# Programación

*intermedio II*

1

# Programación Orientada a Objetos

Un sistema se compone de varios programas.

Cada uno responsable de alguna función.

Piense en un banco, tenemos función de crear cuenta, pedir tarjeta de crédito, entre otras funciones.

Usted creó un registro para un cliente abrir su cuenta.

Después el cliente pide una tarjeta de crédito y la edad necesita ser validada

(si es mayor de 18 años).

Momentos después, él pide un préstamo y usted necesita validar de nuevo la edad.

Al año siguiente él quiso hacer una inversión y necesitó ser comprobado la edad nuevamente.

Usted recordó que necesitaba modificar su edad en todas las funciones antiguas, porque ahora él tiene más edad !

**puede empeorar?**

**Sí!**

Entró un nuevo programador en la empresa, usted puede recordar todos los lugares donde necesita ser validada la edad?

Programación orientada a objetos, puede ayudarte.

Es decir, no va a tener la validación extendida por todo el código.

Si la edad del cliente se queda en un solo lugar facilita cualquier cambio.

Entonces, cuando se hace algún cambio, el resto de las funciones sólo reciben ese valor, sin tener que preocuparse.

2

# Instancias

# INSTANCIAS

Lo que toda cuenta tiene y es importante para nosotros?

- número de cuenta
- nombre del titular de la cuenta
- valor total
- tipo de cuenta

# INSTANCIAS

Lo que toda cuenta hace y es importante para nosotros?

*Esto es, qué nos gustaría "pedir a la cuenta"?*

- saca una cantidad x
- deposita una cantidad x
- imprime el nombre del titular de la cuenta
- devuelve el saldo actual
- transfiere una cantidad x a otra cuenta y
- devuelve el tipo de cuenta

Con eso, tenemos el proyecto de una cuenta bancaria.

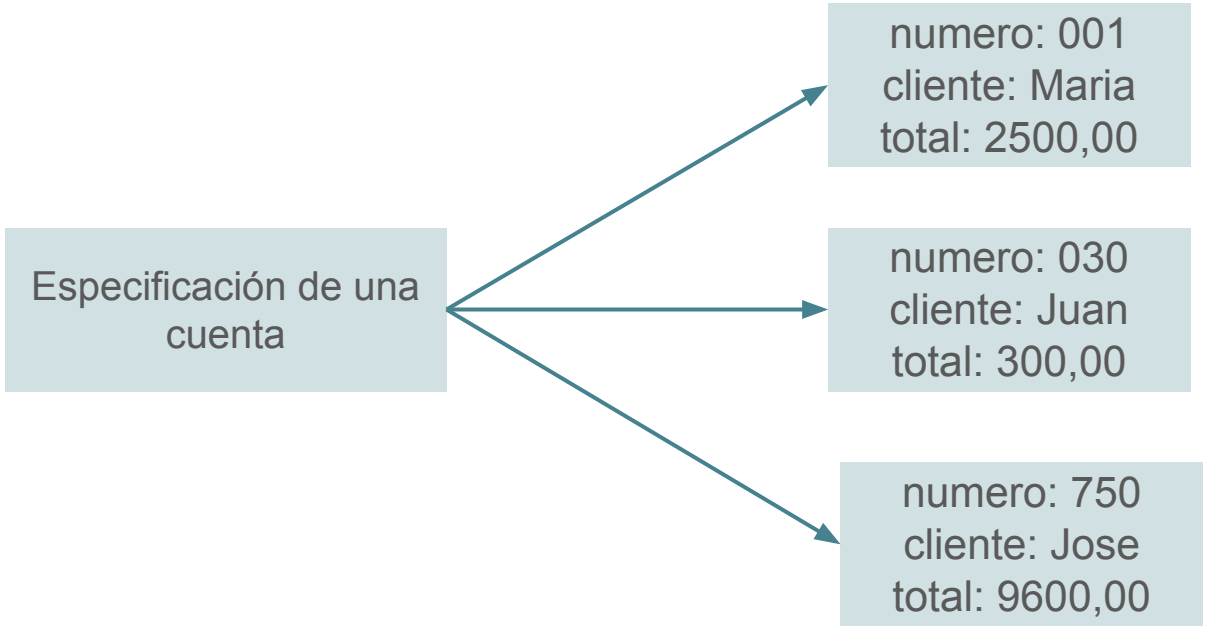
Podemos coger ese proyecto y acceder a su saldo? **No**.

