


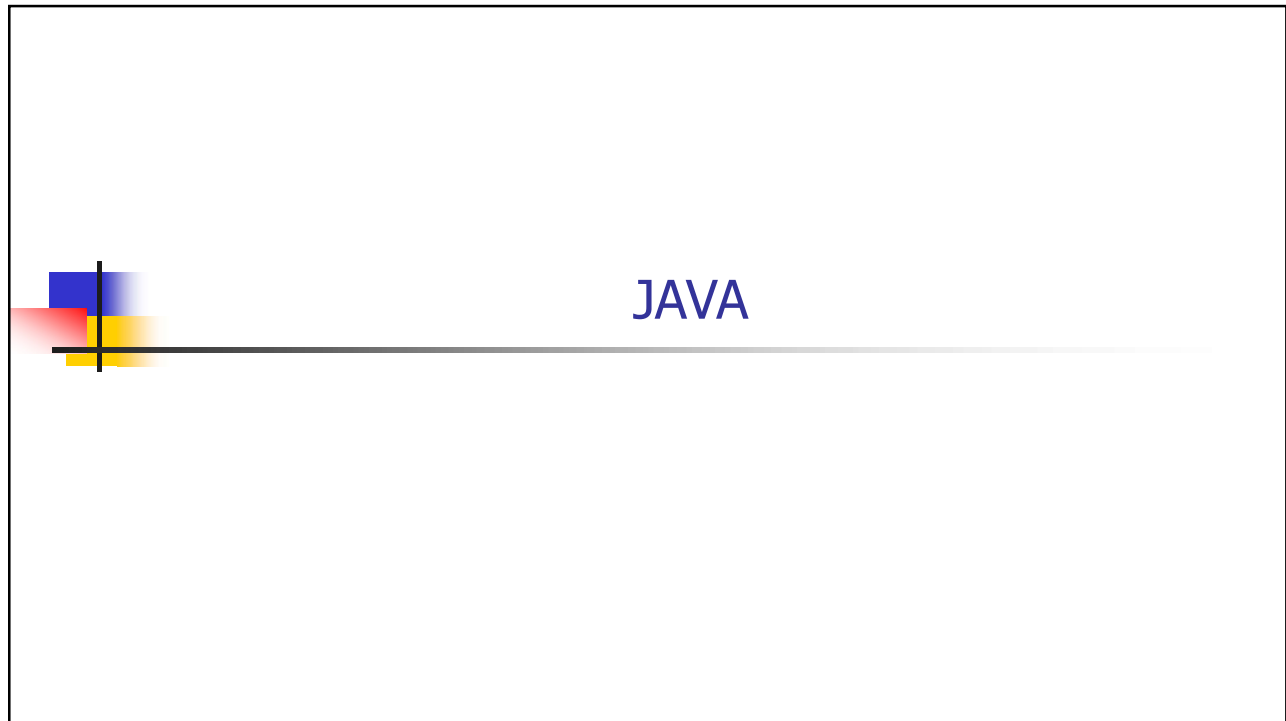
Encapsulamiento y Constructores

JAIME ALBERTO GUZMAN LUNA
Universidad Nacional de Colombia
Medellín



Contenido

- El encapsulamiento
- Los constructores

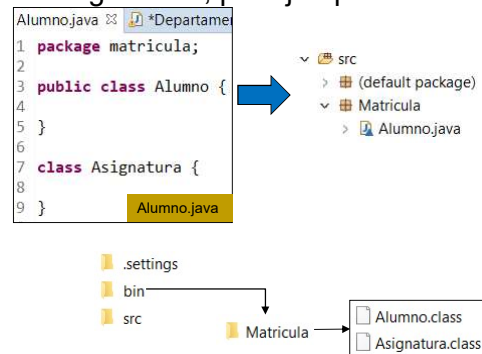


Paquetes

- Las clases se puede organizar por paquetes
 - Un paquete es una unidad modular del lenguaje que permite agrupar clases que están relacionadas
- La pertinencia de una clase a un paquete se debe especificar antes de la declaración (**package**)
 - `package nombrePaquete;`
- Cuando no se declara en que paquete queda una clase esta se crea en el paquete por defecto (**default** en el caso de eclipse)
- Los paquetes en el sistema operativo realmente se manejan como una estructura de directorios

Ejemplo

- Se puede crear un paquete con la información relativa a la matricula de alumnos, que incluya alumnos, asignaturas, por ejemplo.



