Programación Orientada a Objetos

Parcial 3

Notas de curso

Dr. Ezequiel Arceo May

2 de mayo de 2019

Índice

«Un lenguaje de programación es una forma de expresarnos»
— Ezequiel Arceo —

1. Concepto básicos

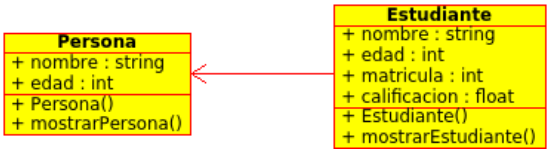
Los conceptos más básicos que debemos entender en la Programación Orientada a Objetos (POO) son:

- Clase
- Objeto
- Abstracción
- Encapsulación
- Herencia
- Polimorfismo

2. Herencia

La **herencia** sucede cuando una clase nueva se crea a partir de una clase existente, obteniendo (heredando) todos sus atributos y métodos.

A la clase de la cual se hereda se le llama *clase padre* o *súper clase*. A la clase que recibe la herencia se le llama *clase hija* o *subclase*.



Con la herencia nos ahorramos volver a definir los atributos y métodos de una clase. Basta con heredarlos de la clase padre y ya.

2.1. Herencia en C++ usando ‘: public‘

Para aprender a usar herencia en C++ seguiremos un pequeño ejemplo.

Primero **declaramos y definimos a la clase padre**, en esta caso será la clase **Persona**, con los atributos **nombre** y **edad**, y los métodos **Persona** y **mostrarPersona**:

```cpp
#include <iostream>
#include <string>
using namespace std;

class Persona{
  private:
    string nombre;
    int edad;
  public:
    Persona(string, int);
    void mostrarPersona();
};
Persona::Persona(string _n, int _e){
  nombre = _n;
  edad = _e;
}
void Persona::mostrarPersona(){
  cout << "Nombre: " << nombre << endl;
  cout << "Edad: " << edad << endl;
}
```

Seguidamente, **declaramos una clase hija que herede de la clase padre**, en este caso la clase **Estudiante**, con los atributos **nombre**, **edad**, **matricula** y **calificacion**, y los métodos **Estudiante** y **mostrarEstudiante**.

Especificamos la herencia escribiendo `:public Persona` entre el nombre de la clase **Estudiante** y el corchete de apertura de su declaración. Este fragmento de código indica que la clase **Estudiante** puede acceder a todo público de la clase **Persona**, sin necesidad de anteponer `Persona::`

```
class Estudiante: public Persona {
  private:
    int matricula;
    float calificacion;
  public:
    Estudiante(string,int,int,float);
    void mostrarEstudiante();
};
```

Ahora es el momento de usar la herencia, accediendo a los miembros de la clase padre desde la clase hija.

El constructor de la clase **Estudiante** tiene 4 atributos, dos de ellos establecidos por el constructor de la clase **Persona**, y solamente tenemos que establecer los atributos específicos de la clase **Estudiante**:

```
Estudiante::Estudiante(string _n, int _e, int _m, float _c): Persona(_n, _e){
  matricula = _m;
  calificacion = _c;
}
```

En el método **mostrarPersona** solamente tenemos que mostrar manualmente los atributos **matricula** y **calificacion**, y delegamos mostrar los demás a la clase padre:

```
void Estudiante::mostrarEstudiante(){
  mostrarPersona(); // uso método de la clase padre
  cout << "Matricula: " << matricula << endl;
  cout << "Calificacion: " << calificacion << endl;
}
```

Listo!, ahora ya podemos usar a la clase **Estudiante**

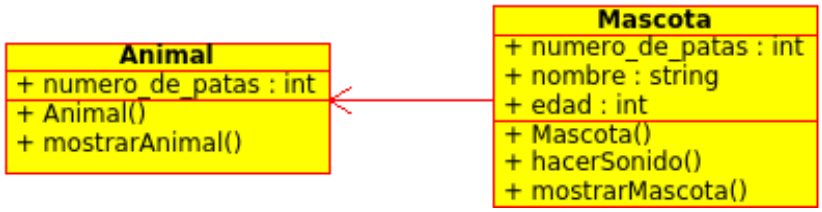
```cpp
int main(){
  Estudiante e1("Ezequiel Arceo May",34,123456,8.5);
  e1.mostrarEstudiante();
  system("pause");
  return 0;
}
```

Actividad 1.

Aplica la herencia usando como clase padre a la clase **Persona** y como clase hija a la clase **Fanatico** con atributos **gustoPor** (de qué ámbito soy fanático) y **preferido** (cuál es mi entidad preferida), y los métodos **Fanatico** y **mostrarFanatico**.

Trabajo B1. Entrega Antes de: Jueves 2 de Mayo, a las 23:59 horas.

