FACULTAD DE INFORMÁTICA

Herencia

SOLUCIONES DEL TALLER

Programación orientada a objetos — Unidad 6

Autor: Luis Hernández Yáñez

La clase **Empleado** otra vez

Mejora de la clase (*resolución en grupo*)

En talleres anteriores trabajamos con una clase **Empleado** que creamos a partir de unas necesidades que teníamos en ese momento:

```
class Empleado {
public:
  Empleado(long int = 0, bool = false, int = 0, double = 0,
           double = 0, double = 0, double = 0);
  // Datos: DNI, ¿casado?, hijos, sueldo base, tipo IRPF,
  //        pago por hora extra, horas extra realizadas.
  Empleado(const Empleado&);
  Empleado& operator=(const Empleado&);
  ~Empleado();
  long int dni() const;
  bool casado() const;
  void casado(bool);
  int hijos() const;
  void hijos(int);
  double sueldoBase() const;
  void sueldoBase(double);
```

(*continúa*)

La clase **Empleado** otra vez

```
  double tipo() const;
  void tipo(double);
  double pagoHoraExtra() const;
  void pagoHoraExtra(double);
  double horasExtra() const;
  void horasExtra(double);
  double complemento() const;
  double sueldoBruto() const;
  double retenciones() const;
  void mostrar() const;
  void mostrarTodo() const;
  void leer();
private:
  long int _dni;
  bool _casado;
  int _hijos;
  double _sueldoBase;
  double _tipo; // porcentaje de IRPF para impuestos
  double _pagoHoraExtra;
  double _horasExtra;
};
```

Como los empleados son personas, se trata de modificar la clase **Empleado**, de forma que refleje ese hecho.
Como la clase **Persona** ya dispone de NIF, elimina el atributo **_dni** y todo lo que tenga que ver con él.

Comprobad lo fácil o difícil que os ha resultado la adaptación.

La clase **Empleado** otra vez

Para que los objetos de la clase **Empleado** adopten todas las características de la clase **Persona** basta con hacer que **Empleado** sea una subclase de **Persona**.

```
class Empleado : public Persona { ...
```

A continuación eliminamos todo lo que tenga que ver con el atributo **_dni**. Además del atributo en sí, tenemos que quitar su accedente (no había mutador) y el primer parámetro del constructor.

Aprovechamos, además, para completar el constructor, ya que ahora debe poder recibir los datos heredados de **Persona**.

La clase **Empleado** otra vez

```
// Clase Empleado - Archivo de cabecera "empleado.h"
#ifndef empleado_h // Evitar inclusiones múltiples
#define empleado_h
#include "persona.h"

class Empleado : public Persona {
public:
  Empleado(string = "", int = 0, string = "", string = "",
           bool = false, int = 0, double = 0, double = 0,
           double = 0, double = 0);
  // Datos: NIF, edad, nombre, apellidos, ¿casado?, hijos,
  // sueldo base, tipo IRPF, pago hora extra y horas extra.
  Empleado(const Empleado&);
  Empleado& operator=(const Empleado&);
  ~Empleado();
  bool casado() const;
  void casado(bool);
  int hijos() const;
  void hijos(int);
  double sueldoBase() const;
  void sueldoBase(double);
```

(*continúa*)

La clase **Empleado** otra vez

```
  double tipo() const;
  void tipo(double);
  double pagoHoraExtra() const;
  void pagoHoraExtra(double);
  double horasExtra() const;
  void horasExtra(double);
  double complemento() const;
  double sueldoBruto() const;
  double retenciones() const;
  void mostrar() const;
  void mostrarTodo() const;
  void leer();
private:
  bool _casado;
  int _hijos;
  double _sueldoBase;
  double _tipo; // porcentaje de IRPF para impuestos
  double _pagoHoraExtra;
  double _horasExtra;
};
#endif
```

>>> empleado.h

La clase **Empleado** otra vez

En la implementación debemos realizar los siguientes cambios:

