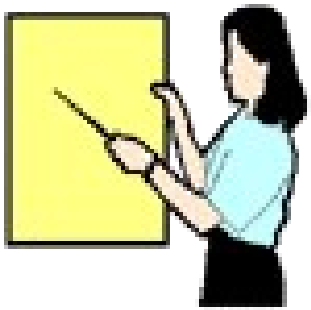
UD2: Estudio de los fundamentos de la programación orientada a objetos.

IES LOS SAUCES – BENAVENTE
CFGS DESARROLLO DE APLICACIONES WEB
PROGRAMACIÓN

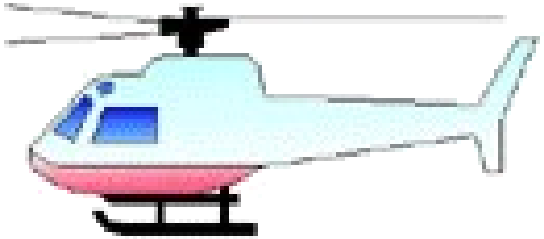
¿Qué es un objeto?

Un objeto es todo lo que tiene entidad, ya sea corporal o espiritual, natural o artificial, concreta, abstracta o virtual.

Antena Parabólica



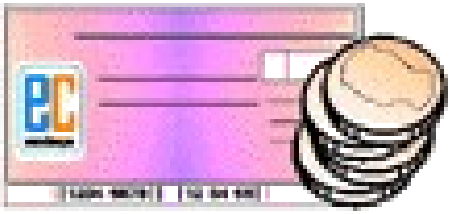
Objeto Profesor



Helicóptero

$a + bi$

Número Complejo



Cuenta Bancaria

Automóvil

¿Qué es un objeto?

Un objeto posee cualidades y capacidades.

CUALIDADES

¿Cómo es el objeto?

¿Qué tiene el objeto?

CAPACIDADES

¿Qué puede hacer el objeto?

Un objeto tiene identidad, estado y comportamiento.

¿Qué es un objeto?

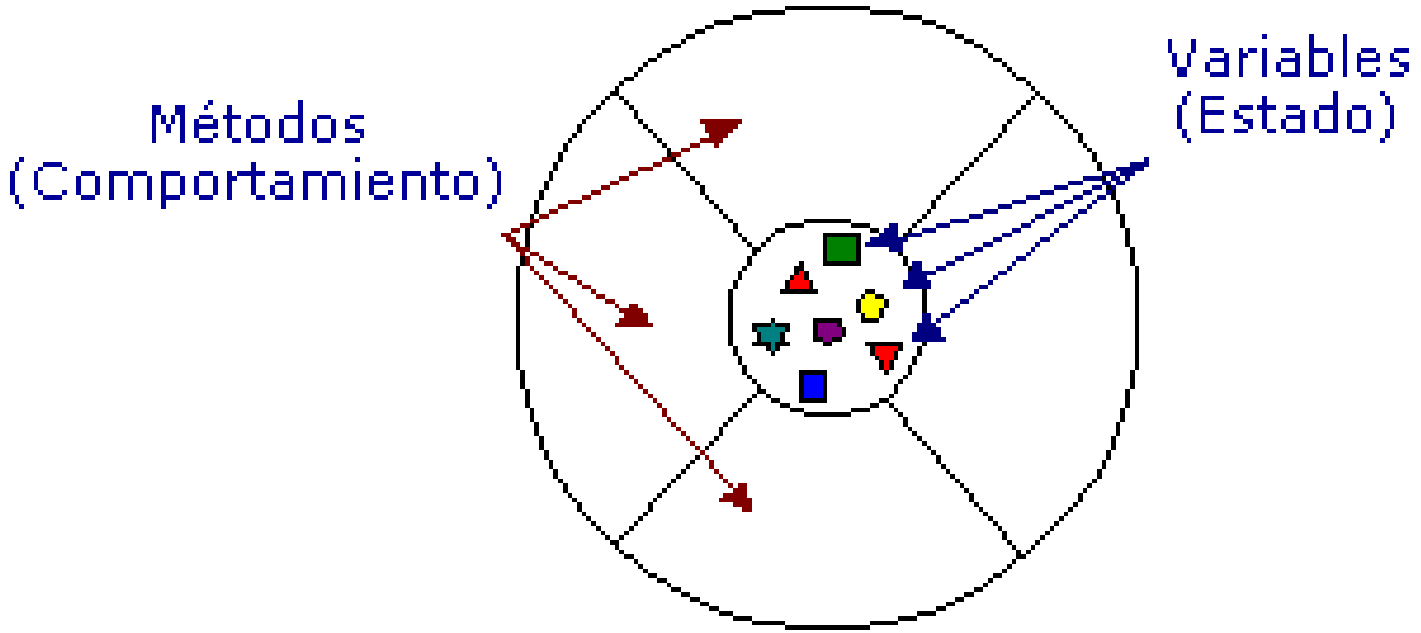
La IDENTIDAD es el nombre que se le ha dado al objeto.

El ESTADO hace referencia a las cualidades (propiedades) del objeto.

