FACULTAD DE INFORMÁTICA

Ejercicios de refuerzo

SOLUCIONES

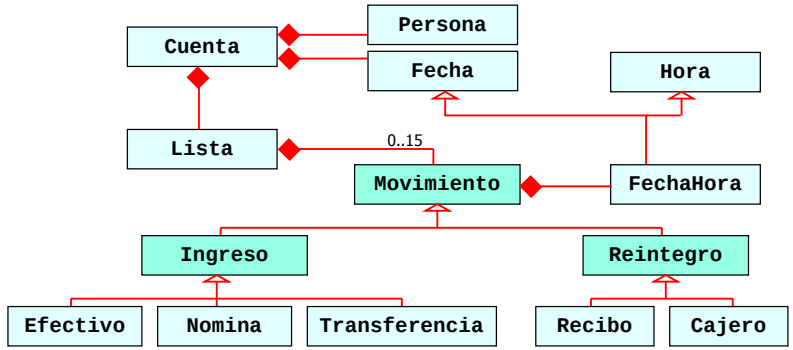
Programación orientada a objetos

Autor: Luis Hernández Yáñez

FdI UCM

Las clases de partida para los ejercicios

Las clases de partida para los ejercicios son las de la aplicación de cuentas bancarias del último taller:



Excepcionalmente hemos añadido en la clase **Cuenta** un método `bool recuperar(int, Movimiento*&)` y otro `void leer()`.

FdI UCM

Ejercicio nº 1

Desarrolla un programa completo que haga lo siguiente (en el orden que se indica): crear un objeto dinámico **ahorros** de clase **Cuenta**, leer los datos de ese objeto, pedir al usuario los datos de los cuatro primeros movimientos de la cuenta y añadirlos a su lista de movimientos. Finalmente, mostrar en la pantalla toda la información de la cuenta con un formato como el siguiente:

```
Cliente: Alba Santos Torres
NIF: 33445577G
Cuenta abierta el viernes 1 de febrero de 2002
Ultimos movimientos:
  El sabado 2 de febrero de 2002:
    Abono de nomina          32672.15 Eur (11:07:36)
    Cajero automatico          250.00 Eur (13:27:00)
  El martes 5 de febrero de 2002:
    Cajero automatico          300.00 Eur (18:12:16)
  El miercoles 6 de febrero de 2002:
    Transferencia a su favor 1200.25 Eur (09:27:12)
Saldo actual: 241764.46 Eur
```

Se valorará el menor uso de variables puntero (el mínimo es dos).

FdI UCM

Ejercicio nº 1

banco.cpp

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

#include <iomanip.h> // Manipuladores
#include "fechahora.h"
#include "transferencia.h"
#include "nomina.h"
#include "cajero.h"
#include "cuenta.h"

int main() {
  cout << fixed << setprecision(2);
  // Manipuladores que establecen la visualización de números
  // en punto fijo (no notación científica) y con 2 decimales

  Cuenta* ahorros = new Cuenta();

  ahorros->leer();

  Movimiento* mov;
```

(*continúa*)

Ejercicio nº 1

```
for(int i = 0; i < 4; i++) {
  int opcion;
  cout <<
  "Tipo de movimiento (1=Nómina, 2=Transferencia, 3=Cajero)? ";
  cin >> opcion;
  switch(opcion) {
    case(1) : mov = new Nomina(); break;
    case(2) : mov = new Transferencia(); break;
    case(3) : mov = new Cajero(); break;
  }
  mov->leer();
  ahorros->nuevo(mov);
}
// Comienza la visualización de información:
cout << endl;
cout << "Cliente: " << ahorros->cliente().nombreCompleto()
     << endl;
cout << "NIF: " << ahorros->cliente().nif() << endl;
cout << "Cuenta abierta el " << ahorros->fecha().completa()
     << endl;
```

(*continúa*)

Ejercicio nº 1

