FACULTAD DE INFORMÁTICA

Más sobre herencia

SOLUCIONES DEL TALLER

Programación orientada a objetos — Unidad 7

Autor: Luis Hernández Yáñez

De vuelta a la biblioteca

Una lista de **Publicacion**es (*resolución en grupo*)

En el taller anterior construimos una jerarquía de clases de publicaciones para la biblioteca. Ahora llega el momento de crear una clase **Lista** que contenga publicaciones (**Libro**s, **Enciclopedia**s, **Acta**s, **Revista**s y **Articulo**s).

Se puede partir de la clase **Lista** que se ha visto en la Unidad.

De vuelta a la biblioteca

```
// Clase Lista (de publicaciones) - Cabecera "listabib.h"
#ifndef listabib_h
#define listabib_h
#include "publicacion.h"
class Lista {
public:
  Lista();
  Lista(const Lista&);
  Lista& operator=(const Lista&);
  ~Lista();
  bool llena() const;
  bool vacia() const;
  bool insertar(Publicacion);
  bool recuperar(int, Publicacion&) const;
  void mostrar() const;
private:
  enum { MAX = 100 };
  Publicacion _array[MAX];
  int _cont;
};
#endif
```

>>> listabib.h

De vuelta a la biblioteca

```
// Clase Lista (de publicaciones) - Implement. "listabib.cpp"
#include <iostream>
#include <string>
using namespace std;
#include "listabib.h"

Lista::Lista() : _cont(0) {}

Lista::Lista(const Lista& otra) {
  _cont = otra._cont;
  for(int i = 0; i < _cont; i++) _array[i] = otra._array[i];
}

Lista& Lista::operator=(const Lista& otra) {
  _cont = otra._cont;
  for(int i = 0; i < _cont; i++) _array[i] = otra._array[i];
  return *this;
}

Lista::~Lista() {}
```

(*continúa*)

De vuelta a la biblioteca

```
bool Lista::llena() const { return _cont == MAX; }

bool Lista::vacia() const { return _cont == 0; }

bool Lista::insertar(Publicacion p) {
  if(_cont == MAX) return false;
  _array[_cont] = p;
  _cont++;
  return true;
}

bool Lista::recuperar(int pos, Publicacion& p) const {
  if(pos < 1 || pos > _cont) return false;
  p = _array[pos-1];
  return true;
}

void Lista::mostrar() const {
  cout << "Elementos de la lista:" << endl;
  for(int i = 0; i < _cont; i++)
    _array[i].mostrar();
}
```


De vuelta a la biblioteca

```
// Prueba de la clase Lista (de publicaciones)
#include "articulo.h"
#include "libro.h"
#include "enciclopedia.h"
#include "revista.h"
#include "actas.h"
#include "listabib.h"

int main() {
  Lista lista;

  lista.insertar(Articulo("Genoma 2", "Londres", "Reino Unido",
                 Fecha(11,4,1995),
                 Persona("", 43, "John", "Maverick"),
                 "BioScience", 134, 161));

  lista.insertar(Libro("Rasputín", "Barcelona", "España",
                 Fecha(15,7,1998),
                 Persona("223344G", 31, "Juan", "Alan Gil"),
                 "76-4235-5478-9", 1, "Ed. Magnus", 231));
```

(*continúa*)

De vuelta a la biblioteca

```
  lista.insertar(Enciclopedia("Guía médica", "Madrid", "España",
                 Fecha(1,12,2001),
                 Persona("123867X", 53, "Miguel", "Vegas Sanz"),
                 "78-3245-6665-8", 3, "MisterDoc", 14));

  lista.insertar(Revista("Faralaes", "Sevilla", "España",
                 Fecha(1,8,1999), 5, 63, "Ed. Lunares"));

  lista.insertar(Actas("Proceedings of IWC 2000", "Boston",
                 "USA", Fecha(17,3,2001), 5, 63,
                 "10th Int. Workshop on Computers"));

  lista.mostrar();

  return 0;
}
```


De vuelta a la biblioteca

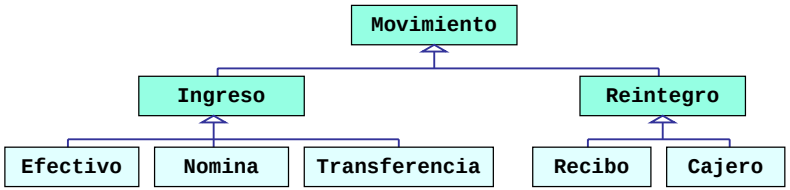
Problemas de tamaño y vinculación

```
Elementos de la lista:

Título: Genoma 2
Lugar de edición: Londres (Reino Unido)
Fecha de edición: martes 11 de abril de 1995

Título: Rasputín
Lugar de edición: Barcelona (España)
Fecha de edición: miércoles 15 de julio de 1998

Título: Guía médica
Lugar de edición: Madrid (España)
Fecha de edición: sábado 1 de diciembre de 2001

Título: Faralaes
Lugar de edición: Sevilla (España)
Fecha de edición: domingo 1 de agosto de 1999

Título: Proceedings of IWC 2000
Lugar de edición: Boston (USA)
Fecha de edición: sábado 17 de marzo de 2001
```

