UNIVERSIDAD POLITÉCNICA DE MADRID
ETS DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN
Campus Sur. Ctra. de Valencia, km. 7
28031 Madrid (ESPAÑA)

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

Programación Avanzada de Aplicaciones

Práctica 1ª: Almacén de Poblaciones

FEBRERO DE 2015

Índice

Introducción

En esta práctica se codificará una pequeña aplicación que haga uso de los conceptos vistos en teoría sobre colecciones de objetos y ordenación en Java. A la vez se utilizarán otros conceptos de programación orientada a objetos ya vistos anteriormente en otras asignaturas.

Descripción de la aplicación

Con vistas a una integración posterior como parte de un software más complejo, se desea disponer de una aplicación informática que permita guardar información sobre algunas poblaciones de algunas provincias utilizando un almacén de poblaciones.

Una población dispone de los siguientes campos definidos para almacenar su información: nombre de la población, provincia a la que pertenece y número de habitantes.

A su vez el almacén de poblaciones permitirá organizar las poblaciones y agruparlas por provincias, cada provincia estará identificada por su nombre. Dentro de cada provincia se podrán guardar las poblaciones correspondientes y realizar las operaciones necesarias para gestionarlas. Los provincias no podrán tener nombres repetidos dentro del almacén, al igual que dentro de cada provincia tampoco podrá haber poblaciones repetidas (con el mismo nombre).

Las poblaciones de cada provincia se almacenarán además ordenadas según distintos criterios de ordenación.

Para codificar la futura aplicación se ha pensado en implementar las dos interfaces siguientes que se encuentran ya definidas en el paquete *paa.provincias* (que se proporciona en formato jar), y cuya documentación javadoc de interés para esta práctica se muestra parcialmente a continuación.

Paquete paa.provincias

Interface Summary	
IAlmacenPoblaciones	Definición de la interfaz con las operaciones necesarias para gestionar un almacén de provincias y poblaciones.
IPoblacion	Definición de la interfaz con las operaciones necesarias para gestionar una población.

==========

paa.provincias
Interface IPoblacion

`public interface` **`IPoblacion`**

Definición de la interfaz con las operaciones necesarias para una población.

Method Summary	
`java.lang.String`	**`getNombre()`** Devuelve el nombre de la población
`java.lang.String`	**`getProvincia()`** Devuelve la provincia a la que pertenece la población
`int`	**`getHabitantes()`** Devuelve el número de habitantes de la población
`void`	**`setNombre`**`(java.lang.String nombre)` Actualiza el nombre de la población
`void`	**`setProvincia`**`(java.lang.String provincia)` Actualiza la provincia a la que pertenece la población
`void`	**`setHabitantes`**`(int habitantes)` Actualiza el número de habitantes de la población

paa.provincias
Interface IAlmacenPoblaciones

`public interface` **`IAlmacenPoblaciones`**

Definición de las operaciones disponibles en un almacén de poblaciones. El almacén está formado por provincias, cada una con su nombre único, no se permiten provincias con el mismo nombre repetido. A su vez en una provincia no se permiten poblaciones duplicadas, o sea, poblaciones con su campo nombre repetido. En cada provincia las poblaciones se almacenan siempre ordenadas, por defecto el criterio inicial es que las poblaciones están ordenadas por orden alfabético del nombre de la población.

Field Summary

static int	**ORDENARPORNOMBRE** Ordenación principal en orden alfabético por el campo nombre de la población.
static int	**ORDENARPORHABITANTES** Ordenación principal por el número de habitantes en sentido ascendente, en segundo lugar, en caso de igualdad, por el orden alfabético del campo nombre.

Method Summary

boolean	**addProvincia**(java.lang.String provincia) Añade una nuevo provincia al almacén de poblaciones si no está ya presente en el almacén. Si ya había una del mismo nombre no se realiza ningún cambio en el almacén y la llamada devuelve false.
boolean	**addPoblación**(java.lang.String provincia, paa.Provinciaes.IPoblacion población) Añade una población a la provincia correspondiente si la población no existe ya en ella, o sea, no hay ya otra población con el mismo nombre. Las poblaciones se almacenarán ordenadas según el criterio de ordenación definido.
boolean	**containsProvincia**(java.lang.String Provincia) Indica si ya existe en el almacén la provincia correspondiente.
boolean	**containsPoblación**(java.lang.String Provincia, java.lang.String nombre) Indica si ya existe en la provincia una población con ese nombre.
boolean	**delProvincia**(java.lang.String Provincia) Borra un provincia del almacén de poblaciones si está presente en el almacén. Si no hay una del mismo nombre no se realiza ningún cambio en el almacén y la llamada devuelve false.
boolean	**delPoblacion**(java.lang.String Provincia, int posicion) Borra la población situada en la posición correspondiente dentro de la provincia. La posición es un entero entre 1 y el número de poblaciones que contiene la provincia. La posición de una población es variable, puesto que cambia si se borran poblaciones de la provincia o se reordena.
boolean	**delPoblacion**(java.lang.String Provincia, paa.Provinciaes.IPoblacion población) Borra de la provincia la población que se pasa como parámetro.
boolean	**delPoblacion**(java.lang.String Provincia, java.lang.String nombre)