Lo que tenemos es el proyecto.

Antes, necesitamos construir una cuenta, para poder acceder a lo que tiene,

y pedirle que haga algo.

# INSTANCIAS




Observe la figura:

a pesar de que el papel del lado izquierdo especifica una cuenta, esta especificación es una cuenta?

Nosotros depositamos y sacamos dinero de ese papel?

**No**, utilizamos la especificación de la cuenta para poder crear instancias que realmente son cuentas, donde podemos realizar las operaciones que creamos.

A pesar de declarar que toda cuenta tiene un saldo, un número y una agencia en el pedazo de papel (como a la izquierda en la figura), son en las instancias de ese proyecto que realmente hay espacio para almacenar esos valores.

Como un proyecto de una casa. Sólo podemos usar el baño de la casa si la construimos. Esto tiene el nombre de instancias.

Instanciamos el papel donde está diseñado y tendremos una casa.

Como un proyecto de una casa.
Sólo podemos usar el baño de la casa si la construimos.
Esto tiene el nombre de **instancias**.
**Instanciamos** el papel donde está diseñado y tendremos una casa.

3

# Classes y Objetos

Al proyecto de la cuenta, o de la casa, llamamos **CLASE** .

Donde definimos, nuestro diseño inicial.

El papel donde colocamos lo que debe tener.

A lo que podemos construir, que sería la cuenta real de una persona, o la casa de verdad, llamamos **OBJETOS**

Otro ejemplo: Una receta de torta.

La pregunta es: usted come una receta de la torta? **No**!

Necesitamos **instanciarla**, o sea,  crear un objeto torta a partir de esa especificación (la clase) para utilizarla.

Podemos crear cientos de tortas a partir de esa clase (la receta, en el caso), pueden ser muy similares, algunos incluso idénticos, pero son objetos diferentes.

# CLASES Y OBJETOS

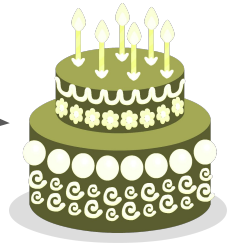
instanciar

**Class receta**

string color
string color_velas

**Objeto Torta 1**

color: marrón
color_velas : rojo



**Objeto Torta 2**

color: amarillo
color_velas :amarillo

**Objeto Torta 3**

color : rosa
color_velas: rosa oscuro

4

# Atributos, Metodos y Parametros

# ATRIBUTOS

```
class cuenta

    int numero

    String nombre_titular

    double valor_total
```

**Aquí declaramos lo que toda cuenta debe tener.**
**Esto son ATRIBUTOS**

Dentro de una clase, también tenemos que declarar lo que cada cuenta hace y cómo se hace esto.

Por ejemplo, de qué manera una Cuenta saca dinero?

Especificamos esto dentro de la propia clase Cuenta, y no en un lugar afuera.

Eso se llaman **métodos**. O sea, es la manera de hacer una operación con un objeto.

