

# Programación Orientada a Objetos

Clase #18

IIC1103 – Introducción a la Programación

Marcos Sepúlveda ([marcos@ing.puc.cl](mailto:marcos@ing.puc.cl))

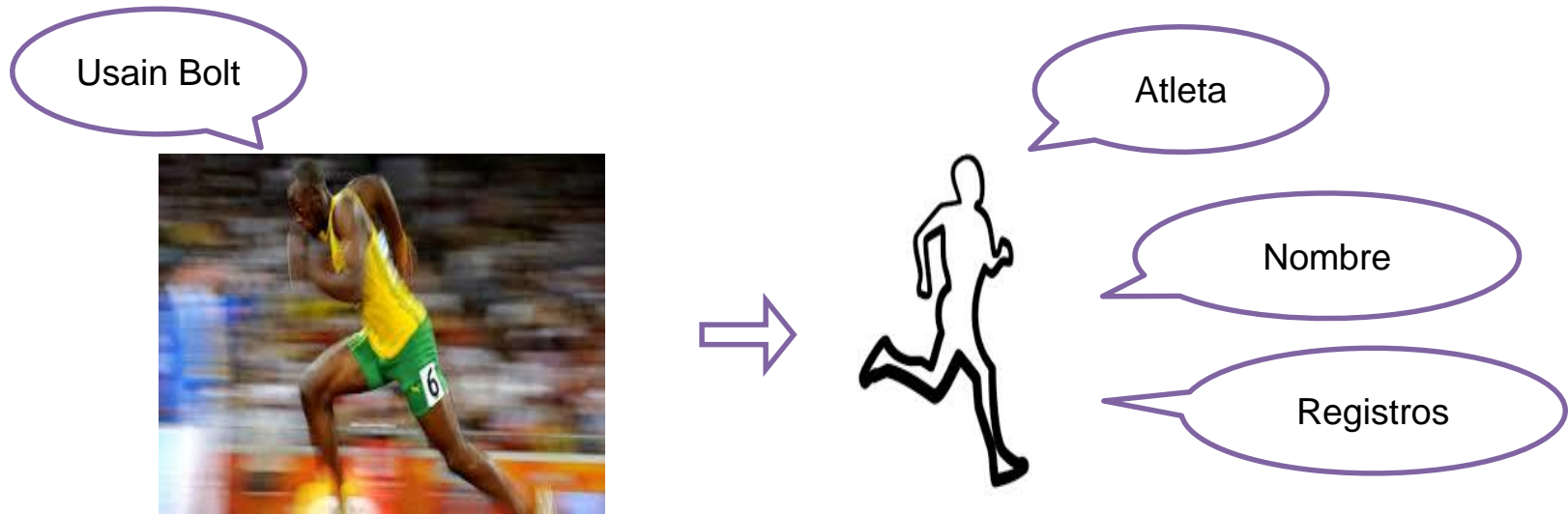
# Veremos hoy ...

---

- ▶ Sobrecarga de operadores
- ▶ Ejemplos

# Programación Orientada a Objetos (POO) – objeto

- ▶ Lo que sabe:
  - Usain Bolt es un atleta
  - Un atleta tiene varios entrenamientos
  - Un entrenamiento tiene varios tiempos
- ▶ Lo que hace:
  - Mejorar
  - Comparar



# POO – módulo de apoyo

tiempo.py

```
def Segundos(tiempo_str):  
    separados = tiempo_str.split(':')  
    segundos = int(separados[0])  
    centesimas = int(separados[1])  
    return segundos + centesimas * 0.01  
  
def FormatoSegundos(segundos_sin_formato):  
    segundos = int(segundos_sin_formato)  
    centesimas = int((segundos_sin_formato - segundos)*100)  
    return "{:02d}:{:02d}".format(segundos, centesimas)
```

