Tecnológico de Monterrey

# Polimorfismo

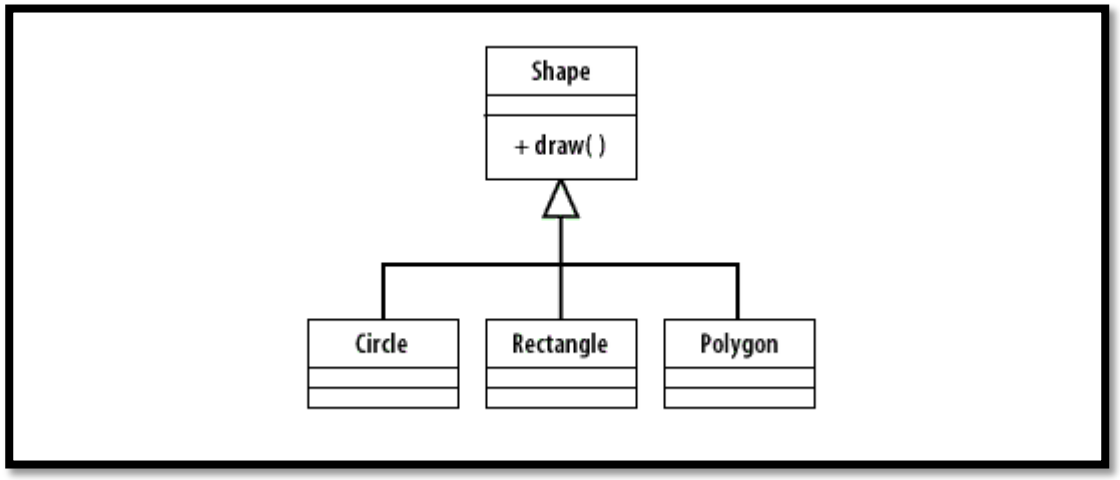
Polymorphism

# Polimorfismo – Conceptos Clave

- Es un concepto que le permite a objetos recibir el mismo mensaje y comportarse de forma distinta ¿what?
- Este concepto es mucho más fácil de explicar con un escenario

  (descrito a continuación)

# Polimorfismo




Retomando el problema de las figuras:
Como es de esperarse, cada figura se dibuja de forma distinta,

Ejemplo:

Un círculo se dibuja:

Un cuadrado se dibuja:

Para efectos demostrativos, ahora cada figura cuando se mande llamar a su método **draw** indicará qué tipo de figura dibujará en modo texto

# Polimorfismo

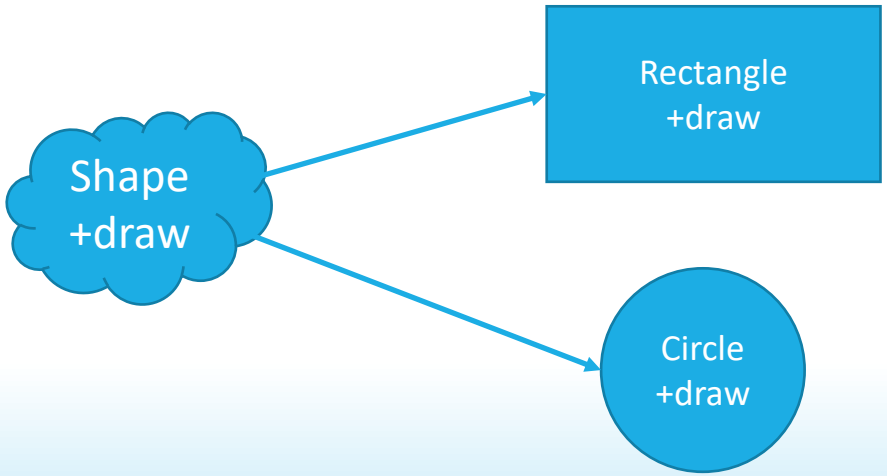
- Por qué es bueno?
- Si se declararan 40 círculos y 30 cuadrados (y no existiera el polimorfismo) algo así se tendría que hacer si los desearamos dibujar

```
int main()
{

    Circle circulo1(2,2,4);
    Circle circulo2(2,2,4);
    Circle circulo3(2,4,4);
    Circle circulo4(2,5,4);
    Circle circulo5(2,6,6);
    Circle circulo6(8,2,4);
    Circle circulo6(8,2,5);
    Circle circulo7(9,2,4);
    Circle circulo8(1,2,3);
    //...
    //...
    //...
    //...
    Circle circulo40(2,2,5);

    // y todavía faltan los cuadrados

    circulo1.draw();
    circulo2.draw();
    circulo3.draw();
    circulo4.draw();
    //...
    //...
    //...
    //...
```

Y todavía faltan los draw de los cuadrados… 70 líneas de código para declaración y otras 70 para dibujar

# Polimorfismo

- Solución?
- Mejor crear un objeto que acepte muchos de tipo Shape (que es el padre) y que guarde una referencia a los hijos de tal forma que si se manda llamar al método draw de Shape, se mande llamar al método draw requerido



Tecnológico de Monterrey

# Polimorfismo – qué se requiere

- La palabra reservada virtual en el método de la clase padre que presentará el polimorfismo
- El método con la misma firma en la(s) clase(s) hija(s)
- Entender Apuntadores: ya que el objeto que guardará las referencias será un apuntador a tipos de figura Shape

