



zarlos

www.wuolah.com/student/zarlos

3491

apuntes-clase-informatica-industrial-muy-completos.pdf

apuntes informatica (muy completos)



3º Informática Industrial I



Grado en Ingeniería Electrónica Industrial y Automática



**Escuela Politécnica Superior
Universidad Carlos III de Madrid**



¿Estudiar con una cachimba? 😊抽烟

Siiiiiiiiii 😊

😊 Aprovecha este descuentazo en toda la web, solo para los estudiantes madrileños. CÓDIGO: UNIBENGALA





montero espinosa
Academia Universitaria



academia universitaria



En nuestra academia TE OFRECEMOS:

LOS MEJORES PROFESORES



Son los protagonistas: Calidad, preparación y motivación es lo que define a nuestro equipo. Ingenier@s y licenciad@s se ciñen estrictamente a los contenidos de la Uni, y lo mejor, van al grano.

CERCA DE TI



Nos ubicamos cerca de las Universidades, para que no pierdas tiempo.

QUE EL RITMO NO PARE



Todas nuestras clases se imparten tanto presenciales como online.

A NUESTROS APUNTES, LOS QUIERE TODO EL MUNDO



Los más completos, claros y ordenados apuntes de teoría. También tenemos la mayor colección de ejercicios de examen resueltos.

Montero Espinosa, patrocinador oficial de tus aprobados.

¡Infórmate! 689 71 67 71

Primeras clases

1º) Presentación personal

2º) Entregar que voy a ir haciendo durante cursos

3º) Hacer de:

- Ir estudiando desde el principio
- Horario de tutorías y estudiar antes de preguntar
- Rítulos de las clases ↗-lecto
- pregunta el nivel
- Sigue un equipo.

4º) POC ↗ Dictado a mano en pluma

5º) VUL

6º) Powerpoints

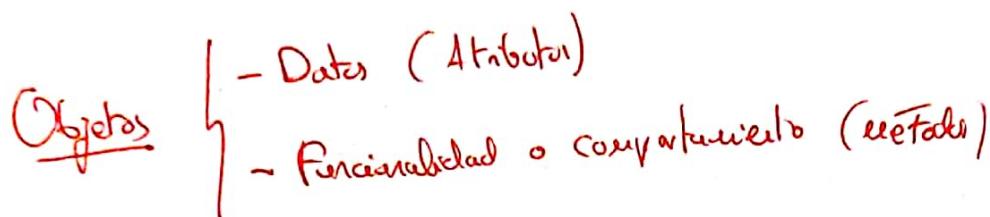


WUOLAH

Scanned by CamScanner

Paradigmas de programación:

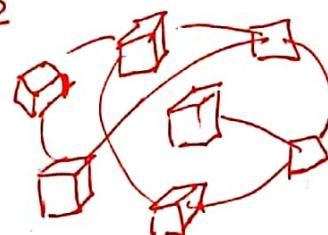
- Estructurado (secuencial)
- Programación Orientada a Objetos (POO)
- Lógica (ej. prolog)
- Listas (lisp)



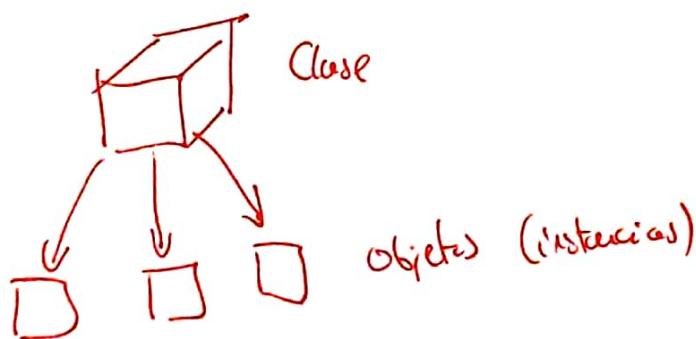
Estructurado



POO



Clase: plantilla para crear objetos. Agrupa objetos con unos atributos y comportamientos comunes. Ej: plan de una casa o de un traje.



Conceptos

- * Clases
- * Objetos (instancias)
- * Atributos
- * Métodos

4) Conceptos clave del paradigma de POO

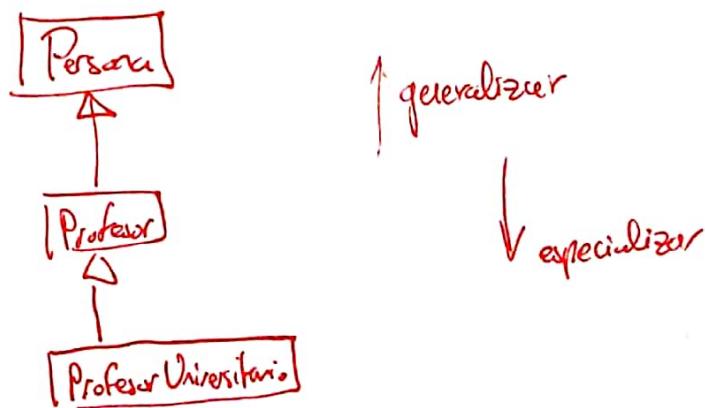
→ **ABSTRACCIÓN:** formalizar un problema pensando en clases, objetos, métodos, atributos. MUNDO \Rightarrow MODELO

→ **ENCAPSULAMIENTO:** seguridad para proteger la información y regular el acceso tanto a operaciones como a los atributos: public, private, protected, package - visibilidad.

* **POLIMORFISMO:** se manda mensajes a un comportamiento (ej: rotarán) que puede implementarse y llevarse a cabo de formas diferentes. Permite iteraciones coherentes y entradas a los objetos sin preocuparnos con cada tipo de objeto lo que se lleva a cabo.

- baila ("Fé") ;
- baila ("Juan") ; . . .

* **HERENCIA:** una clase que hereda de otra clase (la extiende). Y hereda sus atributos y sus comportamientos (métodos).



DE TODOS LOS PERMISOS GRATUITAS

CLASES TEÓRICAS ONLINE

autoescuelalara.com
647 63 53 39



ADEMÁS TE FINANCIAMOS
TU CARNET POR 20€
+ 4 clases de circulación

UML (Sirve para modelar sistemas reales: no sólo para programarlos).

Diagramas de clases

VISIBILIDAD de atributos y métodos

- privado
- + público
- # protegido
- ~ paquete

CONECTORES

→ Herencia (generalización) "a un..."

--> Realización "implementa..."

— Asociación "esta asociado a..." "pertenece a..."

Agregación: es una asociación "débil". Existe y puede existir otros objetos por separado. La cardinalidad vale $n \neq 0$.

Cooperación: asociación "fuerte". Una parte no puede existir fuera del todo.

---> Dependencia (uso): "usa a" → el cambio en la parte de la que depende la parte afecta su funcionamiento.

Multiplicidad

- Restricciones numéricas (cardinalidad) entre las dependencias.

0 .. *

1 .. *

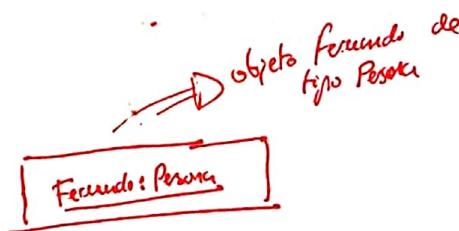
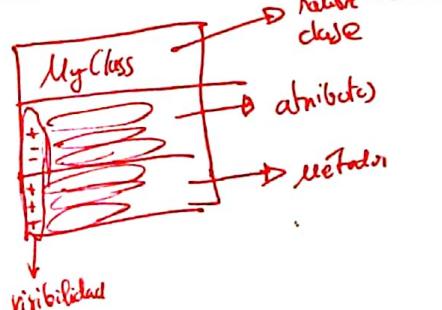
M .. N

3

1

:

CONTENIDOS



(2)



