Universidad de Jaén
Departamento de Informática

Programación Orientada a Objetos
http://bit.ly/poouja

Curso 2019/2020

José Ramón Balsas Almagro
Víctor Manuel Rivas Santos
Ángel Luis García Fernández
Juan Ruiz de Miras

Práctica 4. Clases y Objetos (3)

Índice

Objetivos

- Conocer el formato CSV como método de representación de información estructurada
- Saber declarar, inicializar y usar variables de clase.
- Gestión de errores con excepciones que son clases.

Contexto de la práctica

El contexto sigue siendo el de las prácticas 2 y 3. No obstante, esta es la última sesión en la que vamos a seguir trabajando en el programa del DJ Segismundo Barcetres Rodríguez.

Material de partida

- Puedes usar el proyecto que realizaste en la práctica anterior. Cópialo a un nuevo proyecto antes de hacer ninguna modificación y podrás conservar los dos.
- Si lo deseas, también puedes descargarte el código que hemos creado en GitHub o incluso hacerle un fork y empezar a trabajar a partir de él. Lo puedes localizar en

http://bit.ly/poouja1617pr4

Ejercicios

1. Crea la clase *ParametroNoValido*; servirá para lanzar excepciones de este tipo en los ejercicios que vendrán a continuación. La clase debe tener los siguientes atributos:
    a. *Fichero*: String. Nombre del fichero en el que se realiza el throw de la excepción
    b. *Funcion*: String. Nombre de la función o método en el que se realiza el throw
    c. *Descripción*: String. Una descripción de qué parámetro no es válido y por qué

   Además del constructor parametrizado, añadir a la clase un método llamado *queOcurre()*, que devuelva un string concatenando los atributos *Fichero, Función y Descripción*.

2. Modifica aquellos métodos de la clase *Garito* en los que se lanzaban excepciones simples, de forma que ahora lancen excepciones de clase *ParametroNoValido* correctamente construidas. Añade el método *incrementarPuntuacion (int)* en la clase *Temazo*, que recibe una cierta cantidad de puntos a sumar a la puntuación del temazo. Este método lanzará una excepción de tipo *ParametroNoValido* si la cantidad de puntos que se le pasa como parámetro está fuera del rango [-10,10].

3. Añade a la clase *Temazo* la variable de clase *numTemazos* de tipo entero. Su valor inicial será 0. Añade también el atributo *idTemazo*, también de tipo entero.

4. Modifica los constructores de la clase Temazo para que cada vez que se cree un nuevo objeto se incremente en 1 la variable de clase, y se asigne el valor de esta variable al atributo *idTemazo* del nuevo objeto. Añade el método *getId()* en la clase Temazo, que devolverá el identificador del temazo, tal y como se describe en este ejercicio. Modifica también la función *mostrarTemazo* del módulo de funciones auxiliares, para que también muestre el identificador del temazo.

5. Añadir a las clases *Temazo*, *Garito* y *Fecha* el método ***toCSV()***, que devuelve una cadena con los valores de los atributos de cada objeto, separados por punto y coma, y en el siguiente orden:
    - Para Fecha: *año;mes;dia*
    - Para Garito: *nombre;direccion*
    - Para Temazo: *Id;Título;Intérprete;Duración;Puntuación;NombreUltimoGarito;FechaUltimoUso*

6. En la función main, y teniendo en cuenta que debes capturar y manejar las excepciones que pudieran lanzarse:
    a. Crea un vector de punteros que permita almacenar hasta 10 objetos de tipo Garito. Crea dos objetos para los dos primeros punteros del vector, y cambia su nombre y dirección.
    b. Crea un vector de 5 objetos de tipo Temazo. Modifica las 3 primeras posiciones, con datos (título, intérprete…) de tu elección. Todos deben tener su puntuación a 0, y haberse escuchado por última vez en alguno de los garitos creados anteriormente.
    c. Implementa dos ciclos que recorran los dos vectores, mostrando la información almacenada en los objetos, utilizando el formato CSV.