Paquetes

- Para usar algún componente de un paquete hay que añadir una declaración de importación, que puede ser de un elemento o de todos los elementos.

- Un solo elemento:

```
import matricula.Alumno;
```

Se importa sólo la clase Alumno

- Todo el paquete sin cualificación:

```
import matricula.*;
...
Alumno alumno1;
...
```

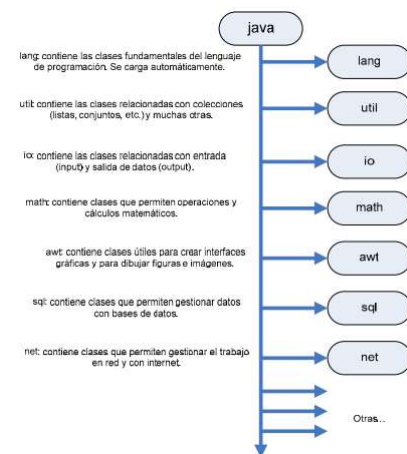
Se importa todo y para usar los elementos no es necesario cualificarlos

- Es posible no usar la cláusula **import**, simplemente cada vez que necesitemos algo escribimos su ubicación completa:

```
package mipaquete;
```

```
class MiClase {
    geometria.figuras.Rectangulo r;
    ...
}
```

Se hace uso de la clase Rectángulo la cual está en el subpaquete figuras que está en el paquete geometría



EJEMPLOS DE PAQUETES

Control de acceso: a nivel de clase

- Las **clases por defecto** son “**propias**” del paquete (**package**) en el que están definidas.
 - son **conocidas** por cualquier otra clase del mismo paquete, pero no se podrán usar en otros programas o paquetes.
- Para que se puedan **usar en otros programas o paquetes** se deben declarar como “**públicas**”, usando el modificador de acceso **public**:
 - public class ...**

A.java

```

1 package p1;
2
3 // clase pública
4 public class A {
5
6
7 }
8
9 //clase del paquete
10 class B {
11
12
13 }
  
```

Paquete p1

C.java

```

1 package p2;
2 import p1.*;
3
4 class C {
5     A a;
6     B b; // Error
7
8 }
  
```

Paquete p2

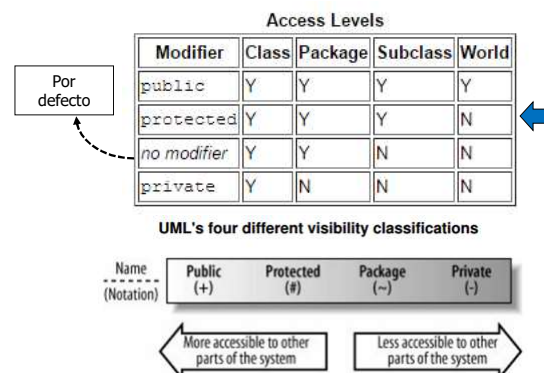


NOTA

Cada archivo fuente sólo puede tener **una clase pública**. Lo normal es declarar cada clase en su propio archivo fuente.