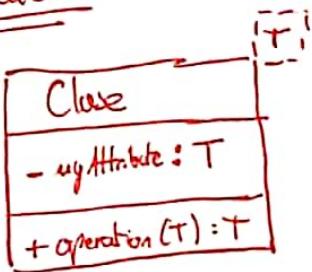
Clases y/o atributos abstractos

Se ponen en cursiva

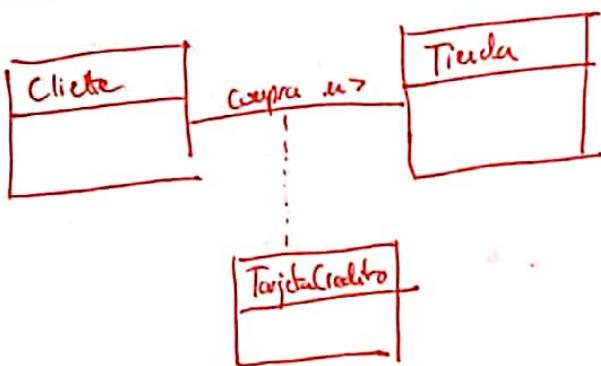
Atributos y/o métodos estáticos

Se subrayan

Template



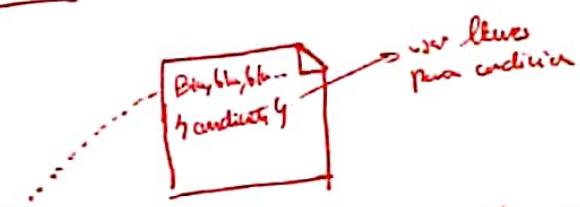
Clase asociada



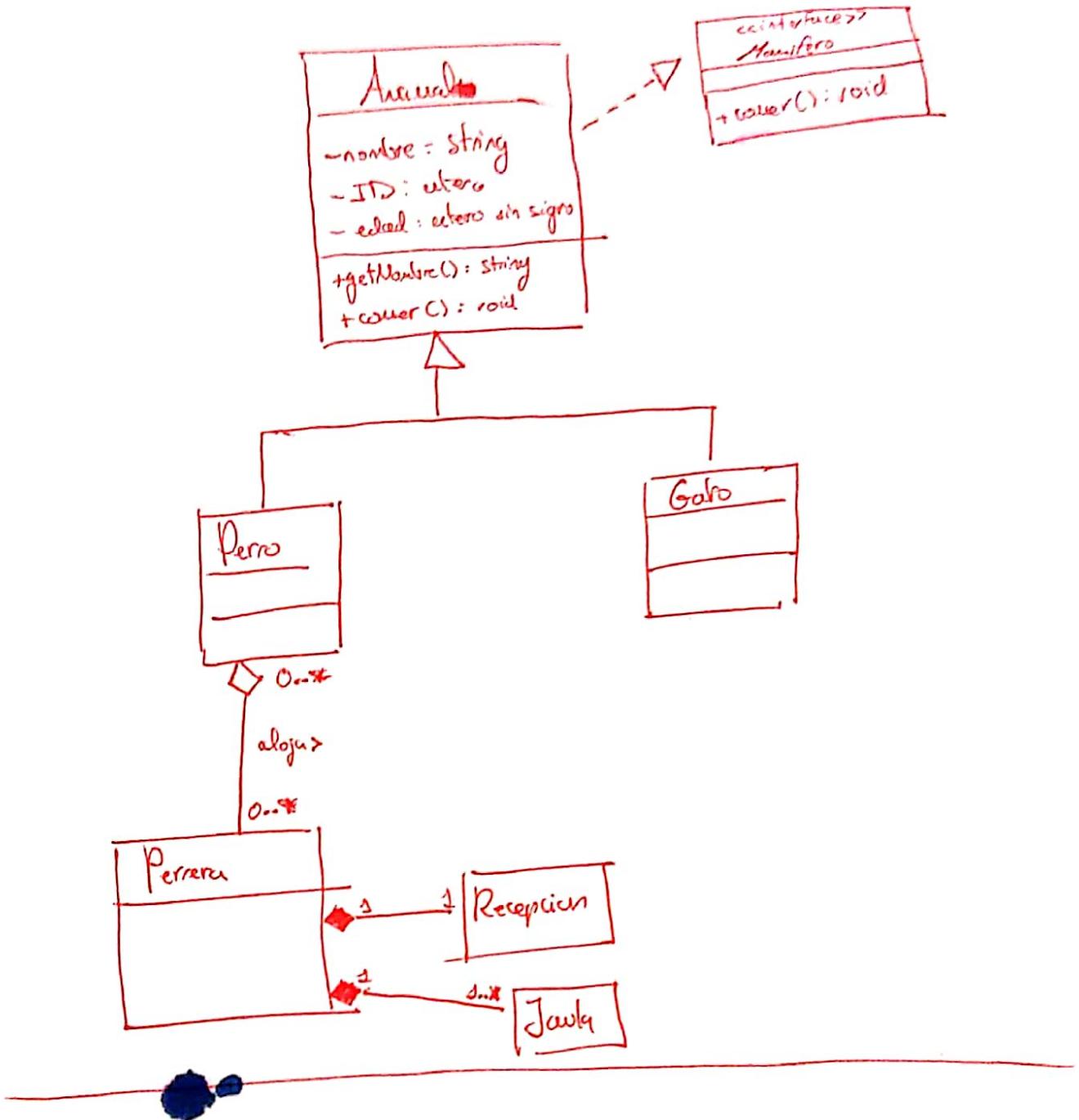
Interface



Notas



Se puede conectar a cualquier clase del diagrama que se desee adicionar



(3)

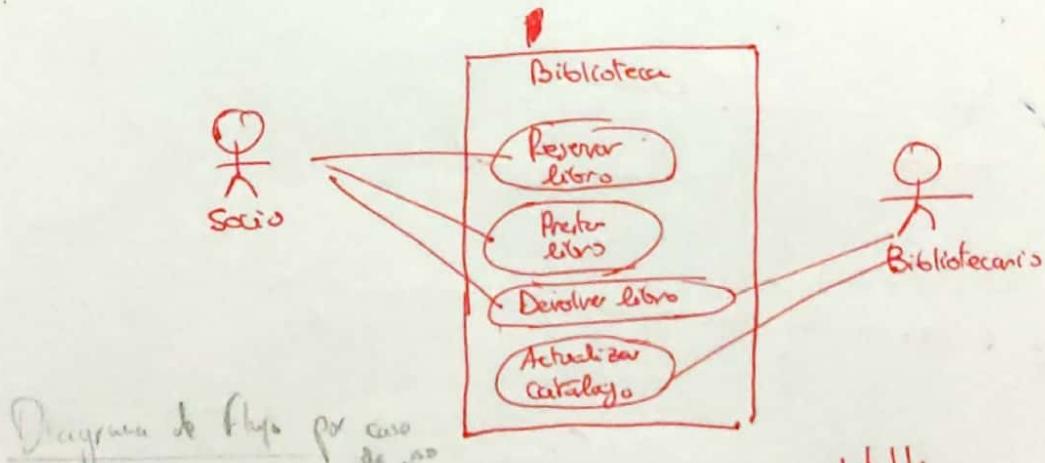
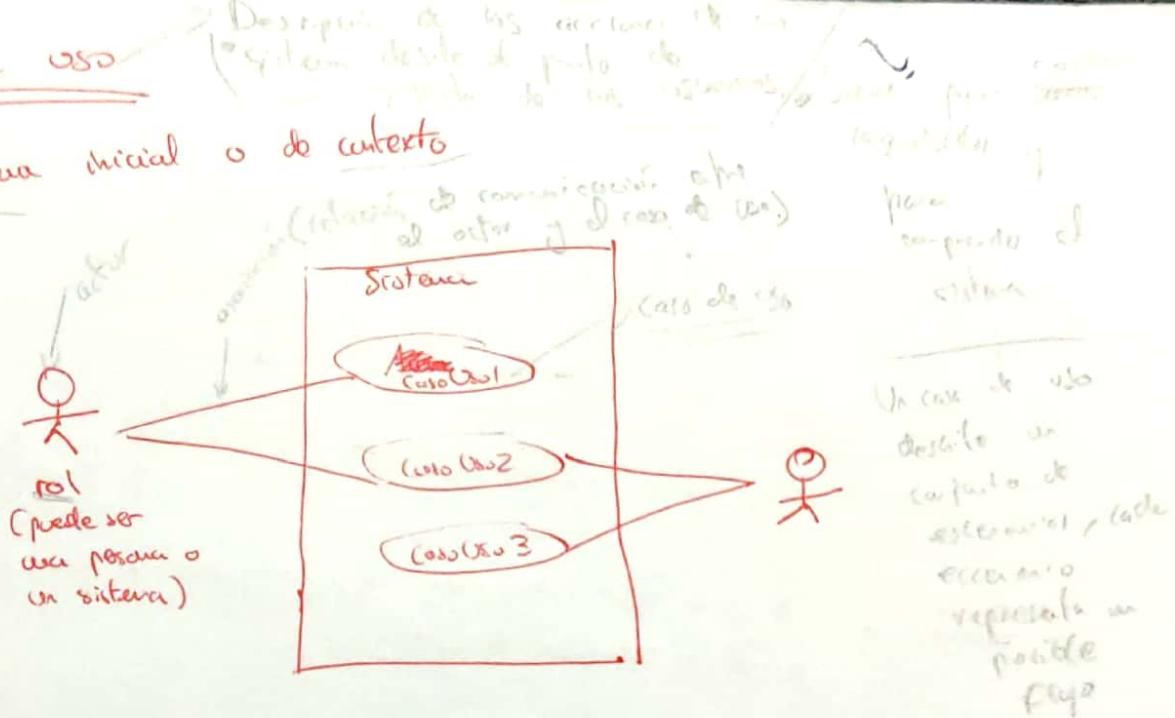
WUOLAH

Scanned by CamScanner

Casos de uso

Diagrama inicial o de contexto

Sirve para capturar los requisitos funcionales del sistema, y reúne el vocabulario del dominio.



Caso de uso	Reservar libro
Actores	Socio
Resumen	- - - - -
Precondiciones	El socio no tiene ninguna reserva pendiente
Postcondiciones	El socio tiene una nueva reserva y el libro tiene una nueva reserva a partir de una fecha
Flujo de Eventos (Escenario)	
Actuar	Sistema
1. - - -	
2. - - -	
3. - - -	
4. - - -	
5. - - -	

DE TODOS LOS PERMISOS GRATUITAS

CLASES TEÓRICAS ONLINE

autoescuelalara.com
647 63 53 39



TU CARNET POR 20€
ADEMÁS TE FINANCIAMOS
+4 clases de circulación

Relaciones entre los casos de uso

• inclusión : un caso de uso necesita del apoyo de otro para llevarse a cabo.

