Fundamentos de Programación

Grado en Ing. Informática

Guión práctico nº 4

Tema 5.- Diseño Descendente

Ejercicio 1

Se va a realizar un programa en c++ para controlar los productos guardados en un almacén. De cada producto tendremos su nombre, su precio y la cantidad que hay en el almacén. Para ello en principio diseñaremos una clase tprod que servirá para tratar un producto, y a continuación pasaremos a realizar otra clase en el mismo programa que maneje una tabla de productos que será el almacén.

Comencemos por la clase tprod para un producto.

```
typedef char cad[20];

class tprod {
        cad nombre;
        float precio;
        int stock;
public: …
};
```

Los atributos nombre, precio y stock almacenarán el nombre, el precio y la cantidad en stock del producto.

Diseñe los siguientes métodos para esta clase:

1. Un **constructor** para esta clase que ponga en el nombre “NO HAY PRODUCTO”, en el precio un 0, y en el stock un 0.
2. **void cambiarnombre(cad nom);** Este método recibe nom como parámetro y lo copia en el atributo nombre.
3. **void cambiarprecio(float prec);** Recibe prec como parámetro y lo copia en el atributo precio.
4. **void cambiarstock(int stoc);** Recibe stoc como parámetro y lo copia en el atributo stock.
5. **void leenombre (cad nom);** Devuelve en el parámetro nom el contenido del atributo nombre.
6. **float leeprecio ();** Devuelve el contenido del atributo precio.
7. **void leestock(int &st);** Devuelve en el parámetro st el contenido del atributo stock.
8. **void vender(int cantidad);** Simula la venta del producto, quitando del stock la cantidad pasada como parámetro, además mostrará por pantalla el precio a cobrar que será el producto del precio unitario por la cantidad vendida. Si no hubiera en el stock suficiente cantidad mostrará un mensaje por pantalla expresando dicha circunstancia no realizándose la venta.

9. **Diseñe un main** que compruebe el funcionamiento correcto de todos estos métodos. Cree un objeto de tipo tprod, visualice su contenido, cambie el nombre, precio y stock con valores leídos desde teclado, muestre por pantalla el nuevo contenido, intente realizar una venta con más cantidad que hay en el stock, y otra con menos, visualice el producto.

Pasemos a tratar la clase almacen:

```
#define  MAX  5
class almacen {
   tprod productos[MAX];
   int nprod;
public:…
};
```

El atributo productos es una tabla que contendrá tantos objetos de tipo tprod como indique el atributo nprod. Si nprod vale 5 quiere decir que habrá productos en las posiciones 0, 1, 2, 3 y 4 de la tabla.

Diseñe los siguientes métodos para esta clase:

10. **almacen();** Constructor que pondrá el almacén vacío.
11. **void vaciar();** Método que pondrá el almacén vacío.
12. **int existe(cad nom);** Método que recibiendo el nombre de un producto como parámetro devolverá en qué posición de la tabla se encuentra almacenado un producto con ese nombre o bien -1 si no está.
13. **void verprod (int pos, tprod &prod);** Método que pondrá en prod el contenido del producto que se encuentra en la posición pos de la tabla de productos.
14. **int insertar(tprod P);** Método que intentará insertar un nuevo producto P pasado como parámetro en la tabla de productos. Si la tabla está llena devolverá un 2 y si ya hay un producto con el mismo nombre devolverá un 1 no insertando en ninguno de estos casos el producto. Si lo ha podido insertar devolverá un 0.
15. **void vertabla ();** Método que visualizará por pantalla el contenido del almacén. Cada producto deberá mostrarse en una línea diferente con su nombre, precio y stock. Si el almacén está vacío expresará esta situación por pantalla.
16. **void vender (int pos, int cant);** Método que recibiendo como parámetros pos y cantidad intentará realizar la venta del producto que está en la posición pos de la tabla y una cantidad cant de ese producto.

Diseñe una función genérica:

17. char menu (); Método que mostrará el siguiente menú:

Devolviendo el código ASCII de la tecla pulsada.

Diseñe un **main** que compruebe el funcionamiento correcto de los métodos de esta clase.

18. Borre el main que realizó en el problema 10.

    Dicho main deberá de crear al menos un objeto de tipo almacen.

    Mostrará el menú anterior, la única manera de que acabe el programa es pulsando la opción de salir **(E o bien e).**

    Si se pulsa A o bien a, visualizará el contenido completo del almacén por pantalla.

