Polimorfismo

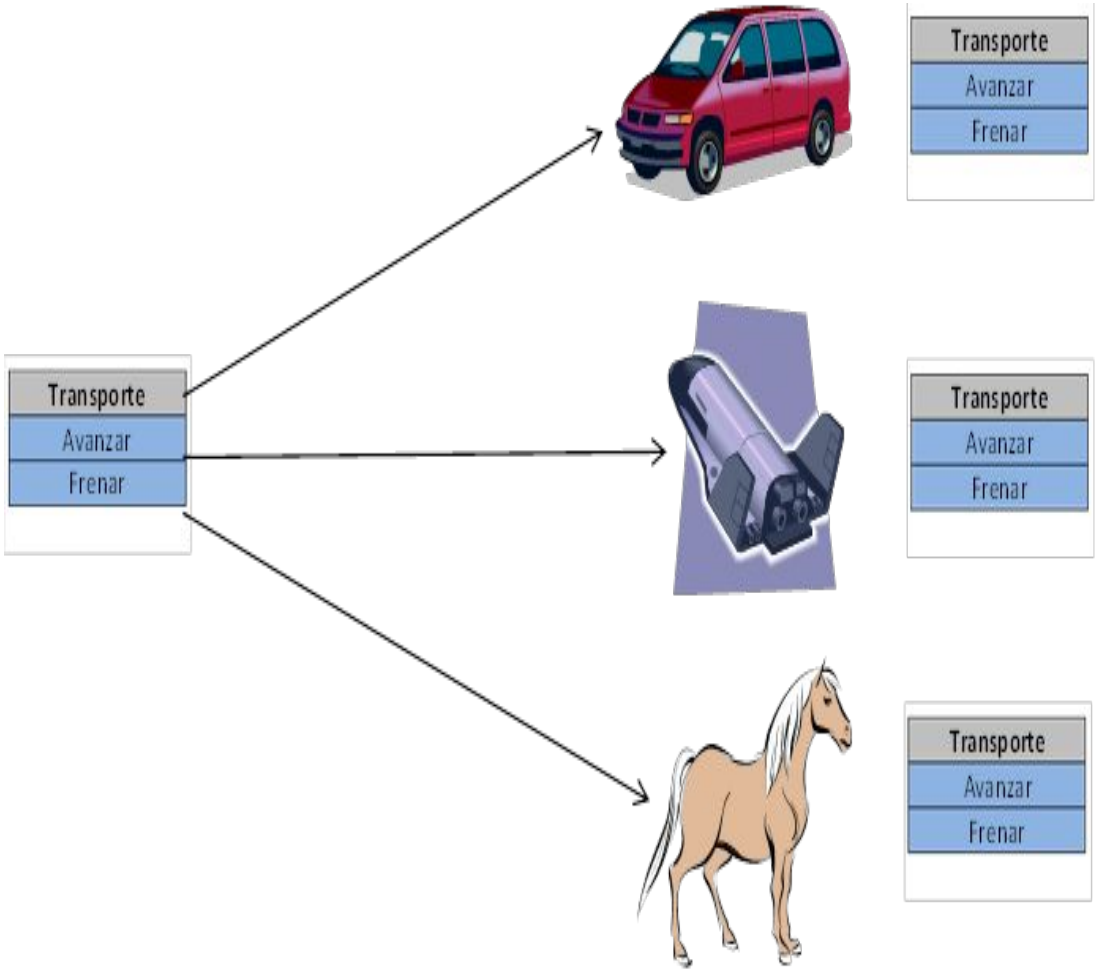
Polimorfismo

Veremos que el polimorfismo y la herencia son dos conceptos estrechamente ligados.

Conseguimos implementar el polimorfismo en jerarquías de clasificación que se dan a través de la herencia.

