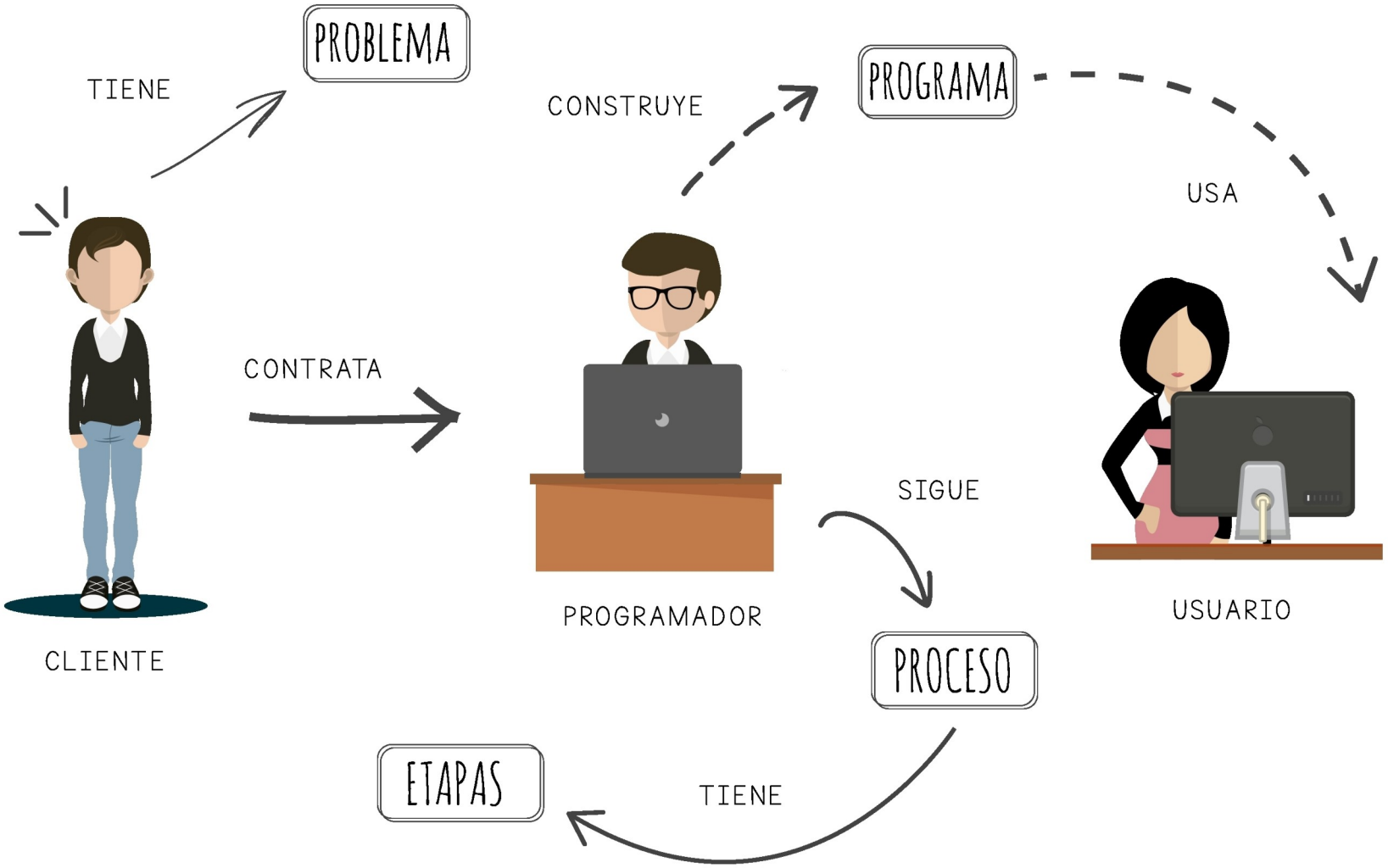
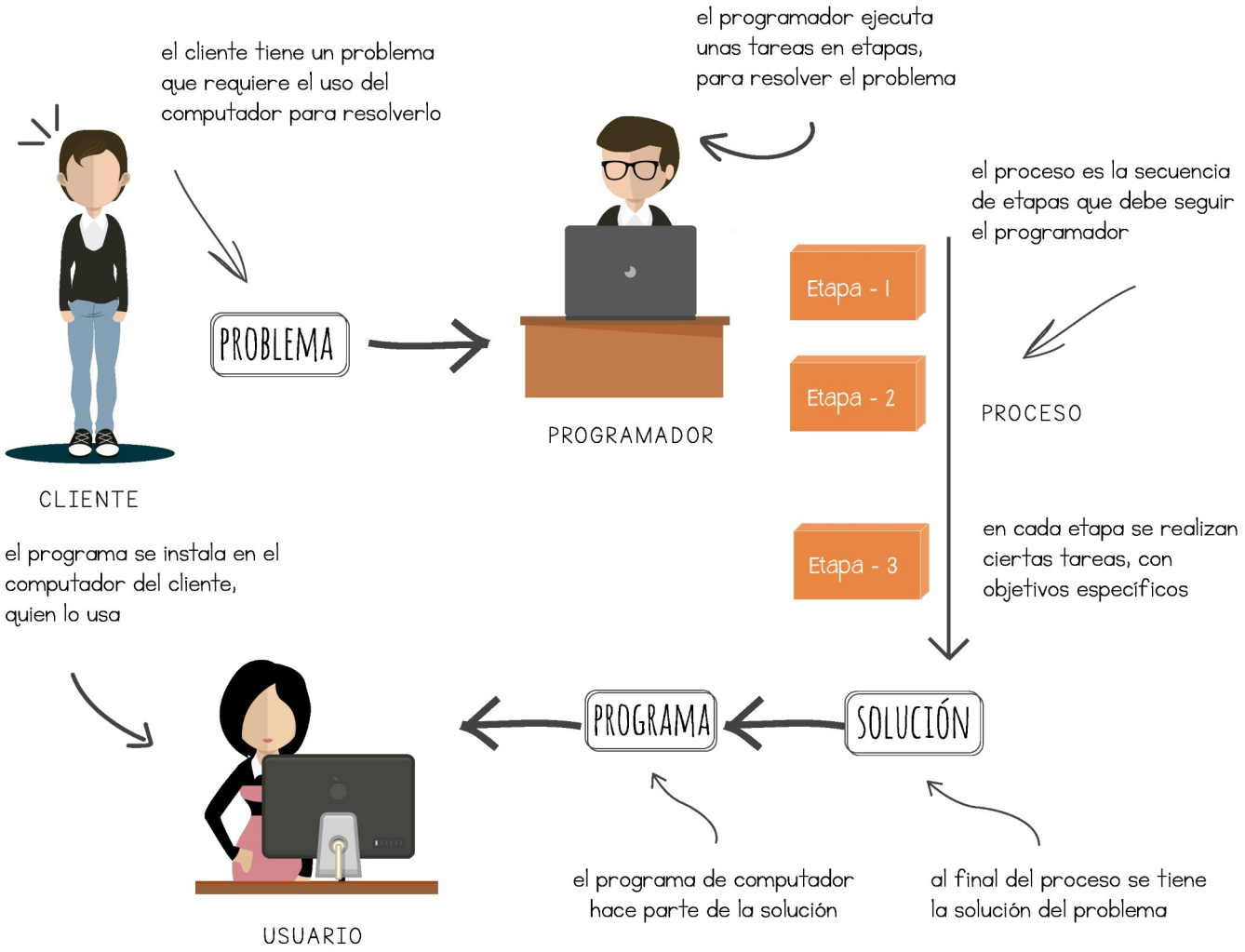
UD4: Desarrollo de una aplicación basada en POO. Utilización de clases predefinidas.

