Universidad de Costa Rica

IE-0217 - Estructuras abstractas de datos y algoritmos para ingeniería

Grupo de Trabajo #3

Profesor: M.Sc. Ricardo Roman Brenes

Laboratorio 2: Herencia, polimorfismo y sobrecargas

Estudiantes: — *Carné:*
Natanael Mojica — A93901
Moisés Campos Zepeda — B31400

I semestre - 2017

Índice

1. Enunciado **1**
1.1. Clase base . . . 1
1.2. Clases derivadas . . . 1
1.3. Sobrecarga de operadores . . . 1
1.4. Graficación . . . 1
1.5. Diagrama de clases . . . 1

2. Código **2**
2.1. figura.h . . . 2
2.2. Clases derivadas . . . 2
2.2.1. Clase circulo . . . 2
2.2.2. Clase rectangulo . . . 5
2.2.3. Clase triangulo . . . 7
2.3. Main . . . 10

3. Conclusiones **13**

4. Anexos **14**

1. Enunciado

Diseñe, desarrolle y documente una serie de clases que modelen figuras geométricas.

1.1. Clase base

Creen una clase base llamada Figura que tenga como atributos:

- Un nombre
- Un color

Y como métodos virtuales:

- El cálculo del área.
- El cálculo del perímetro.

1.2. Clases derivadas

Además cree tres clases Triangulo, Círculo y Cuadrado, que hereden de la clase Figura y contengan sus características propias y además que reimplemente los métodos virtuales de la clase base.

1.3. Sobrecarga de operadores

Dentro de las clases derivadas sobrecargue los operadores unarios ỹ para que respectivamente impriman un desglose de los datos del objeto y los valores calculados de área y perímetro.

1.4. Graficación

Usando la biblioteca SFML https://www.sfml-dev.org/ grafique las figuras que implemento. Para esto agregue a sus clases derivadas un objeto de tipo, ya sea CircleShape o ConvexShape, según sea necesario (círculo o triangulo y rectángulo). Además para los círculos, necesitará modelar el centro como un vértice y para los polígonos, necesitará un arreglo de vértices. Averigüe como se utilizan y se dibujan dichas figuras. Por último, utilizando polimorfismo, implemente un método dibujar que reciba por parámetros un puntero de tipo RenderWindow y que invoque el m´etodo draw de la clase RenderWindow con el atributo de tipo Shape anteriormente agregado a sus clases derivadas. Agregue todas sus figuras a un arreglo de Figuras y dibújelas.

1.5. Diagrama de clases

Por último construya el diagrama de clases de estas cuatro clases siguiendo el estándar de UML.

2. Código

En la propuesta de laboratorio se indica una estructura a seguir para resolver el problema indicado. Basados en esta estructura se implementaron las siguientes clases y código:

2.1. figura.h

La clase de la que heredarán nuestras siguientes clases derivadas. Posee 3 funciones virtuales y 3 parámetros: El área, el perímetro y la información, representados por c_area, perimetro e info.

Listing 1: Clase Figura

```
1 #ifndef _FIGURA_
2 #define _FIGURA_
3
4 #include <iostream>
5
6
7 class figura
8 {
9   public:
10   /**  @brief función virtual c_area utilizada para llamar a la función
11    * de area debida a la hora de correr el programa.
12   **/
13         virtual void c_area()     = 0;
14
15   /**  @brief funciónes virtuales c_area utilizada para llamar a la función
16    * de perímetro debida a la hora de correr el programa.
17   **/
18         virtual void perimetro()  = 0;
19
20   /**  @brief funciónes virtuales c_area utilizada para llamar a la
21    * identificación la hora de correr el programa.
22   **/
23         virtual void info()       = 0;
24
25         //parametros a heredar
26         std::string name,
27                     color;
28         int         id;
29
30 };
31
32 #endif
```

2.2. Clases derivadas

2.2.1. Clase circulo

Clases derivada de figura. Esta clase posee un constructor, el cual inicializa recibiendo un valor radio, y los demás parámetros que reciben las demás clases.

