres.add(genres)

    def add_member(self, new_member):
        self._members.append(new_member)
```

¿self?¿Cuáles son las variables de instancias?¿Y los métodos?¿Qué es **all_genres**?

Variables de instancia vs. de clase

> Una **variable de instancia** es **exclusiva de cada instancia** u objeto.

> Una **variable de clase** es única y es **compartida por todas las instancias** de la clase.

Creamos instancias de Band

In [2]:
```
soda = Band("Soda Stereo")
soda.add_member("Gustavo Cerati")
soda.add_member("Zeta Bosio")
soda.add_member("Charly Alberti")

bangles = Band("The Bangles", genres="pop-rock")
bangles.add_member("Susanna Hoffs")
bangles.add_member("Debbi Peterson")
bangles.add_member("Vicki Peterson")
bangles.add_member("Annette Zilinskas")
```

Mostramos el contenido de **Band.all_genres**

In [3]:
```
for genre in Band.all_genres:
    print(genre)
```

```
pop-rock
rock
```

Objetos y clases

- La **clase** define las variables de instancia y los métodos.
- Los **objetos** son instancias de una clase.
- Cuando se crea un objeto, se ejecuta automáticamente el método __**init()**__ que permite inicializar el objeto.
- La definición de la clase especifica qué partes son públicas y **cuáles vamos a considerar no públicas**.

¿Cómo se especifica privado o público en Python?

Mensajes y métodos

TODO el procesamiento en este modelo es activado por mensajes entre objetos.

- El **mensaje** es el modo de comunicación entre los objetos. Cuando se invoca una función de un objeto, lo que se está haciendo es **enviando un mensaje** a dicho objeto.
- El **método** es la función que está asociada a un objeto determinado y cuya ejecución sólo puede desencadenarse a través del envío de un mensaje recibido.

- La **interfaz pública** del objeto está formada por las variables de instancias y métodos que otros objetos pueden usar para interactuar con él.

Hablemos de @property

> Clase07_2 sobre propiedades

Métodos de clase

¿A qué creen que hacen referencia?

> Corresponden a los mensajes que se envían a la **clase**, no a las instancias de la misma.

- Se utiliza el decorador **@classmethod**.
- Se usa **cls** en vez de **self**. ¿A qué hace referencia este argumento?

In [ ]:
```
class Band():
    """ Define la entidad que representa a una banda   ..   """
    all_genres = set()

    @classmethod
    def clean_genres(cls, confirm=False):
        if confirm:
            cls.all_genres = set()

    def __init__(self, name, genres="rock"):
        self.name = name
        self.genres = genres
        self._members = []
        Band.all_genres.add(genres)