    Si se pulsa B o bien b, querrá el usuario insertar un nuevo producto, solicitará por teclado que ponga el nombre del producto nuevo, el precio y el stock. Intentará insertarlo sabiendo que si el almacén está lleno expresará dicha situación por pantalla y si el producto ya existía también indicará esta otra situación.

    Si se pulsa C o bien c, se intentará realizar una venta, pedirá por teclado que el usuario ponga el nombre del producto, si ese producto no existe lo indicará por pantalla, en caso de que exista le pedirá la cantidad a vender, e intentará realizar la venta, descontando la cantidad vendida del stock e indicando el precio de la venta total por pantalla o bien que no había suficiente cantidad en stock.

    Si se pulsa D o bien d, se pondrá el almacén vacío de productos.

Ejercicio 2

Se quiere realizar el juego del ***Tres en Raya*** (tic tac toe) para lo cual se le solicita al alumno las implementaciones que se detallan a continuación:

a. La clase **`TicTacToe`** con los siguientes atributos y métodos:

- **Atributos privados:**
  - **`char Tablero[3][3]`**

    Contendrá la partida realizada. Los valores posibles para sus casillas serán: **`espacio en blanco`**, **`X`** y **`O`**.

- **Métodos públicos:**
  - **`Tictactoe()`**

    Hará una llamada a **`LimpiarTablero()`** de modo que cuando se cree el tablero quede preparado para jugar.

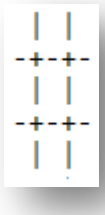
  - **`void LimpiarTablero()`**

    Asignará a todas las casillas del tablero el valor **`espacio en blanco`**.

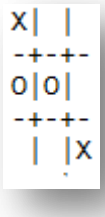
  - **`void Pintar()`**

    Mostrará en pantalla el contenido del tablero con el siguiente formato:

    Tablero vacío:

    Tablero con algunas tiradas:

  - **`bool PonerFicha(char ficha, int fila, int columna)`**

    Si en la posición del tablero indicada por los parámetros **`fila`** y **`columna`** hay un **`espacio en blanco`**, se le asignará a dicha posición el valor contenido en el parámetro **`ficha`** (**`X`** u **`O`**), en caso contrario no se podrá anotar la jugada. Además si la jugada se ha podido realizar devolverá **`true`** y si no devolvería **`false`**.

- **`bool ComprobarFila(char ficha, int fila)`**

  Devuelve **`true`** si se ha conseguido realizar 3 en raya con la **`ficha`** y en la **`fila`** indicadas en los parámetros correspondientes. En caso contrario devolverá **`false`**.

- **`bool ComprobarColumna(char ficha, int columna)`**

  Devuelve **`true`** si se ha conseguido realizar 3 en raya con la **`ficha`** y en la **`columna`** indicadas en los parámetros correspondientes. En caso contrario devolverá **`false`**.

- **`bool ComprobarDiagonal(char ficha, int fila, int columna)`**

  Devuelve **`true`** si en la diagonal o diagonales (en el caso de tratarse de la casilla central del tablero) en la que se encuentra la casilla marcada por los parámetros **`fila`** y **`columna`** se ha conseguido 3 en raya con la **`ficha`** indicada, y **`false`** en caso contrario.

- **`bool TableroCompleto()`**

  Devuelve **`true`** si no se puede seguir jugando por no quedar espacios en blanco, y **`false`** en caso contrario.

b. La función **`void PedirPosicion(char ficha, int &fila, int &columna)`**

   Muestra un mensaje al jugador indicado por el parámetro **`ficha`**, solicitándole que introduzca una coordenada del tablero. Dicha coordenada será devuelta a través de los parámetros **`fila`** y **`columna`**.

   Esta función **no podrá devolver una coordenada incorrecta**, por lo que si el usuario indica un valor para fila o columna erróneo deberá indicárselo con un mensaje y volverlas a solicitar.

   El alumno podrá optar por enumerar las casillas, para el usuario, desde el 1 o bien desde el 0.

c. Un programa principal que nos permita jugar al tres en raya haciendo uso de la clase y función indicadas anteriormente.

   El programa permitirá jugar tantas veces como quieran los jugadores, de modo que tras finalizar cada partida, preguntará si se desea volver a jugar, actuando en consecuencia.

   Al comenzar la partida se mostrará el tablero (que estará en blanco), y la primera ficha en poner será siempre la X. Cada vez que un jugador indique una coordenada válida el proceso a seguir será el mismo:

   1) Se comprueba si la ficha se puede colocar en el tablero, pues sólo estarán disponibles las casillas vacías.