IES LOS SAUCES – BENAVENTE
CFGS DESARROLLO DE APLICACIONES WEB
PROGRAMACIÓN

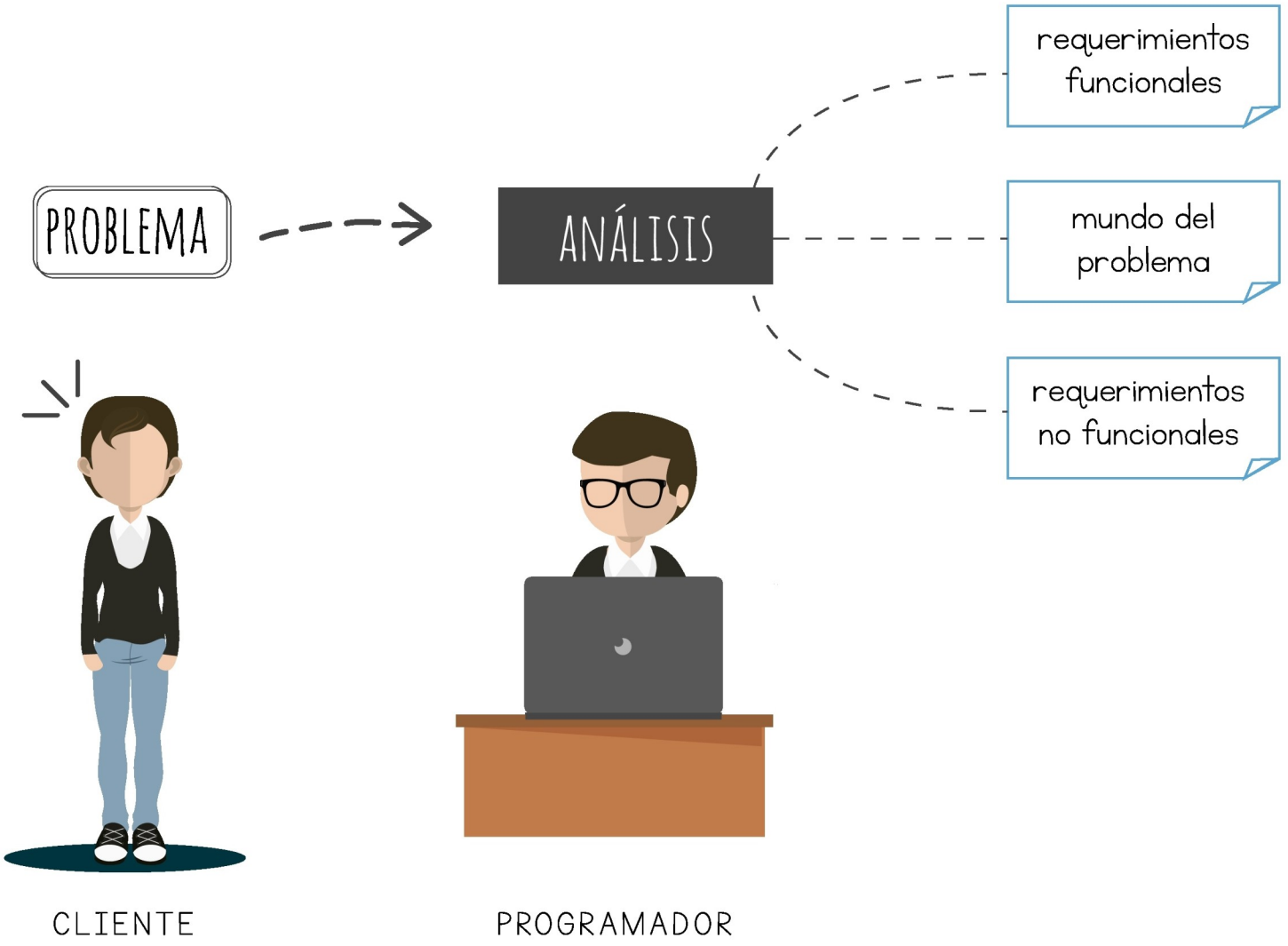
Desarrollo de una aplicación



Desarrollo de una aplicación



Desarrollo de una aplicación



Análisis

Queremos realizar una aplicación que nos permita abrir y mantener una cuenta bancaria.

Para ello deberemos realizar un análisis.

Deberemos conocer lo que quiere el cliente e identificar los requerimientos de la aplicación (funcionales y no funcionales).

Además de identificar los requerimientos también deberemos identificar las entidades del problema.

Desarrollo de una aplicación

el programador cuenta con un conjunto de herramientas y lenguajes para construir la solución

