Programación 2

Examen de teoría (julio 2022)

7 de julio de 2022

Instrucciones

- **Duración: 3 horas**
- El fichero del primer problema debe llamarse `ej1.cc`. Para el segundo problema es necesario entregar seis ficheros, llamados `Util.cc`, `Util.h`, `Cancion.cc`, `Cancion.h`, `Catalogo.cc` y `Catalogo.h`. Si te falta algún fichero, crea uno vacío. En caso contrario no te dejara hacer la entrega. Pon tu DNI y tu nombre en un comentario al principio de todos los ficheros fuente (incluso de los vacíos)
- La entrega se realizará como en las prácticas, a través del servidor del DLSI (`http://pracdlsi.dlsi.ua.es`), en el enlace `Programación 2`. Puedes realizar varias entregas, aunque sólo se corregirá la última
- En la siguiente página web tienes disponibles las transparencias y libro de la asignatura, así como algunos ficheros de ayuda: `http://www.dlsi.ua.es/asignaturas/p2c4`
- Se valorará que el código siga la guía de estilo de la asignatura: nombres de variables/funciones, tabulación correcta, estructuración en funciones, código comentado, etc.
- Si el **código no compila**, se restará un punto de la nota final del ejercicio
- Se penalizará el uso de **librerías y funciones que no se hayan visto en clase** de teoría o prácticas
- Tened el DNI sobre la mesa para poder identificaros en todo momento

Problemas

1. **(5 puntos)**

   El banco *BlackMoney* posee un fichero binario con clientes cuyas tarjetas de crédito tienen saldo negativo. Nos han encargado realizar un programa que permita leer la información de ese fichero y generar un nuevo fichero de texto llamado `clientes.txt` donde algunos de esos datos pueden aparecer "ocultos".

   El fichero binario estará formado por una secuencia de registros con el siguiente contenido y tipo de dato:

   - DNI (entero sin signo)
   - Nombre y apellidos (cadena de texto que permita almacenar nombres de hasta 49 caracteres)
   - Código de la tarjeta de crédito (cadena de texto que permita almacenar números de 16 dígitos)
   - Saldo de la tarjeta (número real)

   Si no se puede abrir correctamente el fichero binario, o no se puede crear el de texto, deberá mostrarse un error y finalizar el programa sin realizar ninguna otra acción. Asumiremos que todos los datos del fichero son correctos, por lo que no habrá que hacer ninguna comprobación sobre los mismos.

   El fichero de texto que se genere tendrá una línea para cada uno de los clientes, donde los diferentes campos estén separados por una coma. Además, contendrá una última línea que muestre el total de la deuda (suma de todos los balances pero con signo positivo) y el promedio de deuda por cliente entre paréntesis (es decir, `total/número clientes`). Un ejemplo de fichero de salida sería el siguiente:

```
12345678,Juan Montes Pi,1849304958685943,-123.54
47567893,Emilio Padilla Linares,3495098748574632,-576.32
51234890,Ana Lucas Candela,4567382930495867,-98.76
Total: 798.62 (266.207 por cliente)
```

Dado que la información de este fichero es muy sensible, debemos de dar la opción al usuario de ofuscar (es decir, ocultar), si así lo desea, la información de la tarjeta de crédito y el DNI a la hora de escribirla en el fichero de texto. Para el caso de la tarjeta de crédito, lo que se hará es mantener los cuatro primeros dígitos y los cuatro últimos, sustituyendo el resto por asteriscos (`'*'`). En el caso del DNI, cada uno de los dígitos se sustituirá por una de las siguientes letras: `TRWAGMYFPD`. Es decir, el `0` se sustituirá por la `T`, el `1` por la `R`, el `2` por la `W` y así sucesivamente hasta el `9`, que se sustituirá por la `D`.

