

Programación Orientada a Objetos (POO)

Pontificia Universidad Javeriana
Gabriel Gómez Corredor

Características

- Objetos
- Representa las entidades como objetos
- Instancias

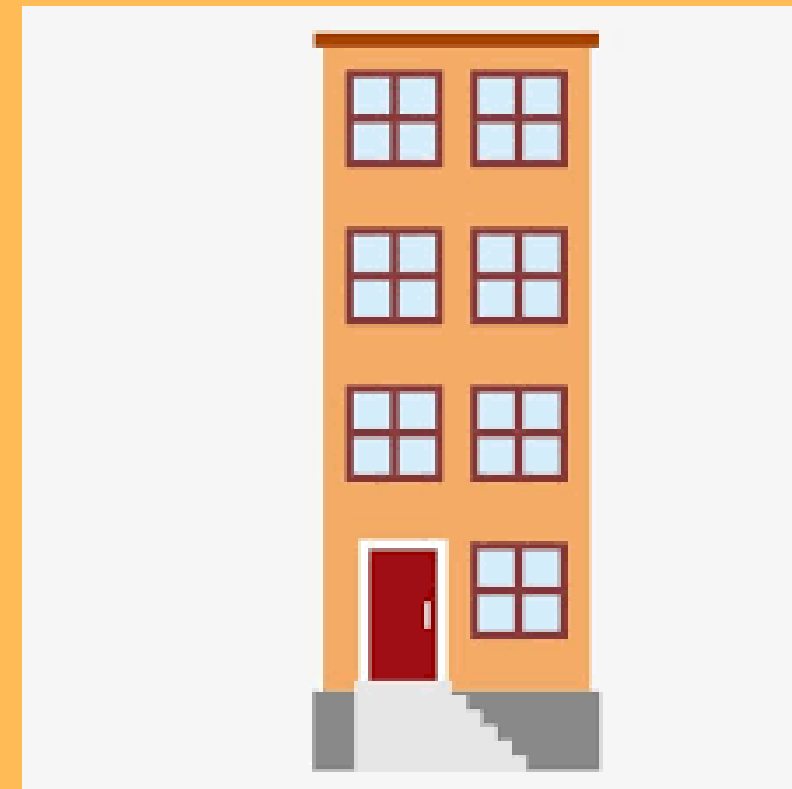
QUE ES ?

Es un paradigma de la programación que facilita la estructuración de los componentes en un software



Objetos

- En POO un objeto va a representar una entidad
- Tiene como función promover la comprensión del mundo real y será la base de la implementación

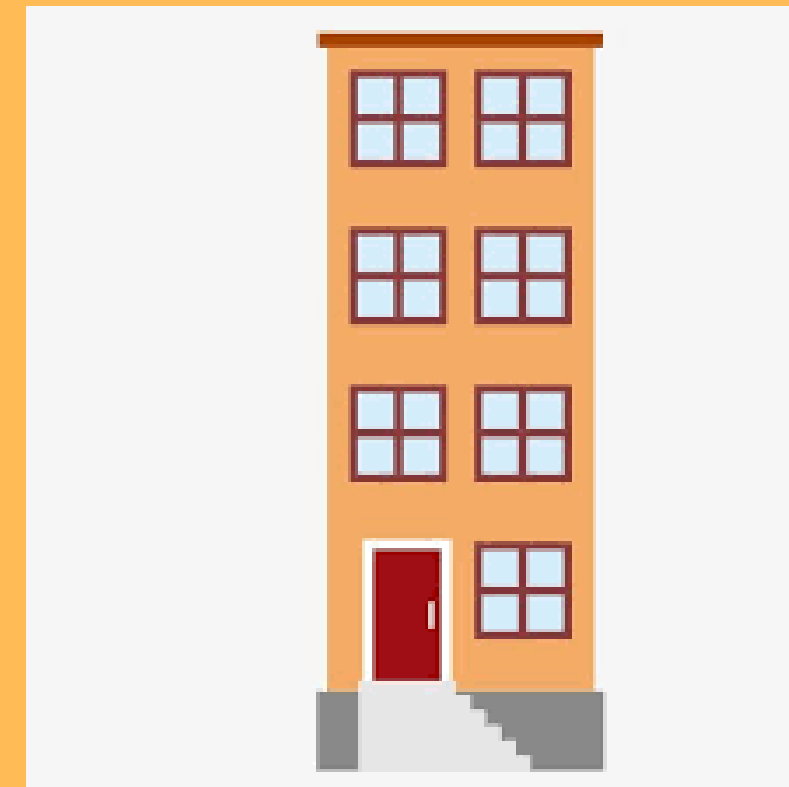


Atributos

- Propiedades del objeto
- Son los datos que tiene el objeto



- placa
- marca
- color
- modelo



- altura
- num_pisos
- cant_residentes



Comportamiento del objeto

El comportamiento del objeto se refiere a los servicios que este presta



Propiedades:

- digito1
- digito2

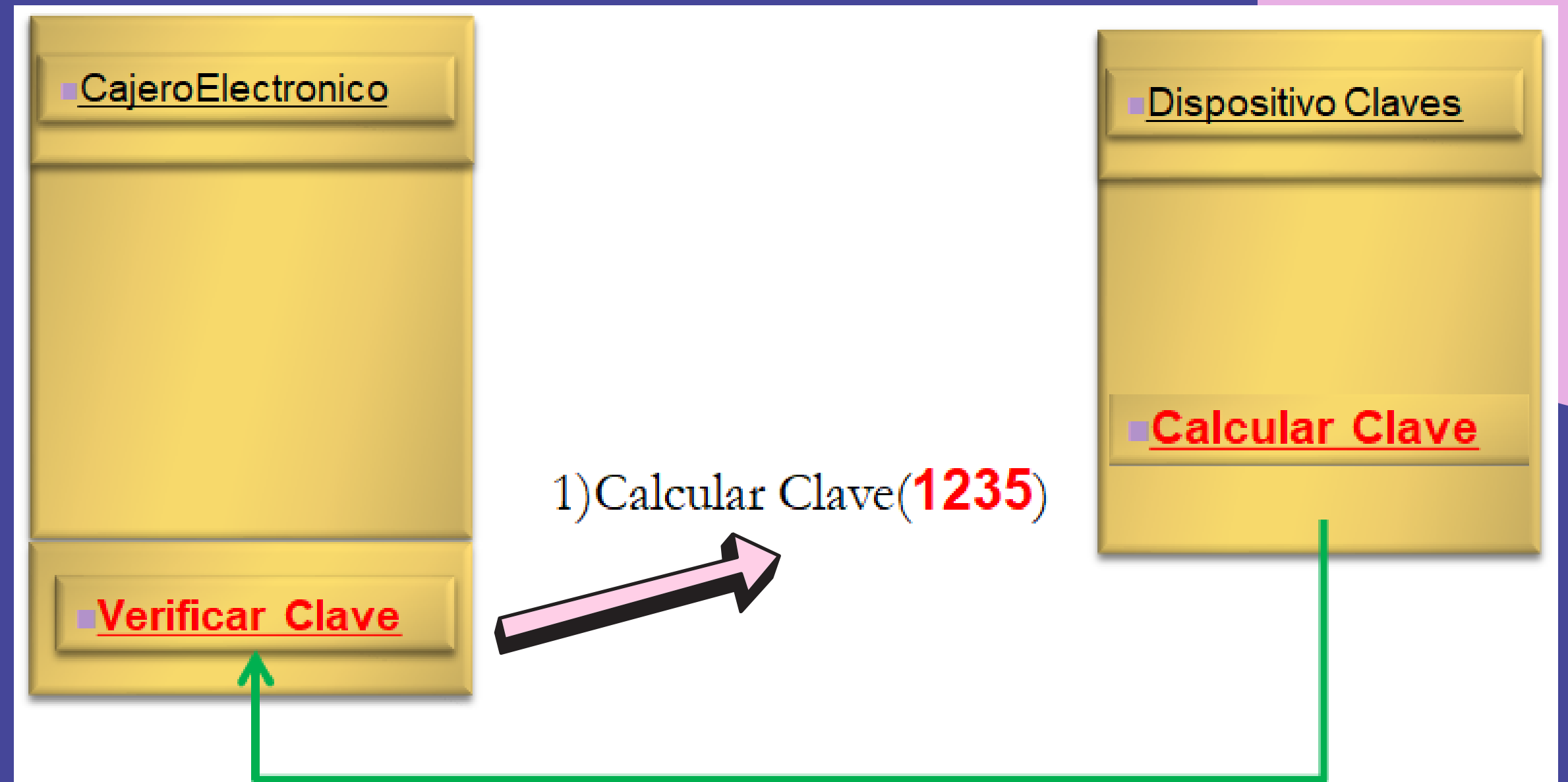
Métodos:

- Sumar()
- Restar()
- Multiplicar()
- Dividir()

Interacción entre objetos

Al enfrentarnos a un software real, nuestros objetos tendrán que interactuar entre ellos

