

POO: Paradigma de programación de 1980



ObjetoPila . push (dato)

Mensaje

↳ Orden que se da a un obj. para que cambie de estado o realice una acción.

Clase ≡ Tipo de dato

Objeto = Instancia de una clase

→ Posee:

- **Atributos / Datos miembro**
Variables propias de la clase que determinan el estado de un objeto.
- **Métodos / Funciones miembro**
Elementos que definen la funcionalidad de la clase.

{ Encapsulamiento }:

- Permite ocultar los atributos de un objeto
- Si se quiere modificar los atributos se tiene que hacer mediante los métodos.
- Actualizar la clase por una versión mejor no traería ninguna complicación, nosotros modificariamos los métodos, el usuario vería todo igual

Herencia

- Se puede crear una clase a partir de otra, puede obtener todos elem. de otra clase, sin tener que volverlos a escribir incluso sin tener el código fuente.
- Se puede modificar el comportamiento de algunos métodos y agregar otros.

{ Polimorfismo }:

- Propiedad: Una misma acción aplicada a diferentes clases puede producir acciones distintas.

"Instanciar" → Crear un objeto

```
class ClaseA objetoA;
```

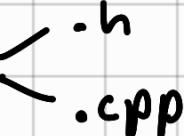
por otro lado:

```
class ClaseA *ptr;
```

ptr es un puntero a un objeto / o un de objetos de clase A.

Se instanciará cuando:

```
ptr = new class ClaseA
```

Cuando creo una clase autom. se crea  .h .cpp

en el .h:

```
class NombreClase {
```

public:

- Aquí van los encabezados de los métodos

private:

- Aquí van todos los atributos, en private pq deben estar ocultos.

```
};
```

.CPP: Implementación de los métodos

Métodos set() y get():

- Debido al encapsulamiento tendré que acceder a los atributos de un objeto mediante métodos.

set(valor) → Para asignar un valor.

get() → Para obtener el valor actual de un atributo.

Objetos en la memoria:

class Rectangulo R1, R2;

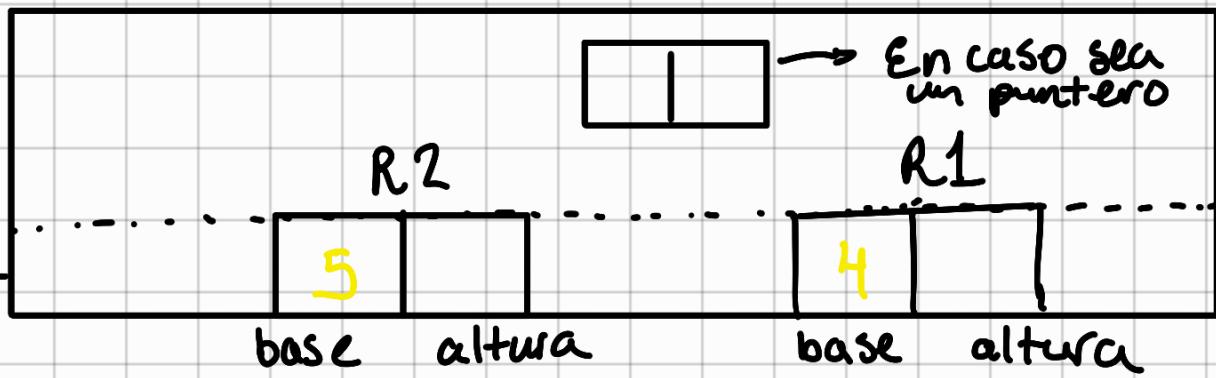
R1.setBase(4);
R2.setBase(5);

CS

Aquí va la implementación de los métodos de la clase

```
void setBase(double b) {  
    base = b  
}
```

DS



Los datos de 1 objeto que se colocan en el stack son sus atributos.

Para que todos los objetos puedan usar el mismo código de un método y este sepa qué atributo/s de qué objeto modificar internamente trabaja con un puntero al objeto

↳ Esto se realiza automáticamente

R1.setBase(3)
↳ Internamente:

R1.setBase(3, &R1)

Pasa la DM del objeto

De donde inician sus atributos en la memoria

void setBase(double b){
 base = b;
}

↳ Internamente:

void setBase(double b, class Rectangulo *this){
 this -> base = b;
}

Constructor:

- Se ejecuta automáticamente cuando se instancia una clase.
- Debe tener el mismo nombre que la clase
- Puede tener argumentos pero no devolver valores.
- Se puede sobre cargar.