    def add_member(self, new_member):
        self._members.append(new_member)
```

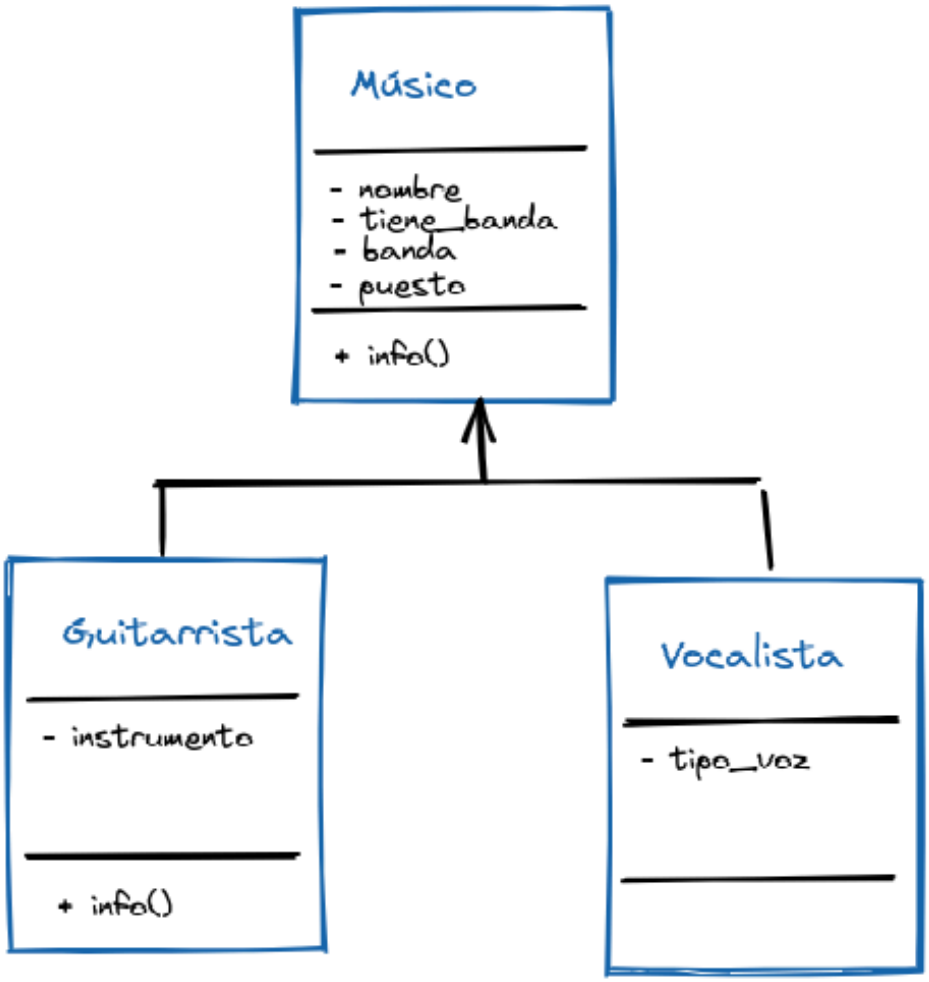
In [ ]:
```
soda = Band("Soda Stereo")
nompa = Band("Nonpalidece", genres="reggae")
```

In [ ]:
```
Band.all_genres
```

In [ ]:
```
Band.clean_genres(True)
```

Ahora, pensemos en los músicos de la banda

Podemos pensar en:



Donde:

- Un guitarrista **"es un"** músico.
- Un vocalista también **"es un"** músico.

Hablemos de herencia

- Es uno de los conceptos más importantes de la POO.
- La herencia permite que una clase pueda *heredar* los atributos y métodos de otra clase, que se **agregan** a los propios.
- Este concepto permite sumar, es decir **extender** una clase.
- La clase que hereda se denomina **clase derivada** y la clase de la cual se deriva se denomina **clase base**.
- Así, **Músico es la clase base** y **Guitarrista y Vocalista son clases derivadas** de Músico.

Ahora en Python

In [1]:
```
class Musician:
    def __init__(self, name, role=None, band=None):
        self.name = name
        self.has_a_band = band!=None
        self._band = band
        self.role = role

    def info(self):
        if self.has_a_band:
            print (f"{self.name} integra la banda {self.band}")
        else:
            print(f"{self.name} es solista ")

    @property
    def band(self):
        if self.has_a_band:
            return self._band
        else:
            return "No tiene banda"

    @band.setter
    def band(self, new_band):
        self._band = new_band
        self.has_a_band = self._band!=None
```

In [2]:
```
class Guitarist(Musician):

    def __init__(self, name, band=None):
        Musician.__init__(self, name, "guitarrista", band)
        self.guitar_type = "guitarra acústica"

    def info(self):
        print (f"{self.name} toca {self.guitar_type}")
```

- ¿Cuál es la clase base? ¿Y la clase derivada? ¿Cuáles son las variables de instancia de un objeto Guitarist?
- ¿Por qué invoco a **Musician__init__()**? ¿Qué pasa si no hago esto?

In [3]:
```
class Vocalist(Musician):

    def __init__(self, name, band=None):
        Musician.__init__(self, name, "vocalista", band)
        self.voice_type = "Barítono"
```

In [4]:
```
bruce = Vocalist('Bruce Springsteen')
brian = Guitarist("Brian May", "Queen")
```

In [5]:
```
bruce.info()
```