Bibliotecario

Dar de baja a socio

Buscar socio

Modificar datos socio

Confortable
común
reutilizable

• extensión : a veces (con determinadas condiciones, pero no siempre) un caso de uso necesita de otro.

Ajente

Evaluar solicitud de crédito

aceptar
rechazar

solicitar ref.
adicional al cliente

• Herencia : todo en los casos de uso viene en los actores

Ajente
Ajente Inmobiliario
Ajente Bursátil (broker)

Enviar solicitud de crédito

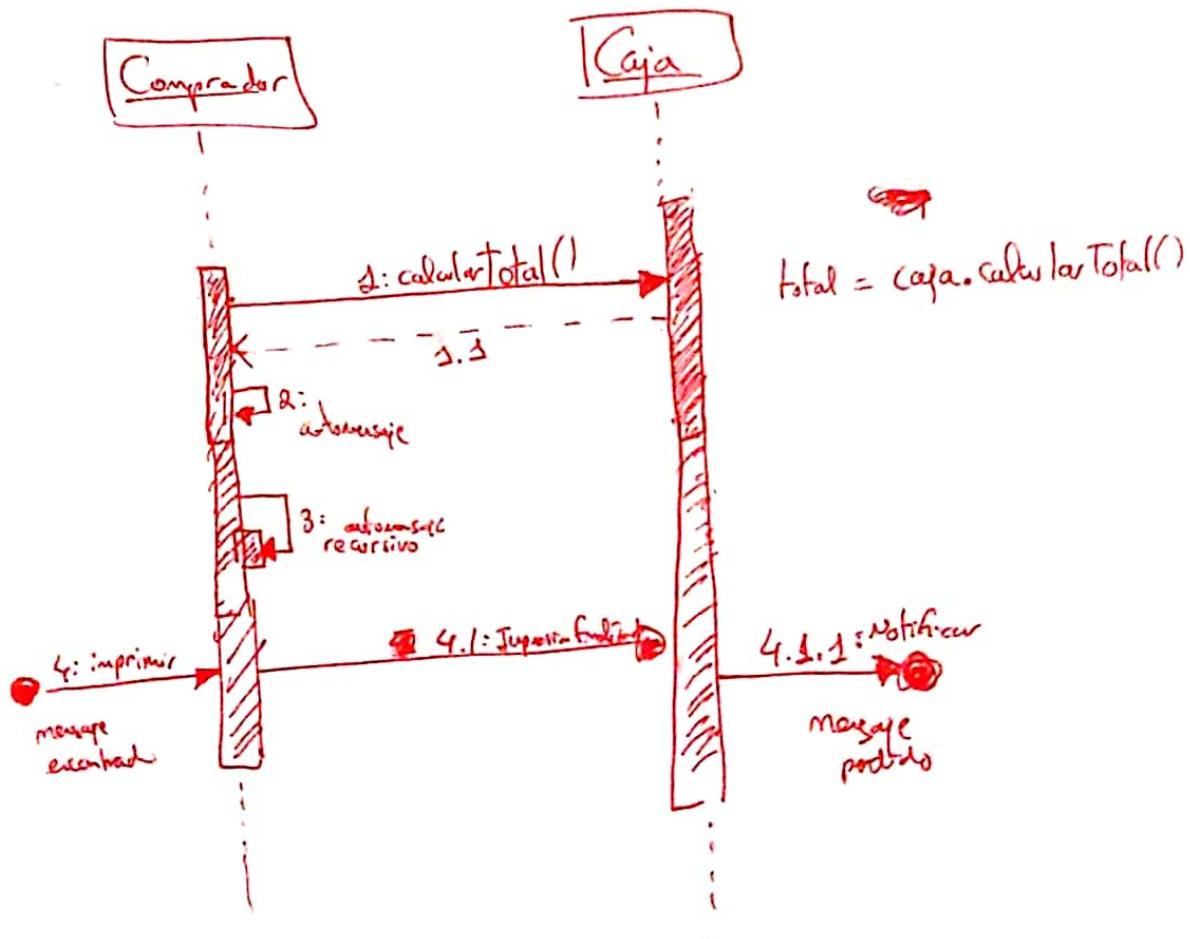
Enviar solicitud de crédito personal

Enviar solicitud de crédito especial

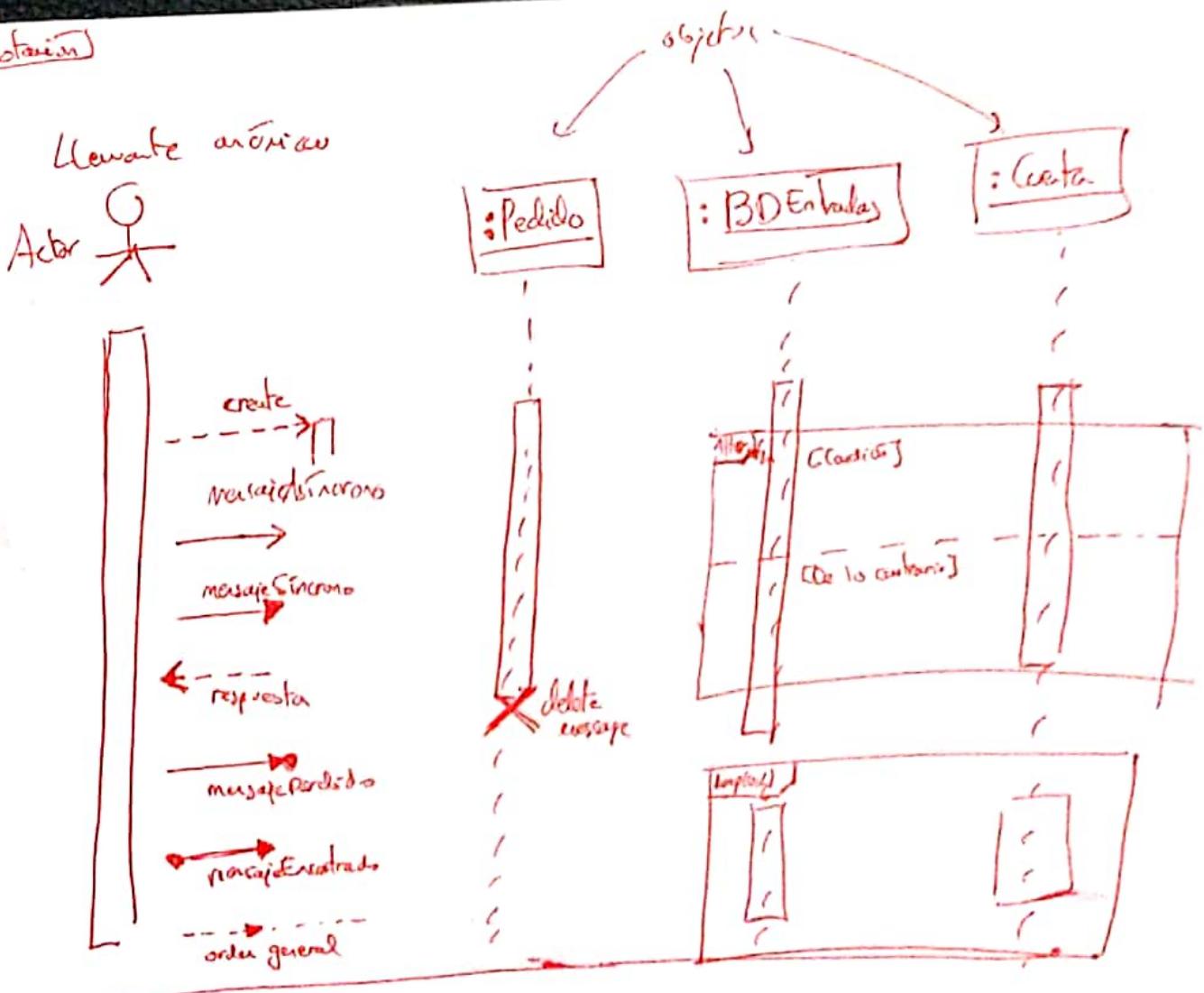


Diagrama de Secuencia

- Comunicación entre objetos, los cuales se envían mensajes entre ellos.
- Indica la secuencia (el orden) en el que se suceden este tipo de mensajes.
- Se lee de arriba hacia abajo (flecha es el eje vertical).
- Los objetos/participantes se muestran como rectángulos
- Los objetos/participantes se muestran como rectángulos
- Cada objeto tiene una línea de vida (línea vertical)

