Grado en Ingeniería Información

# PROGRAMACIÓN II - Sesión 10

## Tema 6.

## Polimorfismo

Curso 2022-2023

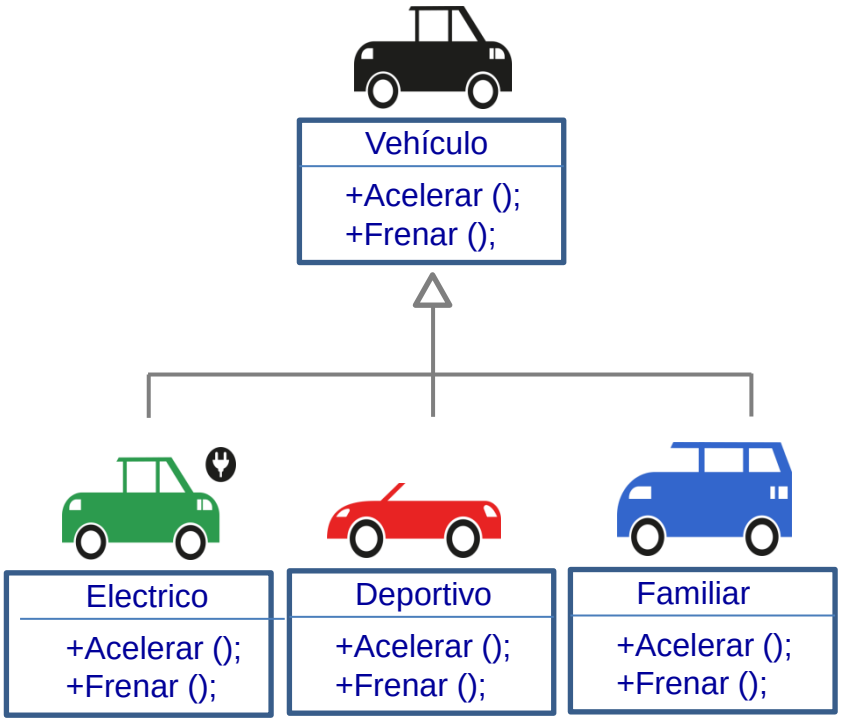
Marta N. Gómez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