```
  cout << "Últimos movimientos:" << endl;
  FechaHora actual;
  int pos = 1; // Empezamos por el primer movimiento
  while(ahorros->recuperar(pos, &mov)) { // Si existe
    if(mov->momento().fecha() > actual.fecha()) {
      actual = mov->momento();
      cout << "  El " << actual.fecha().completa() << ":"
           << endl;
    }
    cout << "    " << mov->concepto() << " " << mov->cantidad()
         << " Eur (" << mov->momento().hora().hora() << ")"
         << endl;
    pos++;
  }
  cout << "Saldo actual: " << ahorros->saldo() << " Eur" << endl;
  delete ahorros;

  return 0;
}
```


Ejercicio nº 2

En la aplicación de cuentas bancarias se hace necesario distinguir entre dos tipos particulares de recibos: de la luz y de una suscripción. Esto no quita que se vayan a seguir usando recibos genéricos.

Para acomodar los tipos concretos de recibos, debes incluir dos clases:

Luz, para los recibos de consumo eléctrico. Además del concepto, la cantidad y el momento necesita dos datos más: el nombre de la compañía eléctrica y los kilowatios-hora consumidos.

Suscripcion, para las suscripciones. Además del concepto, la cantidad y el momento necesita dos datos más: el nombre de la compañía que factura y el número de recibo.

Para estos dos tipos concretos de recibos se mostrará también la información adicional en una segunda línea. En la siguiente página se muestra un ejemplo de salida.

Ejercicio nº 2

```
Cliente: Alba Santos Torres (33445577G), 25 años
Cuenta abierta el viernes 1 de febrero de 2002
Saldo actual: 207971.42 Eur

02/02/2002 13:27:00   Cajero automatico          250.00 Eur
05/02/2002 18:12:16   Pago de recibo             112.45 Eur
06/02/2002 09:27:12   Pago de recibo de la luz   73.24 Eur
  [Hiberdrola, 7.26 Kwh]
12/02/2002 10:08:23   Pago de suscripcion        34.95 Eur
  [Revista PC-World, numero 14]
```

Ejercicio nº 2 - La clase **Recibo** ampliada

```
// 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(), string = "");
  // Datos: Cantidad, momento y empresa
  Recibo(const Recibo&);
  Recibo& operator=(const Recibo&);
  virtual ~Recibo();
  string empresa() const;
  void empresa(string);
  virtual Movimiento* clon() const;
private:
  string _empresa;
};
#endif
```


Ejercicio nº 2 - La clase **Recibo** ampliada

```
// Clase Recibo - Implementación "recibo.cpp"
#include "recibo.h"

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

Recibo::Recibo(const Recibo& otro) : Reintegro(otro)
{ _empresa = otro._empresa; }

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

Recibo::~Recibo() { }

string Recibo::empresa() const { return _empresa; }

void Recibo::empresa(string emp) { _empresa = emp; }

Movimiento* Recibo::clon() const {
  Recibo* m = new Recibo(*this);
  return m;
}
```


Ejercicio nº 2 - La clase **Luz**

```
// Clase Luz - Archivo de cabecera "luz.h"
#ifndef luz_h
#define luz_h

#include "recibo.h"

class Luz : public Recibo {
public:
  Luz(double=0, FechaHora = FechaHora(), string = "", double=0);
  // Datos: Cantidad, momento, empresa y kilowatios-hora
  Luz(const Luz&);
  Luz& operator=(const Luz&);
  ~Luz();
  double kwh() const;
  void kwh(double);
  Movimiento* clon() const;
  void mostrar() const;
private:
  double _kwh;
};
#endif
```


Ejercicio nº 2 - La clase **Luz**

```
// Clase Luz - Implementación "luz.cpp"
#include "luz.h"

Luz::Luz(double cantidad, FechaHora momento, string emp,
         double kwh) : Recibo(cantidad, momento, emp), _kwh(kwh)
{ concepto("Pago de recibo de la luz"); }

Luz::Luz(const Luz& otro) : Recibo(otro) { _kwh = otro._kwh; }

Luz& Luz::operator=(const Luz& otro) {
  Recibo::operator=(otro);
  _kwh = otro._kwh;
  return *this;
}

Luz::~Luz() { }

double Luz::kwh() const { return _kwh; }

void Luz::kwh(double kwhora) { _kwh = kwhora; }
```

(*continúa*)

Ejercicio nº 2 - La clase **Luz**

```
Movimiento* Luz::clon() const {
  Luz* m = new Luz(*this);
  return m;
}

void Luz::mostrar() const {
  Recibo::mostrar();
  cout << "  [" << empresa() << ", " << _kwh << " Kwh]" << endl;
}
```


Ejercicio nº 2 - La clase **Suscripcion**

```
// Clase Suscripcion - Archivo de cabecera "suscripcion.h"
#ifndef suscripcion_h
#define suscripcion_h

