FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia VII

Unidad II Programación orientada a objetos y estructuras de datos aplicadas a automatización

2.3 Herencia y polimorfismo (primera parte)

OBJETIVOS

Objetivos:

- Conceptualizar la herencia dentro de la Programación Orientada a Objetos.
- Implementar la herencia de clases en C++.
- Controlar el acceso de los usuarios de las clases derivadas a los miembros de la clase base.

BIBLIOGRAFÍA

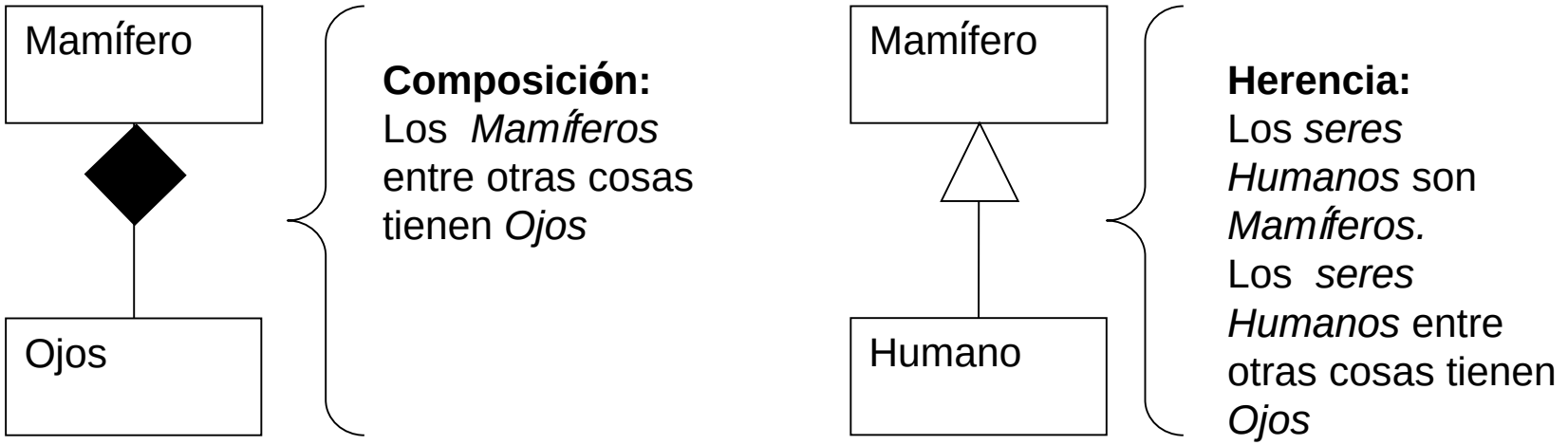
Bibliografía:

- Deitel and Deitel. Como programar en C/C++. Segunda edición o superior.
- García de Jalón, J.; y otros. Aprenda C++ como si estuviera en primero. Universidad de Navarra.

- **En la clase anterior:**

- **Preguntas iniciales:**
  - ¿Qué es un objeto?
  - ¿Qué es una clase?
  - ¿Qué utilidad tiene la POO?
  - ¿Utilizando clases de qué forma conocen que se pueda reutilizar el código previamente implementado?

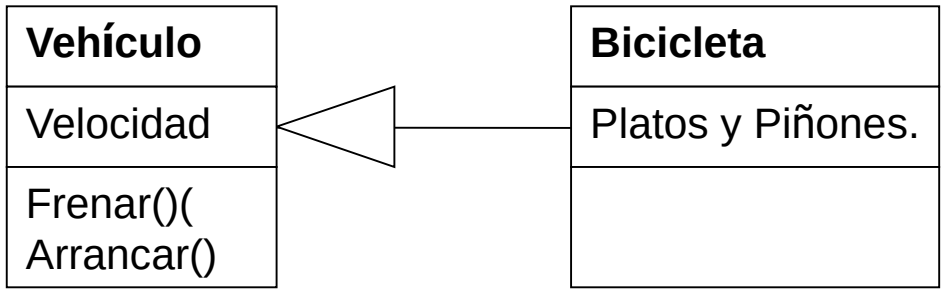
- ***La herencia como una relación entre objetos. Generalización-Especialización.***
  - La mente humana clasifica los *conceptos* de acuerdo a dos dimensiones: ***pertenencia*** y ***variedad***.



- ***Definición de La herencia como una forma más de reutilización de código.***
  - La herencia permite ***definir una clase modificando una o más clases*** ya existentes.
  - Estas modificaciones consisten habitualmente en ***añadir nuevos miembros*** (variables, funciones), a la clase que se está definiendo, aunque también se puede ***redefinir*** variables o funciones miembro ya existentes.
  - La clase de la que se parte en este proceso recibe el nombre de ***clase base***, y la nueva clase que se obtiene se denomina ***clase derivada***.
  - La ***clase base directa*** de una ***clase derivada*** es la clase de la que hereda directamente la ***clase derivada***.

HERENCIA EN POO

- En algunos casos una clase no tiene otra utilidad que la de ser ***clase base*** para otras clases que se deriven de ella. Por ejemplo, se puede definir la clase ***vehiculo*** para después derivar de ella ***coche***, ***bicicleta***, ***patinete***, etc., pero todos los objetos que se declaren pertenecerán a alguna de estas últimas clases; no habrá vehículos que sean sólo ***vehículos***.



- **Ejemplo:** Se desea modelar un círculo basado en su centro: punto cartesiano en el plano, y en su radio. Este debe permitir visualizar en pantalla sus atributos.