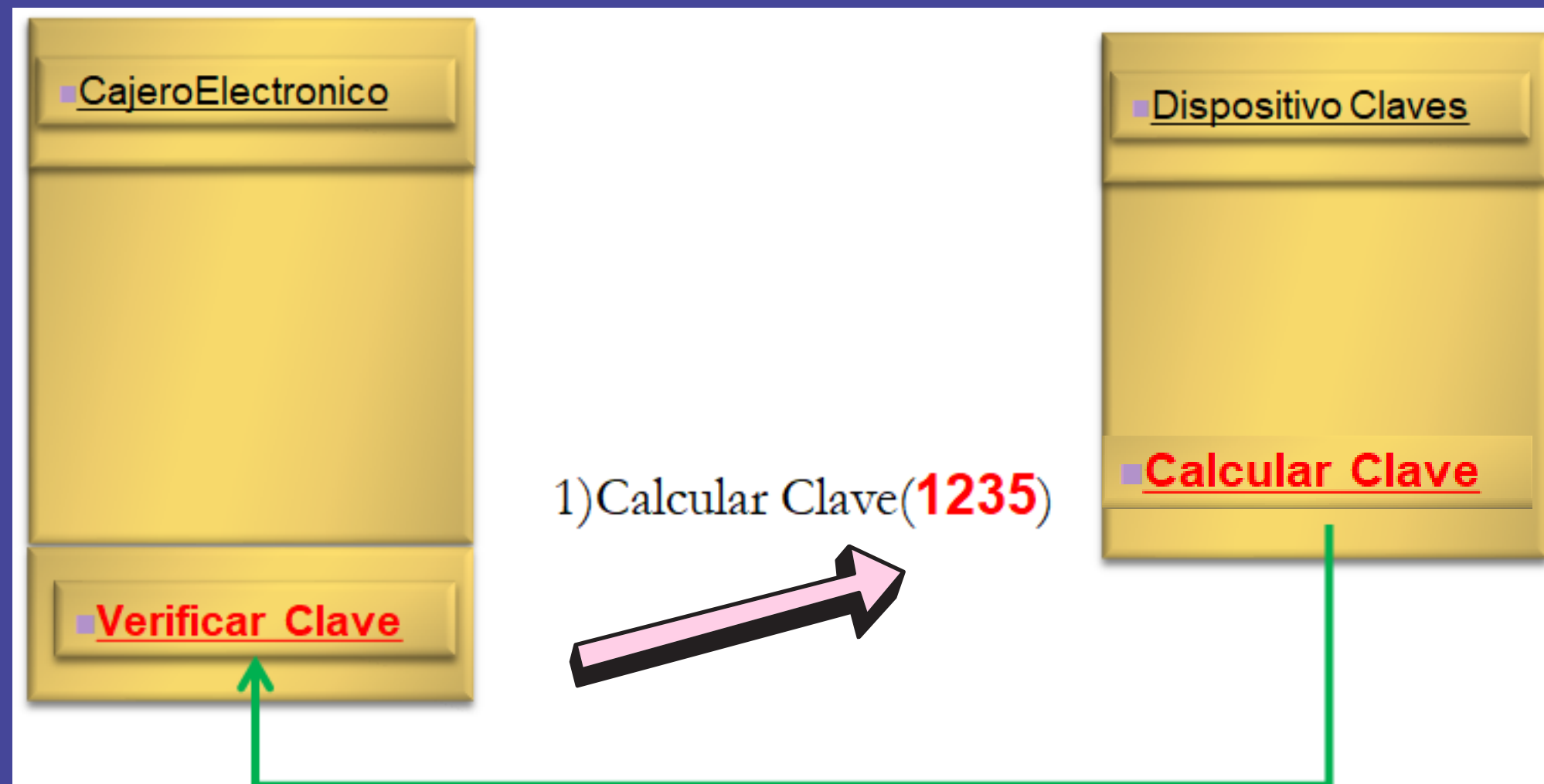
El objeto cajeroElectronico necesitará verificar su clave, para ello necesita del objeto Dispositivo Claves para que haga su labor



Mensajes

Para que un objeto realice una tarea se le debe enviar un mensaje

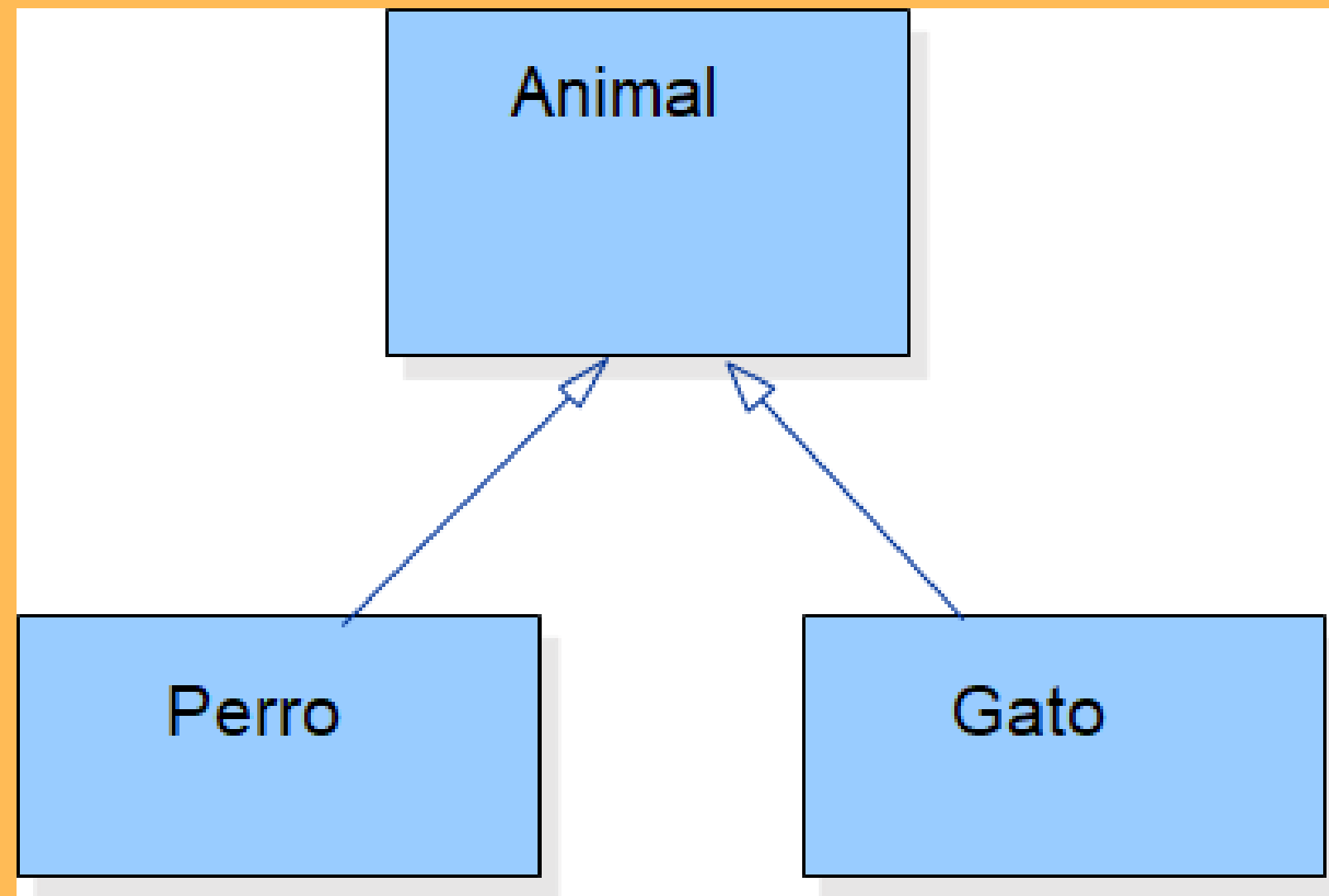
- Solo se le pueden enviar mensajes a objetos que entiendan el mensaje
- El objeto que recibe el mensaje deber tener un método para que maneje el mensaje recibido
- Los valores que viajan en el mensaje se llaman argumentos/parámetros



Herencia

una clase adquiere las:

- propiedades (atributos)
- comportamiento (métodos) de otra.