El COMPORTAMIENTO hace referencia a las capacidades del objeto.

¿Qué es un objeto?

En el paradigma de programación orientada a objetos, un objeto es una unidad dentro de un programa que consta de un estado y un comportamiento.



¿Qué es una clase?

Una clase es la definición del estado y del comportamiento de un tipo de objeto concreto.

El estado de un objeto se representa mediante los atributos o variables miembro o variables de instancia.

El comportamiento de un objeto se representa mediante los métodos o funciones miembro.

¿Qué es una clase?

Representación gráfica de una clase (UML).

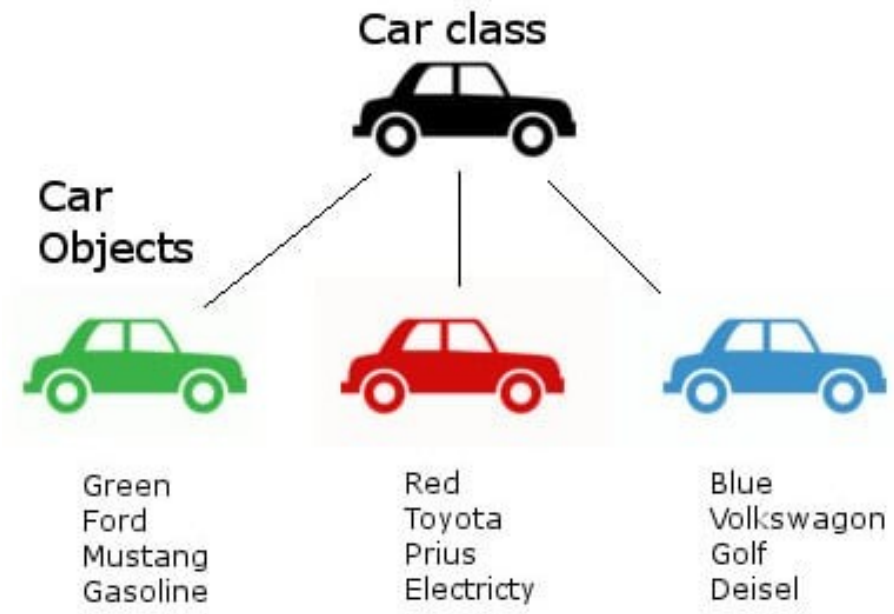
Cuenta
-codigo: String -titular: String -saldo: float
+Cuenta(codigo: String, titular: String, saldo: float) +getCodigo(): String +getTitular(): String +getSaldo(): float +setCodigo(codigo: String): void +setTitular(titular: String): void +setSaldo(saldo: float): void +ingresar(cantidad: float): void +reintegrar(cantidad: float): void

Atributos

Métodos

Clases y objetos

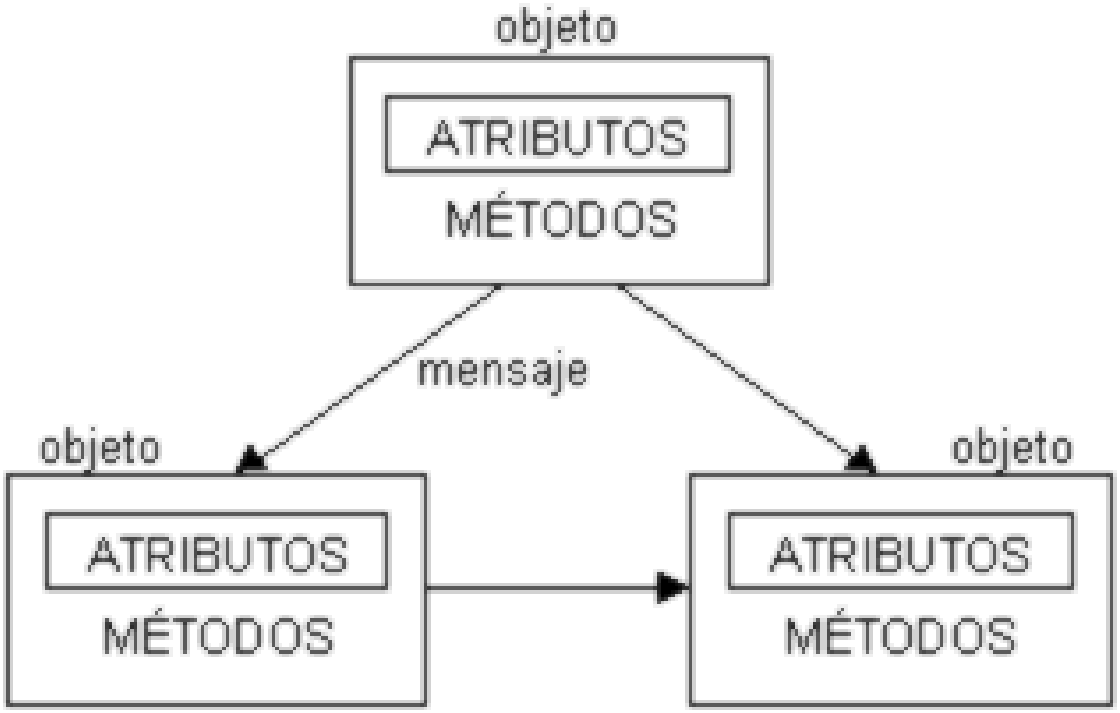
Una clase es la implementación de un tipo de dato.