PROGRAMADOR

HERRAMIENTAS Y LENGUAJES

PROBLEMA

análisis del probema

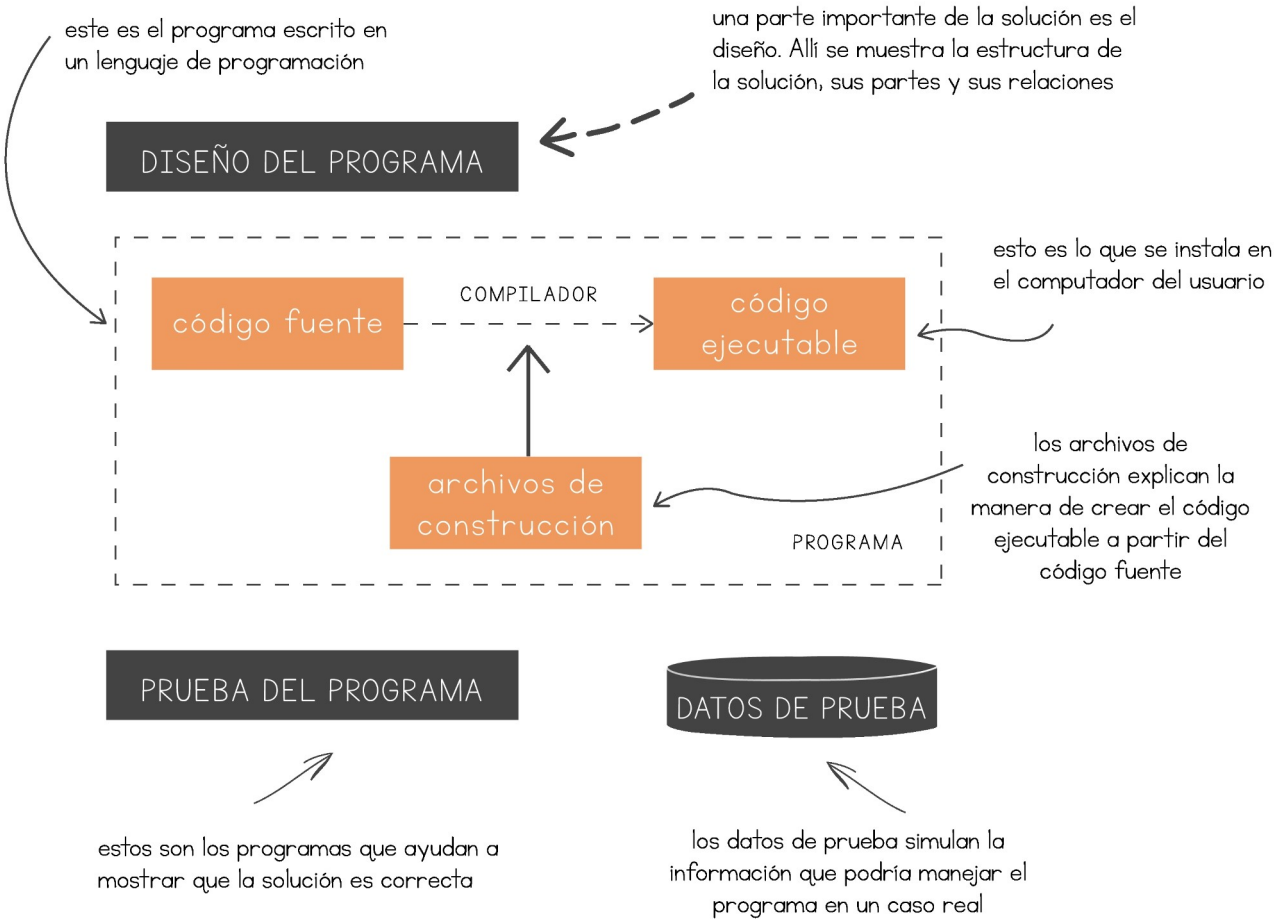
diseño de la solución

construcción de la solución

proceso

SOLUCIÓN

Desarrollo de una aplicación



Diseño

Una vez realizado el análisis deberemos proponer un diseño con la solución.

Para realizar el diseño utilizaremos UML (Unified Modeling Language).

Nos centraremos en dos diagramas:

Diagramas de casos de uso.

Diagramas de clases.

Casos de uso

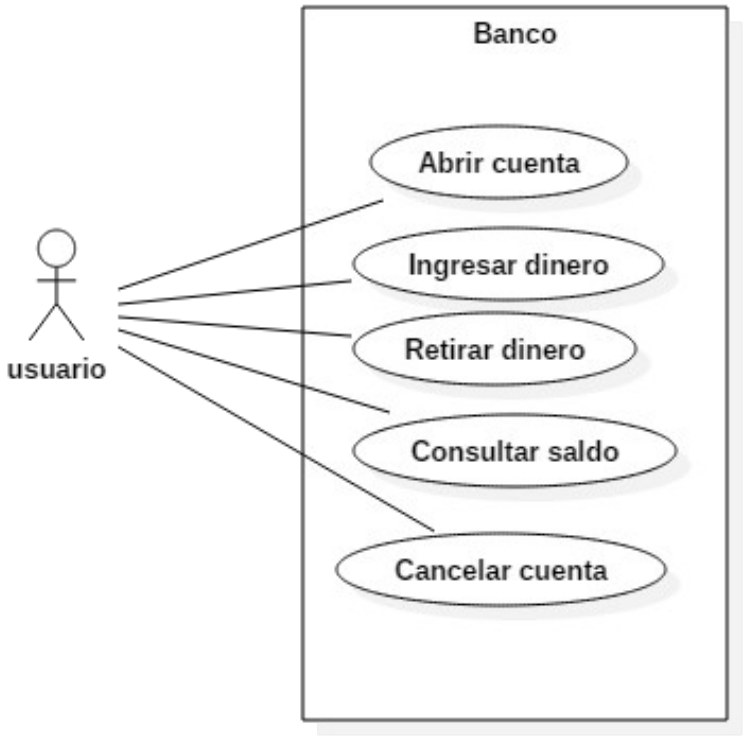
Un caso de uso es la representación de la secuencia de acciones que una aplicación puede ejecutar.

Un caso de uso es el detalle de un requerimiento funcional.

Un caso de uso especifica qué hace la aplicación pero no cómo se hace.

Diagrama de casos de uso

Un diagrama de casos de uso se utiliza para modelar los requerimientos funcionales del sistema.



Especifiación de casos de uso

Plantilla para una especificación de caso de uso:

Caso de uso: *nombre del caso de uso*
Id: identificador del caso de uso
Breve descripción: Descripción breve, objetivo del caso de uso.
Actores principales: Actores que activan el caso de uso.
Actores secundarios: Actores que interactúan con el caso de uso una vez activado.
Precondiciones: Condiciones que deben ser verdaderas antes de que el caso de uso pueda ejecutarse; restricciones en el estado del sistema.
Flujo principal: Pasos del caso de uso.
Postcondiciones: Condiciones que deben ser verdaderas cuando el caso de uso ha terminado.
Flujos alternativos: Describen las desviaciones complejas del flujo principal. Se nombran y habría que describirlos en otros casos de uso.

Especifiación de casos de uso

Caso de uso: *Retirar dinero*

Id: 3

Breve descripción:
El usuario retira dinero de una cuenta.

Actores principales:
Usuario.

Actores secundarios:
Ninguno.

Precondiciones:
1. Tener abierta una cuenta.

Flujo principal:
1. El usuario introduce el código de cuenta.
2. El sistema comprueba que exista el código de cuenta.
   Si el código de cuenta no existe el sistema muestra un mensaje de error.
3. El usuario introduce la cantidad a retirar.
4. El sistema comprueba que el saldo de la cuenta sea superior o igual a la cantidad a retirar.
   Si el saldo de la cuenta es inferior a la cantidad a retirar el sistema muestra un mensaje de error.
5. El sistema actualiza el saldo de la cuenta decrementándolo en la cantidad retirada.
6. El sistema muestra un mensaje indicando que la operación se ha realizado con éxito.

Postcondiciones:
1.

Flujos alternativos:
1.

Entidades

Una vez identificados los requerimientos deberemos identificar las entidades de la aplicación y sus características.

Entidad	Características	Tipo de dato
Cuenta	Código	Cadena de caracteres
	Titular	Cadena de caracteres
	Saldo	Real

Clases

A partir de las entidades identificamos las clases y sus atributos.

Clase	Atributos	Tipo de dato
Cuenta	Código	Cadena de caracteres
	Titular	Cadena de caracteres
	Saldo	Real

Diagrama de clases

Una vez identificadas las clases, se modela la solución de la aplicación utilizando el diagrama de clases de UML.

En el diagrama de clases se dibuja una caja por cada clase compuesta de nombre, atributos y métodos.

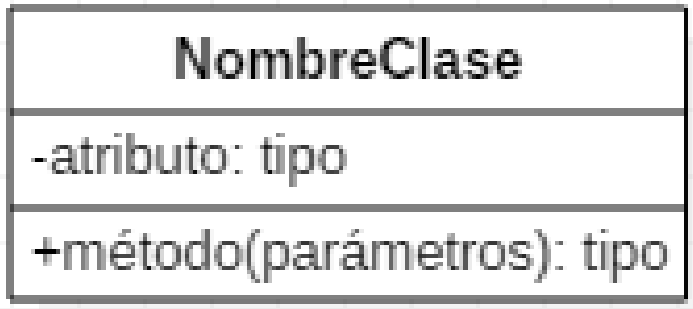
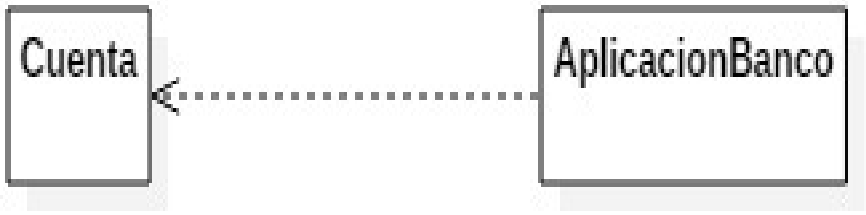