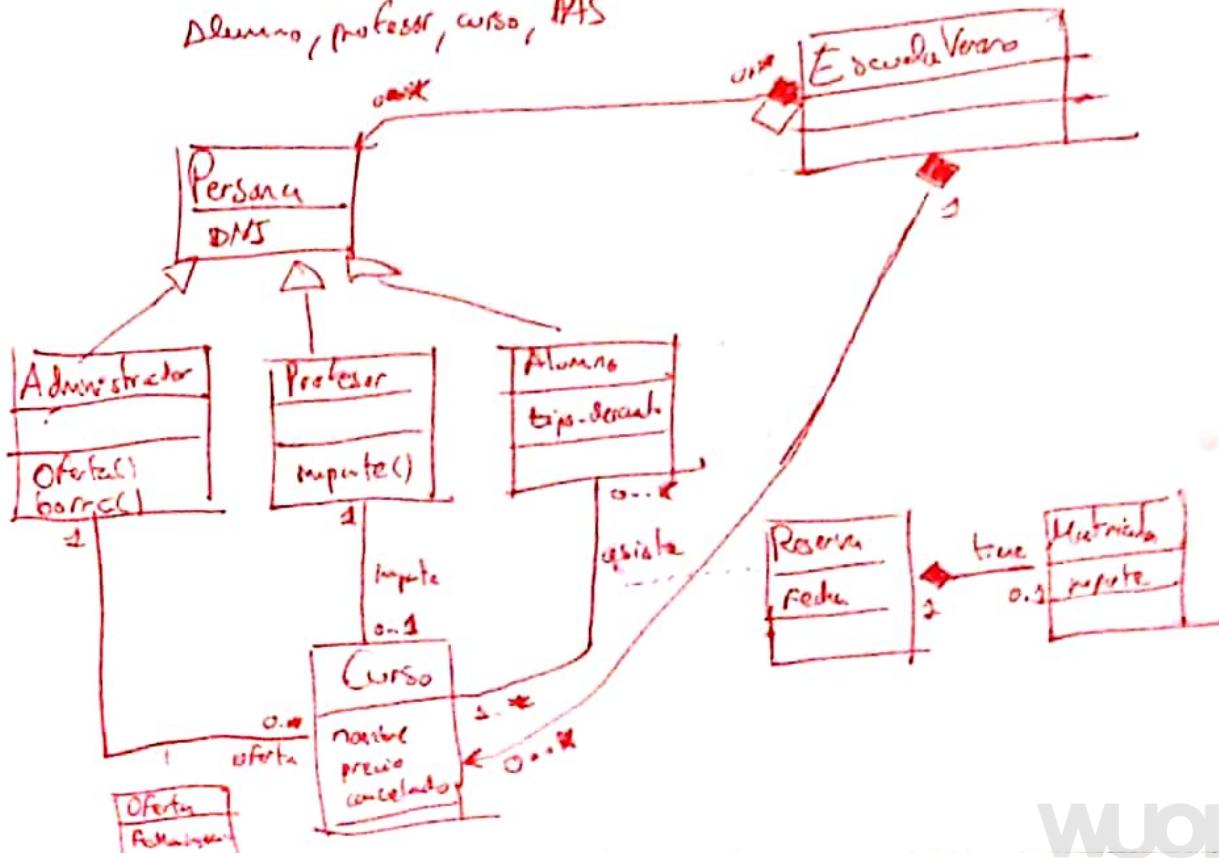


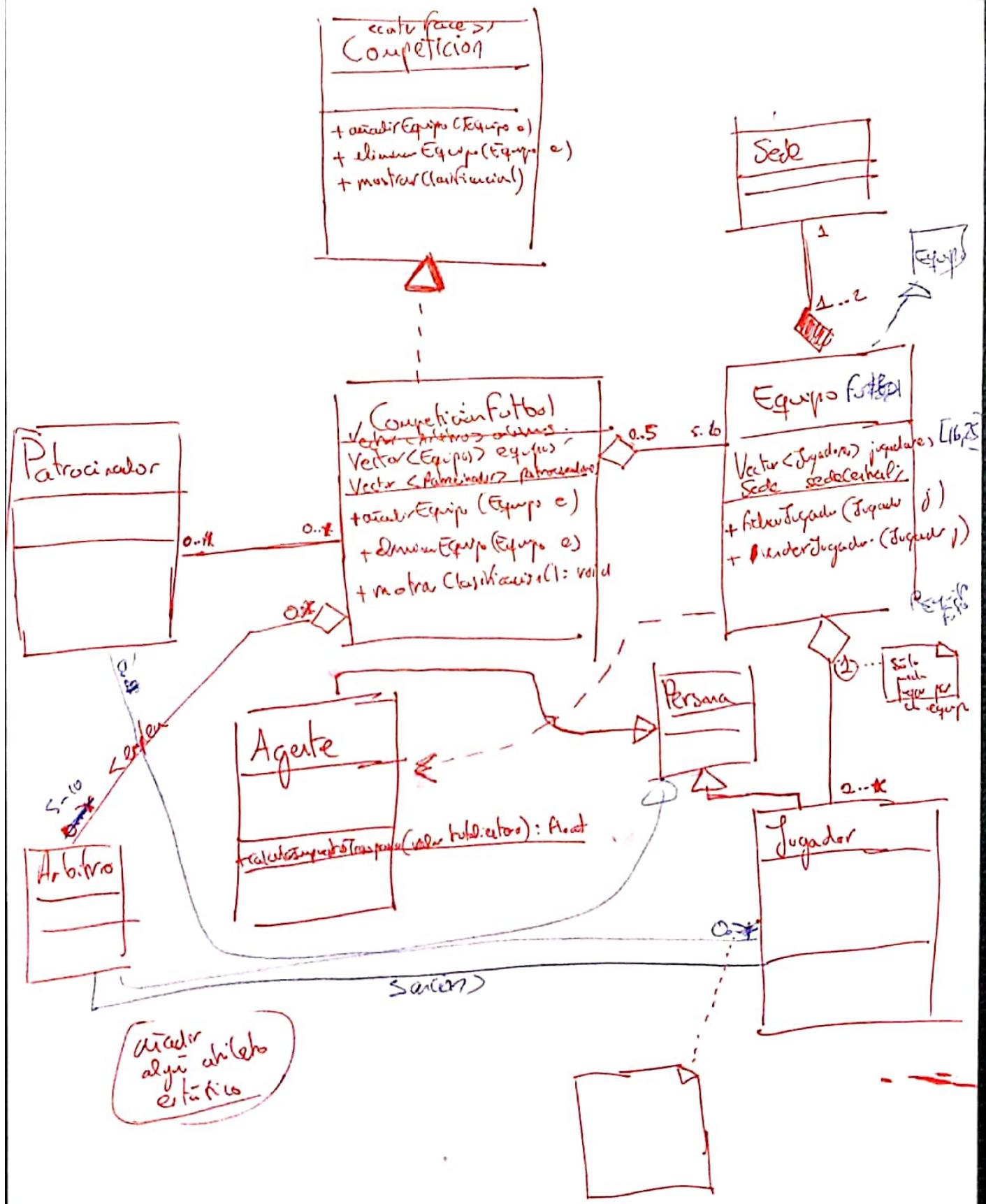
Notebook



Ej. Diagrama de Clase 2

Dilemma, professor, curso, PAS





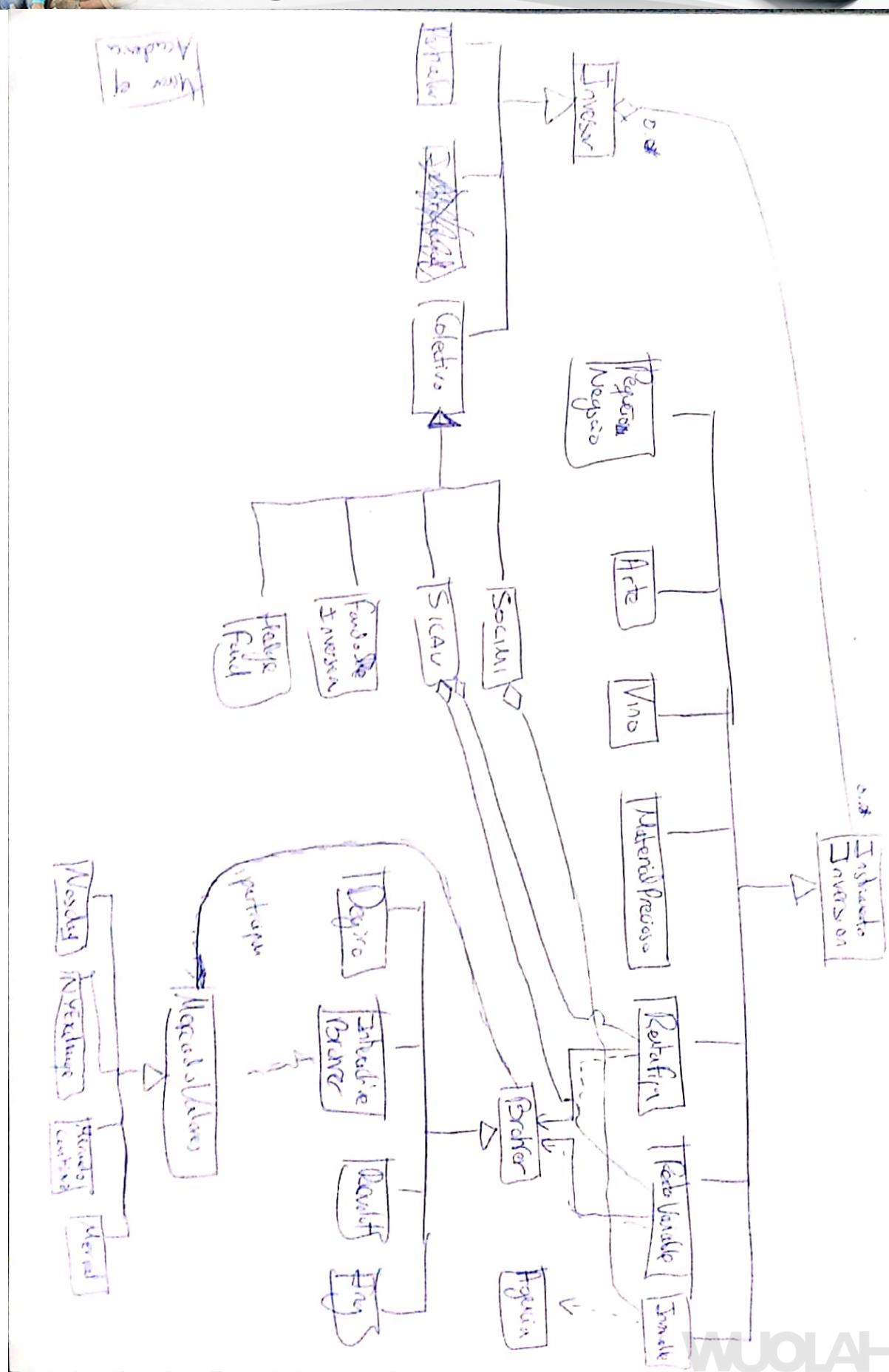


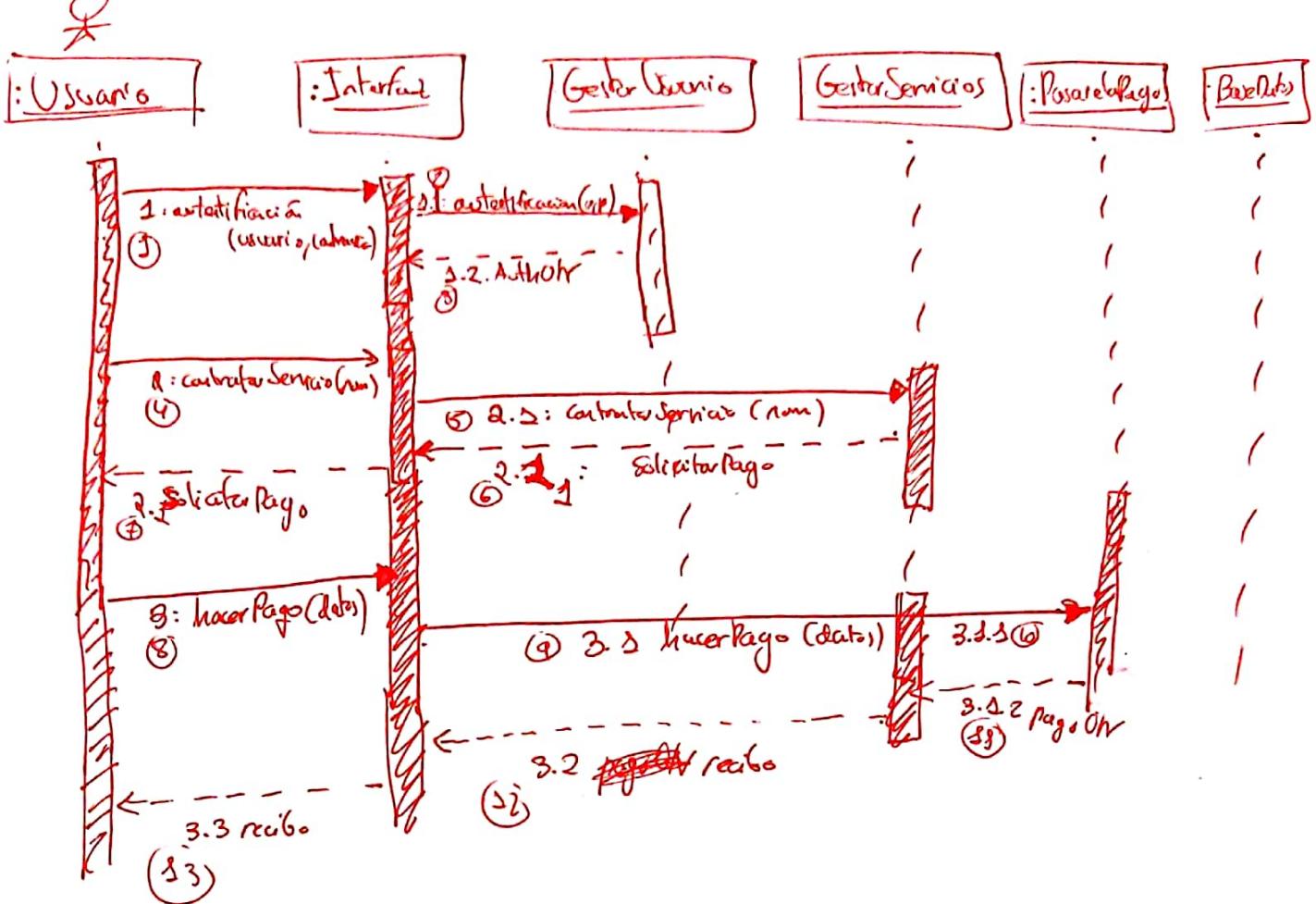
Pack permiso B + 3 clases

Código Promoción: wuolah-2020

39,00€

33,15€





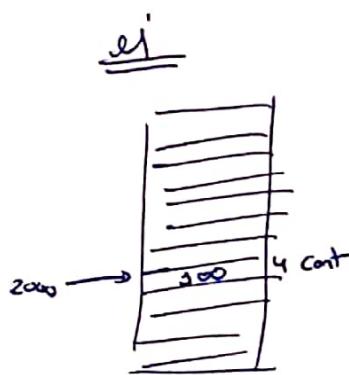
puntero = dirección de memoria de una variable

variable → dirección de memoria de "variable"
↓
"la dirección de"
se la posicio de memoria 2000
Cont = 2000;

m = &cont; (m tendrá el valor 2000)

* ~~Variable~~ → "en la dirección de"
puntero ⇒ devuelve el valor de la variable ubicada en
la dirección que se especifica.

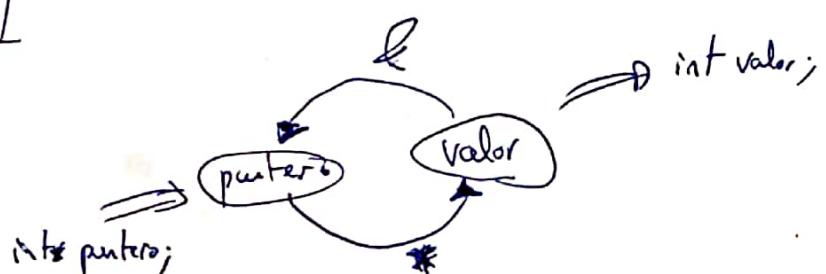
q = *m (q tendrá el valor de 200)



int cont = 200;
lectura mundo
int puntero = &cont; (puntero vale 2000)
lectura puntero
valor = *puntero; (valor vale 200).

para declarar
variables
de tipo
puntero añadir *:

int puntero*; } equivalente
int *puntero; } mezcla puntero
int) *punteros
int x, y, z;
} tipo base del puntero (tipo de dato del valor al que apunta)



$\&$ \Rightarrow tb significa un "y" lógico

$\&f \Rightarrow$ significa & lógico

$*$ \Rightarrow sirve para multiplicar

- declarar variable de tipo puntero

- obtener el valor al que apunta el puntero

$\&$ \Rightarrow - ~~and~~ lógico (op. binaria)

- obtener ~~despues~~ la dirección de memoria de una variable

void main(void)

int destino, fuente;

int* m;

fuente = 30; (30)

m = &fuente; (1865476)

destino = *m; (30)

4

• \Rightarrow accede a un elemento de la estructura

$\rightarrow \Rightarrow$ accede a un elemento de la estructura aplicando sobre el puntero de dicha est.

$\rightarrow \Rightarrow$ equivale (\ast ptEstructura)

```

struct empleado {
    char nombre[80];
    int edad;
    float sueldo;
} emp; // variable para usar la estructura

```

struct empleado* punteroEmpleado = &emp;

empleado sueldo <--> equivalente punteroEmpleado → sueldo;

- existen punteros a punteros (int* & dirección de tipo)
- existen punteros a funciones

LLAMADA A FUNCIONES

- puro por valor (Se copian los valores de arg. a parámetro. Los cambios en el parámetro no afectan al argumento).
- puro por ref. + modificar
 - ↳ se copia la dirección del arg. en el parámetro. Los cambios en el parámetro afectan al argumento.