Control de acceso: Atributos y Métodos (1)

- Los atributos y los métodos de tipo por defecto (**tipo package**), son conocidos en todas las clases del paquete en el que se encuentra su clase.
- A los atributos y los métodos se les puede aplicar tanto el modificador de acceso **public** como el modificador de acceso **private**.
 - Un atributo o método **public** se conocerá en cualquier otro programa o paquete, pudiendo ser entonces accedido por medio de objetos de esa clase (importando la clase en el programa).
 - Un atributo o método **private** sólo se conoce en la propia clase en la que está definido, pudiendo ser accedido tan sólo en ella.
- Nota: El modificador **protected** se estudiará en el tema de la herencia



Las directrices de POO recomiendan:

- Los atributos se declaran privados.
- Los métodos públicos.

Ejercicio 1 (1/2)

Ubique los modificadores de acceso en los espacios en blanco del siguiente código, de tal manera que las clases, atributos y métodos queden lo más encapsulados posible

```
package paquete1;

import paquete2.*;
import paquete2.paquete3.*;

(1) _____ class ClaseA {
(2) _____ int atributoA1;
(3) _____ int atributoA2;
(4) _____ static int atributoA3;

(5) _____ boolean metodoA() {
    return atributoA1 == ClaseD.atributoD1;
}
}

package paquete2;

import paquete1.*;
import paquete2.paquete3.*;

(6) _____ class ClaseB {
(7) _____ int atributoB1;
(8) _____ int atributoB2;
(9) _____ int atributoB3;

(10) _____ boolean metodoB(ClaseC objeto) {
    if (objeto == null) {
        objeto = new ClaseC();
    }

    return atributoB1 == objeto.atributoC2;
}
}
```

```
package paquete1;

import paquete2.*;
import paquete2.paquete3.*;

(11) _____ class ClaseC {
(12) _____ int atributoC1;
(13) _____ int atributoC2;
(14) _____ int atributoC3;

(15) _____ boolean metodoC(ClaseA objeto) {
    return atributoC2 == ClaseA.atributoA3 &&
        objeto.metodoA();
}
}

package paquete2.paquete3;

import paquete1.*;
import paquete2.*;

(16) _____ class ClaseD {
(17) _____ static int atributoD1;
(18) _____ int atributoD2;
(19) _____ int atributoD3;

(20) _____ boolean metodoD(ClaseB objeto) {
    return objeto.atributoB1 == atributoD1 &&
        objeto.atributoB2 == atributoD2 ||
        objeto.metodoB(null);
}

public static void main(String[] args) {
    ClaseD objeto1 = new ClaseD();
    ClaseB objeto2 = new ClaseB();
    System.out.println(objeto1.metodoD(objeto2));
}
}
```



For students — Enter a Quizizz Code
Click the link to join now.
quizizz.com

<https://quizizz.com/join>

Acceso y modificación de atributos

- La forma de acceder los atributos es a través de métodos públicos:
 - Metodos `get()`: método que nos devuelve el valor de un atributo
 - Metodo `set()`: método que nos permite ajustar el valor de un atributo.
- Normalmente se tiene un `get()` y un `set()` por cada atributo

Ejemplo

```
class CuentaBancaria {
    private long numero;
    private String titular;
    private double saldo;
    public long getNumero () {
        return numero;
    }
    public void setNumero (long num) {
        numero = num;
    }
    public String getTitular () {
        return titular;
    }
    public void setTitular (String tit) {
        titular = tit;
    }
    public double getSaldo () {
        return saldo;
    }
    // No setSaldo por que se modifica con
    // ingresar y retirar
    ...
}
```

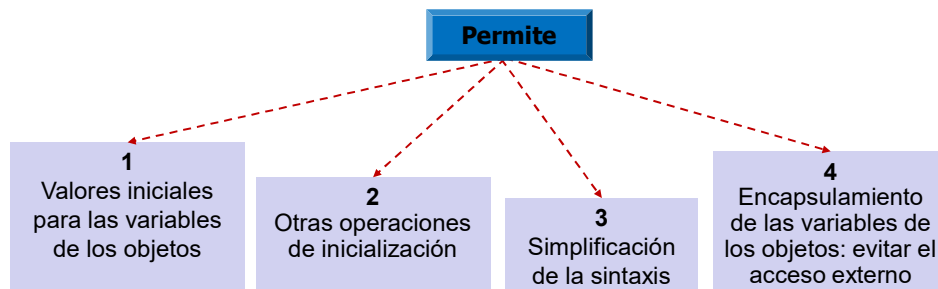


Los constructores



Los constructores(1)

