Práctica 1: programación funcional

Nuevas tecnologías de la programación

Contenido:

1 Objetivos

En esta primera práctica se ponen en juego los conocimientos adquiridos sobre programación funcional en Java SE8. En la parte de código se han eliminado todos los acentos para evitar problemas con la codificación de caracteres.

2 Definición del problema

Se trata de implementar un sistema de gestión de alumnos para asignación de grupos de prácticas en varias asignaturas. La situación de partida es la siguiente:

- se dispone de un archivo de alumnos en el que se incluye la información relevante de todos los alumnos de un determinado curso. Para cada alumno se cuenta con la siguiente información: **dni**, **nombre**, **apellidos** y **dirección de correo**. Todos estos datos están separados por comas y cada línea contiene la información de un alumno. El nombre de este archivo es **datos.txt**. Las 5 últimas líneas del archivo son las siguientes:

```
05567893, ESTER, MERINO CASADO, esmeca@ugr.es
59978643, ESTHER, GUIJARRO DAVILA, esguda@ugr.es
45005654, ELIZABETH, SASTRE BELMONTE, elsabe@ugr.es
22472253, ROSER, REINA ALARCON, roreal@ugr.es
67769807, ORLANDO, GALVAN ANGULO, orgaan@ugr.es
```

- en archivos adicionales se ofrece la información de las asignaciones de alumnos a los grupos de práticas de cada una de las asignaturas. Los códigos de las asignaturas son **LMD**, **ES**, **MP** y **TOC** y los nombres de los archivos tienen el formato **asignacionXX.txt** (donde XX alude a los códigos indicados previamente). Los datos se organizan por líneas. En cada línea aparece un **dni** y **un valor** que indica el grupo al que se ha asignado dicho alumno. La primera línea contiene el código de asignatura y la segunda línea está en blanco. Las 5 primeras líneas del archivo de asignación de la asignatura **ES** (**asignacionES**) son:

```
ES

35871737 1
17067856 1
28636104 1
34210354 1
13718078 1
```

A partir de estos datos el sistema debe aportar la siguiente funcionalidad:

- generar una colección de objetos de la clase **Alumno**, que permitirá almacenar para cada alumno todos sus datos de interés (dni, nombre, apellidos y grupo asignado a cada una de las asignaturas consideradas) y que se obtiene como resultado de procesar los datos de los archivos.

- la colección puede soportarse en una clase llamada **Listado**. El constructor de la clase recibirá como argumento la ruta del archivo de datos de alumnos y debe construir y almacenar todos los objetos de la clase **Alumno** que sean necesarios. El dato miembro base de la clase podría ser el siguiente:

```java
/**
  * Dato miembro para almacenar a los alumnos como mapa con pares
  * <dni - alumno>
  */
 private Map<String, Alumno> lista;
```

- la clase **Listado** contará con el método **cargarArchivoAsignacion** que recibe como argumento un nombre de archivo de asignación y procesa toda su información, haciendo las asignaciones pertinentes en los objetos de los alumnos necesarios.

- se recomienda que la asignación de asignaturas se gestione mediante un mapa o diccionario, donde las claves serán las asignaturas (es conveniente usar un enumerado para ellas) y el valor el grupo asignado. Es decir, en la clase **Alumno** podríamos tener un dato miembro como el siguiente:

```java
 /**
  * Dato mimebro para almacenar las asignaciones de grupo
  */
 private HashMap<Asignatura, Integer> asignacion;
```

Se usará el valor -1 para indicar que el alumno no está asignado a ningún grupo (se usa esta marca especial para todas las asignaturas).

- **Asignatura** es un simple enumerado que podéis definir en un archivo independiente, con el siguiente contenido:

```
/**
 * Enumerado para representar los codigos de las asignaturas
public enum Asignatura {
    LMD, ES, MP, TOC
}
```

- la clase **Listado** debe constar de un método denominado **toString** que compondrá una cadena de caracteres con toda la información relativa a los alumnos.