- ✓ Quitar el accedente del antiguo atributo **_dni**, así como sus usos en el constructor de copia, el operador de asignación, y los métodos **mostrar()** y **leer()**.
- ✓ Reimplementar el constructor predeterminado.
- ✓ Hacer que el constructor de copia y el operador de asignación usen los respectivos de la superclases para que se copien los atributos heredados de **Persona**.
- ✓ Hacer que el método **mostrar()** use el redefinido de la superclase para que se muestren los atributos heredados.
- ✓ Hacer que el método **leer()** use el redefinido de la superclase para que se lean los atributos heredados.

- La adaptación de la clase ha resultado bastante sencilla.

La clase **Empleado** otra vez

```
// Clase Empleado - Archivo de implementación "empleado.cpp"
#include <iostream>
using namespace std;

#include "empleado.h"

Empleado::Empleado(string nif, int edad, string nombre,
  string apellidos, bool casado, int hijos, double sueldo,
  double tipo, double pago, double extras)
  : Persona(nif, edad, nombre, apellidos), _casado(casado),
  _hijos(hijos), _sueldoBase(sueldo), _tipo(tipo),
  _pagoHoraExtra(pago), _horasExtra(extras) { }

Empleado::Empleado(const Empleado& otro) : Persona(otro) {
  _casado = otro._casado;
  _hijos = otro._hijos;
  _sueldoBase = otro._sueldoBase;
  _tipo = otro._tipo;
  _pagoHoraExtra = otro._pagoHoraExtra;
  _horasExtra = otro._horasExtra;
}
```

(*continúa*)

La clase **Empleado** otra vez

```
Empleado& Empleado::operator=(const Empleado& otro) {
  Persona::operator=(otro);
  _casado = otro._casado;
  _hijos = otro._hijos;
  _sueldoBase = otro._sueldoBase;
  _tipo = otro._tipo;
  _pagoHoraExtra = otro._pagoHoraExtra;
  _horasExtra = otro._horasExtra;
  return *this;
}

Empleado::~Empleado() { }

bool Empleado::casado() const { return _casado; }

void Empleado::casado(bool c) { _casado = c; }

int Empleado::hijos() const { return _hijos; }

void Empleado::hijos(int h) { _hijos = h; }
```

(*continúa*)

La clase **Empleado** otra vez

```
double Empleado::sueldoBase() const { return _sueldoBase; }

void Empleado::sueldoBase(double sb) { _sueldoBase = sb; }

double Empleado::tipo() const { return _tipo; }

void Empleado::tipo(double t) { _tipo = t; }

double Empleado::pagoHoraExtra() const { return _pagoHoraExtra;}

void Empleado::pagoHoraExtra(double phe) {_pagoHoraExtra = phe;}

double Empleado::horasExtra() const { return _horasExtra; }

void Empleado::horasExtra(double he) { _horasExtra = he; }

double Empleado::complemento() const {
  return _pagoHoraExtra * _horasExtra; }

double Empleado::sueldoBruto() const {
  return _sueldoBase + complemento(); }
```

(*continúa*)

La clase **Empleado** otra vez

```
double Empleado::retenciones() const {
// El tipo siempre es suficientemente alto, por lo que
// nunca se aplicará un tipo final negativo
  double tipoFinal = _tipo;
  if(_casado) tipoFinal -= 2;
  tipoFinal -= _hijos;
  return sueldoBruto() * tipoFinal / 100;
}

void Empleado::mostrar() const {
  cout << endl;
  Persona::mostrar();
  cout << "Casado: " << (_casado ? "Si" : "No") << endl;
  cout << "Hijos: " << _hijos << endl;
  cout << "Sueldo base: " << _sueldoBase << endl;
  cout << "Porcentaje IRPF: " << _tipo << endl;
  cout << "Pago por hora extra: " << _pagoHoraExtra << endl;
  cout << "Horas extra realizadas: " << _horasExtra << endl;
}
```