• Se logra usando punteros

void inter(int* x, int y) {
 ======
}

```

void main() {
    int i, j;
    i = 50, j = 20;
    inter(&i, &j);
}

```

Las funciones también pueden devolver variables de tipo puntero

~~estructura~~

int* función()

```
≡ int a = 3;  
int* puntero;  
puntero = &a;  
return puntero;
```

← Reservar a depender de
existir

↳

Funciones con número de parámetros variables (...)

ESTRUCTURAS

struct persona {

```
char nombre[50];  
char calle[30];
```

} define la estructura (similar a definir una clase)

↳

struct persona variablePersona; //declara la variable (crea si fueren la instancia de una clase);
↳ eg. persona variablePersona; (Person en persona)

también se puede:

struct {

```
char nombre[50];  
char calle[30];
```

variablePersona;

} si sólo se
va a usar
una variable

struct persona {

```
char nombre[50];  
char calle[30];
```

variablePersona;

persona otraPersona;

- Se puede omitir o bien la etiqueta de la estructura
- o bien las variables de la estructura, pero no ambas

(al definir la estructura)

- Se puede declarar los punteros a estructuras

- C++ por defecto los atributos de la clase son ~~estáticos~~ privados

- class Stack {
 int top;
 int str[20];

public:

void init();
int pop();

}

Stack myStack;

en el constructor se declara y es activa (puede llamar al constructor en tiempo de ejecución)

intancia la clase

creación del objeto

equivalente
Stack* myStack = new Stack();

implementar funciones

void Stack::init() { } // operador de relocación de ámbito

puede ser en otro fichero

myStack.init(); // llamada a un método.

constructores y destructores

- Se llama igual q la clase
- no devolverá nada (no hace falta poner void)

- Sirve para inicializar un objeto

- se invoca cuando se crea el objeto

- cuando se destruye un objeto (acaba su ámbito) o el programa,
se llama a su destructor (si es que tiene).

- El destructor se llama igual q la clase, pero antecedido por

~
- Tampoco devolverá nada

(3)

WUOLAH

Scanned by CamScanner

- En C++ la única diferencia entre class y struct es que por defecto en class son todos los miembros privados y en struct son públicos. struct tb puede tener métodos.
- "Union" es similar pero no puede tener herencia ni métodos abstractos (virtual)

Constructores parametrizados

~~Stack myStack(2,3);~~ → Se llama al constructor
 A Stack* myStack = new Stack(2,3);
 B Stack myStack = Stack(2,3);

Static → sólo declarativa en C++ (variable global o perteneciente a la clase)
 static float PI;
 int a;
 b;
 float Matematica::PI;

Los métodos estáticos sólo sirven para trabajar con atributos estáticos. No tiene que duplicarse.

DE TODOS LOS PERMISOS GRATUITAS

CLASES
TEÓRICAS
ONLINE

autoescuelalara.com
647 63 53 39



**TU
CARNET
POR 20€**
ADEMÁS TE
FINANCIAMOS
+4 clases de circulación

- iostream.h \Rightarrow C++ $\left\{ \begin{array}{l} \text{cout} \\ \text{cin} \end{array} \right.$
- stdio.h \Rightarrow C $\left\{ \begin{array}{l} \text{printf} \\ \text{scanf} \end{array} \right.$
- C++ puede tener métodos sin parámetros en necesidad de poner void en C era necesario.
- e.g.: public void main () \Rightarrow ~~void~~ void main (void)
- C++
- int a;
cin >> a;
- C
- int a
scanf ("%d");
- C++
- Si el programa (el main) accede correctamente se suele devolver 0, si hay algún error típicamente un valor negativo.
- C es incorrecto:
- ```
void función () {
 int i, j;
 for (i = 0; i < 6; i++) {
 j = i + 5;
 int K;
 K = j * 2; // error compilación C. OK en C++
```
- En C++ puedes declarar cada variable cuando la necesites, no necesariamente al principio de cada bloque.

(4)



⇒ operador de resolución de ámbito

- Una función miembro puede llamar a otra función miembro o método
- a un atributo sin recordar de usar el operador punto sobre un objeto. El nombre del objeto y el operador punto sí los deberán utilizarse cuando una función miembro sea llamada por código que no pertenezca a ese clase.

### Sobrecarga de métodos

C++ no se pueden aplicar sobrecargas, por eso de:

int  $\text{abs}(\text{int } i)$   
double  $\text{abs}(\text{double } b)$   
long  $\text{abs}(\text{long } l)$

C++  
int  $\text{abs}(\text{int } i)$

Nota: Los tipos devueltos no le proporcionan suficiente información al compilador para que éste decida cuál es la función que debe llamar:

Ej:  $\text{int abs(int } i)$       ⇒ Fallo compilación.  
long  $\text{abs(int } i)$

⑤

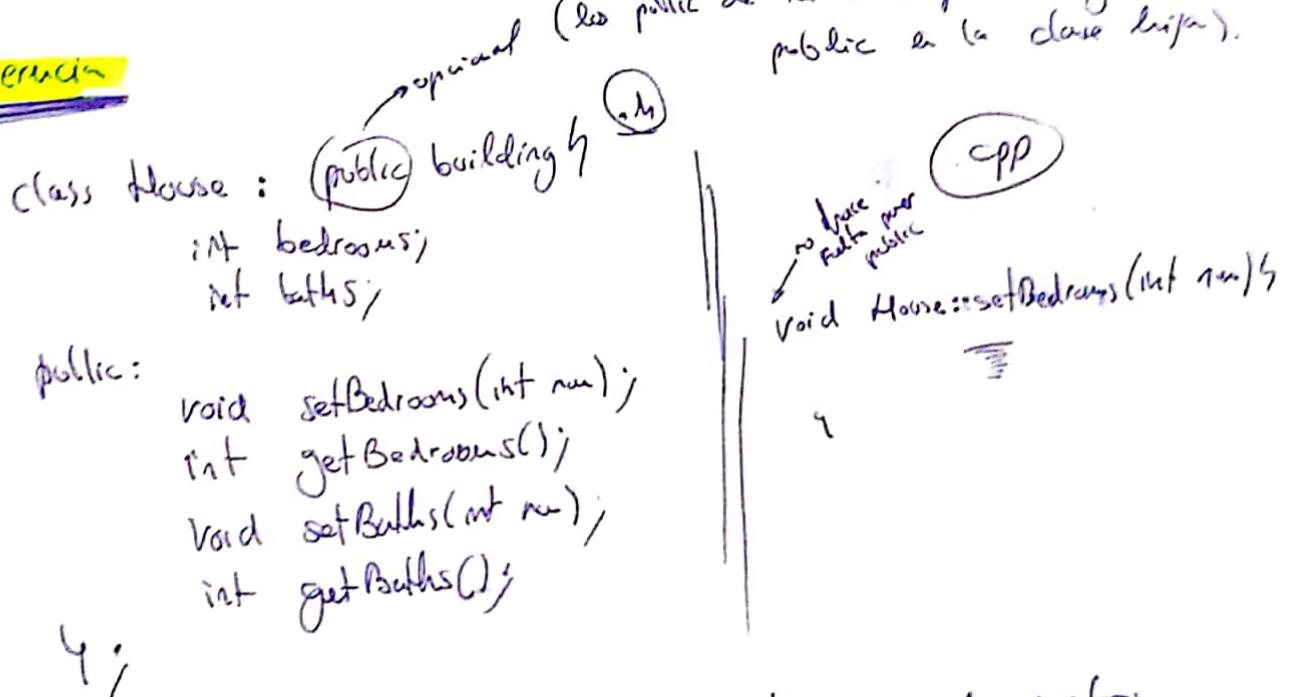
WOLAH

Scanned by CamScanner

## SOPORTARÁ A DE OPERADORES

↳ capt 14.

### Mencion



- ④ Una clase derivada tiene acceso a sus propios miembros y a los miembros p. (o protected) de la clase base.

### Funciones amigas

Es un uso de la palabra friend para dar a una función acceso a los miembros privados de una clase.

el acceso a los miembros privados de una clase.

class MiClase {
 int a, b;
}

public:

friend sumar(MiClase x);

void set\_ab(int i, int j);

};

void MiClase::set\_ab(int i, int j) {

a = i;

b = j;

}

int sumar(MiClase x) {

return x.a + x.b;

};

main() {

MiClase c;

c.set\_ab(3, 4);

cout << sum(c);

return 0;

## Funciones "insertadas" (inline)

Es posible en C++ crear funciones cortas, a las que no se invoca, en lugar de ello, su código se inserta (replica) cada vez que se muestren.