Diagrama de clases

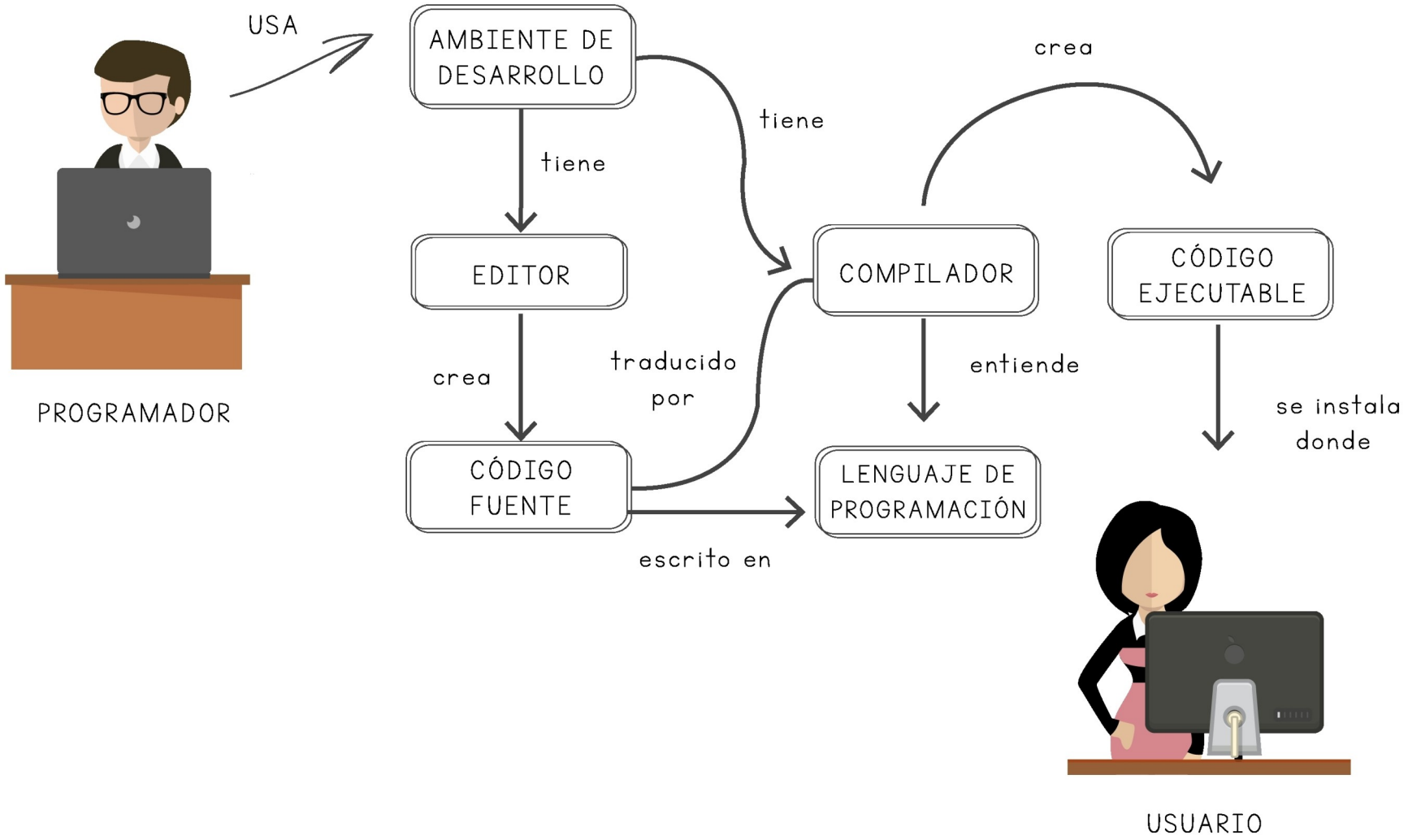
Las clases tienen relaciones con otras clases y estas relaciones también se incluyen en el diagrama de clases.

Nuestra aplicación estará formada por dos clases que tiene una relación de uso.

La clase AplicacionBanco usa la clase Cuenta.



Desarrollo de una aplicación



Codificación

En la última fase escribiremos el código fuente de la aplicación que se diseño en la etapa anterior.

En la fase de codificación utilizaremos un IDE (Integrated Development Environment).

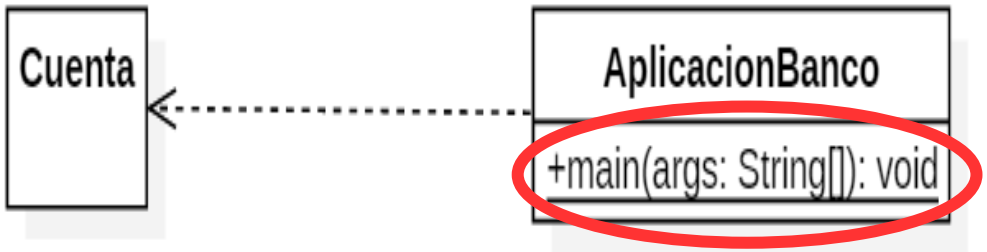
En nuestro caso utilizaremos Apache NetBeans por ser libre.

Estructura de una aplicación

Toda aplicación Java estará formada por una o más clases.

Una de las clases tendrá definido el método estático main.

El método main será el punto de entrada y de salida de la aplicación.



Estructura de una aplicación

Cada clase la almacenaremos en un fichero con el nombre de la clase y extensión .java.

En un fichero se puede incluir más de una clase pero sólo una pública.

Si en un fichero incluimos más de una clase el nombre debe ser el de la clase pública.

Al compilar un fichero fuente (.java) obtendremos tantos ficheros de clase (.class) como clases estén incluidas en él.

package

En Java las clases se organizan en paquetes.

Un paquete es un conjunto de clases, lógicamente relacionadas entre sí, agrupadas bajo un nombre.

El paquete al que pertenece una clase se indica al principio del fichero mediante la palabra reservada package seguida del nombre del paquete.

```
package paquete;
```

El nombre del paquete es un identificador Java. Se escribirá en minúsculas.

package

El nombre del paquete deberá coincidir con el nombre del directorio en el que se encuentra la clase.

Un paquete puede contener otros paquetes.

Los paquetes se pueden organizar de manera jerárquica en subpaquetes.

```
package paquete.subpaquete;
```

Si no indicamos a que paquete pertenece la clase, pertenecerá al paquete predeterminado.

import

Una aplicación puede utilizar clases de distintos paquetes.

Para utilizar una clase de otro paquete debemos incluir la instrucción import.

```
import paquete.Clase;
```

Podemos importar todas las clases de un paquete.

```
import paquete.*;
```

Si no utilizamos la instrucción import deberemos utilizar el nombre completo de la clase paquete.Clase.

class

Al declarar una clase indicaremos el nivel de protección de la misma.

Dos niveles de protección: paquete y pública.

Una clase con nivel de protección de paquete sólo puede ser utilizada por las clases de su paquete.

Una clase pública puede ser utilizada por cualquier otra clase de otro paquete.

Por defecto una clase tiene el nivel de protección de paquete.