#include "recibo.h"

class Suscripcion : public Recibo {
public:
  Suscripcion(double = 0, FechaHora = FechaHora(), string = "",
              int = 1);
  // Datos: Cantidad, momento, empresa y número de recibo
  Suscripcion(const Suscripcion&);
  Suscripcion& operator=(const Suscripcion&);
  ~Suscripcion();
  int recibo() const;
  void recibo(int);
  Movimiento* clon() const;
  void mostrar() const;
private:
  int _recibo;
};
#endif
```


Ejercicio nº 2 - La clase **Suscripcion**

```
// Clase Suscripcion - Implementación "suscripcion.cpp"
#include "suscripcion.h"

Suscripcion::Suscripcion(double cantidad, FechaHora momento,
                         string emp, int num)
  : Recibo(cantidad, momento, emp), _recibo(num)
{ concepto("Pago de suscripcion     "); }

Suscripcion::Suscripcion(const Suscripcion& otro) : Recibo(otro)
{ _recibo = otro._recibo; }

Suscripcion& Suscripcion::operator=(const Suscripcion& otro) {
  Recibo::operator=(otro);
  _recibo = otro._recibo;
  return *this;
}

Suscripcion::~Suscripcion() { }

int Suscripcion::recibo() const { return _recibo; }
```

(*continúa*)

Ejercicio nº 2 - La clase **Suscripcion**

```
void Suscripcion::recibo(int num) { _recibo = num; }

Movimiento* Suscripcion::clon() const {
  Suscripcion* m = new Suscripcion(*this);
  return m;
}

void Suscripcion::mostrar() const {
  Recibo::mostrar();
  cout << "  [" << empresa() << ", numero " << _recibo << "]"
       << endl;
}
```

Ejercicio nº 2 - Prueba de las clases

banco2.cpp

```
#include <iostream>
#include <string>
using namespace std;
#include <iomanip.h>
#include "fechahora.h"
#include "recibo.h"
#include "cajero.h"
#include "luz.h"
#include "suscripcion.h"
#include "cuenta.h"

int main() {
  cout << fixed << setprecision(2);

  Cuenta* ahorros =
    new Cuenta(Persona("33445577G", 25, "Alba", "Santos Torres"),
               Fecha(1, 2, 2002), 208442.06);

  FechaHora fh = FechaHora(Fecha(2,2,2002), Hora(13,27,00));
  ahorros->nuevo(new Cajero(250, fh));
```

(*continúa*)

Ejercicio nº 2 - Prueba de las clases

```
  fh = FechaHora(Fecha(5,2,2002), Hora(18,12,16));
  ahorros->nuevo(new Recibo(112.45, fh));

  fh = FechaHora(Fecha(6,2,2002), Hora(9,27,12));
  ahorros->nuevo(new Luz(73.24, fh, "Hiberdrola", 7.264));

  fh = FechaHora(Fecha(12,2,2002), Hora(10,8,23));
  ahorros->nuevo(new Suscripcion(34.95, fh, "Rev. PC-World",
                                 14));

  ahorros->extracto();

  delete ahorros;

  return 0;
}
```


Ejercicio nº 3

Para desarrollar un árbol genealógico, se necesita disponer de una clase **`MiembroFamilia`** con las siguientes características:

- ✓ Datos sobre cada ejemplar: nombre, apellidos, edad, NIF, fecha de nacimiento, hora de nacimiento, sexo, padre y madre.
- ✓ Los atributos de fecha de nacimiento y hora de nacimiento serán atributos dinámicos.
- ✓ Los atributos de padre y madre serán relaciones con otros objetos de la clase (puede no tenerse información sobre quién es el padre o la madre del miembro de la familia).
- ✓ La clase debe ajustarse a la Forma canónica ortodoxa y disponer de accedentes y mutadores para todos los atributos.
- ✓ La clase debe disponer de un método **`leer()`** que pida los datos y de un método **`mostrar()`** que muestre la información de la forma que indica el ejemplo del principio de la siguiente página.

Ejercicio nº 3

```
Dª Rosa Gómez Pérez (3456789K)
30 años
Nacida el 12 de abril de 1973 a las 17:07:47
Padre: D. Javier Gómez Ruiz
Madre: Dª Rosa Pérez Galván
```

- ✓ La clase debe disponer de un método **`antepasados()`** que muestre todos los antepasados del miembro de la familia. Para cada antepasado se indicará, después del nombre (con el tratamiento oportuno — ver Nota), su fecha y hora de nacimiento:

```
D. Javier Gómez Ruiz, 25-6-1946 09:23:32
Dª Rosa Pérez Galván, 4-4-1947 22:08:12
...
```

Se pide desarrollar por completo la clase **`MiembroFamilia`**.

Nota: siempre que se muestre el nombre de un miembro de la familia se colocará delante el tratamiento que corresponda (**`D.`** para los varones y **`Dª`** para las mujeres).

Ejercicio nº 3 - La clase **Miembro**

```
// Clase Miembro - Archivo de cabecera "miembro.h"
#ifndef miembro_h
#define miembro_h

#include "persona.h"
#include "fecha.h"
#include "hora.h"

class Miembro : public Persona {
public:
  Miembro(string = "", int = 0, string = "", string = "",
          Fecha = Fecha(), Hora = Hora(), bool = true,
          Miembro* = NULL, Miembro* = NULL);
  // Datos: NIF, edad, nombre, apellidos, fecha nac., hora nac.,
  //        sexo (true = varón), vínculo con el padre y
  //        vínculo con la madre.
  Miembro(const Miembro&);
  Miembro& operator=(const Miembro&);
  ~Miembro();
  Persona* clon() const;
```

(*continúa*)

Ejercicio nº 3 - La clase **Miembro**

```
  void fecha(Fecha);
  void hora(Hora);
  void sexo(bool);
  void padre(Miembro*);
  void madre(Miembro*);
  Fecha fecha() const;
  Hora hora() const;
  bool sexo() const;
  Miembro* padre() const;
  Miembro* madre() const;
  void leer();
  void mostrar() const;
  void mostrarbreve() const; // D./Dª nombrecompleto, fecha hora
  void antepasados() const;
private:
  Fecha* _fecha; // atributo dinámico
  Hora* _hora; // atributo dinámico
  bool _sexo; // true = varón | false = mujer
  Miembro* _padre, * _madre; // relaciones
  string nombretrat() const; // D./Dª nombrecompleto
};
#endif
```


Ejercicio nº 3 - La clase **Miembro**

```
// Clase Miembro - Implementación "miembro.cpp"
#include "miembro.h"

Miembro::Miembro(string nif, int edad, string nombre,
                 string apellidos, Fecha fnac, Hora hnac,
                 bool sexo, Miembro* padre, Miembro* madre)
  : Persona(nif, edad, nombre, apellidos),
    _fecha(new Fecha(fnac)), _hora(new Hora(hnac)), _sexo(sexo),
    _padre(padre), _madre(madre) { }

Miembro::Miembro(const Miembro& otro) : Persona(otro) {
  _fecha = new Fecha(*otro._fecha);
  _hora = new Hora(*otro._hora);
  _sexo = otro._sexo;
  _padre = otro._padre;
  _madre = otro._madre;
}

Miembro& Miembro::operator=(const Miembro& otro) {
  Persona::operator=(otro);
  *_fecha = *otro._fecha;
  *_hora = *otro._hora;
```

(*continúa*)

Ejercicio nº 3 - La clase **Miembro**

```
  _sexo = otro._sexo;
  _padre = otro._padre;
  _madre = otro._madre;
  return *this;
}

Miembro::~Miembro() {
  delete _fecha;
  delete _hora;
}

Persona* Miembro::clon() const {
  Miembro* p = new Miembro(*this);
  return p;
}

void Miembro::fecha(Fecha f) { *_fecha = f; }
void Miembro::hora(Hora h) { *_hora = h; }
void Miembro::sexo(bool s) { _sexo = s; }
void Miembro::padre(Miembro* p) { _padre = p; }
void Miembro::madre(Miembro* m) { _madre = m; }
Fecha Miembro::fecha() const { return *_fecha; }
```

(*continúa*)

Ejercicio nº 3 - La clase **Miembro**

```
Hora Miembro::hora() const { return *_hora; }
bool Miembro::sexo() const { return _sexo; }
Miembro* Miembro::padre() const { return _padre; }
Miembro* Miembro::madre() const { return _madre; }

void Miembro::leer() {
  Persona::leer();
  cout << "Fecha de nacimiento:" << endl; _fecha->leer();
  cout << "Hora de nacimiento:" << endl; _hora->leer();
  cout << "Sexo (1 = Varón, 2 = Mujer): ";
  int op; cin >> op;
  _sexo = (op == 1);
}

void Miembro::mostrar() const {
  cout << nombretrat() << " (" << nif() << ")" << endl;
  cout << edad() << " años" << endl;
  cout << "Nacid" << ( _sexo ? "o" : "a") << " el "
    << _fecha->completa() << " a las " << _hora->hora() << endl;
  cout << "Padre: ";
  if(_padre == NULL) cout << "desconocido" << endl;
  else cout << _padre->nombretrat() << endl;
```

(*continúa*)

Ejercicio nº 3 - La clase **Miembro**

