

# CS1112: Programación 2

Unidad 5: POO (parte 2)

Sesión de Teoría - 9

Profesor:

José Antonio Fiestas Iquira [jfiestas@utec.edu.pe](mailto:jfiestas@utec.edu.pe)

Material elaborado por:

Maria Hilda Bermejo, José Fiestas,  
Rubén Rivas, Heider Sánchez



# Índice:

- Unidad 5: POO (Parte 2)
  - Objetos dinámicos
  - Arreglos de objetos

# Logro de la sesión:

Al finalizar la sesión, los alumnos se familiarizan con el paradigma de la programación orientada a Objetos.

- Clase – Objeto
- Métodos de acceso (setter y getters)
- Constructores y destructores
- Objetos dinámicos
- Arreglo de objetos

# 5.3

Unidad 5: POO  
Objetos dinámicos

UTEC

# Clase, Objetos y mensajes (repaso)

---

# Repaso: Clase y Objeto

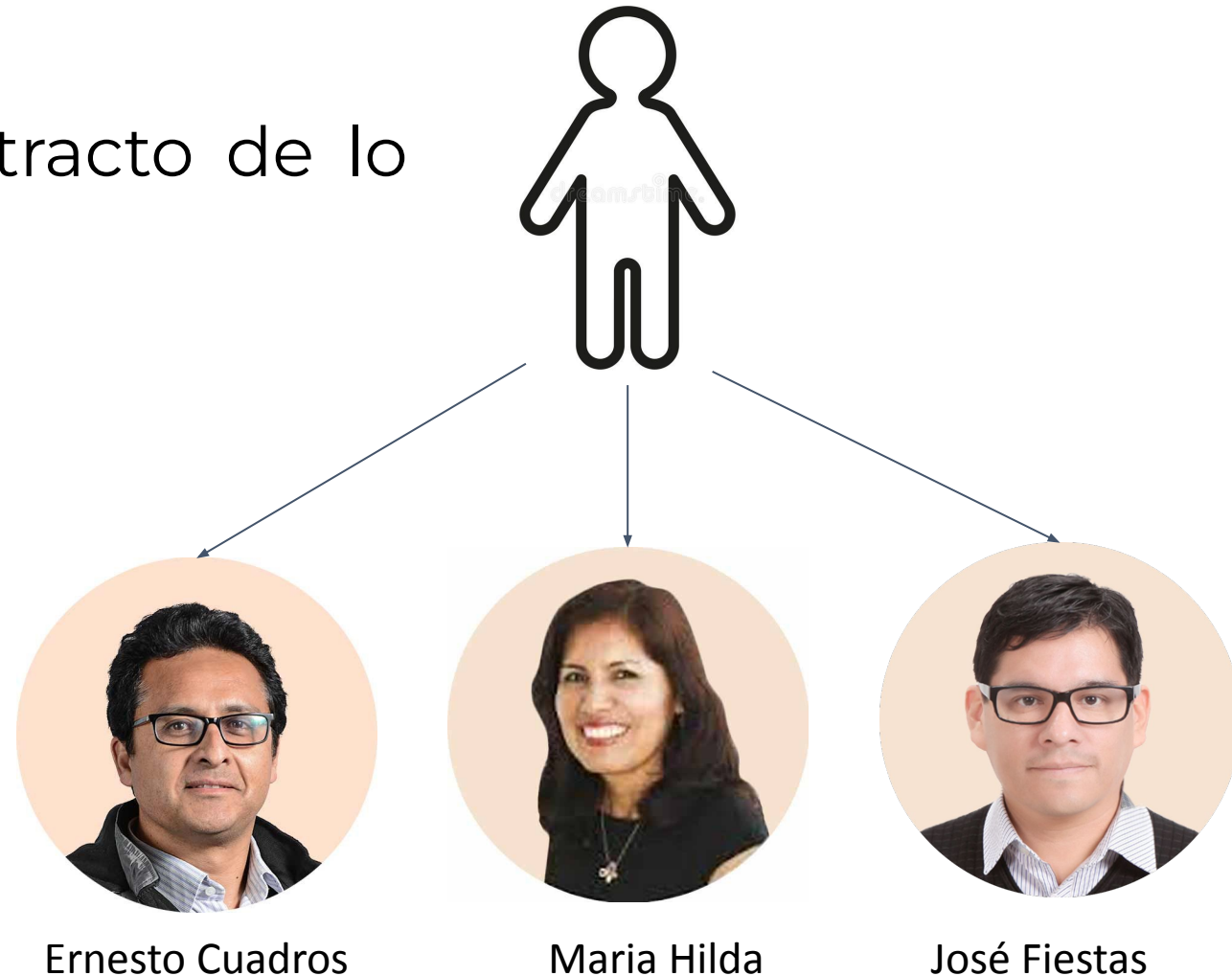
---

## Una Clase:

Una clase es el concepto abstracto de lo que se quiere crear.

## Un Objeto:

Es la instancia de una clase, es decir es un ejemplo concreto de una clase.



# Repaso: Interfaz



Monitor

**Clase**

encender()  
apagar()  
brillo(valor)  
...

**Interfaz**  
**public**

En C++:

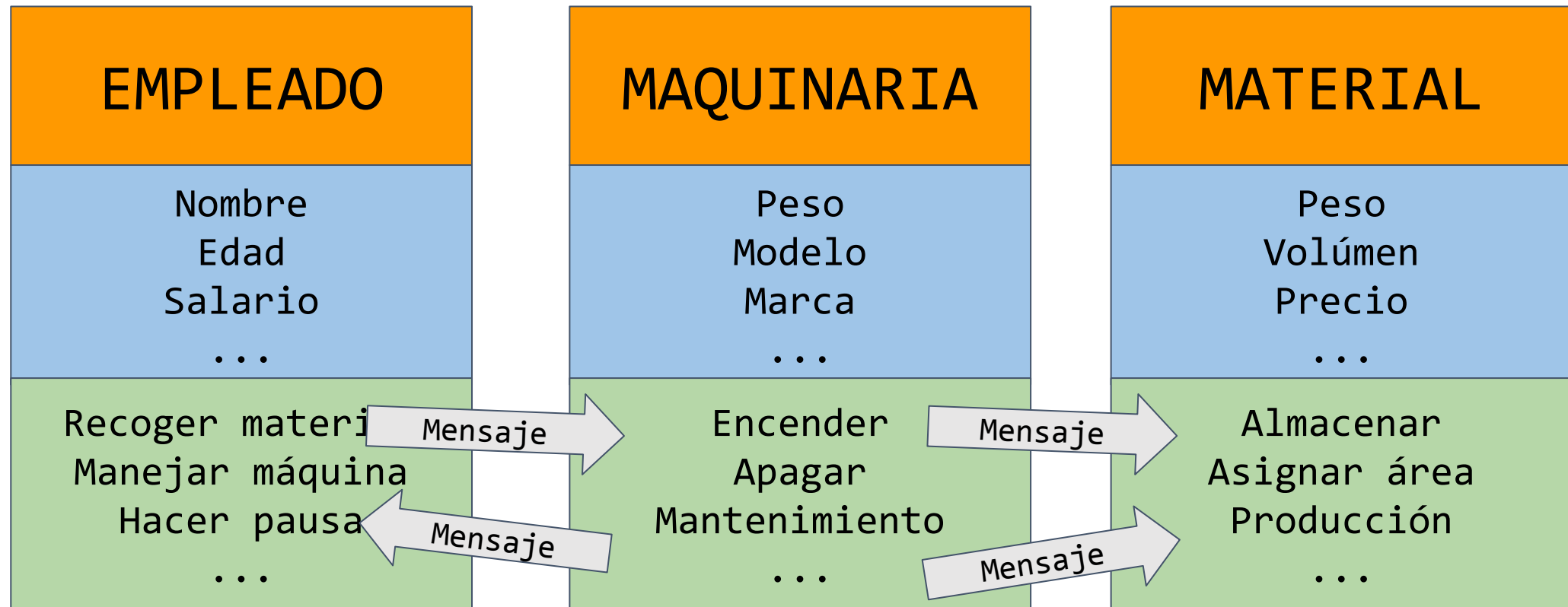
```
Monitor asus1;  
asus1.brillo(90);
```



# Repaso: Mensaje

Los objetos pueden enviar y/o recibir mensajes a objetos de su misma clase o de otra clase.