Movimientos de cuentas bancarias

Otra jerarquía de clases y otra lista polimórfica (*resolución en grupo*)

Hace tiempo desarrollamos una clase **Cuenta** para una aplicación de cuentas bancarias. Las cuentas bancarias se gestionan por medio de movimientos en cuenta. Los movimientos pueden ser de distintos tipos: por un lado pueden ser ingresos en cuenta (incrementos en el saldo) o reintegros (disminuciones del saldo). Contemplaremos tres tipos de ingresos: ingreso en efectivo, abono de nómina y transferencia a favor del cliente. Y dos tipos de reintegros: pago de recibo domiciliado y retirada de dinero en cajero automático.



Movimientos de cuentas bancarias

Para todos los **Movimientos** se deben mantener el concepto (**string**), la cantidad y el momento (**FechaHora**).

Y todos los movimientos se mostrarán con el siguiente formato:

DD/MM/AAAA HH:MM:SS *concepto* *cantidad* Eur

A continuación se muestra, en un ejemplo, el concepto que se debe mostrar, por defecto, para cada tipo de movimiento concreto:

```
29/11/2002 10:36:59   Ingreso en efectivo        125 Eur
30/11/2002 11:37:11   Transferencia a su favor   325.6 Eur
01/12/2002 07:07:46   Cajero automático          50 Eur
03/12/2002 18:51:20   Abono de nomina            1025.73 Eur
04/12/2002 22:38:01   Pago de recibo             120 Eur
```

El concepto se podrá cambiar a un valor distinto del por defecto.

Además, para todos los movimientos se podrá leer su cantidad y su momento (el concepto viene dado por el tipo concreto de movimiento).

Movimientos de cuentas bancarias

- Como todas las clases de movimientos van a disponer de los mismos datos, basta con que los atributos estén definidos en la superclase **Movimiento**: **_concepto**, **_cantidad** y **_momento**.
- El atributo **_concepto** se establecerá a su valor (cadena) por defecto en cada subclase durante la creación de objetos de esas clases. Pero como se debe poder cambiar, incluiremos mutador para él.
- Todos los accedentes y mutadores estarán definidos en la superclase **Movimiento**.
- Como el atributo **_concepto** se maneja de forma especial, el método **leer()** tan sólo leerá la cantidad y el momento. Estará definido en la superclase **Movimiento**.
- Y como todos los movimientos se muestran igual, el método **mostrar()** estará definido en la superclase **Movimiento**.

La clase **Movimiento**

```
// Clase Movimiento - Archivo de cabecera "movimiento.h"
#ifndef movimiento_h
#define movimiento_h

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

class Movimiento {
public:
  Movimiento(string = "", double = 0, FechaHora = FechaHora());
  // Datos: Concepto, cantidad y momento
  Movimiento(const Movimiento&);
  Movimiento& operator=(const Movimiento&);
  ~Movimiento();

  string concepto() const;
  double cantidad() const;
  FechaHora momento() const;
```

(*continúa*)

La clase **Movimiento**

```
  void concepto(string);
  void cantidad(double);
  void momento(FechaHora);

  void leer();
  void mostrar() const;
private:
  string _concepto;
  double _cantidad;
  FechaHora _momento;
};

#endif
```


La clase **Movimiento**

```
// Clase Movimiento - Implementación "movimiento.cpp"
#include "movimiento.h"

Movimiento::Movimiento(string concepto, double cantidad,
  FechaHora momento) : _concepto(concepto), _cantidad(cantidad),
   _momento(momento) { }

Movimiento::Movimiento(const Movimiento& otro) {
  _concepto = otro._concepto;
  _cantidad = otro._cantidad;
  _momento = otro._momento;
}

Movimiento& Movimiento::operator=(const Movimiento& otro) {
  _concepto = otro._concepto;
  _cantidad = otro._cantidad;
  _momento = otro._momento;
  return *this;
}

Movimiento::~Movimiento() { }
```

(*continúa*)

La clase **Movimiento**

```
string Movimiento::concepto() const { return _concepto; }

double Movimiento::cantidad() const { return _cantidad; }

FechaHora Movimiento::momento() const { return _momento; }

void Movimiento::concepto(string cad) { _concepto = cad; }

void Movimiento::cantidad(double num) { _cantidad = num; }

void Movimiento::momento(FechaHora fh) { _momento = fh; }

void Movimiento::leer() {
  cout << endl;
  cout << "Cantidad: "; cin >> _cantidad;
  cout << "Fecha y hora:" << endl;
  _momento.leer();
}

void Movimiento::mostrar() const {
  cout << _momento.reducida() << "   " << _concepto
       << "   " << _cantidad << " Eur" << endl;
}
```


Movimientos de cuentas bancarias

De todos los **Movimientos** se deberá poder obtener el *apunte* que se debe realizar en la cuenta bancaria: el apunte será positivo para los ingresos (la cantidad) y negativo para los reintegros (la cantidad cambiada de signo).