2) Se muestra cómo queda el tablero con la tirada realizada por el jugador al que le corresponda.
3) Se comprueba si se ha conseguido realizar 3 en raya:
    a. Si se ha conseguido, se le muestra un mensaje de enhorabuena al jugador que ha ganado y la partida finaliza.
    b. Si no se ha conseguido, se comprueba si el tablero se ha completado:
        i. Si estuviera completo, se mostrará un mensaje por pantalla indicando que los jugadores han empatado.
        ii. En caso contrario, se cambiará de jugador para realizar la siguiente tirada.

Si la partida ha finalizado se le preguntará a los jugadores si quieren volver a jugar, en caso de no haber finalizado se realizará otra tirada por parte del

Ejercicio 3

Realizar un programa en c++ para controlar las cuentas corrientes de una persona. Para ello definimos la siguiente clase que es capaz de almacenar y gestionar una sola cuenta.

```
typedef char Cadena[50];  //Tipo de datos Cadena
#define MAX_CUENTAS 100   //Número de Cuentas

class Cuenta //Contiene los datos de una cuenta bancaria
{
        float Saldo;       //Saldo de la cuenta
        int NoCuenta;      //Número de la cuenta
        bool Bloqueada;    //true si está bloqueada
    public:
        Cuenta();
        Cuenta(int pNo, float pSal);
        bool ActualizarSaldo(int pSal);
        void ActualizarBloqueo(bool pBloq);
        float DameSaldo();
        int DameNoCuenta();
        bool EstaBloqueada();
};
```

Los atributos **Saldo**, **NoCuenta** y **Bloqueada** almacenarán tal como indica su identificador el saldo de la cuenta, el número de cuenta y si está o no bloqueada.

Diseñe las siguientes funciones y los siguientes métodos para esta clase:

1. Un **constructor** sin parámetros donde se inicialice el saldo, y número de cuenta a 0 y bloqueada a false.

2. Un constructor parametrizado que inicializará los atributos de la clase con el valor de los parámetros, **pNo** (número de cuenta) y **pSal** (saldo inicial de la cuenta). El constructor inicializará el atributo **bloqueada** a **false**.
3. **bool ActualizarSaldo(int pSal);** Este método actualizará el atributo **Saldo** con el valor del parámetro **pSal** siempre y cuando la cuenta no esté **bloqueada**. El método devuelve **true** si se ha actualizado el saldo y **false** en caso contrario.
4. **void ActualizarBloqueo(bool pBloq);** Este método actualizará el atributo **Bloqueada** con el valor del parámetro **pBloq**.
5. **float DameSaldo();** Método que devuelve el atributo **Saldo** de la cuenta.
6. **int DameNoCuenta();** Método que devuelve el número cuenta (**NoCuenta**).
7. **bool EstaBloqueada();** Método que devuelve **true** si la cuenta está bloqueada, **false** en caso contrario.
8. **int BuscarCuenta(Cuenta Ctas[MAX_CUENTAS], int NCuentas, int NoCuenta)** función genérica que recibe un vector de cuentas (**Ctas**), cuantas cuentas están utilizandose (**NCuentas**) y el número de cuenta a buscar (**NoCenta**).

   Esta función devuelve la posición dentro del vector de cuentas que contiene el número de cuenta especificado por parámetro. Si no existe ninguna cuenta en el vector con ese número de cuenta devolverá **-1**.
9. **int MenuCuentas()** función genérica que mostrará el siguiente menú y devolverá la opción seleccionada. Las opciones del menú son las siguientes:

```
    Menú Gestión de Cuentas
 1 Añadir una cuenta a un cliente
 2 Mostrar las cuentas del cliente
 3 Borrar una cuenta del cliente
 4 Modificar Saldo de una cuenta
 5 Modificar Estado de una cuenta
 6 Salir
 Elige Opción:
```

10. Diseñe un **main** que compruebe el funcionamiento correcto de los métodos de esta clase. Para ello definirá las siguientes variables locales de contendrán las cuentas de un cliente de banca:

```
Cuenta DatosCuentas[MAX_CUENTAS];
int nCuentas=0;
```

DatosCuenta es un vector de objetos de tamaño **MAX_CUENTAS** y **nCuentas** contiene el numero de objetos del vector que se están utilizando.

El **`main`** mostrará el menú y realizará las siguientes acciones según la opción elegida.