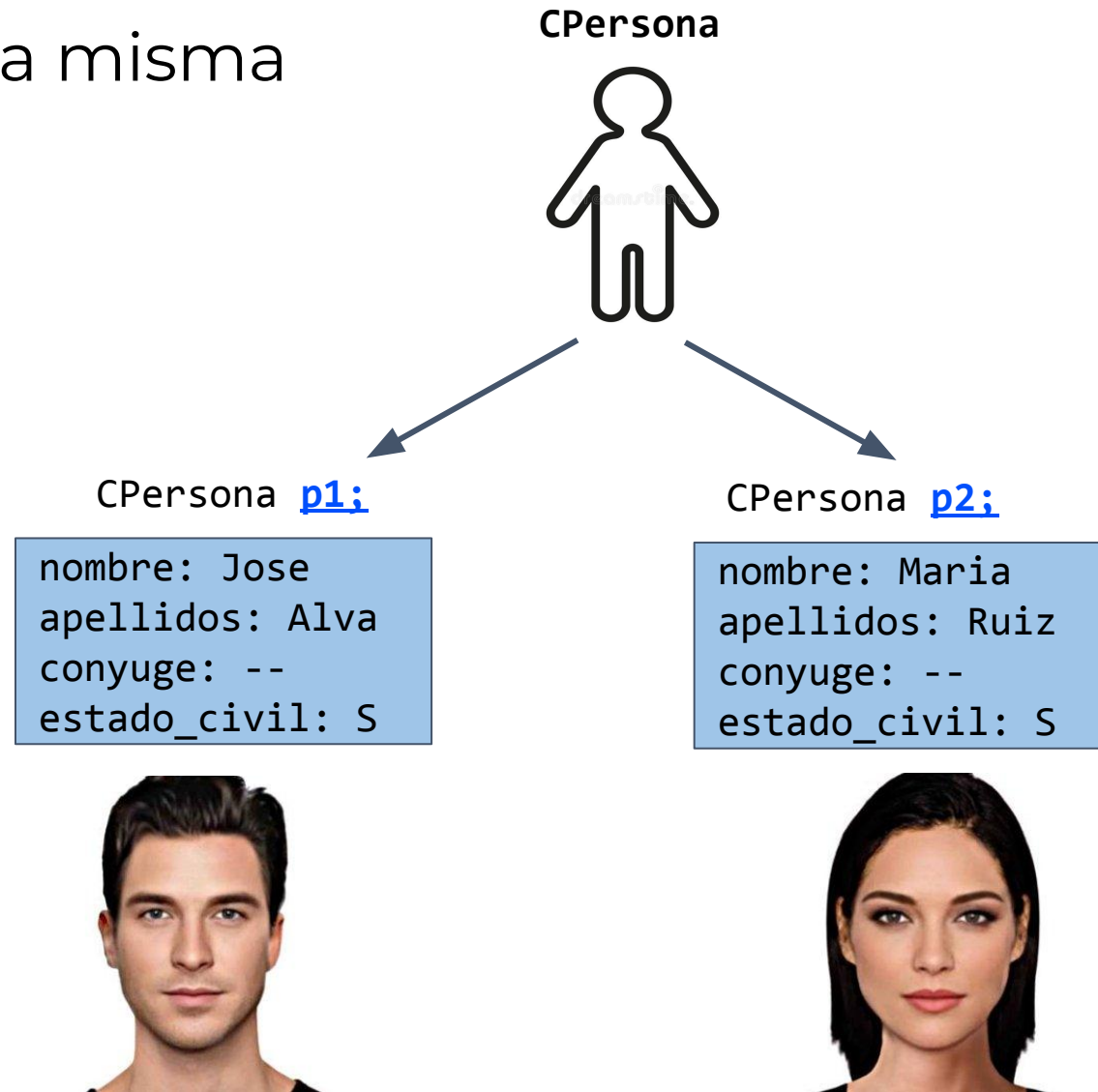
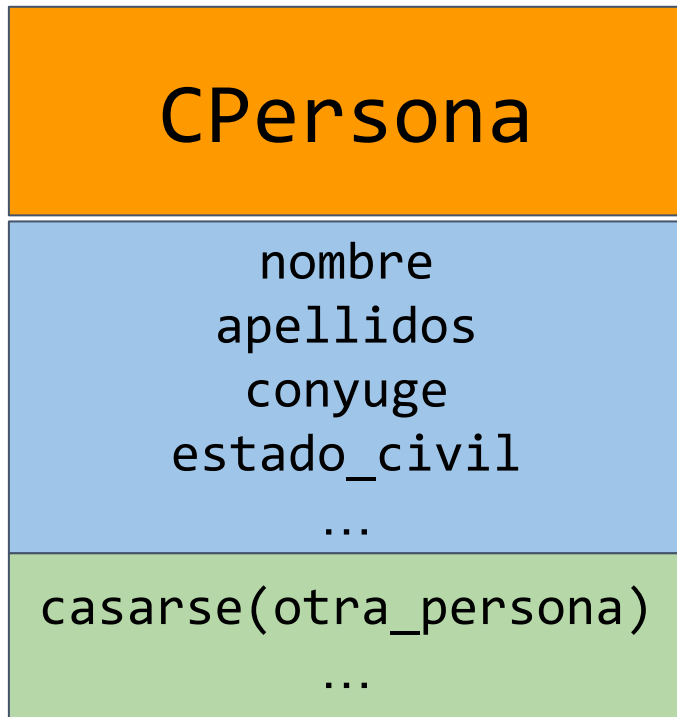
Los mensajes pueden modificar el estado del objeto.





# Mensaje: Ejemplo 1

Mensajes entre objetos de la misma clase



# Mensaje: Ejemplo 1

## Mensajes entre objetos de la misma clase



CPersona p1;

nombre: Jose  
apellidos: Alva  
conyuge: Maria  
estado\_civil: C

CPersona p2;

nombre: Maria  
apellidos: Ruiz  
conyuge: Jose  
estado\_civil: C

p1.casarse(p2)

```
void CPersona::casarse(CPersona &otra_persona)
{
    this->conyuge = otra_persona.nombre;
    otra_persona.conyuge = this->nombre;
    this->estado_civil = 'C';
    otra_persona.estado_civil = 'C';
}
```



# Mensaje: Ejemplo 2

Mensajes entre objetos de diferente clase

## CRobot

Nombre  
posicion  
vidas  
...

chocar(obstaculo)  
morir()  
...

## CObstaculo

Tipo  
posicion  
daño  
...

CRobot [r1;](#)



nombre: Lucky  
posicion: (15,3)  
vidas: 2

CObstaculo [o1;](#)



tipo: Rompe\_lomo  
posicion: (5,3)  
daño: 2

# Mensaje: Ejemplo 2

Mensajes entre objetos de diferente clase

CRobot r1;

nombre: Lucky  
posicion: (5,3)  
vidas: 0

CObstaculo o1;

tipo: Rompe\_lomo  
posicion: (5,3)  
daño: 2

r1.chocar(o1)

```
void CRobot::chocar(CObstaculo &obstaculo)
{
    CRobot::vidas -= obstaculo.getDaño();
    if(CRobot::vidas == 0)
        CRobot::morir();
}
```



# **Objetos dinámicos**

---

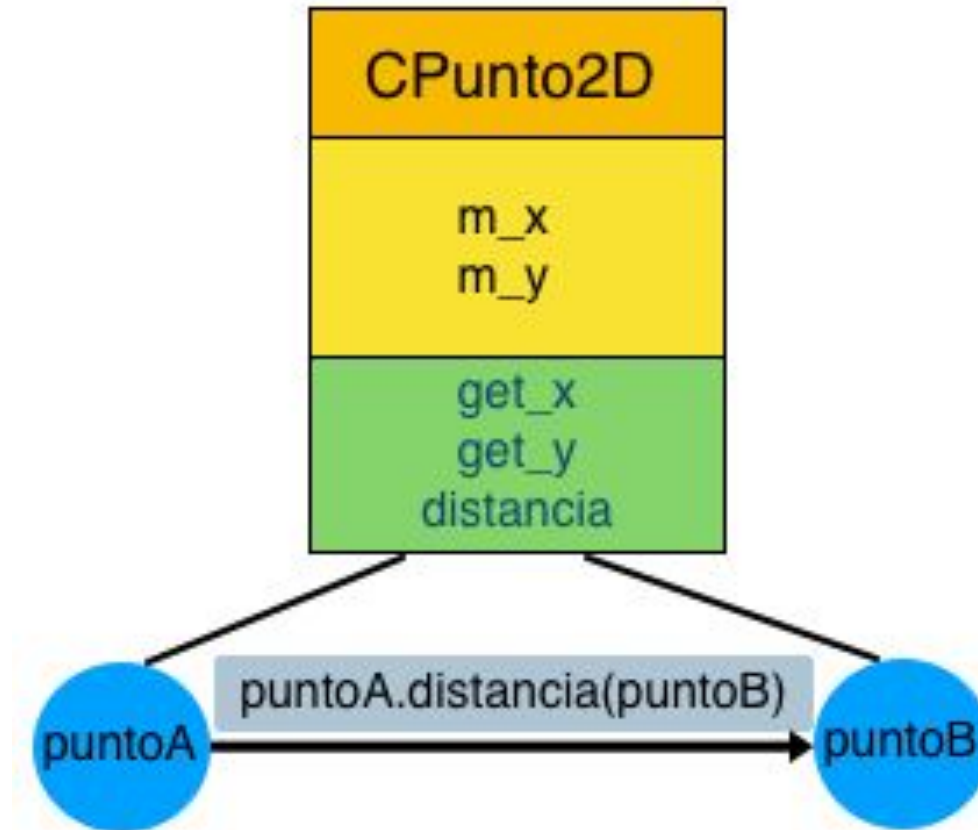
# Ejemplo 01: mensajes entre objetos de la misma clase

una clase para representar puntos en el plano cartesiano.

---

Se definen **objetos** de clase CPunto2D y el método **distancia**

Se envía el mensaje de calcular la **distancia** de un objeto (puntoA) a otro (puntoB)



# Ejemplo 01: objeto dinámico

una clase para representar puntos en el plano cartesiano.

```
typedef int entero;
```

CPunto2D.h

```
class CPunto2D {  
private:  
    entero m_x, m_y;  
public:  
    CPunto2D(){m_x = m_y = 0;}  
    CPunto2D(entero x, entero y);  
    entero getX();  
    void setX(entero x);  
    entero getY();  
    void setY(entero y);  
};
```

```
#include "CPunto2D.h"
```

CPunto2D.cpp

```
CPunto2D::CPunto2D(entero x, entero y) :  
    m_x(x), m_y(y) { }  
  
entero CPunto2D::getX() {  
    return m_x;  
}  
  
void CPunto2D::setX(entero x) {  
    m_x = x;  
}  
  
entero CPunto2D::getY() {  
    return m_y;  
}  
  
void CPunto2D::setY(entero y) {  
    m_y = y;  
}
```



# Ejemplo 01: objeto dinámico

una clase para representar puntos en el plano cartesiano.

```
#include <iostream>
```

