Sesión L05- Agrupando objetos 2

Para realizar esta práctica vamos a crear el directorio *L05-Colecciones2*, dentro de nuestro directorio de trabajo (Practicas). Por lo tanto tendremos que hacer:

```
> cd Practicas
> mkdir L05-Colecciones2
> cd L05-Colecciones2
```

Este será el subdirectorio en el que vamos a trabajar durante el resto de la sesión. Por lo tanto cualquier fichero o directorio que creemos en esta sesión estará dentro de *L05-Colecciones2.*

Copia la carpeta que os hemos proporcionado: ***music-organizer-L05*** en tu zona de trabajo. .

01 Organizador de música

Importa el proyecto maven ***music-organizer-L05*** en tu *workspace* de Eclipse. En este ejercicio vamos a trabajar primero con la clase ***pa.music.organizer.MusicOrganizer***. Ésta se encarga de gestionar nuestros ficheros de música, de forma que podremos añadir, borrar, y consultar dichos archivos. También podremos reproducir los ficheros de audio mp3.

Primero crearemos las siguientes *Run Configurations:*

- ***Player-L05-clean*** (en la subcarpeta "***eclipse-launch-files***", igual que hemos hecho en prácticas anteriores), con la *goal*: `clean`

  Recuerda que este comando borrará el target de tu proyecto. Puedes usar esta Run Configuration cuando lo creas conveniente.
- ***PlayerListados-L05-run***, con las *goals*:

  `compile exec:java -Dexec.mainClass=pa.PlayerListados`

  En este caso compilamos y ejecutamos el el método main de la clase pa.PlayerListados.

Familiarízate con la clase *MusicOrganizer* a partir de su descripción abstracta (vista *Outline*), y los comentarios *javadoc* que se han incluido (recuerda que puedes usar la vista *JavaDoc*). A continuación, tendrás que implementar los siguientes métodos. En el código tienes incluido el javadoc correspondiente indicando lo que hace cada uno de ellos:

- ***loadSongs(String path, String extension, String[] nombresTemas)***
- ***boolean validIndex(int index)***. Este método, una vez implementado, lo debes usar en los métodos que requieran validar el valor del índice pasado por parámetro.
- ***void printFileName(int index***)
- ***void printAllFileNames()*** Mira el ejemplo de ejecución de la Figura 1 para ver cómo hay que imprimir la información
- ***void printTitleName(int index)***. Mira el ejemplo de ejecución de la Figura 1 para ver cómo hay que imprimir la información
- ***void printAllTitleNames()*** Mira el ejemplo de ejecución de la Figura 1 para ver cómo hay que imprimir la información
- ***findFirst(String searchString)***
- ***void addSong(String path, String extension, String title)***
- ***void removeSong(int index)***

A continuación debes crear una una nueva clase ***pa.PlayerListados.*** Esta clase la utilizaremos para realizar diferentes pruebas sobre el código que acabamos de implementar.

- Implementa un método *main( )*, en el que primero creamos un organizador de música (de tipo *pa.music.organizer.MusicOrganizer* (ver **Figura 1**)

  A continuación le añadiremos 7 nombres de canciones a partir de un array con los valores: "tema1A", "tema2B", "tema3A", "tema4B", "tema5A", "tema6B" y "tema7B" (usando el método que corresponda).

  Seguidamente imprime los listados de nombres de canciones, y los nombres de los ficheros. Prueba a consultar el nombre de los ficheros que ocupan las posiciones 3, -1, y 8 .

  Añade un nuevo tema con nombre "tema1C", se trata de un fichero con extensión .mp3, que se encuentra en la carpeta "mp3/nuevoAlbum/" y consulta de nuevo el nombre de fichero de la posición 8.

  Finalmente borra todos los ficheros que contienen "B" y nuestra el listado de canciones, y el de ficheros.