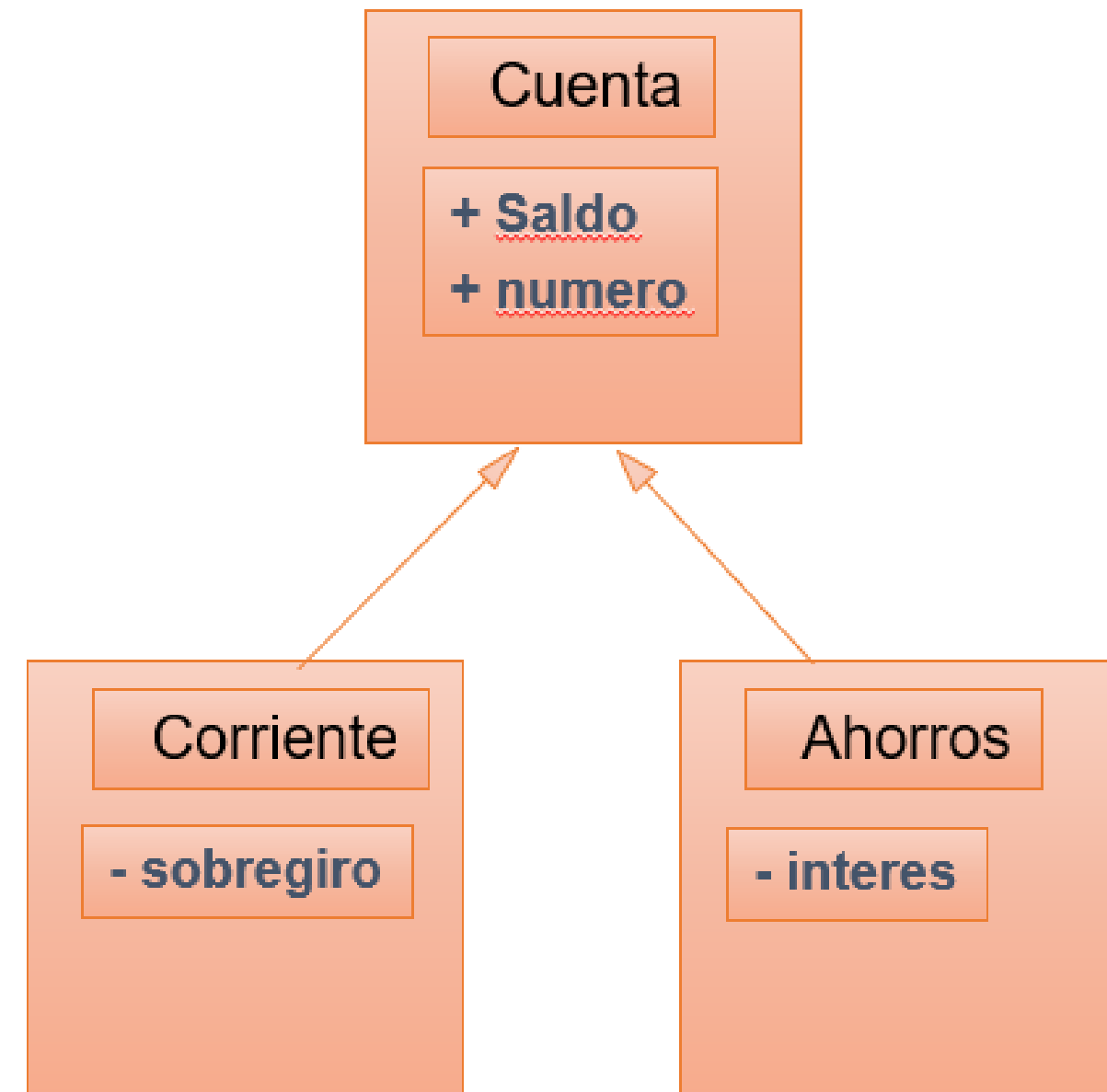


Ejemplo



La clase Corriente hereda
los atributos
de Cuenta

* En total Corriente tiene 3
atributos



Concepto de Herencia

Se puede definir una clase a partir de otra ya existente

Heredando sus atributos y métodos,

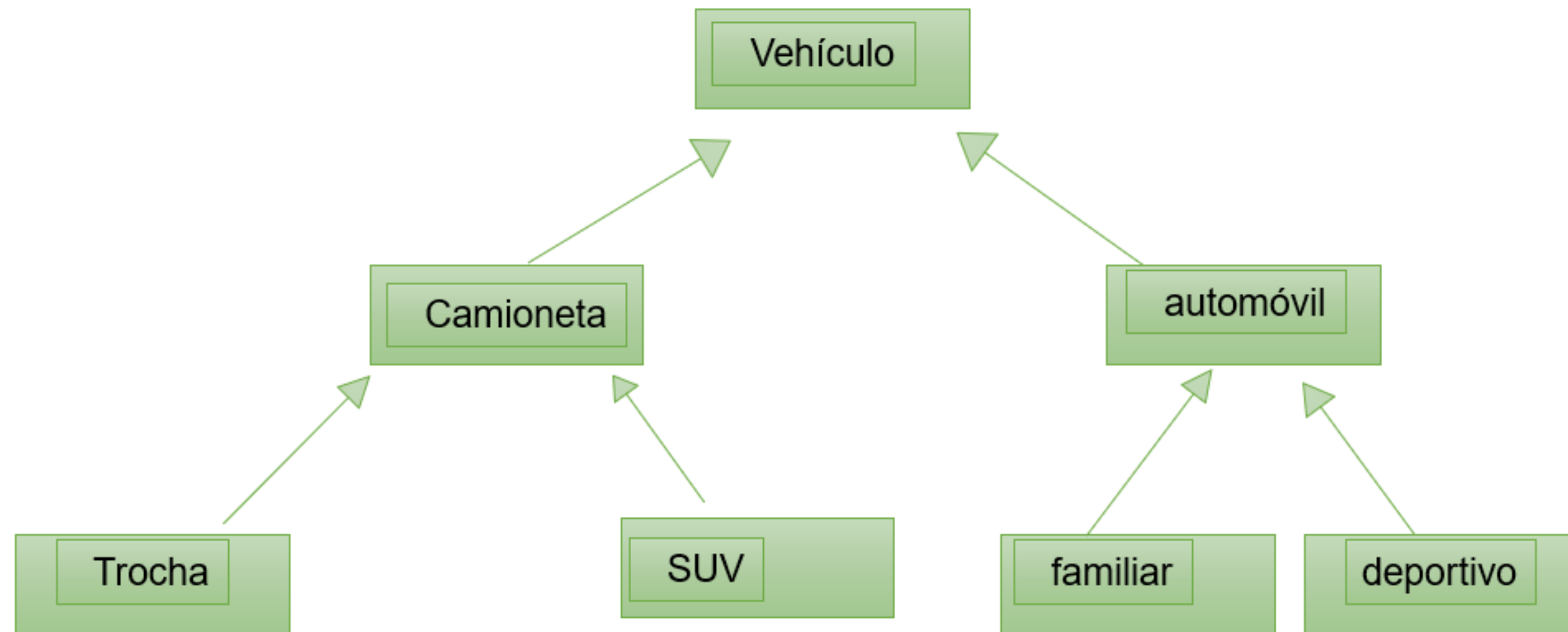
Y siendo posible:

- añadir nuevos elementos (atributos o métodos)
- Redefinir métodos



Jerarquía

Especialización: iniciando arriba y se descompone



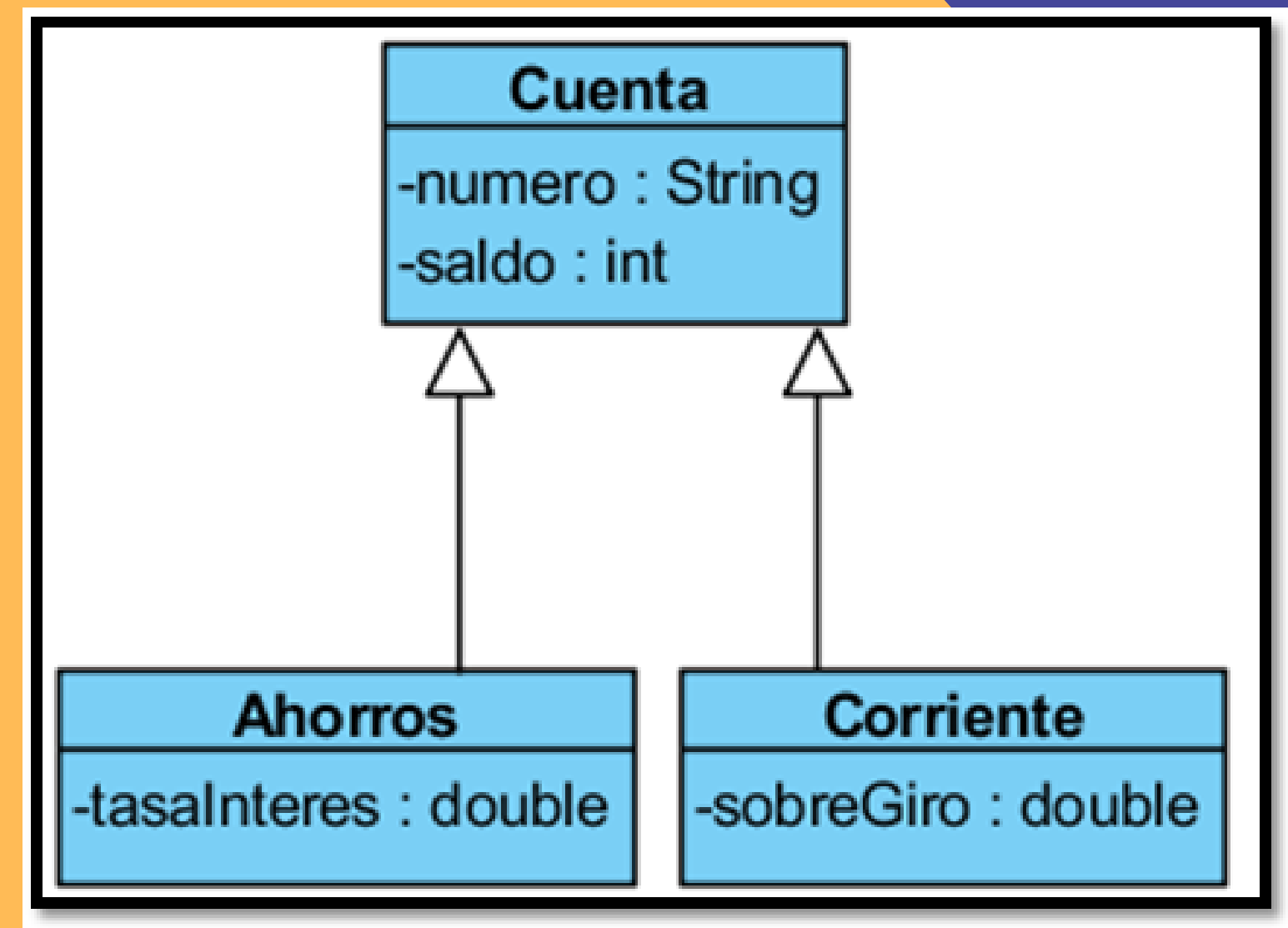
Generalización: iniciando abajo y se abstrae hacia arriba

Relación de Generalización

Características:

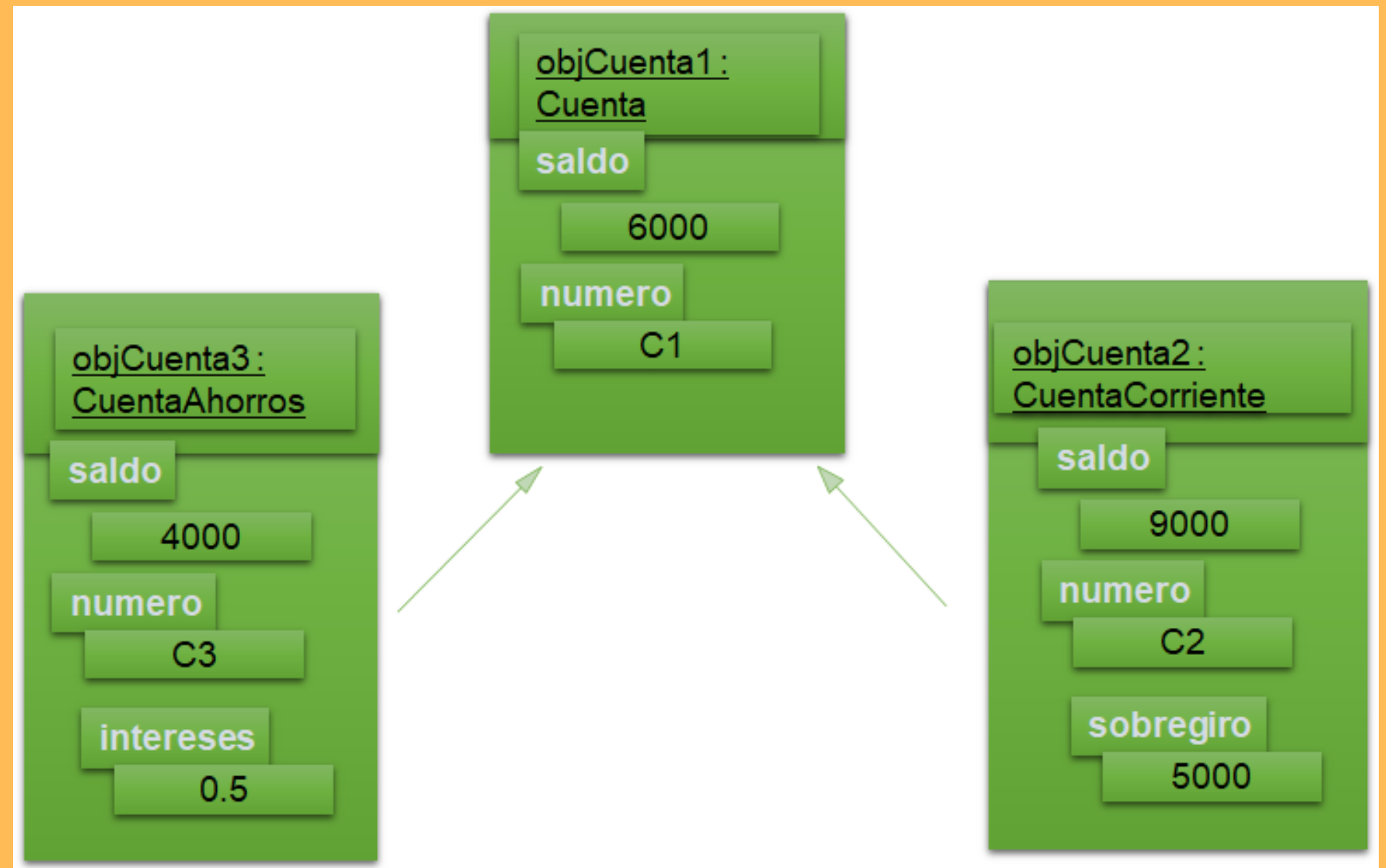
- Clase padre (Superclase)
- Clase hijo (Subclases)

La generalización tiene un triángulo apuntando a la superclase.



No se heredan valores !!