```
  cout << "Madre: ";
  if(_madre == NULL) cout << "desconocida" << endl;
  else cout << _madre->nombretrat() << endl;
}

void Miembro::mostrarbreve() const {
  cout << nombretrat() << ", " << _fecha->reducida()
       << " " << _hora->hora() << endl;
}

void Miembro::antepasados() const {
  if(_padre != NULL)
  { _padre->mostrarbreve();
    _padre->antepasados(); }
  if(_madre != NULL)
  { _madre->mostrarbreve();
    _madre->antepasados(); }
}

string Miembro::nombretrat() const {
  return ( _sexo ? "D. " : "Dª ") + nombreCompleto();
}
```


Ejercicio nº 3 - Una lista de **Personas** (y **Miembros**)

```
// Clase Lista (de personas) - Archivo de cabecera "lista.h"
#ifndef lista_h
#define lista_h
#include "persona.h"
class Lista {
public:
  Lista();
  Lista(const Lista&);
  Lista& operator=(const Lista&);
  ~Lista();
  bool llena() const;
  bool vacia() const;
  int cont() const;
  void insertar(Persona*);
  bool recuperar(int, Persona*&) const;
  void mostrar() const;
private:
  enum { MAX = 100 };
  Persona* _array[MAX];
  int _cont;
};
#endif
```


Ejercicio nº 3 - Una lista de **Personas** (y **Miembros**)

```
// Clase Lista (de personas) - Implementación "lista.cpp"
#include <iostream>
using namespace std;
#include "lista.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]->clon();
}

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

(*continúa*)

Ejercicio nº 3 - Una lista de **Personas** (y **Miembros**)

```
Lista::~Lista()
{ for(int i = 0; i < _cont; i++) delete _array[i]; }

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

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

int Lista::cont() const { return _cont; }

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

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

(*continúa*)

Ejercicio nº 3 - Una lista de **Personas** (y **Miembros**)

```
void Lista::mostrar() const {
  cout << endl;
  for(int i = 0; i < _cont; i++)
    _array[i]->mostrar();
  cout << endl;
}
```


Ejercicio nº 3 - Una prueba

familia.cpp

```
#include "fecha.h"
#include "hora.h"
#include "miembro.h"
#include "lista.h"

int main() {
  Lista miembros;
  Miembro* padre =
    new Miembro("111222F", 98, "Juan", "Soto Verde",
                Fecha(1, 6, 1904), Hora(13, 43, 24), true);
  miembros.insertar(padre);
  Miembro* madre =
    new Miembro("231563Y", 95, "Rosa", "Izas Vegas",
                Fecha(12, 8, 1907), Hora(3, 5, 41), false);
  miembros.insertar(madre);
  Miembro* m = new Miembro("7638426S", 73, "Juan", "Soto Izas",
    Fecha(3, 1, 1930), Hora(16, 35, 7), true, padre, madre);
  miembros.insertar(m);
  padre = m;
  madre = new Miembro("987654E", 89, "Ana", "Aro Rojas",
                      Fecha(27, 4, 1913), Hora(6, 23, 5), false);
  miembros.insertar(madre);
```

(*continúa*)

Ejercicio nº 3 - Una prueba

```
  m = new Miembro("5263547V", 70, "Ana", "Gil Aro",
      Fecha(4, 12, 1932), Hora(0, 15, 30), false, NULL, madre);
  miembros.insertar(m);
  madre = m;
  m = new Miembro("52000435H", 42, "Ana", "Soto Gil",
      Fecha(19, 3, 1960), Hora(20, 15, 5), false, padre, madre);
  miembros.insertar(m);
  madre = m;
  m = new Miembro("52000435H", 45, "Javier", "Sanz Reverte",
                  Fecha(1, 11, 1957), Hora(8, 9, 23), true);
  miembros.insertar(m);
  padre = m;
  m = new Miembro("4443322P", 18, "Javier", "Sanz Soto",
      Fecha(7, 5, 1984), Hora(5, 7, 53), true, padre, madre);
  miembros.insertar(m);
  miembros.mostrar();
  cout << "Antepasados de " << m->nombreCompleto() << ":"
       << endl;
  m->antepasados();

  return 0;
}
```