Listing 2: Header de la clase circulo

```
1 #ifndef _CIRCULO_
```

```
2 #define _CIRCULO_
3
4 #include "figura.h"
5 #include <SFML/Graphics.hpp>
6 //#include <CircleShape.hpp>
7
8 class circulo: public figura
9 {
10   public:
11         circulo(double radio, int id, std::string nombre, std::string color);
12         ~circulo();
13         sf::CircleShape circle;
14         sf::CircleShape dibujar(sf::RenderWindow *rW);
15         void c_area();
16         void perimetro();
17         void info();
18
19         void operator ~(){
20             std::cout << "nombre: "<<this->name<<" - color: "<<this->color<<" -
     radio: "<<this->radio<<std::endl;
21         }
22         void operator!(){
23             std::cout << "area: "<<this->area <<" - perimetro: "<<this->peri<<
     std::endl;
24         }
25
26         circulo* operator=(const circulo *t)
27         {
28             if(this != t){
29                 this->id = t->id;
30                 this->radio = t->radio;
31                 this->name = t->name;
32                 this->color = t->color;
33                 this->peri = t->peri;
34                 this->area = t->area;
35             }
36             return this;
37         }
38
39         inline int get_num_circulos(){
40
41             return circulo::num_circulos;
42         }
43
44   static int num_circulos;
45
46   private:
47
48         double radio,
49                area,
50                peri;
51
52 };
53
54 #endif
```

Listing 3: .cpp de la clase circulo

```
#include "../include/circulo.h"

#define PI 3,14159265358

using namespace std;

int circulo::num_circulos = 0;

circulo::circulo(double radio, int id, string nombre, string color ){

  this->id = id;
  circulo::num_circulos += 1;
  this->radio = radio;
  this->area = 0;
  this->peri = 0;
  this->name = nombre;
  this->color = color;
  circle.setRadius(this->radio);
  circle.setPosition (150,200);
}

circulo::~circulo()
{

}

void circulo::info()
{
    cout << "radio:"<< this->radio <<"area: "<< this->area << "perimetro: "<<this
    ->peri<<endl;

}


sf::CircleShape circulo::dibujar(sf::RenderWindow *rW){

  circle.setFillColor(sf::Color::Red);

  return circle;
}

void circulo::c_area()
{
  this->area = PI * (this->radio * this->radio);
}

void circulo::perimetro()
{
  this->peri = 2*PI*this->radio;
}
```

2.2.2. Clase rectangulo

Clase derivada de figura, para identificar y graficar un rectángulo.Para este caso, la clase posee un constructor, el cual inicializa los parámetros particulares *double lado1* y *double lado2*; los demás parámetros siendo heredados y definidos para este caso particular.

Listing 4: header de la clase rectangulo

```
#ifndef _RECTANGULO_
#define _RECTANGULO_

#include <SFML/Graphics.hpp>
#include "figura.h"

class rectangulo: public figura
{
  public:
        rectangulo( double lado1, double lado2, int id, std::string nombre, std::
    string color );
        ~rectangulo();
        void c_area();
        void perimetro();
        void info();
        sf::ConvexShape dibujar(sf::RenderWindow *rW);

        sf::ConvexShape rect;

    void operator ~(){
      std::cout << "nombre: "<<this->name <<" - color: "<<this->color<<" - lado1: "
    "<<this->lado1<<" - lado2: "<<this->lado2<<std::endl;
    }

    void operator !(){
      std::cout << "area: "<<this->area <<" - perimetro: "<<this->peri<<std::endl;
    }

        rectangulo* operator=(const rectangulo *t)
        {
            if(this != t){
                this->id = t->id;
                this->lado1 = t->lado1;
                this->lado2 = t->lado2;
                this->name = t->name;
                this->color = t->color;
                this->peri = t->peri;
                this->area = t->area;
            }
            return this;
        }

       inline int get_num_rectangulos(){

            return rectangulo::num_rectangulos;
        }