- Métodos que se ejecutan automáticamente al crear los objetos de una clase (*i.e.* al invocar a *new*)



Los constructores(2)

■ Características

- Tiene el mismo nombre de la clase
- No devuelven ningún tipo (ni siquiera void).
- El tipo implícito que devuelve un constructor es el propio tipo de la clase.

■ Sintaxis de los constructores

nombreConstructor (*lista parametros*){
 // bloque de código...

tipoRetorno: No posee.
 Implícitamente devuelve una variable del tipo de la clase.

nombreConstructor: Debe ser el mismo nombre de la clase que se esta definiendo.

lista parámetros: Sucesión de pares de tipo e identificador separado por comas. Corresponderán a uno, varios o todos los atributos de la clase. Si esta lista es vacía, se esta definiendo el **constructor por defecto** de la clase.

Ejemplo de constructores

```
public class Cliente {
    private String nombre;
    private long cedula;
    public Cliente (String str, long num){
        nombre = str;
        cedula = num;
    }
}
```

```
public class CuentaBancaria {
    private long numero;
    private Cliente titular;
    private double saldo;
    public CuentaBancaria (long num, Cliente clt, long s){
        numero = num;
        titular = clt;
        saldo = s;
    }
}
```

Creación de objetos con constructores

Cliente cliente1 = new Cliente ("Luis Gomez", 25672046);

	cliente1
nombre	"Luis Gomez"
cedula	25672046

CuentaBancaria cuenta1= new CuentaBancaria (6831531, cliente1, 100000);

CuentaBancaria cuenta2= new CuentaBancaria (8350284, new Cliente ("Pilar García", 15165442), 200000);



El constructor por defecto

- Si no se definen constructores, Java proporciona uno por defecto

```
class A {
    A () {}
}
```

- Si se define un constructor, el constructor por defecto **NO** es definido

```
class Cliente {
    ...
    Cliente (String str, long num) { ... }
}
```



```
// Bloque main
Cliente clientel = new Cliente ();
// Error: No constructor matching Cliente() found in Cliente
```


Problema 1

- Que saca el siguiente programa?
Explique su respuesta

```
class Start {
    public void Start() {
        System.out.println("Constructor START");
    }
}

public class Test {
    public static void main(String[] args) {
        Start s = new Start();
    }
}
```

■ SOLUCION

NO saca nada porque la clase Test crea un objeto Start con el constructor por defecto el cual no se ha programado. Es importante aclarar que la clase Start solo tiene el método Start y no tiene definido ningún constructor.



Ejercicio-2

- ¿Qué imprime el siguiente programa?

```
public class TestClase1 {
    public static void main (String [] args) {
        Clase1 obj1= new Clase1(6,5);
        System.out.print(obj1.modificar(2)+" ");
        Clase1 obj2= new Clase1(3,8);
        System.out.print(obj2.modificar(1)+" ");
        obj2=obj1;
        System.out.println(obj2.modificar(4)+" ");
    }
}
```

```
class Clase1{
    int p1,p2;
    public Clase1 (int i, int j){
        p1=i;
        p2=j;
    }

    public int modificar(int i){
        p1=p1+i;
        p2=p2+i;
        System.out.print(p2+" ");
        return p1;
    }
}
```

Constante y el constructor

- Un constructor **No** puede contener un atributo final (si ya se le asigno el valor)