Herencia



Ejercicio

Dibujar un diagrama de herencia donde este represente un caso de la vida real

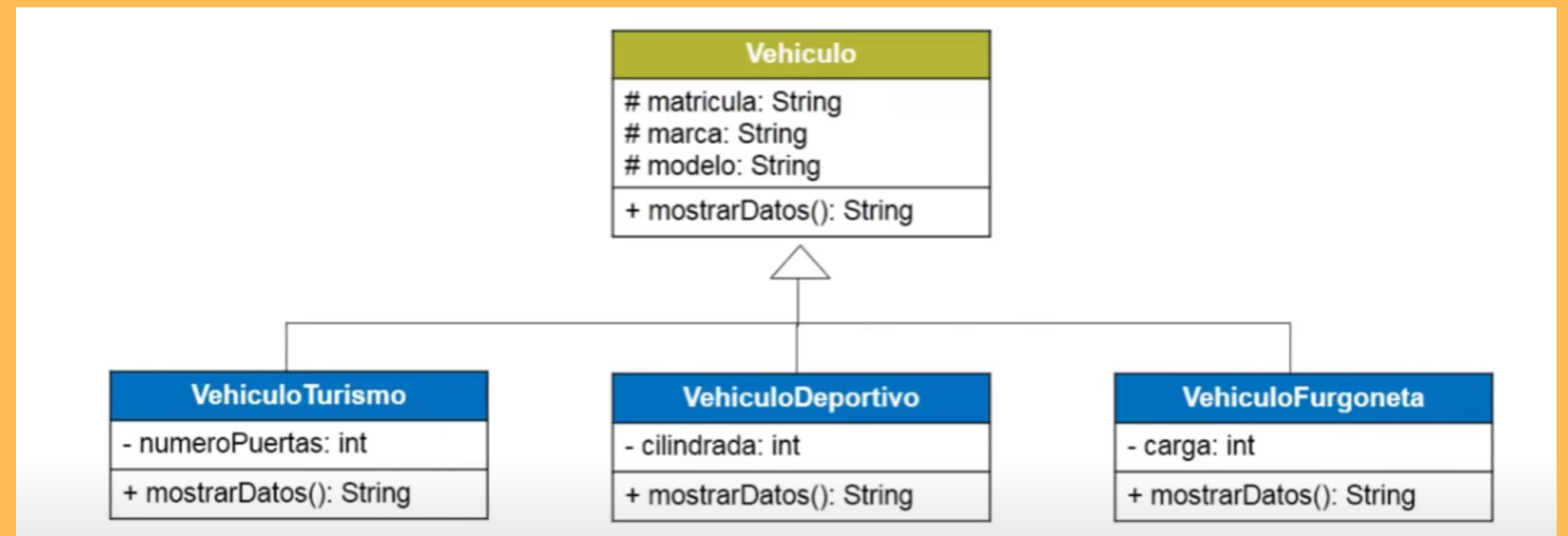
Que contenga:

- una superclase
- más de una subclase
- atributos del padre y atributos extra de los hijos
- métodos

Polimorfismo

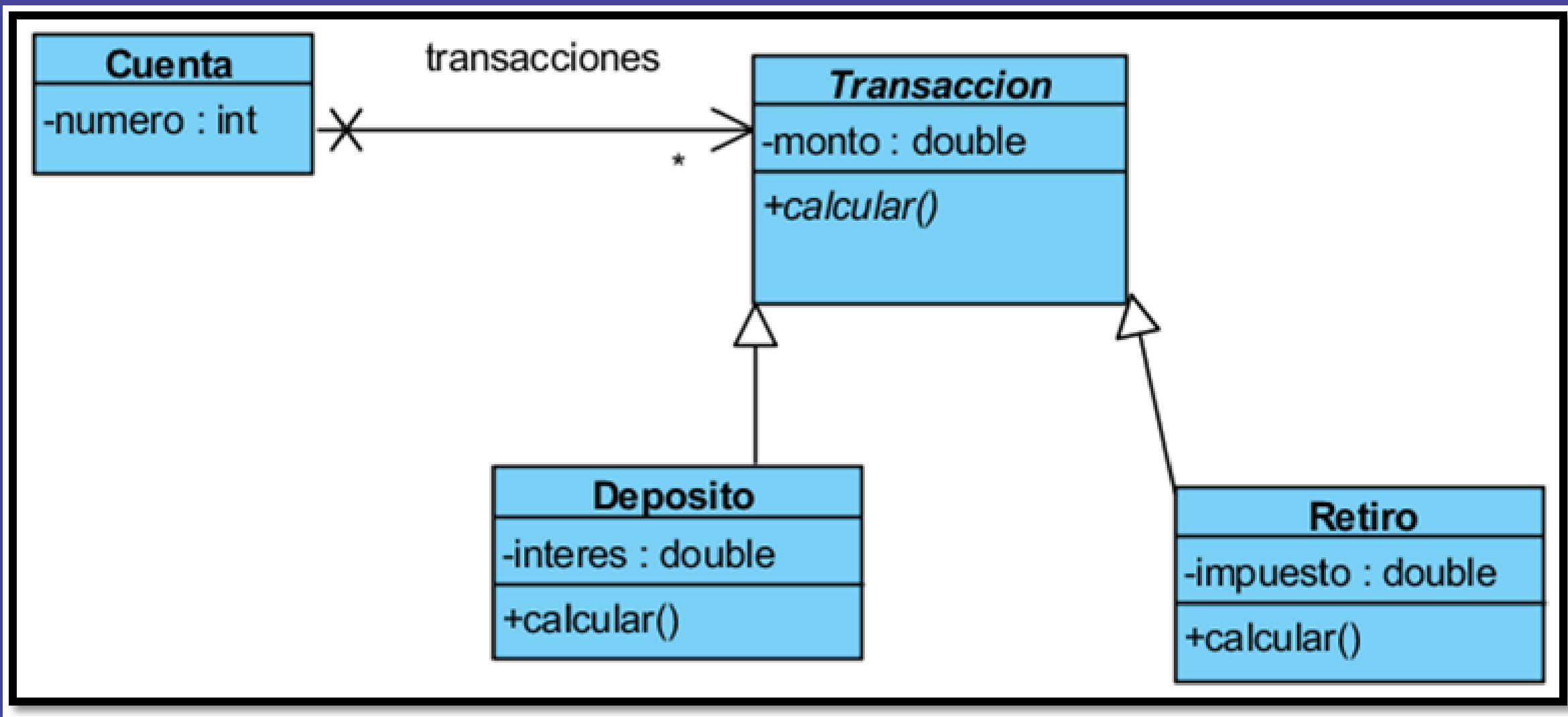
En una relación de herencia, un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases

Poli: muchas
morfismo: forma



Polimorfismo

El mismo método puede causar que diferentes acciones ocurran, dependiendo del tipo del objeto en el cual el método es invocado.



- Método `calcular()` en la superclase es abstracto
- Las clases **Depósito** y **Retiro** deben implementar sus métodos `calcular()`

Encapsulamiento

Nos permite ocultar o restringir el acceso a los atributos y/o métodos de una clase

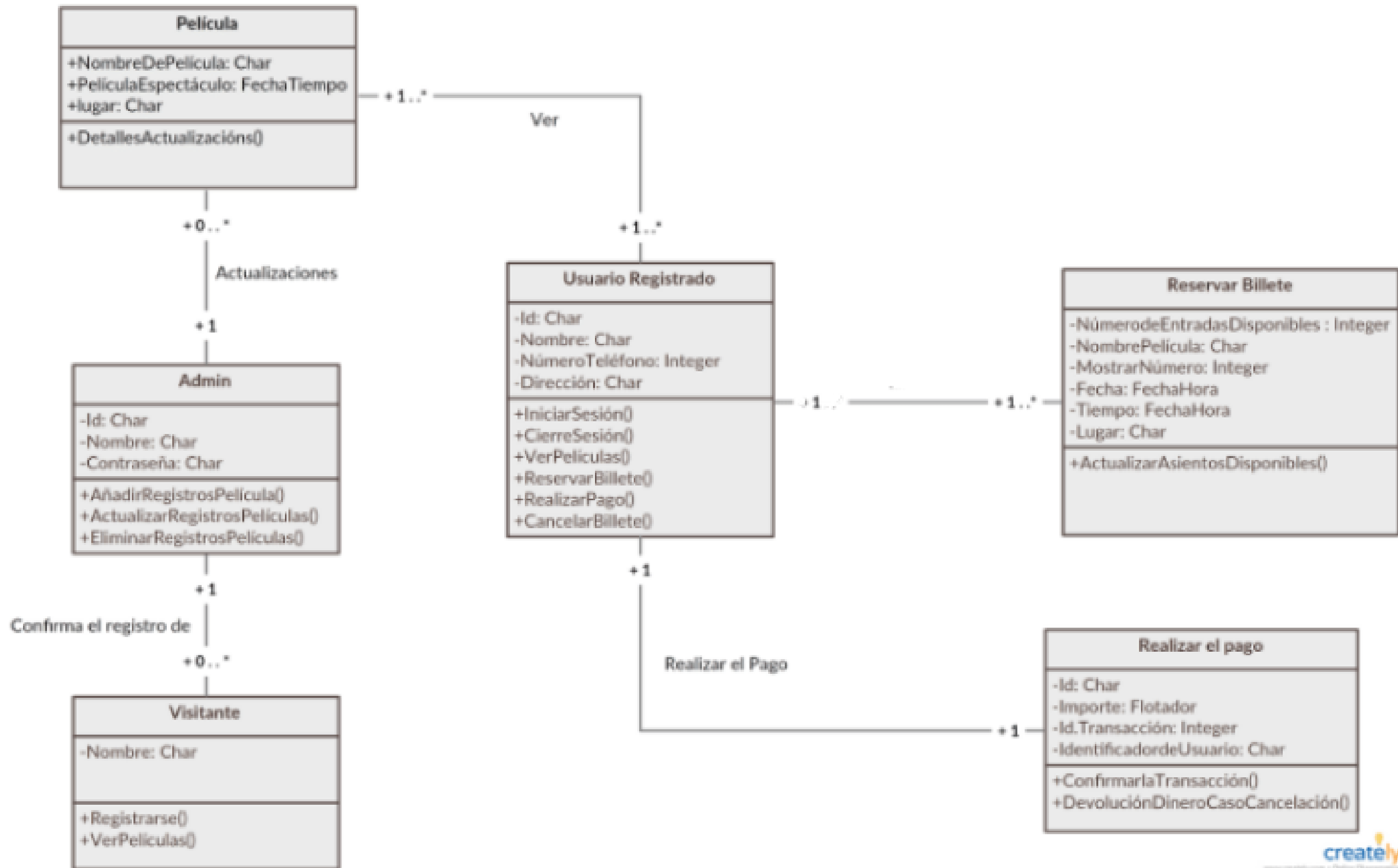
Existen 3 niveles de acceso

- **Public:** Todos pueden acceder a los datos o métodos de una clase (+)
- **Protegido:** No son publicas, solo son accesibles dentro de su clase y por sus subclases (#)
- **Privado:** Solo accesible dentro de la propia clase (-)