Ej:  
~~inline~~ inline int max (int a, int b){  
    =>  
    y  
}

Código + eficiente, pero muy grande.  
compilado

inline es una petición al compilador, no siempre la respetará (ej. un recursivo)  
Los métodos explícitamente directamente en la definición de la clase  
(.h) son inline automáticamente.

class MiClase{  
    int a,b;}

public:

void init(int i, int j){a=i; b=j;}

void show() {cout << a << " " << b << endl;}

y;

Se aplica para constructores y destructores;

# DE TODOS LOS PERMISOS GRATUITAS

**CLASES TEÓRICAS ONLINE**

autoescuelalara.com

647 63 53 39



**TU CARNET POR 20€**  
ADEMÁS TE FINANCIAMOS  
+ 4 clases de circulación

## Constructores parametrizados

- Los constructores tienen que ser públicos.
- Si no se declaran un constructor, el compilador por defecto creará uno vacío y sin argumentos.
- Si se declara al menos un constructor con parámetros habrá que pasar argumentos al crear el objeto (el de por defecto sólo se crea si no hay ningún otro declarado).

```
class Pareja
public:
 Pareja(int a, int b);
};
```

Pareja p1; *fallo compilación*  
 Pareja p2(); *que existe el constructor por defecto vacío, no se pueden usar los parámetros.*

---

```
class Pareja {
private:
 int a, b;
public:
 Pareja(int a, int b);
};
```

---

```
Pareja::Pareja (int a, int b) {
 ::a = a;
 ::b = b;
}
```

*sin tipo de vuelta ni visibilidad*

*alternativa*

Pareja :: Pareja (int a2, int b2): a(a2), b(b2) {}

---

*se puede asignar valores por defecto a los constructores usando el =*

Pareja (int a=0, int b=0): a(a), b(b) {}



**WUOLAH**

Scanned by CamScanner

- Cuando se asigna un objeto a otro se copian todos los valores de los atributos de uno en otro.

Pareja p3(2,3);

Pareja p2;

p2 = p3;

Constructor copia → para comportamiento de copia distinto de la simple copia de objetos "personalizado" es en una asignación de

class Pareja {

public:

, Pareja (int a=0, int b=0): a(a), b(b) {}

Pareja (const Pareja &p);

private:

int a, b;

}

Pareja::Pareja (const Pareja &p): a(p.a), b(p.b) {}

int main () {

Pareja p1(12,32);

Pareja p2(p1);

Pareja p2 = p1;



int \*ptVariable;

int & referenciaVariable;

//

(&ptVariable)

Cf + (alias o dirección de una variable)  
Deben inicializarse al declararse  
No puede ser un valor arbitrario, tiene que ser a una variable  
int i = 2;  
int & ref = i;  
• No recomienda usar → o (&ptVariable)  
• Siempre referenciar a i, no se puede cambiar todo lo que se aplique  
a la ref se aplica directamente al objeto directamente



permite la invocación a funciones por referencia, al igual que usando

pasar por referencia con referencia!

```
void incr(int& a) {
 a++;
```

}

```
int main() {
```

```
 int x=5;
 incr(x);
```

}

es pasar el valor directamente  
pero sin hacerse copiar, sin que  
es por referencia

Pasar por referencia con puntero

```
void incr(int* a) {
 *a++;
```

\*

```
int main() {
```

```
 int x=5;
 incr(&x);
```

\*

Llamada por copia

estimulante y se usan con claves que se quita hacer por copia y no  
por referencia

```
void incr(int a) {
 a++;
```

\*

```
int main() {
```

```
 int x=5;
 incr(x);
```

\*

Una función puede retornar una referencia:

```
int& incr(int& a) {
 a++;
 return a;
```

\*

```
int main() {
 int x=5;
 incr(x)++; //3
 incr(x)=5; //5
```

## REFERENCIAS

```
int a=42;
```

```
int& ra=a;
```

ra = 30; (a vale 30 ahora)

si hacemos

int& raj =< X error de  
compracción

## Paso de Objetos a Funciones

Cuando se manda un método por valor se copia el objeto de argumento a él del parámetro (pero no se lleva al constructor ya que no hace falta inicializar nada). Sin embargo, si se lleva al destrucción se destruye. (se ~~destruye~~ por ref) La copia del objeto se realiza a nivel de bits (duplicado exacto del original).

Cuando un método devuelve un objeto creado dentro de ese método ~~es una variable privada de ese método~~ debe ser asignado a otro objeto externo para que en la asignación se haga la copia de dicho objeto. Ej cuando devolver un String. Permanece vivo dentro del ámbito donde se creó esa función y es una copia de ese valor devuelto.

## Referencias (cont)

- No se puede referenciar a otra referencia (no se puede obtener la dirección de una referencia)
- No se pueden crear arrays de referencias.
- No se pueden crear punteros a referencias.
- Las referencias nulas no están permitidas

## (ADVERTENCIA) !

Intér a, b;  
Puntero varible el va  
nula

8

WUOLAH

Scanned by CamScanner

# DE TODOS LOS PERMISOS GRATUITAS

CLASES  
TEÓRICAS  
ONLINE

autoescuelalara.com



647 63 53 39



ADEMÁS TE  
FINANCIAMOS  
**TU  
CARNET  
POR 20€**  
+ 4 clases de circulación



Clase Persona {

private:

int i;

public:

void setJ(int j) { inline

i = j;

int getJ() { return i; } // informe

}

main () {

Persona personas[3];

int i;

for (i = 0; i < 3; i++) {

personas[i].setJ(i + 3);

// comienza en cero

for (i = 0; i < 3; i++) {

cout << personas[i].getJ() << " ";

}

Persona personas[3] = { 1, 2, 3, 4 };

✓ Funciona pq no lleva de donde ningún constructor con parámetros y existe el vecio implícitamente para permitir crear el array.

(100) (1)



WUOLAH

Scanned by CamScanner

```
class Persona {
```

```
private:
```

```
int h;
int i;
```

```
public:
```

```
Persona (int j, int k) { constructor visible
```

```
h=j;
```

```
i=k;
```

```
int getI() { return i; }
```

```
int getH() { return h; }
```

```
}
```

```
void main () {
```

```
Persona personas[3] = { Persona(3,2), Persona(2,3), Persona(3,4) };
```

```
for (int i=0; i<3; i++) {
```

```
cout << personas[i].getI() << " " << personas[i].getH();
```

```
 }
```

```
return 0;
```

} endl;

```
class Persona {
```

```
private:
```

```
int i;
```

```
public:
```

```
Persona (int n) { i=n; }
```

```
void setI (int n) { i=n; }
```

```
int getI () { return i; }
```

```
}
```

```
void main () {
```

```
Persona personas[3]; // falla compilación
```

```
Persona personas[3] = {3,5,6}; // usa constructor con parámetros
```

Personas(); i=0; }

usar constructor  
vario. para  
una función

int n[ ] = { 1, 2, 3, 4, 5 }; ✓ correcto

int n[2] = { 1, 2, 3, 4, 5 }; ✗ falló compilación, no conocido el tamaño

const int arraySize = 10; ✓ si const siempre asigna valor

int s[arraySize];

- Buena práctica de programación declarar los tamaños de los arrays con variables constantes
- Hace la programación más estable, fija para recorrer los elementos.

int arraySize = 5;  
static int s[arraySize]; } ✗ falló la compilación, los arrays estáticos  
para indicar el tamaño necesitan constantes.

for (int j=0; j<arraySize; j++)  
 cin >> s[j];

→ le da valor a cada elemento desde teclado.

⚠ C++ no te protege de intentar sobrepasar los límites de memoria del Array:

int array[2];

array[5] = 3; → Me he salido del array. (C++ no me avisó.)

• no existe función size para un array (sí para un vector).

array

dinámico

| Pasando arrays a funciones

void mostrarInfoClientes(Persona clientes[], int n); (Se pasan por referencia, sin tener que usar el & en la declaración de las funciones)

void main() { Personas personas[9]; } (el problema se produce porque no se pasa el nombre de la función pq no lo conoce el compilador)

Personas personas[9];

mostrarInfoClientes (personas, 9);

no poner funciones, el compilador lo ignora, si le hubiera puesto 15, hubiera dado =

5) void mostrarInfoClientes ( Persona clientes[], int numClientes ) {

for (int i=0; i<numClientes; i++)  
 cout << clientes[i].getID() << endl;

16/16 ✓ 10

WUOLAH

Scanned by CamScanner

② th se pueden pasar elementos concretos.

```

#include "Person.h"
void mostrarInfoCliente (Persona);
void main () {
 Persona clientes[2] = {Persona(3), Persona(4)};
 mostrarInfoCliente (clientes[1]);
}
void mostrarInfoCliente (Persona p) {
 cout << p.get();
}

```

← Spécifique prototyp pour que puisse compiler

pasa por valor en este caso  
(Person& p)  
↳ por référence

o Si al pasar por ref. le ponemos **const** delante en la declaración  
del método (o función) se llama por referencia pero no permite  
modificar su valor.