```
Bruce Springsteen es solista
```

In [6]:
```
brian.info()
```

```
Brian May toca guitarra acústica
```

In [ ]:
```
bruce.has_a_band
```

In [7]:
```
bruce.band = "E Street Band"
bruce.info()
```

```
Bruce Springsteen integra la banda E Street Band
```

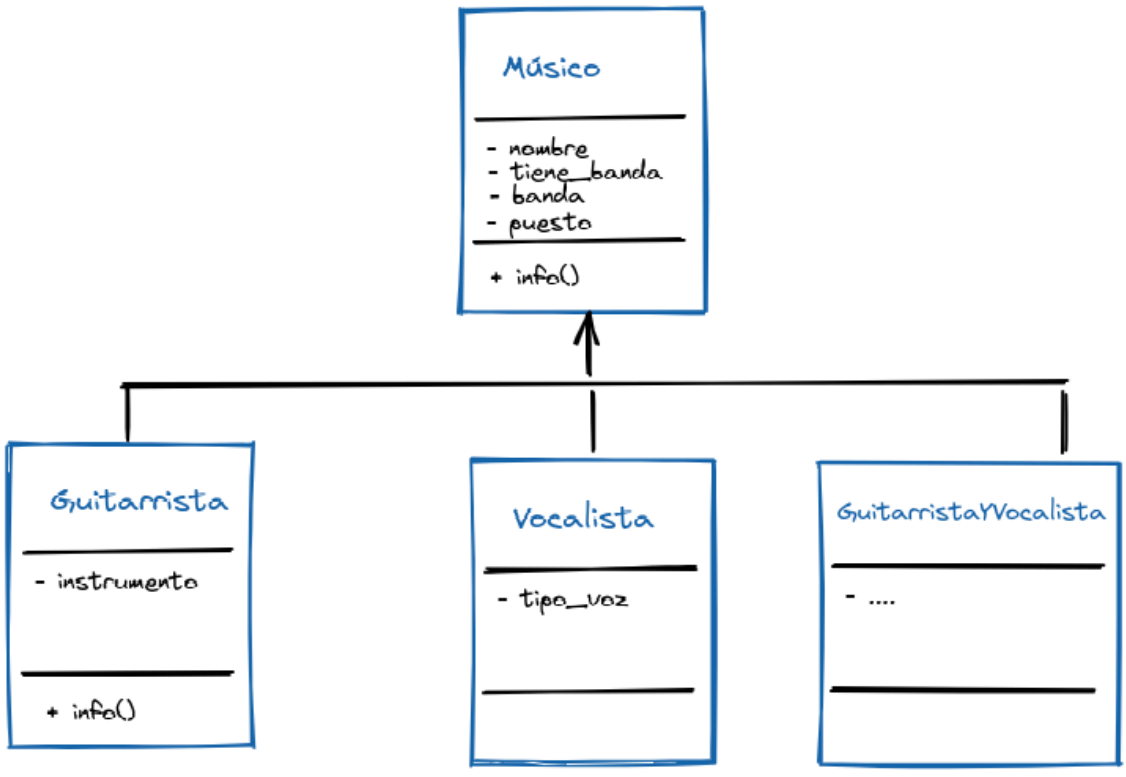
También podemos chequear ...

In [ ]:
```
f"{bruce.name} es vocalista" if isinstance(bruce, Vocalist) else f"{bruce.name} NO es
```

In [ ]:
```
"Guitarrista ES subclase de Musico" if issubclass(Guitarist, Musician) else "Guitarri