Conceptos teóricos

El formato CSV

CSV es el acrónimo de "Comma Separated Values" y hace referencia a un tipo de fichero de formato texto utilizado para almacenar información simple de una hoja de cálculo. A grandes rasgos, podemos decir que cada fila de la hoja de cálculo se almacenará en una línea diferente del archivo (delimitada por el carácter de fin de línea). En cada línea, los valores de cada columna se almacenarán separados por un carácter específico, que normalmente es una coma "," o un punto y coma ";", aunque podrían ser otros. Utilizar uno u otro dependerá de la herramienta con la que queremos exportar/importar la información del fichero (Excel utiliza por defecto ";" mientras que otras hojas de cálculo utilizan ","), y de la convención que se utilice para los decimales (en unos países se usa un punto para separar los decimales, y en otros la coma). Ejemplos de datos en formato CSV son:

```
Juego de tronos ; HBO ; 2011 ; English
Isabel ; RTVE ; 2011 ; Español
```

Lo importante de este tipo de archivos es que son una primera aproximación al almacenamiento de información estructurada y, además, permiten el intercambio de información entre diferentes aplicaciones. El motivo de esto último es que, al ser ficheros en formato texto, su contenido puede ser observado fácilmente con cualquier herramienta y, al estar estructurado (en filas y columnas), puede construirse fácilmente una aplicación que sea capaz de recuperar la información almacenada. De hecho, como hemos comentado, varias aplicaciones comerciales o libres existentes permiten actualmente exportar o importar contenidos utilizando ficheros en este formato.

Por último, comentaremos que aunque el formato CSV es el más extendido por su sencillez, tiene limitaciones a la hora de almacenar información con una estructura compleja. Por este motivo se han desarrollado soluciones mucho más potentes para almacenar información estructurada en ficheros de texto, como son XML o JSON.

Traducción de objetos a formato CSV

Un enfoque habitual para poder almacenar objetos en algún sistema de almacenamiento externo consiste en transformarlos en un formato que sea sencillo de almacenar y posteriormente recuperar, reduciendo la complejidad inherente de la propia estructura del objeto. Una primera aproximación simplificada a este problema podría consistir en utilizar el formato CSV para almacenar un objeto simple, organizando la información de sus atributos en una línea y separados por un carácter específico, como puede ser el punto y coma (**;**). Para ello, podríamos dotar a nuestras clases de métodos que permitan obtener la representación en formato CSV de un objeto, o de un constructor y/o método que permita inicializar un objeto a partir de una representación del mismo en formato CSV. A modo de comentario, este proceso de transformar un objeto en una representación que pueda almacenarse o transmitirse se le conoce con el nombre de serialización (*serialization* en inglés).

Ejemplo de uso de la conversión a CSV:

```
int main () {
   Plato primeros[5];    // Vector de platos
   /*
      Trabajo con los objetos: asignación de valores…
   */
   // Finalmente, se muestran en formato CSV en pantalla
   for ( int i=0; i < 5; i++ )
   {
      std::cout << primeros[i].toCSV () << std::endl;
   }
   return 0 ;
}
```

El siguiente paso consistirá en encontrar una forma de transformar los atributos de un objeto en una secuencia de caracteres donde los diferentes valores están separados por un carácter específico, como por ejemplo el punto y coma (“;”). Simplificando mucho el problema, nos quedaremos únicamente con atributos de tipo simple, como pueden ser cadenas de caracteres y números (enteros y reales). En el caso de los atributos de tipo cadena de caracteres no hay problema, puesto que ya tienen una representación de texto. Sin embargo, en el caso de los atributos de tipo entero o real, necesitamos transformarlos de su representación binaria a su

representación en cadena de caracteres. Para ello utilizaremos la clase *std::stringstream* de la biblioteca estándar <*sstream*> de C++.