En la **Figura 1** puedes ver la salida por pantalla del método ***pa.PlayerListados.main( )***

```
Inicializamos nuestro organizador de música...
Listado de todos los temas:
   1. tema1A
   2. tema2B
   3. tema3A
   4. tema4B
   5. tema5A
   6. tema6B
   7. tema7B
Listado de ficheros:
   1. mp3/album1/tema1A.mp3
   2. mp3/album1/tema2B.mp3
   3. mp3/album1/tema3A.mp3
   4. mp3/album1/tema4B.mp3
   5. mp3/album1/tema5A.mp3
   6. mp3/album1/tema6B.mp3
   7. mp3/album1/tema7B.mp3
El título de la posición 3 es: tema3A
El título de la posición -1 es: Index -1 cannot be negative or zero
El título de la posición 8 es: Index 8 is too large
Añadimos un tema nuevo ("tema1C")...
El fichero mp3 número 8 es: mp3/nuevoAlbum/tema1C.mp3
Borramos  las canciones que contienen "B"
Nuevo listado de nombres de canciones:
   1. tema1A
   2. tema3A
   3. tema5A
   4. tema1C
Nuevo listado de ficheros mp3:
   1. mp3/album1/tema1A.mp3
   2. mp3/album1/tema3A.mp3
   3. mp3/album1/tema5A.mp3
   4. mp3/nuevoAlbum/tema1C.mp3
```

resultado de printAllTitles()

En ROJO mostramos los mensajes que imprimimos por pantalla usando System.out.println() desde el método main() para seguir la pista a lo que vamos haciendo

En VERDE mostramos los VALORES de las variables de tipo String que tendréis que usar en el main(): en este caso se trata de: tema1C y B

Figura 1. Salida por pantalla de la ejecución del método *pa.PlayerListados.main()*

Sube GitHub el trabajo realizado hasta el momento:

```
> git add .  //desde el directorio que contiene el repositorio git
> git commit -m"Terminado el ejercicio 1 de L05: music-organizer-Listados"
> git push
```

02 Escuchando audio con nuestro organizador de música

Vamos a seguir trabajando con el proyecto ***music-organizer.*** Los objetos de tipo *MusicOrganizer* tienen un atributo de tipo ***MusicPlayer***, el cual nos permitirá escuchar música. En la carpeta **src/main/resources/mp3** podrás encontrar varios ficheros que vamos a utilizar.

La clase ***pa.music.player.MusicPlayer*** proporciona métodos que implementan un reproductor de ficheros mp3. Puedes consultar la vista outline de esta clase y también ver su documentación *javadoc* asociada.

Los métodos de *MusicPlayer*, se usan en la clase *MusicOrganizer*, a través de su **atributo *player***, en la **Figura 2** mostramos los métodos que debes usar en tu código.

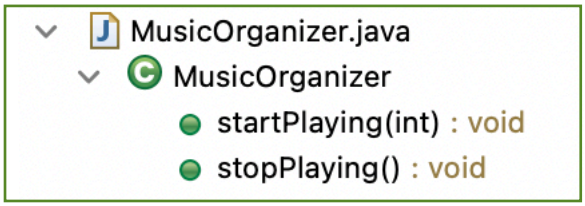


Figura 2. Métodos de *MusicOrganizer* que debes usar para iniciar y parar la reproducción

Crea la Run Configuration ***PlayerTexto-L05-run***, con las goals:

```
compile exec:java -Dexec.mainClass=pa.PlayerTexto -Dexec.cleanupDaemonThreads=false
```

Crea una nueva clase ***pa.PlayerTexto*** con un método *main( )* en el que vamos a probar los métodos de nuestro reproductor mp3 (atributo ***player***), incluido en nuestro organizador de música. Para ello:

- Crea una **instancia** de la clase *MusicOrganizer*, e inicialízala a partir de un array con los nombres de los temas de la carpeta src/main/resources. Usa para ello el método que corresponda, igual que has hecho en el ejercicio anterior. Nota: en el directorio src/main/resources/mp3 se incluye un fichero de texto con los títulos de las canciones para que no tengas que teclearlos, simplemente puedes copiar y pegar.
- **(*)** Muestra por pantalla un **listado con los títulos** de las canciones. y pedirás al usuario que introduzca un número de tema y pulse "enter". (ver Figura 3).

  Debes leer la entrada del usuario con la clase ***Scanner*** usando el método ***nextInt()***. Este método devuelve un entero introducido por teclado. A continuación debes usar el método ***nextLine()*** para leer también el "enter" que habrá introducido el usuario.