# T6. Polimorfismo






# ÍNDICE

UNIVERSIDAD NEBRIJA

# T6. Polimorfismo

## 6.1. Definición

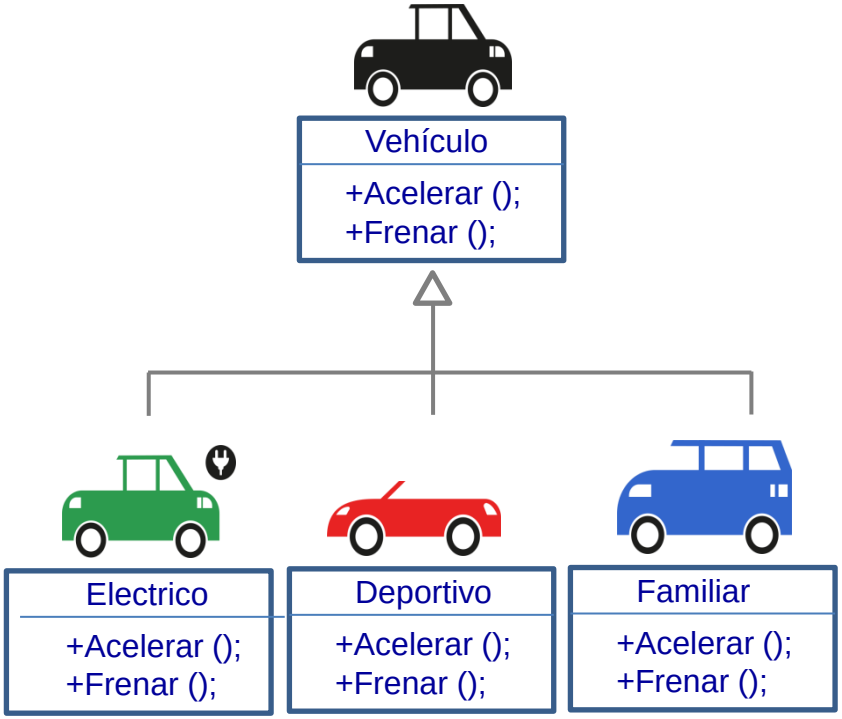
6.2. Tipos de Ligadura

6.3. Métodos Virtuales

6.4. Destructores Virtuales

6.5. Clases Abstractas

6.6. Conversión entre Objetos




# ÍNDICE

## POLIMORFISMO

Mecanismo que permite que **un objeto** pueda **interpretar de diferentes maneras un mismo tipo de mensaje** y **efectuar diferentes operaciones** en cada caso.

El **objeto que recibe** el mensaje es el **responsable de invocar el método adecuado** en función de algún **criterio adicional** que permita **distinguir entre las posibles interpretaciones**.

## TIPOS DE POLIMORFISMO

- **Paramétrico** (sobrecarga de funciones o sobrecarga de operadores)

  Este tipo de polimorfismo existe entre los **métodos de una clase**. El criterio adicional son los parámetros de dichos métodos. Se definen ***métodos distintos*** con el ***mismo nombre, pero con parámetros diferentes***.

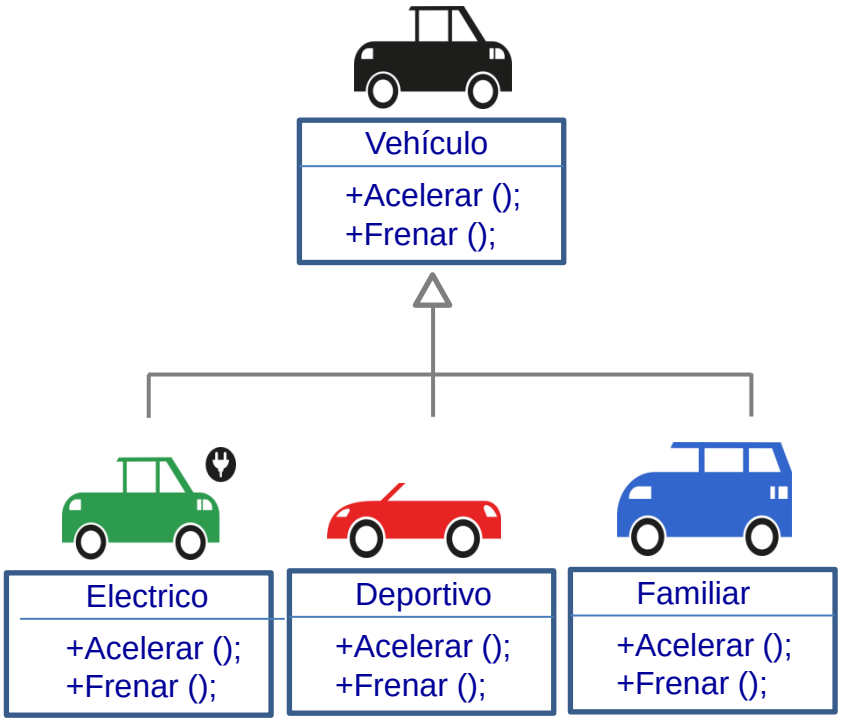
- **De subclases**

  Este tipo de polimorfismo permite que **métodos definidos en una clase** sean ***redefinidos*** posteriormente en **una subclase** de la misma. Los **métodos** son **sintácticamente iguales** pero pertenecen a **diferentes clases**.

UNIVERSIDAD NEBRIJA

# T6. Polimorfismo






# ÍNDICE

## Ligadura o *Binding*

Es la **conexión** entre la **llamada a un método** o **función** con su **implementación**.

