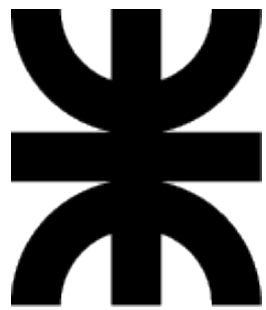
Informática II
Herencia de clases en C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Introducción

Herencia

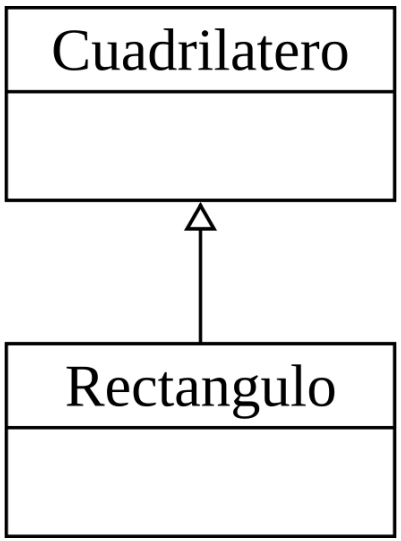
- Las nuevas clases se crean a partir de clases existentes, utilizando sus atributos y comportamiento y agregando nuevas capacidades.
- Es una de las principales características para la reutilización de software.

Introducción

Herencia

- Las nuevas clases se crean a partir de clases existentes, utilizando sus atributos y comportamiento y agregando nuevas capacidades.
- Es una de las principales características para la reutilización de software.

Ejemplo: Un rectángulo **es un** cuadrilátero (como lo es un cuadradro, un paralelogramo o un trapezoide)

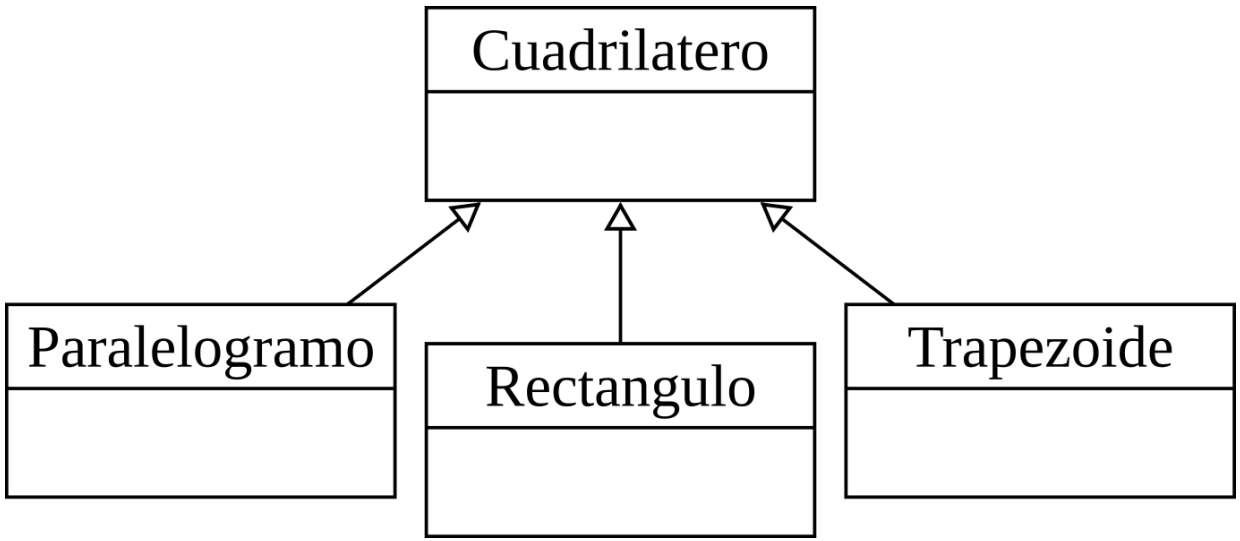


Introducción

Herencia

- Las nuevas clases se crean a partir de clases existentes, utilizando sus atributos y comportamiento y agregando nuevas capacidades.
- Es una de las principales características para la reutilización de software.