Diagrama de clases

Características

- Entidades
- Relaciones
- Cardinalidad
- Navegación



Ejercicio

Realizar un diagrama de clases, que represente un caso de la vida real

Que contenga:

- Todas las características de POO

Recordemos conceptos POO

<https://padlet.com/gomezcgabriel1998/vozm001xfiy4zdf>

Lucidchart

Crear una universidad con entidades, atributos, y relaciones

Clases vs Objeto

Clase: es la estructura o el molde que puede tomar un objeto

Objeto: Es un tipo de dato que surge a partir de una clase

Clase

```
public class Vehiculo {  
    // Declaramos los atributos  
    protected String color;  
    protected int puertas;  
    protected int asientos;  
  
    public String getColor(){  
        return this.color;  
    }  
    public int getPuertas(){  
        return this.puertas;  
    }  
    public int getAsientos(){  
        return this.asientos;  
    }  
}
```

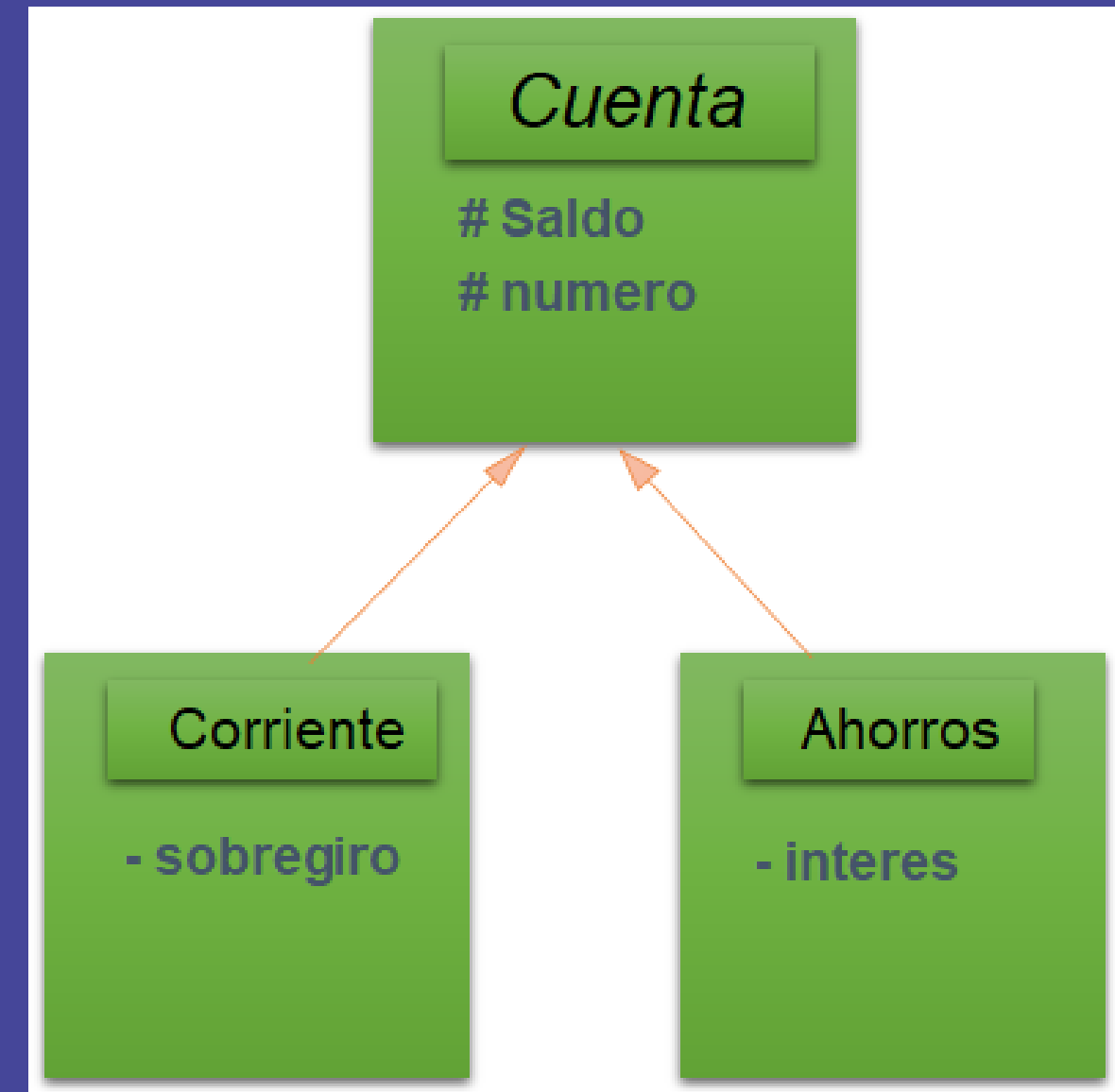
Objeto

```
public class Polimorfismo {  
    public static void main(String[] args) {  
        // Creamos los 2 objetos Vehiculo  
        Vehiculo objVehiculo1 = new Vehiculo("Rojo",2,3);  
        Vehiculo objVehiculo2 = new Vehiculo();  
        // Mostramos los datos para ambos objetos  
        System.out.println("Color Vehiculo1 : " + objVehiculo1.getColor());  
        System.out.println("Puertas Vehiculo1 : " + objVehiculo1.getPuertas());  
        System.out.println("Asientos Vehiculo1 : " + objVehiculo1.getAsientos());  
        System.out.println();  
        System.out.println("Color Vehiculo2 : " + objVehiculo2.getColor());  
        System.out.println("Puertas Vehiculo2 : " + objVehiculo2.getPuertas());  
        System.out.println("Asientos Vehiculo2 : " + objVehiculo2.getAsientos());  
    }  
}
```

Clases abstractas

- Una clase abstracta es aquella que es muy general y que no puede ser instanciada, es decir **no** se pueden crear objetos de ella.

Se da en las relaciones de herencia



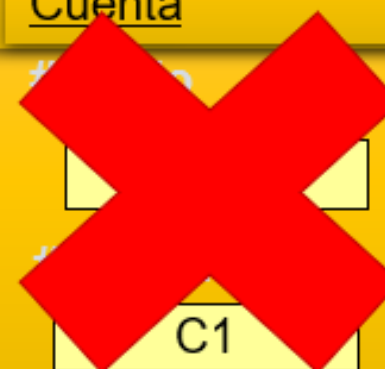
Clases abstractas

Cuando se define una superclase, no necesitamos crear instancias de dicha superclase

No se puede hacer

```
Cuenta cta = new Cuenta("C100", 200);
```

objCuenta1 :
Cuenta



C1

objCuenta2 :
CuentaCorriente

saldo
9000

numero
C2

sobregiro
5000

objCuenta3 :
CuentaAhorros

saldo
4000

numero
C3

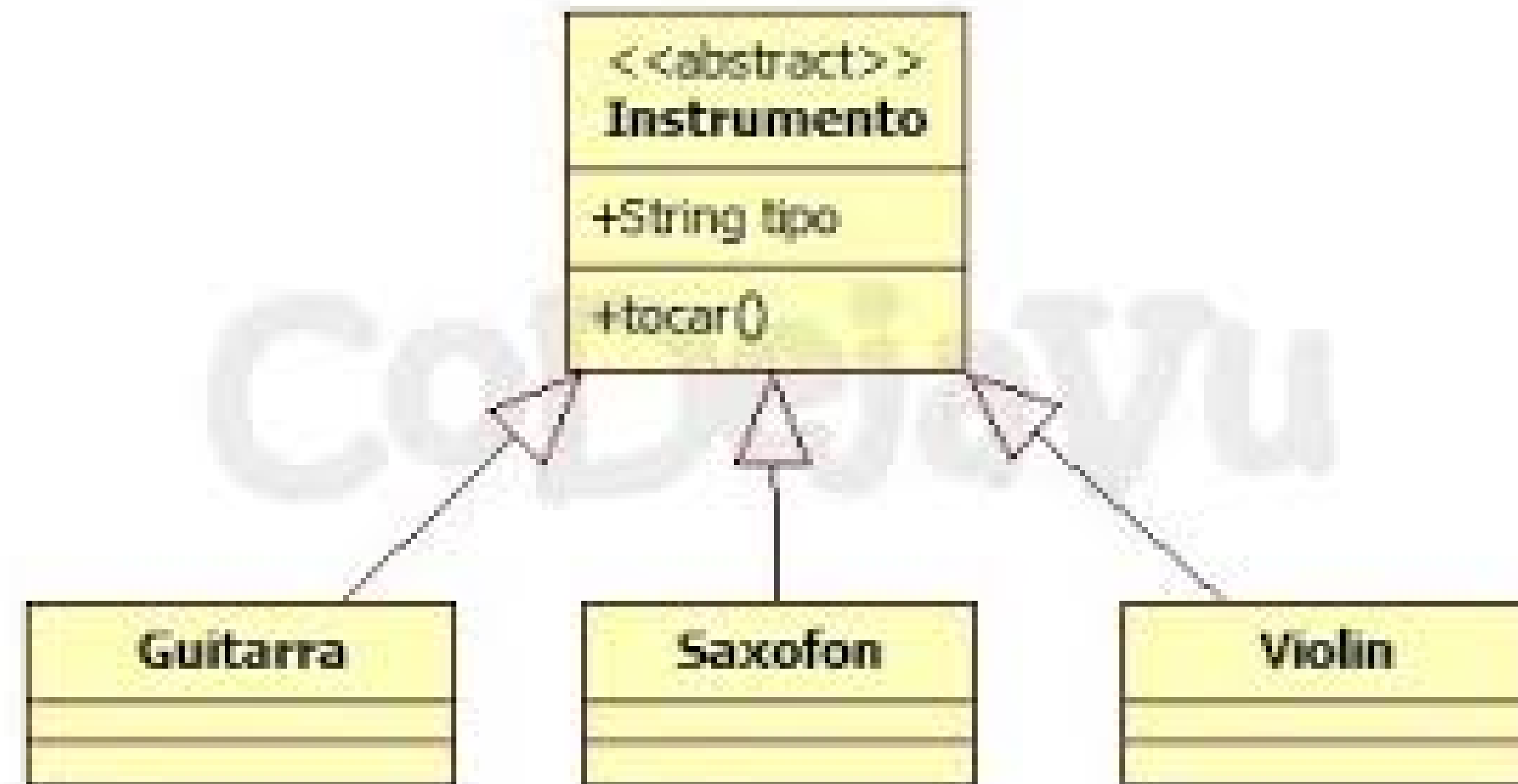
intereses
0.5

¿Qué ofrecen estas clases?

- Las clases abstractas representan un conjunto de servicios (métodos) que ofrece la clase.

El término “abstracto” en programación puede entenderse como que está definido pero no implementado.