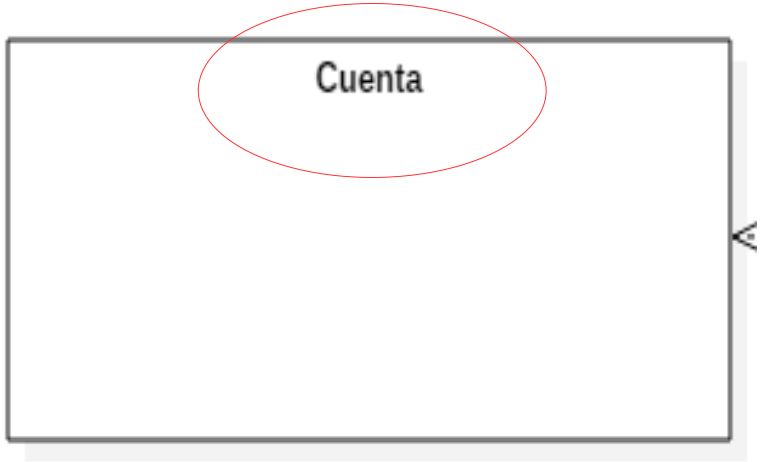
class

Para declarar una clase pública deberemos utilizar la palabra reservada public.

```
[public] class NombreClase{ }
```

```
package banco;
public class Cuenta{


}
```



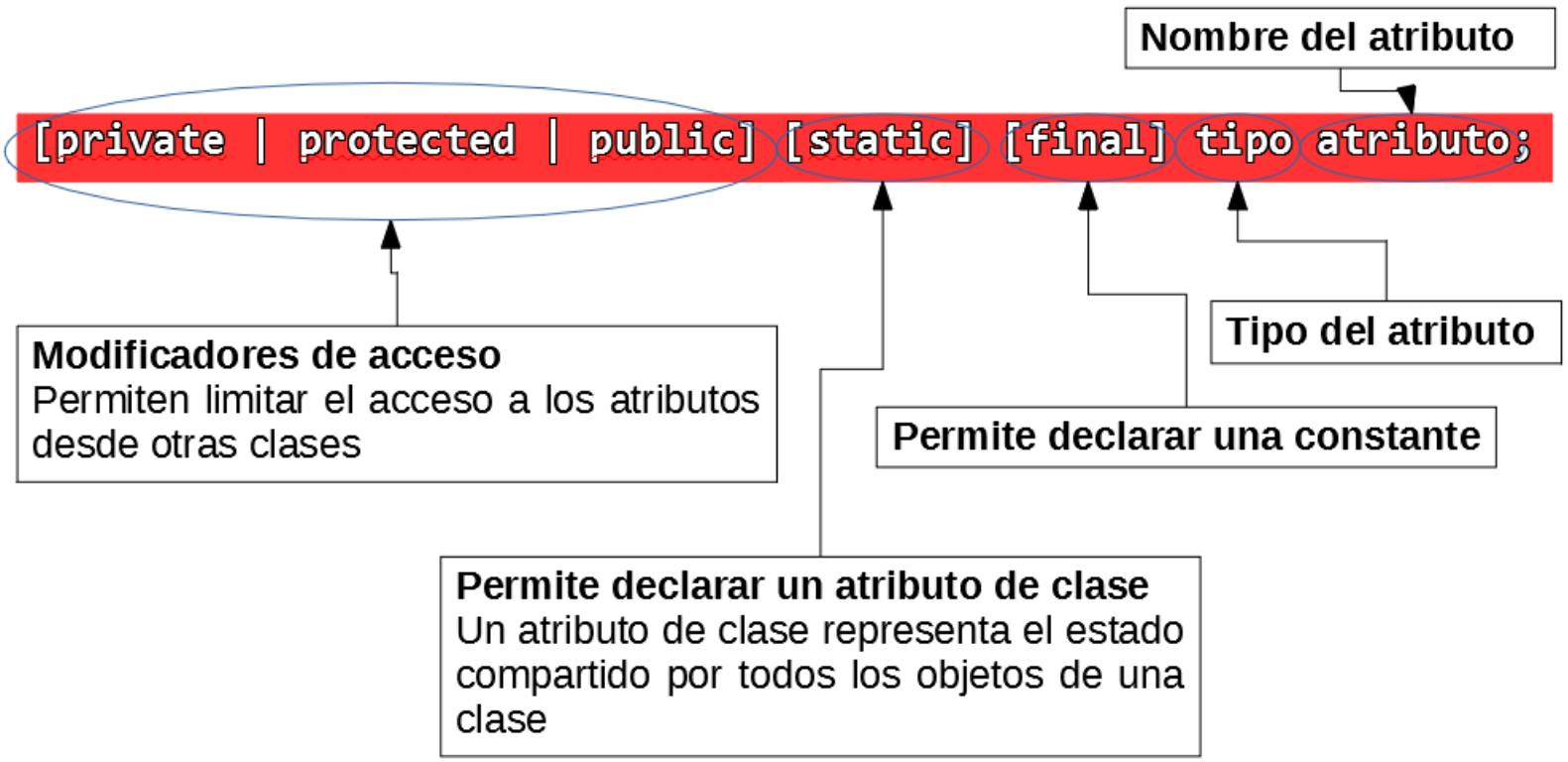
class

```
[package paquete;]
[import paquete.Clase;]
[public] class NombreClase{
    atributos
    constructores
    métodos get y set
    métodos
}
```

Atributos

Los atributos representan el estado del objeto.

Para declarar un atributo:



Atributos

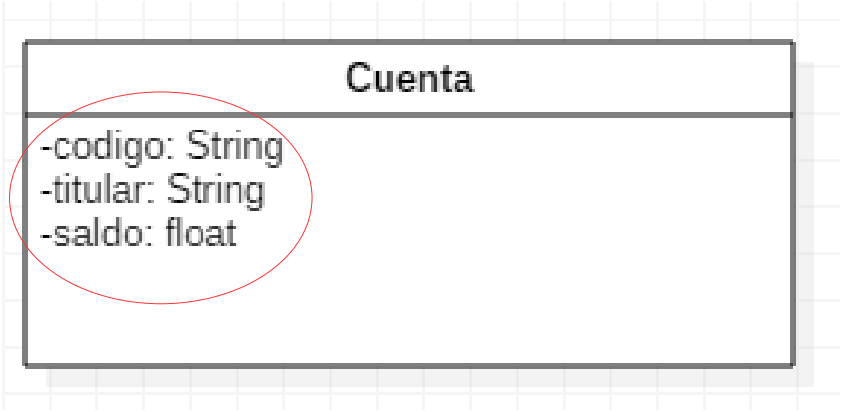
Modificadores de acceso.

	MODIFICADOR	CLASE	PAQUETE	SUBCLASE	OTROS
+	**public**	✔	✔	✔	✔
#	**protected**	✔	✔	✔	✘
-	**private**	✔	✘	✘	✘
~	*sin modificador*	✔	✔	✘	✘

Atributos

```
package banco;
public class Cuenta{
    private String codigo;
    private String titular;
    private float saldo;

}
```

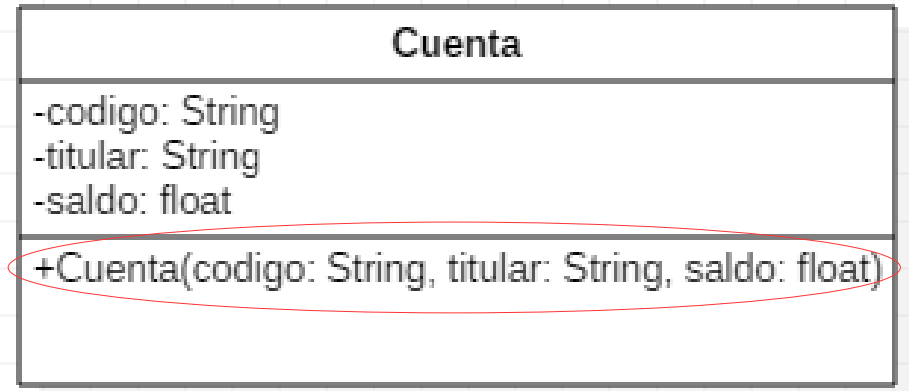


Constructores

Los constructores se utilizan para instanciar las clases, es decir, para crear objetos de una clase.

```
package banco;
public class Cuenta{
    private String codigo;
    private String titular;
    private float saldo;

    public Cuenta(String codigo, String titular, float saldo){
        this.codigo=codigo;
        this.titular=titular;
        if(saldo>0){
            this.saldo=saldo;
        }
    }
}
```