CPunto2D.h

```
typedef int entero;
```

```
typedef float decimal;
```

```
class CPunto2D {
```

```
private:
```

```
    entero m_x, m_y;
```

```
public:
```

```
    CPunto2D(){m_x = m_y = 0;}
```

```
    CPunto2D(entero x, entero y);
```

```
    CPunto2D(CPunto2D &p);
```

```
    entero getX()          {return m_x;}
```

```
    void setX(entero x)     {m_x = x;}
```

```
    entero getY()          {return m_y;}
```

```
    void setY(entero y)     {m_y = y;}
```

```
    decimal distancia(CPunto2D &p);
```

```
};
```

```
#include "CPunto2D.h"  
#include <cmath>
```

CPunto2D.cpp

```
CPunto2D::CPunto2D(entero x, entero y)
```

```
{
```

```
    m_x = x;
```

```
    m_y = y;
```

```
}
```

```
CPunto2D::CPunto2D(CPunto2D &p)
```

```
{
```

```
    m_x = p.m_x;
```

```
    m_y = p.m_y;
```

```
}
```

```
decimal CPunto2D::distancia(CPunto2D &p)
```

```
{
```

```
    double dist = 0;
```

```
    dist += pow(p.m_x - m_x, 2);
```

```
    dist += pow(p.m_y - m_y, 2);
```

```
    dist = sqrt(dist);
```

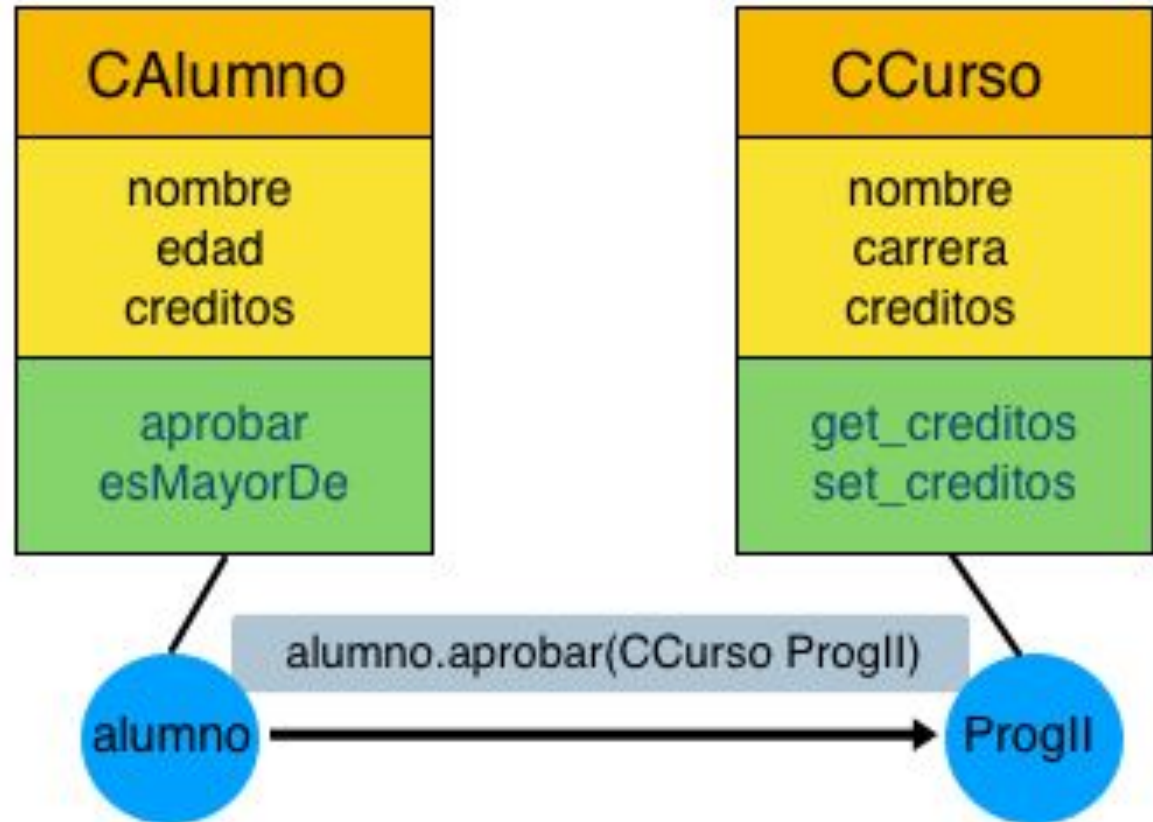
```
    return dist;
```

```
}
```

# Ejemplo 02: mensajes entre objetos de distinta clase

Se definen **objetos** de clase CAlumno y CCurso.

Se envía el mensaje de **aprobar** de un objeto de clase CAlumno (alumno) a otro de clase CCurso (ProglI)



```
class CAumno {  
    string nombre;  
    int edad;  
    int credits;  
public:  
    CAumno(){}  
    CAumno(string nombre, int credits);  
  
    void aprobar(CCurso curso);  
    bool esMayorDe(CAumno otro);  
};
```

CAumno.h

```
#include "CAumno.h"
```

CAumno.cpp

```
CAumno::CAumno( string n, int c) :  
    nombre(n), credits(c) {}  
  
void CAumno::aprobar(CCurso *curso){  
    this->credits += curso->getCredits();  
}  
  
bool CAumno::esMayorDe(CAumno &otro)  
{  
    return CAumno::credits > otro.credits;  
}
```

# Objeto estático: puntero a objeto en el stack

```
CPunto2D    p1(5, 2);  
CPunto2D*   ptrP1 = &p1;
```

*ptrP1*

0x1008

```
cout<<(*ptrP1).x<<endl;
```

5

```
cout<<(*ptrP1).y<<endl;
```

2

```
cout<<ptrP1->x<<endl;
```

5

```
cout<<ptrP1->y<<endl;
```

2

```
cout<<ptrP1<<endl;
```

0x1008

```
cout<<&ptrP1->x<<endl;
```

0x1008

```
cout<<&ptrP1->y<<endl;
```

0x100C

Dirección

Memoria

0x1008

0x1009

0x100A

0x100B

0x100C

0x100D

0x100E

0x100F

5

2

*x*

*y*

*p1*

**ALERTA!** Bajo el supuesto que *x* e *y* son miembros públicos.

# Objeto dinámico: asignación dinámica de memoria

```
CPunto2D* p2 = new CPunto2D(5, 2);
```

*p2*

0x1008

```
cout<<(*p2).x<<endl;  
cout<<(*p2).y<<endl;  
cout<<p2->x<<endl;  
cout<<p2->y<<endl;  
cout<<p2<<endl;  
cout<<&p2->x<<endl;  
cout<<&p2->y<<endl;
```

5  
2  
5  
2  
0x1008  
0x1008  
0x100C

Bajo el supuesto que **x** e **y** son miembros **públicos**.

Dirección

Memoria

0x1008  
0x1009  
0x100A  
0x100B  
0x100C  
0x100D  
0x100E  
0x100F

5

2

*x*

*y*

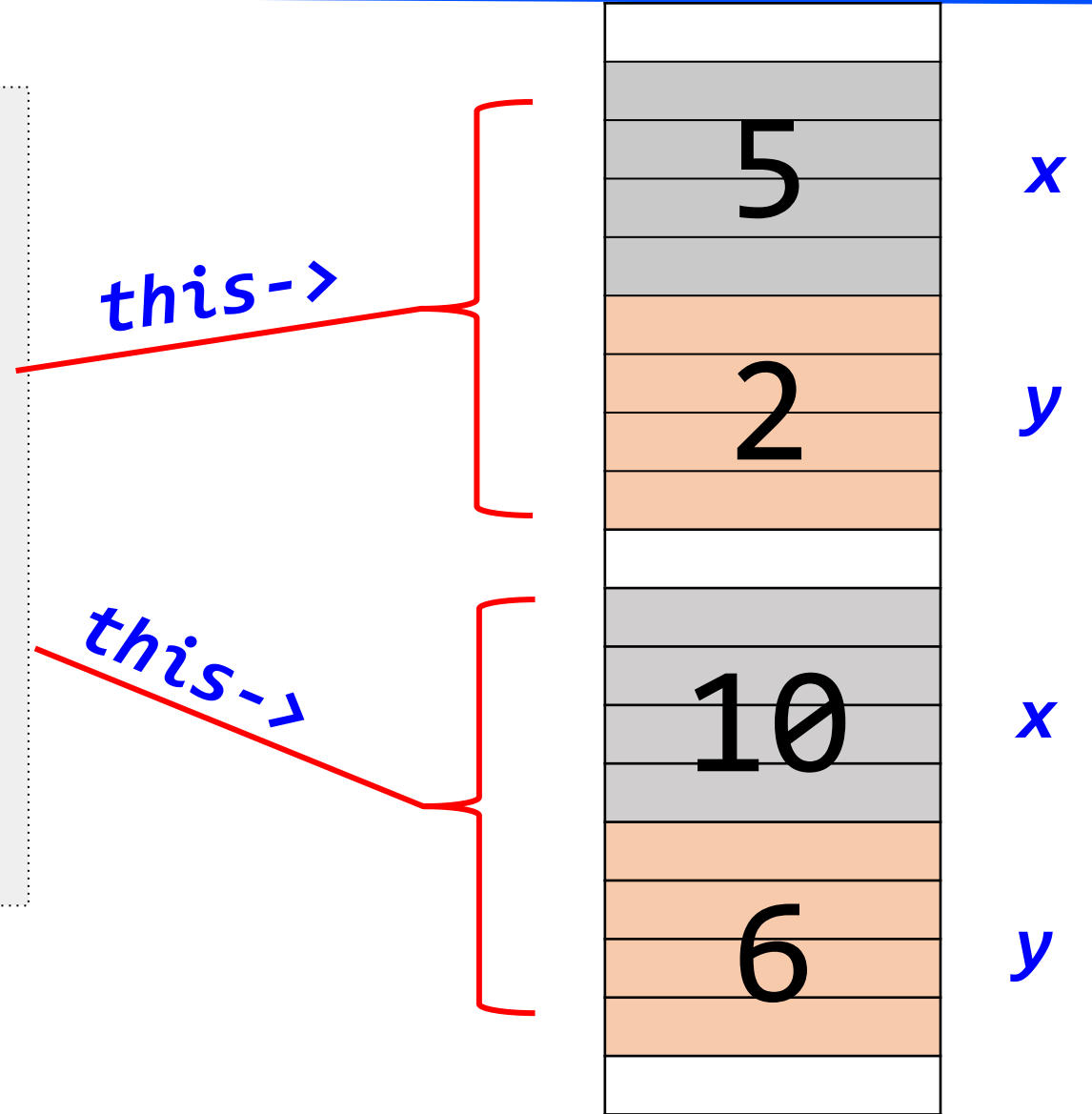
*\*p2*

**Objeto dinámico:** asignación dinámica de memoria

```
int main()
{
    CPunto2D* p1 = new CPunto2D(5,2);

    CPunto2D* p2 = new CPunto2D(10,6);

}
```