  Por ejemplo:

```
Scanner s = new Scanner(System.in);
//para leer un entero (supongamos que valor_entero es de tipo int
valor_entero=s.nextInt();
//para leer el resto de la línea, incluido el "enter"
s.nextLine();
...
//al finalizar todo el proceso, usaremos el método close()
s.close();
```

- Si el usuario introduce un número negativo o superior al número de temas almacenado en nuestro organizador de música, mostraremos el mensaje **"Entrada errónea. Intente de nuevo"**, hasta que el usuario introduzca un número correcto.
- Una vez que obtengamos un número válido, reproduciremos el tema de nuestro organizador de música que ocupe dicha posición (a través del método ***startPlaying()*** de nuestro organizador de música).

- En cualquier momento el usuario puede pulsar la tecla "enter", que leeremos con *nextLine( )* de nuestro objeto *Scanner*, y detendremos la reproducción (a través del método ***stopPlaying( )*** de nuestro organizador de música.
- Finalmente preguntaremos al usuario si quiere escuchar otro tema. Si la respuesta es "s", entonces repetiremos de nuevo todo el proceso desde **(*)**.
- Para ejecutar, **La Figura 3** muestra la salida por pantalla de una posible ejecución del método *pa.PlayerTexto.main()*.

```
Cargando las canciones en el reproductor...
   1. Best Friend - Sofi Tukker feat NERVO
   2. I Wish - Andrew Rayel
   3. Infinitely Falling - Fly By Midnight
   4. Invincible - Christina Novelli
   5. Just Around The Hill - Sash
   6. Respect_The Promise_When In Rome - Erasure_Kylie Minogue
   7. Robarte un Beso - Carlos Vives
   8. Save Your Tears - Ariana Grande
   9. Simples Corazones - Fonseca-feat Melendi
   10. Stay With Me - AVIRA
   11. Vagabundo - Sebastián Yatra
   12. With My Own Eyes - Sash
Elija el número del tema y pulse enter
6
Voy a reproducir el tema "6. Respect_The Promise_When In Rome-Erasure_Kylie Minogue"
Pulsa enter para parar...
   (aquí hemos pulsado solo la tecla enter)
Reproductor detenido
Quiere escuchar otra canción? (s/n)
s
   (volvemos a mostrar las canciones)

Elija el número del tema y pulse enter
13
Entrada erronea. Intente de nuevo:
0
Elija el número del tema y pulse enter
-2
Entrada erronea. Intente de nuevo:
8
Voy a reproducir el tema "8. Save Your Tears - Ariana Grande"
Pulsa enter para parar...
   (aquí hemos pulsado solo la tecla enter)
Reproductor detenido
Quiere escuchar otra canción? (s/n)
n

Gracias por usar nuestro reproductor!!
```

Figura 3. Salida por pantalla de la ejecución del método *pa.PlayerTexto.main()*

En ROJO mostramos los mensajes que imprimimos por pantalla usando System.out.println() desde el método main()

Resaltamos en VERDE las entradas del usuario. Cuando un usuario teclear la entrada, siempre va seguida de la tecla enter

Sube GitHub el trabajo realizado hasta el momento:

```
> git add .    //desde el directorio que contiene el repositorio git
> git commit -m"Terminado el ejercicio 2 de L05: music-organizar-PlayerTexto"
> git push
```

03 Añadimos un entorno gráfico a nuestro organizador de música

Ya hemos visto que la librería estándar de Java proporciona muchísimas clases, organizadas en paquetes, que podemos usar importándolas en nuestros proyectos. Todas estas clases disponen de su documentación javadoc, y por lo tanto tenemos la información de qué comportamientos podemos usar, cuál es su signatura, y una descripción de lo que hace cada un de los métodos de dichas clases.

Para que nuestra aplicación resulte más cómoda de usar por parte de un usuario, vamos a añadir un entorno gráfico desde el cual invocaremos a nuestro código de MusicOrganizer para escuchar canciones.

En concreto, hemos añadido el paquete ***pa.gui*** (**G**raphical **U**ser **I**nterface), que contiene la clase ***EntornoGrafico***.

Dicha clase sólo contiene dos métodos: un constructor, y el método ***startAplicacion( )***, que es el que va a mostrar una ventana con una lista desplegable y varios botones para poder usar nuestro organizador de música.

Para poder usar la clase ***EntornoGrafico*** debes sustituir el siguiente comentario del constructor de la clase por la acción que se indica en dicho comentario:

```
//aquí tienes que cargar las canciones del array que pasamos por parámetro
```

Puedes consultar el código del método ***startApplicacion()*** para familiarizarte con su funcionamiento.

