

Clase 5 y 6. Polimorfismo

Clases abstractas, vectores y Dynamic_cast

MODELO EDUCATIVO
TEC21

Polimorfismo



Explicación



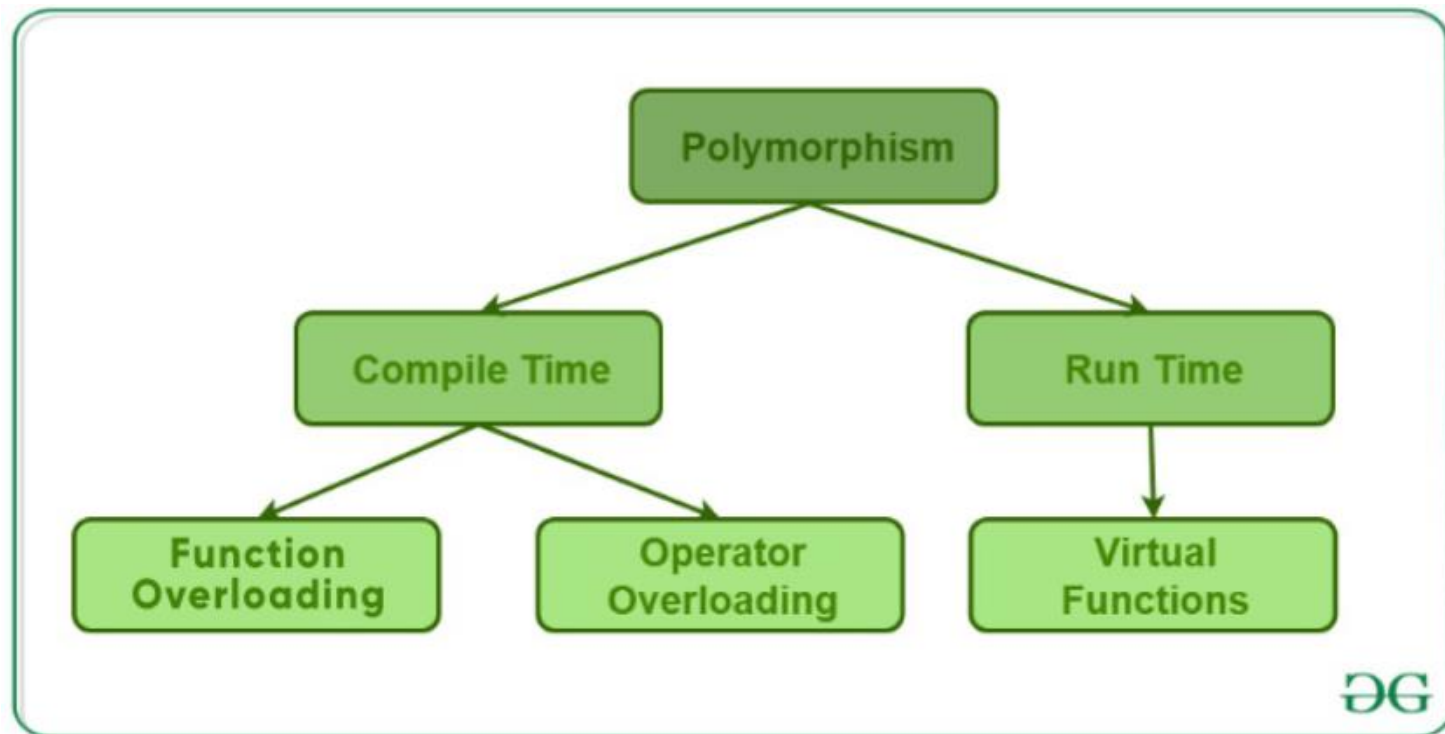
Apuntadores

```
// Es un tipo especial de variable pero que guarda direcciones de memoria
// Se denota con un * y tiene un tipo
int x = 10;
int *ptrInt;
ptrInt=&x;
// despleguemos el valor del apuntador y del valor al cual apunta
cout << ptrInt << " es la direccion de memoria donde está x" << endl;
cout << *ptrInt << " es el valor al cual apunta" << endl;
// podemos reservar un espacio de memoria en tiempo de ejecución
int *ptrInt2 = new int(20);
cout << ptrInt2 << " es la direccion de memoria" << endl;
cout << *ptrInt2 << " es el valor al cual apunta" << endl;
//podemos crear apuntadores a objetos
Shape *ptrFig = &fig;
//o podemos crearlos en tiempo de ejecución
Shape *ptrFig2 = new Shape(2, 3);
/* Pero cuando tenemos apuntadores a objetos, para poder acceder a los métodos
   de esos objetos usamos el operador -> */
cout << ptrFig2->draw() << endl;
```