Constructores

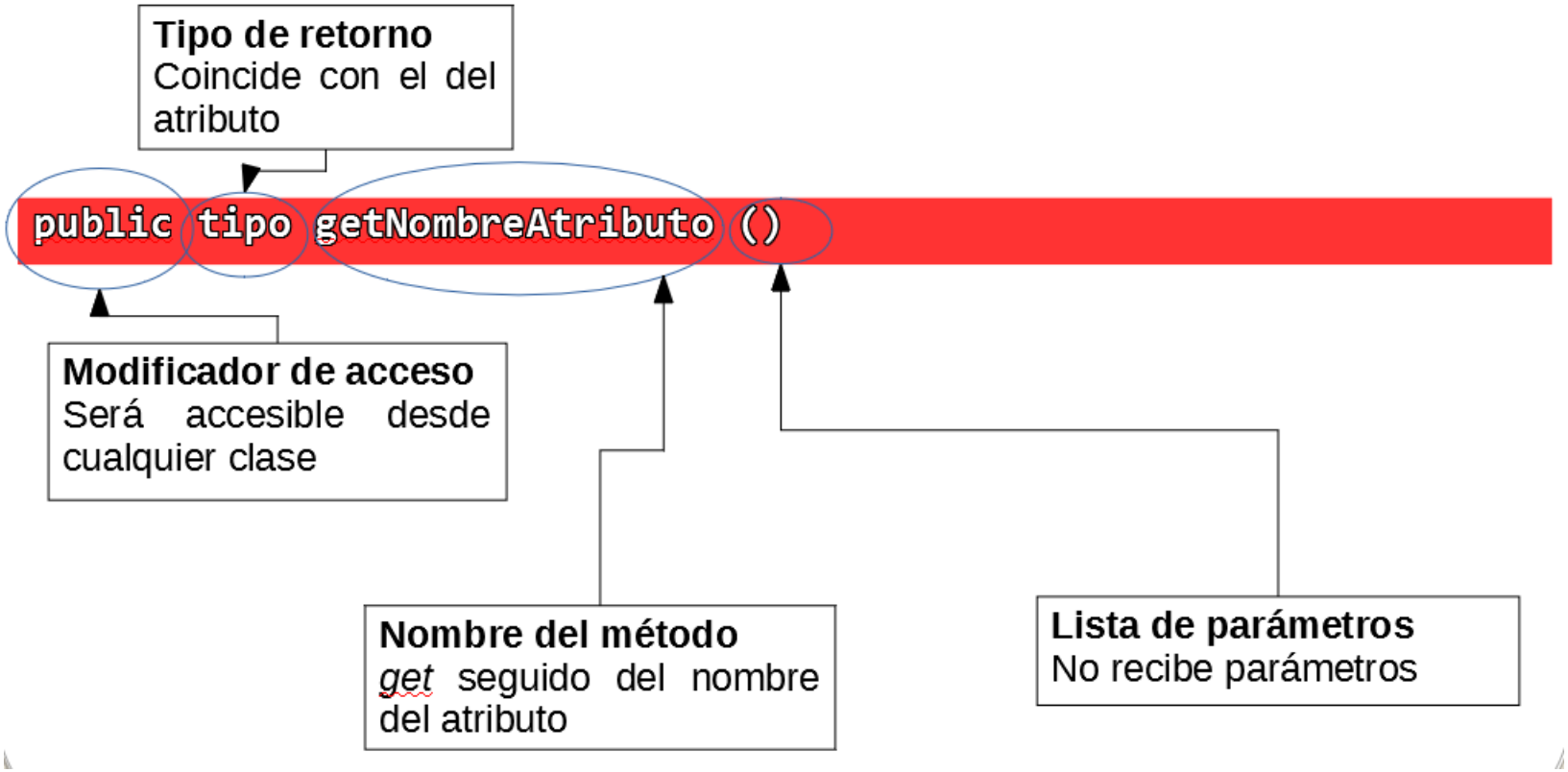
Puede haber más de un constructor pero se tendrán que diferenciar en los parámetros que reciben.

La existencia de más de un constructor se denomina sobrecarga de constructores.

Si no existe ningún constructor habrá uno implícito que inicializará los atributos a sus valores por defecto.

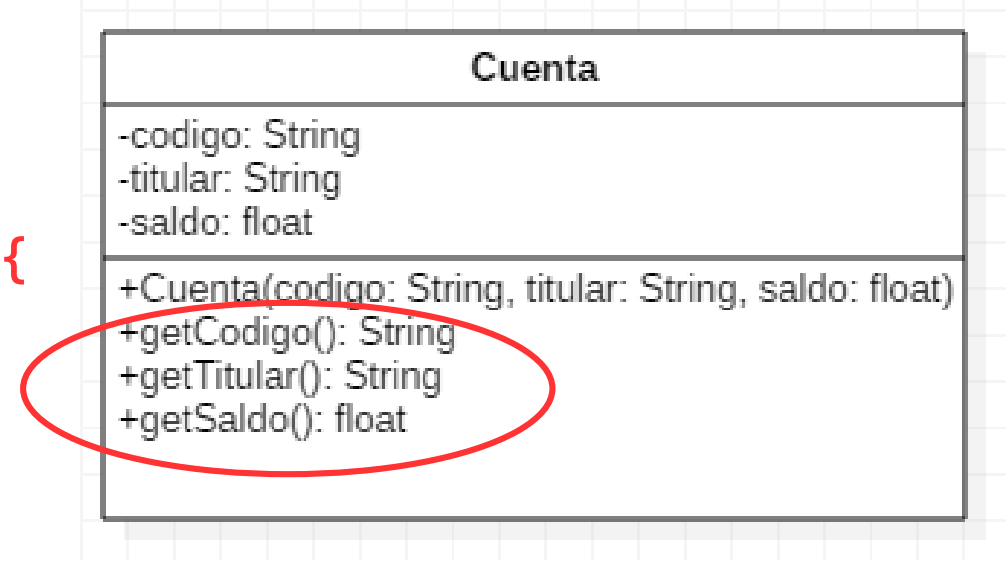
Métodos get

Los métodos get permiten consultar el valor de un atributo.