La vinculación o ligadura es **crítica** cuando se está utilizando **polimorfismo** porque se pueden obtener **resultados distintos** en función de **la implementación elegida según la llamada**.

# Ligadura o *Binding*

Tipos de ligadura o vinculación:

- **Ligadura temprana o estática** (early binding)

  La vinculación entre la llamada y la implementación se realiza en **tiempo de compilación**, antes de ejecutar el programa.

- **Ligadura en tiempo de ejecución o dinámica** (late binding)

  La vinculación entre la llamada y la implementación se realiza en **tiempo de ejecución**, utilizando como **criterio** el **tipo de objeto**.

  Esta ligadura y el uso de **funciones o métodos virtuales** hacen posible el ***polimorfismo de subclases***.

```
//---------------CLASE PERSONA---------------------
class Persona
{    private:
         string nombre, apellidos;
     public:
         Persona (const string &nom, const string &apell):
             nombre(nom), apellidos(apell) {}
         void mostrar() const;
         void presentar() const;
};

//---------------CLASE EMPLEADO---------------------

class Empleado : public Persona
{    private:
         int salario;
     public:
         Empleado (const string &nom, const string &ape, int s):
                   Persona(nom, ape), salario(s) {}
         void mostrar() const;
         void datos() const;
};
```

```
//---------------CLASE PERSONA----------------------

void Persona::presentar() const  {
    cout << "\n\n\t\tHola te presento a ";
    mostrar ();     // Se ejecuta el método de la clase Persona
}

void Persona::mostrar() const {
    cout << nombre << "  " << apellidos << endl << endl << endl;
}
```

```
//----------------CLASE EMPLEADO----------------------

void Empleado::mostrar() const {
    cout << "\n\n\t\tEl salario de ";

    Persona::mostrar();

    cout << "\t\tes de " << salario << " euros al mes.\n\n\n";
}

void Empleado::datos() const {
     cout << "\n\t\tQue tiene un salario de " <<
             salario << " euros al mes.\n\n\n";
}
```

# Tipos de Ligadura

```
int main ()
{
    Empleado trab1 ("EVA","SERRANO", 1700);
    Persona pers1 ("MANUEL", "NAVARRO");


    cout << "\n\n\tSe llama con un objeto de la clase EMPLEADO\n\n";
    trab1.presentar ();  // Se ejecuta el método de la clase Persona
    trab1.datos();       // Se ejecuta el método de la clase Empleado
    cout << endl;

    cout << "\n\n\tSe llama con un objeto de la clase PERSONA\n\n";
    pers1.presentar();   // Se ejecuta el método de la clase Persona

    cout << "\n\n\tSe llama con un objeto de la clase EMPLEADO\n\n";
    trab1.mostrar();     // Se ejecuta el método de la clase Empleado

    cout << "\n\n\t";
    return 0;
}
```

# Tipos de Ligadura

```
int main ()
{
    Empleado trab1 ("EVA","SERRANO", 1700);
    Persona pers1 ("MANUEL", "NAVARRO");


    cout << "\n\n\tSe llama con un objeto de la clase EMPLEADO\n\n";
    trab1.presentar ();  // Se ejecuta el método de la clase Persona
    trab1.datos();       // Se ejecuta el método de la clase Empleado
    cout << endl;

    cout                                                          ";
    pers1                                                         ona

    cout                                                          n";
    trab1                                                         eado

    cout
    retur
}
```

```
Se llama con un objeto de la clase EMPLEADO


        Hola te presento a EVA  SERRANO


        Que tiene un salario de 1700 euros al mes.
```