# Pack permiso B + 3 clases

Código Promoción: wuolah-2020

39,00€  
33,15€

## ARRAYS MULTIDIMENSIONALES

⇒ falta de valores (16 objetos)

$$\text{int } b[2][2] = \begin{bmatrix} 13, & 24, & 13, 44 \\ 22, & 13, & 13, 22 \end{bmatrix}$$

void función (Personas clientes [ ] [ 5 ] ) {  
=

Y

## VECTOR

#include &lt;vector&gt;

void mostrarVector ( vector&lt;int&gt; &amp; ) ;

void main () {  
vector<int> enteros(7);  
cout << "tamaño del vector es: " << enteros.size() << endl;  
mostrarVector (enteros); }  
Método por ref

void mostrarVector ( vector<int> & enteros) {  
for (int i=0; i< enteros.size(); i++) {  
cout << \*enteros.at(i) << endl;  
enteros[i] = redefinición de tipo?  
} }

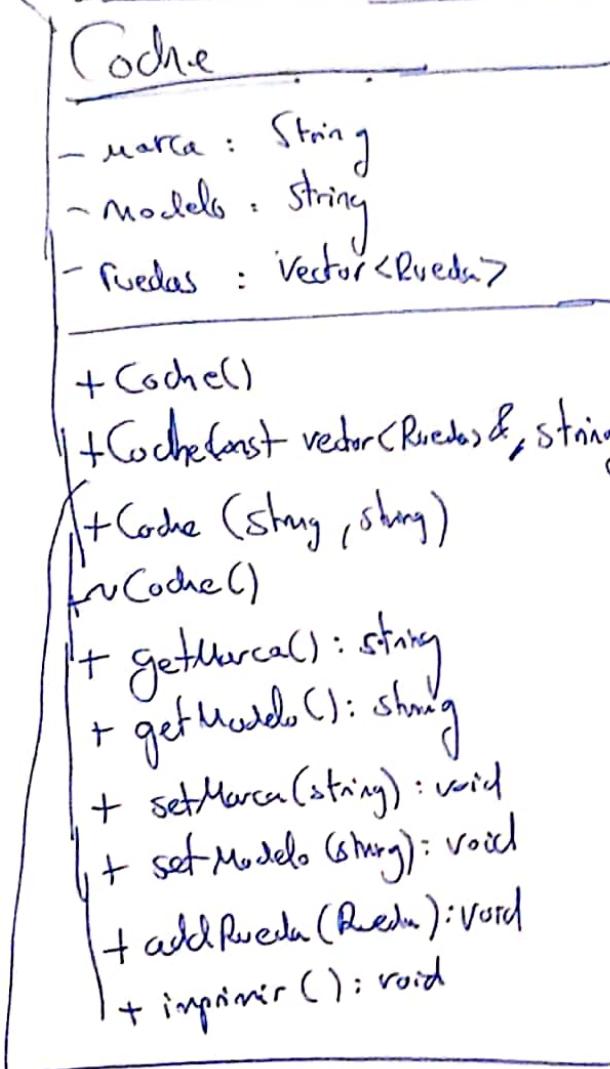
Y  
añadir elementos → enteros.push\_back(4);  
Valor a vector al final

(18)  
(23)

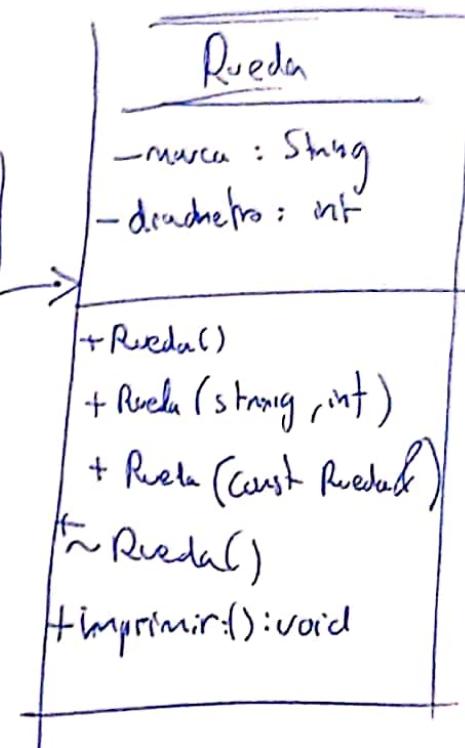
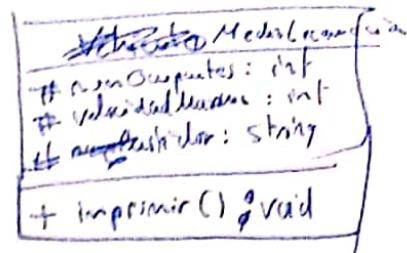
- `front()` → devuelve el primer elemento del vector
- `back()` → te devuelve el último elemento del vector
- `pop_back()` ⇒ saca el último elemento del vector
- `empty()` ⇒ booleano si está vacío el vector
- `erase()` ⇒ borra el vector sin elementos
- `clear()`
- `size()` ⇒ num. elementos en el vector
- `capacity()` ⇒ tamaño real del array subyacente
- `insert()` ⇒ inserta en una otra posición del vector
- `begin()` ⇒ devuelve un iterador al primer elemento del vector
- `end()` ⇒ devuelve un iterador al último elemento del vector

XV  
Objetos

## EJEMPLO



+ Coche (int, int, string, string, string)

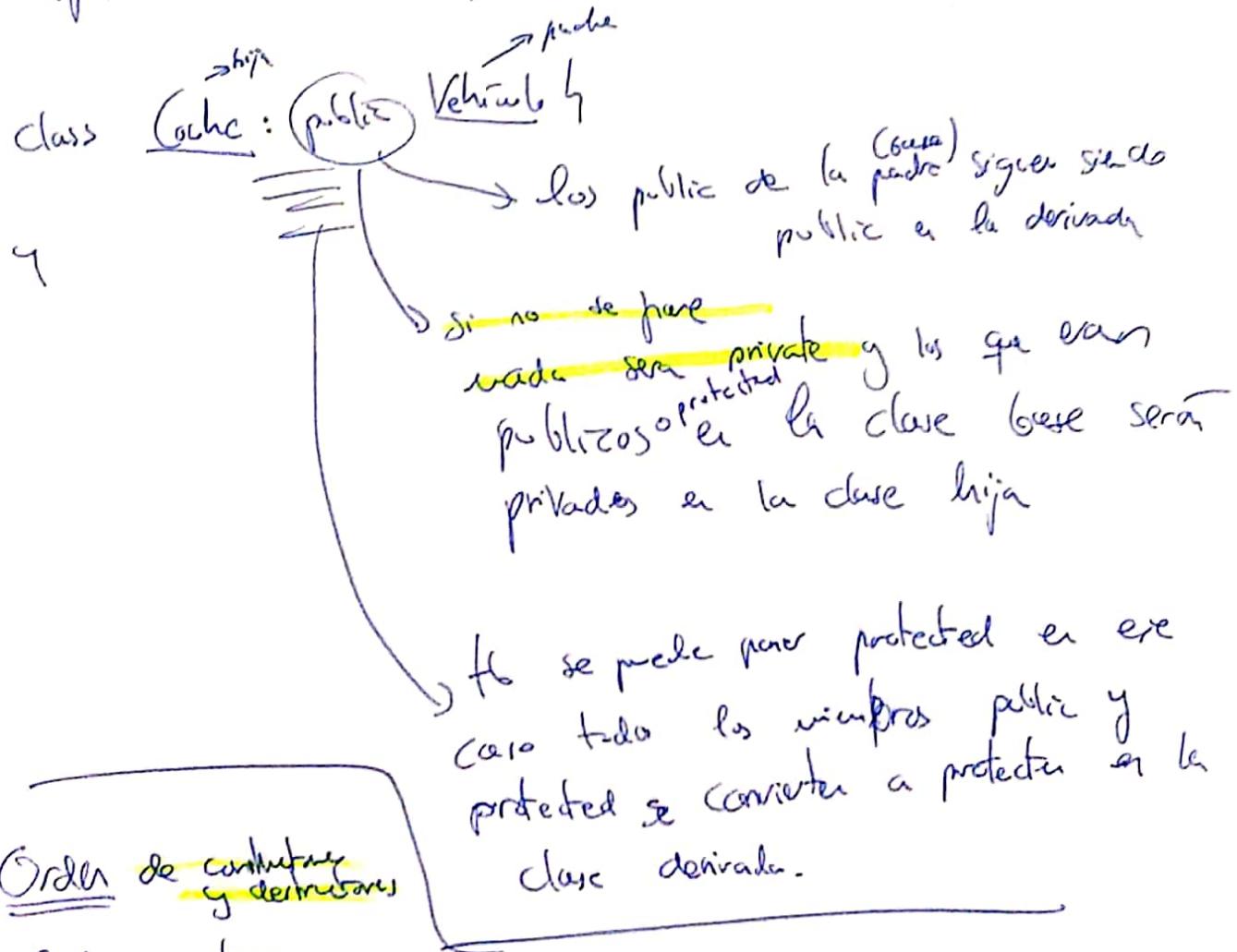


Scanned by CamScanner

## HERENCIA

Padre  $\Rightarrow$  clase base

Hija  $\Rightarrow$  clase derivada



Constructendo base  
Constructendo derivada  
Destruyendo derivada  
Destruyendo base

void main ()  
{  
 Hija hija;  
 return 0;

Class Base

public:  
Base() {cout << "Cons base" << endl;}  
~Base() {cout << "Destr. base" << endl;}

{};

Class Hija

public:  
Hija() {cout << "Cons hija" << endl;}  
~Hija() {cout << "Destr. hija" << endl;}

{};

13

Scanned by CamScanner

Iniciá tu clase heredando los atributos de la clase padre

clase Base ↳

```
private:
 int x, y;
public:
 Base(int x, y);
 Base();
```

clase Hija ↳

```
private:
 int w;
public:
 Hija(int x, y) : Base(x, y) {
 w = 10;
 }
```



Bengala