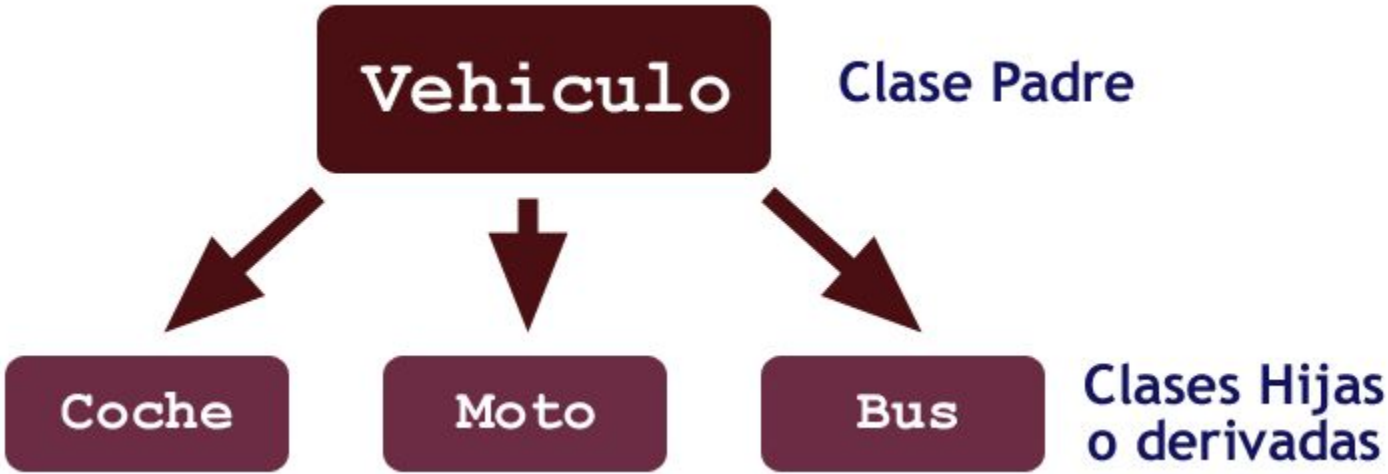
Polimorfismo

En el ámbito de la Programación Orientada a Objetos se entiende por polimorfismo la capacidad de llamar a funciones distintas con un mismo nombre. Estas funciones pueden actuar sobre objetos distintos dentro de una jerarquía de clases, sin tener que especificar el tipo exacto de los objetos



Polimorfismo

Por ejemplo, tenemos una clase vehículo y de ella dependen varias clases hijas como coche, moto, autobús, etc.



El polimorfismo es la capacidad de manejar distintas clases heredadas de una clase base de la misma forma. Este ejemplo simplificado muestra un posible resultado a implementar polimorfismo:

Polimorfismo

Una clase puede exhibir comportamientos (métodos) que no tiene definidos en sí mismo sino en otras clases que se derivan de ella. Es decir, una misma clase puede adoptar "muchas formas" diferentes dependiendo de las redefiniciones presentes en clases que se derivan de ella.

Polimorfismo estático (en tiempo de compilación):

Esto ocurre cuando se da un proceso conocido como "early binding" o "static binding" que no es otra cosa distinta a que la memoria que se reserva para un polimorfismo es reservada en tiempo de compilación.

Esta función, por pertenecer a la interfaz pública de esta clase base es heredada por las clases derivadas de la misma.

Polimorfismo dinámico (en tiempo de ejecución):

Se le conoce como dinámico porque sucede en tiempo de ejecución de la aplicación, es decir la memoria que se reserva para este **polimorfismo** es **reservada en el momento que la aplicación lo requiere (proceso conocido como "late binding" o "dynamic binding").** Esta situación se da mediante el uso de **métodos virtuales de clases base** que son redefinidos en clases derivadas. Es estrictamente necesario el uso de la palabra reservada `virtual` ya que este polimorfismo depende de un proceso en el cual un puntero a funciones consulta una tabla de funciones virtuales para resolver si un método que se invoca es el que está definido en la clase base o si por el contrario corresponde a una redefinición en una de las clases derivadas.

Funciones Virtuales union Dinamica

Una función virtual es una función en una clase base que se declara usando la palabra reservada virtual Si existe, en alguna clase derivada otra definición para dicha función declarada como virtual en la clase base, le indica al compilador que no debe usar enlace estático para la misma, se determinará en tiempo de ejecución qué versión se invoca. A menudo a las funciones virtuales se les llama métodos

Sólo es obligatorio hacerlo en la clase en la que se declara por 1º vez Simplemente se recomienda hacerlo para facilitar la lectura.

La definición de las clases derivadas e informar cuáles funciones son virtuales y por tanto serán seleccionadas dinámicamente en tiempo de ejecución

Ejemplo

- Nombre
- Número de Cuenta
- DNI
- Fondo

Ejemplo

Ejemplo :https://replit.com/join/yycimgjldq-maximo-arielari

```
class Shape{
protected:  string description{""};
public:
    Shape() = default;
    Shape(string msj);
    virtual void draw();};
Shape::Shape(string msj){
  description=msj;}
void Shape::draw()  {
  cout << "Dibujando " << description << "\n"; }
```

```
class Circle : public Shape{
protected:  double radius{};
public:
    Circle() = default;
    Circle(string msj, double rad);
    virtual void draw(); };
Circle::Circle(string msj, double rad):Shape( msj)
{ radius=rad; }
void Circle::draw() {
      cout << "Dibujando " << description
            << " con radio " << radius << "\n";}
```

Ejemplo

```
class Shape{
public:
    Shape() = default;
    Shape(std::string description)
        : description(description){};
    // Función virtual a implementar en clase derivada
    virtual void draw() const{
        std::cout << "Dibujando " << description << "\n"; }
protected: std::string description{""}; };
```

Ejemplo

```
class Circle : public Shape{
public:
    Circle() = default;
    Circle(std::string description, double radius)
        : Shape(description), radius(radius){};
    // Implementación de la función virtual
    virtual void draw() const{
        std::cout << "Dibujando " << description
                  << " con radio " << radius << "\n";}
protected:   double radius{};};
```

Bibliografia

Programación en C++ autor: Luis Joyanes Aguilar.