```

```
46          static int num_rectangulos;
47
48
49    private:
50          double area,
51              lado1,
52              lado2,
53              peri;
54 };
55 #endif
```

Listing 5: .cpp rectangulo

```
1  #include "../include/rectangulo.h"
2
3
4  using namespace std;
5
6  int rectangulo::num_rectangulos = 0;
7
8  rectangulo::rectangulo( double lado1, double lado2, int id, std::string nombre,
       std::string color )
9  {
10
11   this->id = id;
12   this->lado1 = lado1;
13   this->lado2 = lado2;
14   this->area = 0;
15   this->peri = 0;
16   this->name = nombre;
17   this->color = color;
18   this->rect.setPointCount(4);
19     rectangulo::num_rectangulos += 1;
20
21 }
22
23 rectangulo::~rectangulo()
24 {
25
26 }
27
28 void rectangulo::info(){
29
30     cout << "lado 1 y 2:"<< this->lado1 <<"| "<<this->lado2<<"area: "<< this->
       area << "perimetro: "<<this->peri<<endl;
31
32 }
33
34 void rectangulo::c_area()
35 {
36
37   this->area = this->lado1 * this->lado2;
38 }
39
40 void rectangulo::perimetro()
41 {
```

```
42
43   this->peri = 2*(this->lado1 + this->lado2);
44 }
45
46 sf::ConvexShape rectangulo::dibujar(sf::RenderWindow *rW){
47
48   rect.setPoint(0, sf::Vector2f(400,125));
49   rect.setPoint(1, sf::Vector2f(400+this->lado1, 125));
50   rect.setPoint(2, sf::Vector2f(400+this->lado1, 125+this->lado2));
51   rect.setPoint(3, sf::Vector2f(400, 125+this->lado2));
52   rect.setFillColor(sf::Color(20, 125, 250));
53
54   return rect;
55 }
```

2.2.3. Clase triangulo

Clase derivada de figura, para identificar y graficar un triángulo. En este último caso, la clase posee en su constructor, el parámetro particular *double lado*, en razón de que tratamos de dibujar un *triángulo equilátero*; los demás parámetros siendo heredados y definidos para este caso particular.

Listing 6: header de la clase triángulo

```
1
2 #ifndef _TRIANGULO_
3 #define _TRIANGULO_
4 #include "figura.h"
5 #include <SFML/Graphics.hpp>
6 class triangulo: public figura
7 {
8   public:
9         triangulo(double lado, int id, std::string nombre, std::string color);
10        ~triangulo();
11        void c_area();
12        void perimetro();
13        void info();
14        sf::CircleShape tri;
15        sf::CircleShape dibujar(sf::RenderWindow *rW);
16
17    void operator ~(){
18      std::cout << "nombre: "<<this->name <<" - color: "<<this->color<<" - lado "
    <<this->lado<<std::endl;
19    }
20    void operator !(){
21      std::cout << "area: "<<this->area <<" - perimetro: "<<this->peri<<std::endl
    ;
22    }
23
24        triangulo* operator=(const triangulo *t)
25        {
26            if(this != t){
27                this->id = t->id;
28                this->lado = t->lado;
29                this->name = t->name;
30                this->color = t->color;
```

```
31                  this->peri = t->peri;
32                  this->area = t->area;
33              }
34              return this;
35          }
36
37          inline int get_num_triangulos(){
38
39              return triangulo::num_triangulos;
40          }
41
42          static int num_triangulos;
43
44
45    private:
46
47          double  area,
48          lado,
49          peri;
50
51 };
52 #endif
53 }, language=C++]
54 #ifndef _TRIANGULO_
55 #define _TRIANGULO_
56
57 #include "figura.h"
58 #include <SFML/Graphics.hpp>
59
60 class triangulo: public figura
61 {
62   public:
63          triangulo(double lado, int id, std::string nombre, std::string color);
64          ~triangulo();
65          void c_area();
66          void perimetro();
67          void info();
68          sf::CircleShape tri;
69          sf::CircleShape dibujar(sf::RenderWindow *rW);
70
71      void operator ~(){
72        std::cout << "nombre: "<<this->name <<" - color: "<<this->color<<" - lado "
      <<this->lado<<std::endl;
73      }
74      void operator !(){
75        std::cout << "area: "<<this->area <<" - perimetro: "<<this->peri<<std::endl
      ;
76      }
77
78          triangulo* operator=(const triangulo *t)
79          {
80              if(this != t){
81                  this->id = t->id;
82                  this->lado = t->lado;
83                  this->name = t->name;
84                  this->color = t->color;
```

```
85                  this->peri = t->peri;
86                  this->area = t->area;
87              }
88              return this;
89          }
90
91          inline int get_num_triangulos(){
92
93              return triangulo::num_triangulos;
94          }
95
96          static int num_triangulos;
97
98
99    private:
100
101         double  area,
102         lado,
103         peri;
104
105 };
106 #endif
```

