ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Scala:
'Scala en Ingeniería de Datos'

Autor: Charles Flores

Actualizado Noviembre 2023

CONTENTS

Esta es la documentación y enunciado del proyecto entregable de la asignatura de Scala, donde se intenta cubrir y evaluar los principales conceptos de la asignatura:

- Programación funcional
- Programación orientada a objetos
- Tipos de datos - Colecciones
- Generación de artefactos
- Pruebas unitarias

CHAPTER

ONE

DESARROLLO DE UNA APLICACIÓN DE GESTIÓN DE DATOS DE VUELOS EN SCALA

Objetivos del Trabajo: El objetivo de este proyecto final es aplicar los conceptos fundamentales de programación en Scala para desarrollar una aplicación robusta y eficiente. Los estudiantes demostrarán su habilidad para trabajar con colecciones, funciones de orden superior, definición de clases, manipulación de datos CSV, generación de artefactos y uso de ficheros de configuración.

Descripción del Proyecto: Desarrollar una aplicación en Scala que funcione como un sistema de carga y manipulación de datos. La aplicación debe ser capaz de leer, procesar y analizar datos provenientes de archivos CSV. El sistema permitirá al usuario realizar operaciones sobre los datos como filtrado, mapeo, y agregación.

Descripción de los Datos:

Se usará un dataset extraído de la página web de la Oficina de Estadísticas de Transporte de los Estados Unidos (Bureau of Transportation Statistics). Este dataset contiene información sobre vuelos comerciales en los Estados Unidos durante el año. Los datos se encuentran en formato CSV y se pueden ver más información sobre él desde el siguiente enlace

En el zip de la tarea se encontrán ficheros csv descriptivos junto con el código fuente de plantilla.

Requisitos Específicos:

1. **Manejo de Colecciones:** Utilizar diversas colecciones de Scala (como Listas, Mapas, Sets) para almacenar y procesar datos.
2. **Definición de Clases y Herencia:** Crear clases para representar diferentes entidades y relaciones. Utilizar herencia y/o traits para compartir funcionalidades entre clases.
3. **Carga y Manipulación de Archivos CSV:** Leer datos de archivos CSV. Implementar funcionalidades para filtrar y transformar estos datos.
4. **Generación de Artefactos:** Compilar la aplicación en un artefacto JAR ejecutable, asegurando que todas las dependencias estén incluidas.
5. **Uso de Ficheros de Configuración:** Configurar la aplicación utilizando ficheros de configuración externos para manejar variables como rutas de archivos, parámetros de conexión a bases de datos, etc.
6. **Funciones de orden superior:** Uso de las funciones `map`, `filter`, `flatMap`, y `foreach` para procesar colecciones de datos si son necesarias.
7. **Definición de Tests Unitarios:** Implementar tests unitarios para probar la funcionalidad de la aplicación.

Entregables:

1. **Código Fuente:** Todo el código fuente de la aplicación desarrollado por el estudiante: El zip que se envíe deberá contener:
   - la carpeta `project`
   - la carpeta `src`

- la carpeta `built.sbt`
- si el zip contiene más carpetas o ficheros esto repercutirá en la nota que se pueda obtener.

Criterios de Evaluación:

- Correcta implementación y uso de colecciones.
- Funcionalidad de carga y procesamiento de datos CSV.
- Funcionalidad de generación de ficheros serializados.
- Creación exitosa del artefacto JAR.
- Generación de ficheros de salida esperados.

Nota: Se anima a los estudiantes a realizar pruebas unitarias para asegurar la calidad del código y hacer uso de funciones de orden superior (`map`, `filter`, `flatMap`, `foreach`).

1.1 Detalles del proyecto

La aplicación Scala que vas a generar, se encargará de tomar un CSV con datos de vuelos y aeropuertos, y generar un ficheros serializados:

- con datos de los vuelos para un subconjunto de aeropuertos de origen que no se hayan retrasado.
- con datos de los aeropuertos de destino de los vuelos que se hayan retrasado.

Para ello, se proporciona un fichero CSV con datos de vuelos y aeropuertos, y un fichero de configuración con los parámetros necesarios para la ejecución de la aplicación.

El zip adjunto a la tarea contiene:

- un fichero `built.sbt` con las dependencias necesarias para la ejecución de la aplicación,
- una carpeta `project` con los ficheros necesarios para la compilación de la aplicación,
- una carpeta `src` con el código fuente de la aplicación que incluye un fichero `application.conf` con los parámetros de configuración de la aplicación,
- un fichero `flights.csv` con los datos de vuelos y aeropuertos,
- un fichero `Term.csv` con la descripción de los campos del dataset extraído de la web y que están en el fichero `flights.csv`
- un fichero `Documentation.csv` con la descripción de las columnas del dataset `flights.csv`

El contenido de la carpeta `src` es el siguiente:

```
src
├── main
│   ├── resources
│   │   └── application.conf
│   └── scala
│       ├── AirportInfo.scala
│       ├── FileUtils.scala
│       ├── FlightDate.scala
│       ├── Flight.scala
│       ├── FlightsLoaderConfig.scala
│       ├── FlightsLoader.scala
│       └── Time.scala
└── test
```

(continues on next page)

(continued from previous page)

```
    └── scala
        ├── FileUtilsTest.scala
        ├── FlightDateTest.scala
        └── TimeTest.scala
```

Todos estos ficheros, excepto los tests, deben ser modificados para completar el proyecto. Si se explora cada uno, vamos a encontrar anotaciones de la forma `TODO: ...` que indican qué se debe hacer en cada caso. **Fíjate bien en los TODOs**, también hay algunos en el `build.sbt` y en el `application.conf`.