Crea la Run Configuration ***PlayerGrafico-L05-run***, con los goals:

```
compile exec:java -Dexec.mainClass=pa.PlayerGrafico -Dexec.cleanupDaemonThreads=false.
```

Crea una nueva clase ***pa.PlayerGrafico*** con un método *main( )* en el que vamos a probar nuestro organizador de música, pero usando el entorno gráfico. Para ello, en el método main():

- Crea una **instancia** de la clase *EntornoGrafico.* En este caso necesitas pasar como parámetro un array con los nombres de las canciones que queremos reproducir. Es la misma colección que has usado en el ejercicio anterior, con los nombres de las canciones de src/main/resources/mp3
- A continuación simplemente tienes que invocar al método ***startAplicacion( )*** sobre el objeto de tipo *EntornoGrafico* que acabas de crear.

La Figura 4 muestra la salida por pantalla de una posible ejecución del método *pa.Mp3PlayerGrafico.main()*.

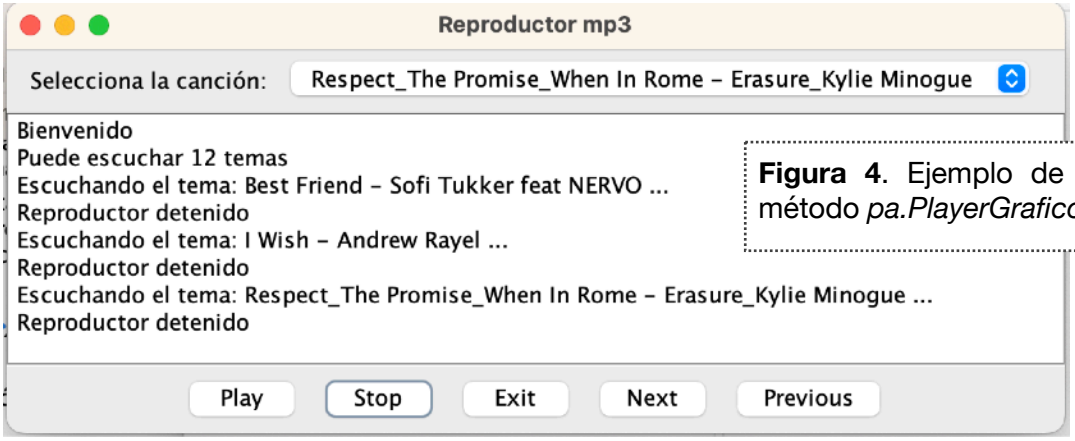


Figura 4. Ejemplo de ejecución del método *pa.PlayerGrafico.main()*

Sube GitHub el trabajo realizado hasta el momento:

```
> git add .  //desde el directorio que contiene el repositorio git
> git commit -m"Terminado el ejercicio 3 de L05: music-organizer, graficos"
> git push
```

04 Trabajando con colecciones. Creamos el proyecto desde cero

Seguimos trabajando con colecciones de elementos.

Crea primero la subcarpeta ***favoritos-L05*** (dentro de la carpeta *L05-Colecciones2*).

Ahora crea la subcarpeta ***eclipse-launch-files*** (dentro de la carpeta *favoritos-L05*)

Puedes hacerlo desde ***File→New →Maven Project*** o desde la vista Project Explorer, pulsando con botón derecho en el espacio en blanco de esa visata, con la opción ***New → Project... → Maven → Maven Project***

- Marca la casilla: "*Create a simple project (skip archetype selection)*"
- Desmarca la casilla "Use default Workspace location" y selecciona la ubicación del nuevo proyecto (usa el botón *Browse*).
  Importante: Tienes que buscar y seleccionar la carpeta *favoritos-L05* (que has creado antes.

En la siguiente pantalla introducimos los siguientes datos:

- *Group Id*: `pa`
- *Artifact Id*: `favoritos-L05`
- *Packaging*: `jar`

Una vez creado el proyecto, puedes borrar los elementos "src/test/java" y "src/test/resources" (desde la vista Project Explorer). ya que no vamos a necesitarlos.