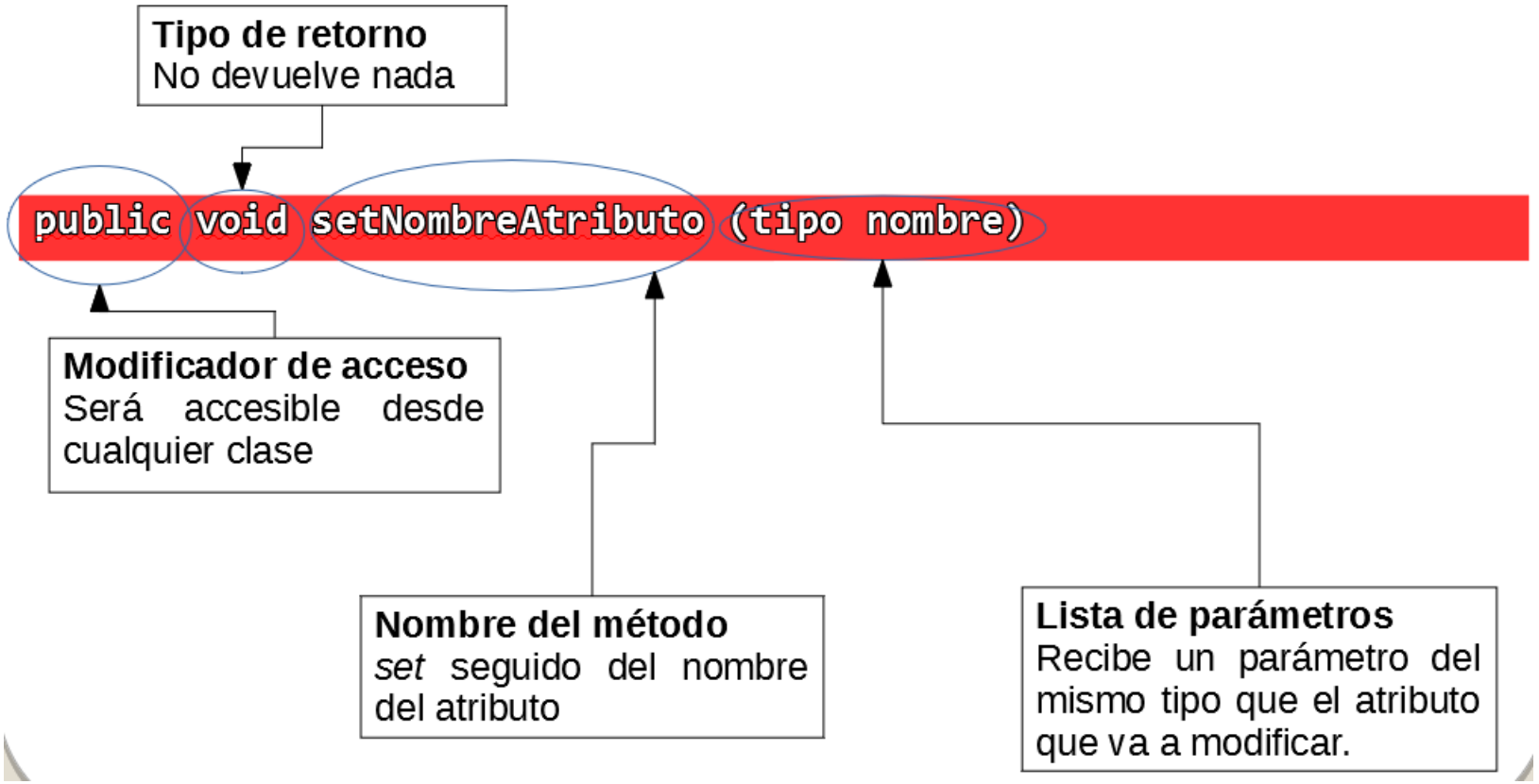
Métodos get

```
package banco;
public class Cuenta{
    private String codigo;
    private String titular;
    private float saldo;
    public Cuenta(String codigo, String titular, float saldo){
        this.codigo=codigo;
        this.titular=titular;
        this.saldo=saldo;
    }
    public String getCodigo(){
        return codigo;
    }
    public String getTitular(){
        return titular;
    }
    public float getSaldo(){
        return saldo;
    }
}
```



Métodos set

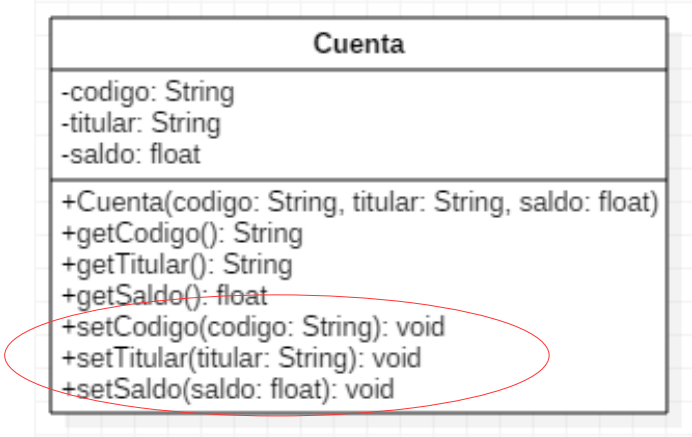
Los métodos set permiten modificar el valor de un atributo.



Métodos set

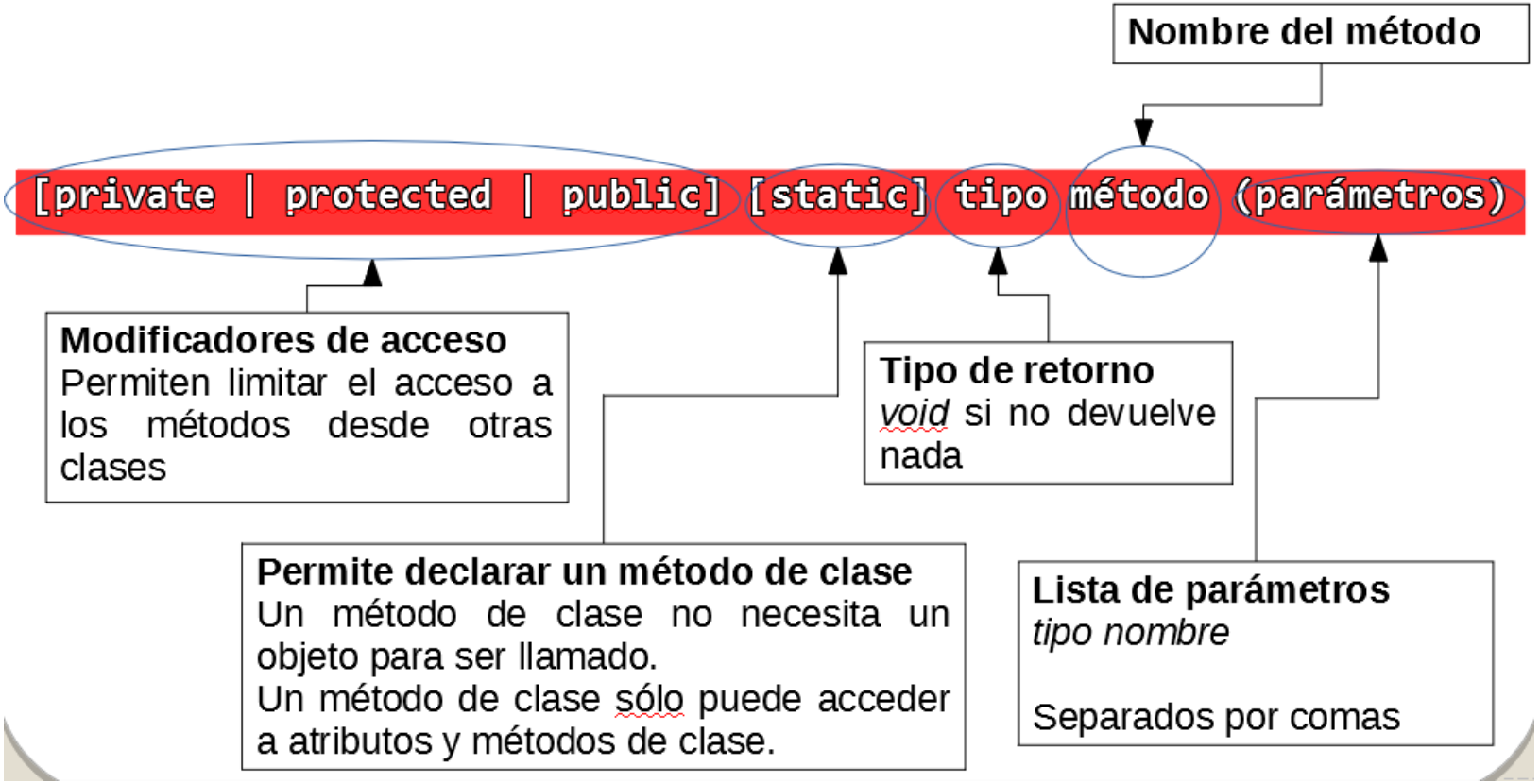
```
package banco;
public class Cuenta{
    private String codigo;
    private String titular;
    private float saldo;

 public void setCodigo(String codigo){
    this.codigo=codigo;
 }
 public void setTitular(String titular){
    this.titular=titular;
 }
 public void setSaldo(float saldo){
    if(saldo>=0){
        this.saldo=saldo;
    }
 }
}
```



Métodos

Los métodos definen el comportamiento de un objeto de una clase.