Un objeto es una instancia de una clase.

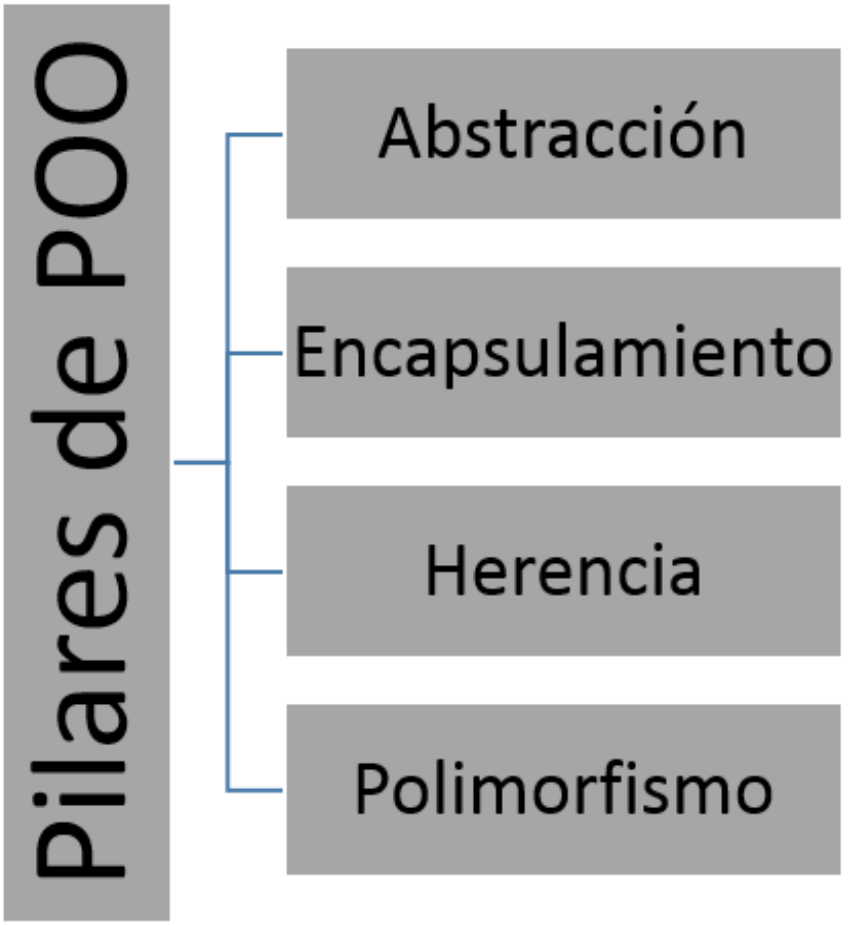
Clases y objetos

El acceso a los objetos se hace mediante mensajes.



Programación orientada a objetos

La programación orientada a objetos (POO) es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos.



Programación orientada a objetos

ABSTRACCIÓN

Consiste en captar las características esenciales de un objeto, así como su comportamiento.

ENCAPSULAMIENTO

Consiste en agrupar atributos y métodos en una clase.

Nos permite ocultar el estado del objeto.

HERENCIA

Permite crear jerarquía de clases relacionadas.

Es la relación existente entre dos clases: derivada y base.

POLIMORFISMO

Propiedad por la que es posible enviar mensajes iguales a objetos de tipos distintos.

