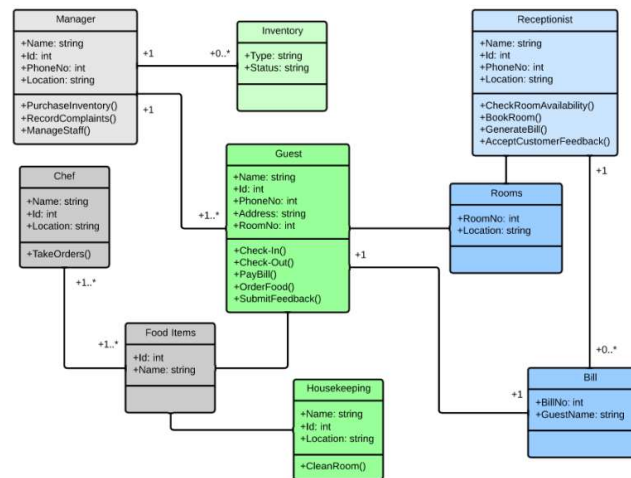


UT 4: DIAGRAMAS DE ESTADO



Entornos de Desarrollo

Miguel Trigueros Muñoz

Basado en el trabajo de Luis del Moral Martínez

versión 24,01

Bajo licencia CC BY- NC-SA 4.0

CONTENIDO

Introducción

1. Modelización
2. Desarrollo de software

POO

3. Bases Diseño y POO
4. Elementos POO
 1. Clases y objetos
 2. Mensajes y métodos
 3. Atributos y estado
 4. Constructores

5. Características

1. Abstracción
2. Encapsulación
3. Herencia
4. Polimorfismo

UML

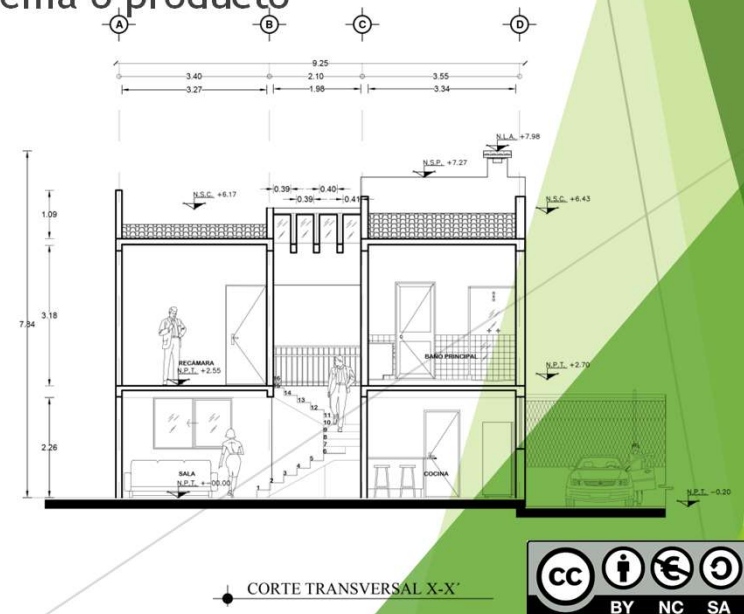
6. Diagramas de clases
7. Ejemplos

- *InteliJ*
- *Apuntes pdf*
- *ppt*

introducción

1. Modelización

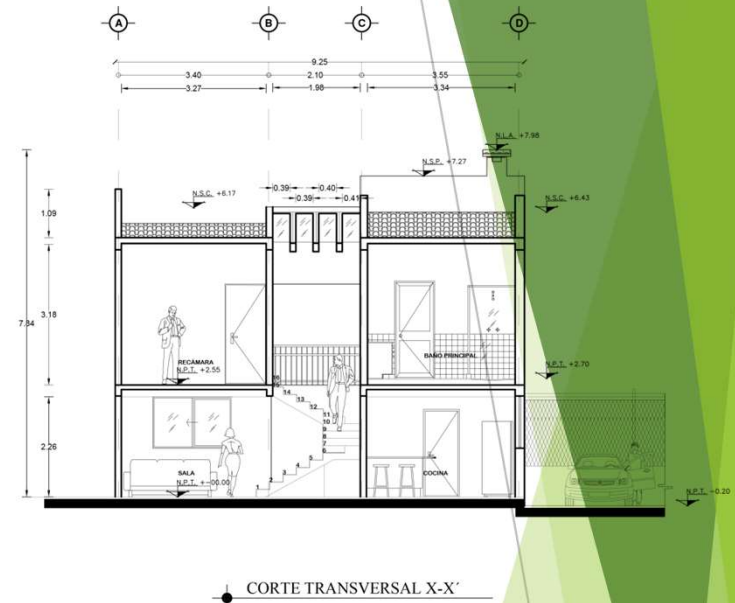
- ▶ Modelizar es simplificar la realidad
 - ▶ Abstracción: elegir elementos **relevantes**
 - ▶ Planos
 - ▶ Generales
 - ▶ Detallados
 - ▶ Nos guían en la construcción
 - ▶ Funciones:
 - ▶ Cómo es o queremos que sea nuestro sistema o producto
 - ▶ Mostrar estructura o comportamiento
 - ▶ Documentar las decisiones



1. Modelización

Razones para modelar:

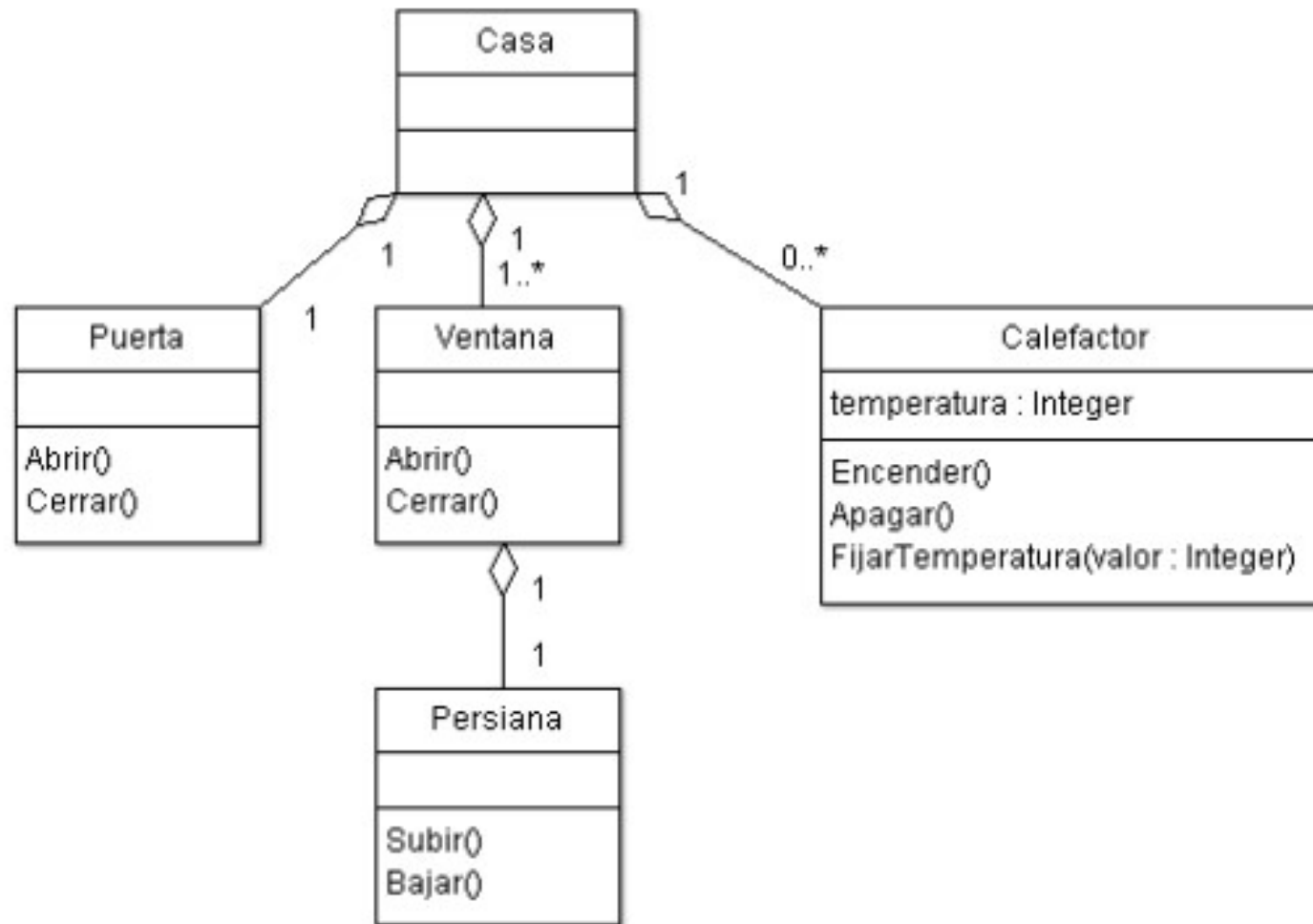
1. Clarificación de Requisitos
2. Visualización del Diseño
3. Identificación de Problemas de Diseño
4. Facilita la Toma de Decisiones
5. Alineación con Objetivos del Negocio
6. Ahorro de Recursos
7. Documentación Efectiva
8. Mejora la Colaboración