```

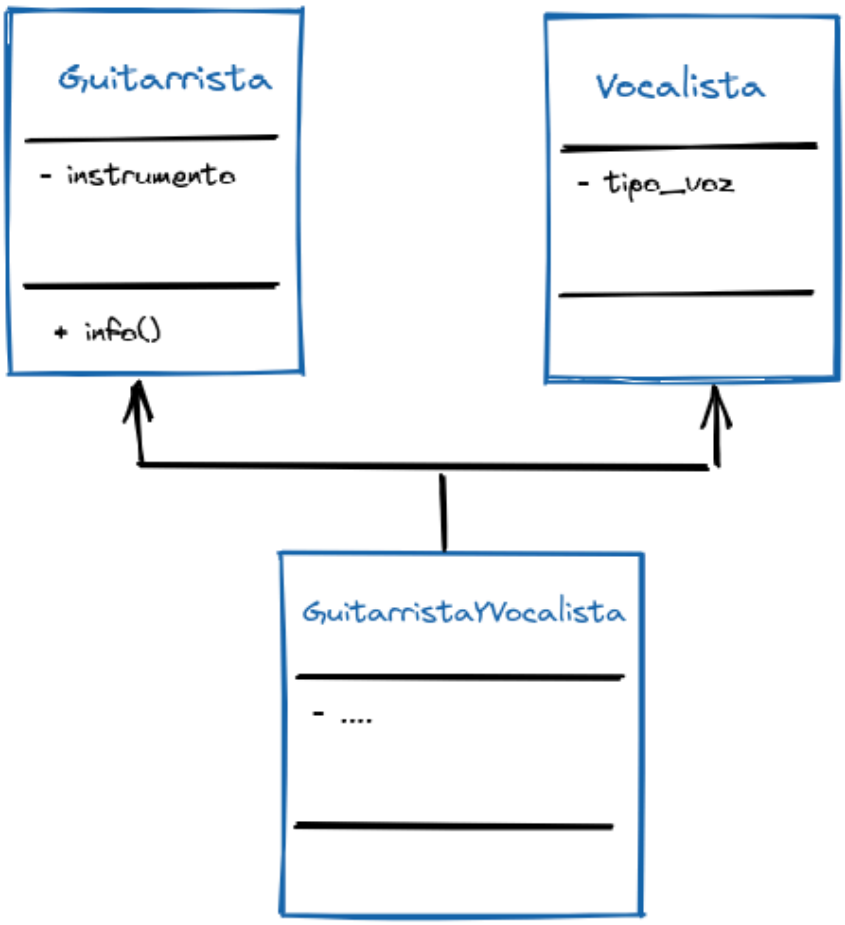
Bruce Springsteen es un vocalista, pero también es un guitarrista...

Podríamos pensar en algo así:



Aunque en Python podemos hacer algo mejor ...

Python tiene herencia múltiple



- Un guitarrista y vocalista **"es un"** guitarrista, pero también **es un** vocalista..

En Python ...

In [8]:
```
class Guitarist(Musician):
    def __init__(self, name, band=None):
        Musician.__init__(self, name, "guitarrista", band)
        self.guitar_type = "guitarra acústica"

