UCA

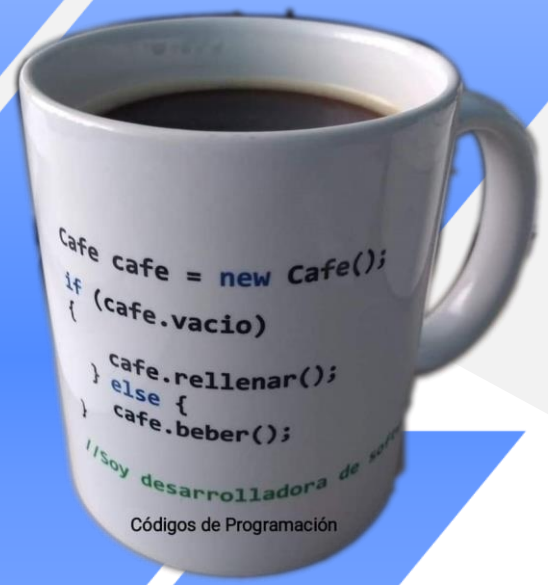
Programación Orientada a Objetos I

JAVA – CLASES (I)

Clases

- Una clase es un tipo de dato definido por el programador que contiene datos y funciones, llamados ***miembros*** de la clase.
- Los **datos miembros** definen el estado (atributos y valores) de cada objeto.
- Las **funciones miembros** definen el comportamiento de cada objeto (métodos).
- La definición de una clase consta de dos partes:
  - La primera es el encabezado y está formada por el nombre de la clase precedido por la palabra reservada *class* y opcionalmente la restricción de acceso (visibilidad).
  - La segunda parte es el cuerpo y está encerrado entre llaves *{}*.

```
[public] class NombreClase {
        // cuerpo: datos y métodos miembros
}
```