Ofrecen métodos



Sobreescritura en la herencia

- Métodos abstractos deben ser obligatoriamente sobrescritos en las subclases que heredan de la superclase que los contiene
- las subclases deben encargarse de implementar dicho método

```
public abstract class Cuenta {  
  
    public abstract double disponible();  
}
```

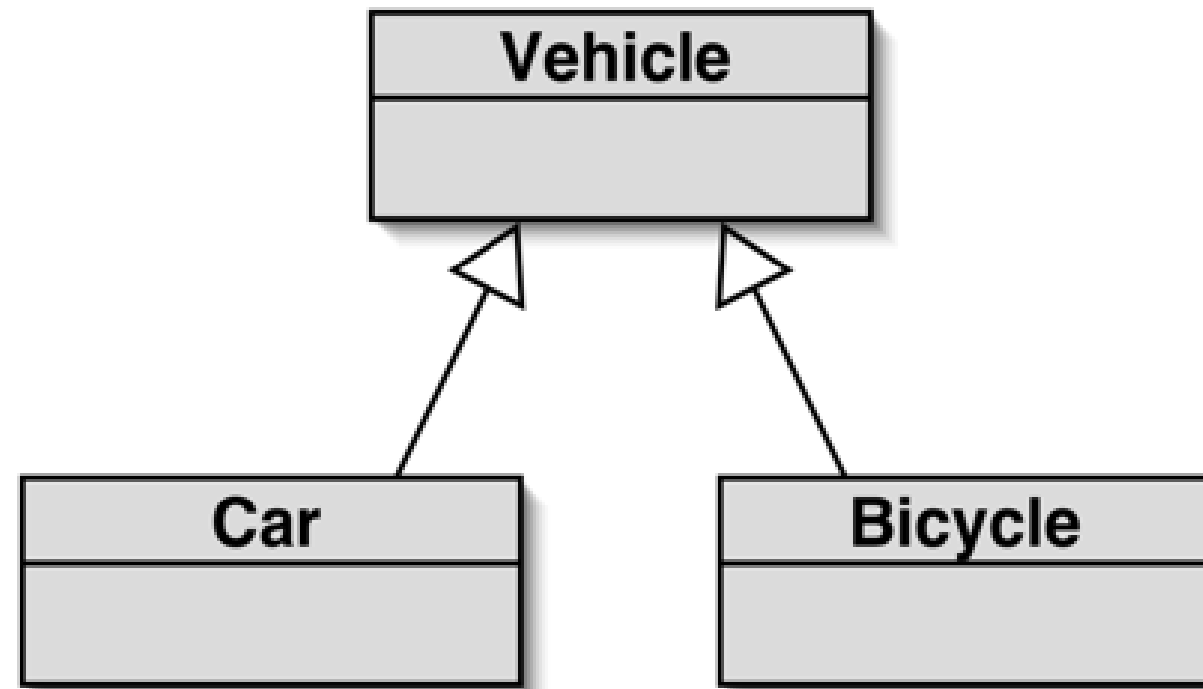
```
public class Ahorros extends Cuenta {  
  
    private double tasaInteres;  
  
    @Override  
    public double disponible() {  
        return this.getSaldo() + this.tasaInteres;  
    }  
}
```

```
public class Corriente extends Cuenta {  
  
    private double sobreGiro;  
  
    @Override  
    public double disponible() {  
        return this.getSaldo() + this.sobreGiro;  
    }  
}
```



Polimorfismo ?

Objetos de las subclases pueden ser asignados variables de la superclase



```
Vehicle v2 = new Car();  
Vehicle v3 = new Bicycle();
```

Ejercicio

Crear un diagrama de clases con una relación de herencia teniendo a la superclase como una clase abstracta

Modularidad

Que es?

Es una practica de programación que consiste en descomponer un programa en pequeños módulos que pueden compilarse por separado pero pueden comunicarse con otros modulos

Modularidad

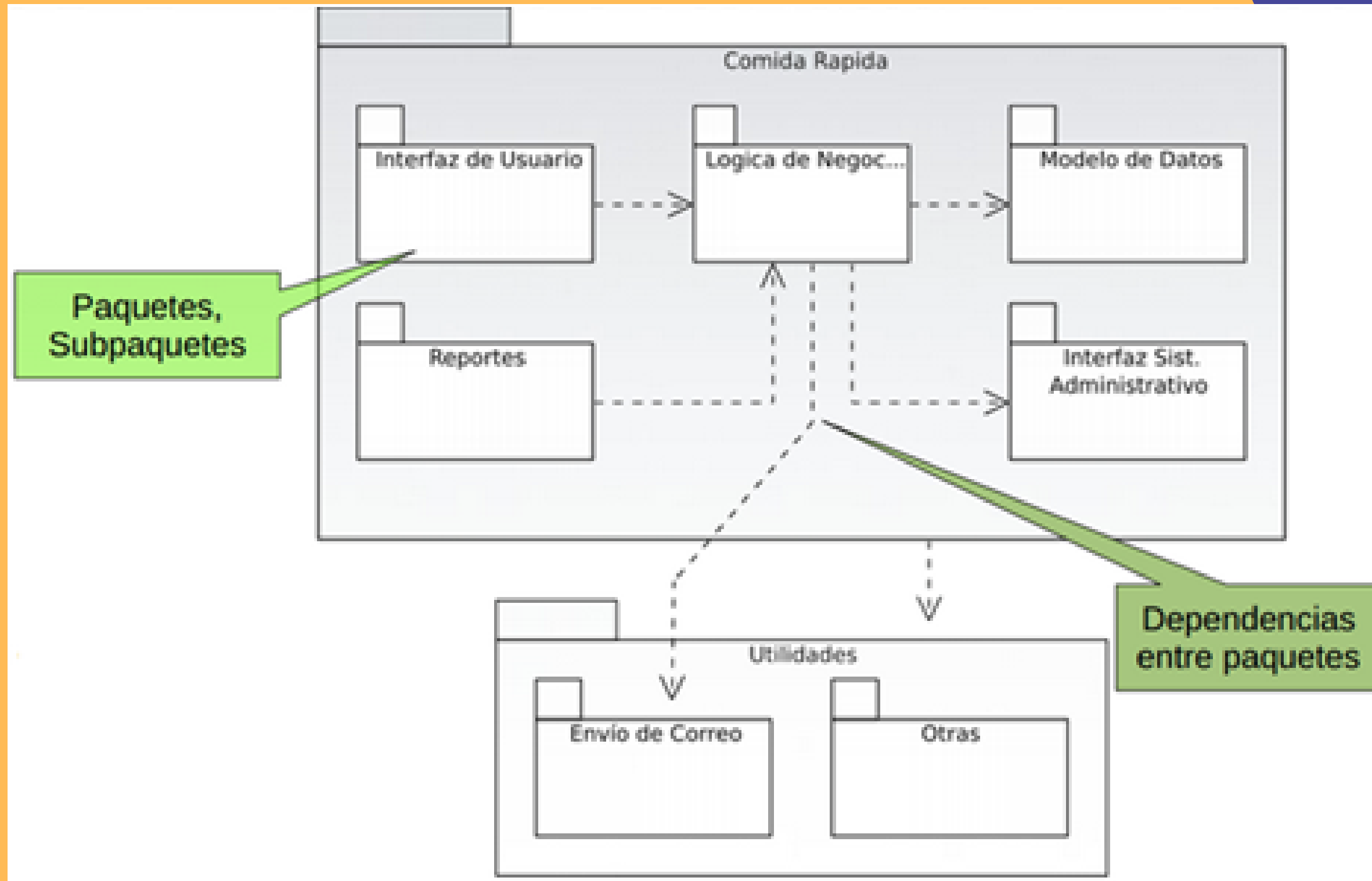
Principios:

- Capacidad de descomponer un sistema complejo
- Capacidad de componer a través de sus módulos
- Comprensión del sistema en partes

Divide y vencerás



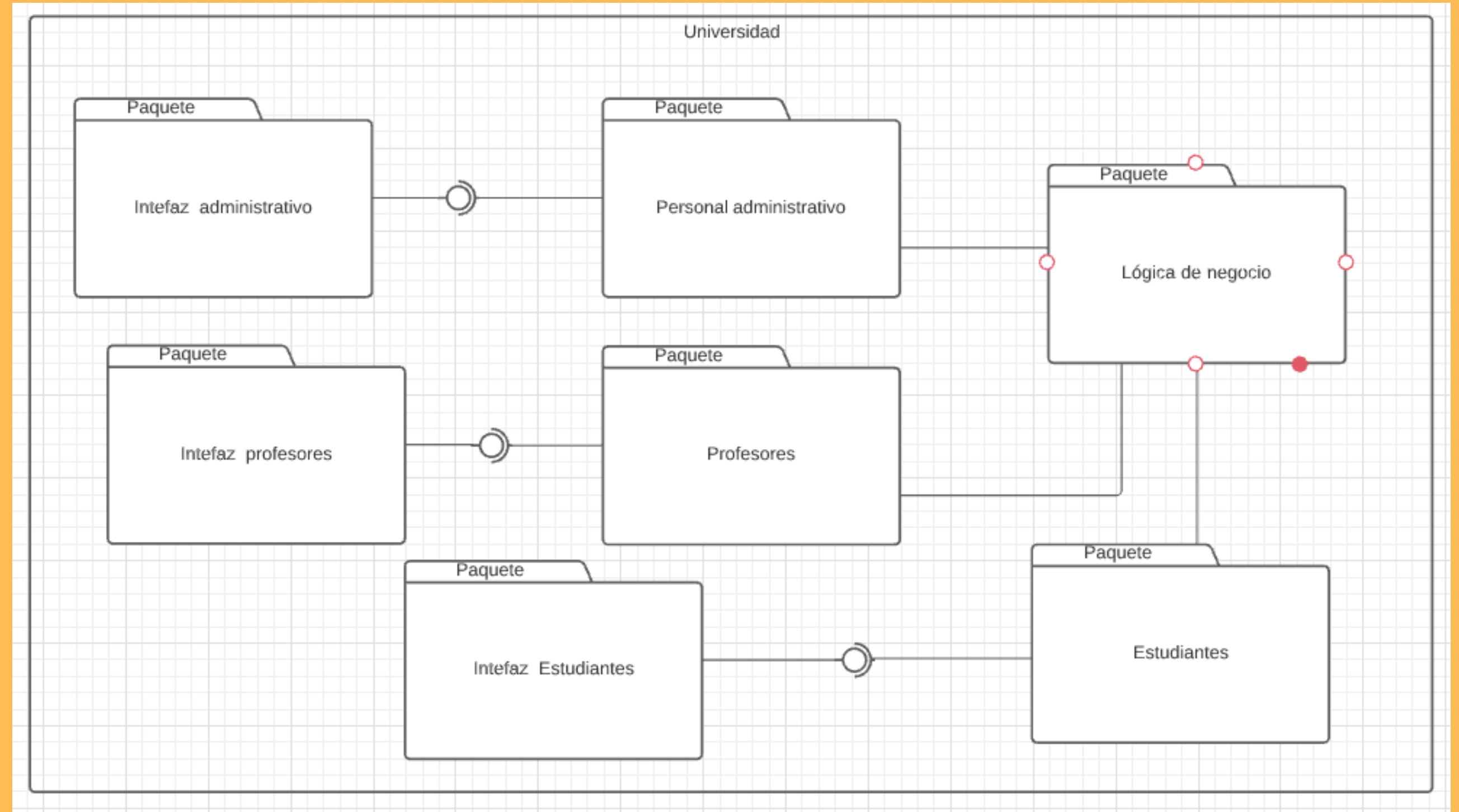
Modularidad



Modularidad

Descomponer el programa en módulos

Comunicación entre módulos



Ocultamiento

Algunos de los elementos de nuestro objeto no pueden ser vistos ni mucho menos modificados por otros objetos ajenos