NOTA: El valor de la constante puede ser asignada en el constructor si no se asigna directamente en el atributo:

```
class A {
    int x;
    float y;
    final int z;
```

```
    A (int param1, float param2, int param3 ) {
        x = param1;
        y = param2;
        z = param3;
    }
}
```

```
class A {
    int x;
    float y;
    final int z=7;
```

```
    A (int param1, float param2, int param3 ) {
        x = param1;
        y = param2;
        z = param3;
    }
}
```

cannot assign a value to final variable z

La variable This en los constructores

- En un constructor se puede usar **this**, para conseguir una referencia a un atributo del objeto asociado y evitar la ambigüedad de variables

```
class Persona {
    String nombre;
    String apellido;
    String direccion;

    public Persona(String nombre, String apellido, String direccion) {
        super();
        this.nombre = nombre;
        this.apellido = apellido;
        this.direccion = direccion;
    }

    public String toString() {
        return "Persona = " + this.nombre + " " + apellido + " - Dir: " +
            direccion;
    }

    public static void main(String[] args) {
        Persona p3 = new Persona("Pepe", "Garcia", "Gran Via 14");
        System.out.println(p3);
    }
}
```

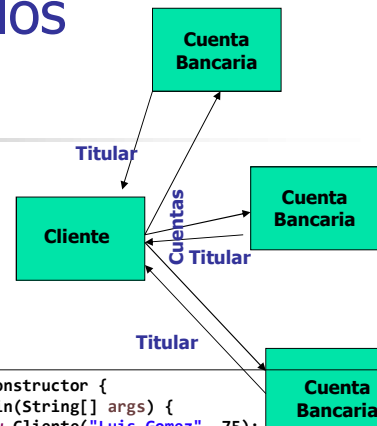
Las variables precedidas de **this**, hacen referencia al atributo de la clase. Las que no tiene el **this** son las variables del constructor

<terminated> Persona [Java Application] C:\Program Files
Persona = Pepe Garcia - Dir: Gran Via 14

La variable *This* en los constructores

```
class Cliente {
    String nombre;
    long cedula;
    Cliente (String str, long num){
        nombre = str; cedula = num;
    }
    CuentaBancaria cuentas []= new CuentaBancaria[20];
    int ncuentas=0;
    void nuevaCuenta (CuentaBancaria cuenta) {
        cuentas [ncuentas++] = cuenta;
    }
}

class CuentaBancaria {
    long numero;
    Cliente titular;
    double saldo;
    CuentaBancaria (long num, Cliente clt, long s) {
        numero = num; titular = clt; saldo = s;
        clt.nuevaCuenta (this);
    }
}
```



```
public class EjemploConstructor {
    public static void main(String[] args) {
        Cliente cliente1 = new Cliente("Luis Gomez", 75);
        CuentaBancaria cuenta1 = new CuentaBancaria(100, cliente1, 200000);
        CuentaBancaria cuenta2 = new CuentaBancaria(200, cliente1, 500000);
        System.out.println ("cliente1.nombre: " + cliente1.nombre);
        System.out.println ("cliente1.ncuentas: " + cliente1.ncuentas);
        System.out.println ("cliente1.cuentas[0].numero: " +
            cliente1.cuentas[0].numero);
        System.out.println ("cuenta1.numero: " + cuenta1.numero);
    }
}
```

El mismo objeto cuenta

```
cliente1.nombre: Luis Gomez
cliente1.ncuentas: 2
cliente1.cuentas[0].numero: 100
cuenta1.numero: 100
```

Python



El Encapsulamiento



Módulos en Python

- Un **módulo** (module) en Python es un **archivo .py** que alberga un conjunto de **funciones**, **variables** o **clases** y que puede ser usado por otros módulos.
- Permiten reutilizar código y organizarlo mejor en **namespaces**.
 - Ejemplo: mimodulo.py
- Un módulo puede ser usado o importado en otro archivo
 - Usando **import** se puede importar todo el contenido.
 - `import mimodulo`
 - Se puede importar únicamente los componentes que interesen
 - `from mimodulo import Rectangulo`
 - importar todos los componentes del módulo con *****
 - `from mimodulo import *` → **NO SE RECOMIENDA**

```
# mimodulo.py
class Rectangulo:
    """Define un rectángulo"""
    def __init__(self, b, h):
        self.b = b
        self.h = h

    def area(self):
        return self.b * self.h
```

```
# otromodulo.py
from mimodulo import Rectangulo

rectangulo = Rectangulo(20, 10)
print("Área del rectángulo: ", rectangulo.area())
```

Área del rectángulo: 200

Nota: Cuando se quiere utilizar un módulo desde un directorio distinto ocurrirá un problema.

