

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Destruyctores

Los destruyctores también existen en Java pero no se utilizan apenas porque no es necesario liberar memoria dinámica de manera manual.

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Recordatorio: clase Fecha

```
class Fecha {  
public:  
    Fecha(int dia, int mes, int anyo);  
    Fecha(int anyo);  
    Fecha();  
  
    int get_dia() const;  
    void set_dia(int dia);  
    int get_mes() const;  
    void set_mes(int mes);  
    int get_anyo() const;  
    void set_anyo(int anyo);  
    void imprimir();  
  
private:  
    int dia;  
    int mes;  
    int anyo;  
};
```

Los destructores también existen en Java pero no se utilizan apenas porque no es necesario liberar memoria dinámica de manera manual.

Lo de siempre la clase fecha

Recordatorio: clase Persona

La clase persona que creo que ha aparecido alguna vez por ahí

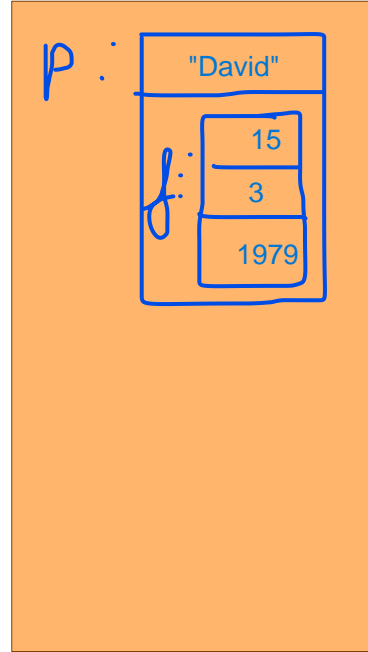
```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : nombre(nombre), fecha_nacimiento(dia, mes, anyo) { }  
        constructor  
private:  
    std::string nombre;  
    Fecha fecha_nacimiento; dos atributos  
};
```



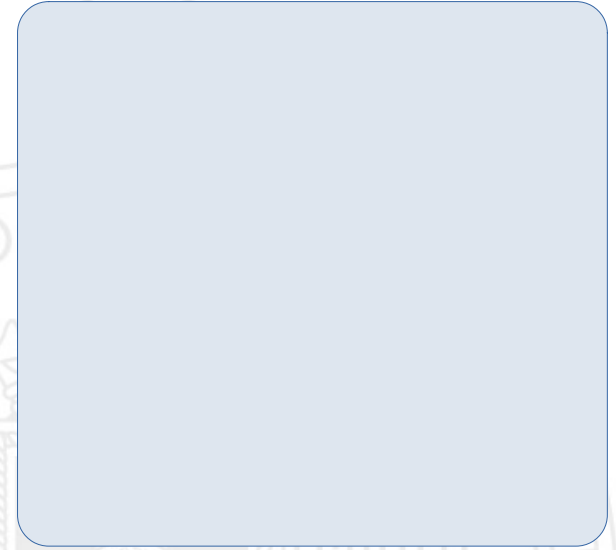
Ejemplo de uso

```
void ejemplo() {  
    Persona p("David", 15, 3, 1979);  
}
```

Pila



Heap



Cambio en la representación

Puede ser necesario, puede ser innecesario, puede ser incluso poco acertado, pero le sirve para explicar la existencia de los destructores

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : nombre(nombre) {  
        this->fecha_nacimiento = new Fecha(dia, mes, anyo);  
    }  
    como ahora es un puntero se inicializa con un new
```

```
private:  
    std::string nombre;  
    Fecha *fecha_nacimiento;  
};
```

Antes era un objeto fecha, ahora es un puntero a un objeto de tipo fecha

Cambio en la representación

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : nombre(nombre), fecha_nacimiento(new Fecha(dia, mes, anyo)) { }  
  
private:  
    std::string nombre;  
    Fecha *fecha_nacimiento;  
};
```

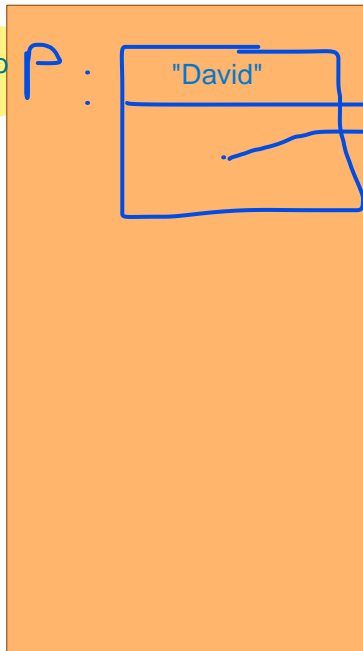
se podría utilizar una lista de inicialización

