Programación 2
Curso 2013/2014

Práctica 3
Minitunes (orientado a objetos)

En esta práctica se implementarán las mismas funcionalidades de la práctica 2 usando programación orientada a objetos.

Normas generales

1. El último día para entregar esta parte de la práctica es el **viernes 23 de mayo, hasta las 23:59**. No se admitirán entregas fuera de plazo.

2. Se debe entregar la práctica dividida en ficheros, y un `makefile` para compilarlos. Estos ficheros son el programa principal (`minitunes.cc`), un fichero con las declaraciones de constantes (`Constants.h`), y los archivos correspondientes a todas las clases indicadas en el diagrama: `Song.cc`, `Song.h`, `Collection.cc`, `Collection.h`, `Playlist.cc`, `Playlist.h`, `Utils.cc`, `Utils.h`. No es necesario entregar los ficheros de la clase `Player` ya que se proporcionan en el Campus Virtual.

Normas generales comunes a todas las partes de la práctica

1. Lee atentamente estas instrucciones.

2. La práctica se debe entregar exclusivamente a través del servidor de prácticas del DLSI, al que se puede acceder desde la página principal del departamento (`www.dlsi.ua.es`, "Entrega de prácticas") o directamente en `http://pracdlsi.dlsi.ua.es`.

   - No se admitirán entregas por otros medios (correo electrónico, Campus Virtual, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en el Campus Virtual.
   - La práctica se puede entregar varias veces, pero sólo se corregirá la última entrega.

3. El programa debe poder ser compilado sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se

recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.

4. La práctica debe ser un trabajo original de los alumnos; **en caso de detectarse indicios de copia de una o más prácticas se suspenderá la asignatura a todos los alumnos implicados**.

5. Los ficheros fuente deben estar adecuadamente documentados, con comentarios donde se considere necesario. Además, no se pueden utilizar variables globales de ninguna clase, los únicos símbolos globales permitidos son las funciones, los tipos y las constantes.

6. La corrección de la práctica se hará de forma automática, por lo que es imprescindible respetar estrictamente los formatos de salida que se indican en este enunciado. Además, al principio de todos los ficheros fuente entregados se debe incluir un comentario con el DNI de los alumnos que entregan la práctica, con el siguiente formato:

   `// DNI1` *tuDNI* *Primer nombre de la pareja*
   `// DNI2` *tuDNI* *Segundo nombre de la pareja*