Ejemplo Clase Base:

```
class Figure{
  protected:
            float x,y;
  public:
            Figure(float=0, float=0);
            void setPoint(float, float);
            float getX() const {return x;};
            float getY() const {return y;};
            float area() const;
            void print() const;
  };
```

```
Figure::Figure(float a, float b) {
        x = a;
        y = b;
}
void Figure::setPoint(float a, float b) {
        x = a;
        y = b;
}
float Figure::area() {
        return 0;
}
void Figure::print() {
  cout<<"Point=["<<x<<","<<y<<"]"<<endl;
}
```

➢ Ejemplo Clase Derivada:

```
class Circle : public Figure{
  protected:
     float radius;
  public:
     Circle(float=0, float=0, float=0);
     void setRadius(float r)
     {if (r>=0) radius = r;}
     float getRadius() const { return radius;}
     float area() const;
     float print() const;
 };
```

```
Circle::Circle(float a, float b, float r)
          :Figure(a, b) {
   if (r>=0) radius = r;
}
float Circle::area() const {
       return radius*radius*3.14159;
}
void Circle::print() {
       Figure::print();
       cout<< "Radius = "<< radius << endl;
}
```

Herencia pública. Miembros protegidos.

Datos de la clase base.	Clase derivada de una clase Base **public:**
private:	No accesible.
protected:	protected
public:	public

- Sea B una clase base y D una clase derivada de B.
- Los miembros públicos de B son accesibles a todas las funciones de un programa.
- Los miembros privados de B son accesibles solo a las funciones miembros de B.
- Los miembros protegidos de B son accesibles solo a las funciones miembros de B y D.
- Observaciones sobre el ejemplo:
  - 1. B: Figure, D: Circle.
  - 2. Circle tiene acceso a x e y de su clase base Figure. Y a todas las funciones públicas de Figure.

Restricciones.

- Hay algunos elementos de la clase base que ***no pueden ser heredados***:
  - Constructores
  - Destructores
  - Funciones y datos estáticos de la clase

Nota: Existen otros elementos que no son heredados pero no están dentro de los objetivos de este curso por lo que se omiten.

Constructores y destructores de clases derivadas: inicializador base.

- Los constructores no son heredados, sin embargo pueden ser utilizados o llamados por los constructores de la clase derivada.
- En este caso debe utilizarse un inicializador, como en las clases compuestas.

Ejemplo:

```
Circle::Circle( float r, float a, float b ): Figure(a,b)  // inicializador de clase base directa.
{
 :
}
```

- El constructor de la clase derivada siempre llamará primero al constructor de la clase base aunque este no se escriba y el constructor por omisión hará esta tarea también.
- Los destructores se ejecutan en orden inverso, primero se llamará al destructor de la clase derivada y luego al de la clase base.

Redefinición de funciones miembros de la clase base en una clase derivada.

```
class Figure{
  protected:
        float x,y;
  public:
        Figure(float=0, float=0);
        void setPoint(float, float);
        float getX() const;
        float getY() const;
        float area() const;
        void print() const;
};
```

```
float Figure::area() {
      return 0;
}

void Figure::print() {
cout<<"Point=["<<x<<","<<y<<"]"<<endl;
}
```

```
class Circle : public Figure{
  protected:
     float radius;
  public:
     Circle(float=0, float=0, float=0);
     void setRadius(float r) ;
     float getRadius() const;
     float area() const;
     float print() const;
};
```

```
float Circle::area() const {
      return radius*radius*3.14159;
}

void Circle::print() {
      Figure::print();
      cout<< "Radius = "  << radius << endl;
}
```

Redefinición de funciones miembros de la clase base en una clase derivada.

- Si en la clase derivada (**Circle**) no se hubieran redefinido los métodos **area()** y **print(),** en un objeto de la clase **Circle,** la llamada a los métodos **area()** y **print(),** sería un llamado a estos métodos pertenecientes a la clase **Figure.**

- Al haber redefinido los métodos **area()** y **print()** en la clase **Circle**, en un objeto de la clase **Circle,** la llamada a estos métodos sería un llamado a los nuevos redefinidos en **Circle**.

- Si en el método **Circle::print()** en lugar de usar **Figure::print()** se llamara sólo a **print()**, entonces se convertiría en un método recursivo, o sea, se llamaría a si mismo.

Creación de objetos de una jerarquía de clases. Llamado a métodos.

- **Ejemplo:**

```
int main()
{
    Figure *figurePtr, f(3.1, 3.5);
    Circle *circlePtr, c(2.7, 1.2, 8.9);
    figurePtr = &f;
    circlePtr = &c;
    figurePtr ->print();              // imprime sólo el punto
    c.print();                        //imprime punto y radio
    float a= circlePtr->area()        // Área del círculo
    :
    :
}
```

Tipos de herencia. Herencia protegida y privada.

Datos de la clase base.	Clase derivada de una clase Base con **protected:**	Clase derivada de una clase Base con **private:**
private:	No accesible.	No accesible.
protected:	protected	private
public:	protected	private

- Con la herencia protegida, los miembros protegidos y públicos de la clase base se hacen protegidos. Este caso es muy raro pero puede suceder si se quiere que el programador usuario de la clase derivada sólo acceda a los nuevos miembros públicos definidos en esta clase y no a los de la clase base.
- Con la herencia privada, los miembros protegidos y públicos de la clase base se hacen privados. Este caso es aun más raro, y pocas veces práctico. Podría suceder cuando una clase derivada requiere redefinir el acceso de las siguientes clases de la jerarquía a sus miembros heredados a través de nuevas funciones de acceso. El programador usuario de la clase derivada sólo podrá acceder a los nuevos miembros definidos en esta clase como públicos.

CONCLUSIONES