- Abre tu programa de las figuras y sigue con la maestra la explicación breve de apuntadores.

Polimorfismo – Conceptos Clave

- Es un concepto que le permite a un mismo mensaje ser desplegado en más de una forma.



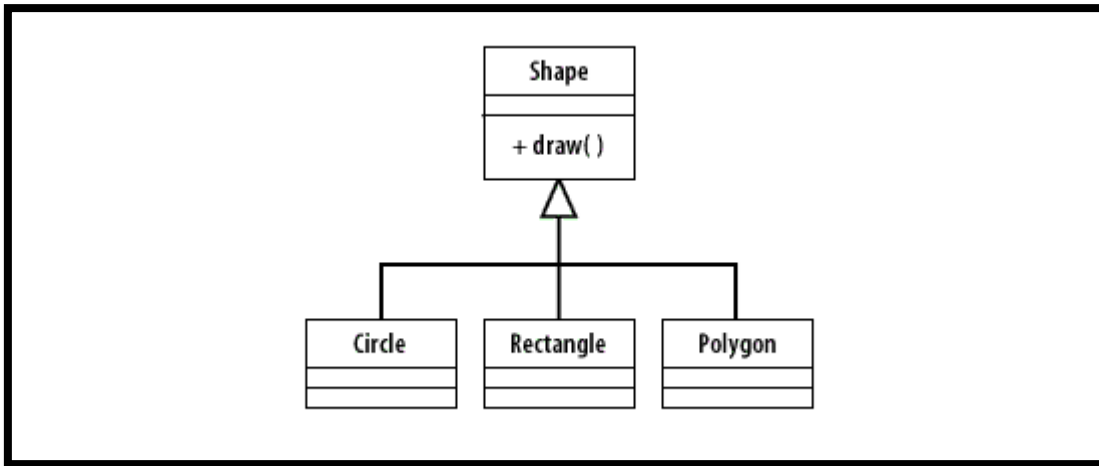
Recordando y aprendiendo...

- Un objeto de la clase derivada puede tratarse como un objeto de la clase base, pero no al revés.
 - `Circle c1(1,1,5);`
 - `Shape fig;`
 - `fig = c1;`
 - Pero ¿qué sucede con fig? ¿Cómo se comporta?
- Con apuntadores también lo podemos hacer...
 - Un apuntador de tipo de la clase base puede referenciar a un objeto de la clase derivada.
 - `ClaseBase *basePtr;`
 - `ClaseDerivada cd;`
 - `basePtr=&cd;`

En polimorfismo en tiempo de ejecución

- Las funciones que se van a sobrescribir les ponemos la palabra reservada **virtual** en su declaración para que puedan actuar polimórficamente con un manejador de la clase base.
- **override** es una palabra reservada que permite asegurarnos que no cambiaremos por “descuido” la firma del método.