Podemos considerar que un objeto de la clase *stringstream* tiene la misma funcionalidad que los objetos *cin* y *cout*; es decir, sus mismos métodos y operadores, con la diferencia que las operaciones realizadas no tienen repercusión sobre la pantalla o el teclado. Por el contrario, las operaciones realizadas sobre un *stringstream* se realizarán sobre una estructura interna que podrá ser recuperada posteriormente como una cadena de caracteres. Veamos un ejemplo:

```
#include <sstream>
#include <string>
#include <iostream>

int main () {
   std::stringstream ss;
   std::string nombre("Francisco Manuel"),
               apellidos("Gómez Pérez"), resultado;
   int edad=34;
   float peso=81.5;

   // Enviamos información al objeto stringstream
   // Los operadores << van transformando en texto cada tipo
   // de dato
   ss << nombre << ";" << apellidos << ";" << edad << ";"
      << peso;

   // Recuperamos el contenido de ss en una cadena
   // Podría usarse: ss>>resultado, pero usamos str
   // para que también se incluyan los espacios en blanco
   resultado = ss.str ();

   // Mostramos la cadena:
   // Francisco Manuel;Gómez Pérez;34;81.5
   std::cout << resultado <<std::endl;
   return 0;
}
```

En la siguiente sesión estudiaremos cómo utilizar la clase *stringstream* para realizar la lectura

de objetos almacenados de esta forma. Para más detalles sobre la manipulación de flujos de texto, es recomendable consultar el capítulo correspondiente del libro de Joyanes (2010) propuesto en la bibliografía, o cualquier otro que trate sobre C++.

Variables de clase

Al definir una nueva clase, definimos los atributos y métodos que tendrán los objetos que definamos posteriormente. Normalmente, cada uno de los objetos que definamos almacenará sus valores específicos para cada uno de los atributos, es por ello que normalmente a los atributos se les llama variables de instancia, porque sus valores variarán para cada una de las instancias declaradas.

Sin embargo, es posible definir atributos que estén compartidos por todos los objetos de una misma clase; son las variables de clase. Así, para cada objeto instanciado se reservará memoria para cada una de sus variables de instancia, sin embargo sólo se reservará memoria una vez para cada variable de clase. Todos los objetos de la clase compartirán dicha variable, pudiendo acceder a ella y modificarla (en caso de que no se haya definido como constante).

La inclusión de una variable de clase se realiza poniendo la palabra reservada ***static*** delante de la declaración del atributo, como en el siguiente ejemplo:

```
static int contador;
```

En el siguiente ejemplo mostramos una implementación de la clase *Fecha* con una variable de clase que almacena el instante en que se creó la última instancia perteneciente a dicha clase.

```
/// @file Fecha.h
#include <ctime>
class Fecha {
private:
   int dia, mes, anio;
   static time_t ultimaInstanciacion;
public:
   Fecha ( int d, int m, int a, time_t &demora );
   int getDia() const;
   void setDia( const int d );
private:
   void incrementaAnio();
};
```

La nueva variable de clase creada puede ser usada desde cualquier método de la clase.

Continuando con nuestro ejemplo, vamos a mostrar cómo se podría indicar desde el constructor de la clase los segundos que han pasado desde la instanciación del anterior objeto de la clase:

```
/// @file Fecha.cpp
#include "Fecha.h"

// Inicialización de las variables de clase: en el .cpp, y fuera de
// cualquier método
time_t Fecha::ultimaInstanciacion=time( 0 );

Fecha::Fecha ( int d, int m, int a, time_t &demora )
      : dia(d), mes(m), anio(a)
{
      // Actualiza el parámetro por referencia con el tiempo
      // transcurrido desde que se instanció el objeto anterior
      time_t actualInstaciacion = time(0);
      demora = actualInstaciacion - ultimaInstanciacion;

      // Modifica la variable de clase, para que guarde el
      // instante de la instanciación del último objeto
      ultimaInstanciacion = actualInstanciacion;
}
// No se incluye la implementación del resto de los métodos de
// la clase
```

En el ejemplo anterior, cada vez que se ejecute el constructor de Fecha nos devolverá en el parámetro *demora*, cuántos segundos han pasado desde la última vez que se instanció un objeto de esa clase. Este ejemplo muestra una característica fundamental de las variables de clase: deben ser inicializadas antes de poder ser usadas. Como se puede observar, hemos inicializado la variable dentro del fichero .cpp, en una instrucción que está fuera de cualquier método de la clase.