Ejemplo: Un rectángulo **es un** cuadrilátero (como lo es un cuadradro, un paralelogramo o un trapezoide)

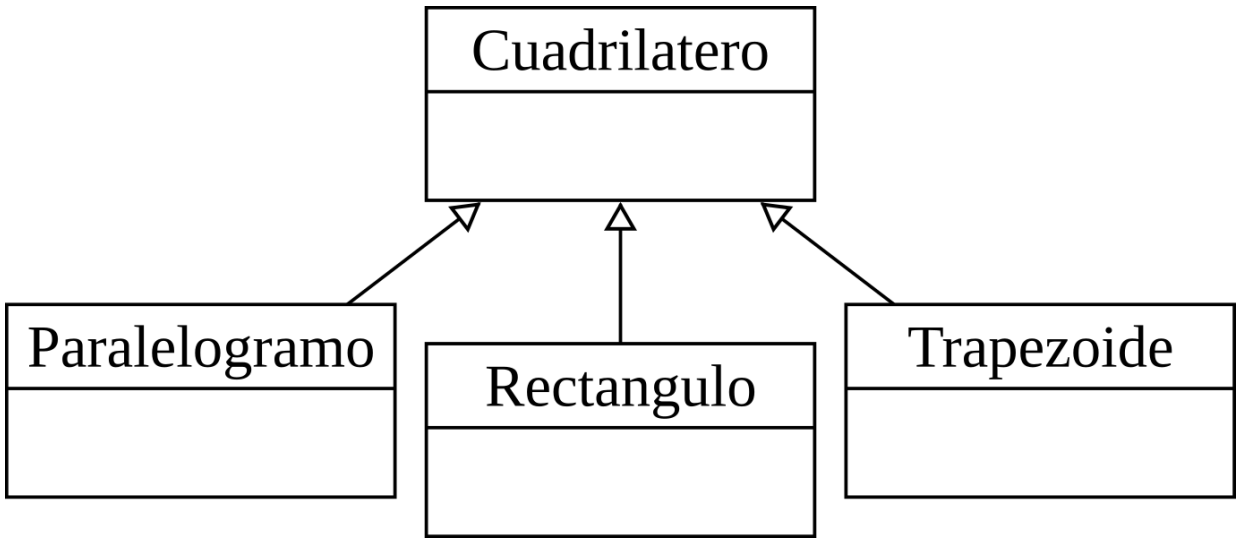


Introducción

Herencia

- Las nuevas clases se crean a partir de clases existentes, utilizando sus atributos y comportamiento y agregando nuevas capacidades.
- Es una de las principales características para la reutilización de software.

Ejemplo: Un rectángulo **es un** cuadrilátero (como lo es un cuadradro, un paralelogramo o un trapezoide)



(un cuadrilátero **no es un** rectángulo)

Introducción

- Cuando se crea una nueva clase, esta puede **heredar** los datos miembros y funciones miembros de una **clase base** definida previamente.

Introducción

- Cuando se crea una nueva clase, esta puede **heredar** los datos miembros y funciones miembros de una **clase base** definida previamente.
- A esta nueva clase se la conoce como **clase derivada**.

Introducción

- Cuando se crea una nueva clase, esta puede **heredar** los datos miembros y funciones miembros de una **clase base** definida previamente.
- A esta nueva clase se la conoce como **clase derivada**.
- Cada clase derivada puede a su vez ser clase base de otras clases.

Introducción

- Cuando se crea una nueva clase, esta puede **heredar** los datos miembros y funciones miembros de una **clase base** definida previamente.
- A esta nueva clase se la conoce como **clase derivada**.
- Cada clase derivada puede a su vez ser clase base de otras clases.
- En la **herencia simple** una clase deriva de una única clase base.