- la clase **Listado** permitirá obtener los contadores que indiquen el número de alumnos asignados a cada grupo, para cada asignatura. El método encargado de esta funcionalidad se llamará **obtenerContadoresGrupos** y debe devolver un objeto de tipo

```
Map<Asignatura, Map<Integer,Long>>
```

- el método indicado en el punto anterior puede apoyarse en la existencia de un método auxiliar llamado **obtenerContadoresGruposDeAsignatura**, que recibe como argumento el código de una asignatura (uno de los posibles valores del enumerado) y devuelve los contadores de alumnos asignados a cada grupo. Es decir, la declaración del método sería:

```
Map<Integer,Long> obtenerContadoresGruposDeAsignatura(
                                            Asignatura asignatura)
```

- la clase permitirá también obtener un listado de todos los alumnos que no están asignados a una asignatura concreta, cuyo nombre se pasará como argumento. La declaración de este método se muestra a continuación:

```
List<Alumno> buscarAlumnosNoAsignados(String asignatura)
```

- pueden incluirse todos los métodos auxiliares que se considere conveniente, tanto en la clase **Alumno** como **Listado**. Por ejemplo, en la clase **Alumno** sería conveniente disponer de un método que indique si un alumno tiene grupo asignado en una determinada asignatura. La declaración del método podría ser la siguiente:

```
boolean cursarAsignatura(Asignatura asignatura)
```

- se usará **junit** para comprobar que la funcionalidad implementada no contiene errores. Se proporciona un conjunto de pruebas que deben integrarse en el proyecto y que deben pasarse sin errores para que se considere válido el trabajo de la práctica.

3 Detalles de implementación

Se recomienda que se usen todos los mecanismos posibles de programación funcional. Esto implica que el programa debería contar con el menor número posible de iteraciones externas y variables.

Con respecto a la forma de procesar los datos del archivo se dan las siguientes indicaciones para facilitar el trabajo. En primer lugar, para obtener un flujo con las líneas contenidas en un archivo de texto basta con usar las siguientes sentencias, pasando como argumeno al método **get** el nombre del archivo a leer:

```
// Se leen las lineas del archivo
Stream<String> lineas = Files.lines(Paths.get(nombreArchivo));
```

Una posible forma de proceder consistiría en disponer en la clase **Listado** de un método auxiliar (**crearAlumno**) que reciba como argumento una línea leída del archivo y genera el objeto de la clase **Alumno** asociado. El análisis de la línea puede hacerse de la forma siguiente:

```
// Se define el patron para las comas que hacen de separadores
Pattern pattern = Pattern.compile("(,)");

// Se procesa la linea
List<String> infos = pattern.splitAsStream(linea).collect(Collectors.toList());
```

De esta forma el resultado es una lista de objetos de la clase **String**. Cada objeto contiene la información de uno de los datos miembro de interés (dni, nombre, apellidos y correo electrónico) que se usarán en la llamada al constructor de la clase **Alumno**.

La asignación de grupos de prácticas para las diferentes asignatura se hará poteriormente. Esto hace necesario contar en la clase **Alumno** con los métodos que permitan asignar a un alumno un grupo de prácticas en una determinada asignatura.

4 Casos de prueba

Para facilitar el desarrollo se ofrece un conjunto de pruebas que permita ir comprobando el correcto funcionamiento de los métodos desarrollado. Se include aquí el código completo. Observad la forma de uso del objeto de la clase **Listado**, porque puede servir de ayuda para realizar la implementación.

```
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

import listado.Alumno;
import listado.Listado;
import listado.Asignatura;

import java.io.IOException;
import java.util.List;
import java.util.Map;