# Tipos de Ligadura

```
int main ()
{
    Empleado trab1 ("EVA","SERRANO", 1700);
    Persona pers1 ("MANUEL", "NAVARRO");


    cout << "\n\n\tSe llama con un objeto de la clase EMPLEADO\n\n";
    trab1.presentar ();  // Se ejecuta el método de la clase Persona
    trab1.datos();       // Se ejecuta el método de la clase Empleado
    cout << endl;

    cout << "\n\n\tSe llama con un objeto de la clase PERSONA\n\n";
    pers1.presentar();   // Se ejecuta el método de la clase Persona

    Se llama con un objeto de la clase PERSONA

          Hola te presento a MANUEL  NAVARRO

}
```

se EMPLEADO\n\n";
la clase Empleado

# Tipos de Ligadura

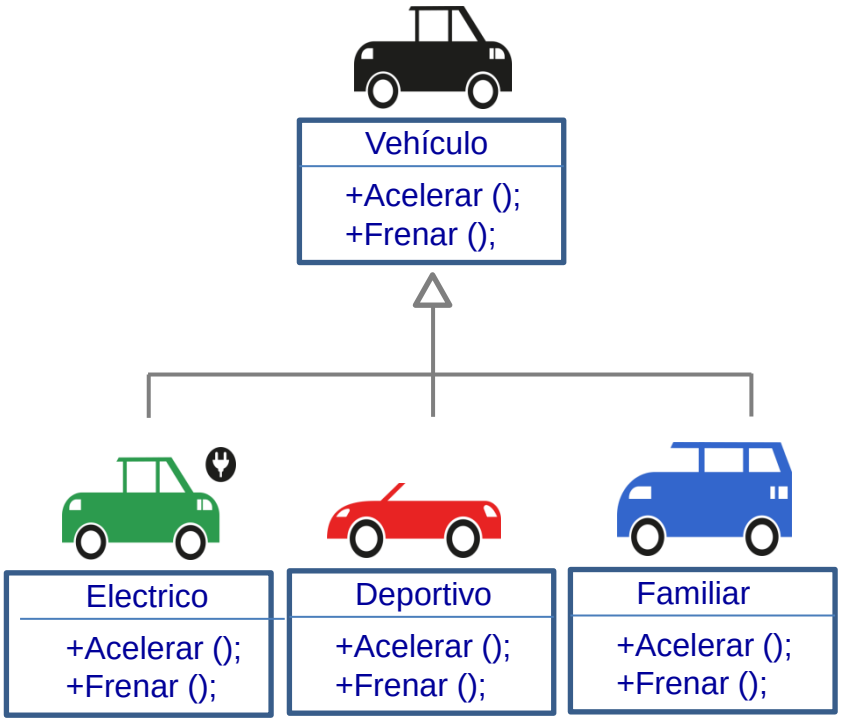
```
int main ()
{
    Empleado trab1 ("EVA","SERRANO", 1700);
    Persona pers1 ("MANUEL", "NAVARRO");

    cout                                                    EMPLEADO\n\n";
    trab                                                    a clase Persona
    trab                                                    a clase Empleado
    cout

    cout                                                    PERSONA\n\n";
    pers                                                    a clase Persona

    cout << "\n\n\tSe llama con un objeto de la clase EMPLEADO\n\n";
    trab1.mostrar();     // Se ejecuta el método de la clase Empleado

    cout << "\n\n\t";
    return 0;
}
```

```
Se llama con un objeto de la clase EMPLEADO

        El salario de EVA  SERRANO

        es de 1700 euros al mes.
```

UNIVERSIDAD NEBRIJA

# T6. Polimorfismo






# ÍNDICE

## MÉTODOS o FUNCIONES VIRTUALES:

Funciones incluidas en **varios niveles de una jerarquía de clases** con el ***mismo nombre*** pero con ***distinta definición***.

Permiten **vincular** adecuadamente la **llamada al método** que recibe el mensaje con **su implementación, pero en tiempo de ejecución**.

## MÉTODOS o FUNCIONES VIRTUALES:

Los **métodos virtuales** **siempre** se definen en las **clases Base**.