Introducción

- Cuando se crea una nueva clase, esta puede **heredar** los datos miembros y funciones miembros de una **clase base** definida previamente.
- A esta nueva clase se la conoce como **clase derivada**.
- Cada clase derivada puede a su vez ser clase base de otras clases.
- En la **herencia simple** una clase deriva de una única clase base.
- En la **herencia múltiple** una clase se deriva de varias clases bases.

Introducción

- Una clase derivada puede agregar nuevos datos y funciones miembros.

Introducción

- Una clase derivada puede agregar nuevos datos y funciones miembros.
- En general, una clase derivada es “más grande” que su clase base.

Introducción

- Una clase derivada puede agregar nuevos datos y funciones miembros.
- En general, una clase derivada es “más grande” que su clase base.
- Una clase derivada es más específica que su clase base y representa a un grupo más pequeño de objetos.

Introducción

- Una clase derivada puede agregar nuevos datos y funciones miembros.
- En general, una clase derivada es "más grande" que su clase base.
- Una clase derivada es más específica que su clase base y representa a un grupo más pequeño de objetos.

En C++ se puede definir tres tipos de herencias: pública, protegida y privada.

Introducción

- Una clase derivada puede agregar nuevos datos y funciones miembros.
- En general, una clase derivada es "más grande" que su clase base.
- Una clase derivada es más específica que su clase base y representa a un grupo más pequeño de objetos.

En C++ se puede definir tres tipos de herencias: pública, protegida y privada.

- Con la herencia pública, cada objeto de la clase derivada es también un objeto de la clase base.
- Sin embargo, lo inverso no es verdad, los objetos de la clase base no son objetos de la clase derivada.

Herencia – Ejemplos

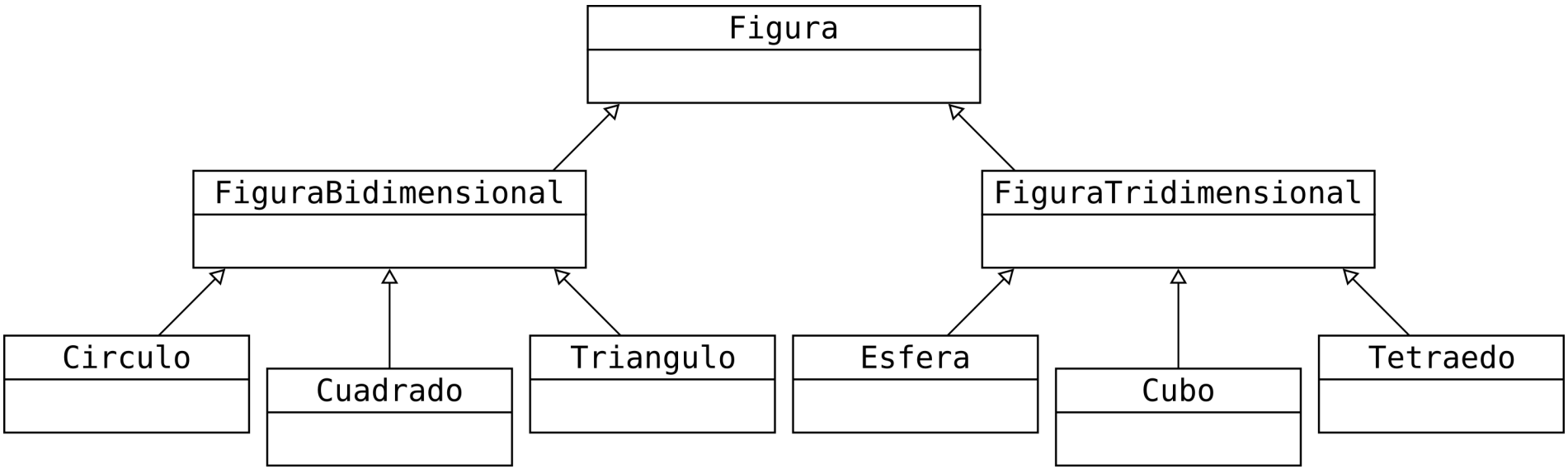
Más ejemplos de herencia:

Figura:	`Circulo`, `Triangulo`, `Rectangulo`
Estudiante:	`EstudianteUniversitario`, `EstudianteTitulado`
Empleado:	`EmpleadoDocente`, `EmpleadoAdministrativo`
Cuenta:	`CuentaCheques`, `CuentaAhorros`

Herencia – Ejemplos

La herencia forma una estructura jerárquica en forma de árbol.

Ejemplo: jerarquía de clases `Figura`



Herencia – Definición

Para indicar que la clase `Circulo` deriva de la clase `FiguraBidimensional`, la clase `Circulo` se define como

Herencia – Definición

Para indicar que la clase `Circulo` deriva de la clase `FiguraBidimensional`, la clase `Circulo` se define como

```
class Circulo : public FiguraBidimensional {
  . . .
};
```

Herencia – Definición

Para indicar que la clase `Circulo` deriva de la clase `FiguraBidimensional`, la clase `Circulo` se define como

```
class Circulo : public FiguraBidimensional {
  . . .
};
```

- Esto se conoce como herencia pública (la más utilizada).
- Los miembros públicos y protegidos se heredan como miembros públicos y protegidos, respectivamente.
- Las funciones amigas no se heredan.

Herencia – Definición

Para indicar que la clase `Circulo` deriva de la clase `FiguraBidimensional`, la clase `Circulo` se define como

```
class Circulo : public FiguraBidimensional {
  . . .
};
```

- Esto se conoce como herencia pública (la más utilizada).
- Los miembros públicos y protegidos se heredan como miembros públicos y protegidos, respectivamente.
- Las funciones amigas no se heredan.

Se utiliza una nueva forma de control de acceso a miembro, el acceso protegido –`protected` (además de `private` y `public`).

Miembros `protected`

- El acceso `protected` es un nivel intermedio de protección entre acceso público y privado.
- Los miembros `protected` de la clase base se acceden mediante miembros y amigas de la clase base y por medio de miembros y amigas de la clase derivada.
- La clase derivada puede acceder a los miembros públicos y protegidos de la clase base utilizando su nombre.

Miembros `protected`

- El acceso `protected` es un nivel intermedio de protección entre acceso público y privado.
- Los miembros `protected` de la clase base se acceden mediante miembros y amigas de la clase base y por medio de miembros y amigas de la clase derivada.
- La clase derivada puede acceder a los miembros públicos y protegidos de la clase base utilizando su nombre.

Recordar que:

- Se puede acceder a los miembros `public` de la clase base desde todas las funciones del programa.
- Los miembros `private` son accesible solo de las funciones miembros y `friend` de la clase base.

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  class Punto {
2    public:
3      Punto(double = 0.0, double = 0.0); // constructor predeterminado
4      void establecerPunto(double , double );// establece coordenadas
5      double obtenerX() const { return x; } // obtiene la coordenada x
6      double obtenerY() const { return y; } // obtiene la coordenada y
7      void imprimir() const;
8
9
10 };
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  class Punto {
2    public:
3      Punto(double = 0.0, double = 0.0); // constructor predeterminado
4      void establecerPunto(double , double );// establece coordenadas
5      double obtenerX() const { return x; } // obtiene la coordenada x
6      double obtenerY() const { return y; } // obtiene la coordenada y
7      void imprimir() const;
8    protected: // accesible para las clase derivadas
9      double x, y; // las coordenadas x e y de Punto
10 };
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  class Punto {
2    public:
3      Punto(double = 0.0, double = 0.0); // constructor predeterminado
4      void establecerPunto(double , double );// establece coordenadas
5      double obtenerX() const { return x; } // obtiene la coordenada x
6      double obtenerY() const { return y; } // obtiene la coordenada y
7      void imprimir() const;
8    protected: // accesible para las clase derivadas
9      double x, y; // las coordenadas x e y de Punto
10 };
```