Listing 7: .cpp de la clase triángulo

```
1  #include <math.h>
2  #include "../include/triangulo.h"
3
4  using namespace std;
5
6  int triangulo::num_triangulos = 0;
7
8  triangulo::triangulo(double lado,  int id, string nombre, string color){
9
10   this->id = id;
11   triangulo::num_triangulos += 1;
12   this->lado = lado;
13   this->area = 0;
14   this->peri = 0;
15   this->name = nombre;
16   this->color = color;
17   tri.setRadius (float(0.66 * this->lado));
18   tri.setPointCount(3);
19
20 }
21
22 triangulo::~triangulo()
23 {
24
25 }
26
27 void triangulo::info()
28 {
29      cout << "largo"<< this->lado <<"area: "<< this->area << "perimetro: "<<this->
       peri<<endl;
30
```

```
31 }
32
33 void triangulo::c_area()
34 {
35
36   this->area = (sqrt(3)/4) * this->lado * this->lado;
37 }
38
39 void triangulo::perimetro()
40 {
41
42   this->peri = 3 * this->lado;
43 }
44
45 sf::CircleShape triangulo::dibujar(sf::RenderWindow *rW){
46     tri.setFillColor(sf::Color::Green);
47     return tri;
48 }
```

2.3. Main

Donde se ejecutan las instancias de las 3 clases, generando después de compilar 2 ejemplos por clase tanto para la clase circulo como para la clase rectángulo. Además de 3 instancias de triangulo. Sin embargo solo se genera un dibujo de cada clase como demostración.