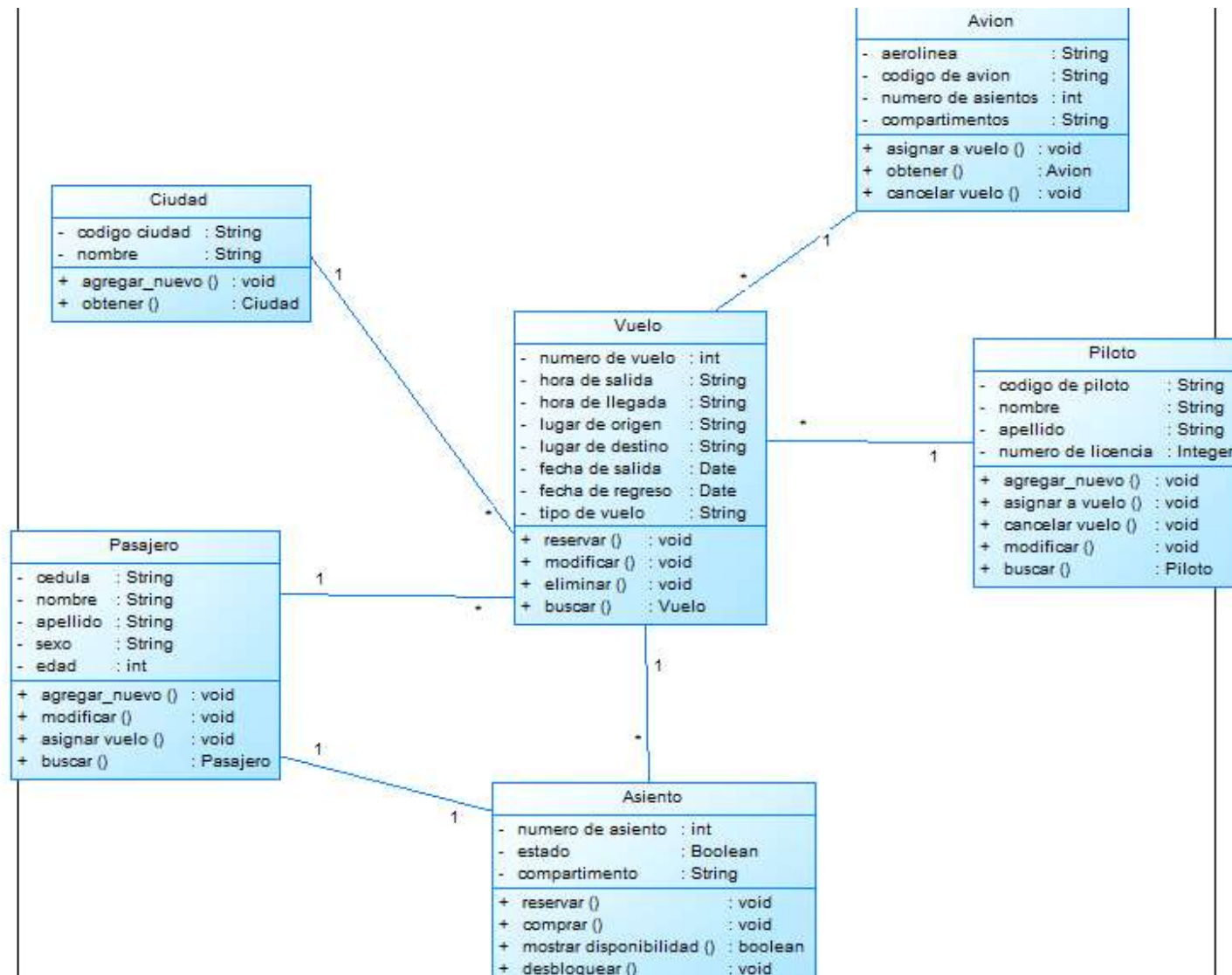
2. Desarrollo de software: documentos y diagramas

- ▶ En cada etapa del proceso de desarrollo de software se producen documentos:
- ▶ En la etapa de ANÁLISIS diagramas de CASOS DE USO
- ▶ En la etapa de DISEÑO diagramas **CLASES** y SECUENCIA
- ▶ Diagramas de estado:
 - ▶ De Clases
- ▶ Diagramas de comportamiento:
 - ▶ De Casos De Uso
 - ▶ De Secuencia