# Objeto dinámico: liberación de memoria

```
int main()
{
    CPunto2D*p1=new CPunto2D(5, 2);

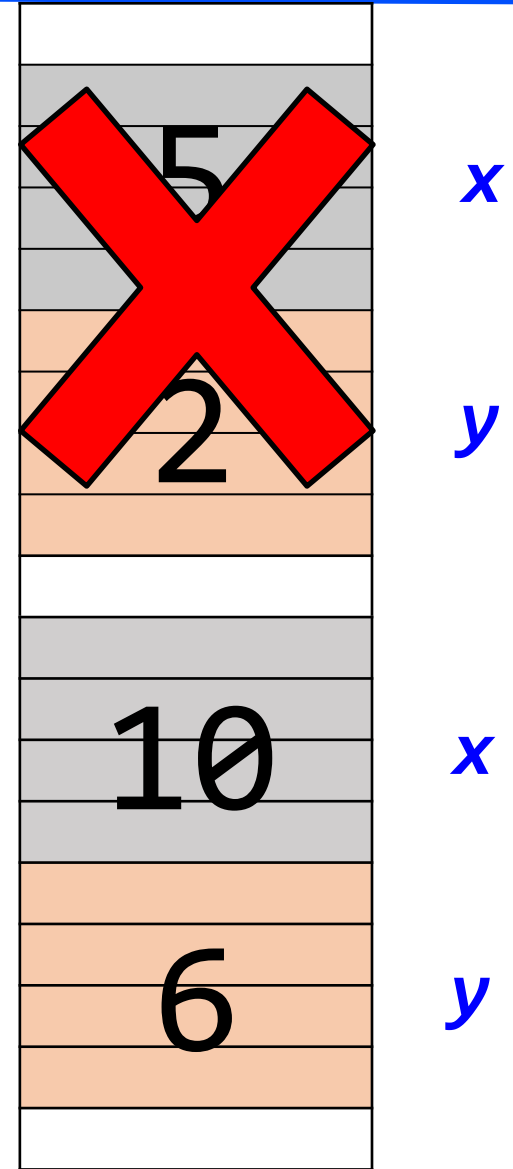
    CPunto2D*p2 = new CPunto2D(10,6);

    delete p1;

}
```

*this->*

*this->*





# Objeto dinámico: liberación de memoria

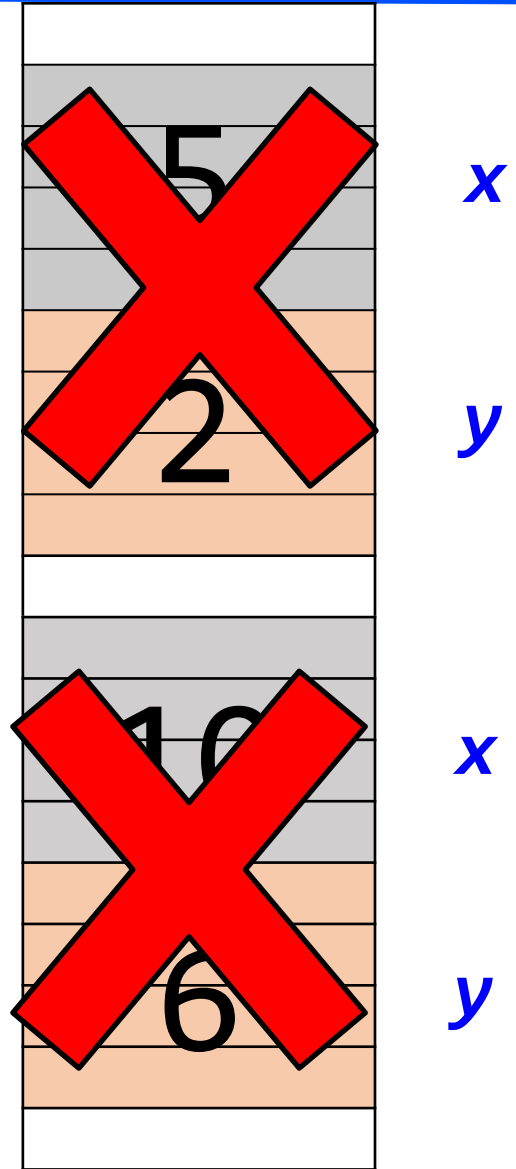
```
int main()
{
    CPunto2D*p1=new CPunto2D(5, 2);

    CPunto2D*p2 = new CPunto2D(10,6);

    delete p1;
    delete p2;
}
```

*this->*

*this->*



# 5.4

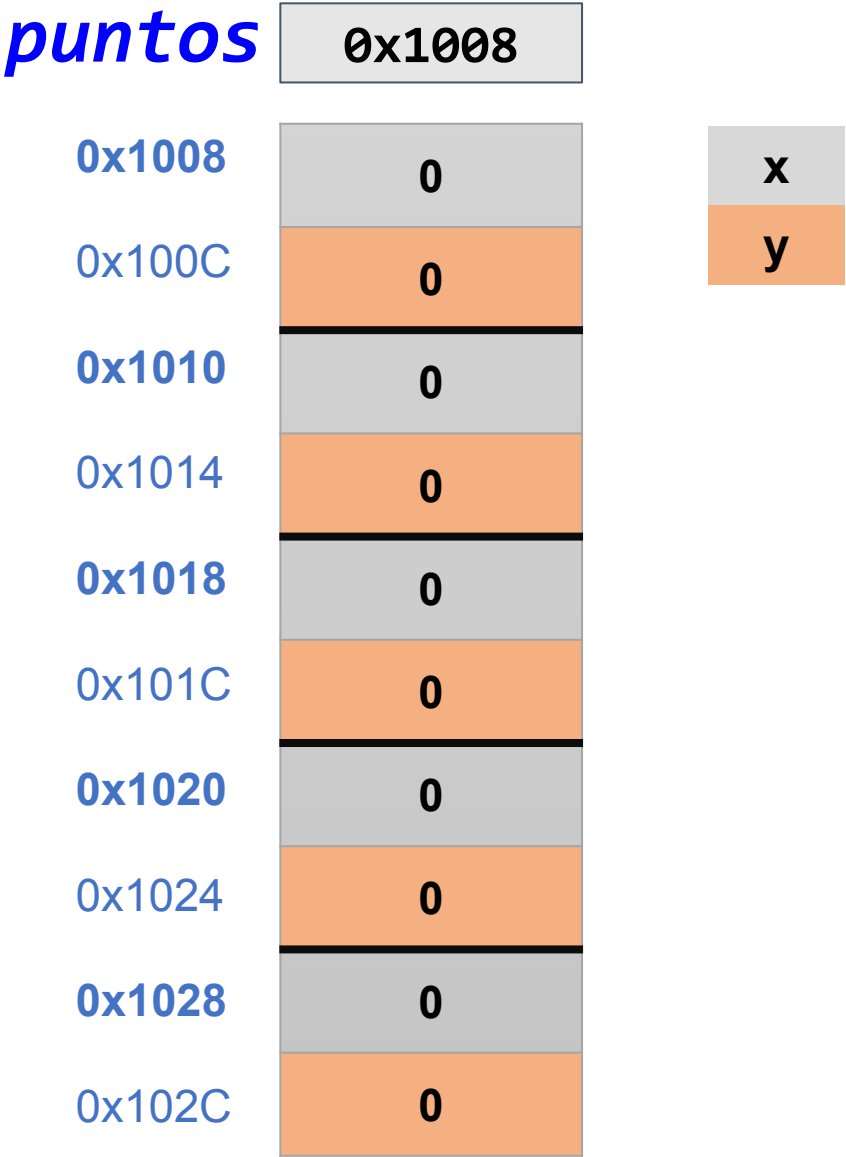
Unidad 5: POO  
arreglos de objetos

UTEC

# Arreglo de objetos: asignación dinámica de memoria

```
int main() {
    CPunto2D* puntos = new CPunto2D[5];
    //se ejecuta el constructor sin parámetros
    //CPunto2D(){x = y = 0;}

    return 0;
}
```



# Arreglo de objetos: asignación dinámica de memoria

```
int main() {  
    CPunto2D* puntos = new CPunto2D[5];  
    puntos[0] = CPunto2D(5,4);  
    puntos[1] = CPunto2D(6,2);  
    puntos[2] = CPunto2D(1,1);  
    puntos[3] = CPunto2D(3,1);  
    puntos[4] = CPunto2D(2,2);  
  
    for (int i = 0; i < 5; ++i) {  
        cout<<puntos[i].getX()<<" , "  
            <<(*puntos + i).getY()<<endl;  
    }  
    return 0;  
}
```