Listing 8: método main

```
1
2 #include <iostream>
3 #include <SFML/Graphics.hpp>
4
5 #include "../include/figura.h"
6 #include "../include/circulo.h"
7 #include "../include/rectangulo.h"
8 #include "../include/triangulo.h"
9
10
11 using namespace std;
12
13 int main(int argc, char *argv[])
14 {
15
16    sf::RenderWindow *rW = new sf::RenderWindow(sf::VideoMode(720, 480), "lab2");
17     // circulo(double radio, int id, std::string nombre, std::string color)
18     circulo * c1 = new circulo( 2.0, 25, "primer_circulo", "Blue");
19     circulo *c2 = new circulo(120.0, 45, "segundo_circulo", "Green");
20
21
22     rectangulo r1(135.5, 185.78, 20, "rec1", "Pink"    );
23     rectangulo r2(25.2, 32.5, 50, "rec2", "Black");
24
25                 //double lado,  int id, string nombre, string color
26     triangulo t1(150.25,  1, "triangulo1", "Orange");
27     triangulo t2(25,  2, "triangulo2", "Pink");
28     triangulo t3(985.005, 3, "triangulo3", "Black");
```

```
29
30
31
32      r1.c_area();
33      r1.perimetro();
34      !r1;// imprime area y perimetro
35      ~r1;//imprime caracteristicas del objeto
36      r2 = r1;
37      !r2;
38      ~r2;
39
40      c1->c_area();
41      c2->c_area();
42      c2->perimetro();
43      c1->perimetro();
44
45      ~(*c1);
46      !(*c1);
47      ~(*c2);
48      !(*c2);
49      c1 = c2;
50      ~(*c1);
51      ~(*c1);
52      !(*c1);
53
54      t1.c_area();
55      t1.perimetro();
56      t2.c_area();
57      t2.perimetro();
58      ~t1;
59      ~t2;
60      !t2;
61      !t1;
62
63      cout << "número de instancias círculos: " << c1->get_num_circulos() << endl;
64      cout << "número de instancias triángulos: " << t2.get_num_triangulos() <<
        endl;
65      cout << "número de instancias rectángulos: " << r2.get_num_rectangulos() <<
        endl;
66
67
68    //  c2->dibujar(rW1);
69      while (rW->isOpen())
70      {
71        sf::Event event;
72        while (rW->pollEvent(event))
73        {
74            if (event.type == sf::Event::Closed)
75                rW->close();
76        }
77
78        rW->clear();
79        rW->draw(r1.dibujar(rW));
80        rW->draw(c2->dibujar(rW));
81        rW->draw(t1.dibujar(rW));
82        rW->display();
```

```
83      }
84
85
86      delete rW;
87
88      /* IMPORTANTE, debido a la instruccion :
89       *                                          c1=c2
90       * se utiliza un solo delete, ya que al hacer la copia de objeto c2 en c1,
        ambos punteros c1 y c2 apuntaran
91       * a la misma dirección de memoria, por ello se aplica delete unicamente a c1
92       * esto hara que se libere la  memoria asignada
93       * **/
94
95      delete c1;
96
97  }
```

3. Conclusiones

- Los métodos virtuales nos permiten alterar cuales métodos serían llamados en el *runtime*, dado el orden implementado tablas virtuales.
- Implementando una sobrecarga de operadores para los unarios y ! a través de la palabra clave *this* para apuntar a los atributos miembros de si misma.
- Durante la graficación es pertinente entender el funcionamiento de las clases integradas a través de la biblioteca de SFML. Además es pertinente el estudio y selección efectiva de los métodos CircleShape o ConvexShape. para cada clase.
- SFML permite ademas del dibujo de figuras, su manipulación espacial y resulta en una herramienta utilizar a la hora de visualizar los elementos creados e n el código.

4. Anexos

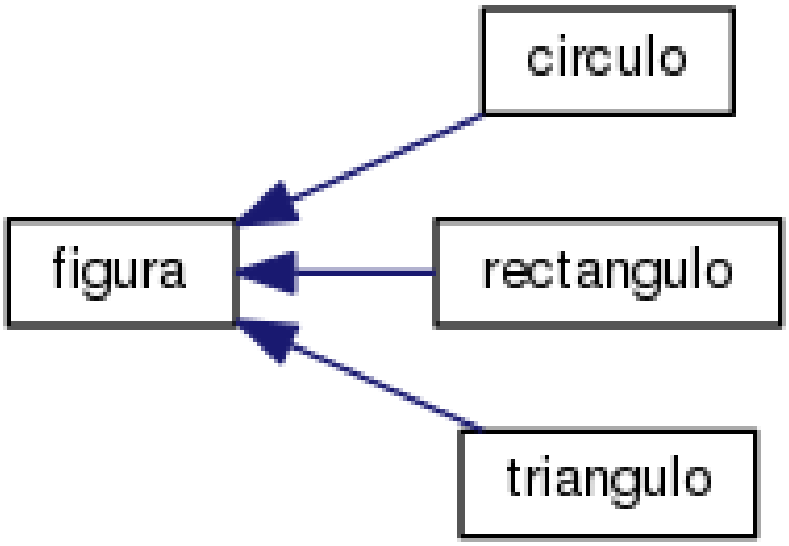

Figura 1: Diagrama de herencia de clases, creado con el doxyfile