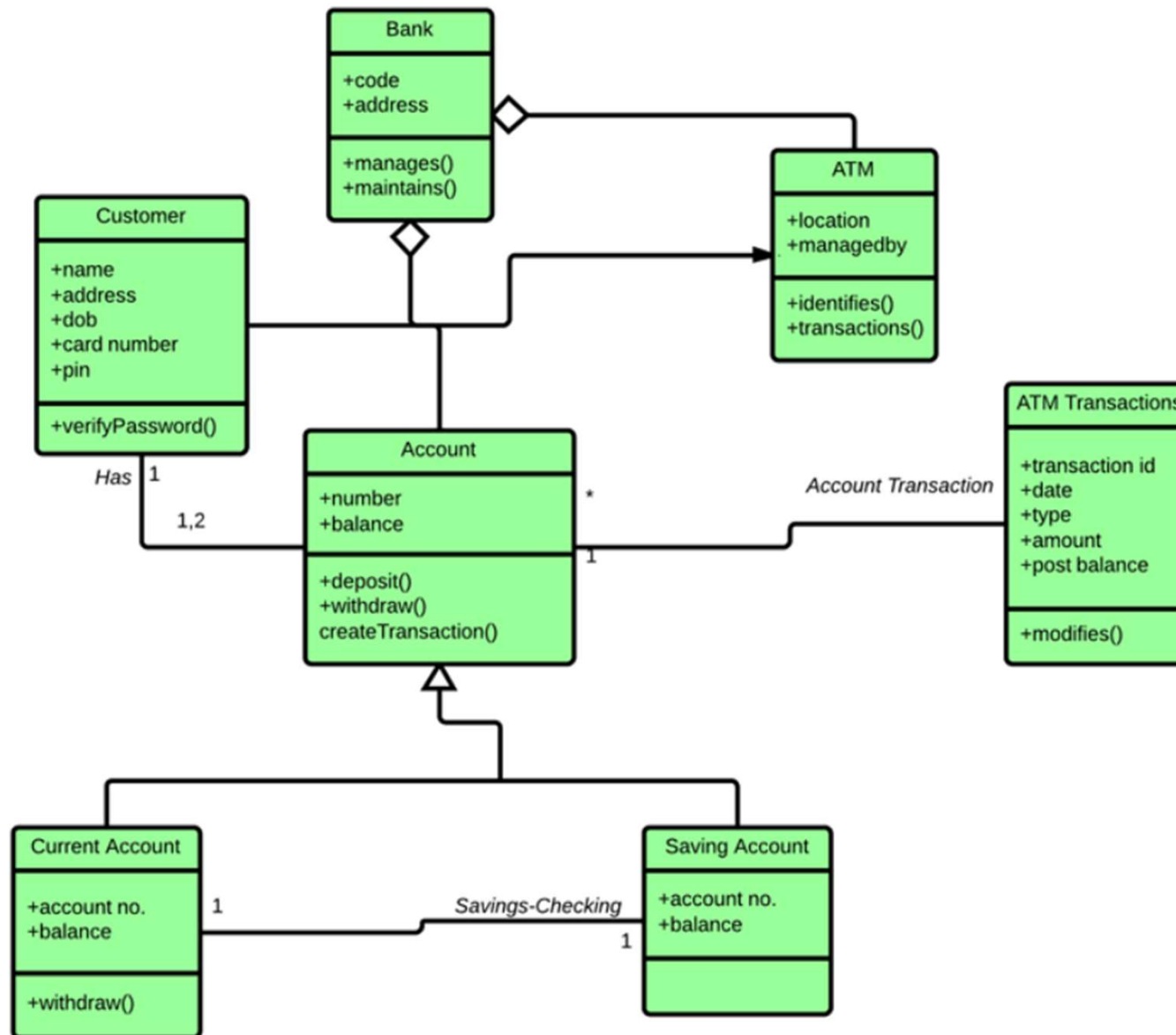
Ejemplos: Casa



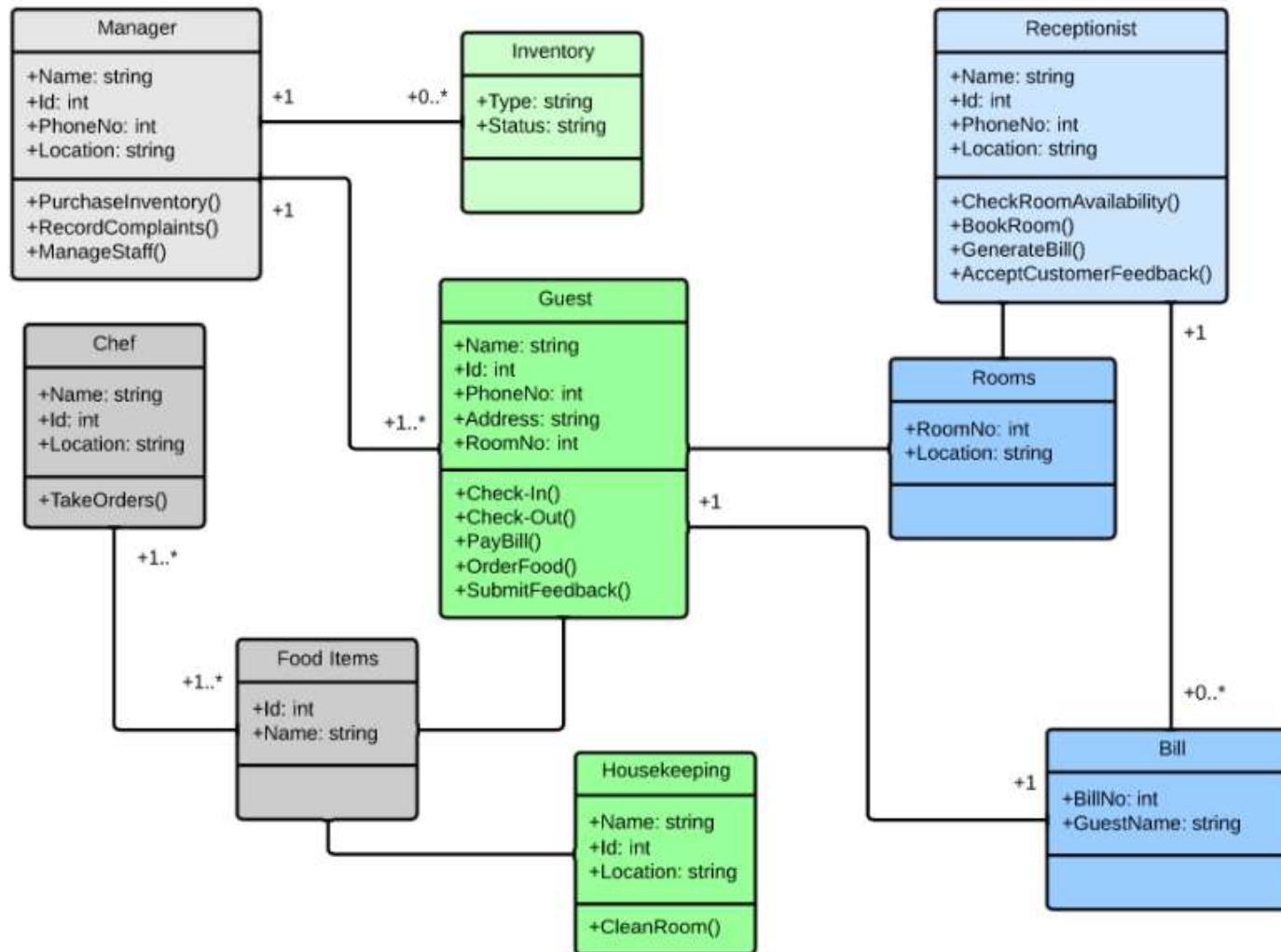
Ejemplos: Aerolínea



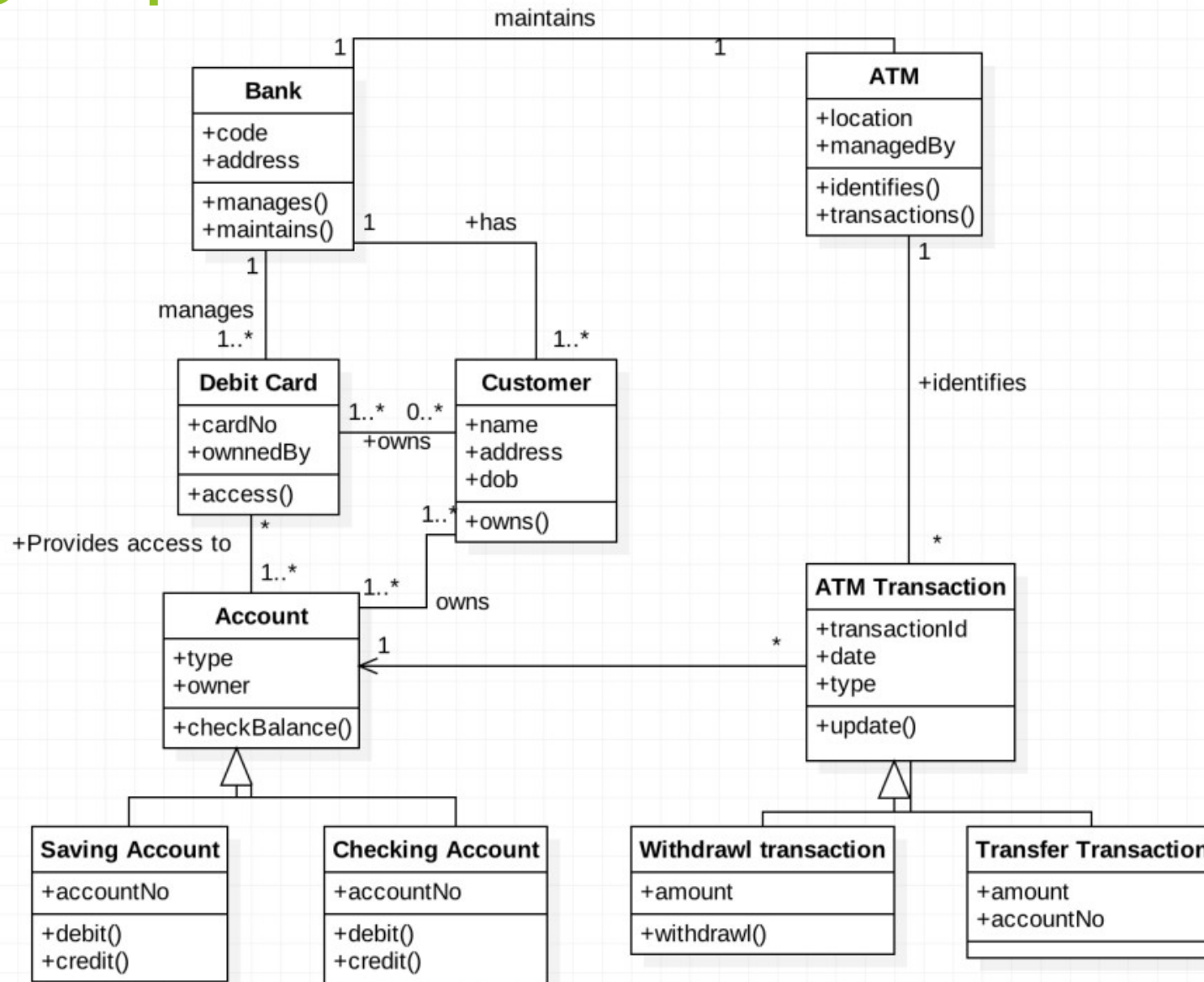
Ejemplos: Banco



Ejemplos: Hotel



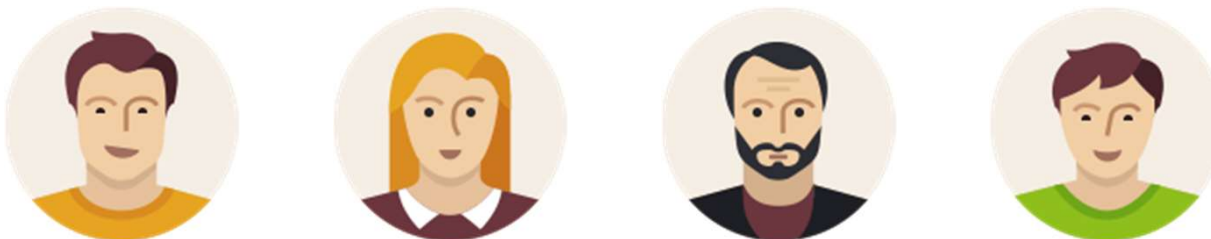
Ejemplos: Banco



Programación Orientada a Objetos

3. Bases del diseño **DOO** y la programación orientada a objetos **POO**

- ▶ Paradigma más ampliamente utilizado
 - ▶ Facilita el trabajo en equipo y la reutilización
- ▶ Abstracción de la realidad
- ▶ Envuelve información y comportamiento en **OBJETOS**
- ▶ Los objetos se construyen desde una plantilla llamada **CLASE**
- ▶ Una clase se compone de **información y comportamiento**
 - ▶ Atributos y métodos



Person
- ID - Name - Email
+ speak() + run() + walk()

3. Bases del diseño DOO y la programación POO

- ▶ Ciclo de vida de un programa orientado a objetos
 1. Creación de los objetos (a medida que son necesarios)
 2. El usuario interactúa con el programa y los objetos intercambian mensajes
 - ▶ Es posible que se creen nuevos objetos o se destruyan algunos existentes
 3. Los objetos que no son necesarios se destruyen (ya sea de forma programada o automatizada)
 - ▶ En Java existe un sistema encargado de la destrucción de objetos no utilizados (Garbage collector)
- ▶ ¿Qué significa que un objeto se cree o destruya?