# POO – clase

deporte.py

```
from tiempo import Segundos
from tiempo import FormatoSegundos

class Atleta:
    def __init__(self, nombre):
        self.nombre = nombre
        self.registros = []

    def AgregarRegistro(self, registro):
        self.registros.append(registro)

    def Mejor(self):
        minimo = -1
        for registro in self.registros:
            fecha = registro[0]
            for marca in registro[1]:
                seg = Segundos(marca)
                if (minimo == -1) or (seg < minimo):
                    minimo = seg
        return FormatoSegundos(minimo)
```

# POO – instancia de una clase

poo - atletas.py

```
from deporte import Atleta

usain = Atleta("Usain Bolt")
usain.AgregarRegistro(['2012-6-17', ['22:58','22:58','22:39']])
usain.AgregarRegistro(['2012-6-18', ['22:38','22:35']])

justin = Atleta("Justin Gatlin")
justin.AgregarRegistro(['2012-6-18', ['23:10','23:09','23:00']])

print(usain.nombre,
      "ha tenido un mejor tiempo de", usain.Mejor(),"segundos")
print(justin.nombre,
      "ha tenido un mejor tiempo de", justin.Mejor(),"segundos")
```

Usain Bolt ha tenido un mejor tiempo de 22:35 segundos  
Justin Gatlin ha tenido un mejor tiempo de 23:00 segundos

# POO – sobrecarga de operadores

---

- ▶ La **sobrecarga de operadores** permite que ciertos operadores se apliquen entre objetos de una clase definida por el programador.
- ▶ Por ejemplo, para comparar dos objetos de una clase (si son iguales, distintos, uno es mayor que otro, etc.) o para facilitar el uso de la función `print()` sobre los objetos de la clase.

# POO – sobrecarga de operadores: algunos usos

- ▶ `__str__()`
  - Método de la clase que permite generar un string. Es invocado por `print()`, cuando recibe un objeto de dicha clase como parámetro.
- ▶ `__eq__()`
  - Método que permite utilizar el operador `==` entre objetos de la clase.
- ▶ `__lt__()`
  - Método que permite utilizar el operador `<` entre objetos de la clase.
- ▶ `__le__()`
  - Método que permite utilizar el operador `<=` entre objetos de la clase.
- ▶ `__gt__()`
  - Método que permite utilizar el operador `>` entre objetos de la clase.
- ▶ `__ge__()`
  - Método que permite utilizar el operador `>=` entre objetos de la clase.



# POO – clase

deporte.py

```
class Atleta:

    def __str__(self):
        return "Hola, soy " + self.nombre + \
            ", mi mejor tiempo es: " + self.Mejor() + " segundos"

    def EsMasVeloz(self, otro_atleta):
        return self.Mejor() < otro_atleta.Mejor()

    def __lt__(self, otro_atleta):
        return self.Mejor() < otro_atleta.Mejor()

    def __eq__(self, otro_atleta):
        return self.Mejor() == otro_atleta.Mejor()

    def __le__(self, otro_atleta):
        return self.Mejor() <= otro_atleta.Mejor()
```

# POO – sobrecarga de operadores

poo - atletas.py

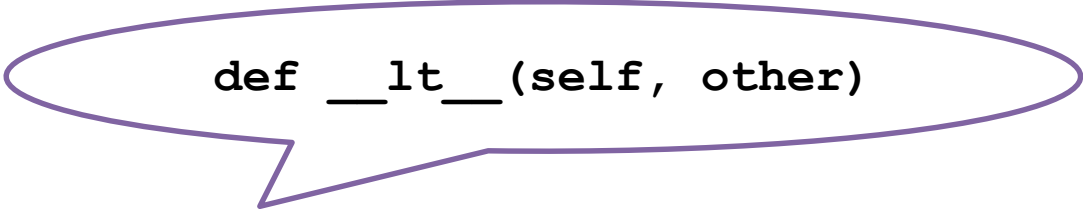
```
print("-----")
print(usain)
print("...")
print(justin)

print("-----")
print("Saludo del más veloz:")
if usain.EsMasVeloz(justin):
    print(usain)
else:
    print(justin)

print("-----")
print("Saludo del más veloz:")
if usain < justin:
    print(usain)
else:
    print(justin)
```

```
-----
Hola, soy Usain Bolt, mi mejor tiempo es: 22:35 segundos
...
Hola, soy Justin Gatlin, mi mejor tiempo es: 23:00 segundos
-----
Saludo del más veloz:
Hola, soy Usain Bolt, mi mejor tiempo es: 22:35 segundos
-----
Saludo del más veloz:
Hola, soy Usain Bolt, mi mejor tiempo es: 22:35 segundos
```