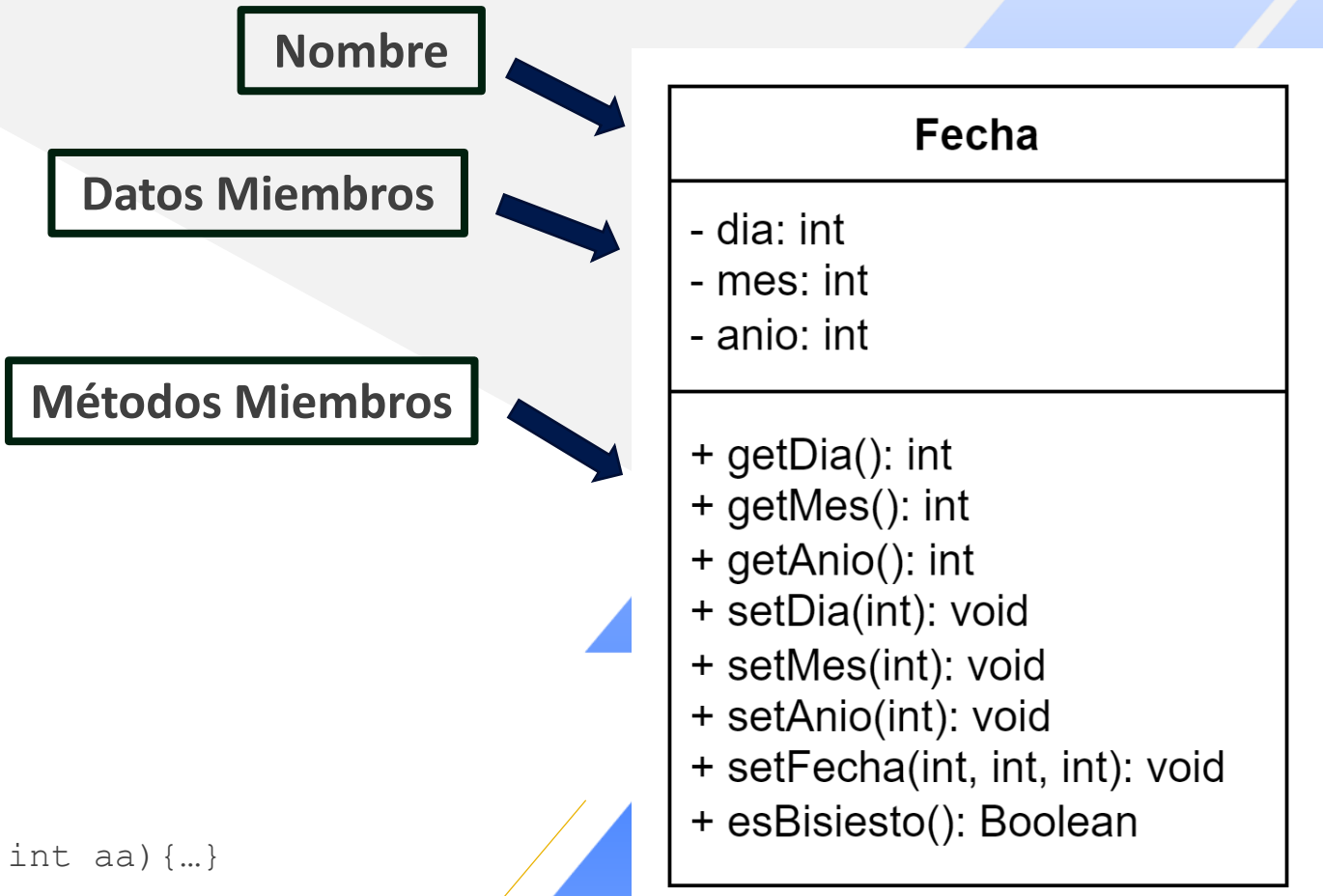
Clases

- La declaración de un **dato miembro** se realiza igual que el resto de las variables.
- La declaración de una **función miembro** se hace igual que las demás funciones.
- Un mismo identificador para un dato miembro puede ser utilizado para varias clases, pero no en la misma.
- Estos especificadores o restricciones de acceso forman parte del cuerpo de la misma y sirven para indicar el nivel de acceso a los datos y funciones miembros.
- Para restringir el acceso a los miembros, JAVA provee las palabras claves:
  - *public*: es accesible por cualquier clase, dentro o fuera de la misma.
  - *private*: puede ser utilizado por los métodos de su propia clase, pero no por métodos de otras clases ni por métodos de una **clase derivada** (clases hijas).
  - *protected*: se comporta igual que uno privado para las clases externas, pero como público para las clases derivadas.
  - *package*: se comporta igual que uno privado para las **clases externas al paquete**, pero actúa como un miembro público para los métodos de las clases del mismo paquete. Se trata de acceso por defecto en JAVA.
- Las definiciones de las funciones o métodos miembro de una clase se realizan dentro del cuerpo de la misma.

Clases

```
public class Fecha {
  private int dia;
  private mes;
  private anio;

  public int getDia(){…}
  public int getMes(){…}
  public int getAño(){…}
  public void setDia(int dd){…}
  public void setMes(int dd){…}
  public void setAnio(int dd){…}
  public void setFecha(int dd,int mm,int aa){…}
  public boolean esBisiesto(){…}
}
```



Inicialización de un objeto

- La declaración de un objeto se realiza de la misma manera que la de una variable.
- Cada objeto **debe** ser declarado
- Cada objeto **debe** ser instanciado mediante la palabra reservada ***new***

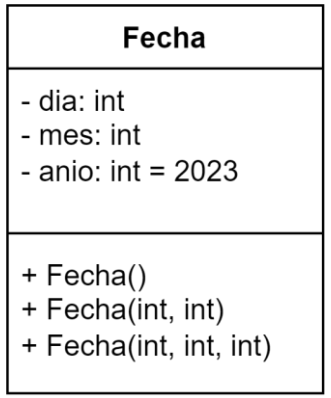
Ejemplo:

```
NombreClase nombreObjeto;
nombreObjeto = new NombreClase();
NombreClase nombreObj2 = new NombreClase();
Fecha unaFecha = new Fecha();
```