Person
-ID
-Name
-Email
+speak()
+run()
+walk()

3. Bases DOO y POO

► Programación convencional

- Procedimientos - funciones
- Datos - variables
 - Globales
 - Locales (se pasan como parámetros)



► POO

- Agrupa ambos

► Atributos →

► Métodos →

Person

- ID
- Name
- Email

+speak()
+run()
+walk()

Pero al final seguimos teniendo
archivos de texto que
almacenan código
git BANCO//0
Ejecutar estructurada
vs POO

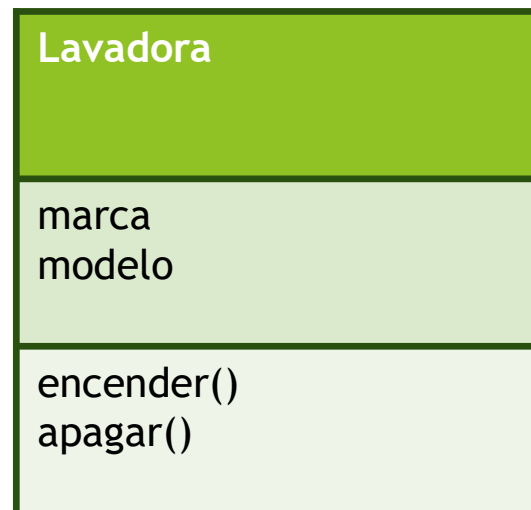
4. ELEMENTOS POO

- Clases - objetos
- Métodos - mensajes
- Atributos - estado

▶ Clases
▶ Objetos

▶ Métodos
▶ Mensajes

▶ Atributos
▶ Estado



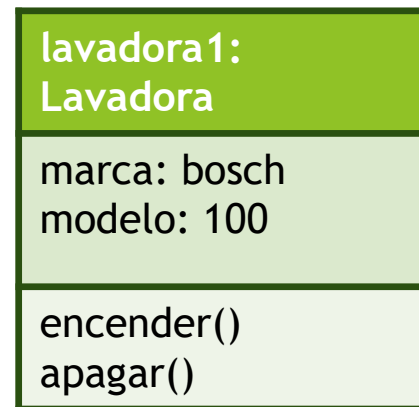
4. ELEMENTOS POO

► Clases

- Es un tipo de datos (string, int, char...)
 - Propiedades - atributos
 - Estructura de datos propia
 - Funcionalidad - métodos
 - Qué pueden hacer los OO



Instanciación



- Clases - objetos → ESTRUCTURA
- Métodos - mensajes
- Atributos - estado

► Objetos

- Instancia de la clase
 - GIT//1 cuenta
- Atributos con valores
- Pueden ser múltiples



```
class CuentaBancaria {  
    /*  
    *  
    * | CuentaBancaria |  
    * |  
    * | numeroCuenta |  
    * | saldo |  
    * |  
    * | transferir |  
    * | imprimirSaldo |  
    * |  
    *  
    */  
}
```



4. ELEMENTOS POO

► Mensajes

► Invocar un método de otro O

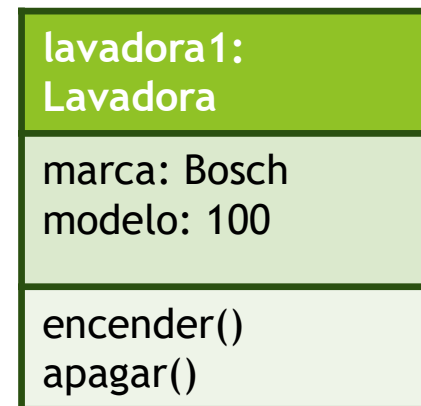
- Agente activo
- Agente pasivo
- GIT//2 cuenta



lavadora1.encender()

► Métodos (FUNCIONALIDAD)

- Cambian estado del objeto
 - encender() git//4 método
- Calculan cierto valor
 - miCalculadora.sumar() GIT//5
- Ciclo de vida git//3
 - Constructor
 - Destructor



4. ELEMENTOS POO

- Clases - objetos
- Métodos - mensajes
- Atributos - estado

► Atributos

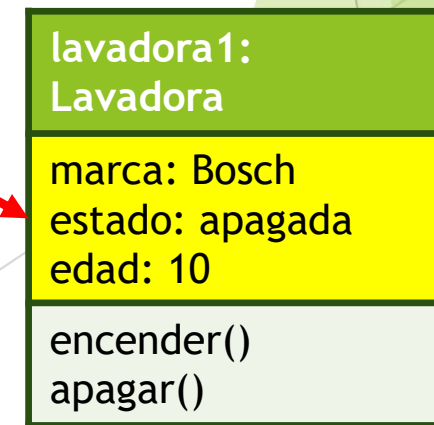
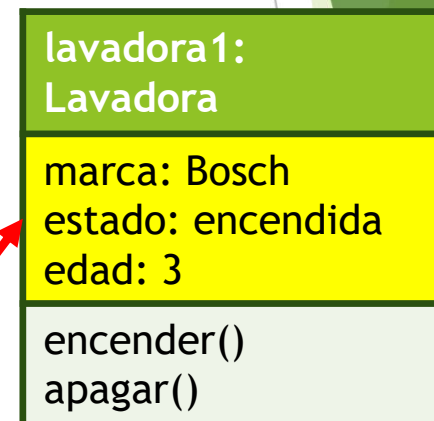
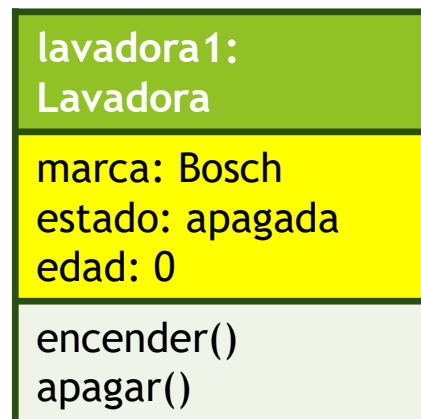
- Estructura de **datos** de la clase
 - Información permanente
 - (marca)
 - Cambiante
 - (edad)

► Estado

- Conjunto de atributos
- En un momento dado
- Foto fija
- **git//6 ejecutar**



Instanciación
→

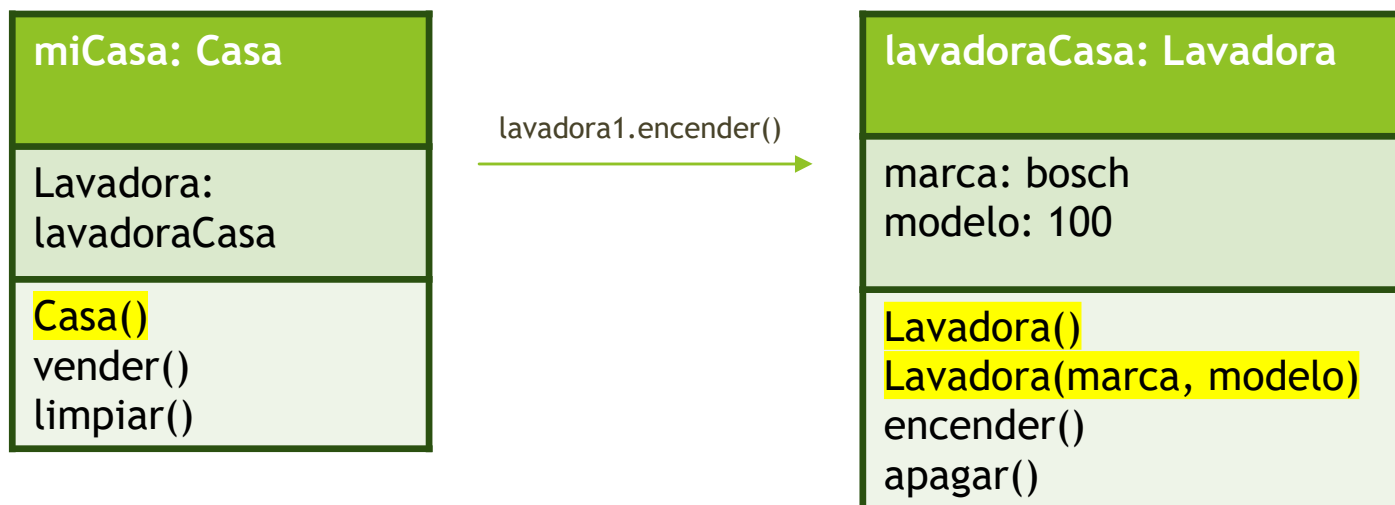


4. ELEMENTOS POO

- Clases - objetos
- **Métodos - constructor**
- Atributos - estado

► Constructor

- Método especial que instancia el objeto
- Existe siempre uno por defecto
 - (que es eliminado si escribo yo otro)
- Podemos añadir nuevos **constructores** (*sobrecarga*) **git//7**



Proponer actividad A1:
Diagrama de clases
Banco, Calculadora, Instituto con Draw.io

5. CARACTERÍSTICAS

- ▶ Abstracción
- ▶ Encapsulación
- ▶ Herencia
- ▶ Polimorfismo

5. CARACTERÍSTICAS

► Abstracción

- Modelo un objeto del mundo real
- Se seleccionan **ciertas** características
- Dependiendo del contexto (análisis, casos de uso, requisitos...)