Encapsulamiento: Por medio de permiso de accesibilidad en atributos

Ocultamiento: Por medio de métodos de la clase (getters y setters)

Ocultamiento

```
class Vehiculo{

    private String placa, marca, modelo;
    private int kilometraje;

    Vehiculo( String placa, String marca, String modelo,int kilometraje){
        setPlaca(placa);
        setMarca(marca);
        setModelo(modelo);
        setKilometraje(kilometraje);
    }
    /* Metodos Modificadores */
    public void setPlaca(String n){ placa = n; }
    public void setMarca(String a){ marca = a; }
    public void setModelo(String c){ modelo = c; }
    public void setKilometraje(int e){ kilometraje = e; }
    /* Metodos Accesores */
    public String getPlaca(){ return placa; }
    public String getMarca(){ return marca; }
    public String getModelo(){ return modelo; }
    public int getKilometraje(){ return kilometraje; }
```

Ejercicio

- Crear un diagrama que represente una aplicación que ofrezca servicios a usuarios

Ejercicio: Dueños de mascotas

1. Crear las clases persona y perro, la persona tendrá como atributos cedula, nombre y edad. En cuanto a la clase perro, esta tendrá como atributos raza y nombre.
2. Crear el método constructor en ambas clases.
3. Crear los métodos set y get en ambas clases.
4. Crear un nuevo atributo en la clase persona que se llame mascota. Creando métodos set y get para este atributo nuevo.
5. Instanciar 3 personas y 3 perros
6. Asignarle un perro a cada persona
7. Imprimir en pantalla: nombre de la persona junto con el nombre de su mascota



Ejercicio

Crear una relación de herencia, creando una clase padre y una clase que herede de ella

- Notas:
- Utilizar atributos protegidos
- utilizar métodos mágicos vistos (constructor, getters, setters, toString y call)
- Crear instancias del padre e hijo y mostrar en pantalla por medio del toString sus atributos



Arquitectura de software

¿Qué es ?

La arquitectura de software representa la estructura o las estructuras del sistema de software

Estructura a nivel de software y desarrollo



Estructura a nivel de software y desarrollo

- Requerimientos
- Análisis
- Diseño
- Documentación (SPMP, SRS, SDD)
- Desarrollo/Implementación
- Pruebas

Software Project Management Plans
Software Requirements Specification
Software Design Description
Software Architecture Document
Documento de Visión
User Documentation
Testing Documentation
Etc.



Introducción patrón MVC

¿Qué son los patrones?

Los patrones en la informática son guías o modelos que nos ayudan a organizar nuestro software

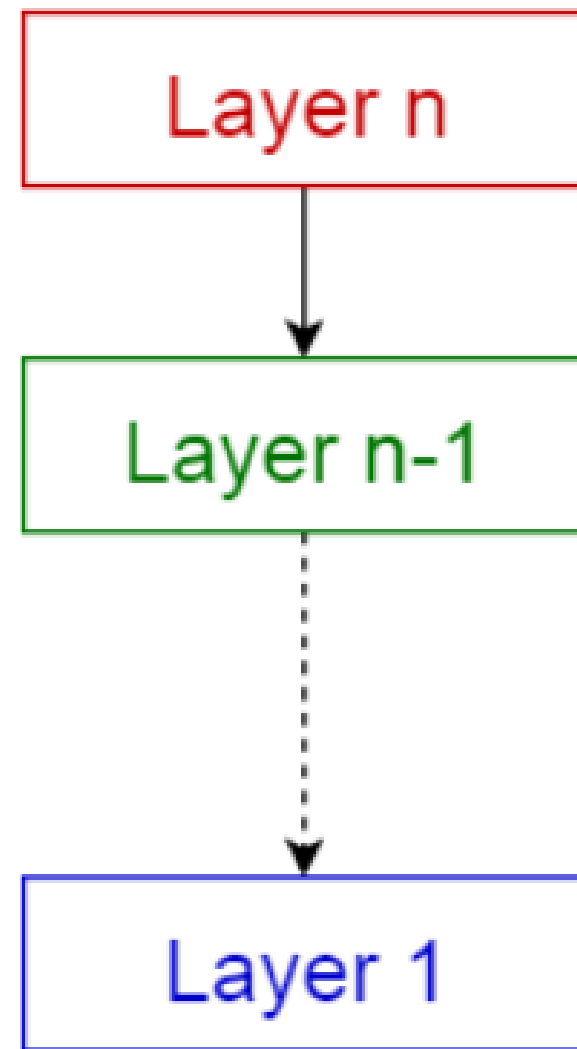


Tipos de patrones

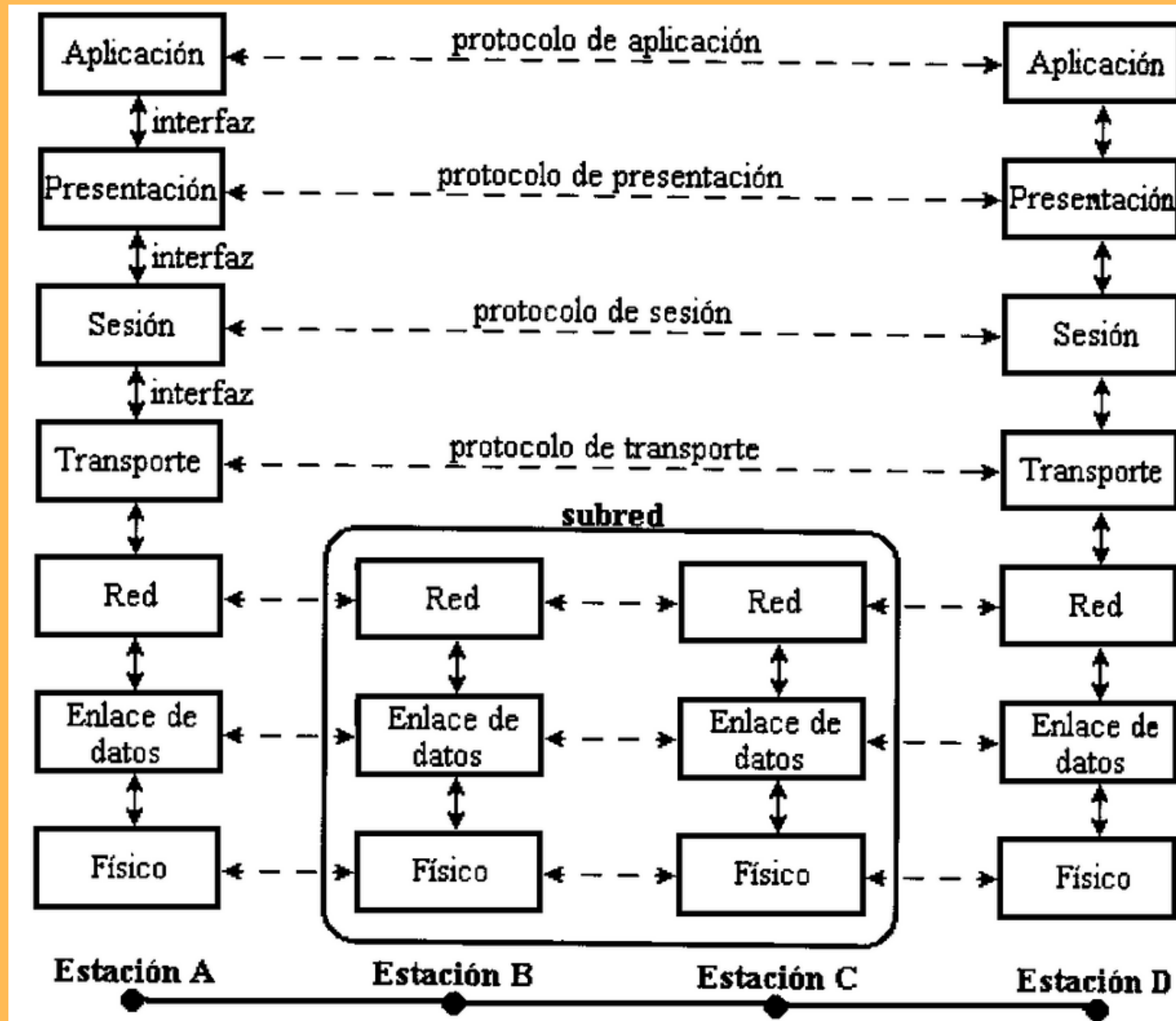
- Patrón de capas
- Patrón cliente-servidor
- Patrón maestro-esclavo
- Patrón de filtro de tubería
- Patrón de intermediario
- Patrón de igual a igual
- Patrón de bus de evento
- Modelo-vista-controlador
- Patrón de pizarra
- Patrón de intérprete