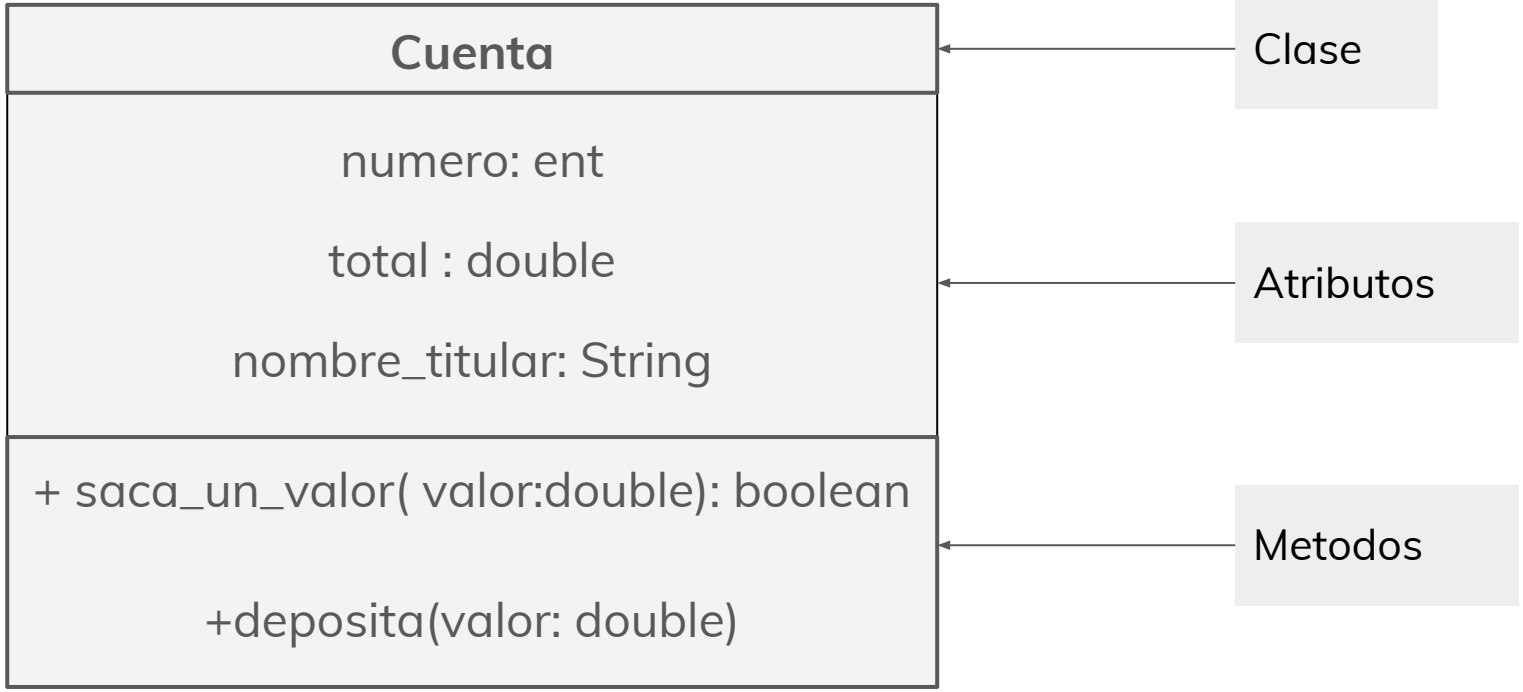
# MÉTODOS

```
metodo SacarPlata (double ctd_dinero_sacado)
    double nuevo_total = total - ctd_dinero_sacado
    total = nuevo_total
```