```
1  class Circulo : public Punto { // Circulo hereda de Punto
2
3
4
5
6
7
8
9
10
11 };
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  class Punto {
2    public:
3      Punto(double = 0.0, double = 0.0); // constructor predeterminado
4      void establecerPunto(double , double );// establece coordenadas
5      double obtenerX() const { return x; } // obtiene la coordenada x
6      double obtenerY() const { return y; } // obtiene la coordenada y
7      void imprimir() const;
8    protected: // accesible para las clase derivadas
9      double x, y; // las coordenadas x e y de Punto
10 };
```

```
1  class Circulo : public Punto { // Circulo hereda de Punto
2    public:
3      // constructor predeterminado
4      Circulo(double r = 0.0, double x = 0.0, double y = 0.0);
5      void establecerRadio(double ); // establece el radio
6      double obtenerRadio() const; // devuelve el radio
7      double area() const; // calcula el área
8      void imprime() const;
9
10
11 };
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  class Punto {
2    public:
3      Punto(double = 0.0, double = 0.0); // constructor predeterminado
4      void establecerPunto(double , double );// establece coordenadas
5      double obtenerX() const { return x; } // obtiene la coordenada x
6      double obtenerY() const { return y; } // obtiene la coordenada y
7      void imprimir() const;
8    protected: // accesible para las clase derivadas
9      double x, y; // las coordenadas x e y de Punto
10 };
```

```
1  class Circulo : public Punto { // Circulo hereda de Punto
2    public:
3      // constructor predeterminado
4      Circulo(double r = 0.0, double x = 0.0, double y = 0.0);
5      void establecerRadio(double ); // establece el radio
6      double obtenerRadio() const; // devuelve el radio
7      double area() const; // calcula el área
8      void imprimir() const;
9    protected:
10     double radio;
11 };
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```
1  #include <iostream>
2  #include "punto3.h"
3  #include "circulo3.h"
4  using namespace std;
5
6  int main()
7  {
8      Punto p1(7.5, 9.5);
9      Circulo c1(5.2, 2.5, 2.5);
10
11     cout << "Punto p1: "; p1.imprimir();
12     cout << "\nCírculo c1: "; c1.imprimir();
13
14     c1.establecerPunto(3.3, 4.4);
15     cout << "\nCírculo c1: "; c1.imprimir();
16     cout << "\nÁrea: " << c1.area() << endl;
17     return 0;
18 }
```

Ejemplo – Clase `Punto` y `Circulo` (mod.)

```cpp
#include <iostream>
#include "punto3.h"
#include "circulo3.h"
using namespace std;

int main()
{
    Punto p1(7.5, 9.5);
    Circulo c1(5.2, 2.5, 2.5);

    cout << "Punto p1: "; p1.imprimir();
    cout << "\nCírculo c1: "; c1.imprimir();

    c1.establecerPunto(3.3, 4.4);
    cout << "\nCírculo c1: "; c1.imprimir();
    cout << "\nÁrea: " << c1.area() << endl;
    return 0;
}
```

```
Punto p1: [7.5, 9.5]
Círculo c1: Centro = [2.5, 2.5]; Radio = 5.20
Círculo c1: Centro = [3.30, 4.40]; Radio = 5.20
Área: 84.95
```

Ejemplo: `Punto`, `Circulo` y `Cilindro`

Ver código fuente ejemplo del D&D modificado:

- `punto3.h`, `punto3.cpp` y `fig19_08.cpp`
- `circulo3.h`, `circulo3.cpp` y `fig19_09.cpp`
- `cilindro3.h`, `cilindro3.cpp` y `fig19_10.cpp`