MEMORIA PRÁCTICA 1

Jaime García Arjona – i82gaarj@uco.es
Sofía Salas Ruiz – i82sarus@uco.es

En esta primera práctica hemos realizado una iniciación al desarrollo con Java, a la vez que hemos empezado a familiarizarnos con el IDE Eclipse. Además, para la implementación de los programas requeridos en el enunciado hemos incorporado patrones de diseño.

EJERCICIO 1

En la primera parte de la práctica se nos pide implementar un *gestor de contactos*, el cual almacenará objetos de tipo Contacto con una serie de atributos (nombre, apellidos, fecha de nacimiento, email y un conjunto de intereses). Nuestro gestor deberá ser implementado mediante el patrón de diseño *Singleton*, además de proporcionar al usuario una serie de operaciones tales como: dar de alta contacto, dar de baja contacto, consultar datos de contacto, etc. Finalmente se implementará un programa Java que pruebe la funcionalidad del gestor implementado.

Para el gestor de contactos, decidimos implementarlo de la siguiente forma:

Creamos la clase Contacto, que gestiona su información. Esta clase contiene los atributos:

- Nombre (string)
- Apellidos (string)
- Email (string)
- Fecha de nacimiento (java.util.Date)
- Lista de intereses

La lista de intereses está implementada con un ArrayList de objetos del tipo Interés. Esta clase contiene:

- Un ID
- Un nombre

Además, creamos la clase GestorUsuarios, que es la que contiene todos los métodos necesarios para añadir, eliminar, y buscar contactos, y algunos métodos más. Para esta clase hemos hecho uso del patrón de diseño Singleton, ya que no debe haber más de una instancia de esta clase al mismo tiempo.

Los métodos del gestor de usuarios más relevantes son los siguientes:

- *getInstance()*: devuelve la instancia del Gestor de usuarios. Forma parte del patrón de diseño Singleton.
- *getFilePath()*: obtiene la ruta del fichero a través de un fichero de propiedades "config.properties".
- *loadContacts()*: carga los contactos desde el fichero en memoria. Una de las dificultades que tuvimos fue a la hora de convertir los temas de interés leídos del fichero de texto en objetos.
- *loadInterests()*: carga los intereses desde el fichero en memoria.

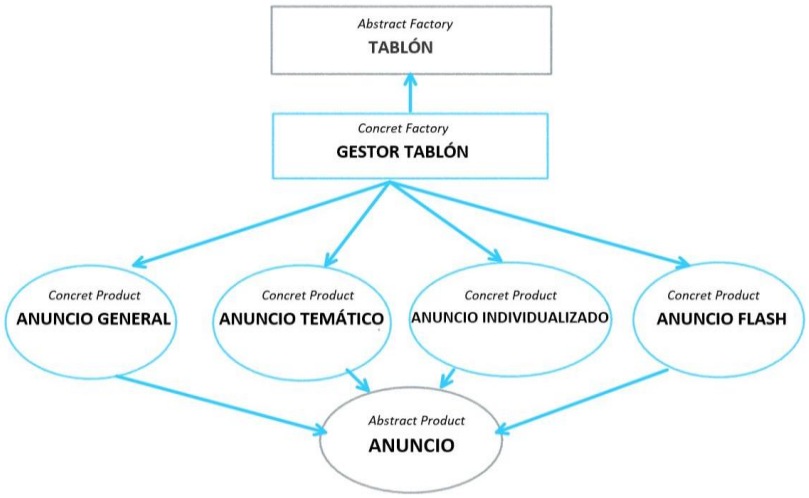
- *altaContacto()*: realiza todo el proceso de escritura en el fichero para dar de alta un contacto. Al final se vuelve a llamar a loadContacts() para que actualice los contactos en memoria.
- *bajaContacto()*: elimina un contacto del fichero.
- *actualizarContacto()*: se le pasa un email y el contacto actualizado. Da de baja el antiguo y da de alta el nuevo.
- *search()*: esta función está sobrecargada. Podemos buscar por nombre y apellidos, por email, por edad o por intereses. En caso de que encuentre uno o más contactos, los devolverá. En caso contrario, devuelve "null".
- *getInteresByID()*: devuelve el objeto de tipo Interés según el ID que se le pase. Devuelve "null" si no existe. Esta función resulta útil a la hora de buscar contactos según los temas de interés.