Constructor de un objeto

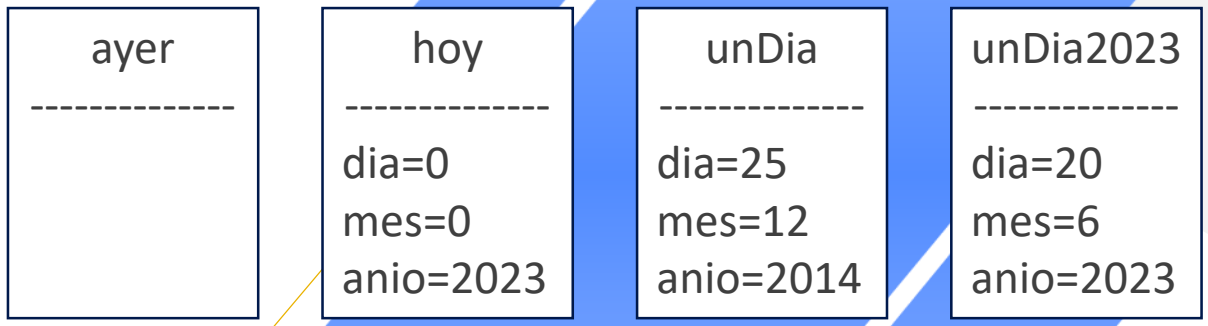
- Un constructor es una función miembro especial de la clase, que es llamada al instanciar un objeto.
- Su objetivo es crear e inicializar un objeto de forma tal que contengan valores válidos.
- Si no se declara un constructor, JAVA crea uno por default sin argumentos e inicializando todos sus datos miembros con valores por defecto acorde al tipo de dato.
- El constructor, secuencial y recursivamente, reserva memoria para cada uno de los datos miembros.
- Posee el mismo nombre que la clase a la que pertenece.
  Ej. el constructor de la clase Factura se denominará Factura.
- No retorna ningún valor y no puede ser declarado estático.
- Se pueden tener **constructores múltiples o alternativos** con **el mismo nombre y diferentes argumentos**, para inicializar el estado de un objeto de distintas formas (sobrecarga de funciones).

Constructor




```
public class Fecha {
   private int dia, mes, anio=2023;
   Fecha() {}
   Fecha(int pDia, int pMes) {dia=pDia; mes=pMes;}
   Fecha(int dd, int mm, int aa) {dia=dd; mes=mm; anio=aa;}
}
```

```
Fecha ayer;
Fecha hoy = new Fecha();
Fecha unDia = new Fecha(25,12,2014);
Fecha unDia2023 = new Fecha(20,6);
```



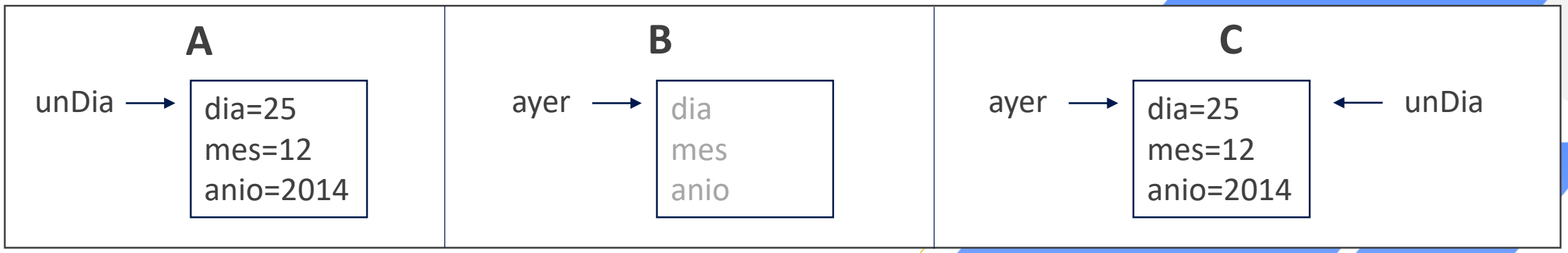
Asignación de objetos

- Otra forma de inicializar un objeto **ya declarado** es copiar un objeto existente.
- No se copia el contenido sino que ambas variables pasan a referenciar al mismo espacio de memoria.
- Eso significa que al modificar un atributo en un objeto se va a observar el cambio en el otro objeto.

  Ejemplo:

```
A) Fecha unDia = new Fecha(25,12,2014);
B) Fecha ayer;
C) ayer = unDia;
```

- El mismo efecto se puede obtener declarando e inicializando en la misma línea: `Fecha ayer = unDia;`



Antes de seguir…

- Cuantas horas le dedicaron a programar esta semana? Solo se puede aprender haciendo, equivocándose, encontrando el error, corregir , volver a equivocarse, volver a corregir, volver a hacer… y eso lleva tiempo

Con las horas de clase **<u>NO ALCANZA</u>**.