- En la clase **Ingreso** el método **apunte()** devolverá la cantidad.
- En la clase **Reintegro** el método **apunte()** devolverá la cantidad con signo menos.
- Las demás clases simplemente establecerán el **_concepto** en el constructor.

Después de desarrollar las clases de la jerarquía, crea otra clase, **Lista**, para mantener listas de **Movimientos**.

La clase **Ingreso**

```
// Clase Ingreso - Archivo de cabecera "ingreso.h"
#ifndef ingreso_h
#define ingreso_h

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

class Ingreso : public Movimiento {
public:
  Ingreso(string = "", double = 0, FechaHora = FechaHora());
  // Datos: Concepto, cantidad y momento
  Ingreso(const Ingreso&);
  Ingreso& operator=(const Ingreso&);
  ~Ingreso();
  double apunte() const;
};

#endif
```


La clase **Ingreso**

```
// Clase Ingreso - Implementación "ingreso.cpp"

#include "ingreso.h"

Ingreso::Ingreso(string concepto, double cantidad,
  FechaHora momento)
  : Movimiento(concepto, cantidad, momento) { }

Ingreso::Ingreso(const Ingreso& otro) : Movimiento(otro) { }

Ingreso& Ingreso::operator=(const Ingreso& otro) {
  Movimiento::operator=(otro);
  return *this;
}

Ingreso::~Ingreso() { }

double Ingreso::apunte() const { return cantidad(); }
```

Accedente (los atributos son privados en la superclase)

La clase **Reintegro**

```
// Clase Reintegro - Archivo de cabecera "reintegro.h"
#ifndef reintegro_h
#define reintegro_h

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

class Reintegro : public Movimiento {
public:
  Reintegro(string = "", double = 0, FechaHora = FechaHora());
  // Datos: Concepto, cantidad y momento
  Reintegro(const Reintegro&);
  Reintegro& operator=(const Reintegro&);
  ~Reintegro();
  double apunte() const;
};

#endif
```


La clase **Reintegro**

```
// Clase Reintegro - Implementación "reintegro.cpp"

#include "reintegro.h"

Reintegro::Reintegro(string concepto, double cantidad,
  FechaHora momento)
  : Movimiento(concepto, cantidad, momento) { }

Reintegro::Reintegro(const Reintegro& otro) : Movimiento(otro)
  { }

Reintegro& Reintegro::operator=(const Reintegro& otro) {
  Movimiento::operator=(otro);
  return *this;
}

Reintegro::~Reintegro() { }

double Reintegro::apunte() const { return - cantidad(); }
```

La clase **Efectivo**

```
// Clase Efectivo - Archivo de cabecera "efectivo.h"

#ifndef efectivo_h
#define efectivo_h

#include "ingreso.h"

class Efectivo : public Ingreso {
public:
  Efectivo(double = 0, FechaHora = FechaHora());
  // Datos: Cantidad y momento
  Efectivo(const Efectivo&);
  Efectivo& operator=(const Efectivo&);
  ~Efectivo();
};

#endif
```


La clase **Efectivo**

```
// Clase Efectivo - Implementación "efectivo.cpp"

#include "efectivo.h"

Efectivo::Efectivo(double cantidad, FechaHora momento)
  : Ingreso("Ingreso en efectivo     ", cantidad, momento) { }

Efectivo::Efectivo(const Efectivo& otro) : Ingreso(otro) { }

Efectivo& Efectivo::operator=(const Efectivo& otro) {
  Ingreso::operator=(otro);
  return *this;
}

Efectivo::~Efectivo() { }
```


La clase **Nomina**

```
// Clase Nomina - Archivo de cabecera "nomina.h"

#ifndef nomina_h
#define nomina_h

#include "ingreso.h"

class Nomina : public Ingreso {
public:
  Nomina(double = 0, FechaHora = FechaHora());
  // Datos: Cantidad y momento
  Nomina(const Nomina&);
  Nomina& operator=(const Nomina&);
  ~Nomina();
};

#endif
```