Declaración de una clase

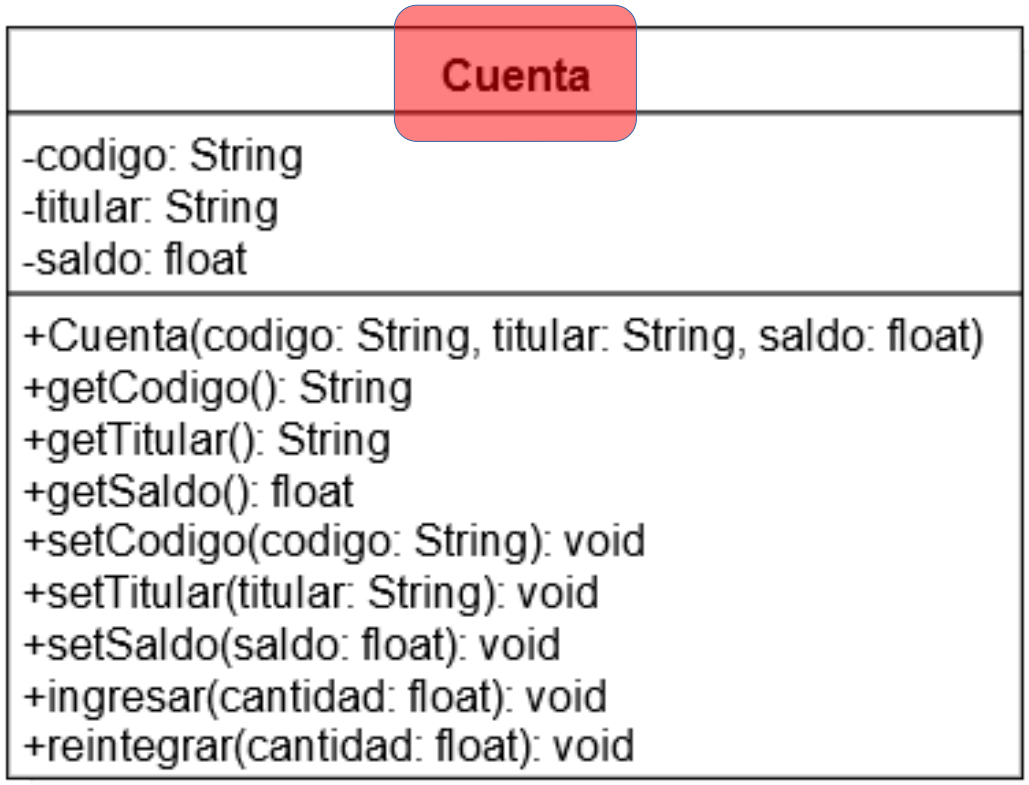
El nombre de una clase deberá ser un sustantivo en notación UpperCamelCase.

```
public class NombreClase{
    declaración de atributos

    constructores

    declaración de métodos
}
```

Declaración de una clase




```
public class Cuenta{


}
```

Declaración de atributos

- **Los atributos se declararán con un identificador y tipo (primitivo o referenciado).**
- **Los nombres de los atributos se escribirán en notación lowerCamelCase.**
- **Cuando el atributo sea una constante se escribirá con letras mayúsculas.**
- **La elección del nombre del atributo debe ser mnemónico.**

Declaración de atributos

- **Los atributos son accesibles desde dentro de la clase.**
- **El acceso desde otras clases se puede limitar mediante los modificadores de acceso.**

public	**accesibles desde cualquier clase**
protected	**accesibles en hijas y en el paquete**
private	**accesibles solamente desde la clase**
sin modificador	**accesibles en el paquete**

- **El proceso de declarar atributos con el modificador de acceso private se denomina ocultamiento de datos.**

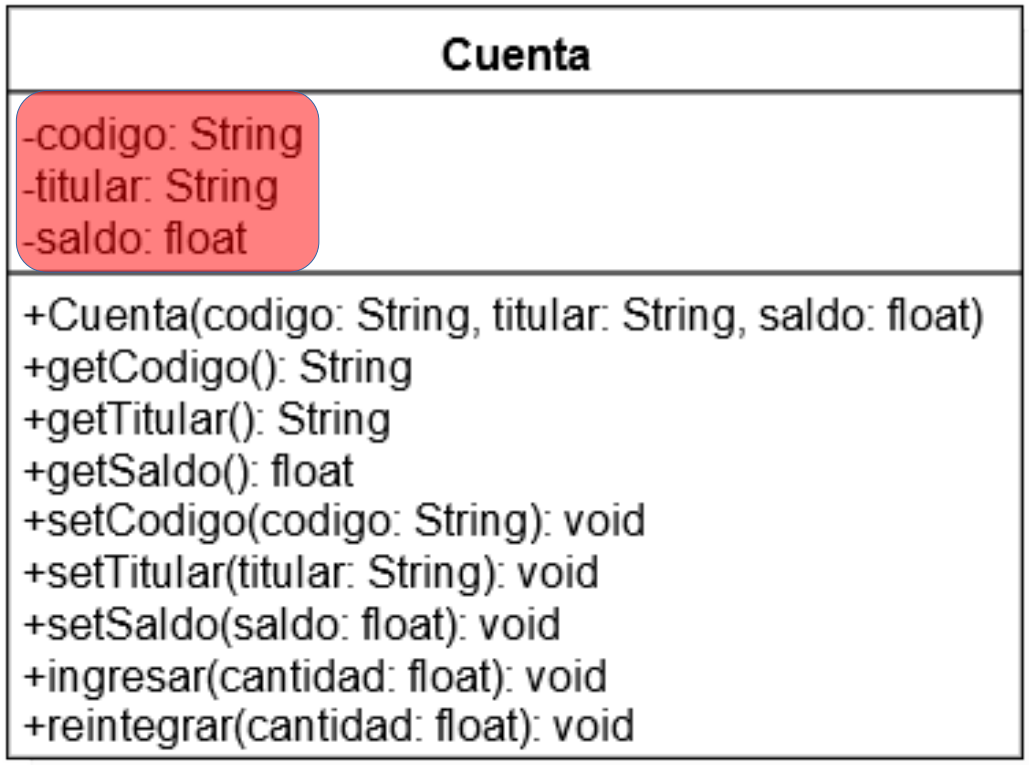
Se aconseja declarar los atributos como private.