```

```java
/**
 * Práctica 1 NTP
 */
public class ListadoTest {
    private static Listado listado;

    /**
     * Codigo a ejecutar antes de realizar las llamadas a los métodos
     * de la clase; incluso antes de la propia instanciación de la
     * clase. Por eso el método debe ser estatico
     */
    @BeforeClass
    public static void inicializacion() {
        System.out.println("Metodo inicializacion conjunto pruebas");
        // Se genera el listado de alumnos
        try {
            listado = new Listado("./data/datos.txt");
        }catch(IOException e){
            System.out.println("Error en lectura de archivo de datos");
        };

        // Una vez disponibles los alumnos se leen las listas
        // de asignaciones de alumnos a cada grupo de las diferentes
        // asignaturas consideradas
        try {
            listado.cargarArchivoAsignacion("./data/asignacionES.txt");
            listado.cargarArchivoAsignacion("./data/asignacionLMD.txt");
            listado.cargarArchivoAsignacion("./data/asignacionMP.txt");
            listado.cargarArchivoAsignacion("./data/asignacionTOC.txt");
        } catch (IOException e) {
            System.out.println("Error en lectura de archivos de asignacion");
        }
        System.out.println();
    }

    /**
     * Test para comprobar que se ha leido de forma correcta la
     * informacion de los alumnos (al menos que el listado contiene
     * datos de 100 alumnos)
     * @throws Exception
     */
    @Test
    public void testConstruccionListado() throws Exception{
        assert(listado.obtenerLongitud() == 100);
    }

    /**
     * Test del procedimiento de asignacion de grupos procesando
     * los archivos de asignacion. Tambien implica la prueba de
     * busqueda de alumnos sin grupo asignado en alguna asignatura
     * @throws Exception
     */
    @Test
    public void testCargarArchivosAsignacion() throws Exception {
        // Se obtienen los alumnos no asignados a cada asignatura
        // y se comprueba su valor
        assert(listado.buscarAlumnosNoAsignados(Asignatura.LMD.toString()).size() == 2);
        assert(listado.buscarAlumnosNoAsignados(Asignatura.ES.toString()).size() == 2);
```

```
73          assert(listado.buscarAlumnosNoAsignados(Asignatura.TOC.toString()).size() == 2);
74          assert(listado.buscarAlumnosNoAsignados(Asignatura.MP.toString()).size() == 2);
75      }
76
77      /**
78       * Prueba para el procedimiento de conteo de grupos para cada una
79       * de las asignaturas
80       */
81      @Test
82      public void testObtenerContadoresGruposDeAsignatura(){
83          // Se obtienen los contadores para la asignatura ES
84          Map<Integer, Long> contadoresES=listado.obtenerContadoresGruposDeAsignatura(
85                  Asignatura.ES);
86          // Se comprueba que los valores son -1, 2 | 1, 37 | 2, 28 | 3, 33
87          Long contadoresReferencia[]={2L,37L,28L,33L};
88          Long contadoresCalculados[]=new Long[4];
89          assertArrayEquals(contadoresES.values().toArray(contadoresCalculados),
90                            contadoresReferencia);
91      }
92
93      /**
94       * Prueba del procedimiento general de obtencion de contadores
95       * para todas las asignaturas
96       * @throws Exception
97       */
98      @Test
99      public void testObtenerContadoresGrupos() throws Exception {
100         // Se obtienen los contadores para todos los grupos
101         Map<Asignatura, Map<Integer, Long>> contadores=
102                                         listado.obtenerContadoresGrupos();
103
104         // Se comprueban los valores obtenenidos con los valores por referencia
105         Long contadoresReferenciaLMD[]={2L,40L,29L,29L};
106         Long contadoresReferenciaES[]={2L,37L,28L,33L};
107         Long contadoresReferenciaMP[]={2L,31L,30L,37L};
108         Long contadoresReferenciaTOC[]={2L,33L,33L,32L};
109
110         // Se comprueban los resultado de obtenerContadoresGrupo con los
111         // de referencia
112         Long contadoresCalculados[]=new Long[4];
113         Map<Integer, Long> integerLongMap = contadores.get(Asignatura.ES);
114
115         assertArrayEquals(contadores.get(Asignatura.LMD).values().
116                 toArray(contadoresCalculados),contadoresReferenciaLMD);
117         assertArrayEquals(contadores.get(Asignatura.ES).values().
118                 toArray(contadoresCalculados),contadoresReferenciaES);
119         assertArrayEquals(contadores.get(Asignatura.MP).values().
120                 toArray(contadoresCalculados),contadoresReferenciaMP);
121         assertArrayEquals(contadores.get(Asignatura.TOC).values().
122                 toArray(contadoresCalculados),contadoresReferenciaTOC);
123     }
124 }
```

5 Material a entregar

Al final de la realización de la práctica se entregará un archivo comprimido con el contenido completo de la práctica, tal y como se integra en el proyecto con el entorno de desarrollo que hayais usado. Se incluirá también un pequeño documento indicando el entorno de desarrollo y una breve valoración de la práctica (si los conceptos vistos son novedosos, si os ha parecido de interés, problemas encontrados, etc) en tres o cuatro líneas.

La fecha de entrega se fijará más adelante. La entrega se hará mediante la plataforma **PRADO**.