*puntos*

0x1008

0x1008

5

x

0x100C

4

y

0x1010

6

0x1014

2

0x1018

1

0x101C

1

0x1020

3

0x1024

1

0x1028

2

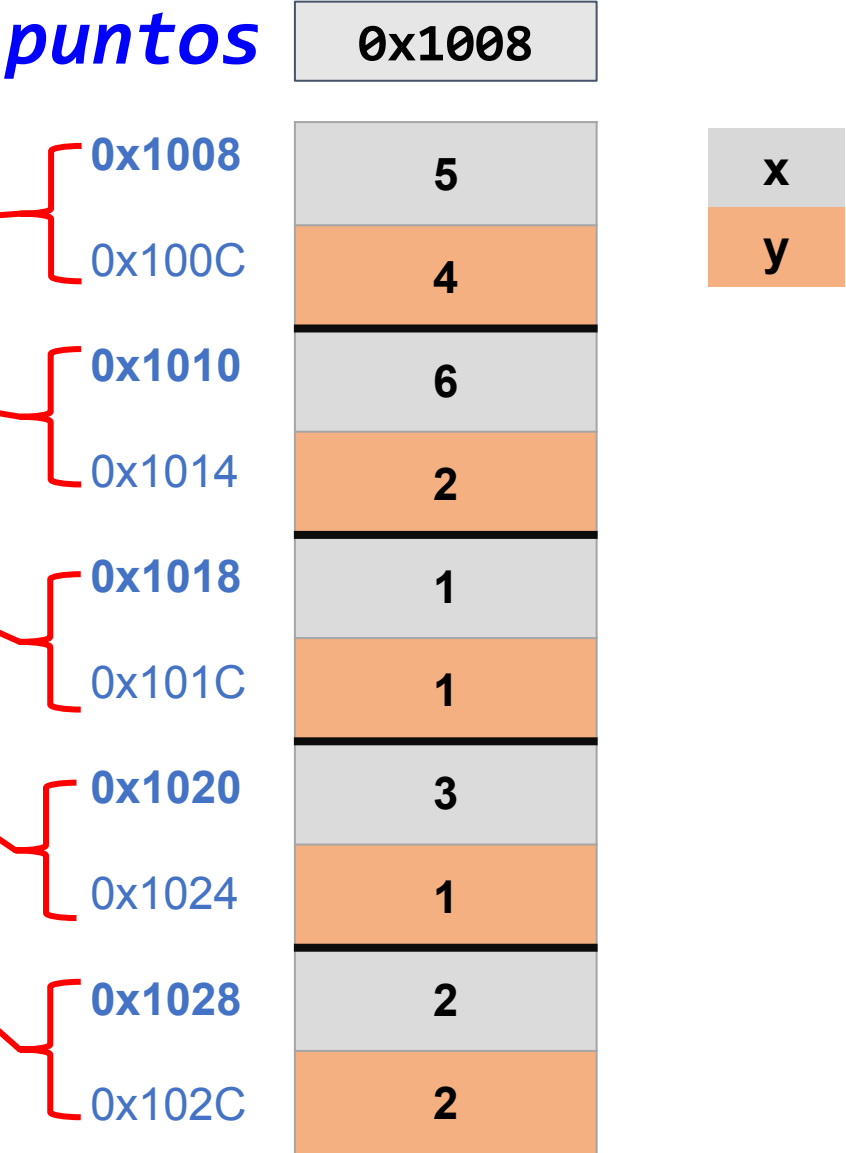
0x102C

2

# Arreglo de objetos: asignación dinámica de memoria

```
int main() {
    CPunto2D* puntos = new CPunto2D[5]{
        CPunto2D(5,4),
        CPunto2D(6,2),
        CPunto2D(1,1),
        CPunto2D(3,1),
        CPunto2D(2,2)
    };

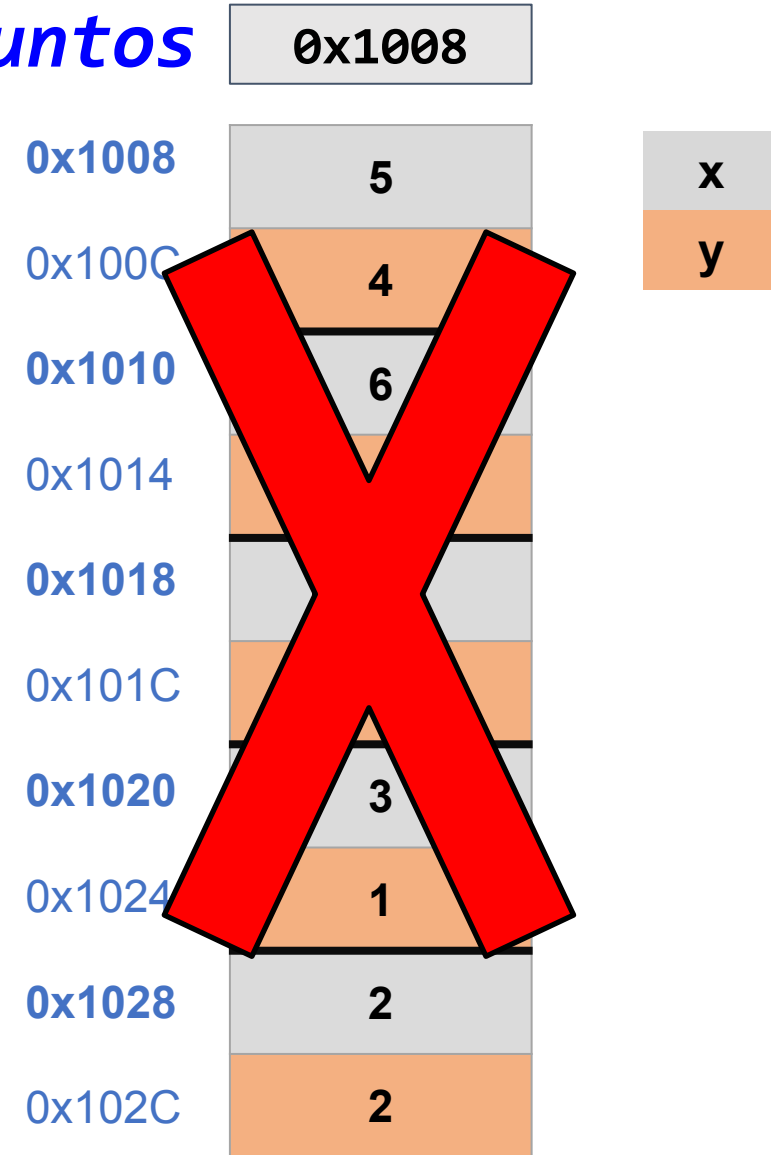
    for (int i = 0; i < 5; ++i) {
        cout<<puntos[i].getX()<<" "
            <<(*puntos + i).getY()<<endl;
    }
    return 0;
}
```



# Arreglo de objetos: liberación de memoria

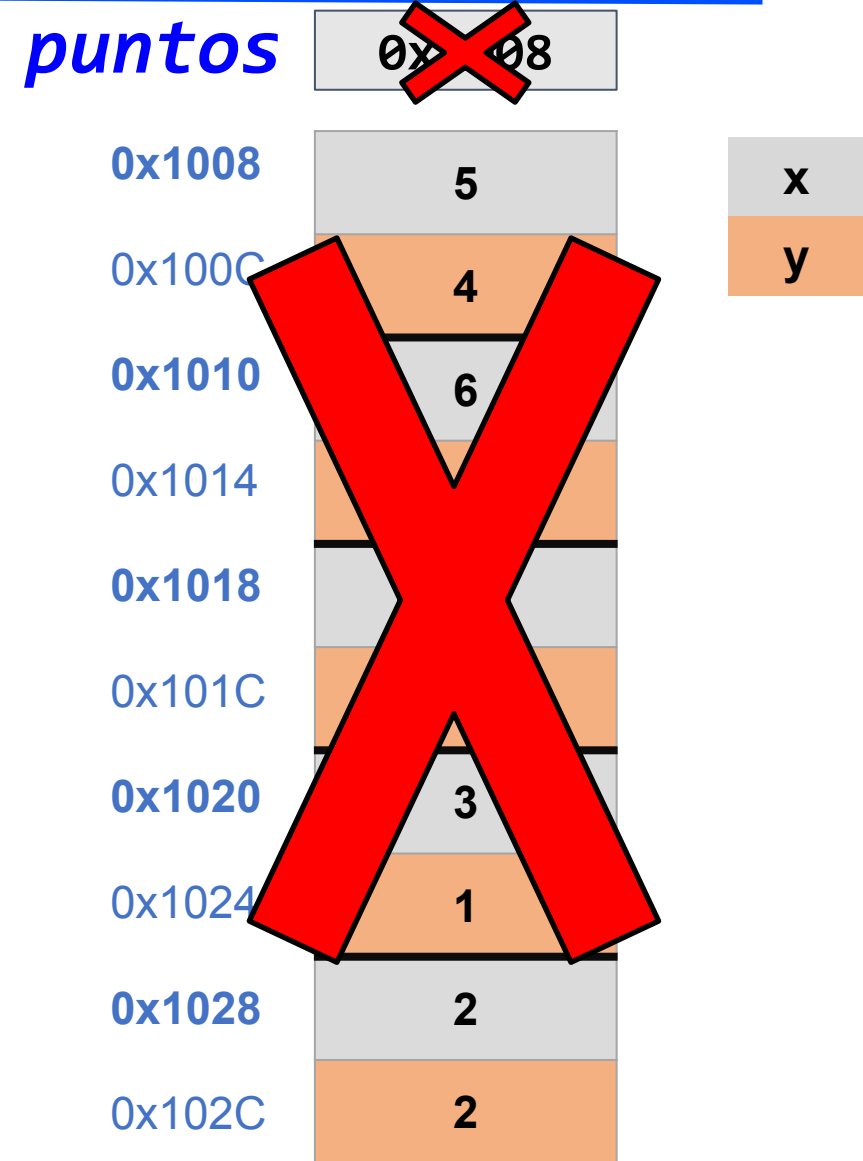
```
int main() {  
    CPunto2D* puntos = new CPunto2D[5]{  
        CPunto2D(5,4),  
        CPunto2D(6,2),  
        CPunto2D(1,1),  
        CPunto2D(3,1),  
        CPunto2D(2,2)  
    };  
  
    delete[] puntos;  
  
    return 0;  
}
```