• `ModuleNotFoundError: No module named 'mimodulo'`

Sobre name y main

- `if __name__ == '__main__':`
 - Cada módulo tiene una variable especial `__name__`
 - (recuerde, Python usa guiones bajos dobles para variables especiales, como el método `__init__` de una clase) que especifica el nombre del módulo cuando se importó.
 - Cuando el módulo se ejecuta directamente con `python module.py`, nunca se importa, por lo que la variable `__name__` asigna como valor la cadena `"__main__"`.
- Tips:
 - Un problema muy recurrente es cuando creamos un módulo con una clase con sus métodos y añadimos algunas sentencias a ejecutar como se detalla:



Establezca como política en todos sus scripts al realizar código de pruebas el `if __name__ == "__main__":`, en caso de que otro código lo importe en el futuro.

```
# ejemplo1.py
class Rectangulo:
    """ Define un rectángulo """
    def __init__(self, b, h):
        self.b = b
        self.h = h

    def area(self):
        return self.b * self.h

rectangulo = Rectangulo(20, 10)
print("Área del rectángulo: ", rectangulo.area())
```

se ejecutará, y esto puede no ser lo que queramos

```
# ejemplo1b.py
from ejemplo1 import Rectangulo
```

Área del rectángulo: 200

- Solución: En el módulo `ejemplo1.py` reemplazar por lo siguiente que ejecuta el código si el módulo es el `__main__`:

```
if __name__ == '__main__':
    rectangulo = Rectangulo(20, 10)
    print("Área del rectángulo: ", rectangulo.area())
```

Paquetes

- Los módulos se puede organizar por paquetes
 - Un paquete es una colección de módulos en un directorio (carpeta)
 - El nombre de un paquete es el nombre del directorio
- Crear un paquete
 - Se debe crear un archivo especial `__init__.py` vacío en el directorio donde tengamos todos los módulos que queremos agrupar.
 - Si olvidamos este archivo, no podremos importar módulos desde esa carpeta.
- La jerarquía de carpetas se verá así, enrutada en un directorio en la carpeta del proyecto, comúnmente llamado **src**:

```
src/
+-- main.py
+-- ecommerce/
+-- __init__.py
+-- database.py
+-- products.py
+-- payments/
| +-- __init__.py
| +-- common.py
| +-- square.py
| +-- stripe.py
+-- contact/
+-- __init__.py
+-- email.py
```

- Ahora, si utilizamos un script desde el mismo directorio donde se encuentra el paquete podemos acceder a los módulos, pero esta vez refiriéndonos al paquete y al módulo, así:

```
# script (main.py) en directorio src
from ecommerce.products import Rectangulo

if __name__ == '__main__':
    rectangulo = Rectangulo(20, 10)
    print("Área del rectángulo: ", rectangulo.area())
```

Paquetes

- Algunos **paquetes** y **módulos** esenciales de Python son:
 - **collections**: permite el manejo de colas
 - **concurrent**: permite el manejo de tareas en paralelo.
 - **Email**: permite el manejo de correo electrónico
 - **Html**: Este paquete define utilidades para manipular HTML
 - **tkinter**: Finalmente, y posiblemente el que me hace más ilusión de todos. Tkinter es el módulo de interfaz gráfica de facto en Python.
- **pickle**: Es un módulo que abarca las funciones necesarias para trabajar con archivos y objetos.
- **datetime**: Permite el manejo de fechas y horas.
- **sys**: Permite conseguir información del entorno del sistema operativo
- **Math**: Permite realizar operaciones aritméticas, trigonométricas y otras

C:\Users\(\nombre_de_usuario)\AppData\Local\Programs\Python\Python39\Lib

Visibilidad o control de acceso

- No existe: todos los atributos y métodos son públicos.
- Por convención se recomienda utilizar el guión bajo **_** antes del nombre de un **atributo** o de un **método** para indicar que es “privado”.