Opción 1: Si hay espacio suficiente en el vector de cuentas (**`DatosCuenta`**), solicitará por teclado el número de la nueva cuenta y a continuación buscará en el vector de cuentas si existe ya una cuenta con dicho número. Si existe mostrará un mensaje de error, y en caso contrario solicitará el saldo de la cuenta y actualizará el vector de cuentas con una nueva cuenta.
En caso de no haber espacio suficiente mostrará un mensaje de error.

Opción 2: listara por pantalla los datos de cada cuenta del vector de cuentas (**`DatosCuenta`**).

Opción 3: Solicitará por teclado el número de cuenta a ser eliminada y buscará una cuenta en el vector que contenga dicho número de cuenta. Si no la encuentra mostrará un mensaje de error, si la encuentra eliminará dicha cuenta desplazando todos los objetos que hay en el vector de cuentas una posición hacia la izquierda a partir de la posición de la cuenta a eliminar.

Opción 4: Solicitará por teclado el número de cuenta a ser actualizada y buscará dicha cuenta en el vector de cuentas. Si no la encuentra mostrará un mensaje de error, y si la encuentra actualizará dicha cuenta con el saldo solicitado por teclado. Si no se actualiza el saldo de la cuenta, mostrará un mensaje de error por estar la cuenta bloqueada.

Opción 5: Solicitará por teclado el número de cuenta a ser actualizada y buscará dicha cuenta en el vector de cuentas. Si no la encuentra mostrará un mensaje de error, y si la encuentra solicitará por teclado un carácter (s o n) para indicar si se desea bloquear la cuenta o no. A continuación actualizará el bloque de la cuenta según la opción elegida.

Opción 6: Termina el programa.

Ejercicio 4

Realizar un programa en c++ para gestionar los clientes de un banco. Este problema reutiliza prácticamente todo el código del ejercicio anterior de Cuentas. Para almacenar un cliente y todas sus cuentas bancarias definimos la siguiente clase:

```
typedef char Cadena[50];  //Tipo de datos Cadena
#define MAX_CUENTAS 10    //Número de Cuentas
#define MAX_CLIENTES 100   //Número de clientes

class Cliente
{
        Cadena Nombre;      //Nombre y dirección
        Cadena Direccion;
        Cuenta Cuentas[MAX_CUENTAS]; //cuentas corrientes
        int NoCuentas;               //Nº de cuentas abiertas
    public:
        Cliente();
        void ActualizarCliente(Cadena pNomb, Cadena pDir);
        void DameNombre(Cadena pNom);
        void DameDireccion(Cadena pDir);
        int BuscarCuenta(int pNoCuenta);
        bool CrearCuenta(Cuenta pCu);
        bool ActalizarCuenta(Cuenta pCu);
        bool BorrarCuenta(int pNoCuenta);
        int DameNoCuentas();
        Cuenta DameCuenta(int pos);
        void Mostrar(char Campo);
};
```

Los atributos **Nombre**, **Direccion**, **Cuentas** y **NoCuentas** almacenarán tal como indica su identificadores el nombre y dirección del cliente, cuentas corrientes (vector de objetos de clase **Cuenta**) y el número de cuentas que posee el cliente.

Diseñe las siguientes funciones y los siguientes métodos para esta clase:

1. Un **constructor** sin parámetros donde se inicialice las cadenas a vacio y el número de cuentas a cero.
2. **`void ActualizarCliente(Cadena pNomb, Cadena pDir);`** Método que actualiza los atributos nombre y dirección con los parámetros **pNomb** y **pDir** respectivamente. El atributo número de cuentas (**NoCuentas**) se inicializa a **0**.
3. **`void DameNombre(Cadena pNom);`** Método que devuelve el nombre del cliente mediante el parámetro **pNom**.

4. **`void DameDireccion(Cadena pDir);`** Método que devuelve la dirección del cliente mediante el parámetro **`pDir`**.
5. **`int BuscarCuenta(int pNoCuenta);`** Método que buscará en todas las cuentas del cliente aquella cuyo número de cuenta coincide con el valor del parámetro **`pNoCuenta`**.
6. **`bool CrearCuenta(Cuenta pCu);`** Método para crear una cuenta nueva al cliente siempre y cuando tenga espacio, en caso de no tenerlo el método devuelve **`false`**.

   El objeto cuenta ya inicializado es pasado al método mediante el parámetro **`pCu`**. El método buscará entre las cuentas del cliente aquella cuyo número de cuenta coincida con el que posee la cuenta pasada por parámetro.