*puntos*



# Arreglo de objetos: liberación de memoria

```
int main() {  
    CPunto2D* puntos = new CPunto2D[5]{  
        CPunto2D(5,4),  
        CPunto2D(6,2),  
        CPunto2D(1,1),  
        CPunto2D(3,1),  
        CPunto2D(2,2)  
    };  
  
    delete[] puntos;  
    puntos = nullptr;  
  
    return 0;  
}
```



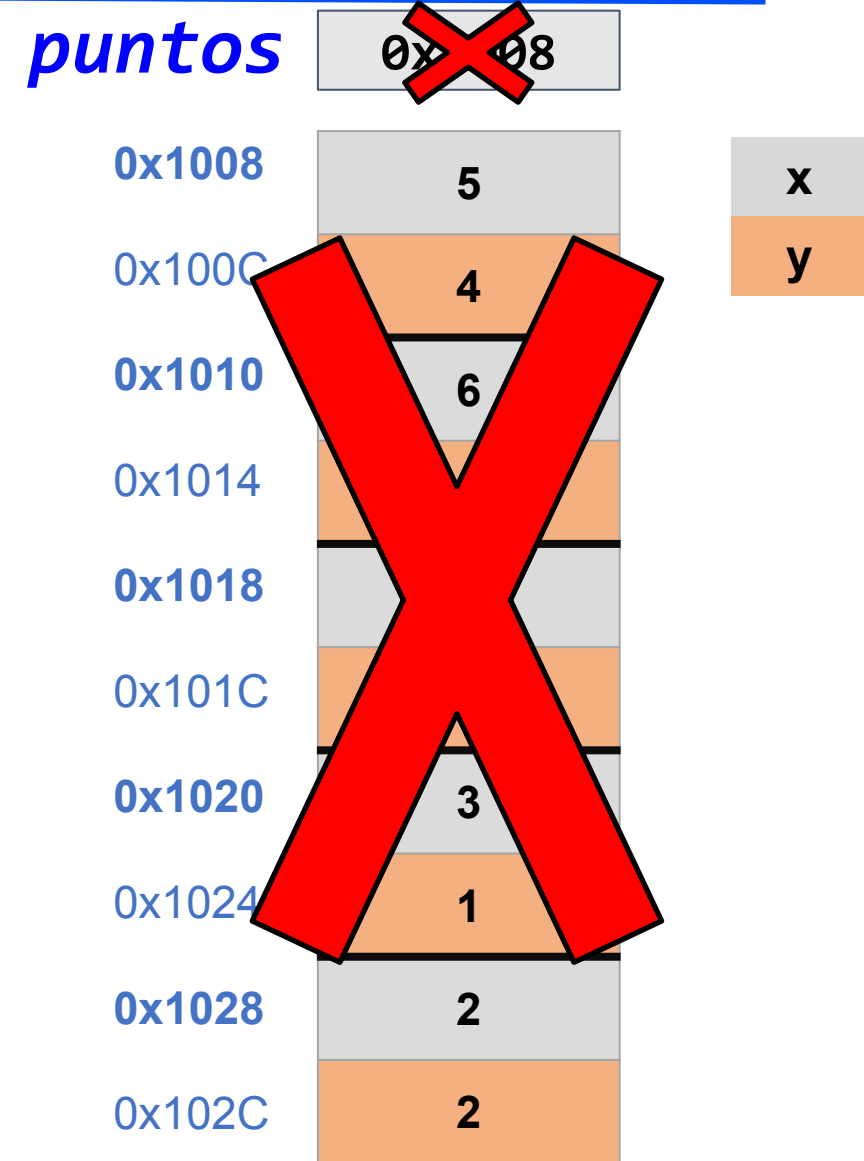


# Usando vectores

---

# vector de objetos: la liberación de memoria es automatica

```
int main() {  
    vector<CPunto2D> puntos;  
    puntos = {  
        CPunto2D(5,4),  
        CPunto2D(6,2),  
        CPunto2D(1,1),  
        CPunto2D(3,1),  
        CPunto2D(2,2) };  
    return 0;  
}
```



Tema opcional:  
Arreglo dinámico  
de punteros dobles a objetos (matrices)

---

## Arreglo de objetos: asignación dinámica de memoria

```
int main() {
    CPunto2D** puntos = nullptr;

    return 0;
}
```

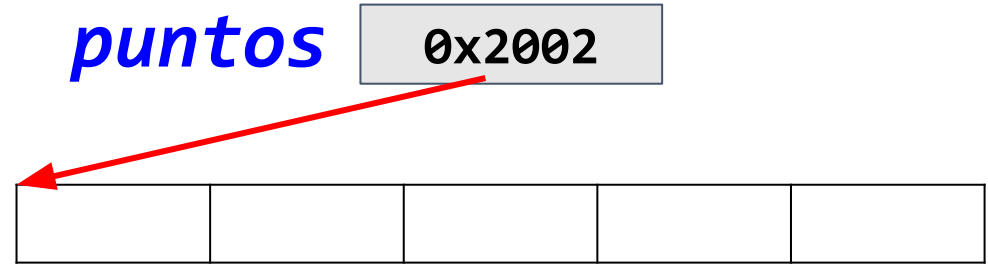
**puntos**



## Arreglo de objetos: asignación dinámica de memoria

```
int main() {
    CPunto2D** puntos = nullptr;
    puntos = new CPunto2D*[5];

    return 0;
}
```



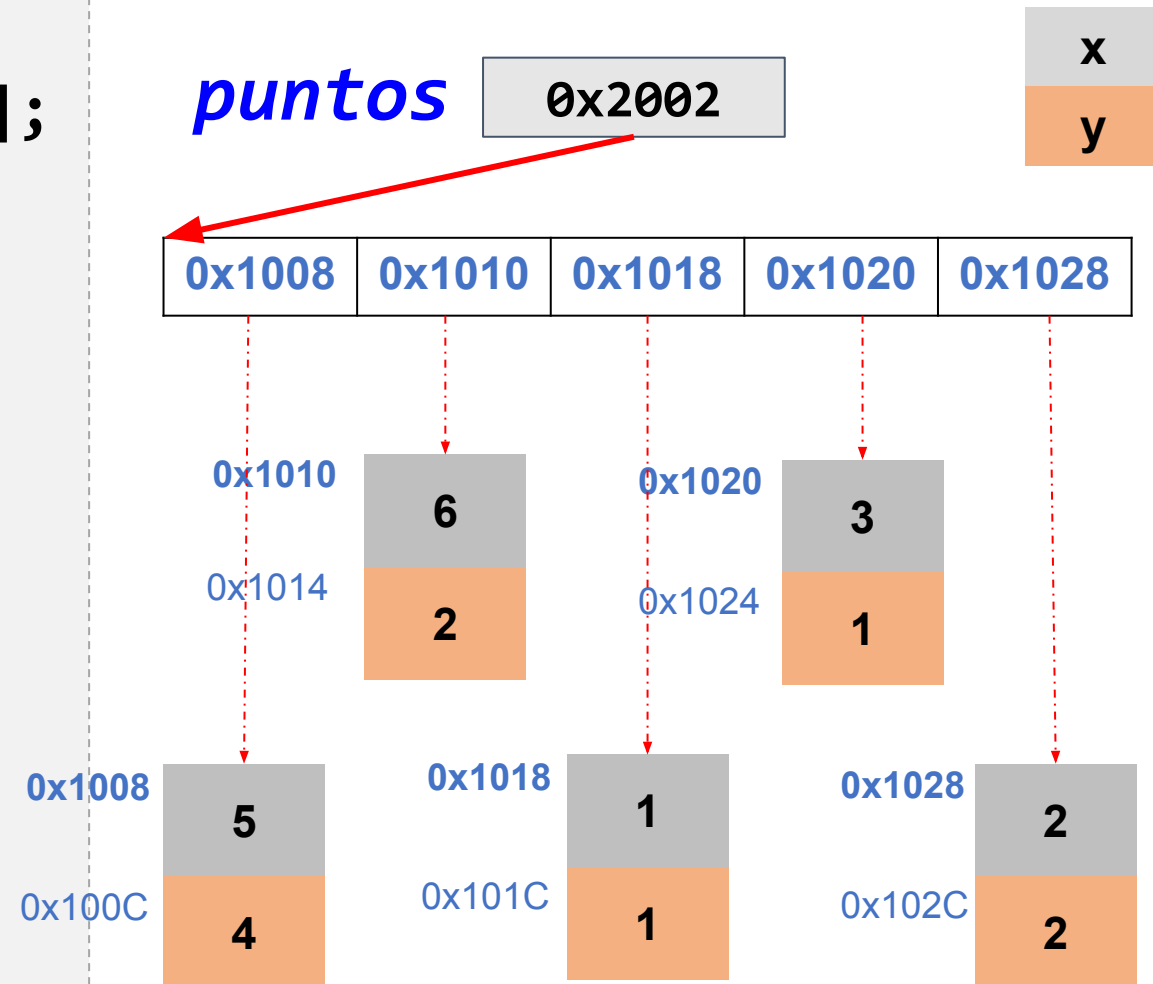
**Se crea un arreglo de 5 punteros.  
Un puntero en Clion utiliza 8 bytes.**