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 dinámico

- ¿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 deseáramos 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);

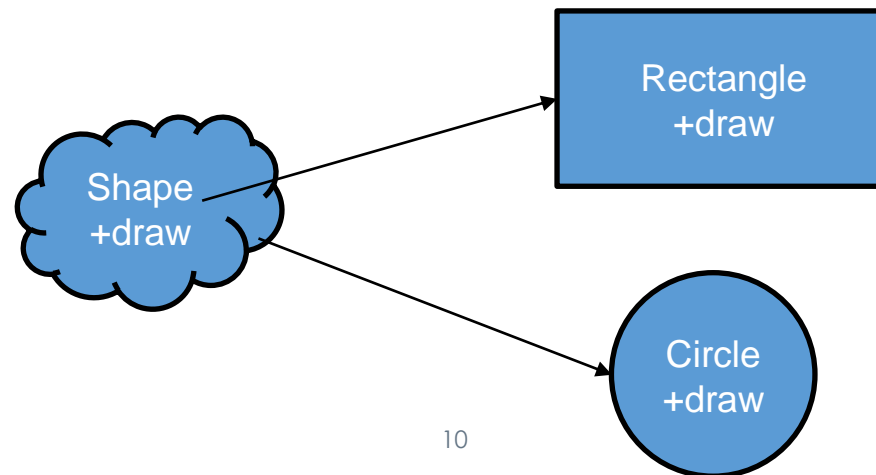
    // y todavía faltan los cuadrados

    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

- ¿Cuál es la solución?
- Crear un arreglo o vector que acepte objetos de tipo Shape (que es el padre) o de sus clases derivadas, si se manda llamar al método draw, se ejecute el que corresponda al objeto que está guardado.



Código de Shape (.h) mejorado

```
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     virtual string draw();
18
19     // los métodos debajo se usarán para imprimir los v
20     // de x y de y , NO ES LA MEJOR SOLUCION
21     // pero se mejorara más adelante
22     int getValueX();
23     int getValueY();
24 };
25
26
27 #endif // SHAPE_H_INCLUDED
28
```

La palabra virtual en un método de la clase padre.

Código de Shape (.cpp) mejorado

```
1  #include "Shape.h"
2
3  Shape::Shape ()
4  {
5      x = 0;
6      y = 0;
7  }
8
9  Shape::Shape(int valX, int valY)
10 {
11     x = valX;
12     y = valY;
13 }
14
15 string Shape::draw()
16 {
17     return "soy una figura" ;
18 }
19
20 int Shape::getValueX()
21 {
22     return x;
23 }
24
25 int Shape::getValueY()
26 {
27     return y;
28 }
29
```

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

```
1  #include "Circle.h"
2
3  Circle::Circle()
4  {
5      r = 0;
6  }
7
8  Circle::Circle(int valX, int valY, int valR):Shape(valX, valY)
9  {
10     r = valR;
11 }
12
13
14 string Circle::draw()
15 {
16     return "soy un circulo" ;
17 }
18
```

Se implementa el método draw en la clase hija.

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)

```
1  #include "Rectangle.h"
2
3
4  Rectangle::Rectangle()
5  {
6      x = 0;
7      y = 0;
8      xl = 0;
9      yl = 0;
10 }
11
12
13 Rectangle::Rectangle(int valX, int valY, int valXl, int valYl):Shape(valX, valY)
14 {
15
16     xl = valXl;
17     yl = valYl;
18 }
19
20 string Rectangle::draw()
21 {
22     return "soy un rectangulo";
23 }
24
```

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 una 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().

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.