Implementa el siguiente diagrama de clases usando herencia



- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `POO_TrabajoB1_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `POO_TrabajoB1_NOMBRE_DEL_AUTOR.*`

Ejemplo:
ASUNTO: `POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ`
ADJUNTO: `POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ.cpp`

3. Polimorfismo

El **polimorfismo** es la cualidad que tiene los objetos de responder de distintas formas al mismo mensaje.

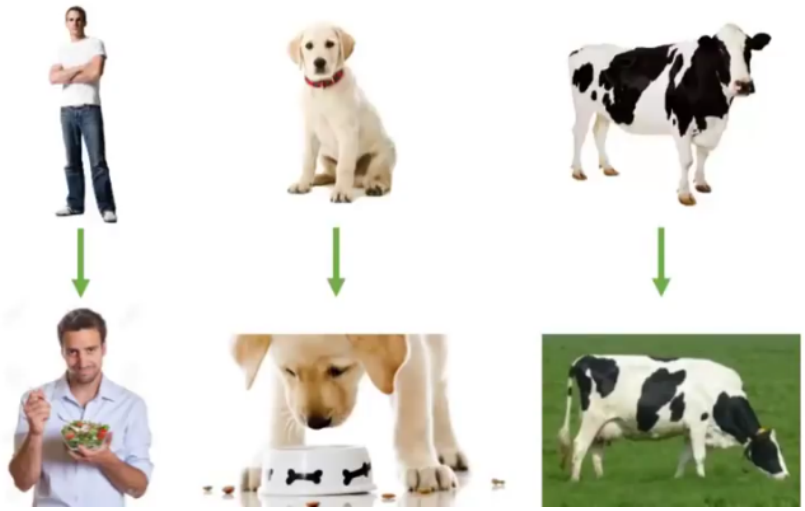


Figura 1: Considere objetos de las clases **Persona**, **Perro** y **Vaca**, cada uno de ellos puede **comer**, pero lo hacen de forma distinta.

3.1. Polimorfismo en C++ usando 'virtual'

En al sección anterior vimos que todas las clases implementadas tenían un método llamado **mostrarNombreDeLaClase**. Si quisiéramos indicar a cada elemento de una clase que simplemente se muestre, usando su método **mostrar** estaríamos ante la necesidad de aplicar polimorfismo.

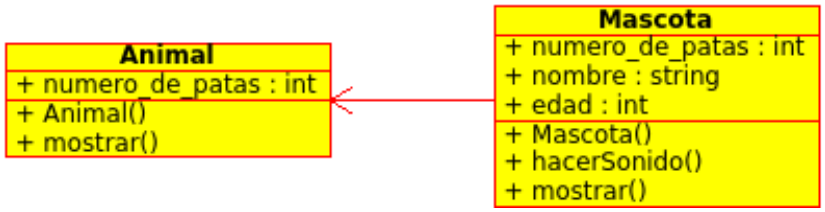
```cpp
#include <iostream>
#include <string>
using namespace std;

class Persona{
  private:
    string nombre;
    int edad;
  public:
    Persona(string, int);
    virtual void mostrar(); // Polimorfismo
};
Persona::Persona(string _n, int _e){
  nombre = _n;
  edad = _e;
}
void Persona::mostrar(){
  cout << "Nombre: " << nombre << endl;
  cout << "Edad: " << edad << endl;
}
class Estudiante: public Persona {
  private:
    int matricula;
    float calificacion;
  public:
    Estudiante(string,int,int,float);
    void mostrar(); // Polimorfismo
};
Estudiante::Estudiante(string _n, int _e, int _m, float _c): Persona(_n, _e){
  matricula = _m;
  calificacion = _c;
}
void Estudiante::mostrar(){
  Persona::mostrar(); // Polimorfismo
  cout << "Matricula: " << matricula << endl;
  cout << "Calificacion: " << calificacion << endl;
}
int main(){
  Persona persona("Chabelo",100);
  persona.mostrar();
  Estudiante alumno("Ezequiel Arceo May",34,123456,8.5);
  alumno.mostrar();
  return 0;
}
```

Como hemos visto en el ejemplo anterior (el mismo que en la sección previa), para usar polimorfismo anteponemos la palabra `virtual` a cada función polimórfica en la clase padre. Las clases hijas declaran funciones con el mismo nombre (pero sin `virtual`). Las clases hijas pueden llamar a las funciones polimórficas de sus clases padres usando el ámbito de clase, es decir `NombreDeLaClasePadre::nombreDelMetodo`.

Trabajo B2. Entrega Antes de: Jueves 2 de Mayo, a las 23:59 horas.

Implementa el siguiente diagrama de clases usando herencia y polimorfismo



- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `POO_TrabajoB2_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `POO_TrabajoB2_NOMBRE_DEL_AUTOR.*`

Ejemplo:
ASUNTO: `POO_TrabajoB2_JUAN_PEREZ_FERNANDEZ`
ADJUNTO: `POO_TrabajoB2_JUAN_PEREZ_FERNANDEZ.cpp`