# Arreglo de objetos: asignación dinámica de memoria

```
int main() {
    CPunto2D** puntos = new CPunto2D*[5];
    puntos[0] = new CPunto2D(5,4);
    puntos[1] = new CPunto2D(6,2);
    puntos[2] = new CPunto2D(1,1);
    puntos[3] = new CPunto2D(3,1);
    puntos[4] = new CPunto2D(2,2);

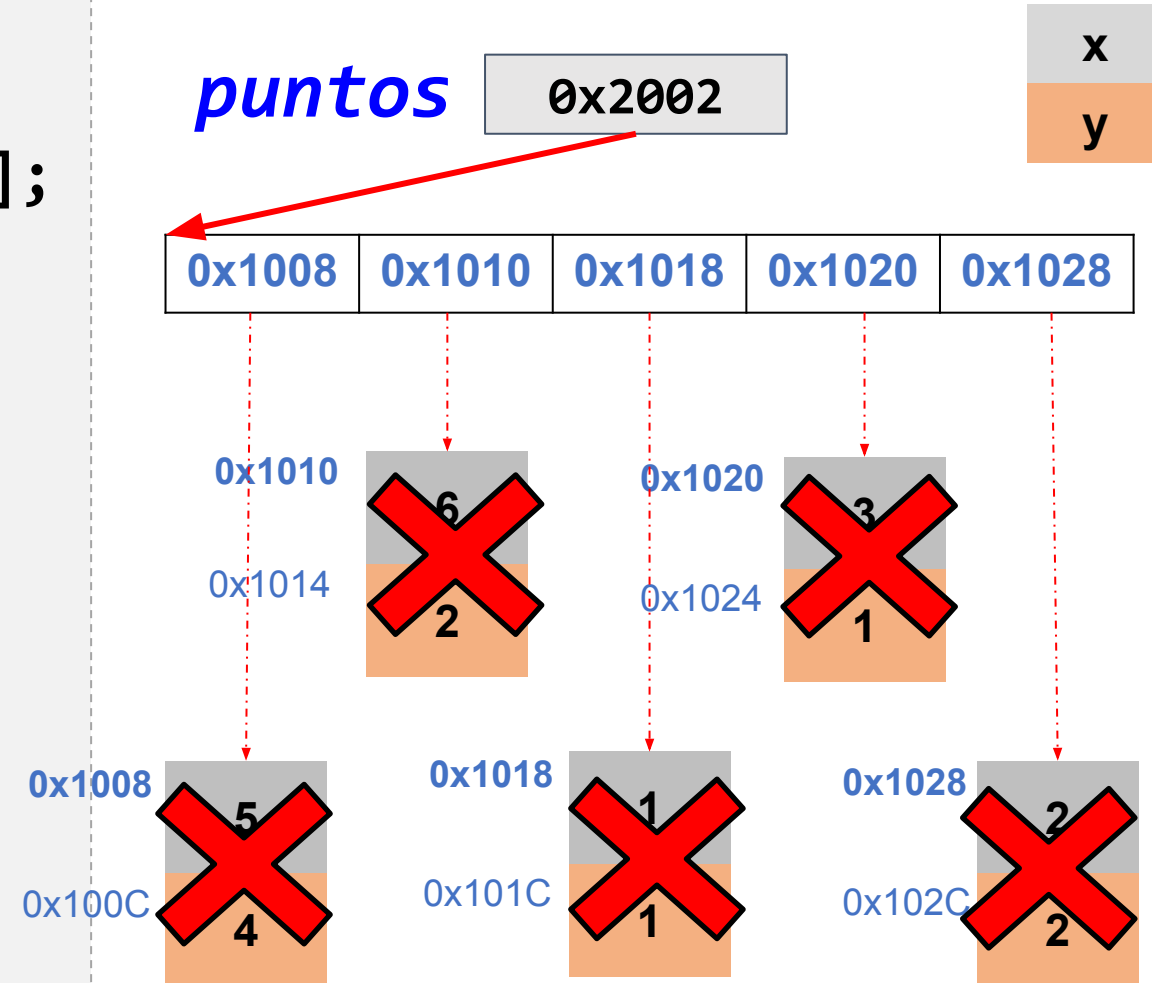
    for (int i = 0; i < 5; ++i) {
        cout<<puntos[i]->getX()<<" , "
            <<puntos[i]->getY()<<endl;
    }

    return 0;
}
```



# Arreglo de objetos: liberación memoria

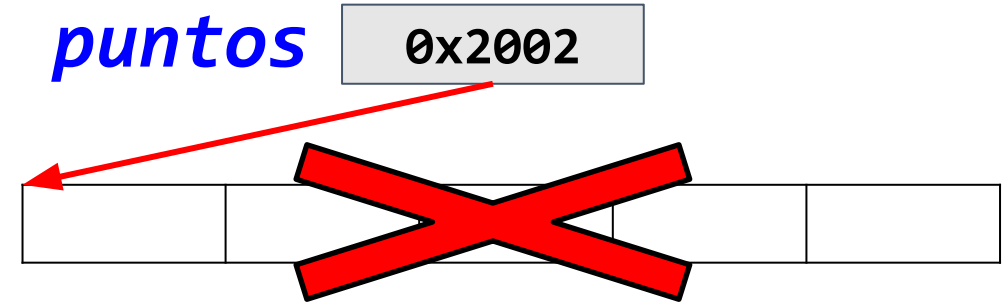
```
int main() {  
    CPunto2D** puntos = new CPunto2D*[5];  
    puntos[0] = new CPunto2D(5,4);  
    puntos[1] = new CPunto2D(6,2);  
    puntos[2] = new CPunto2D(1,1);  
    puntos[3] = new CPunto2D(3,1);  
    puntos[4] = new CPunto2D(2,2);  
  
    for (int i = 0; i < 5; ++i)  
        delete puntos[i];  
  
    return 0;  
}
```





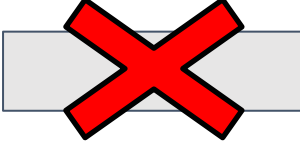
# Arreglo de objetos: liberación memoria

```
int main() {  
    CPunto2D** puntos = new CPunto2D*[5];  
    puntos[0] = new CPunto2D(5,4);  
    puntos[1] = new CPunto2D(6,2);  
    puntos[2] = new CPunto2D(1,1);  
    puntos[3] = new CPunto2D(3,1);  
    puntos[4] = new CPunto2D(2,2);  
  
    for (int i = 0; i < 5; ++i)  
        delete puntos[i];  
    delete[] puntos;  
    return 0;  
}
```



# Arreglo de objetos: liberación memoria

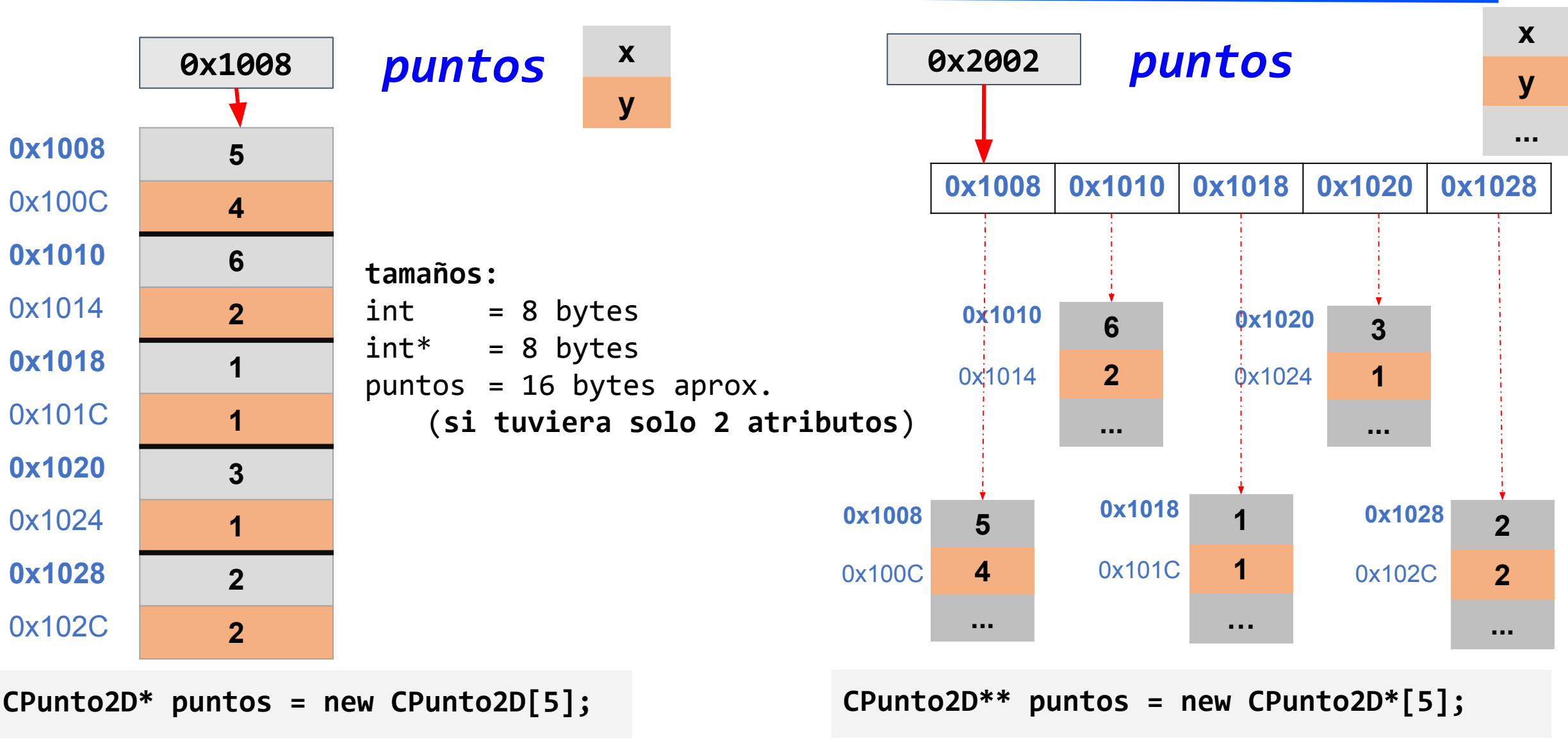
```
int main() {  
    CPunto2D** puntos = new CPunto2D*[5];  
    puntos[0] = new CPunto2D(5,4);  
    puntos[1] = new CPunto2D(6,2);  
    puntos[2] = new CPunto2D(1,1);  
    puntos[3] = new CPunto2D(3,1);  
    puntos[4] = new CPunto2D(2,2);  
  
    for (int i = 0; i < 5; ++i)  
        delete puntos[i];  
    delete[] puntos;  
    puntos = nullptr;  
    return 0;  
}
```

*puntos* 

¿Qué ventajas y desventajas tiene un arreglo con punteros dobles?