- El cálculo es que necesitan ***de 1 y 2 veces las horas de clase de tiempo de PRÁCTICA*** es decir 5 a 10 hs. aprox.

- Hay que tener afianzados los conocimientos de las materias anteriores, especialmente la lógica y la utilización de los condicionales, ciclos y bucles.

Herencia

- Mecanismo que permite definir **clases derivadas** a partir de otras clases ya existentes.
- La clase derivada o clase hija o subclase hereda todas las propiedades de la clase existente, que recibe el nombre de clase base o clase padre o **super clase**.
- Puede ser, a su vez, una clase base.
- Hereda **todos los miembros de la clase base.**
- Puede acceder a los miembros ***public*** y ***protected*** de la clase (o clases) bases como si fueran miembros de la misma.
- No tiene acceso a los miembros ***private*** de la clase base.
- Añade a los miembros heredados sus propios miembros.
- Los miembros heredados por una clase derivada pueden, a su vez, ser heredados por más clases derivadas de ella.

Herencia

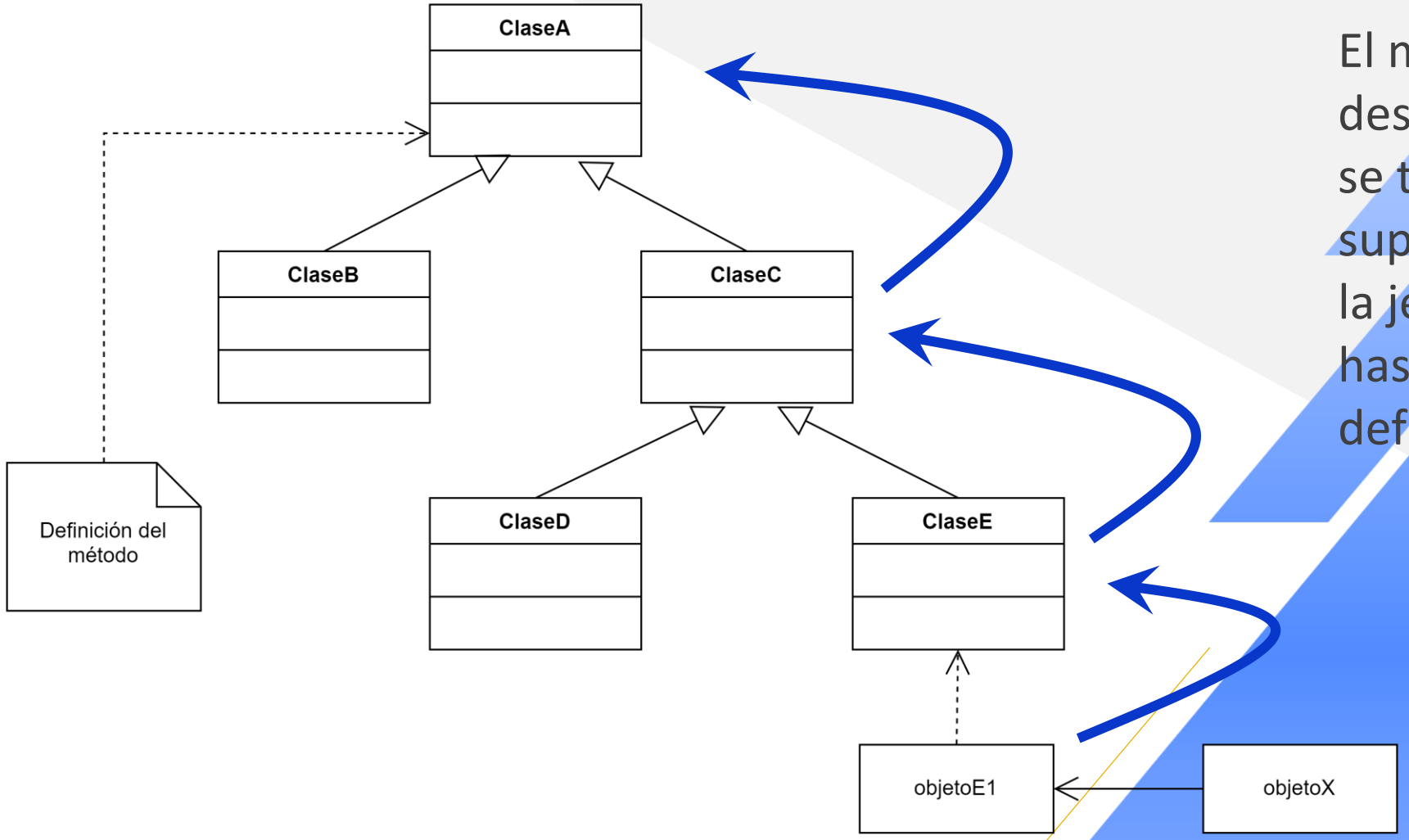
Se declara la herencia mediante la palabra reservada ***extends***

```
class NombreClaseDerivada extends NombreClaseBase {
    // cuerpo de la clase derivada
}
```