	Borra de la provincia la población cuyo nombre se pasa como parámetro.
`java.util.Set<java.lang.String>`	**`getProvincias`**`()` Devuelve un conjunto con los nombres de los provincias que existen en el almacén de poblaciones.
`paa.Provincias.IPoblacion`	**`getPoblacion`**`(java.lang.String Provincia, int posicion)` Devuelve la población situada en la posición correspondiente dentro de la provincia. La posición es un entero entre 1 y el número de poblaciones que contiene la provincia. La posición de una población es variable, puesto que cambia si se borran poblaciones de la provincia o se reordenan.
`paa.Provincias.IPoblacion`	**`getPoblacion`**`(java.lang.String Provincia, java.lang.String nombre)` Devuelve de la provincia la población que posee el nombre que se pasa como parámetro.
`java.util.SortedSet<paa.Provincias.IPoblacion>`	**`getPoblaciones`**`(java.lang.String Provincia)` Devuelve todas las poblaciones de la provincia correspondiente ordenadas según el criterio actual.
`int`	**`getNumPoblaciones`**`(java.lang.String Provincia)` Devuelve el número de poblaciones que contiene la provincia cuyo nombre se indica.
`boolean`	**`ordenarPor`**`(java.lang.String Provincia, int ordenarPor)` Permite cambiar el criterio de ordenación de la correspondiente provincia, las poblaciones almacenadas serán reordenadas según el nuevo criterio de ordenación que se indica como parámetro utilizando las constantes predefinidas: ORDENARPORNOMBRE ú ORDENARPORHABITANTES (ver su descripción en fields).
`boolean`	**`guardar`** `(java.lang.String nombreFichero)` Guarda el almacén de poblaciones en el fichero cuyo nombre se pasa como parámetro. Devuelve true si se ha guardado correctamente y false si hay algún tipo de error o excepción.
`boolean`	**`recuperar`** `(java.lang.String nombreFichero)` Recupera un almacén de poblaciones del fichero cuyo nombre se pasa como parámetro. Si el fichero no existe o hay algún tipo de excepción devuelve false y crea un almacén vacío. Si lo recupera, devuelve true.

En esta práctica se realizará una implementación de las dos interfaces anteriores codificando dos clases ***Poblacion*** y ***AlmacenPoblaciones*** que se corresponden con cada interface anterior respectivamente. Estas dos clases se incluirán en un paquete llamado

almacen que creará el alumno. También será necesario codificar las clases que implementen los criterios de ordenación.

Además se realizará un pequeño programa de pruebas para comprobar que funcionan todas las operaciones anteriores, tanto en el caso de que se ejecuten correctamente como incorrectamente, comprobando su funcionamiento. Este programa de pruebas no es necesario que disponga de menús interactivos con el usuario, puede ser simplemente un conjunto de llamadas para probar los métodos y una comprobación de los resultados devueltos.

Como puede verse en la documentación anterior, hay dos métodos auxiliares que se encargarán respectivamente, de guardar en un fichero binario, utilizando el mecanismo de serialización de objetos, toda la información del almacén de poblaciones, y de recuperar posteriormente del fichero binario el objeto serializado.

Desarrollo de la práctica

Diseño previo

En esta fase se elegirán las colecciones que se consideran más idóneas para la solución de la práctica y para la implementación de la clase **AlmacenPoblaciones**. Es aconsejable obtener la validación de esta fase por parte del profesor de laboratorio antes de empezar a codificar el resto de la práctica.

Fase de codificación

En esta fase se procederá a codificar la práctica, para lo que **se tendrán en cuenta los siguientes requisitos obligatorios**:

- Todas las operaciones de entrada/salida deben estar en la clase con el programa principal (la clase que implemente el programa de pruebas). En la codificación de las demás clases no se realizará ninguna operación de entrada/salida con el usuario.
- Todas las variables miembro de las clases serán declaradas con acceso privado (*private*) y siempre que sea posible como variables de instancia (sin *static*), salvo las constantes.
- Se comentarán las clases y todos los métodos adecuadamente (descripción, parámetros, valor de retorno si lo hay) para javadoc.
- Se utilizará la nomenclatura estándar de Java (mayúsculas, etc.) para los nombres de variables, métodos, etc. Y los nombres estándar de métodos *get/set* para los que acceden a atributos de la clase.

Calendario de entrega

El diseño previo debe ser validado por el profesor de laboratorio en la primera sesión **antes de comenzar a codificar la práctica,** en el grupo de laboratorio correspondiente. **Al finalizar la práctica** se entregará la siguiente documentación en formato electrónico a través de la tarea de entrega habilitada en el campus virtual en moodle:

- El proyecto Eclipse de toda la aplicación comprimido en un .zip.

La duración de la práctica es de 2 semanas.

Normas de evaluación

- No se admitirá la entrega de ninguna práctica fuera de los plazos de entrega.
- El incumplimiento de los requisitos indicados[1] para la codificación de la práctica hará que no sea evaluada.
- La práctica debe funcionar en su totalidad correctamente, por lo que se debe poner especial cuidado en las pruebas.
- La copia de las prácticas entre alumnos, implicará automáticamente la aplicación de la normativa de la UPM para estas situaciones a todos los alumnos involucrados.

[1] Si considera que necesita incumplir alguno, debe consultarlo previamente con su profesor de laboratorio.