**Sintaxis:**

Incluir la palabra reservada ***virtual*** delante del prototipo del método.

***virtual <tipo> <nombre_funcion> (<lista_parametros>)[{}];***

```
class Persona {
    private:
        string nif;
        int edad;
        string nombre, apellidos;
    public:
        Persona():nif(""), edad(0), nombre(""), apellidos("") {}
        Persona(string const &identif, int aa, string const &nom,
                string const &apel):
                nif(identif), edad(aa), nombre(nom), apellidos(apel) {}

        void mostrar() const;
        void okMatricula() const;
};

class Estudiante : public Persona {
    private:
        int curso;
    public:
        Estudiante (string const &id, int a, string const &nom,
                    string const &ape, int cur):
            Persona (id, a, nom, ape), curso(cur) {};

        void mostrar() const;
};
```

# Métodos Virtuales

```
//---------------CLASE PERSONA---------------------

void Persona::mostrar() const {
    cout << nombre << " " << apellidos << endl;
    cout << "\n\tNIF: " << nif << " EDAD: " << edad;
}

void Persona::okMatricula() const {
     cout << "\n\n\tEl estudiante  ";
     mostrar();   // Ejecuta el método de la clase Persona
     cout << "\n\tHa sido matriculado en el curso correctamente.\n\n";
}
```

# Métodos Virtuales

```
//---------------CLASE ESTUDIANTE---------------------

void Estudiante::mostrar() const {
    Persona::mostrar();

    cout << "\n\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

```
#include "Persona.h"

using namespace std;

int main ()
{
    Estudiante estud ("123456789S", 20, "Eva","Sanz", 3);

    estud.okMatricula ();  // Se ejecuta el método de la clase Persona

    cout << "\n\n\t";
    return 0;
}
```

# Métodos Virtuales

```
//---------------CLASE ESTUDIANTE---------------------

void Estudiante::mostrar() const {
    Persona::mostrar();

    cout << "\n\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

```
El estudiante  Eva Sanz

NIF: 123456789S EDAD: 20
Ha sido matriculado en el curso correctamente.
```

```
#include "Persona.h"

using namespace std;

int main ()
{
    Estudiante estud ("123456789S", 20, "Eva","Sanz", 3);

    estud.okMatricula ();  // Se ejecuta el método de la clase Persona

    cout << "\n\n\t";
    return 0;
}
```

```
class Persona {
    private:
        string nif;
        int edad;
        string nombre, apellidos;
    public:
        Persona():nif(""), edad(0), nombre(""), apellidos("") {}
        Persona(string const &identif, int aa, string const &nom,
                string const &apel):
                nif(identif), edad(aa), nombre(nom), apellidos(apel) {}
        virtual ~Persona() {}
        virtual void mostrar() const;
        void okMatricula() const;
};

class Estudiante : public Persona {
    private:
        int curso;
    public:
        Estudiante (string const &id, int a, string const &nom,
                    string const &ape, int cur):
            Persona (id, a, nom, ape), curso(cur) {};

        void mostrar() const;
};
```

# Métodos Virtuales

```
//---------------CLASE PERSONA---------------------

void Persona::mostrar() const {
    cout << nombre << " " << apellidos << endl;
    cout << "\n\tNIF: " << nif << " EDAD: " << edad;
}

void Persona::okMatricula() const {
     cout << "\n\n\tEl estudiante  ";
    mostrar();   // Ejecuta el método de la clase que lo llama
    cout << "\n\tHa sido matriculado en el curso correctamente.\n\n";
}
```

# Métodos Virtuales

```
//---------------CLASE ESTUDIANTE---------------------

void Estudiante::mostrar() const {
    Persona::mostrar();

    cout << "\n\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