(*continúa*)

La clase **Empleado** otra vez

```
void Empleado::mostrarTodo() const {
  mostrar(); // muestra la información básica
  cout << "Complemento horas extra: " << complemento() << endl;
  double sueldo = sueldoBruto();
  cout << "Sueldo bruto: " << sueldo << endl;
  double ret = retenciones();
  cout << "Retenciones I.R.P.F.: " << ret << endl;
  sueldo -= ret;
  cout << "Sueldo neto: " << sueldo << endl; }

void Empleado::leer()  {
  cout << "Introduzca los datos del empleado" << endl;
  Persona::leer();
  cout << "Casado? (s/n) ";
  char c; cin >> c;
  _casado = (c=='s') ? true : false;
  cout << "Hijos: "; cin >> _hijos;
  cout << "Sueldo base: "; cin >> _sueldoBase;
  cout << "Porcentaje IRPF: "; cin >> _tipo;
  cout << "Pago por hora extra: "; cin >> _pagoHoraExtra;
  cout << "Horas extra realizadas: "; cin >> _horasExtra; }
```

En el mundo real

Para una biblioteca (*resolución en grupo*)

Ya hemos creado anteriormente una clase **`Libro`** que modela la información que se mantiene en una biblioteca sobre cada libro.

La biblioteca no sólo va a contener libros, sino también otros tipos de publicaciones, las que indicamos a continuación junto con la información de la que se quiere disponer de cada una de ellas:

`Enciclopedia`: título, autor, ISBN, volúmenes, edición, editorial, lugar y fecha de edición.

`Revista`: título, volumen, número, editorial, ciudad, país y fecha.

`Actas`: título, congreso, volumen, número, ciudad, país y fecha.

`Artículo`: título, autor, revista (o actas), página inicial, página final, ciudad, país y fecha.

Tenéis que crear las clases necesarias, teniendo en cuenta la existencia de la clase **`Libro`** y las similitudes entre las clases.

En el mundo real

Lo primero que hemos de hacer es descubrir las clases que hay que crear. Y éstas no sólo vienen dadas por las que nos piden, sino que van a resultar necesarias clases intermedias de la jerarquía de clases.

Partimos de las clases concretas que nos piden:
`Libro`, **`Enciclopedia`**, **`Revista`**, **`Actas`** y **`Articulo`**.
(**`Libro`** es la que tenemos ya implementada, aunque la existencia de las nuevas y sus similitudes van a hacer que sea reimplementada).

Esas clases serán las clases *terminales* de la jerarquía (las hojas del árbol, las clases que no tienen subclases).

Las características comunes de esas clases se llevarán a nuevas clases intermedias de la jerarquía, clases de las que derivaremos las primeras.

Bastará con que nos centremos en los atributos de las clases.

En el mundo real

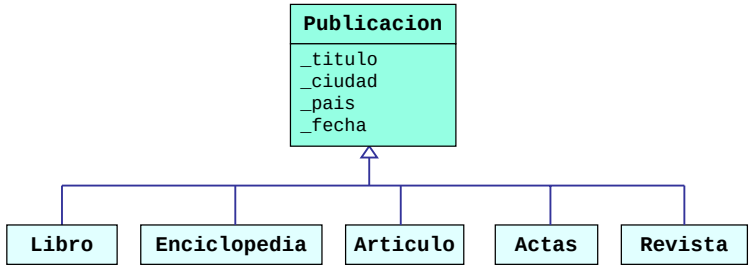
Elaboramos una relación de atributos de las clases de partida:

- ✓ **`Libro`**: título, autor, ISBN, edición, editorial, ciudad, país, fecha de edición y páginas.
- ✓ **`Enciclopedia`**: título, autor, ISBN, volúmenes, edición, editorial, ciudad, país y fecha de edición.
- ✓ **`Revista`**: título, volumen, número, editorial, ciudad, país y fecha.
- ✓ **`Actas`**: título, congreso, volumen, número, ciudad, país y fecha.
- ✓ **`Artículo`**: título, autor, revista (o actas), página inicial, página final, ciudad, país y fecha.