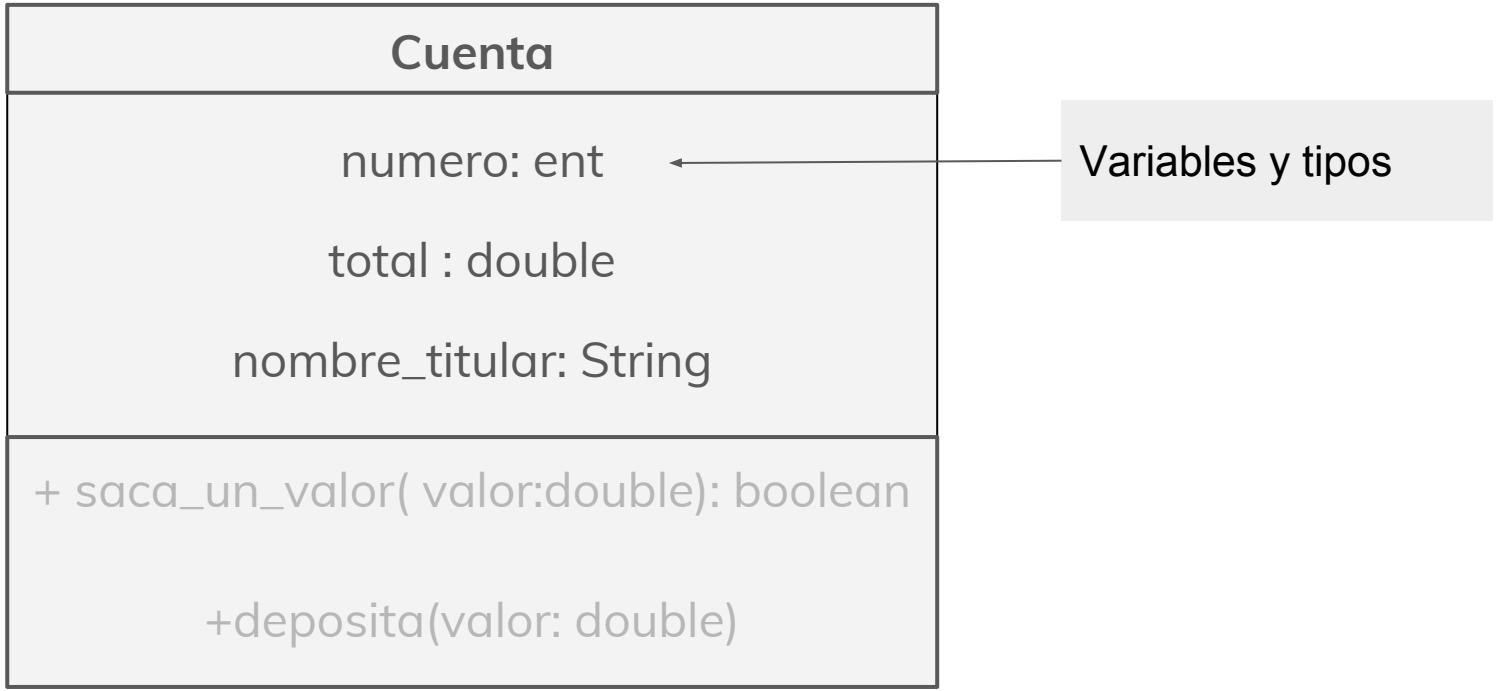
# MÉTODOS

```
metodo DepositaPlata (double ctd_deposito)

    double nuevo_total = total - ctd_deposito

    total = nuevo_total
```

# ATRIBUTOS Y MÉTODOS



# ATRIBUTOS


Cuenta
numero: ent
total : double
nombre_titular: String
+ saca_un_valor( valor:double): boolean
+deposita(valor: double)
Variables y tipos

# MÉTODOS

**+deposita(valor: double)**

**Nombre del Metodo → DEPOSITA**

**valor que necesitamos de entrada : double**

O sea, para hacer un deposito necesitamos un valor que es tipo double.

Ej: Depositamos un valor de 2345,65

# MÉTODOS

**+ saca_un_valor( valor:double): boolean**

**Nombre del Metodo → SACA_UN_VALOR**

**valor que necesitamos de entrada : double**

**retorno de el metodo: Boolean**

O sea, para sacar plata necesitamos saber el valor que es tipo double.

Ej: Sacamos un valor de 2345,65

Ademas eso, necesitamos saber si hay esa plata en la cuenta. O sea, un boolean, va retornar SI o NO.