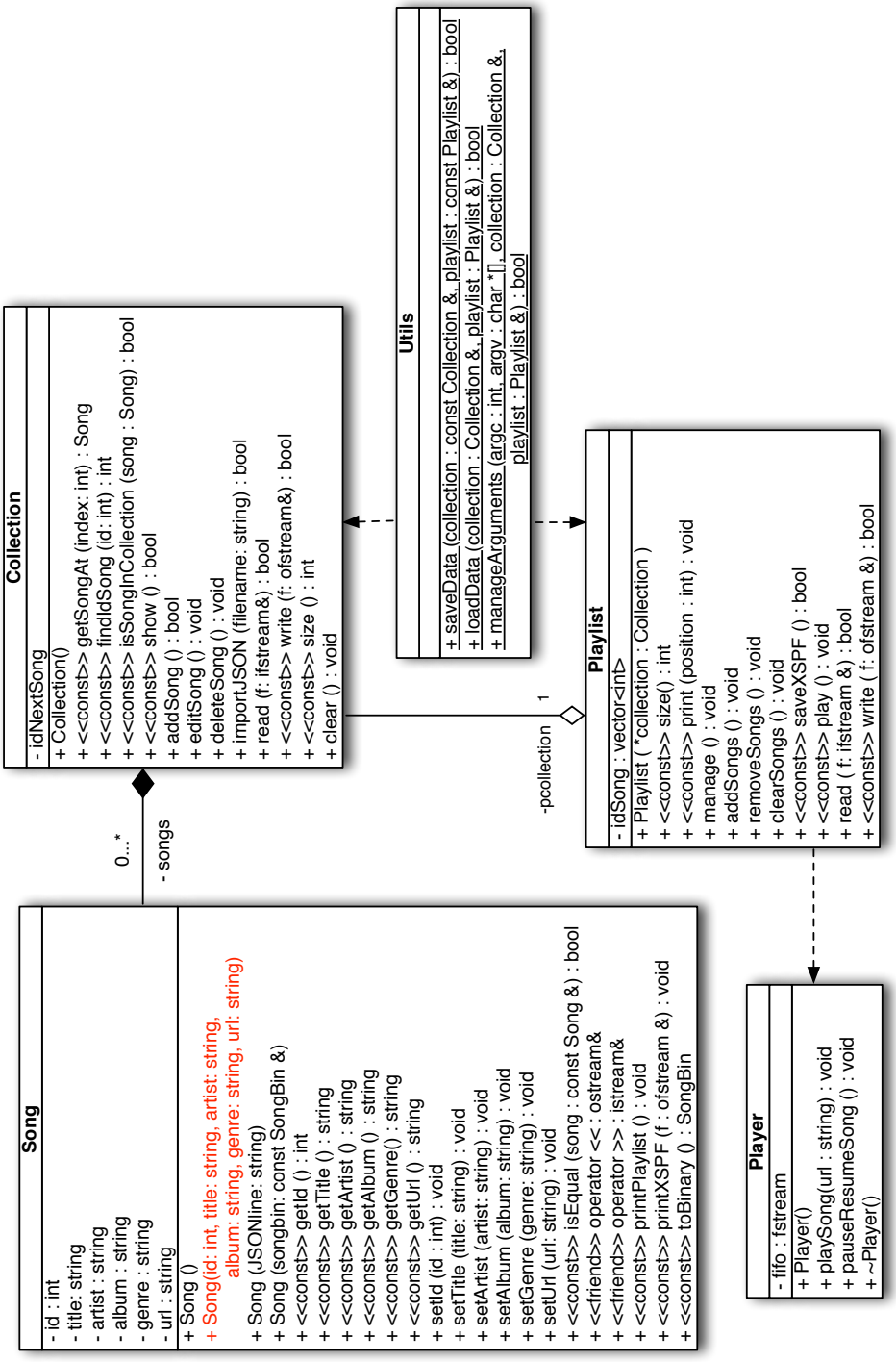
   donde *tuDNI* es el DNI del alumno correspondiente (sin letras), **tal y como aparece en el Campus Virtual**, y *Nombre* es el nombre completo; por ejemplo, el comentario podría ser:

   ```
   // DNI1   12345678  GARCIA GARCIA, JUAN MANUEL
   // DNI2   87654321  PEREZ PEREZ, ANA
   ```

7. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

8. Si quieres implementar más rápido la práctica, **imprime** este enunciado. Es mucho mejor que ir intercalando continuamente en la pantalla el código y el enunciado.

Implementación



Clases y métodos

Para implementar esta práctica es conveniente partir de la práctica anterior, añadiendo una a una las clases en el orden que aparecen en el siguiente listado y recompilando tras cada actualización. Cada vez que se añada una nueva clase será necesario modificar el fichero `makefile` para compilar. Es muy recomendable comprobar que todo funciona correctamente tras cada cambio.

Para comenzar, se deben descargar los ficheros que se proporcionan de la clase `Player` (los ficheros .cc y .h) y modificar el código correspondiente en el fichero `minitunes.cc` de la práctica 2 (función `playPlaylist`) para que compile con esta nueva clase.

Una vez funcione el programa con la nueva clase `Player` deberás empezar a crear el resto de clases en el orden que aparece en esta sección.

La primera es la clase `Song`. Para implementarla, debes quitar la definición del registro `Song` en tu código, y sustituirlo por una clase. Tras compilar, saldrán algunos errores que deberás corregir. Por ejemplo, lo que antes se implementaba como `s=song.title` ahora será `s=song.getTitle()`.

Las clases y métodos deben implementarse con el mismo nombre y con los mismos parámetros que aparecen en el diagrama. Si se considera necesario, se pueden (y deben) añadir nuevos métodos y variables en la parte privada, pero **nunca en la parte pública, que no se puede modificar.**

Aparecen también algunos métodos marcados en rojo. Significa que no necesitaréis invocarlos desde vuestro código, pero es necesario implementarlos para que funcionen las pruebas unitarias del autocorrector. Se recomienda dejarlos para el final, cuando hayáis terminado el resto de la práctica.

En el diagrama, todos los métodos que empiezan por `get` devuelven el valor indicado en su nombre y las `set` lo escriben. Las constantes que sólo se usan en una clase suelen declararse en el .cc de la misma. Si tienes alguna constante que se use en varias clases, puedes declararla en el fichero `Constants.h`.