Como se puede ver, la aplicación está dividida en:

- la carga de datos,
- el procesamiento de los mismos,
- la generación de los ficheros serializados,
- y los tests unitarios (que no hay que modificar, pero sí se pueden añadir más).

Ten en cuenta, que en todo el código fuente, **donde aparezca un `???` es que falta por implementar algo**. Hay un TODO adicional que no está puesto en el código, pero que es necesario para una correcta ejecución de la aplicación, y por consiguiente una correcta implementación y es que el código fuente del proyecto debe estar en el paquete `org.ntic.entregable`.

IMPORTANTE: El código fuente debe estar en el paquete `org.ntic.entregable`

1.2 Validación antes de la entrega

Para validar que la aplicación funciona correctamente, se proporciona un fichero de configuración `application.conf` con los parámetros necesarios para la ejecución de la aplicación y desde la raíz de tu proyecto, ejecuta:

```
sbt run
```

Si todo ha ido bien, la consola del IDE nos devolvería algo como:

```
 sbt run
[info] welcome to sbt 1.9.7 (Private Build Java 1.8.0_382)
[info] loading global plugins from /home/charles/.sbt/1.0/plugins
[info] loading settings for project flightsLoader-build from plugins.sbt ...
[info] loading project definition from /flightsLoader/project
[info] loading settings for project root from build.sbt ...
[info] set current project to flightsLoader (in build file:/flightsLoader)
[info] running org.ntic.entregable.FlightsLoader
[success] Total time: 5 s, completed 15-oct-2023 21:18:04
```

Además, en la carpeta de outputDir definida en el `application.conf` encontraremos los ficheros serializados generados por la aplicación. Si en el `application.conf` hemos establecido que el outputDir es `output` y sólo establecemos el filtrado de 3 aeropuertos (también en la configuración: "ABE", "ACK", "ACT"), el contenido de la carpeta `output` debería ser el siguiente:

```
ll /output/*.obj
/output/ABE_delayed.obj
/output/ABE.obj
/output/ACK_delayed.obj
```

(continues on next page)

(continued from previous page)

```
/output/ACK.obj
/output/ACT_delayed.obj
/output/ACT.obj
```

También recomendamos asegurarse que se genera un `jar` ejecutable, para ello, desde la raíz del proyecto ejecuta, también desde la consola del IDE:

```
sbt clean assembly
```

El resultado de la ejecución sería parecido a la de `sbt run`, pero además, en la carpeta `target/scala-2.13` encontraríamos el fichero `flights_loader.jar` que es el fichero `jar` ejecutable de la aplicación.

```
# si listamos el contenido de la carpeta target/scala-2.13
-rw-r--r-- 1 charles charles  36K oct 15 21:17 flights_loader_2.13-0.1.0-SNAPSHOT.jar
drwxr-xr-x 3 charles charles 4,0K oct 15 20:43 classes
-rwxr-xr-x 1 charles charles 6,0M oct 15 20:43 flights_loader.jar
drwxr-xr-x 2 charles charles 4,0K oct 15 20:43 sync
drwxr-xr-x 3 charles charles 4,0K oct 15 20:43 update
drwxr-xr-x 2 charles charles 4,0K oct 15 20:43 zinc
```

1.3 Entrega

INCUMPLIR ALGÚN PUNTO DE LOS SIGUIENTES REPERCUTIRÁ EN LA NOTA QUE SE PUEDA OBTENER

Como se ha comentado anteriormente, la entrega consiste en un zip con el código fuente de la aplicación desarrollado por el estudiante. El zip de entrega de la tarea debe contener única y exclusivamente:

- un fichero `built.sbt` con las configuraciones establecidas para la ejecución de la aplicación y generación del artefacto JAR y **sin comentarios de TODOs**,
- una carpeta `project` con los ficheros necesarios para la compilación de la aplicación,
- una carpeta `src` con el código fuente de la aplicación y **sin comentarios de TODOs**,
- un fichero `application.conf` con los parámetros de configuración de la aplicación.

  IMPORTANTE: El código fuente debe estar en el paquete `org.ntic.entregable`

  IMPORTANTE: NO SE ADJUNTAN LOS FICHEROS .obj GENERADOS

  IMPORTANTE: El zip de entrega de la tarea debe contener única y exclusivamente los ficheros mencionados anteriormente, si el zip contiene más carpetas o ficheros esto repercutirá en la nota que se pueda obtener.

  IMPORTANTE: El zip debe tener el nombre completo del estudiante, en mi caso mi zip de entrega sería: `charles_alfredo_flores_espinoza.zip`, todo en MINUSCULAS, SIN ESPACIOS y usando _ como separador y SIN CARÁCTERES ESPECIALES.

  INCUMPLIR ALGÚN PUNTO DE LOS ANTERIORES REPERCUTIRÁ EN LA NOTA QUE SE PUEDA OBTENER