A continuación intentamos detectar atributos que haya en TODAS las clases. ¡Los hay! Son: título, ciudad, país y fecha de edición.

Entonces, creamos una nueva clase (por ejemplo, **`Publicacion`**), le pasamos a ella esos atributos comunes, los quitamos de las clases de partida y hacemos que éstas se deriven de la nueva clase.

En el mundo real



- ✓ **`Libro`**: autor, ISBN, edición, editorial y páginas.
- ✓ **`Enciclopedia`**: autor, ISBN, volúmenes, edición y editorial.
- ✓ **`Revista`**: volumen, número y editorial.
- ✓ **`Actas`**: congreso, volumen y número.
- ✓ **`Artículo`**: autor, revista (o actas), página inicial y página final.

En el mundo real

- ✓ **Libro**: autor, ISBN, edición, editorial y páginas.
- ✓ **Enciclopedia**: autor, ISBN, volúmenes, edición y editorial.
- ✓ **Revista**: volumen, número y editorial.
- ✓ **Actas**: congreso, volumen y número.
- ✓ **Artículo**: autor, revista (o actas), página inicial y página final.

Ya no quedan atributos comunes a todas las clases. Por tanto, ahora intentamos detectar grupos de clases con atributos comunes.

Aquí pueden empezar a surgir diferentes alternativas. Podemos distinguir dos grupos de clases en base a que tengan "autor" o no. O podemos distinguir dos grupos de clases en base a que tengan "ISBN" o no. O podemos distinguir dos grupos de clases en base a que tengan "editorial" o no. Y un largo etcétera.

Intentaremos que las clasificaciones resulten *creíbles* (*realistas*). Como suele ocurrir, la mejor guía es la experiencia.

En el mundo real

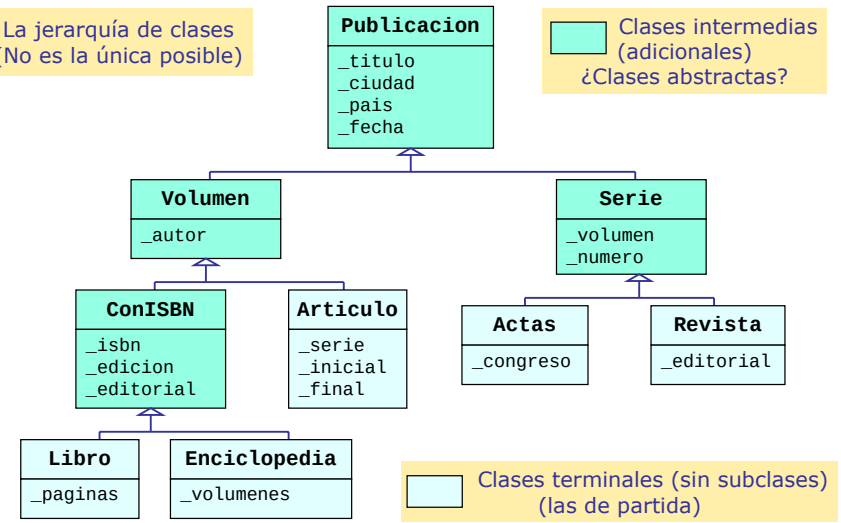
- ✓ **Libro**: autor, ISBN, edición, editorial y páginas.
- ✓ **Enciclopedia**: autor, ISBN, volúmenes, edición y editorial.
- ✓ **Revista**: volumen, número y editorial.
- ✓ **Actas**: congreso, volumen y número.
- ✓ **Artículo**: autor, revista (o actas), página inicial y página final.