Declaración de atributos

- **Los atributos podrán ser de instancia o de clase (modificador static).**
- **Los atributos de instancia representan el estado particular de un objeto.**
- **Los atributos de clase representan el estado compartido por todos los objetos de una clase.**
- **Para acceder a un atributo de clase no es necesario instanciar la clase (crear un objeto).**

Declaración de atributos

```
[private | protected | public] [static] [final] tipo atributo;
```




```
public class Cuenta{
  // atributos de la clase
  private String codigo;
  private String titular;
  private float saldo;
}
```

Declaración de métodos

- **Los métodos definen el comportamiento de un objeto de una clase dada.**
- **Un método define la secuencia de instrucciones que se ejecuta para llevar a cabo una operación.**
- **Un método nos dice cómo hemos de usar los objetos de una clase.**
- **Estructura de un método: cabecera y cuerpo.**

Cabecera de un método

- **Determina su interfaz**

Cuerpo de un método

- **Define su implementación**

Declaración de métodos (cabecera)

modificadores tipo nombre (parámetros)

- **Los modificadores de acceso indican donde se puede utilizar el método.**
- **Los métodos son accesibles desde dentro de la clase.**
- **El acceso desde otras clases se puede limitar mediante los modificadores de acceso.**

public	**accesibles desde cualquier clase**
protected	**accesibles en hijas y en el paquete**
private	**accesibles solamente desde la clase**
sin modificador	**accesibles en el paquete**

Declaración de métodos (cabecera)

modificadores tipo nombre (parámetros)

- **Los métodos podrán ser de instancia o de clase (modificador static).**
- **Los métodos de clase solamente tendrán acceso a atributos y métodos de clase.**
- **Para acceder a un método de clase no es necesario instanciar la clase (crear un objeto).**

Declaración de métodos (cabecera)

modificadores tipo nombre (parámetros)

- **El tipo devuelto indica de que tipo es la salida del método.**
- **El tipo devuelto podrá ser: tipo primitivo, tipo referenciado o void (se utiliza cuando el método no devuelve ningún valor).**
- **El nombre del método debe ser un identificador válido en Java en notación lowerCamelCase.**
- **Puede haber varios métodos con el mismo nombre (sobrecarga).**
- **Se deberán diferenciar en el tipo/número de parámetros.**

Declaración de métodos (cabecera)

modificadores tipo nombre (parámetros)

- **Los parámetros son las entradas que necesita el método para realizar la operación que tiene asignada.**
- **Un método puede no necesitar parámetros.**
- **Los parámetros definidos en la cabecera de un método se denominan parámetros formales.**
- **En Java, todos los parámetros se pasan por valor.**
- **El método utiliza una copia local de los parámetros.**

Declaración de métodos (cabecera)

modificadores tipo nombre (parámetros)

- **Para cada parámetro deberemos indicar su tipo y su identificador.**
- **Si un método tiene varios parámetros, los distintos parámetros se separan por comas.**
- **El orden de los parámetros es relevante.**

```
[private | protected | public] [static] tipo método (parámetros)
```

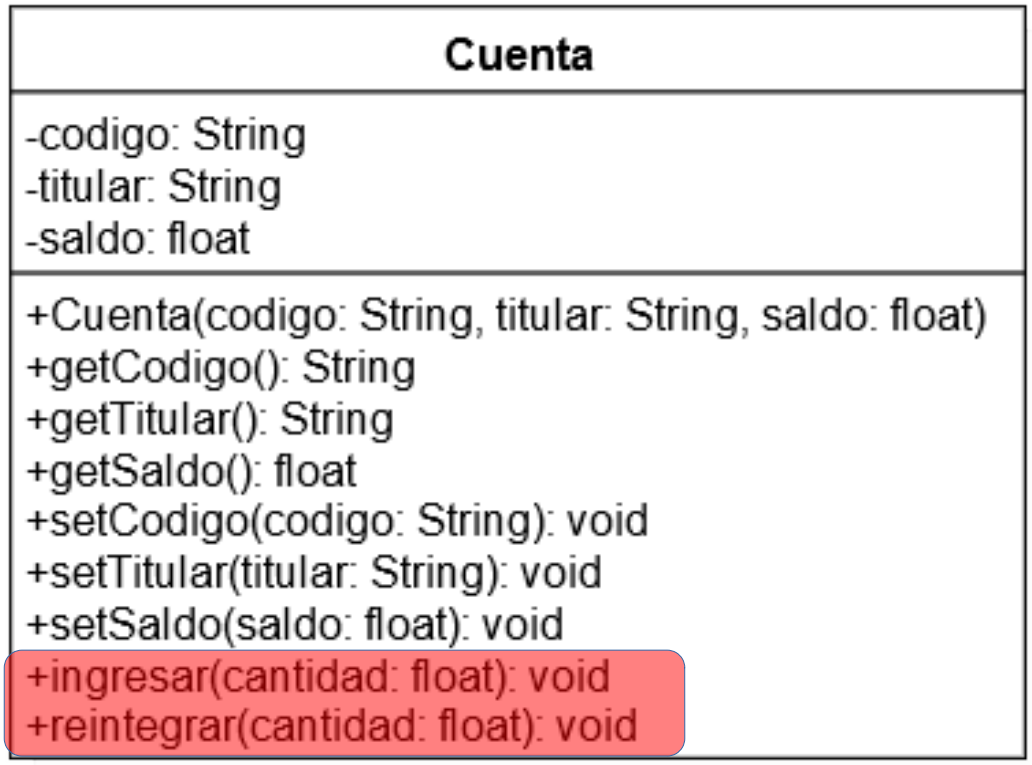
Declaración de métodos (cuerpo)

