Tecnológico de Monterrey

Herencia

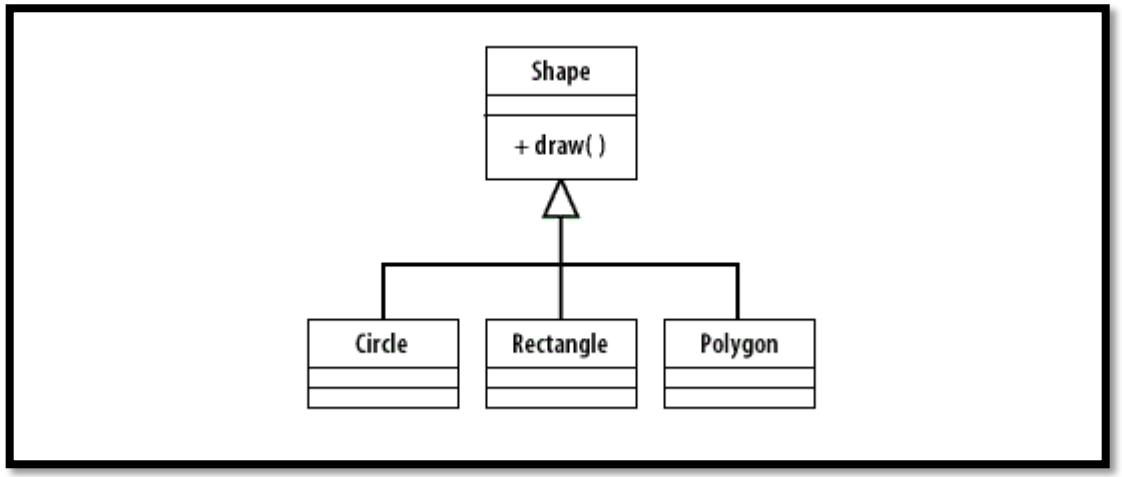
Inheritance

Tecnológico de Monterrey

Herencia – Conceptos Clave

- Es un concepto que le permite a una clase transmitirle todo lo que ya tiene, a clases que derivan de ésta.
- La herencia permite el reuso de código, o en otras palabras: “No reinventar la rueda”
- Hay lenguajes de programación que permiten solamente herencia simple (ej: un solo padre). C++ permite tener uno o más padres

Herencia - UML



La notación UML permite representar este concepto.

Para este caso lo que este diagrama nos dice es:

La figura **padre** se llama Shape y contiene
un método público (símbolo +) llamado draw()
Las figuras hijas se llaman Circle, Renctagle y Polygon

Notar que por definición las clases Hijas ya tienen el método
draw (ya que fue definido en la clase padre)
por lo que no es necesario agregarlo

Tecnológico de Monterrey

Herencia – Representación en código

- Para dejar claro el concepto de herencia el código que se presenta a continuación asume lo siguiente:
  - La clase Shape contiene los atributos X , Y que representan el origen de una figura (para un círculo es el centro, para cualquier otra figura es su esquina superior derecha
  - La clase Circle tiene el atributo radio
  - La clase Rectangle tiene los atributos Alto y Ancho (nota para la práctica)
  - La clase Polygon tiene el atributo NumeroDeLados (no para la práctica)

- Para efectos demostrativos, el método **draw** imprime “soy una figura” sin importar si es Rectangle, Circle o Shape

Código de Shape (.h)

*Shape.h | Shape.cpp | *main.cpp | Circle.h | Circle.cpp

```
1   #ifndef SHAPE_H_INCLUDED
2   #define SHAPE_H_INCLUDED
3
4   #include <iostream>
5
6   using namespace std;
7
8   class Shape
9   {
10  private:
11      int x;
12      int y;
13
14  public:
15      Shape();
16      Shape(int, int);
17      string draw();
18
19
20      // los métodos debajo se usarán para imprimir los valores
21      // de x y de y , NO ES LA MEJOR SOLUCION
22      // pero se mejorara más adelante
23      int getValueX();
24      int getValueY();
25  };
26
27  #endif // SHAPE_H_INCLUDED
28
```

Tecnológico de Monterrey

Código de Shape (.cpp)

*Shape.h | **Shape.cpp** | *main.cpp | Circle.h | Circle.cpp

```cpp
#include "Shape.h"

Shape::Shape()
{
    x = 0;
    y = 0;
}

Shape::Shape(int valX, int valY)
{
    x = valX;
    y = valY;
}

string Shape::draw()
{
    return "soy una figura";
}

int Shape::getValueX()
{
    return x;
}

int Shape::getValueY()
{
    return y;
}
```

Código de Circle (.h)

*Shape.h | Shape.cpp | *main.cpp | Circle.h | Circle

```
1  #ifndef CIRCLE_H_INCLUDED
2  #define CIRCLE_H_INCLUDED
3
4  #include "Shape.h"
5
6  class Circle:public Shape
7  {
8
9  private:
10     int r;
11
12 public:
13     Circle();
14     Circle(int,int ,int);
15
16 };
17
18
19 #endif // CIRCLE_H_INCLUDED
20
```

La notación para heredar es poner “:” después
de la declaración de la clase, el modificador de acceso
(private, protected, public) y
La clase de la que hereda

En este caso Circle hereda **todo** lo que se definió en Shape
esto es, tiene X, Y, los constructores y el método draw,
esto incluye lo que se definió en el archive Shape.cpp

Quieres saber más?
https://www.learncpp.com/cpp-tutorial/115-inheritance-and-access-specifiers/

Código de Circle (.cpp)

*Shape.h | Shape.cpp | *main.cpp | Circle.h | **Circle.cpp**

```cpp
#include "Circle.h"

Circle::Circle()
{
    r = 0;
}

Circle::Circle(int valX, int valY, int valR):Shape(valX, valY)
{
    r = valR;
}
```

Para el constructor no default, hay que llamar al constructor del padre para inicializar los valores.

Notar que hay menos métodos, ya que en la clase Shape se definieron

Código de main (.cpp)

Shape.h | Shape.cpp | **main.cpp** | Circle.h | Circle.cpp

```cpp
#include <iostream>
#include "Shape.h"
#include "Circle.h"

using namespace std;

int main()
{

    Shape figural(1,2);

    cout <<  figural.draw() <<  " con valor x en : " << figural.getValueX() << " y  valor y en : " << figural.getValueY() << endl;

    Circle circulo1(2,3,5);

    cout <<  circulo1.draw() <<  " con valor x en : " << circulo1.getValueX() << " y  valor y en : " << circulo1.getValueY() << endl;

    return 0;

}
```

En este ejemplo se crean dos figuras, un Shape y un Circle, notar como circulo1 que es de tipo Circle, puede usar draw sin problemas

Tecnológico de Monterrey

Práctica

- Definir las clases Rectangle y Polygon, ambas heredan de Shape