Vemos que **Libro** y **Enciclopedia** tienen bastantes atributos en común (4), por lo que parece que estarán en un mismo grupo. Por otro lado, **Revista** y **Actas** tienen dos atributos en común, por lo que parece que también estarán en otro grupo. ¿Qué pasa con **Articulo**? No tiene ningún atributo en común con el segundo grupo y sí uno con el primer grupo.

Esto nos lleva a plantear una superclase para **Revista** y **Actas** (por ejemplo **Serie**), otra (por ejemplo **Volumen**) para **Articulo** y una nueva clase (por ejemplo, **ConISBN**) que sea superclase de **Libro** y **Enciclopedia**.

En el mundo real

La jerarquía de clases
(No es la única posible)



Clases intermedias (adicionales)
¿Clases abstractas?

Clases terminales (sin subclases)
(las de partida)

En el mundo real

Ahora ya podemos pasar a implementar todas esas clases.

El proceso es laborioso, pero en absoluto complicado.

Tomando como guía la implementación anterior de la clase **Libro**, vemos que los servicios que deben proporcionar las clases son los accedentes, los mutadores, el método **leer()** y el método **mostrar()**. Aparte, por supuesto, de ajustarse a la FCO.

Aquí mostraremos únicamente las interfaces de las clases (cabeceras **.h**). Las implementaciones son muy sencillas y se proporcionan en la web.

Eso sí, debemos aprovechar los métodos de las superclases donde podamos: constructor (pasarle argumentos), constructor de copia, operador de asignación, **leer()** y **mostrar()**.

La clase **Publicacion**

La clase raíz de la jerarquía

```
#ifndef publicacion_h
#define publicacion_h

#include <iostream>
#include <string>
using namespace std;
#include "fecha.h"

class Publicacion {
public:
  Publicacion(string = "", string = "", string = "",
              Fecha = Fecha());
  // Datos: título, ciudad, país, fecha de edición.
  Publicacion(const Publicacion&);
  Publicacion& operator=(const Publicacion&);
  ~Publicacion();

  string titulo() const;
  string ciudad() const;
  string pais() const;
```

(*continúa*)

La clase **Publicacion**

```
  Fecha fecha() const;

  void titulo(string);
  void ciudad(string);
  void pais(string);
  void fecha(Fecha);

  void leer();
  void mostrar() const;
private:
  string _titulo;
  string _ciudad, _pais;
  Fecha _fecha;
};

#endif
```


La clase **Volumen**

```
#ifndef volumen_h
#define volumen_h

#include <iostream>
#include <string>

using namespace std;
#include "persona.h"
#include "publicacion.h"

class Volumen : public Publicacion {
public:
  Volumen(string = "", string = "", string = "",
          Fecha = Fecha(), Persona = Persona());
  // Datos: título, ciudad, país, fecha de edición, autor.
  Volumen(const Volumen&);
  Volumen& operator=(const Volumen&);
  ~Volumen();
```

(*continúa*)

La clase **Volumen**

```
  Persona autor() const;
  void autor(Persona);

  void leer();
  void mostrar() const;
private:
  Persona _autor;
};

#endif
```

La clase ConISBN

```
#ifndef conisbn_h
#define conisbn_h

#include <iostream>
#include <string>
using namespace std;
#include "volumen.h"

class ConISBN : public Volumen {
public:
  ConISBN(string = "", string = "", string = "", Fecha =Fecha(),
          Persona = Persona(), string = "", int = 1, string="");
  // Datos: título, ciudad, país, fecha de edición, autor,
  //        ISBN, edición, editorial.
  ConISBN(const ConISBN&);
  ConISBN& operator=(const ConISBN&);
  ~ConISBN();

  string isbn() const;
  int edicion() const;
  string editorial() const;
```

(*continúa*)

La clase ConISBN

```
  void isbn(string);
  void edicion(int);
  void editorial(string);

  void leer();
  void mostrar() const;
private:
  string _isbn;
  int _edicion;
  string _editorial;
};

#endif
```