Coche
<ul style="list-style-type: none">- equipo- piloto- carreras
<ul style="list-style-type: none">+ repostar()+ cambiarRueda()



Coche
<ul style="list-style-type: none">- marca- modelo- edad
<ul style="list-style-type: none">+ comprar()+ vender()

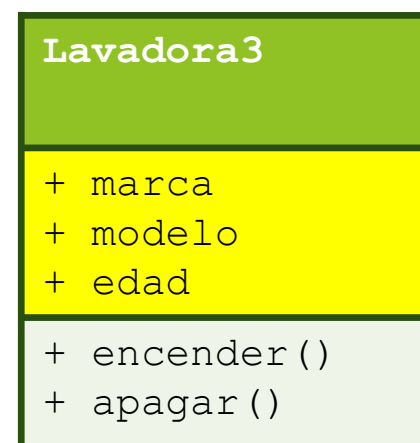
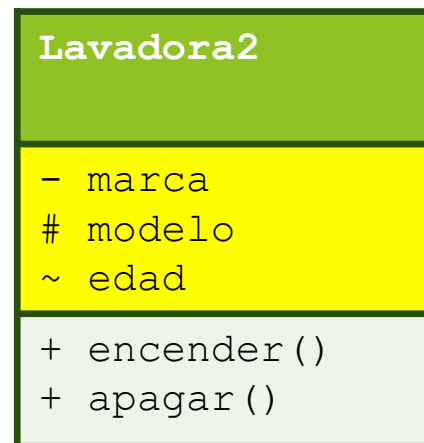
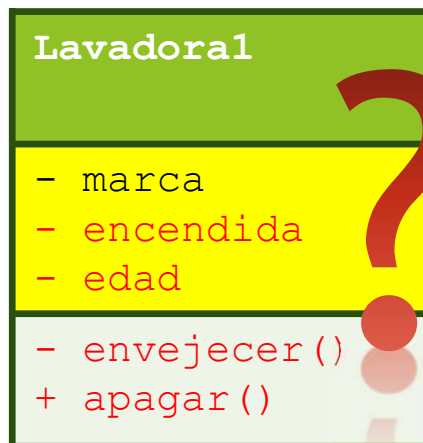
Coche
<ul style="list-style-type: none">- propietario- matrícula
<ul style="list-style-type: none">+ multar()+ anular()

5. CARACTERÍSTICAS

► ENCAPSULAMIENTO - Visibilidad

- Los objetos de una clase pueden ver o no los atributos de otra
- El acceso a los atributos y métodos de un objeto está restringido según unos criterios:
 - git
- A continuación, se muestran los modificadores de visibilidad en java junto a su símbolo UML:

- + public
 - Accesible
- - private
 - No accesible
- # protected
 - Solo **subclases, aunque estén en otro paquete**
- ~ (package)  por defecto  M0
 - Solo clases del mismo paquete



**Proponer A2:
Codifica las
clases coche
cambiando
también la
visibilidad**

Diapositiva 23

M0

Si no se escribe ningún modificador de visibilidad JAVA adjudica la visibilidad de paquete por defecto.

Miguel; 2024-02-04T07:15:30.802

5. CARACTERÍSTICAS

► HERENCIA

- Consiste en crear nuevas clases extendiendo clases existentes
- Permite la **reutilización** de código
- La clase Padre dona **TODOS** sus atributos y métodos a la clase Hija

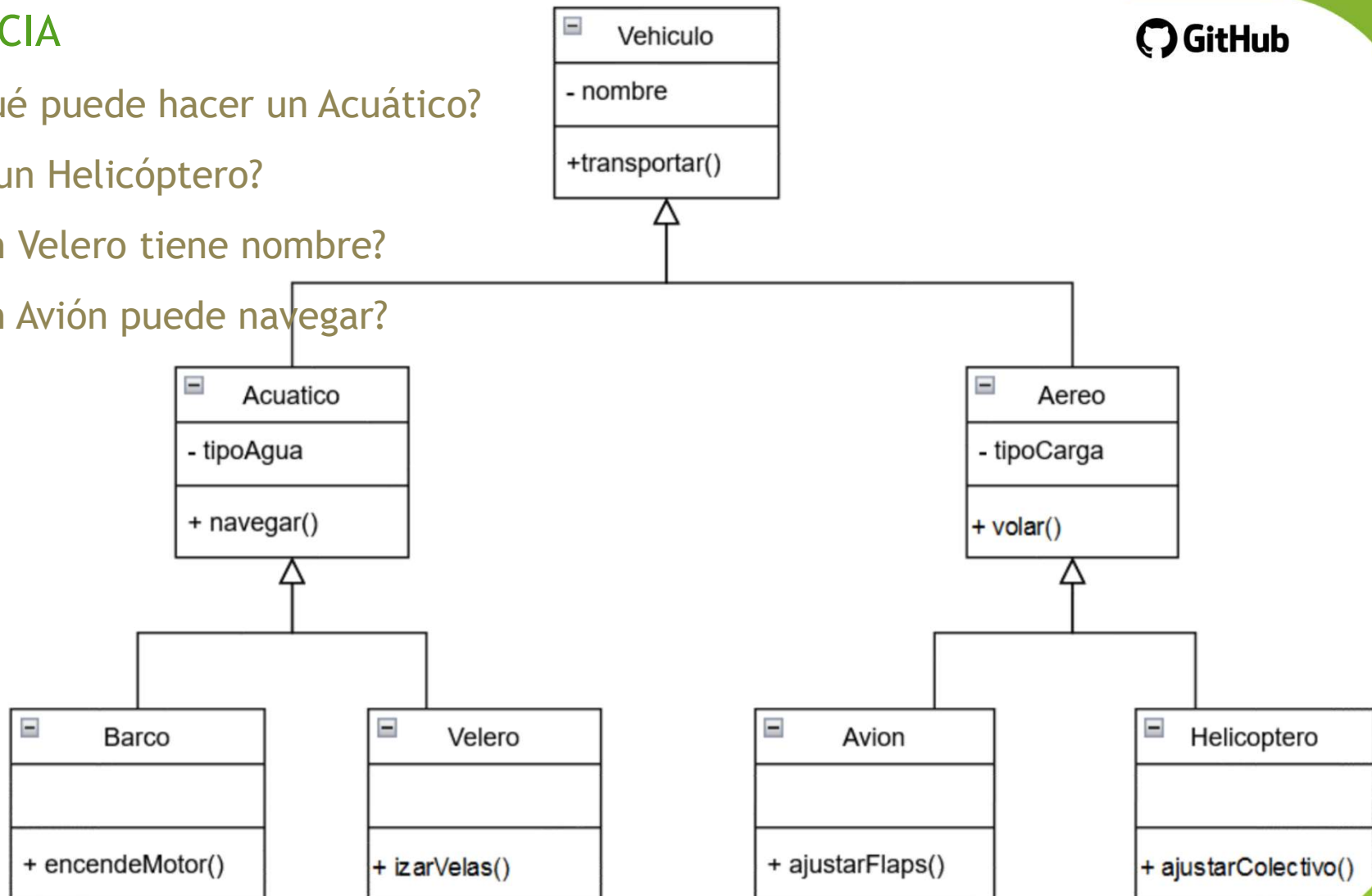
Palabra servada "extends"

```
class Barco extends Acuatico{}
```

5. CARACTERÍSTICAS

► HERENCIA

- ¿Qué puede hacer un Acuático?
- ¿Y un Helicóptero?
- ¿Un Velero tiene nombre?
- ¿Un Avión puede navegar?