Classes Abstractas



Clases Abstractas

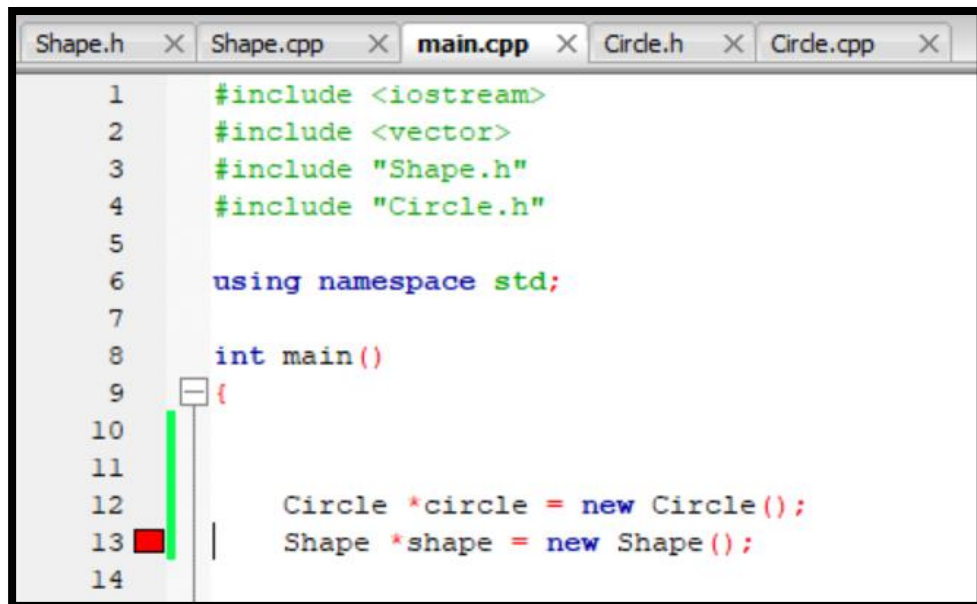
- Clases que no se pueden instanciar
- Generalmente se usan como base para clases hijas
- Para que una clase sea abstracta en C++ debe tener una función virtual pura (virtual y con 0 asignado)

```
class AB {  
public:  
    virtual void f() = 0;  
};
```

Ejemplo:

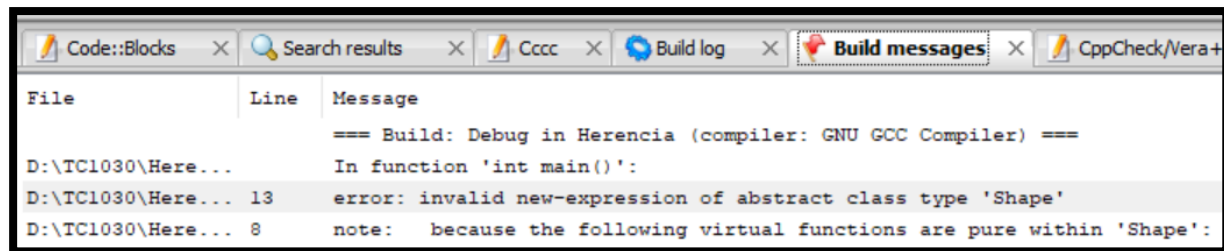
```
Shape.h X Shape.cpp X main.cpp X Cirde.h X Cirde.cpp X
1  #ifndef SHAPE_H_INCLUDED
2  #define SHAPE_H_INCLUDED
3
4  #include <iostream>
5
6  using namespace std;
7
8  class Shape
9  {
10 private:
11     int x;
12     int y;
13
14 public:
15     Shape();
16     Shape(int, int);
17     virtual string draw() = 0;
18
19     // los métodos debajo se usarán para imprimir los valores
20     // de x y de y , NO ES LA MEJOR SOLUCION
21     // pero se mejorara más adelante
22     int getValueX();
23     int getValueY();
24 };
25
26
27 #endif // SHAPE_H_INCLUDED
```

Ejemplo - continuación



```
1  #include <iostream>
2  #include <vector>
3  #include "Shape.h"
4  #include "Circle.h"
5
6  using namespace std;
7
8  int main()
9  {
10
11
12     Circle *circle = new Circle();
13     Shape *shape = new Shape();
14
```

Notar que si se deseara instanciar la clase Shape no sería posible y marcaría un error de compilación



File	Line	Message
=== Build: Debug in Herencia (compiler: GNU GCC Compiler) ===		
D:\TC1030\Here...		In function 'int main()':
D:\TC1030\Here...	13	error: invalid new-expression of abstract class type 'Shape'
D:\TC1030\Here...	8	note: because the following virtual functions are pure within 'Shape':