- **En el cuerpo del método se implementa el algoritmo necesario para realizar la tarea de la que es responsable el método.**
- **El cuerpo de un método debe estar delimitado por llaves.**
- **Cuando un método devuelve un valor, la implementación del mismo debe terminar con una sentencia return.**

```
return expresión;
```

- **Donde el tipo de la expresión debe coincidir con el tipo declarado en la cabecera del método.**

Declaración de métodos




```
public class Cuenta{
   // métodos de la clase
   public void ingresar(float cantidad){
      saldo+=cantidad;
   }
   public void reintegrar(float cantidad)
   {
      saldo-=cantidad;
   }
}
```

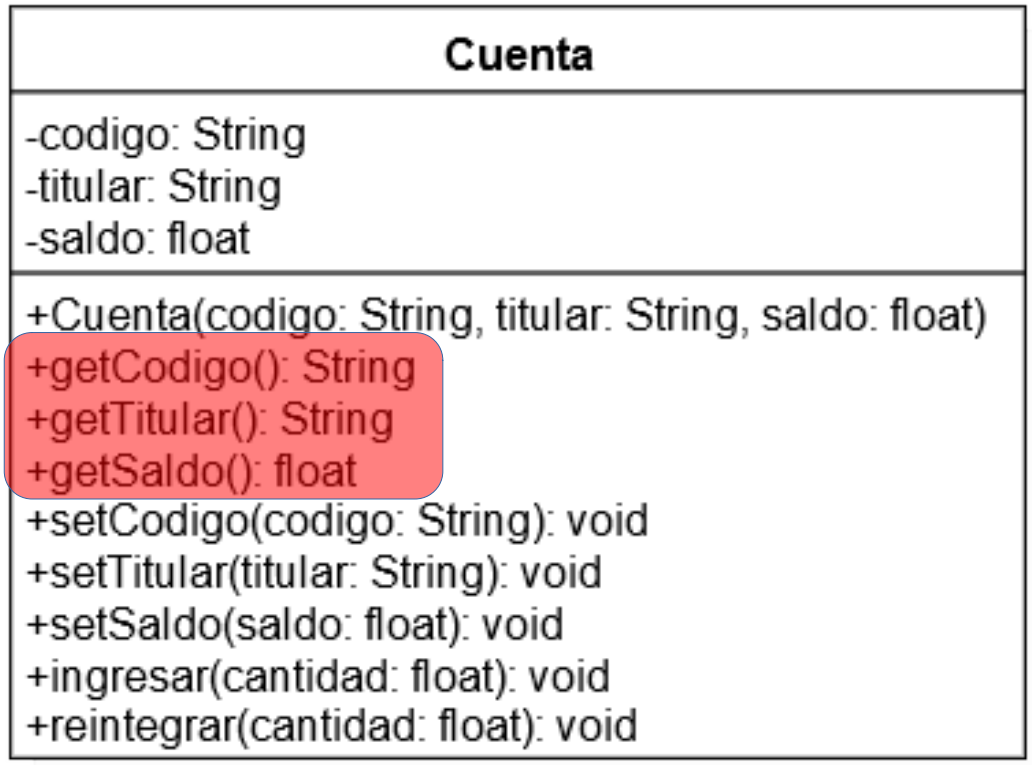
Métodos get y set

- **Son métodos utilizados para acceder a los atributos de la clase declarados con el modificador de acceso private.**
- **Se declarará un método set y un método get para cada atributo al que queramos permitir el acceso desde fuera de la clase.**
- **Deberán ser declarados con el modificador de acceso public.**

Métodos get

- **Un método get permitirá consultar el valor de un atributo.**
- **El nombre de un método get será get seguido del nombre del atributo que se pretende consultar.**
- **El tipo devuelto por un método get coincidirá con el tipo del atributo.**
- **Un método get no recibe parámetros.**