Finalmente, edita el fichero pom.xml y añade las líneas en rojo que indicamos a continuación:

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>pa</groupId>
  <artifactId>colecciones-L05</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
  </properties>
</project>
```

La etiqueta `<packaging>jar</packaging>` es opcional. Indica que el empaquetado del proyecto es jar, pero si se omite, se asume como empaquetado por defecto "jar".

La etiqueta `<properties>` es NECESARIA para construir el proyecto.

Crea la *Run Configuration* ***maximo-L05-run*** (en la subcarpeta "***eclipse-launch-files***", igual que hemos hecho en prácticas anteriores), con las goals:

```
clean compile exec:java -Dexec.mainClass=pa.MaximoValor
```

Fíjate que aquí estamos borrando el directorio target (clean), compilando (compile), y ejecutando el método main de la clase pa.Favoritos (exec: ...) en un único comando maven.

Crea la clase ***pa.MaximoValor***, que contiene un único método (método ***main()***), en el que implementaremos lo siguiente (ver traza ejecución mostrada en la **Figura 5**).

- Pediremos al usuario que introduzca por teclado unos valores numéricos, con o sin decimales. A priori no sabemos cuántos números serán. Necesitarás almacenar los valores con tipo Double. Piensa en por qué tienen que ser de tipo Double, y no de tipo double y otro tipo.
- Dado que leeremos los datos introducidos por teclado necesitas usar la clase ***Scanner*** y usar en el constructor el parámetro ***System.in***. En clase hemos explicado por qué. Queremos indicar los decimales separados por un punto en lugar de una coma. Para ello tendremos que aplicar al objeto de tipo Scanner el método `useLocale(Locale.ENGLISH)`. Si no lo hacemos, por defecto se usarán comas para los decimales.
- Para leer los datos puedes usar los siguientes métodos sobre el objeto de tipo ***Scanner***:
  - ***hasNextDouble()***, que devuelve cierto si el siguiente valor de entrada se corresponde con un double, y falso si no lo es (por ejemplo, si hemos introducido una letra en lugar de un número)
  - ***nextDouble()***, que lee un valor doble de la entrada.
- Después de leer los datos debemos usar el método close sobre el objeto de tipo ***Scanner*** (más adelante explicaremos por qué).
- Una vez que hayas obtenido la colección de datos de entrada, debes buscar el elemento con mayor valor y mostrar por pantalla la colección de valores introducidos, "señalando" cuál es el valor mayor (si hay varios se señalarán todos ellos). (ver traza ejecución mostrada en la **Figura 5**). Resaltamos en rojo las salidas por pantalla del método main():

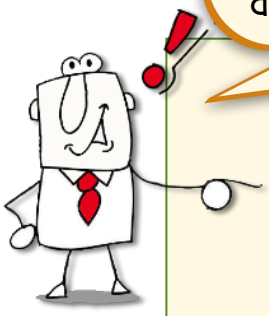
Finalmente sube los cambios a GitHub:

```
Introduzca los datos con o sin decimales
separados con espacios.
Teclee una letra para finalizar:
2 0.3 8
14 23 6
3.9 23
c

Valores numéricos y máximo valor:
2.0
0.3
8.0
14.0
23.0 <== máximo valor
6.0
3.9
23.0 <== máximo valor
```

El usuario puede introducir los números en la misma línea separados por espacios, o uno por línea, es indiferente.

```
> git add .   //desde el directorio que contiene el repositorio git
> git commit -m"Terminado el ejercicio 4 de L05: notas"
> git push
```



¿Qué conceptos y cuestiones me deben quedar CLAROS después de hacer la práctica?

- Cuándo usar un array o un ArrayList
- Cómo inicializar y modificar un ArrayList
- Diferentes formas de recorrer un ArrayList y diferencias entre ellas
- Cómo copiar un ArrayList en otro
- Entender bien la idea de que un objeto se pasa SIEMPRE por referencia. y qué repercusiones tiene por ejemplo con los ArrayList, o con los arrays
- Tengo que saber usar de forma "fluida" la vista Outline de Eclipse para "navegar" rápidamente de una parte del código a otra. Y también para conocer las definiciones abstractas de las clases y consultar su documentación javadoc
- Para qué sirve la sentencia "import" en Java
- Cómo crear un proyecto Maven, y cómo crear nuevas Run Configurations para ejecutar tu programa.