Ejemplo:

```
public class Derivada extends Base {
    // ...
}
```

Herencia: Funcionamiento

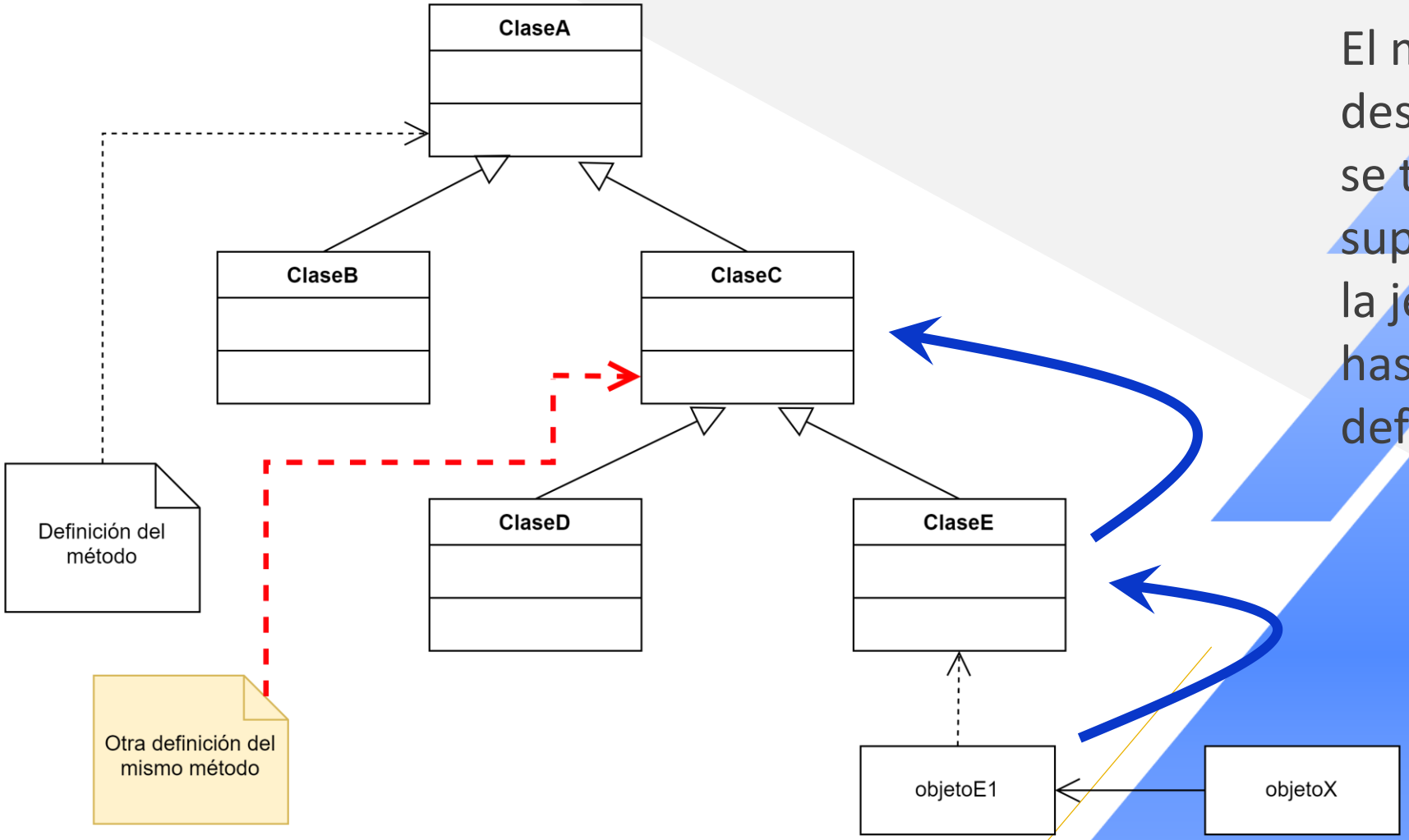


El mensaje se envía desde una subclase y se transmite a las super clases través de la jerarquía definida hasta encontrar una definición del método