Ejemplo de uso

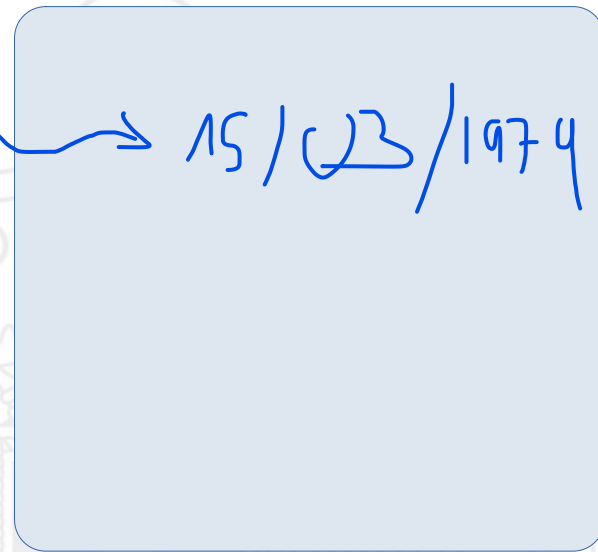
```
void ejemplo() {  
    Persona p("David", 15, 3, 1979);  
}
```

¿delete p->fecha_nacimiento? No porque el atributo es privado
no podemos acceder a él desde fuera de la clase.

Pila




Heap



Necesitamos una manera de
eliminar el objeto Fecha justo
antes de que p salga
de ámbito

Destruyores en C++

- Un **destructor** es un método especial que es invocado cada vez que el objeto correspondiente se libera.
 - Si el objeto está en la pila, el destructor es invocado cuando la variable que contiene dicho objeto sale de ámbito.
 - Si el objeto está en el *heap*, el destructor es invocado cuando se aplica `delete` sobre el objeto.
- El nombre del método destructor es el mismo que el de la clase en el que está definido, pero anteponiendo el símbolo `~`. 
- El método destructor no tiene ni parámetros, ni tipo de retorno.

Añadiendo un destructor a Persona

```
class Persona {  
public:  
    Persona(std::string nombre, int dia, int mes, int anyo)  
        : fecha_nacimiento(new Fecha(dia, mes, anyo)) { }
```

```
~Persona() {  
    delete fecha_nacimiento;  
}
```

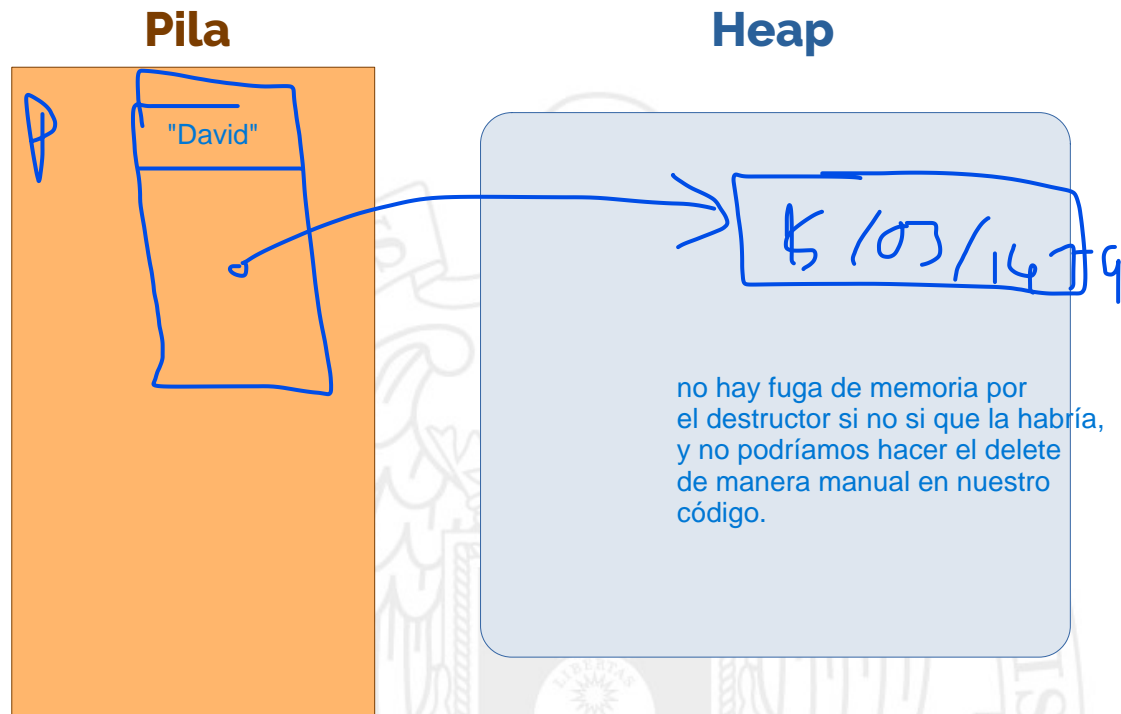
Método destructor

```
private:  
    std::string nombre;  
    Fecha *fecha_nacimiento;  
};
```

delete sobre el atributo fecha de nacimiento

Ejemplo de uso

```
void ejemplo() {  
    Persona p("David", 15, 3, 1979);  
  
    después se llama al destructor  
}
```



Resource acquisition is initialization (RAII)

Es un patrón

- En la gran mayoría de casos, la reserva de memoria (new) que se realice en el constructor debe tener asociada su liberación (delete) en el destructor.
- Excepciones:
 - La memoria reservada se ha liberado antes de invocar el destructor.
 - La memoria reservada está compartida entre varias instancias.
- El principio RAII no solo se aplica a memoria, sino también a otros recursos (apertura/cierre de ficheros, conexiones a bases de datos, etc.)