Player

Se proporciona su implementación para comenzar la práctica.

Song

Deberemos cambiar el registro `Song` por esta clase. Para empezar, convierte los campos del registro de la práctica anterior en atributos privados. Crea un constructor sin parámetros e implementa los getters (métodos que empiezan con `get`) y setters (por `set`). Recompila. Te saldrán varios errores que deberás corregir.

Una vez arreglados los errores, implementa el resto de funciones de la clase `Song`. El método `isEqual` devolverá `true` si el título y autor de la canción actual coinciden con la que se le pasa por parámetro. El resto de

Clase	Método	Función equivalente en práctica 2
Song	Song()	Constructor para crear una canción vacía
	Song(int,string,...)	Constructor para crear una canción con parámetros
	Song(string)	getSongFromJSONline
	Song(SongBin)	binaryToSong
	printPlaylist	printPlaylistSong
	printXSPF	printXSPFSong
	toBinary	songToBinary
	operator≫	demandSong
	operator≪	printSong
	isEqual	Método para comparar dos canciones
	getters/setters	Métodos para ver o modificar los valores de los atributos
Collection	Collection	createCollection
	findIdSong	findIdSong
	isSongInCollection	isSongInCollection
	show	showCollection
	addSong	addSong
	editSong	editSong
	deleteSong	deleteSong
	importJSON	importJSON
	read	readCollection
	write	writeCollection
	getSongAt	Método para devolver una canción del vector
	size	Método que devuelve el tamaño del vector
	clear	Método que vacía el vector y pone idNextSong=1
Playlist	Playlist	Constructor a partir de un puntero
	size	Función que devuelve el tamaño del vector idSong
	print	printPlaylist
	manage	managePlaylist
	addSongs	addSongsPlaylist
	removeSongs	removeSongsPlaylist
	clearSongs	clearSongsPlaylist
	saveXSPF	saveXSPFPlaylist
	play	playPlaylist
	read	readPlaylist
	write	writePlaylist
Utils	saveData	saveData
	loadData	loadData
	manageArguments	manageArguments
Player	Player	Método para inicializar el reproductor
	playSong	playSong
	pauseResumeSong	pauseResumeSong
	~Player()	endPlayer()

Cuadro 1: Correspondencia entre los métodos de las prácticas 2 y 3. Los que están marcados en azul no tienen correspondencias.

métodos son como en la práctica anterior, tal como puede verse en la siguiente tabla. Puedes dejar para el final los operadores (operator≪, operator≫). Créalos inicialmente como métodos públicos con los nombres `printSong` y `demandSong` (como en la práctica anterior), y cuando ya tengas toda la clase implementada los puedes cambiar por operadores.

La declaración del registro `SongBin` debe hacerse en el fichero `Song.h`, justo antes de la declaración de la clase.

Collection

La clase `Collection` tiene un vector de canciones (`songs`) y el identificador de la siguiente canción a añadir (`idNextSong`). Los únicos métodos que no estaban en la práctica anterior son `getSongAt` para devolver la canción que está en la posición que se le pasa por parámetro, `size` para ver el tamaño del vector, y `clear` para vaciarlo poniendo también `idNextSong=1`.

Playlist

Para mantener las referencias entre el playlist y la colección evitando duplicar todos los datos usaremos un puntero. Para ello, implementa el constructor de `Playlist` de la siguiente manera:

```
Playlist::Playlist(Collection *collection)  {
  pcollection=collection;
}
```

Cuando declares un objeto de la clase `Playlist` en el `main`, hazlo del siguiente modo:

```
Playlist playlist(&collection);
```

De esta manera, desde el playlist podemos consultar los métodos de la colección usando `->`. Por ejemplo, para acceder al método `show` de la clase `Collection` pondríamos:

```
pcollection->show();
```

Utils

Todos los métodos de esta clase son estáticos, lo cual significa que no hace falta que creemos ningún objeto de la clase `Utils`. Si queremos acceder a alguno de estos métodos, debemos poner, por ejemplo:

```
Utils::saveData(collection,playlist);
```

Como regla general, si necesitas añadir un método que no está en el diagrama lo puedes hacer en la parte privada de la clase que lo use. En caso de que se trate de un método que se invoque desde varias clases, puedes ponerlo en la parte pública de `Utils` siempre y cuando sea estático (ésta es la única clase en la que está permitido añadir métodos en la parte pública).