Métodos get

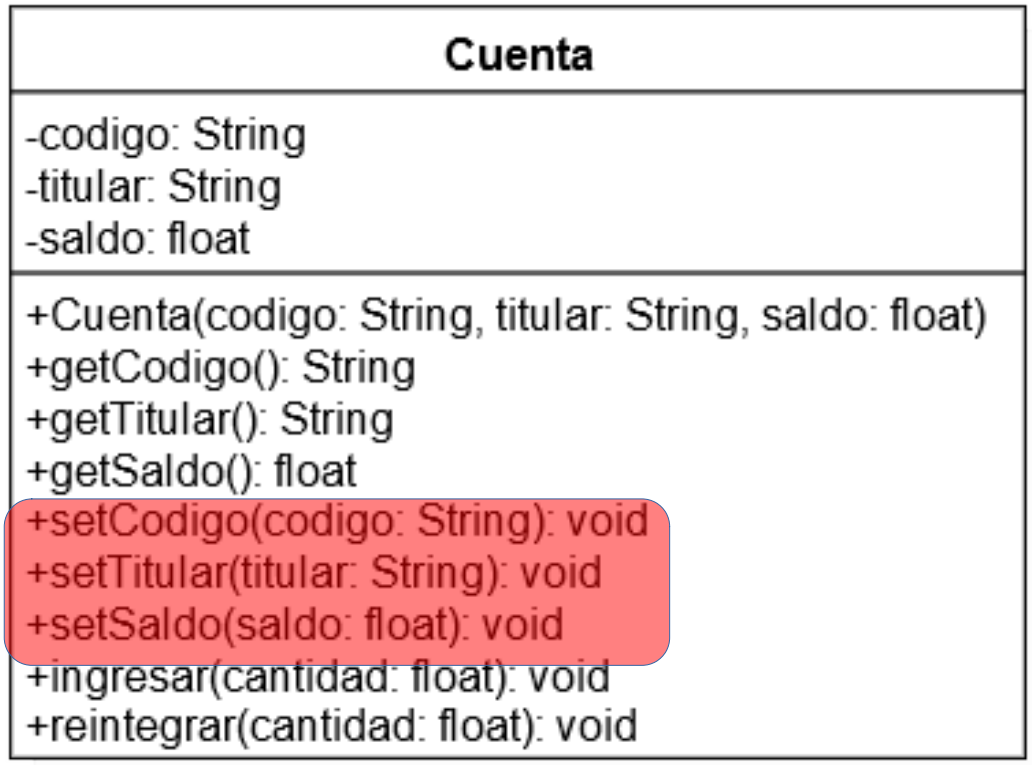



```
public class Cuenta{
   // métodos get
   public String getCodigo(){
      return codigo;
   }
   public String getTitular(){
      return titular;
   }
   public float getSaldo(){
      return saldo;
   }
}
```

Métodos set

- **Un método set permitirá modificar el valor de un atributo.**
- **El nombre de un método set será set seguido del nombre del atributo que se pretende modificar.**
- **El tipo devuelto por un método set será void.**
- **Un método set recibirá un parámetro del mismo tipo del atributo que se pretende modificar.**

Métodos set




```
public class Cuenta{

    // métodos set

    public void setCodigo(String codigo){

        this.codigo=codigo;

    }

    public void setTitular(String titular){

        this.titular=titular;

    }

    public void setSaldo(float saldo){

        this.saldo=saldo;

    }

}
```

Constructores

- **Son métodos especiales que permiten inicializar los atributos de las instancias de una clase (objetos).**
- **Suelen llevar el modificador de acceso public.**
- **Un constructor nunca devuelve nada.**

El nombre del constructor debe coincidir con el nombre de la clase.