La clase Enciclopedia

```
#ifndef enciclopedia_h
#define enciclopedia_h

#include <iostream>
#include <string>
using namespace std;
#include "conisbn.h"

class Enciclopedia : public ConISBN {
public:
  Enciclopedia(string = "", string = "", string = "",
               Fecha = Fecha(), Persona = Persona(),
               string = "", int = 1, string = "", int = 1);
  // Datos: título, ciudad, país, fecha de edición, autor,
  //        ISBN, edición, editorial, número de volúmenes.
  Enciclopedia(const Enciclopedia&);
  Enciclopedia& operator=(const Enciclopedia&);
  ~Enciclopedia();

  int volumenes() const;
```

(*continúa*)

La clase Enciclopedia

```
  void volumenes(int);

  void leer();
  void mostrar() const;
private:
  int _volumenes;
};

#endif
```

La clase **Articulo**

```
#ifndef articulo_h
#define articulo_h

#include <iostream>
#include <string>
using namespace std;
#include "volumen.h"

class Articulo : public Volumen {
public:
  Articulo(string = "", string = "", string = "",
           Fecha = Fecha(), Persona = Persona(), string = "",
           int = 1, int = 1);
  // Datos: título, ciudad, país, fecha de edición, autor,
  //        serie, pág. inicial, pág. final.
  Articulo(const Articulo&);
  Articulo& operator=(const Articulo&);
  ~Articulo();
```

(*continúa*)

La clase **Articulo**

```
  string serie() const;
  int inicial() const;
  int final() const;

  void serie(string);
  void inicial(int);
  void final(int);

  void leer();
  void mostrar() const;
private:
  string _serie;
  int _inicial, _final;
};

#endif
```


La clase **Serie**

```
// Clase Serie - Archivo de cabecera "serie.h"

#ifndef serie_h // Evitar inclusiones múltiples
#define serie_h

#include <iostream>
#include <string>
using namespace std;
#include "publicacion.h"

class Serie : public Publicacion {
public:
  Serie(string = "", string = "", string = "",
        Fecha = Fecha(), int = 1, int = 1);
  // Datos: título, ciudad, país, fecha de edición, volumen
  //        y número.
  Serie(const Serie&);
  Serie& operator=(const Serie&);
  ~Serie();
```

(*continúa*)

La clase **Serie**

```
  int volumen() const;
  int numero() const;

  void volumen(int);
  void numero(int);

  void leer();
  void mostrar() const;
private:
  int _volumen, _numero;
};

#endif
```

La clase **Actas**

```
#ifndef actas_h
#define actas_h

#include <iostream>
#include <string>
using namespace std;
#include "serie.h"

class Actas : public Serie {
public:
  Actas(string = "", string = "", string = "",
        Fecha = Fecha(), int = 1, int = 1, string = "");
  // Datos: título, ciudad, país, fecha de edición, volumen,
   número, congreso.
  Actas(const Actas&);
  Actas& operator=(const Actas&);
  ~Actas();

  string congreso() const;
```

(*continúa*)

La clase **Actas**

```
  void congreso(string);

  void leer();
  void mostrar() const;
private:
  string _congreso;
};

#endif
```


La clase **Revista**

```
#ifndef revista_h
#define revista_h

#include <iostream>
#include <string>
using namespace std;
#include "serie.h"

class Revista : public Serie {
public:
  Revista(string = "", string = "", string = "",
          Fecha = Fecha(), int = 1, int = 1, string = "");
  // Datos: título, ciudad, país, fecha de edición, volumen,
  //        número, editorial.
  Revista(const Revista&);
  Revista& operator=(const Revista&);
  ~Revista();

  string editorial() const;
```

(*continúa*)

La clase **Revista**

```
  void editorial(string);

  void leer();
  void mostrar() const;
private:
  string _editorial;
};

#endif
```