# MÉTODOS

```
+ saca_un_valor( valor:double): boolean
```

```
metodo saca_un_valor (double valor)
    if valor > total_cuenta
        print ("No tenes dinero suficiente")
    else
        total_cuenta = total_cuenta - valor
        print ("Saqueo efectuado con éxito")
```

# MÉTODOS

```
+ transfere_para ( valor:double, num_cuenta_beneficiario: ent): boolean
```

```
metodo transfere_para (double valor, entero num_cuenta)
    if valor > total_cuenta
        print ("No tenes dinero suficiente para transferir")
    else
        total_su_cuenta = total_su_cuenta - valor
        print ("Saqueo efectuado con éxito")
```

# PARAMETROS

```
+ transfere_para ( valor:double, num_cuenta_beneficiario: ent): boolean
```

## PARAMETROS

Son valores que el metodo necesita.

En el ejemplo, para hacer una transferencia, necesitamos el valor para saber si tenes plata suficiente y la cuenta del beneficiario, para onde el dinero va.


# Herencia

Piense en un banco, con un empleado.

Tenemos un gerente, que también es un empleado, pero ejecuta más funciones que un empleado.

Ejemplo:

El gerente tiene más acceso al sistema que un empleado.

Necesitamos crear otra clase llamada gerente?

Y poner todos los métodos de nuevo?

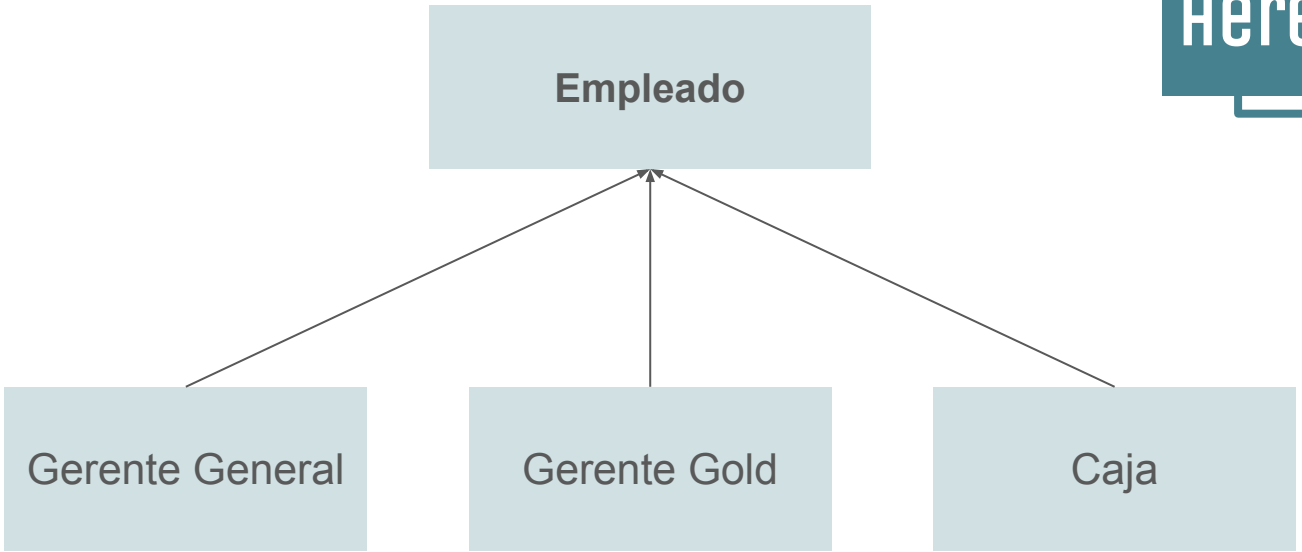
**puede empeorar? sí**

Si el banco reestructurar los cargos, y crear:

- **gerente general del banco, gerente de los clientes gold, y caja del banco.**

*Y todos ellos tienen en común varias funciones.*

# Herencia

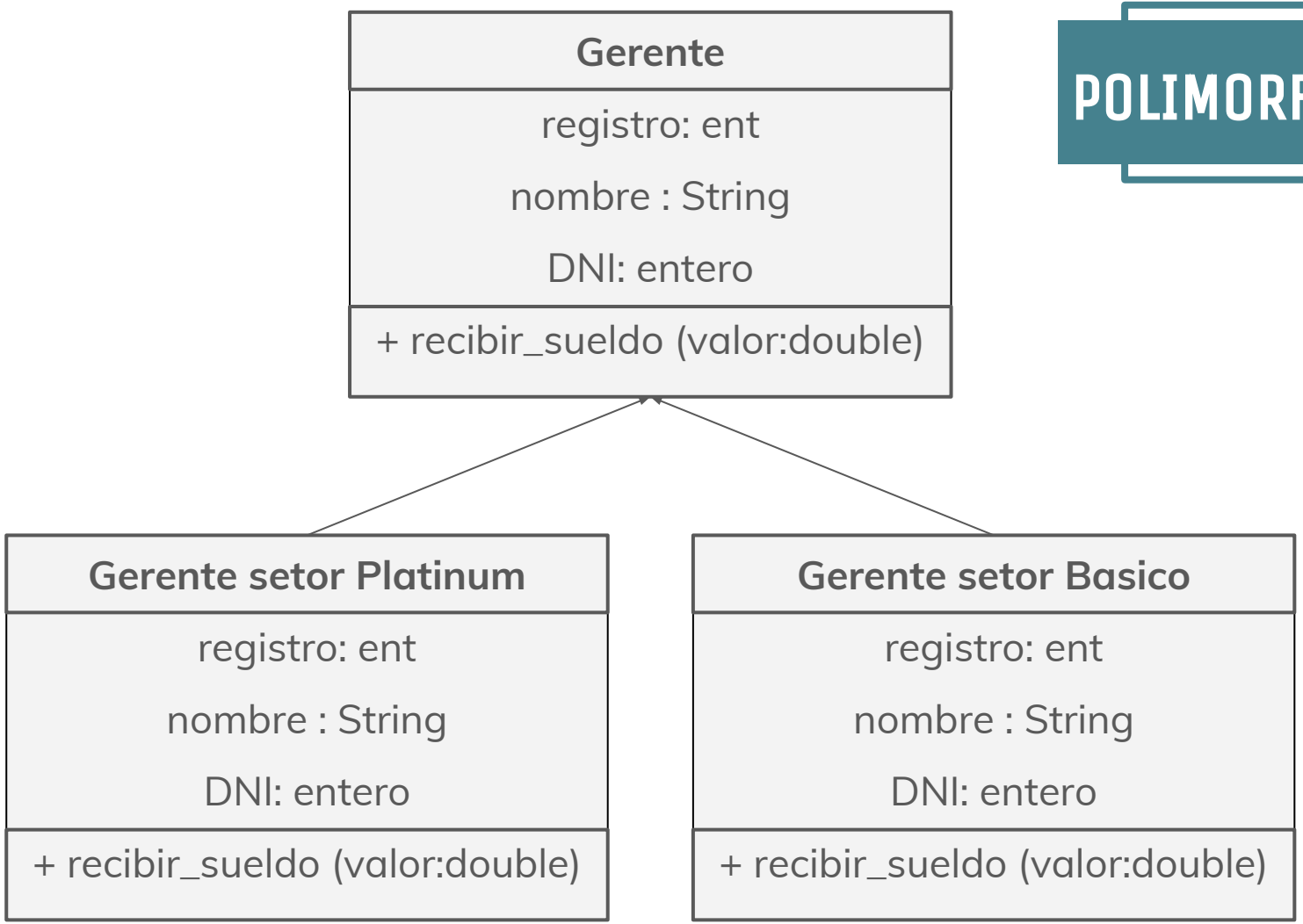



Observe la figura:
Todos cargos, son empleados.
Gerente general, Gerente Gold y Caja son empleados.