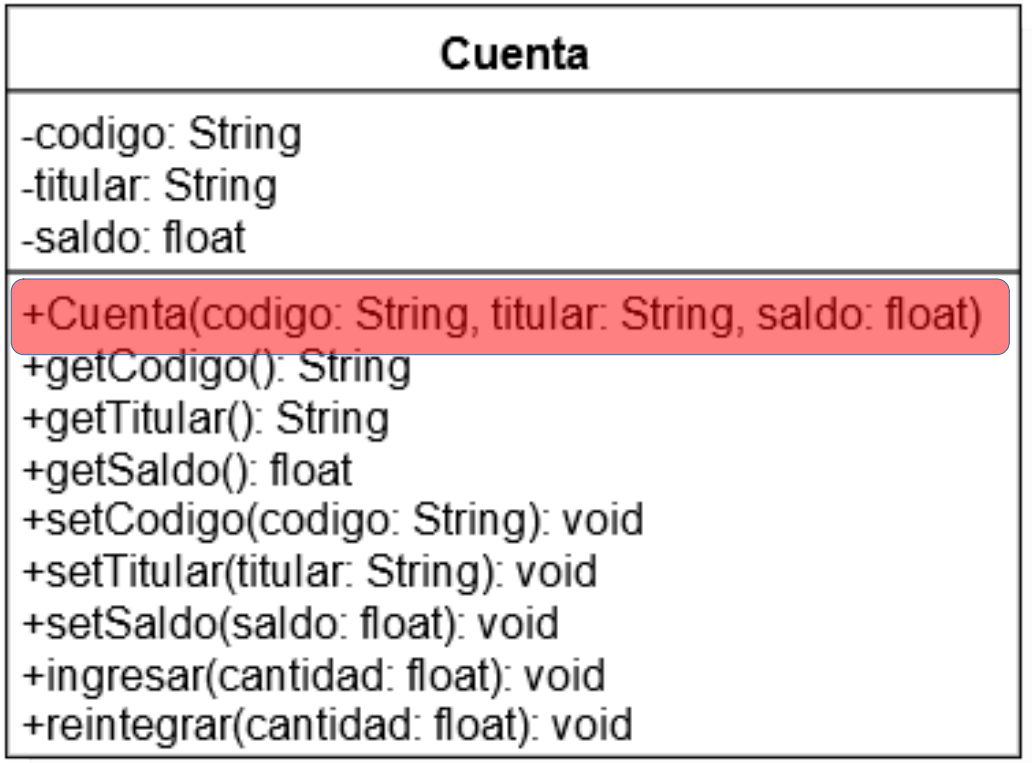
Constructores

- **Una clase puede tener cero, uno o más constructores (sobrecarga de constructores)**
- **Se diferenciarán en el tipo/número de parámetros que reciben.**
- **Si no tiene ningún constructor habrá uno implícito.**
- **El constructor implícito inicializa los atributos a sus valores por defecto (0 los numéricos, false para boolean y null para los referenciados).**

Constructores

```
public class Cuenta{
   // constructor
   public Cuenta(String codigo, String titular, float saldo){
      this.codigo=codigo;
      this.titular=titular;
      this.saldo=saldo;
   }


}
```



Uso de objetos

- **Una vez declarada una clase, el nombre de la clase se convierte en un nuevo tipo de dato referenciado.**
- **La declaración de una variable de tipo referenciado no crea el objeto.**
- **La declaración de una variable de tipo referenciado reserva una zona de memoria donde se almacenará una referencia a un objeto del tipo especificado.**

Uso de objetos

Para declarar una variable de tipo referenciado:

```
[modificador] NombreClase identificador;
```

```
public class TestCuenta{
    public static void main (String[] args){
        // declaración de un objeto de la clase Cuenta
        Cuenta objetoCuenta;
    }
}
```

Uso de objetos

- **Una vez declarada la variable podremos asignarle un objeto ya creado o crear un nuevo objeto.**
- **Cuando se crea un objeto se crea una instancia de una clase.**
- **Para la creación de un objeto utilizaremos el operador new.**
- **El operador new irá seguido de un constructor de la clase a la que pertenezca el objeto que pretendemos crear.**

Uso de objetos

- El operador **new** reservará memoria para guardar los datos del objeto.
- El operador **new** devolverá una referencia a esa memoria.
- El operador **new** llamará al constructor de la clase.
- El constructor se encargará de inicializar el objeto.

Para crear un objeto:

```
new Constructor(parámetros);
```

```
public class TestCuenta{

    public static void main (String[] args){

        Cuenta objetoCuenta;

       // creación de un objeto de la clase Cuenta
       objetoCuenta=new Cuenta(“A1”,”TITULAR”,1.0F);
    }

}
```

Uso de objetos

- **Una vez creado el objeto ya podremos acceder a sus miembros (atributos y métodos).**
- **El acceso a los atributos y métodos estará marcado por los modificadores de acceso con los que hayan sido declarados.**
- **Normalmente los atributos se habrán declarado con el modificador de acceso private, que impide el acceso a los mismos desde fuera del objeto.**
- **Para el acceso a los atributos private se habrán declarado los métodos set y get.**

Uso de objetos

- **El acceso a los atributos de un objeto se realiza a través del operador punto (.), que separa al identificador del objeto del identificador del atributo.**

objeto.atributo

- **El acceso a los atributos estáticos de un objeto se realiza a través del operador punto (.), que separa el nombre de la clase del identificador del atributo.**

NombreClase.atributo

Uso de objetos

- **El acceso a los métodos de un objeto se realiza a través del operador punto (.), que separa al identificador del objeto del identificador del método.**

objeto.metodo(parámetros)

- **El acceso a los métodos estáticos de un objeto se realiza a través del operador punto (.), que separa el nombre de la clase del identificador del método.**

NombreClase.metodo(parámetros)

Uso de objetos

```
public class TestCuenta{

    public static void main (String[] args){

        Cuenta objetoCuenta;

        objetoCuenta=new Cuenta(“A1”,”TITULAR”,1.0F);


        float saldo;
        saldo=objetoCuenta.getSaldo();
        System.out.println(“Saldo: “+saldo);
        objetoCuenta.setSaldo(100.0F);
        saldo=objetoCuenta.getSaldo();
        System.out.println(“Saldo: “+saldo);
    }

}
```

Uso de objetos

- **Para acceder a un atributo dentro del mismo objeto lo haremos utilizando el nombre del atributo.**
- **La palabra reservada this hace referencia al objeto actual.**
- **Deberemos utilizar this cuando un atributo esté oculto por una declaración de variable o parámetro.**

this.atributo

- **Para acceder a un método dentro del mismo objeto lo haremos utilizando el nombre del método acompañado de los parámetros recibidos por dicho método.**

Uso de objetos

- **Java nos permite crear objetos y no tener que preocuparnos por su destrucción.**
- **El entorno de ejecución Java elimina los objetos cuando ya no se están utilizando.**
- **Este proceso se denomina recolección de basura.**
- **Un objeto es candidato a la recolección de basura cuando no hay referencias a él.**
- **Podemos eliminar la referencia a un objeto asignando a la variable el valor null.**