Otro ejemplo con clase abstracta

600 Capítulo 13 Programación orientada a objetos: polimorfismo

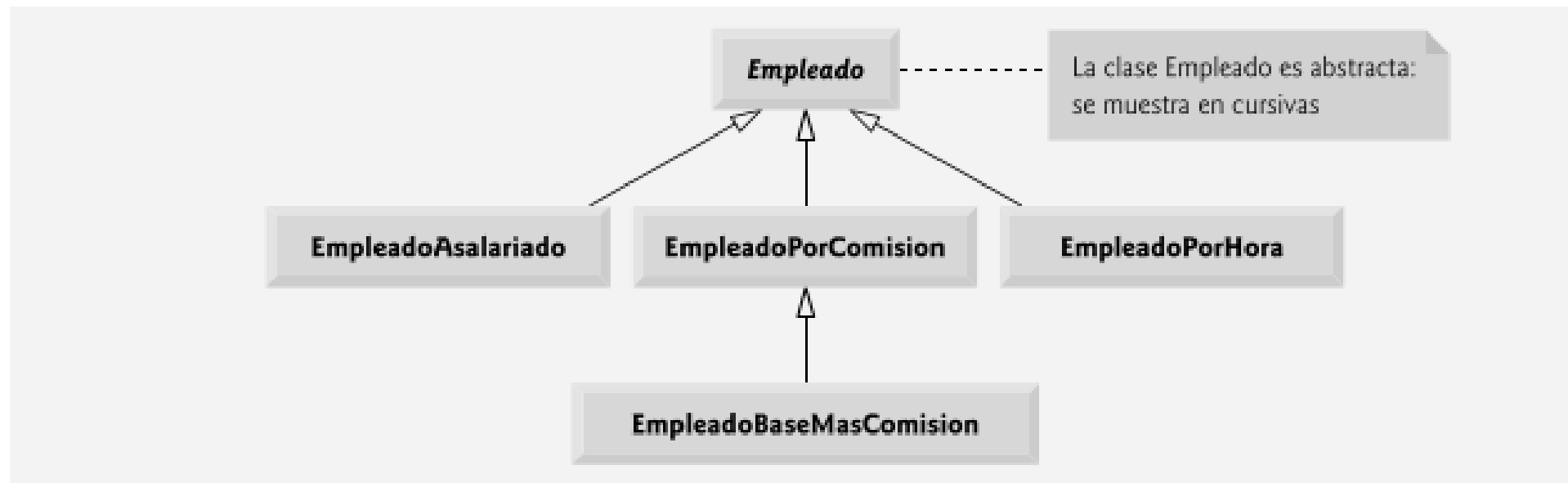


Figura 13.11 | Diagrama de clases de UML de la jerarquía Empleado.

Ejercicio

- Vamos a poner Shape una función virtual Pura. La función `area()`. Su tipo de retorno deberá ser `double`.
- Implementar cómo obtener el área en las clases hijas
- Usemos un vector en lugar de un arreglo en el programa principal
- Verifica el polimorfismo con `area`

Otros elementos a considerar...

- Dynamic Cast
- Uso de referencias y apuntadores
- Uso de vectores

Dynamic cast y downcasting

- Convierte un apuntador de una clase a un apuntador de otra si es posible.
- En un arreglo de apuntadores a clase Shape, yo podría distinguir cuando sean objetos Circle y aplicar algún método que sólo pertenece a Circle.

```
if(Circle* c=dynamic_cast<Circle *>(shapesPtr[i])){  
    c->otra();  
}
```

- DownCasting

Vector

- Contenedor lineal (como una lista). Se redimensionan automáticamente cuando se inserta o borra un elemento del arreglo.
- Tiene métodos relacionados con la capacidad por ejemplo `size()` para saber el número de elementos que tiene.
- Métodos para acceder los elementos por ejemplo `at` o los operadores `[]`
- Métodos para modificar como `push_back` para agregar un elemento o `insert`.



Quiz



Quiz: Polimorfismo

- Realizar Quiz del tema: Polimorfismo en Canvas
- Contiene 5 preguntas
- Tendrá 2 intentos



Dudas y Ejercicios



Cierre y avisos

Actividades por realizar:

- Ejercicio de polimorfismo y clases abstractas.
- Autoestudio de sobrecarga de operadores.
- No olviden entregar la primera parte de la situación problema.