A continuación se muestra el ejemplo anterior con los datos de DNI y tarjeta de crédito ofuscados:

```
RWAGMYFP,Juan Montes Pi,1849********5943,-123.54
GFMYFPDA,Emilio Padilla Linares,3495********4632,-576.32
MRWAGPDT,Ana Lucas Candela,4567********5867,-98.76
Total: 798.62 (266.207 por cliente)
```

El programa podrá recibir por línea de comando los siguientes argumentos:

- `-f <fichero_entrada>`: indica el nombre del fichero de entrada binario que queremos leer. Este argumento es obligatorio
- `-dni`: indica que queremos generar el fichero de texto ofuscando el campo con la información del DNI, tal y como se ha indicado más arriba. Este argumento es opcional
- `-t`: indica que queremos generar el fichero de texto ofuscando el valor de la tarjeta de crédito. Este argumento es opcional

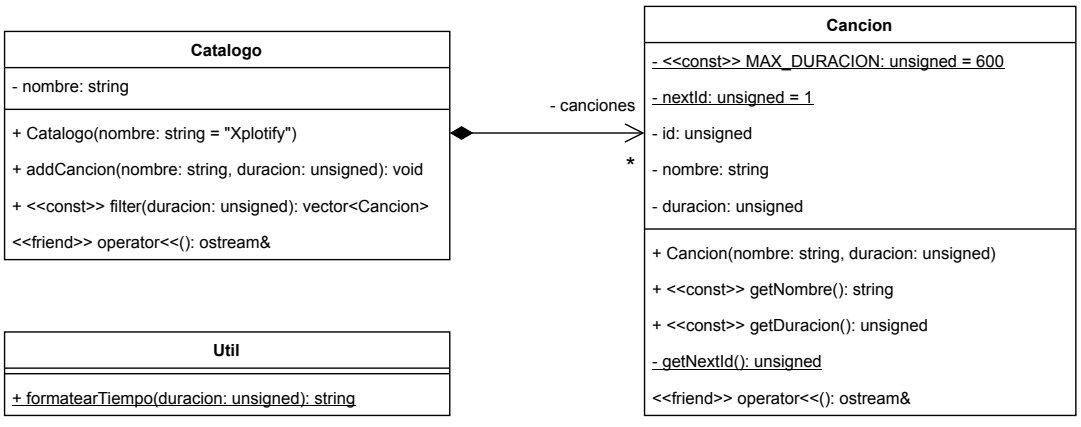
Si no se recibe el argumento `-dni` ni el argumento `-t`, se escribirá el fichero con toda la información sin ofuscar. Estos parámetros pueden aparecer en cualquier orden. A continuación se muestran algunos ejemplos de llamadas válidas:

```
$ programa -f medebespasta.bin
$ programa -f morosos.bin -t
$ programa -t -f deudores.dat
$ programa -dni -f morosos.dat -t
```

Si se introduce cualquier otro parámetro, se repite alguno de los que son válidos o falta el nombre de fichero, se debe mostrar un error por pantalla y finalizar el programa sin llevar a cabo ninguna acción.

2. **(5 puntos)**

Nos han encargado hacer un sistema para gestionar el catálogo de un servicio de música por *streaming*. De momento nos encargaremos de registrar en el catálogo las canciones disponibles. Tras reunirnos con nuestro equipo de desarrollo, diseñamos la aplicación pensando en el paradigma orientado a objetos, tal y como se aprecia en el diagrama UML de la figura que se muestra más abajo. Las clases a implementar, junto a sus métodos, se describen a continuación. **Puedes añadir nuevos métodos y atributos privados a las clases si lo necesitas, pero no públicos**.



Clase Util

La clase `Util` contiene un método auxiliar que será usado por las otras dos clases:

- `formatearTiempo`: método estático que recibe una duración en segundos y devuelve un string con la duración formateada en minutos y segundos, separados por el carácter ':'. Si los segundos son menores que 10 deberá rellenarse su valor con un cero a la izquierda. Aquí puedes ver algunos ejemplos de la salida que debe obtenerse:

  `formatearTiempo(241)` devolverá `4:01`
  `formatearTiempo(255)` devolverá `4:15`

Clase Cancion

La clase `Cancion` representa una canción de nuestro catálogo. Almacena el identificador de la canción (`id`), el nombre (`nombre`) y la duración en segundos (`duracion`). Como métodos, la clase incluye un constructor con parámetros y el *getter* de cada atributo. Esta clase también almacena, de manera común para todos los objetos, el siguiente identificador que debe considerarse cuando se incluya una nueva canción (`nextId`).