   Si no la encuentra, asignara el objeto **`pCu`** al final del vector de cuentas del cliente y devolverá **`true`** y en caso contrario el método solo devolverá **`false`**.
7. **`bool ActalizarCuenta(Cuenta pCu);`** Método que actualizar la cuenta del cliente con los datos de la cuenta pasada por parámetro. Para ello, el método buscará aquella cuenta del cliente cuyo número de cuenta coincida con el que posee el objeto cuenta **`pCu`**. Si la encuentra actualizará la cuenta del cliente y devolverá **`true`**, en caso contario devolverá **`false`**.
8. **`bool BorrarCuenta(int pNoCuenta);`** Método que elimina la cuenta del cliente cuyo número de cuenta es pasado por parámetro. Si la encuentra el método la eliminará del vector de cuentas y devolverá **`true`**, en caso contrario devolverá **`false`**.
9. **`int DameNoCuentas();`** Método que devuelve el número de cuentas que posee el cliente.
10. **`Cuenta DameCuenta(int pos);`** Método que devolverá la cuenta del cliente que está en la posición que indica el parámetro **`pos`**.
11. **`void Mostrar(char Campo);`** Método que mostrará el nombre y todas las cuentas del cliente según indique el parámetro **`Campo`**. Si **`Campo`** contiene **`'s'`** o **`'t'`** mostrará el nombre y la dirección del cliente. Si el **`Campo`** contiene **`'c'`** o **`'t'`** mostrará todas las cuentas del cliente.
12. **`int BuscarCliente(Cliente Ctes[MAX_CLIENTES], int NCtes, Cadena Nombre);`** Función genérica que busca un cliente en el vector de cliente **`Ctes`** cuyo nombre coincidida con el nombre pasado por parámetro **`Nombre`**. El parámetro **`NCtes`** indica el número de objetos cliente que tienen información en el vector **`Ctes`**.

    Esta función devolverá la posición del cliente encontrado o **`-1`** si no lo encuentra.
13. **`int Menu();`** Función genérica que mostrará el siguiente menú y devolverá la opción seleccionada. Las opciones del menú son las siguiente:

```
        Menú Principal
  1 Añadir un cliente
  2 Actualizar Dirección del Cliente
  3 Mostar un cliente
  4 Mostar todos los clientes
  5 Submenú Gestión de Cuentas
  6 Salir
  Elige Opción:
```

14. **`int MenuCuentas()`** función genérica que mostrará el siguiente menú y devolverá la opción seleccionada. Las opciones del menú son las siguientes:

```
    Menú Gestión de Cuentas
  1 Añadir una cuenta a un cliente
  2 Mostrar las cuentas del cliente
  3 Borrar una cuenta del cliente
  4 Modificar Saldo de una cuenta
  5 Modificar Estado de una cuenta
  6 Salir
  Elige Opción:
```

15. Diseñe un **main** que compruebe el funcionamiento correcto de los métodos de esta clase Cliente. Para ello definirá las siguientes variables locales de contendrán los clientes de un banco:

```
Cliente Datos[MAX_CLIENTES];
int nClientes;
```

Datos es un vector de objetos de tamaño **MAX_CLIENTES** y **nClientes** contiene el numero de objetos del vector que se están utilizando.

El **main** mostrará el menú y realizará las siguientes acciones según la opción elegida.
Opción 1: Solicitará por teclado el nombre y la dirección del cliente a crear, a continuación actualizará con estos datos el objeto correspondiente del vector de clientes.

Opción 2: Solicitará por teclado el nombre del cliente a actualizar, lo buscará en el vector de clientes y si lo encuentra solicitará la dirección y actualizará el clientes correspondiente con la información solicitada. Si no lo encuentra mostrará un mensaje de error.

Opción 3: Solicitará por teclado el nombre del cliente a mostrar y después lo buscará en el vector de clientes. Si lo encuentra muestra toda sus información, si no mostrará un mensaje de error.

Opción 4: Mostrará todos datos de los clientes así como todas sus cuentas corrientes.

Opción 5: Solicitará por nombre cuyas cuentas va a ser gestionadas. Si lo encuentra dentro del vector de clientes mostrará un submenú con el mismo contenido que el ejercicio anterior y todas las operaciones descritas en dicho ejercicio utilizarán el objeto del vector cliente que ha sido localizado. Si no encuentra el cliente mostrará un mensaje de error.

Opción 6: Termina el programa.