```
#include "Persona.h"

using namespace std;

int main ()
{
    Estudiante estud ("123456789S", 20, "Eva","Sanz", 3);

    estud.okMatricula ();  // Se ejecuta el método de la clase Persona

    cout << "\n\n\t";
    return 0;
}
```

# Métodos Virtuales

```
//---------------CLASE ESTUDIANTE---------------------

void Estudiante::mostrar() const {
    Persona::mostrar();

    cout << "\n\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

```
#include "Persona.h"

using namespace std;

int main ()
{
    Estudiante estud ("1234

    estud.okMatricula ();

    cout << "\n\n\t";
    return 0;
}
```

```
El estudiante  Eva Sanz

NIF: 123456789S EDAD: 20

Esta matriculado en el 3 curso.


Ha sido matriculado en el curso correctamente.
```

Los **métodos virtuales heredados** son también **virtuales en la clase derivada**.

Los únicos **métodos de una clase que no pueden ser virtuales** son:

- **Constructores**
- **Métodos estáticos**
- **Operadores *new* y *delete***

La declaración de un **método como virtual** indica al compilador que **la vinculación entre el mensaje y el método** será en **tiempo de ejecución** (***ligadura dinámica***) basándose en el **tipo o la clase del objeto que realiza la llamada**.

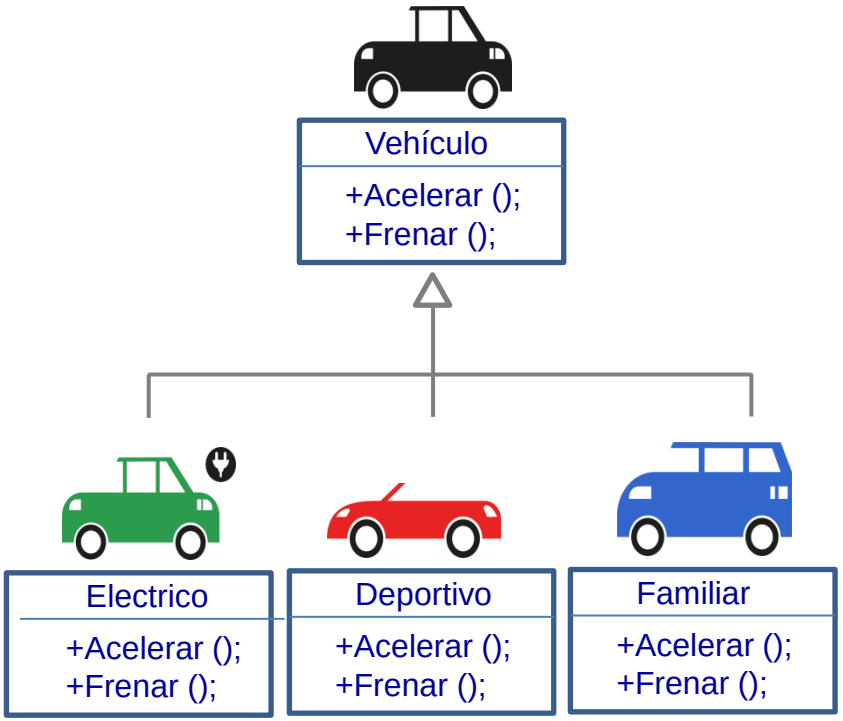
## Motivos para NO DECLARAR los métodos como virtuales

- **Comportamiento necesario**. Se quiere llamar con un **mensaje** a un objeto para que se **ejecute el método de la clase base** y **no al método de la clase derivada** a la que pertenece el objeto.
- **Rendimiento**. La **vinculación estática** se hace cuando se **compila**, sin embargo, la **vinculación dinámica** tiene que buscar el método adecuado a ejecutar durante la **ejecución del programa**.

UNIVERSIDAD NEBRIJA

# T6. Polimorfismo






# ÍNDICE

**REGLA IMPORTANTE:**

**Si en una clase existen métodos o funciones virtuales, si se crea un destructor en ella éste tiene que ser virtual.**