FACULTAD DE INFORMÁTICA

Introducción a la POO
Clases y objetos

TALLER

Programación orientada a objetos — Unidad 1

Autor: Luis Hernández Yáñez

FdI UCM

Cuestiones

¿Por qué? ¿Cómo? ¿Cuándo? (*resolución individual*)

¿Por qué es más correcto implementar todos los métodos fuera de la estructura **class**?

¿En qué casos se te ocurre que no sería adecuado incluir en una clase ciertos mutadores? ¿Y ciertos accedentes?

¿En qué casos crees que los pasos de mensajes no se escribirían con la notación punto (*receptor•mensaje*)?

¿Por qué en los métodos para los pasos de mensajes al propio objeto receptor no se coloca delante el objeto receptor y el punto?

FdI UCM

Objetos dentro de objetos y encadenamiento de mensajes

¿Qué aparecerá en la pantalla? (*resolución individual*)

Dadas las clases que se muestran a continuación, averigua qué se mostrará en la pantalla con el programa principal del final.

- Archivo de cabecera **claseA.h**:

```
class ClaseA {
public:
  void init(int = 0);
  int mas(int = 0);
  int por(int = 1);
private:
  int _a;
};
```

- Archivo de implementación **claseA.cpp**:

```
#include "claseA.h"
// Están definidos argumentos implícitos
void ClaseA::init(int val) { _a = val; }
int ClaseA::mas(int val) { return _a + val; }
int ClaseA::por(int val) { return _a * val; }
```

. . . / . . .

FdI UCM

Objetos dentro de objetos y encadenamiento de mensajes

- Archivo de cabecera **claseB.h**:

```
#include "claseA.h"
class ClaseB {
public:
  void init(int = 0);
  int mas(int = 0);
  int por(int = 1);
  void igual(ClaseA);
private:
  ClaseA _obj;
};
```

- Archivo de implementación **claseB.cpp**:

```
#include "claseB.h"
// Están definidos argumentos implícitos
void ClaseB::init(int val) { _obj.init(val); }
int ClaseB::mas(int val) { return _obj.mas(val); }
int ClaseB::por(int val) { return _obj.por(val); }
void ClaseB::igual(ClaseA otro) { _obj = otro; }
```

. . . / . . .

Objetos dentro de objetos y encadenamiento de mensajes

- Archivo de cabecera **claseC.h**:

```
#include "claseA.h"
#include "claseB.h"
class ClaseC {
public:
  void init(int = 0, int = 0);
  int mas();
  int por();
  void obj1(ClaseA);
  void obj2(ClaseB);
private:
  ClaseA _obj1;
  ClaseB _obj2;
};
```

- Archivo de implementación **claseC.cpp**:

```
#include "claseC.h"

void ClaseC::init(int val1, int val2) // Argumentos implícitos
{ _obj1.init(val1); _obj2.init(val2); }
```

. . . / . . .

Objetos dentro de objetos y encadenamiento de mensajes

```
int ClaseC::mas() { return _obj1.mas() + _obj2.mas(); }
int ClaseC::por() { return _obj1.por() * _obj2.por(); }
void ClaseC::obj1(ClaseA obj) { _obj1 = obj; }
void ClaseC::obj2(ClaseB obj) { _obj2 = obj; }
```

- Programa principal:

```
#include <iostream>
using namespace std;
#include "claseA.h"
#include "claseB.h"
#include "claseC.h"

int main()
{
  ClaseA objA;
  objA.init(5);
  cout << objA.por(5) << endl; // endl provoca un salto de línea
  ClaseB objB;
  objB.init(10);
  cout << objB.por(5) + objA.mas(3) << endl;
```

. . . / . . .

Objetos dentro de objetos y encadenamiento de mensajes

```
  ClaseC objC;
  objC.init(12, 7);
  cout << objC.mas() << endl;
  cout << objC.por() << endl;
  objB.igual(objA);
  objC.obj1(objA);
  objC.obj2(objB);
  cout << objC.mas() << endl;
  cout << objC.por() << endl;

  return 0;
}
```

Te habrás fijado que has tenido que seguir el encadenamiento de los mensajes que se produce al ejecutarse el código de los métodos.

Ejercicio práctico

Una primera clase (*resolución en grupo*)

1. Crear una clase que se denomine **Contador1** que defina un único atributo **_cont** (un entero).
   El contador se podrá inicializar a cualquier valor no negativo.
   Otros servicios que han de proporcionar los objetos de esta clase son: incrementar el contador (en una unidad), decrementar el contador (en una unidad) y mostrar el valor actual del contador.
   Implementa todos los métodos dentro de la estructura **class**.
   El contador nunca podrá tener un valor negativo.
   Prueba la clase en una función **main()**.

Ejercicio práctico

Cada cosa en su sitio (*resolución en grupo*)

2. A partir de la clase anterior, crea otra clase **`Contador2`** que sea igual que la anterior, pero que tenga todos los métodos implementados fuera de la estructura **`class`**.
   Prueba la clase en una función **`main()`**.
   ¿Qué has tenido que cambiar en la función **`main()`**?
   Compara la interfaz de esta clase y la de la anterior: ¿cuál queda más clara?

   Lo que hay que hacer:
   1. Copiar los métodos a continuación de la estructura **`class`**.
   2. *Limpiar* las cabeceras dentro de la estructura **`class`** para que sean prototipos (sustituir el cuerpo por ; y quitar los nombres de los parámetros).
   3. Fuera de la estructura **`class`**, añadir el nombre de la clase y el operador de resolución de ámbito delante de los nombres de las funciones y reformatear el código.

En el mundo real

Una clase más útil (*resolución en grupo*)

Crear una clase **`Empleado`** que modele la información que una empresa mantiene sobre cada empleado: número de DNI (entero largo), sueldo base (real), pago por hora extra (real), horas extra realizadas en el mes (real), tipo (porcentaje) de IRPF (real), casado o no (verdadero/falso) y número de hijos (entero).
La clase debe contemplar accedentes y mutadores para todos los atributos menos mutador para el correspondiente al DNI.
Al inicializar cada objeto se podrá proporcionar el DNI correspondiente. Los demás servicios que deberán proporcionar los objetos de la clase serán los siguientes:

Cálculo y devolución del complemento correspondiente a las horas extra realizadas.

Cálculo y devolución del sueldo bruto.

. . . / . . .

En el mundo real

Cálculo y devolución de las retenciones (IRPF) a partir del tipo, teniendo en cuenta que el porcentaje de retención que hay que aplicar es el tipo menos 2 puntos si el empleado está casado y menos 1 punto por cada hijo que tenga; el porcentaje se aplica sobre todo el sueldo bruto.

Visualización de la información básica del empleado.

Visualización de toda la información del empleado. la básica más el sueldo base, el complemento por horas extra, el sueldo bruto, la retención de IRPF y el sueldo neto.

Todos los métodos se han de implementar fuera de la estructura **`class`**. Prueba la clase en una función **`main()`**.