Tipos:

- Constructor por defecto:
No posee argumentos.
- Constructor:
Posee argumentos.
- Constructor copia:
Inicializa el objeto a través de otro objeto.

Destructor:

Sirve para liberar los atributos dinámicos que puede contener el obj.

- Se ejecuta automáticamente cuando se sale del ámbito del objeto (de la función donde se instanció la clase).
- Mismo nombre que la clase pero con '~' al inicio.
- Sin argumentos ni valor de retorno
- No se puede sobre cargar

Por eficiencia los objetos se pasan siempre por referencia, cuando no voy a modificar un objeto lo paso como constante, pero al hacer esto no puedo usar sus métodos, por lo que deberá colocarse const después del encabezado (en la imp. tmb) para permitir usar los métodos deseados.

double getbase() const;

En caso tenga un puntero pasa lo siguiente:

class Rectangulo *ptr;

↑
No es un
objeto

1er caso ptr = new class Rectangulo;

↓
Se separa
espacio para
1 objeto

Se ejecuta
el constructor

2ndo
caso

ptr = new class Rectangulo [n];

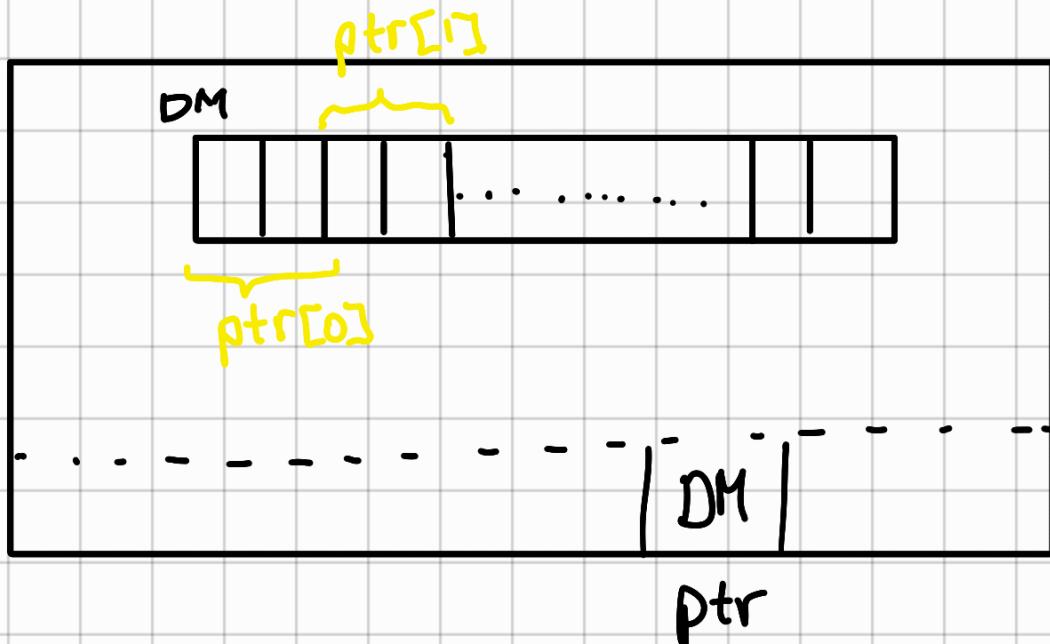
↓
Se crea un arreglo
de n objetos.

Cada objeto usa el
constructor por defecto

automáticamente

El destructor se ejecuta una vez se haya salido del ámbito de la función, en caso se haya creado objetos estáticos.

En el caso de tener un arreglo de objetos el destructor no se ejecuta automáticamente



Si ya no necesito usar el arreglo debo liberar todos sus elementos mediante:

`delete [] ptr;`

Importante:-

Los objetos deben ser pasados por referencia no solo por eficiencia... GAAA

`c2.copia(c1)`

Si en la f() "copia" c1 fueran pasados por valor, al llamarla la f() se crearía un nuevo objeto (i.e ob1) donde en este se copiaría los atributos de c1, al terminar la f() se ejecutaría el destructor

para obj1, en caso apunten a los mismas DM' asignadas dinámicamente estas ya no podrán ser utilizadas debido al destructor.



void Rectangulo:: copia(**const** class Rectangulo& a){

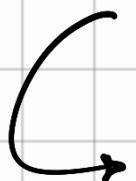


void Rectangulo:: copia(class Rectangulo a){

Sobrecarga de operadores:

Se implementará como si fuera un método.

a + b;



a.**+**(b);