Patrón por capas



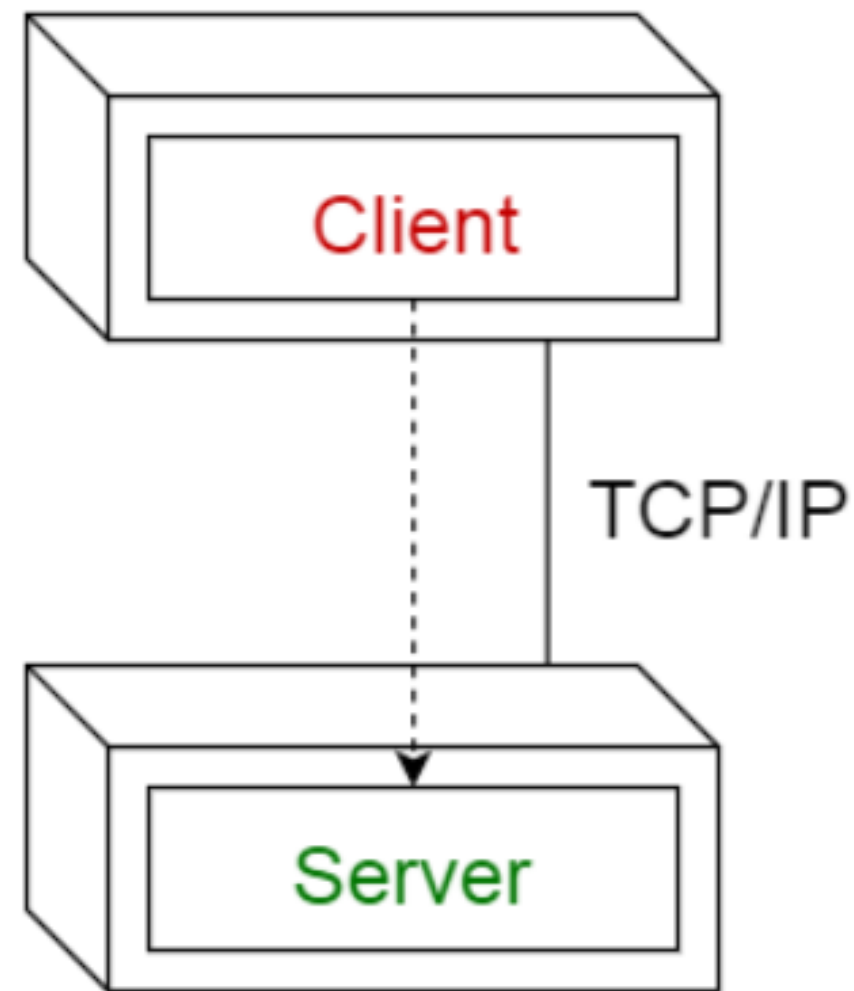
Patrón de capas



Cada capa provee servicios que serán consumidos por la capa $N + 1$



Patrón cliente-servidor



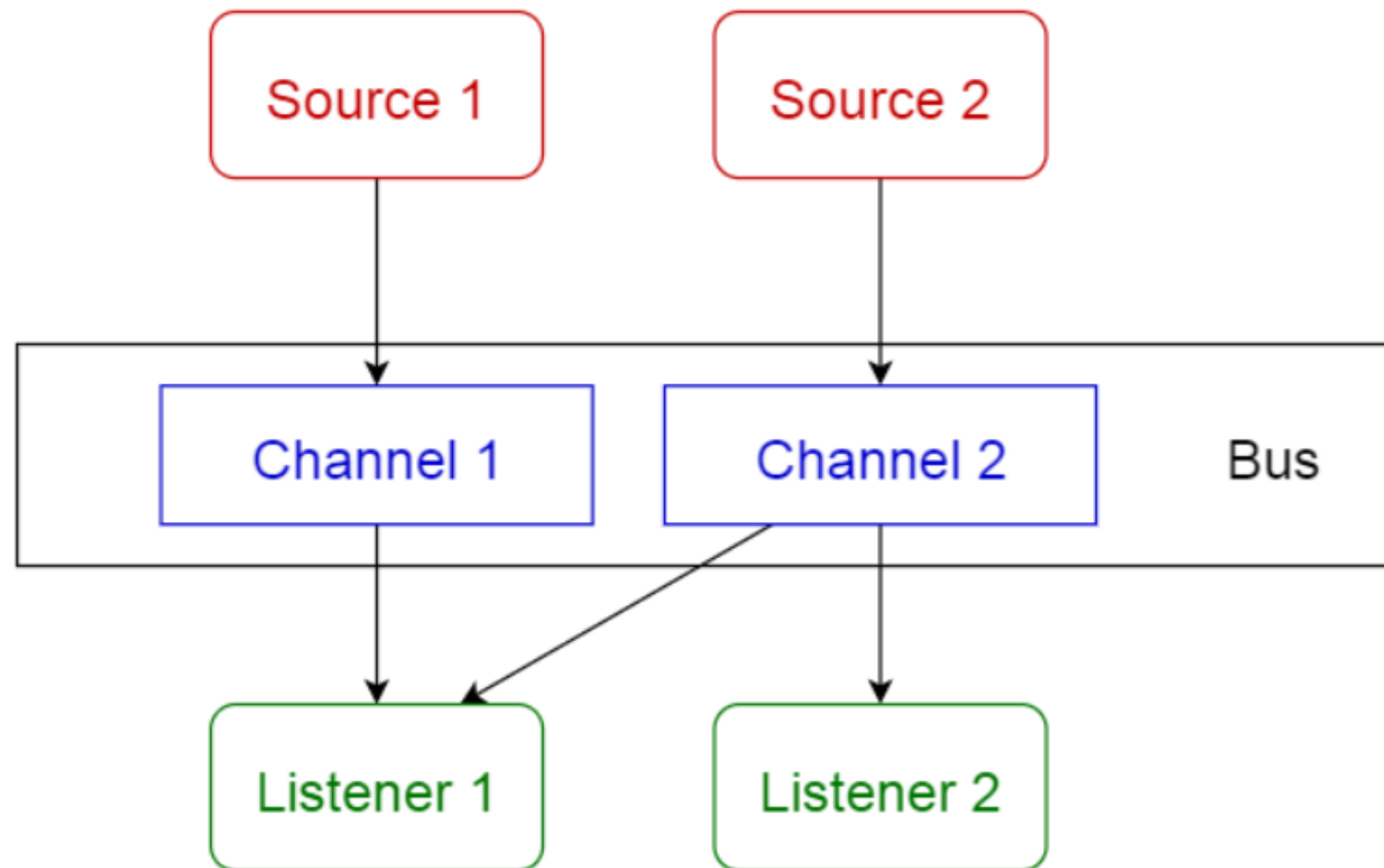
Patrón cliente-servidor

Componente servidor ofrecerá servicios a los que múltiples clientes puedan usar



Patrón bus de evento

También conocido
como patrón
publicador/suscriptor



Patrón de bus de evento



Patrón MVC

¿ Qué es el patrón MVC ?

Es un patrón de arquitectura de software que modela o guía la manera en cómo organizar y estructurar los componentes de un sistema de software, sus responsabilidades y las relaciones entre cada uno de ellos

- Fue descrito en el año 1979 por Trygve Reenskaug, hoy en día profesor emérito de informática de la Universidad de Oslo
- Se utilizaba para interfaces gráficas de usuario de escritorio





Patrón MVC

Hoy en día

- escritorio
- clientes
- servidores web
- servicios web
- Single Page Applications (SPA)

Lenguajes

- Smalltalk
- Java
- Ruby
- C++
- Python,
- PHP
- JavaScript
- NodeJS, etc.



Patrón MVC

La arquitectura MVC se basa en separar los componentes del software en 3 elementos:

- **M**odelo
- **V**ista
- **C**ontrolador

Patrón MVC

Modelo

Son datos del dominio, es decir, aquellas **entidades** que nos servirán para almacenar información del sistema

Encargado de gestionar el almacenamiento y recuperación de datos y entidades del dominio



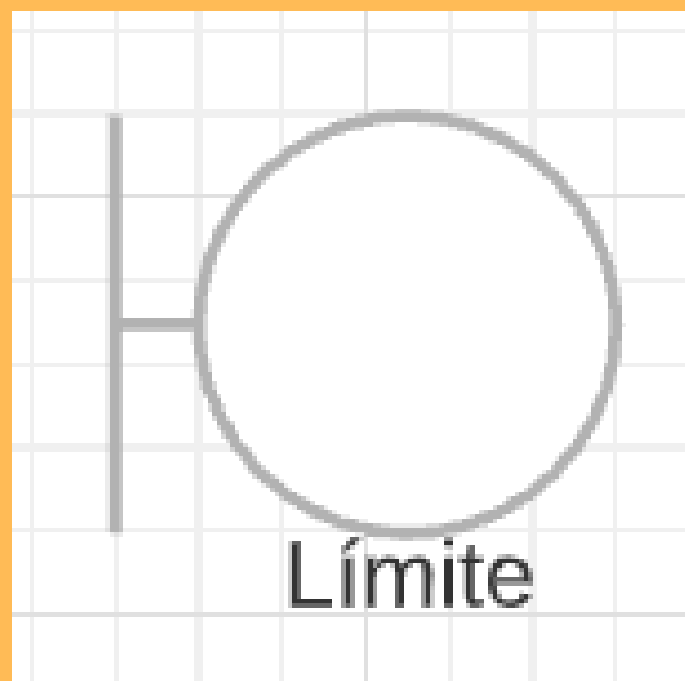
Persistencia!



Patrón MVC

Vista

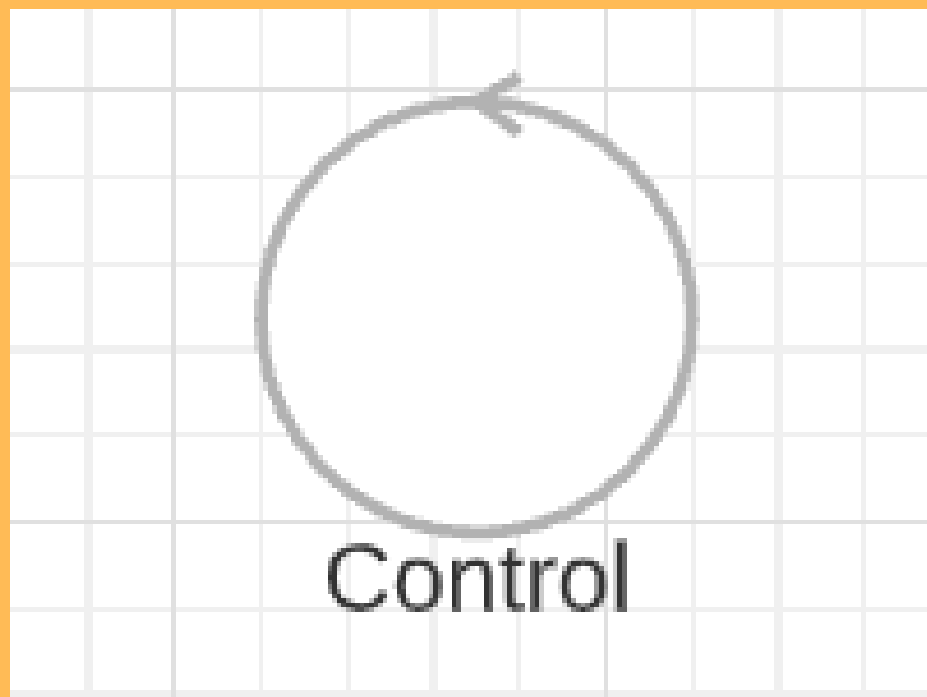
Los componentes de la Vista son los responsables de generar la interfaz de nuestra aplicación. (paginas, pantallas, o cualquier tipo de comunicación con el usuario visualmente)



Patrón MVC

Controlador

- Intermediarios entre el usuario y el sistema
- Realizarán también tareas de transformación de datos para hacer que los componentes de la Vista y el Modelo se entiendan
- valores de campos de un formulario recibidos mediante el protocolo HTTP, a objetos que puedan ser comprendidos por el Modelo



Lógica de negocio