La clase **Nomina**

```
// Clase Nomina - Implementación "nomina.cpp"

#include "nomina.h"

Nomina::Nomina(double cantidad, FechaHora momento)
  : Ingreso("Abono de nómina        ", cantidad, momento) { }

Nomina::Nomina(const Nomina& otro) : Ingreso(otro) { }

Nomina& Nomina::operator=(const Nomina& otro) {
  Ingreso::operator=(otro);
  return *this;
}

Nomina::~Nomina() { }
```

La clase **Transferencia**

```
// Clase Transferencia - Archivo de cabecera "transferencia.h"

#ifndef transferencia_h
#define transferencia_h

#include "ingreso.h"

class Transferencia : public Ingreso {
public:
  Transferencia(double = 0, FechaHora = FechaHora());
  // Datos: Cantidad y momento
  Transferencia(const Transferencia&);
  Transferencia& operator=(const Transferencia&);
  ~Transferencia();
};

#endif
```


La clase **Transferencia**

```
// Clase Transferencia - Implementación "transferencia.cpp"

#include "transferencia.h"

Transferencia::Transferencia(double cantidad, FechaHora momento)
  : Ingreso("Transferencia a su favor", cantidad, momento) { }

Transferencia::Transferencia(const Transferencia& otro) :
   Ingreso(otro) { }

Transferencia& Transferencia::operator=
  (const Transferencia& otro) {
  Ingreso::operator=(otro);
  return *this;
}

Transferencia::~Transferencia() { }
```


La clase **Recibo**

```
// Clase Recibo - Archivo de cabecera "recibo.h"

#ifndef recibo_h
#define recibo_h

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

class Recibo : public Reintegro {
public:
  Recibo(double = 0, FechaHora = FechaHora());
  // Datos: Cantidad y momento
  Recibo(const Recibo&);
  Recibo& operator=(const Recibo&);
  ~Recibo();
};

#endif
```


La clase **Recibo**

```
// Clase Recibo - Implementación "recibo.cpp"

#include "recibo.h"

Recibo::Recibo(double cantidad, FechaHora momento)
  : Reintegro("Pago de recibo          ", cantidad, momento) { }

Recibo::Recibo(const Recibo& otro) : Reintegro(otro) { }

Recibo& Recibo::operator=(const Recibo& otro) {
  Reintegro::operator=(otro);
  return *this;
}

Recibo::~Recibo() { }
```

La clase **Cajero**

```
// Clase Cajero - Archivo de cabecera "cajero.h"

#ifndef cajero_h
#define cajero_h

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

class Cajero : public Reintegro {
public:
  Cajero(double = 0, FechaHora = FechaHora());
  // Datos: Cantidad y momento
  Cajero(const Cajero&);
  Cajero& operator=(const Cajero&);
  ~Cajero();
};

#endif
```


La clase **Cajero**

```
// Clase Cajero - Implementación "cajero.cpp"

#include "cajero.h"

Cajero::Cajero(double cantidad, FechaHora momento)
  : Reintegro("Cajero automático       ", cantidad, momento) { }

Cajero::Cajero(const Cajero& otro) : Reintegro(otro) { }

Cajero& Cajero::operator=(const Cajero& otro) {
  Reintegro::operator=(otro);
  return *this;
}

Cajero::~Cajero() { }
```

La clase **Lista** de movimientos (**listamov.h** y **listamov.cpp**) es igual que la clase **Lista** de publicaciones (**listabib.h** y **listabib.cpp**) pero cambiando **Publicacion** por **Movimiento** en todas partes.

Prueba de la **Lista** de **Movimientos**

```
#include <iostream>
#include <string>
using namespace std;

#include "fechahora.h"
#include "efectivo.h"
#include "transferencia.h"
#include "nomina.h"
#include "recibo.h"
#include "cajero.h"
#include "listamov.h"

int main() {
  FechaHora fh(Fecha(29,11,2002),Hora(10,36,59));
  Lista lista;

  lista.insertar(Efectivo(125, fh));

  fh = fh + 90012; // Un tiempo después
  lista.insertar(Transferencia(325.60, fh));
```

(*continúa*)

Prueba de la **Lista** de **Movimientos**

```
  fh = fh + 70235; // Un tiempo después
  lista.insertar(Cajero(50, fh));

  fh = fh + 215014; // Un tiempo después
  lista.insertar(Nomina(1025.73, fh));

  fh = fh + 100001; // Un tiempo después
  lista.insertar(Recibo(120, fh));

  lista.mostrar();

  return 0;
}
```

Una lista polimórfica sin problemas de tamaño y (aparentemente) sin problemas de vinculación

```
29/11/2002 10:36:59   Ingreso en efectivo        125 Eur
30/11/2002 11:37:11   Transferencia a su favor  325.6 Eur
01/12/2002 07:07:46   Cajero automático         50 Eur
03/12/2002 18:51:20   Abono de nómina           1025.73 Eur
04/12/2002 22:38:01   Pago de recibo            120 Eur
```