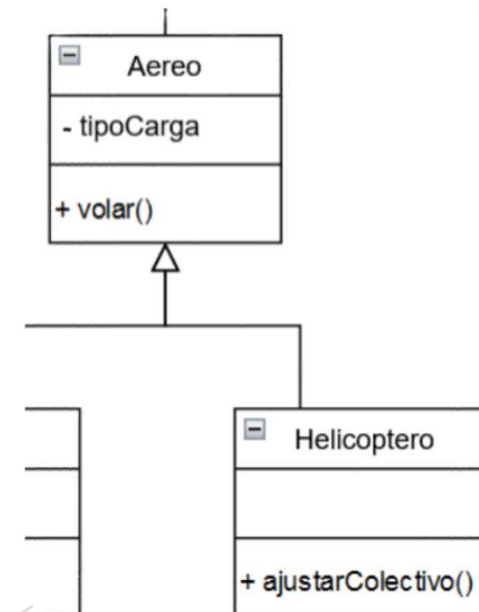
5. CARACTERÍSTICAS

► HERENCIA

- Consiste en crear nuevas clases extendiendo clases existentes
- Permite la **reutilización** de código
- La clase Padre dona **TODOS** sus atributos y métodos a la clase Hija
 - **PERO** La visibilidad de los miembros de la clase Padre afecta al acceso a dichos miembros de la clase Hija
 - Modificador *protected*: permite exponer atributos y métodos a las **subclases**, aunque estén en otro paquete

Palabra servada "extends"

```
class Barco extends Acuatico{}
```

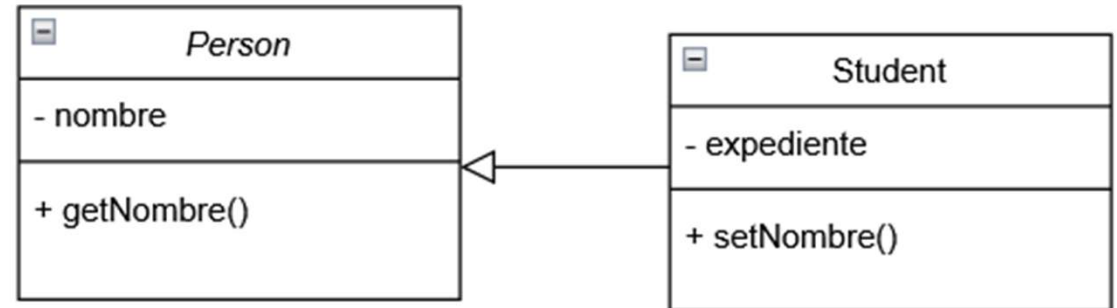


5. CARACTERÍSTICAS

► HERENCIA y VISIBILIDAD

- Los miembros privados de la clase Padre no son accesibles para la clase hija.

- Student no ve su nombre
- Pero sí puede getNombre()



- ¿Qué pasaría si fuera *protected*?

```
public class Person {  
  
    //Atributos  
  
    private String nombre = "Alfredo";  
  
    //Métodos  
  
    public String getNombre(){  
        return nombre;  
    }  
}
```

```
3 public class Student extends Person{  
4  
5     //Atributos  
6  
7     private String expediente = "12234";  
8  
9     //Métodos  
10  
11     public String verNombre(){  
12         return nombre;  
13         return getNombre();  
14     }  
15 }
```

An orange arrow points from the `return nombre;` line in the `verNombre()` method to the `nombre` attribute, highlighting that the child class cannot access the private attribute of the parent class.

5. CARACTERÍSTICAS

► HERENCIA

- Consiste en crear nuevas clases extendiendo clases existentes
- Permite la **reutilización** de código
- La clase Padre dona **TODOS** sus atributos y métodos a la clase Hija
 - **PERO** La visibilidad de los miembros de la clase Padre afecta al acceso a dichos miembros de la clase Hija
 - Modificador *protected*: permite exponer atributos y métodos a las **subclases**, aunque estén en otro paquete
- Las subclases **Hijas** pueden modificar su comportamiento frente a la clase **Padre** o superclase
 - Polimorfismo

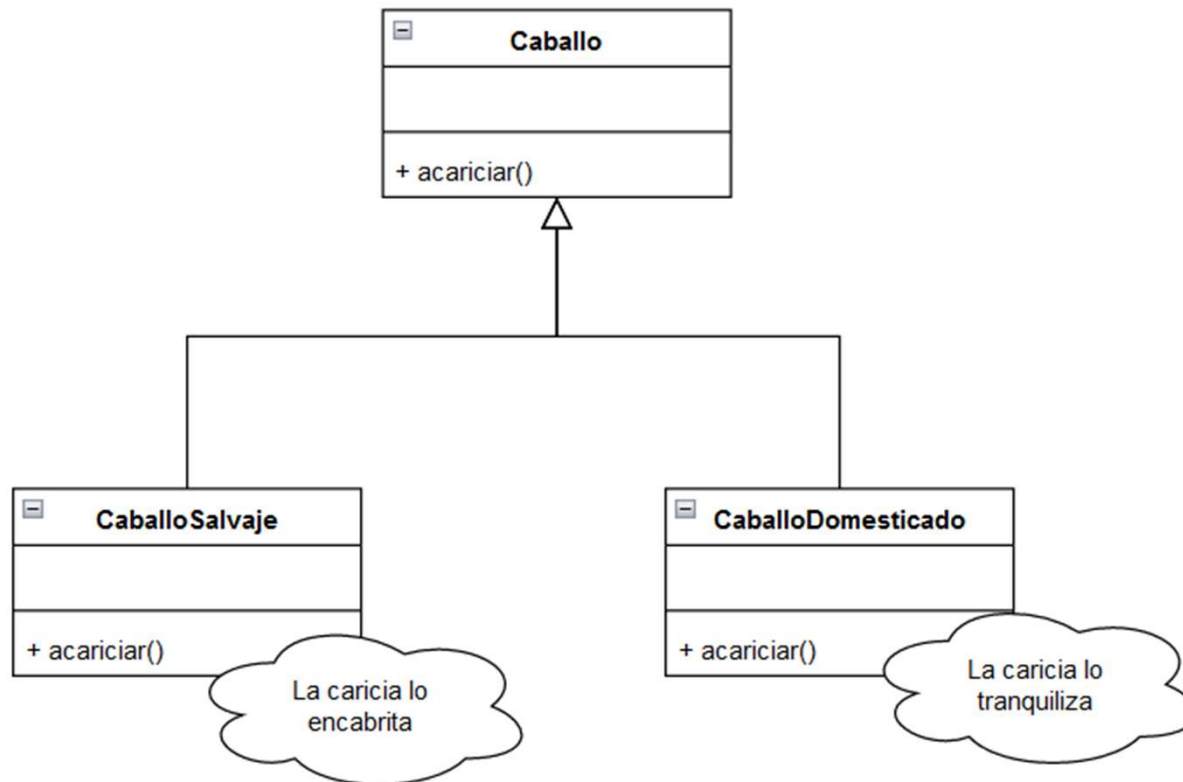
Palabra servada "extends"

```
class Barco extends Acuatico{}
```

5. CARACTERÍSTICAS

► HERENCIA y POLIMORFISMO

- Las subclases pueden adaptar los métodos de la clase a su realidad
- Dos subclases reescriben el mismo método de maneras diferentes
- La llamada al método común produce comportamientos distintos según la subclase a la que corresponda el objeto

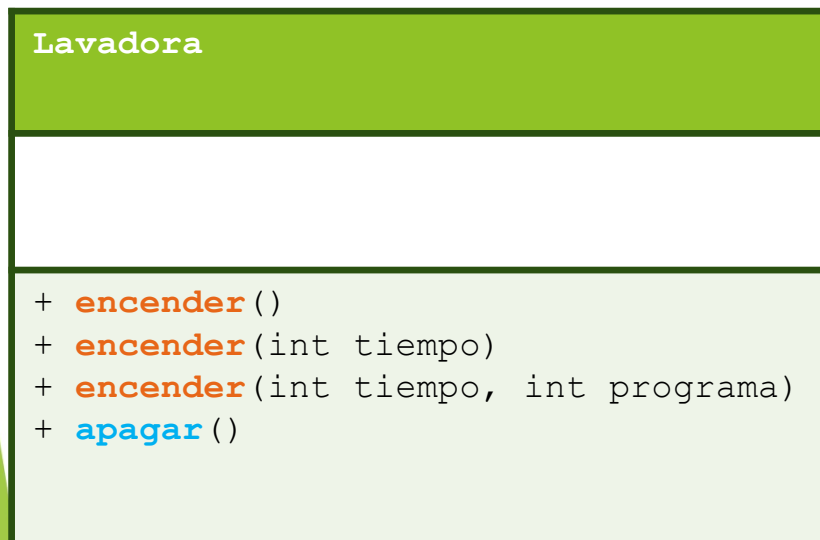


5. CARACTERÍSTICAS

► POLIMORFISMO

► Sobrecarga vs sobrescritura

- **Sobrecarga**: mismo método con diferentes implementaciones según número y tipo de parámetros
- **Sobrescritura**: en la subclase se reescribe un método de la superclase



Proponer A3:
Modela e
implementa una
herencia de dos
niveles con
sobrescritura

UML

6. DIAGRAMAS DE CLASES UML

► Tabla para la clase con tres celdas:

► Nombre

► Atributos

► Visibilidad

► :tipo

► [multiplicidad]