    def info(self):
        print (f"{self.name} toca {self.guitar_type}")
```

In [9]:
```
class Vocalist(Musician):
    def __init__(self, name, band=None):
        Musician.__init__(self, name, "vocalista", band)
        self.voice_type = "Barítono"

    def info(self):
        if self.has_a_band:
            print (f"{self.name} CANTA en la banda {self.band}")
        else:
            print(f"{self.name} es solista ")
```

In [10]:
```
class VocalistAndGuitarist(Guitarist, Vocalist):
```

```
    def __init__(self, name, band=None):
        Vocalist.__init__(self, name, band)
        Guitarist.__init__(self, name, band)
```

In [11]:
```
bruce = VocalistAndGuitarist('Bruce Springsteen')
bruce.info()
```

```
Bruce Springsteen toca guitarra acústica
```

In [12]:
```
celeste = VocalistAndGuitarist("Celeste Carballo")
celeste.info()
```

```
Celeste Carballo toca guitarra acústica
```

A tener en cuenta ...

- MRO "Method Resolution Order"
- Por lo tanto, es MUY importante el orden en que se especifican las clases bases.
- Más información en documentación oficial

In [13]:
```
VocalistAndGuitarist.__mro__
```

Out[13]:
```
(__main__.VocalistAndGuitarist,
 __main__.Guitarist,
 __main__.Vocalist,
 __main__.Musician,
 object)
```

In [14]:
```
class VocalistAndGuitarist(Vocalist, Guitarist):

    def __init__(self, name, band=None):
        Vocalist.__init__(self, name, band)
        Guitarist.__init__(self, name, band)
```

In [15]:
```
bruce = VocalistAndGuitarist('Bruce Springsteen')
bruce.info()
```

```
Bruce Springsteen es solista
```

In [16]:
```
celeste = VocalistAndGuitarist("Celeste Carballo")
celeste.info()
```

```
Celeste Carballo es solista
```

In [15]:
```
VocalistAndGuitarist.__mro__
```

Out[15]:
```
(__main__.VocalistAndGuitarist,
 __main__.Vocalist,
 __main__.Guitarist,
 __main__.Musician,
 object)
```

¿Qué términos asociamos con la programación orientada a objetos?

Destacados ...

- Encapsulamiento

    - **class**, métodos privados y públicos, propiedades.
- Herencia
    - Clases bases y derivadas.
    - Herencia múltiple.
- **¿Alguno más?**

Polimorfismo

- Capacidad de los objetos de distintas clases de responder a mensajes con el mismo nombre.
- Ejemplo: + entre enteros y cadenas.

In [ ]:
```
print("hola " + "que tal.")
print(3 + 4)
```

¿Podemos sumar dos músicos?

In [17]:
```
adele = Musician("Adele")
sting = Musician("Sting", "The Police")

print(adele + sting)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 4
      1 adele = Musician("Adele")
      2 sting = Musician("Sting", "The Police")
----> 4 print(adele + sting)

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

In [18]:
```
class Musician:
    def __init__(self, name, instrument=None, band=None):
        self.name = name
        self.has_a_band = band!=None
        self._band = band
        self.instrument = instrument

    def info(self):
        if self.has_a_band:
            print (f"{self.name}  integra la banda {self.band}")
        else:
            print(f"{self.name} es solista ")

    def __add__(self, other):
        return (f"Nuevo dúo: {self.name} y {other.name}")

    @property
    def band(self):
        if self.has_a_band:
            return self._band
        else:
            return "No tiene banda"

    @band.setter
    def band(self, new_band):
        self._band = new_band
        self.has_a_band= self._band!=None
```

In [19]:
```
adele = Musician("Adele")
sting = Musician("Sting", "The Police")

print(adele + sting)
```

```
Nuevo dúo: Adele y Sting
```

¿Polimorfismo en nuestros músicos?

In [ ]:
```
bruce.info()
brian.info()
```

DESAFIO - Probamos en casa

¿Qué podemos decir de las variables de instancias cuyo nombre comienza con __?

In [ ]:
```
class A:
    def __init__(self, x, y, z):
        self.varX = x
        self._varY = y
        self.__varZ = z

    def demo(self):
        return f"ESTOY en A: x: {self.varX} -- y:{self._varY} --- z:{self.__varZ}"

class B(A):
    def __init__(self):
        A.__init__(self, "x", "y", "z")

    def demo(self):
        return f"ESTOY en B: x: {self.varX} -- y:{self._varY} --- z:{self.__varZ}"
```

In [ ]:
```
objB = B()
print(objB.demo())
```

Para los que quieran seguir un poco más ...

- https://realpython.com/python-classes/
- https://realpython.com/inheritance-composition-python/
- https://realpython.com/instance-class-and-static-methods-demystified/

Seguimos la próxima ...