Herencia: overriding

- La superposición de métodos (shadowing u overriding) es la forma de ***crear un método en la subclase*** que tenga ***la misma identificación de un método en una superclase***
- Cuando hablamos de la misma identificación significa que tienen el mismo nombre, número y tipo de argumentos (como la sobrecarga de funciones).
- Este nuevo método ***oculta*** el método de la superclase.

Herencia: Funcionamiento



El mensaje se envía desde una subclase y se transmite a las super clases través de la jerarquía definida hasta encontrar una definición del método

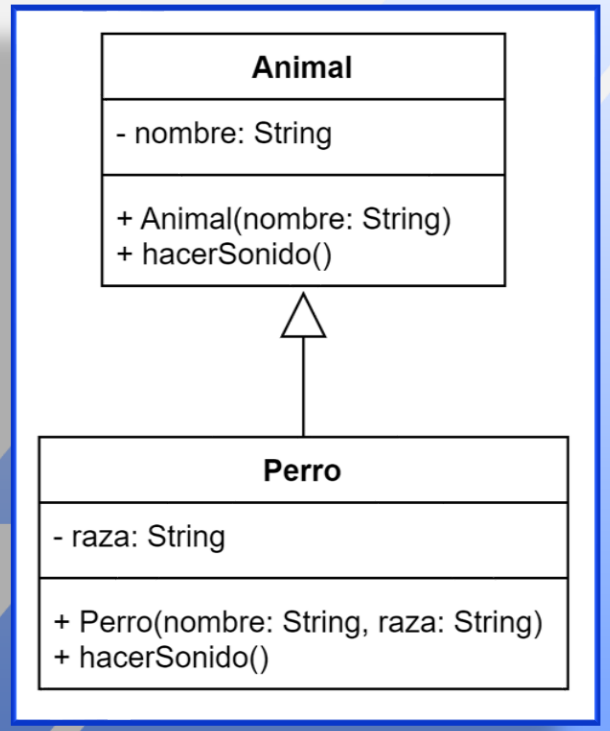
Herencia

- Los miembros declarados *private* en una clase sólo pueden ser accedidos por las **funciones miembro de la clase**.
- Los miembros declarados *protected* pueden **además** ser accedidos por las **funciones miembro de una clase derivada**.
- Los miembros *public* pueden ser accedidos por **cualquier función**.
- Los constructores de la clase base **no son heredados** por sus clases derivadas, como tampoco pueden ser superpuestos.
- Al crear un objeto, primero se llama al constructor de la clase base y luego, al de la clase derivada.