Métodos

Puede haber varios métodos con el mismo nombre pero se tendrán que diferenciar en los parámetros que reciben.

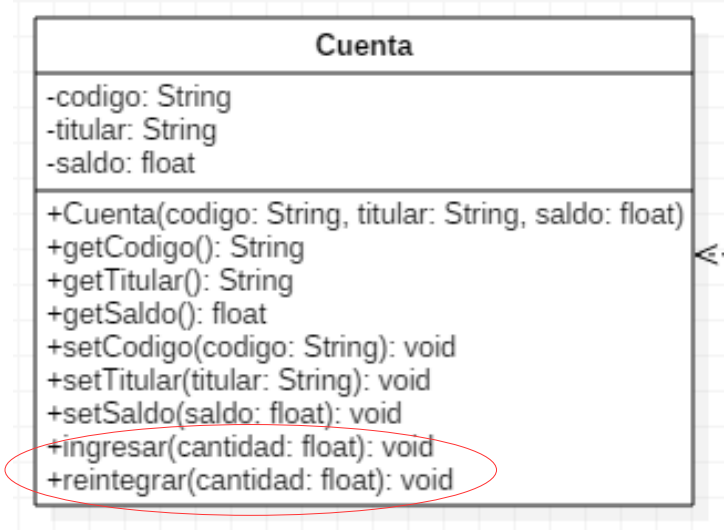
Cuando hay varios métodos con el mismo nombre se denomina sobrecarga de métodos.

	MODIFICADOR	CLASE	PAQUETE	SUBCLASE	OTROS
+	**public**	✔	✔	✔	✔
#	**protected**	✔	✔	✔	✘
-	**private**	✔	✘	✘	✘
~	*sin modificador*	✔	✔	✘	✘

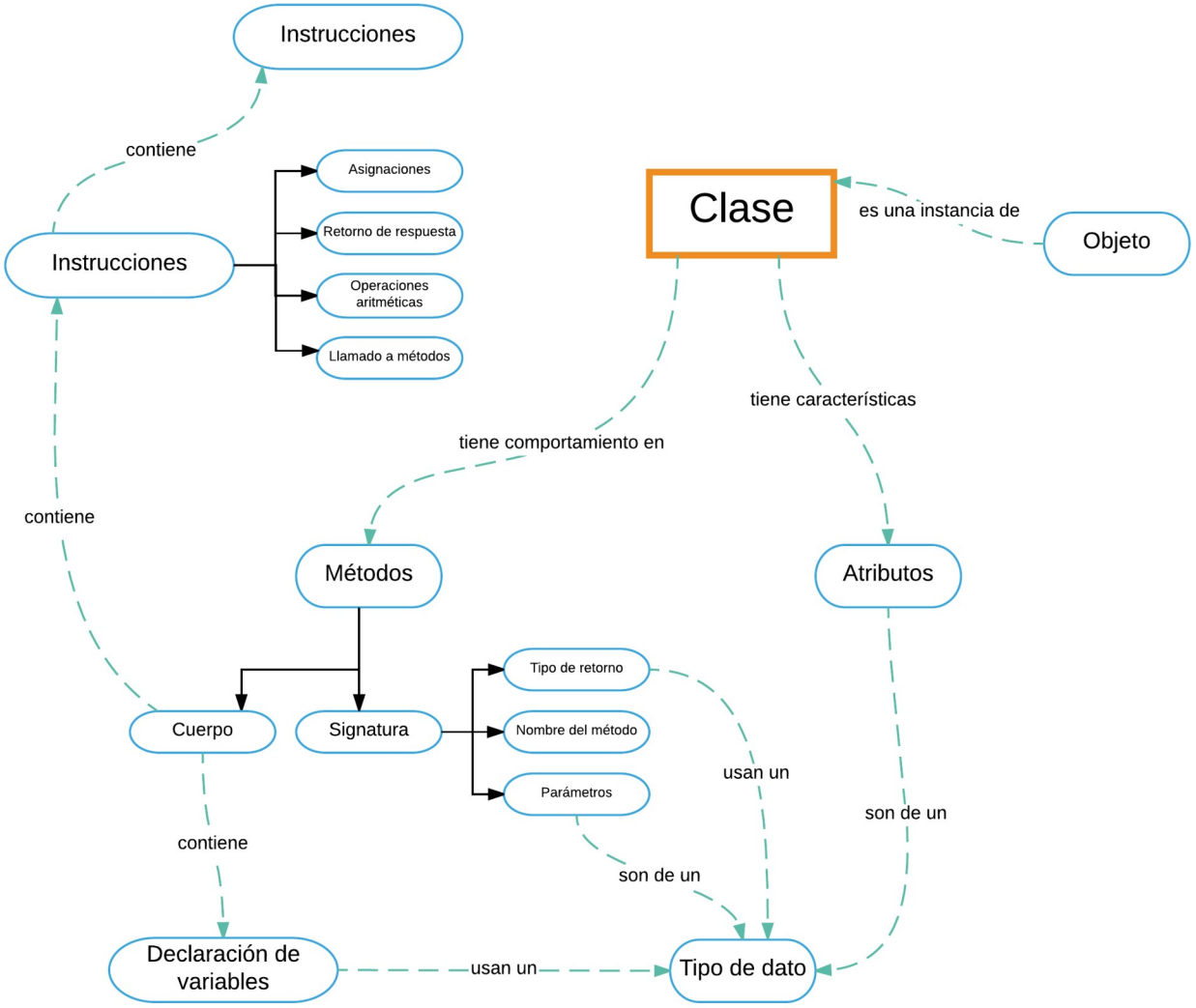
Métodos

```
package banco;
public class Cuenta{
    private String codigo;
    private String titular;
    private float saldo;

 public void ingresar(float cantidad){
    if(cantidad>0){
        saldo+=cantidad;
    }
 }
 public void reintegrar(float cantidad){
    if(cantidad>0 && cantidad<=saldo){
        saldo-=cantidad;
    }
 }
}
```



Clase


Instrucciones
contiene
Asignaciones
Retorno de respuesta
Operaciones aritméticas
Llamado a métodos
Instrucciones
Clase
es una instancia de
Objeto
tiene comportamiento en
tiene características
contiene
Métodos
Atributos
Tipo de retorno
Cuerpo
Signatura
Nombre del método
Parámetros
usan un
son de un
contiene
son de un
Declaración de variables
usan un
Tipo de dato

main

```
package banco;
public class AplicacionBanco{
    public static void main(String[] args){
        Cuenta cuenta1;

        cuenta1=new Cuenta("3285 7031 25 2595659938","Juan",100);
        cuenta1.ingresar(100);
        cuenta1.reintegrar(50);
        System.out.println(cuenta1.getSaldo());
    }

}
```