---

# Arreglo con puntero simple vs puntero doble



# Ventajas y Desventajas de arreglo de punteros a punteros

---

## **Ventaja:**

- Reduce la manipulación innecesaria de objetos existentes
- Aunque requiere más memoria, inicialmente solo requiere espacio suficiente para almacenar punteros.
- Facilita la adición de nuevos de objetos, reduciendo el uso de memoria y tiempo en el proceso
- Permite mover o compartir un objeto entre más de un arreglo fácilmente.

## **Desventajas:**

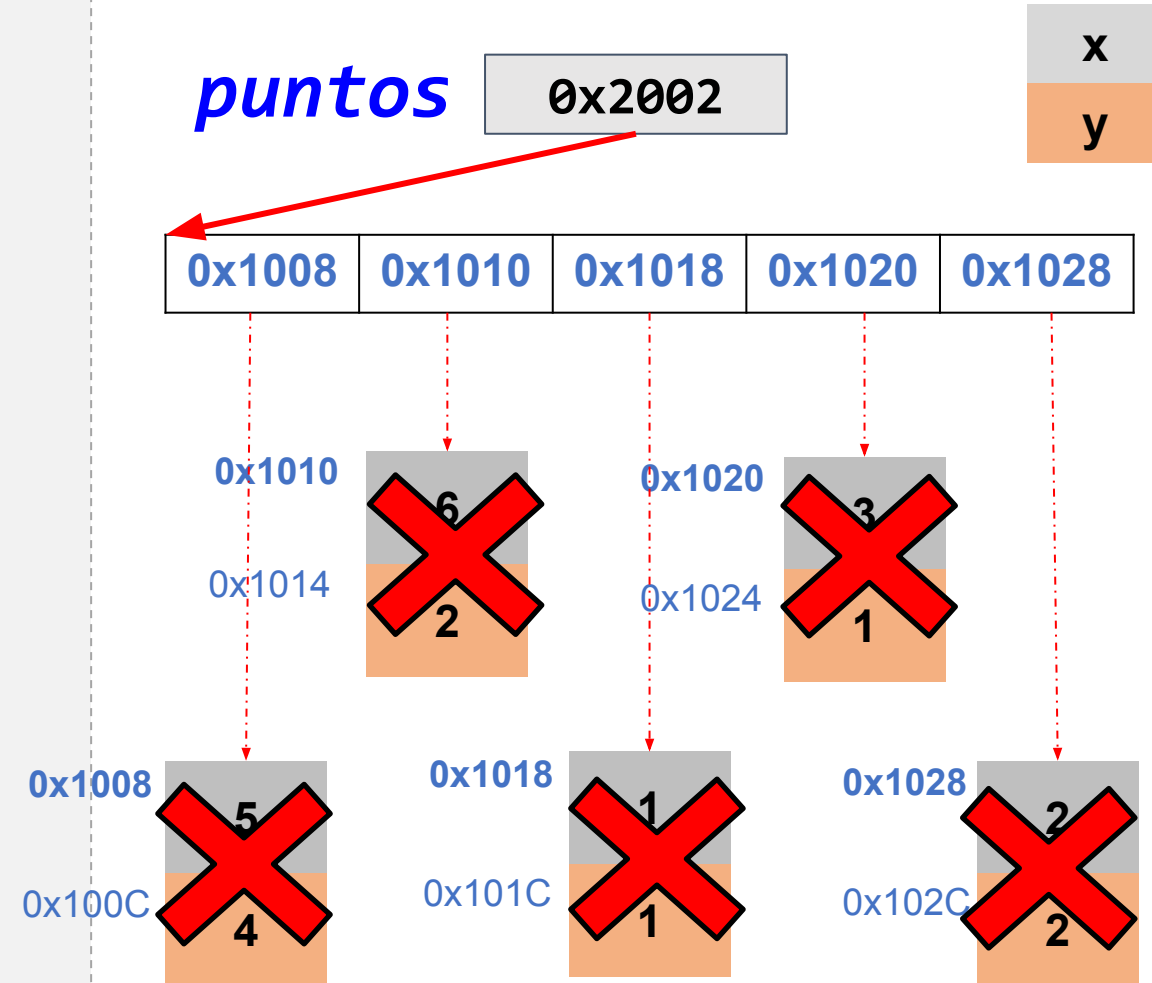
- Requiere de memoria adicional para almacenar de una lista de punteros.
- Es un proceso más complejo que requiere no solo liberar los objetos sino la lista de punteros.

# Usando vectores

---

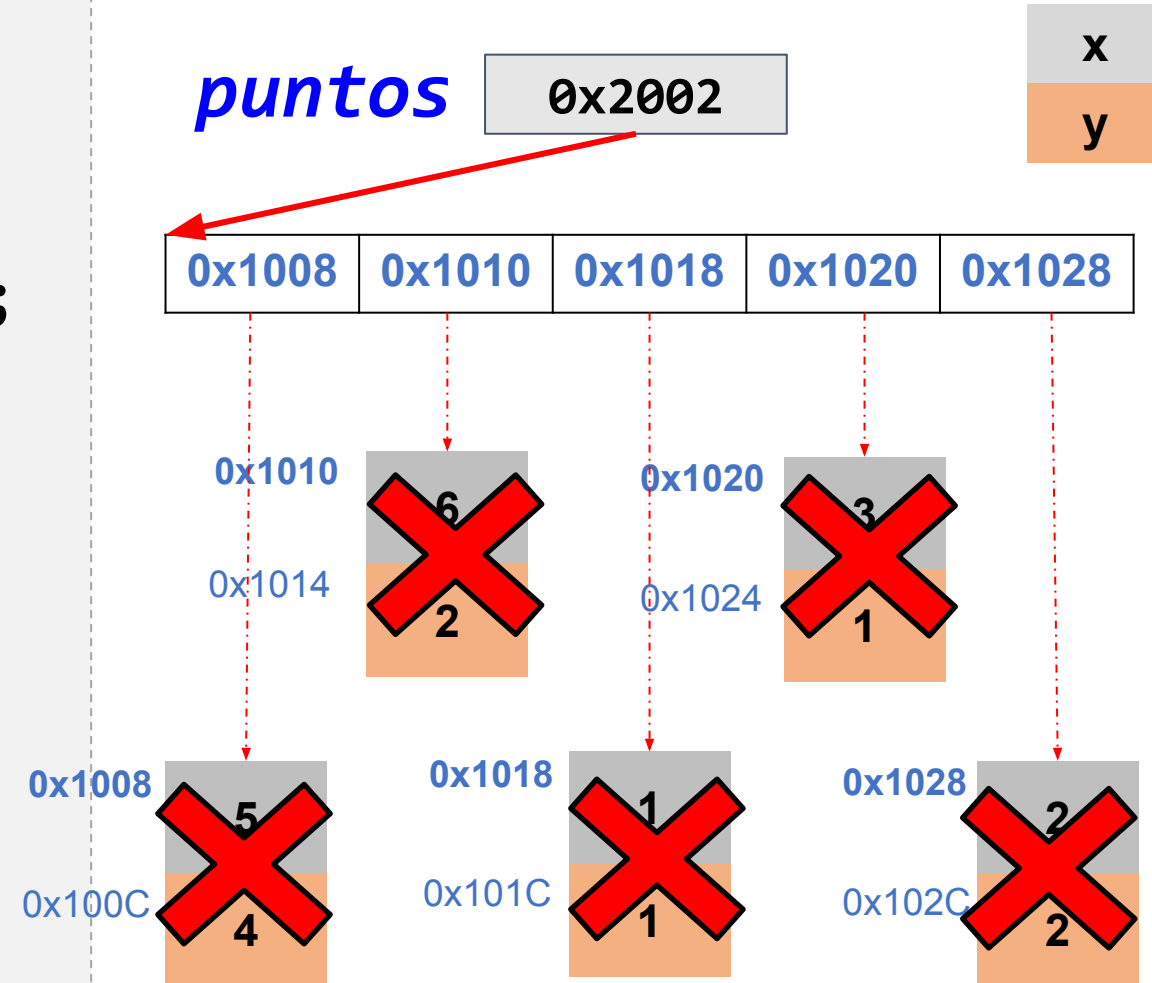
# vector de puntero a objetos: liberación semi-automática

```
int main() {  
    vector<CPunto2D*> puntos;  
    puntos {  
        new CPunto2D(5,4),  
        new CPunto2D(6,2),  
        new CPunto2D(1,1),  
        new CPunto2D(3,1),  
        new CPunto2D(2,2) };  
  
    for (int i = 0; i < 5; ++i)  
        delete puntos[i];  
  
    return 0;  
}
```



# vector de puntero inteligentes a objetos: liberación automática

```
int main() {  
    vector<shared_ptr<CPunto2D>> puntos;  
    puntos {  
        make_shared<CPunto2D>(5,4),  
        make_shared<CPunto2D>(6,2),  
        make_shared<CPunto2D>(1,1),  
        make_shared<CPunto2D>(3,1),  
        make_shared<CPunto2D>(2,2) };  
    return 0;  
}
```





¡Nos vemos en la  
siguiente clase!

**UTEC**  
UNIVERSIDAD DE INGENIERÍA  
Y TECNOLOGÍA