CuentaBancaria	
- numero: int	
- titular: string	
- saldo: double	
+ ingresar (cantidad: double): void	
+ retirar (cantidad: double): void	
- confirmar (): void	

```

class CuentaBancaria:
    def __init__(self, numero, titular, saldo):
        self._numero = numero
        self._titular = titular
        self._saldo = saldo

    def ingresar(self, cantidad):
        self._saldo = self._saldo + cantidad

    def retirar(self, cantidad):
        if cantidad >= self._saldo:
            print("saldo insuficiente")
        else:
            self._saldo = self._saldo - cantidad
            self._confirmar()

    def _confirmar(self):
        print("Transacción exitosa")
  
```

Visibilidad o control de acceso

- En Python es posible el uso del doble guión bajo `__` antes del nombre de un atributo o de un método para indicar que es "privado".
 - Este se usa cuando existe colisión de nombres con nombres definidos en las subclases-Herencia.
- Los nombres de atributos o métodos con doble guión bajo (eje: `__private`) no funcionan de la misma manera que en Java o C++.
 - Estos activan una mutilación de nombres cuyo propósito es evitar colisiones accidentales de espacio de nombres en las subclases:
 - `MyClass.__private` se convierte en: `MyClass._MyClass__private`.

```
class Demo:
    def __secret(self):
        print('Nadie puede saber!')

    def public(self):
        self.__secret()
```

```
demo = Demo()
demo.__secret()
```

AttributeError: 'Demo' object has no attribute '__secret'

```
demo = Demo()
demo.public()
```

Nadie puede saber!

```
demo = Demo()
demo._Demo__secret()
```

Nadie puede saber!

Nota: Nunca crear variables u operaciones con doble guion bajo. El doble guion bajo se utiliza normalmente para hacer referencia a variables propias de Python.

El método dir de Python

- Se puede hacer uso de `dir` para ver el listado de métodos y atributos de una clase.

```
class Clase1:
    atributo_clase = "Hola" # Accesible desde el exterior
    __atributo_clase = "Hola" # No accesible

    # No accesible desde el exterior
    def __mi_metodo(self):
        print("Haz algo")
        self.__variable = 0

    # Accesible desde el exterior
    def metodo_normal(self):
        # El método si es accesible desde el interior
        self.__mi_metodo()
```

```
mi_clase = Clase1()
print(dir(mi_clase))
```

```
['_Clase1__atributo_clase', '_Clase1__mi_metodo', '__class__',
 '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
 '__ge__', '__getattr__', '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
 '__str__', '__subclasshook__', '__weakref__', 'atributo_clase',
 'metodo_normal']
```



Acceso y modificación de atributos

- En Python al igual que en JAVA la forma de acceder los atributos es a través de métodos públicos:
 - **Metodos get():** método que nos devuelve el valor de un atributo
 - **Metodo set():** método que nos permite ajustar el valor de un atributo.
- Normalmente se tiene un **get()** y un **set()** por cada atributo

```
class CuentaBancaria:
    def __init__(self, numero, titular, saldo):
        self._numero = numero
        self._titular = titular
        self._saldo = saldo

    ➡ def getNumero (self):
        return self._numero
    ➡ def setNumero (self, num):
        self._numero = num
    ➡ def getTitular (self):
        return self._titular
    ➡ def setTitular (self, tit):
        self._titular = tit
    ➡ def getSaldo (self):
        return self._saldo
        # No setSaldo por que se modifica con
        # ingresar y retirar
```