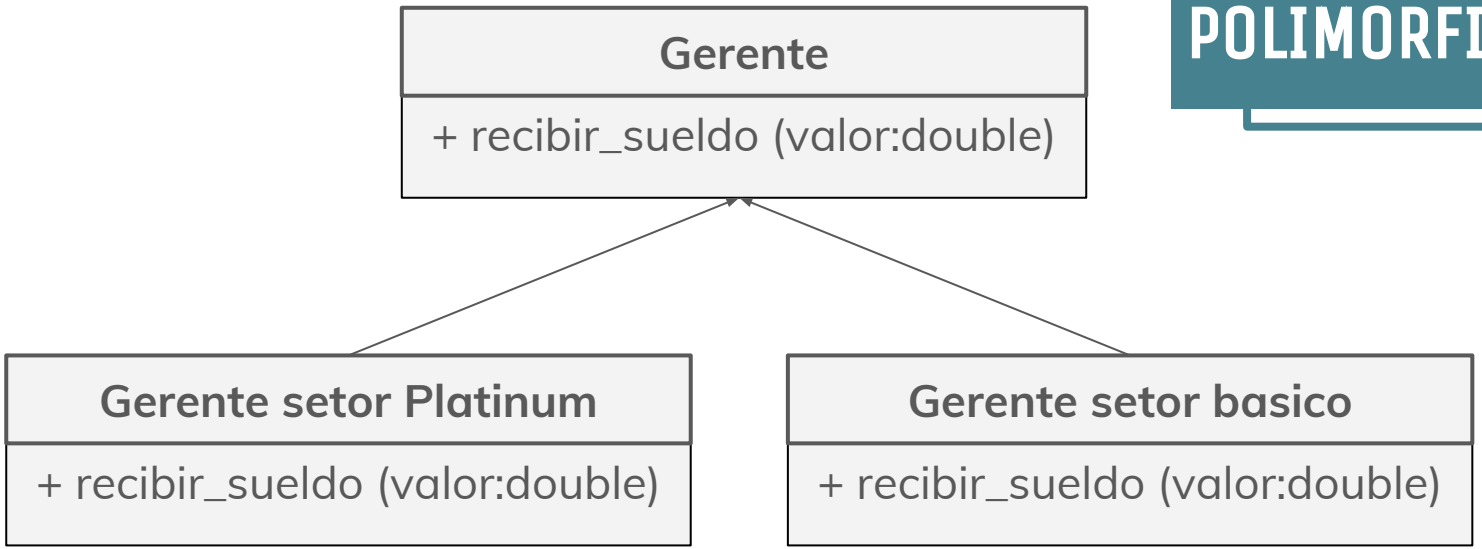
6

# Polimorfismo

# POLIMORFISMO



# POLIMORFISMO




El gerente Platinum recibe el **1%** sobre la cantidad de cuentas abiertas en el mes.
Es decir, si al final del mes la agencia tiene más de 100 cuentas abiertas el gerente gana un 1% más en su sueldo

El gerente Basico recibe el 2,5% sobre la cantidad de cuentas abiertas en el mes.
Es decir, si al final del mes la agencia tiene más de 100 cuentas abiertas el gerente gana un 2,5% más en su sueldo.

# POLIMORFISMO

| Gerente |
| --- |
| + recibir_sueldo (valor:double) |

O sea, tenemos un mismo método, pero con comportamiento diferente para cada Objecto.

Cada uno de los gerentes, que reciben los metodos por herencia, pueden hacer calculos distintos.

En el ejemplo uno recibe 1% y el otro 2,5%

7

# Aprendido y recomendaciones

# O QUE APRENDEMOS

- Clases
- Objectos
- Instancias
- Herencia
- Polimorfismo

# RECOMENDACIONES

- Clase Abstracta
- Interface
- Banco de Datos
- Basico de Java

# GRACIAS