# POO – sobrecarga de comparación



```
def __lt__(self, other)
```

```
usain < justin
```

- ▶ Se dice **sobrecarga** porque el operador `<` no está definido para comparar objetos de cualquier tipo, por lo tanto hay que definir el operador en la clase.
- ▶ Nota: `__lt__` y `__gt__`
  - Basta con definir uno de los dos, el otro se calcula automáticamente al invertir el orden entre los objetos

# Ejercicio – Panoramas en Santiago

- ▶ Entusiasmado con la llegada de la primavera, se quiere implementar una clase para almacenar los atractivos turísticos de Santiago, y buscar las mejores alternativas.
- ▶ Para ello, se le pide implementar la clase **Panorama** y completar los fragmentos de código presentados a continuación, para que el programa funcione.



Fuente: <http://telefericomet.cl>

# Ejercicio – Panoramas en Santiago

- ▶ La clase **Panorama** almacena el nombre de un panorama, su índice de belleza, su índice de accesibilidad, y su índice de precio. Los índices son números entre 1 y 10 (uno es lo más bajo).
- ▶ Su método más complejo es `obtener_calidad(self)` que calcula y retorna la calidad del panorama utilizando la siguiente fórmula:  
$$\text{belleza} * 50 + \text{accesibilidad} * 25 + \text{precio} * 25$$
- ▶ `panorama.py` –  
aquí deberás implementar la clase **Panorama**
- ▶ `principal.py` –  
contiene código que usa la clase **Panorama** y que debes completar

# Ejercicio – Panoramas en Santiago

---

- ▶ **panorama.py** –  
aquí deberás completar la clase **Panorama**
- ▶ **principal.py** –  
contiene código que usa la clase **Panorama** y que debes completar

# panorama.py

---

```
class Panorama:
    # Constructor de la clase
    def __init__(self, nombre, belleza, accesibilidad, precio):
        # completar 1

    # Retorna la calidad del panorama
    def obtener_calidad(self):
        # completar 2

    # Retorna string para poder mostrar datos de un panorama con print
    def __str__(self):
        # completar 3

    # True si un panorama tiene menor calidad que otro
    def __lt__(self, otro):
        # completar 4
```

# principal.py

```
from panorama import Panorama

lista_p = []

par = Panorama("Plaza Armas", 5, 10, 10)
lista_p.append(par)
tel = Panorama("Teleférico", 10, 8, 8)
lista_p.append(tel)
zoo = Panorama("Zoológico", 7, 8, 5)
lista_p.append(zoo)
slu = Panorama("Santa Lucía", 8, 6, 10)
lista_p.append(slu)

# Mostrar lista ordenada de los panoramas
# completar 5

# Mostrar el mejor de los panoramas
# completar 6
```

```
>>>
```

**Todos los panoramas**

- Zoológico 675**
- Plaza Armas 750**
- Santa Lucía 800**
- Teleférico 900**

**El mejor panorama es:**

**Teleférico 900**



# principal.py

- ▶ Extender para leer datos desde un archivo, implementando la función `leer_panoramas`:

```
from panorama import Panorama

def leer_panoramas(nombre_archivo):
    # completar extra

lista_p = leer_panoramas("panoramas.csv")

# Mostrar lista ordenada de los panoramas
# completar 5

# Mostrar el mejor de los panoramas
# completar 6
```

`"panoramas.csv"`

```
Nombre;Belleza;Accesibilidad;Precio
Plaza Armas;5;10;10
Teleférico;10;8;8
Zoológico;7;8;5
Santa Lucía;8;6;10
```