Sus métodos funcionan de la siguiente forma:

- `Cancion`: constructor que crea una canción con el nombre y la duración indicados por parámetros. Además, asigna al campo `id` del objeto el siguiente identificador, haciendo uso del método `getNextId` (explicado más abajo). Si no se indica el nombre (es una cadena vacía) o la duración excede el valor de la constante `MAX_DURACION`, el constructor debe lanzar una excepción
- `getNombre`: devuelve el nombre de la canción
- `getDuracion`: devuelve la duración (en segundos)
- `getNextId`: devuelve el siguiente identificador válido e incrementa el valor del atributo `nextId` en una unidad

Además, debe definirse el operador de salida. Este operador comenzará escribiendo el identificador, seguido por un punto y un espacio en blanco, el nombre de la canción y la duración entre paréntesis. La duración deberá formatearse haciendo uso del método `formatearTiempo` de la clase `Util`. Por ejemplo, la canción con identificador 1, nombre "Back in black" y duración 255 segundos se imprimiría así:

```
1. Back in black (4:15)
```

Clase Catalogo

La clase `Catalogo` representa el catálogo de canciones de nuestro servicio de streaming. El catálogo está compuesto por una serie de canciones. Además, cada objeto de esta clase almacena el nombre del servicio (`nombre`).

Sus métodos funcionan de la siguiente forma:

- `Catalogo`: constructor que crea un catálogo con el nombre indicado por parámetro, que por defecto tendrá el valor `Xplotify`
- `addCancion`: recibe los datos de una canción (nombre y duración) para añadirla al catálogo. El método debe crear el objeto de tipo `Cancion` e incluirlo en la lista de canciones. Si el objeto no se puede crear (ver constructor de `Cancion`), debe mostrar el mensaje `"La canción no se ha podido crear"`. Se debe usar una estructura `try/catch` para capturar la posible excepción lanzada por el constructor de `Cancion`
- `filter`: realiza una búsqueda en el catálogo de todas las canciones cuya duración sea menor o igual a la duración pasada como parámetro. Devolverá los resultados como un vector de canciones

Además, debe definirse el operador de salida. Este operador comenzará escribiendo el nombre del catálogo seguida por un salto de línea. Tras ello se mostrará la lista de canciones. Finalmente, se debe mostrar la duración total del catálogo, también formateada con el método `formatearTiempo` de la clase `Util`. Por ejemplo, un catálogo con dos canciones produciría la siguiente salida:

```
Xplotify
1. Back in black (4:15)
2. Don't talk to strangers (4:51)
Duracion 9:06
```

Programa principal

Junto al enunciado se proporciona un fichero de ejemplo `main.cc`, así como un `makefile` para hacer la compilación. Este programa contiene el siguiente código:

```
int main(){
  Catalogo catalogo;

  catalogo.addCancion("Back in black",255);
  catalogo.addCancion("Don't talk to strangers",291);
  catalogo.addCancion("The Trooper",241);
  catalogo.addCancion("Coma",613);
  cout << catalogo;

  cout << endl;
  cout << "Resultados de la búsqueda:" << endl;
  vector<Cancion> canciones=catalogo.filter(260);
  for(unsigned i=0;i<canciones.size();i++){
    cout << canciones[i] << endl;
  }
}
```

Al ejecutar este programa se debería producir la siguiente salida:

```
La canción no se ha podido crear
Xplotify
1. Back in black (4:15)
2. Don't talk to strangers (4:51)
3. The Trooper (4:01)
Duracion 13:07

Resultados de la búsqueda:
1. Back in black (4:15)
3. The Trooper (4:01)
```