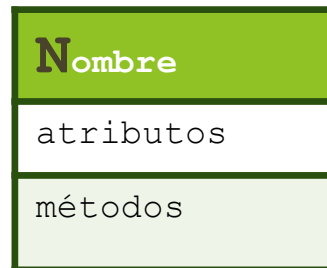
► =valor

► Métodos

► Visibilidad

► parámetros(tipo)

► :tipo retorno



► Se deben nombrar con la notación CamelCase en mayúsculas

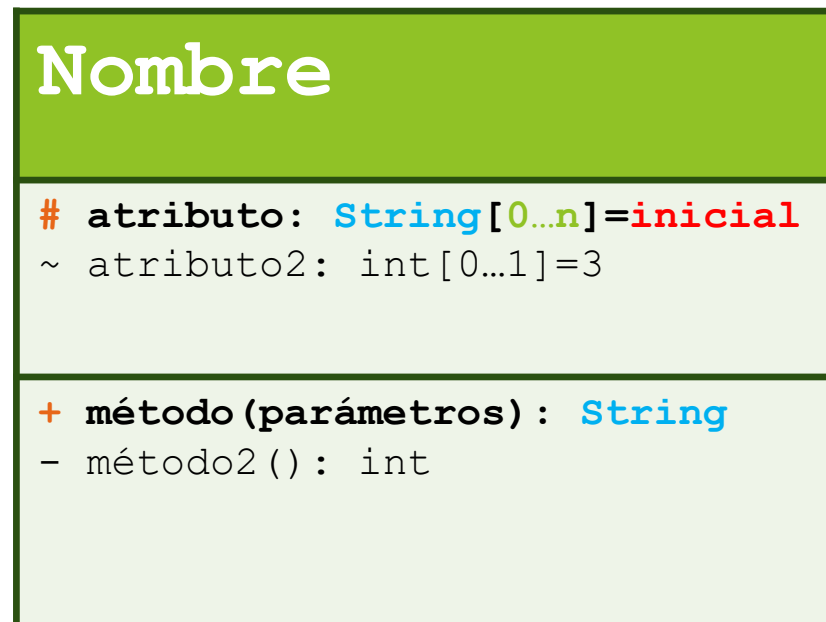
► Casa, Coche, Fichero

► En Java, el nombre de la clase debe ser el mismo que el nombre del fichero

► Atributos: suelen ser privados

► Métodos: suelen ser públicos

► camelCase: abrir(), sumar()

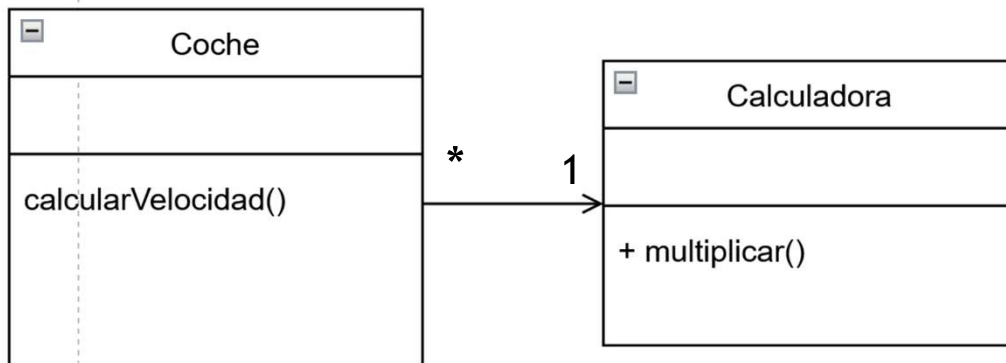


Proponer A4:
Modela 3 clases
con todos los
elementos del
estándar UML

6. DIAGRAMAS DE CLASES

► Relaciones entre clases

- **Asociación:**
- relación semántica entre los objetos
 - Nombre
 - Multiplicidad
 - Navegabilidad
 - No hay pertenencia



```
// Ejemplo de asociación unidireccional
public class A {
    //Atributos
    //Métodos
    public probar(){
        b.hacer();
    }
}

public class B {
    //Atributos
    //Métodos
    public hacer(){}
}

public class Main{
    public A a = new A();
    public B b = new B();
}
```

6. DIAGRAMAS DE CLASES

► Relaciones entre clases

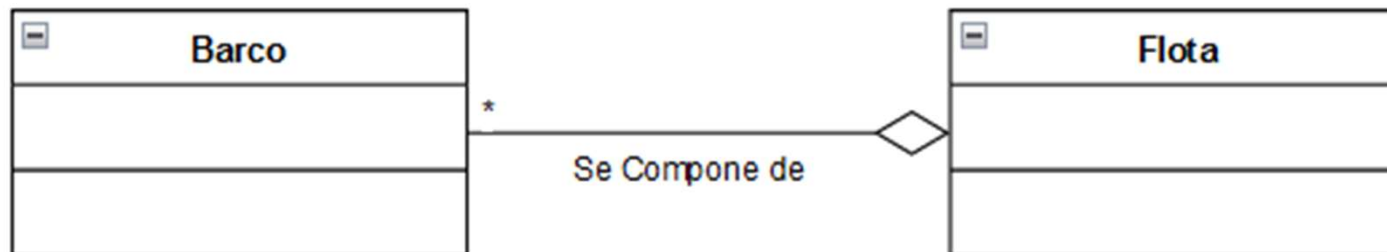
► Agregación:

- un objeto del todo tiene objetos de la parte
 - Nombre
 - Multiplicidad
 - Navegabilidad
- Pertenencia no muy fuerte
- La parte puede existir sin el todo
- El todo puede dejar de tener una de las partes

```
// Ejemplo de agregación
public class A {
    //Atributos
    private B bDeA = b;
    // ...
}

public class B {
    //Atributos
    // ...
}

public class Main{
    public A a = new A();
    public B b = new B();
}
```



6. DIAGRAMAS DE CLASES

► Relaciones entre clases

► Composición:

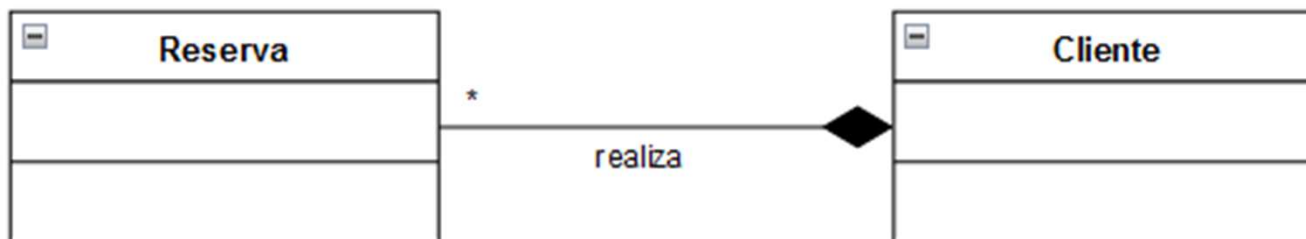
► La parte tiene que formar parte del todo

- Nombre
- Multiplicidad
- Navegabilidad

► Pertenencia muy fuerte

► La parte no puede existir sin el todo

```
// Ejemplo de composición
public class A {
    //Atributos
    private B b;
    //Constructor
    public A() {
        b = new B();
    }
    //...
}
```

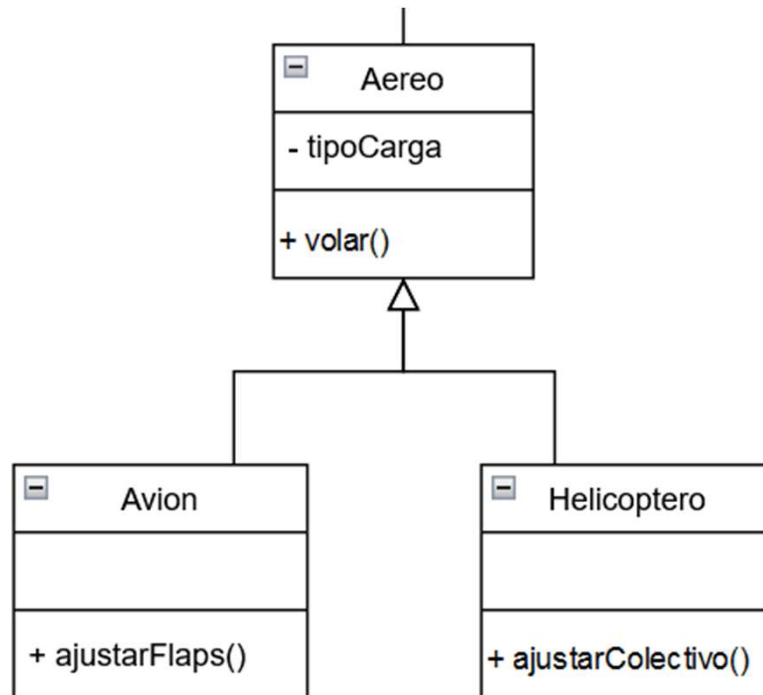


6. DIAGRAMAS DE CLASES

► Relaciones entre clases

► Herencia:

- La subclase tiene los elementos de la superclase



// Ejemplo de herencia

```
public class A {  
    // ...  
}
```

```
public class B extends A {  
    // ...  
}
```

6. DIAGRAMAS DE CLASES

► Relaciones entre clases

► Realización:

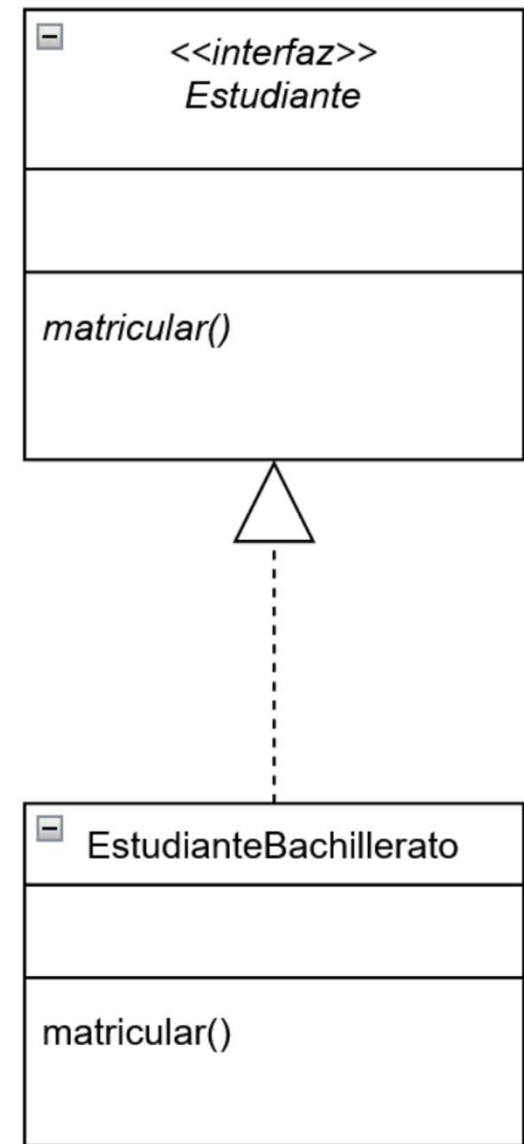
- Es una especie de herencia
- La clase (“subclase”) que realiza la interfaz (“superclase”) debe implementar los métodos que presenta el contrato de la interfaz

► Interfaces

- Clases totalmente *abstractas*
- Solo tienen métodos

```
// Ejemplo de realización (implementación)
public interface Interface {
    //El método solamente se declara
    public void metodo();
}

public class A implements Interface {
    public void metodo() {
        // Implementación del método
    }
}
```



6. DIAGRAMAS DE CLASES

► Modificadores habituales

► *abstract*

- Método que no se implementa en la superclase
- Debe implementarse en la subclase
 - (se debe *sobrescribir*)
- La clase que contiene un método abstracto debe declararse también abstracta

```
public abstract class Figura {  
    public abstract double calcularArea();  
}
```


6. DIAGRAMAS DE CLASES

► Modificadores habituales

► static (a veces subrayado en UML, mejor escribir static)

- En java **NO** se puede aplicar a la clase completa

```
no usages new *  
public static class Flota {  
  
}
```



- Elemento (método o atributo) propio de la clase y no de sus objetos
- Existen sin instanciar la clase

```
public class Ejemplo {  
    public static int contador = 0;  
    public static void incrementarContador() {  
        contador++;  
    }  
}
```


6. DIAGRAMAS DE CLASES

► Modificadores habituales

► final (a veces subrayado en UML, mejor escribir final)

- Indica que el elemento no puede ser modificado
- Si es un atributo, su valor es constante
- Si es una clase, no puede tener descendencia (no hereda)
- Método que no se puede sobrescribir al crear una subclase

```
public class Ejemplo {  
    public final int valorConstante = 10;  
    public final void metodoFinal() {  
        // código del método  
    }  
}
```

7. EJEMPLOS

▶ PASOS

1. Detectar las clases

- ▶ Buscar en los sustantivos
- ▶ Deben tener atributos y métodos...

2. Buscar las relaciones

- ▶ Los clientes REALIZAN pedidos...
- ▶ Los vehículos RECORREN rutas...

3. Analizar posibles generalizaciones (herencia)

- Se comparten atributos o métodos en distintas clases

4. Si no se indica expresamente, detalla un tipo de dato adecuado

Nombre	Nombre ≥ 1	Nombre de la clase asociada en la que se enseña una determinada habilidad en la clase o asignatura	
Exposiciones en clase	Exposiciones ≥ 1 Exposiciones ≥ 1	Exposiciones en la clase o asignatura. En el caso de no haber exposiciones en la clase o asignatura, se debe poner 0. Exposiciones en la clase o asignatura.	Exposiciones ≥ 1 Exposiciones ≥ 1 Exposiciones ≥ 1 Exposiciones ≥ 1
Matrículas	Matrículas ≥ 1	Matrículas en la clase o asignatura. En el caso de no haber matrículas en la clase o asignatura, se debe poner 0. Matrículas en la clase o asignatura.	Matrículas ≥ 1 Matrículas ≥ 1 Matrículas ≥ 1 Matrículas ≥ 1
Exposiciones	Exposiciones ≥ 1	Exposiciones en la clase o asignatura. En el caso de no haber exposiciones en la clase o asignatura, se debe poner 0. Exposiciones en la clase o asignatura.	Exposiciones ≥ 1 Exposiciones ≥ 1 Exposiciones ≥ 1 Exposiciones ≥ 1
Verificación	Verificación ≥ 1	Verificación en la clase o asignatura. En el caso de no haber verificación en la clase o asignatura, se debe poner 0. Verificación en la clase o asignatura.	Verificación ≥ 1 Verificación ≥ 1 Verificación ≥ 1 Verificación ≥ 1

Proponer P1:
Modela los 3
siguientes
diagramas de
clases

7. EJEMPLO 1: seguros

Se requiere diseñar un sistema para la gestión de pólizas de seguro. El sistema debe contemplar la información relativa a los empleados, que pueden ser vendedores o jefes de área, y los detalles asociados a las pólizas vendidas.

Para cada empleado , se debe registrar su número de identificación (un solo DNI) y su nombre. Para los vendedores se debe almacenar un único número identificativo de vendedor, que cuando se da de alta vale siempre 0, y la zona en la que operan, mientras que para los jefes se requiere mantener el registro de su salario.

Además, un único vendedor es responsable de la venta de cada póliza. Por lo tanto, se debe registrar información sobre las pólizas vendidas por cada vendedor.

En relación con las pólizas de seguro, se deben incluir datos como el número de póliza, el importe de esta y el nombre del beneficiario. También es necesario registrar la fecha en la que se efectuó la venta de cada póliza.

El sistema debe ser capaz de proporcionar funcionalidades que permitan mostrar el nombre del empleado, el salario del jefe, el número identificativo del vendedor y el nombre del beneficiario asociado a la póliza vendida.

7. EJEMPLO 2: hoteles

Se busca desarrollar un sistema para administrar hoteles, algunos de los cuales están vinculados a una cadena específica. Se desea recolectar información sobre las áreas metropolitanas, los hoteles, las cadenas de hoteles y los tipos de tarjetas de crédito aceptadas.

Para cada área metropolitana, es necesario conocer su nombre, así como la provincia a la que pertenece y el país al que está asociada.

Los hoteles, por otro lado, necesitan registrar detalles como su nombre, dirección, número de habitaciones, número de teléfono, clasificación por estrellas y los precios de sus habitaciones simples y dobles.

Algunos de estos hoteles están afiliados a cadenas específicas, de las cuales es importante mantener registros del nombre y del director.

Además, los hoteles aceptan diferentes tipos de tarjetas de crédito, aunque solo se necesita saber el nombre de cada tipo de tarjeta.

El sistema debe ser capaz de proporcionar funcionalidades para acceder a esta información detallada sobre las áreas metropolitanas, los hoteles, las cadenas de hoteles y los tipos de tarjetas de crédito aceptadas en los hoteles.

7. EJEMPLO 3: Empresa

Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes, que son personas. Las personas se caracterizan por su nombre y fecha de nacimiento. Se debe poder calcular su edad.

Los empleados tienen un sueldo bruto, algunos empleados son directivos y tienen una categoría. Los empleados tienen un solo directivo. Los directivos asignan tareas a los empleados subordinados. Los directivos deben saber a qué empleados pueden asignar tareas. Un directivo puede asignar una tarea a otro directivo.

Las tareas consisten exclusivamente en un texto con las instrucciones a realizar. Los empleados guardan un listado con sus tareas pendientes y las trasladan a otro listado de tareas completadas cuando las ejecutan.

De los clientes además se necesita conocer su teléfono de contacto y enviar una felicitación el día de su cumpleaños.

La aplicación necesita mostrar los datos de empleados y clientes.