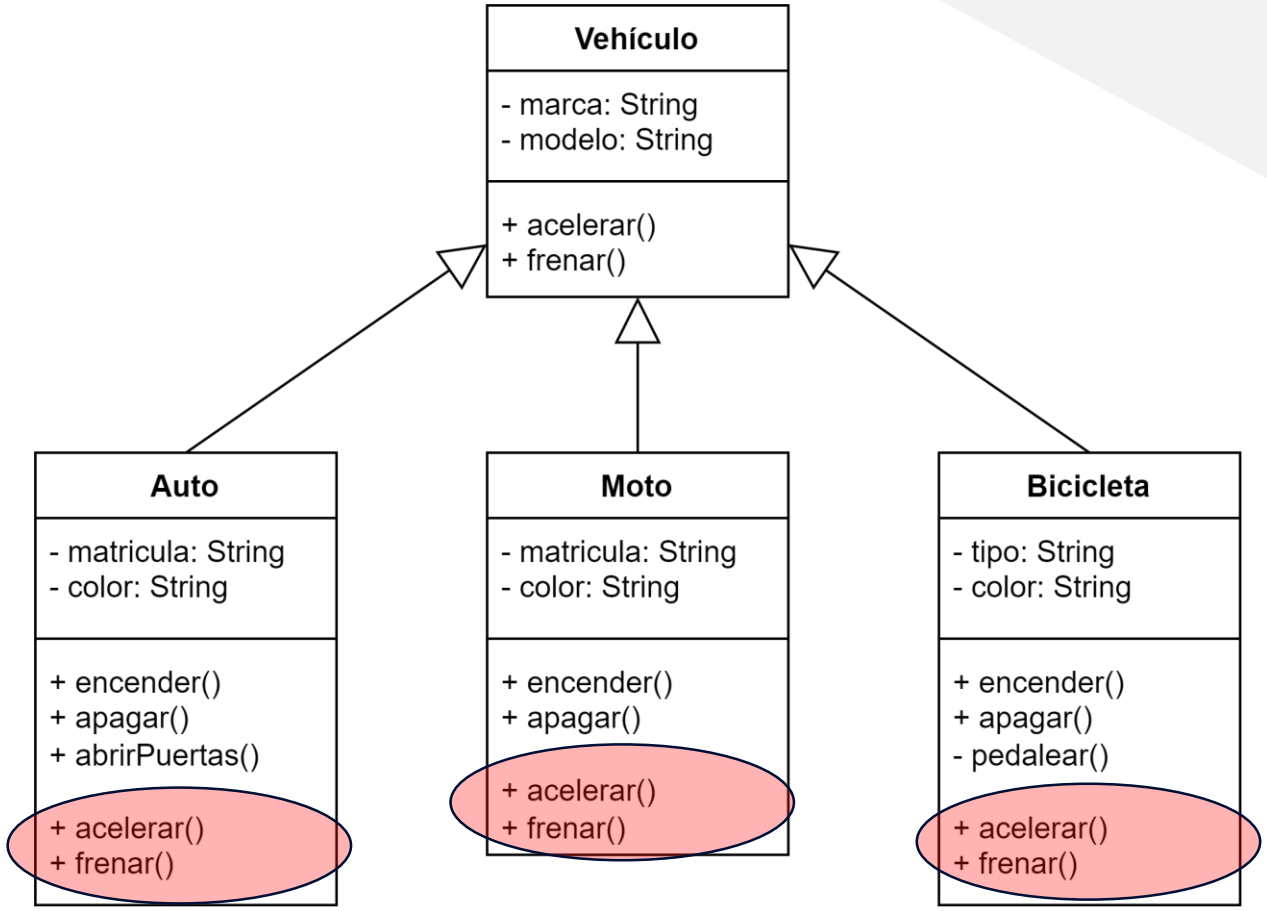
Herencia: *this y super*

- La palabra clave *this* se utiliza para **hacer referencia a los miembros de la instancia actual** de una clase. Ayuda a diferenciar los atributos de la instancia de los parámetros y variables locales con el mismo nombre.
- Si se necesita llamar al **método definido en una clase base**, desde dentro de una clase que ha reemplazado ese método, se puede hacer anteponiendo la palabra clave *super*.
- Como los constructores no posee un nombre específico, para hacer referencia a algún constructor de la clase padre, se debe utilizar la palabra clave *super* como reemplazo del constructor

```
class Animal {
    String nombre;
    Animal(String nombre) {
        this.nombre = nombre;}      // this evita ambigüedad
    void hacerSonido() {
        System.out.println("El animal hace un sonido.");}
}
class Perro extends Animal {
    String raza;
    Perro(String nombre, String raza) {
        super(nombre);        // Llama al constructor de Animal
        this.raza = raza;}
    @Override
    void hacerSonido() {
        super.hacerSonido(); // Llama al método hacerSonido de la superclase Animal
        System.out.println("El perro ladra.");}
}
public class EjemploSuper {
    public static void main(String[] args) {
        Perro miPerro = new Perro("Max", "Labrador");
        miPerro.hacerSonido();}
}
```



Para practicar



Hacer el código Java del siguiente diagrama de clases, teniendo en cuenta:

- Cada método solo debe imprimir por pantalla el nombre de la clase y la acción realizada.
  Ejemplo: el método acelerar() de Moto debe mostrar: "Moto acelera"
- Cada tipo de vehículo acelera o frena de diferente manera, por eso los métodos en círculos rojos deben implementarse en cada clase.