# Código de Shape (.h) mejorado

```
#ifndef SHAPE_H_INCLUDED
#define SHAPE_H_INCLUDED

#include <iostream>

using namespace std;

class Shape
{
private:
    int x;
    int y;

public:
    Shape();
    Shape(int, int);
    virtual string draw();


    // los métodos debajo se usarán para imprimir los v
    // de x y de y , NO ES LA MEJOR SOLUCION
    // pero se mejorara más adelante
    int getValueX();
    int getValueY();
};

#endif // SHAPE_H_INCLUDED
```

La palabra virtual en un método de la clase padre

# Código de Shape (.cpp) mejorado

```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;
}
```

Notar que la palabra reservada virtual solamente se Indica en la declaración (no hay cambios aquí)

# Código de Circle (.h) mejorado

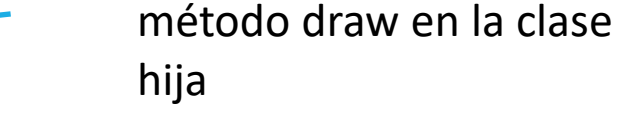
```
1   #ifndef CIRCLE_H_INCLUDED
2   #define CIRCLE_H_INCLUDED
3
4   #include "Shape.h"
5
6   class Circle:public Shape
7   {
8
9   private:
10      int r;
11
12  public:
13      Circle();
14      Circle(int,int ,int);
15      string draw();
16
17  };
18
19
20  #endif // CIRCLE_H_INCLUDED
21
```

En la clase Circle (hija)
Se declara el método draw

Notar que el método tiene exactamente la misma firma que la clase padre Shape

# Código de Circle (.cpp) mejorado

```cpp
#include "Circle.h"

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

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


string Circle::draw()
{
   return "soy un circulo" ;
}
```



Se implementa el método draw en la clase hija

Tecnológico de Monterrey

# Código de Rectangle (.h)

```
1   #ifndef RECTANGLE_H_INCLUDED
2   #define RECTANGLE_H_INCLUDED
3
4   #include "Shape.h"
5
6   class Rectangle: public Shape
7   {
8
9   private:
10      int x;
11      int y;
12      int xl;
13      int yl;
14
15
16  public:
17      Rectangle();
18      Rectangle(int,int ,int, int);
19      string draw();
20
21  };
22
23
24  #endif // RECTANGLE_H_INCLUDED
25
```

# Código de Rectangle (.cpp)

```cpp
#include "Rectangle.h"


Rectangle::Rectangle()
{
    x = 0;
    y = 0;
    xl = 0;
    yl = 0;

}

Rectangle::Rectangle(int valX, int valY, int valXl, int valYl):Shape(valX, valY)
{

    xl = valXl;
    yl = valYl;
}

string Rectangle::draw()
{
    return "soy un rectangulo";
}

```

# Código de main (.cpp) Explicación

```
1   #include <iostream>
2   #include "Shape.h"
3   #include "Circle.h"
4   #include "Rectangle.h"
5
6   using namespace std;
7
8   int main()
9   {
10
11
12      Shape *Shapes[5];
13      Shapes[0] = new Circle();
14      Shapes[1] = new Rectangle();
15      Shapes[2] = new Rectangle(1,2,5,6);
16      Shapes[3] = new Circle(4,2,1);
17      Shapes[4] = new Rectangle();
18
19      for (int i = 0; i < 5; i++)
20      {
21          Shape *current = Shapes[i];
22          cout <<  current->draw()  << "\n";
23      }
24
25
26      return 0;
27
28  }
29
```

Se declara un apuntador que manejará las referencias a los objetos de tipo Shape, Circle

Se declaran objetos de tipo Circle y Rectangulo y se guarda su referencia en un arreglo de “Shapes”

Con este ciclo se pueden imprimir todas las figuras que estén guardadas en el Vector

# Resultado

```
soy un circulo
soy un rectangulo
soy un rectangulo
soy un circulo
soy un rectangulo

Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

Como se puede observar, se imprime lo que cada objeto indica que es de acuerdo
A su método draw()

# En la vida real

- Imagina un escenario donde se quiere calcular la nómina a pagar de 30,000 empleados donde hay diferentes formas de pagar:
  - sueldo base
  - sueldo base más comisiones
  - por horas
  - por destajo (piezas entregadas por ejemplo)

- Esto implicaría, sin polimorifismo, crear un tipo de objeto por cada uno prácticamente de forma manual y mandar llamar al método de calcularPago de cada uno de forma manual – el polimorfismo permite que con un ciclo se pueda hacer…

  O en otras palabras, en vez de 30,000 líneas de código que sean solamente 2

Podrían ser en realidad unas 30 líneas de código no optimizadas, pero apantalla más 30,000

Tecnológico de Monterrey

# Práctica

- Definir la clase Polygon, ésta hereda de Shape y crear su método draw
- Añadir 4 objetos de tipo Rectangle y 4 objetos de tipo Polygon
- Imprimir lo que son usando el mismo ciclo for

- Hint: NO HAY que tocar el ciclo for, solo agregar más objetos después de la línea 17