Los constructores

Constructores

- Como lo vimos anteriormente, Python tiene un constructor e inicializador de datos llamado `__init__`
- Se puede catalogar en dos tipos
 - Constructor por defecto: este no lleva mas parámetros que el `self`.

```
class ClaseDemo:
    def __init__(self):
        # inicializando atributo de instancia
        self._numero = 100
```

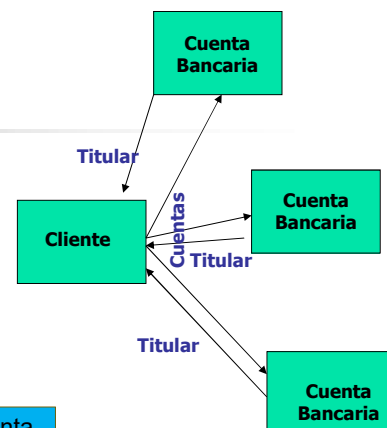
- Constructor parametrizado: Este lleva parámetros adicionales al `self` asociados a los atributos de la clase.

```
class Contacto:
    def __init__(self, nombre, email):
        self._nombre = nombre # atributo de instancia
        self._email = email # atributo de instancia
```

La variable *self* en los constructores

```
class Cliente:
    def __init__(self, nombre, cedula):
        self.nombre = nombre
        self.cedula = cedula
        self.cuentas = []
        self.ncuentas = 0
    def nuevaCuenta(self, cuenta):
        self.cuentas.append(cuenta)
        self.ncuentas = self.ncuentas + 1

class CuentaBancaria:
    def __init__(self, numero, titular, saldo):
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        titular.nuevaCuenta(self)
```



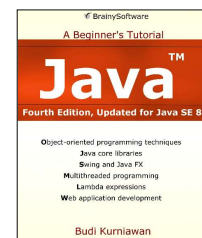
El mismo objeto cuenta

```
cliente1 = Cliente("Luis Gomez", 69)
cuenta1 = CuentaBancaria(100, cliente1, 200000)
cuenta2 = CuentaBancaria(200, cliente1, 500000)
print("cliente1.nombre: ", cliente1.nombre)
print("cliente1.ncuentas: ", cliente1.ncuentas)
print("cliente1.cuentas[0].numero: ", cliente1.cuentas[0].numero)
print("cuenta2.saldo: ", cuenta2.saldo)
```

```
cliente1.nombre: luis Gomez
cliente1.ncuentas: 2
cliente1.cuentas[0].numero: 100
cuenta1.numero: 100
```

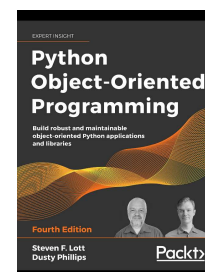
Lecturas (1)

- Chapter 4 Objects and Classes.
 - Libro: Kurniawan, Budi. Java: A Beginner's Tutorial, Updated for Java SE 8. Brainy Software. Edición de Kindle.
 - Temas de interes:
 - Java Packages
 - Encapsulation and Access Control
 - Loading, Linking, and Initialization
 - Static import



Lecturas (2)

- Capítulo 2: Objects in Python
 - Libro: Lott, Steven F.; Phillips, Dusty. Python Object-Oriented Programming: Build robust and maintainable object-oriented Python applications and libraries, 4th Edition . Packt Publishing.
 - Temas de interes:
 - Modules and packages
 - Absolute imports
 - Relative imports
 - Packages as a whole





Referencias

- Presentación basada en:
 - Apuntes del Curso Programación Orientado a Objetos. Pablo Castells. Escuela Politécnica Superior, Universidad Autónoma de Madrid.
 - Apuntes del Curso de java. Luis Hernández y Carlos Cervigón. Facultad de Informática. Universidad Católica de Madrid.