El programa principal (EJ1.java) comienza mostrando un menú de opciones, donde el usuario podrá registrar, eliminar o buscar contactos. El menú se vuelve a mostrar hasta que el usuario introduzca el número correspondiente a la opción de Salir.

EJERCICIO 2

Para la segunda parte de la práctica hemos tenido que implementar un *tablón de anuncios*, el cual almacenará y gestionará objetos de tipo Anuncio con unos atributos generales (identificador, título, usuario propietario, etc). Además, se nos indica que habrá 4 tipos de anuncios (general, temático, individualizado, flash) que podrán tener características adicionales. Ha sido necesario programar una factoría (patrón *Abstract Factory*). El tablón tendrá una serie de operaciones a las que podrá acceder el usuario una vez se identifique en el programa, para esta gestión de usuarios reutilizaremos el programa del Ejercicio 1.

Lo primero que tuvimos que decidir era el diseño que íbamos a darle a nuestro programa a través de la factoría, es decir, cómo íbamos a distribuir el Abstract Factory, Concret Factory, Abstract Product y Concret Product. Hemos creado un producto abstracto Anuncio, el cual tendrá los atributos comunes a todos los tipos de anuncios, que hereda en 4 productos concretos que corresponden con los tipos especificados en el enunciado. En el caso de la factoría se ha creado una abstracta que contiene las cabeceras de los métodos para después implementarlos en una factoría concreta que será el gestor del tablón.



Cada uno de esos elementos se corresponde con un fichero del proyecto, además de otro que corresponde al main del programa. A continuación, se irán detallando las decisiones y complicaciones encontradas según cada uno de los ficheros.

Anuncio.java

Hemos implementado lo que sería el constructor principal de la clase Anuncio con los atributos comunes. Además de los atributos imprescindibles que pedía el enunciado, hemos añadido al Abstract los atributos: **estado** (editado-publicado-archivado-en-espera), **fecha de publicación** y **tipo** (general-temático-individualizado-flash). El motivo para añadir estos atributos es poder implementar las operaciones que se piden para el tablón.

GeneralAnuncio.java

Solamente hemos escrito el constructor del objeto GeneralAnuncio, indicando su tipo y usando la función del Ej1 *Gestor.getContactos()* para establecer a todos los usuarios como destinatarios.

TematicoAnuncio.java

En esta clase hemos añadido como atributo un ArrayList en el que se almacenan los temas de interés que caracterizan a cada anuncio y los cuales determinarán a que usuarios se les va a mostrar dichos anuncios.

IndivAnuncio.java

Aquí solo hemos implementado el constructor, pero a diferencia del tipo general, los usuarios destinatarios son elegidos por el usuario que crea el anuncio en lugar de ser todos los contactos registrados en el sistema.

FlashAnuncio.java

Además del constructor, hemos añadido dos datos tipo Date para establecer las fechas inicio y fin entre las cuales el anuncio estará visible en el tablón.

Tablon.java

Este fichero constituye el Abstract Factory de nuestro programa, en él se encuentran todas las cabeceras de los métodos que debe ejecutar nuestro tablón. Hemos creado un método *créate* y *edit* para cada tipo de anuncio, los demás son comunes a todos.

GestorTablon.java

Aquí se han implementado todos los métodos declarados en Tablon.java. En cuanto a decisiones:

- Los usuarios destinatarios para los anuncios temáticos se han obtenido mediante una función *obtenerDestinarios(ArrayList<Interes> intereses)* la cual recibe los intereses asignados al anuncio y selecciona los usuarios que tienen al menos uno de ellos.
- Se ha creado un ArrayList por cada tipo de anuncio para almacenarlos conforme se crean y poder acceder a ellos posteriormente además de otro ArrayList tablón en el que se guardan los anuncios visibles en el tablón en ese momento.
- También se ha implementado una funcio *cambioFecha(String fecha)* que recibe un string de formato dd/MM/yyyy HH:mm y lo transforma en variable tipo Date.

- Para poder buscar cada anuncio en los ArrayList de cada tipo hemos implementado 4 funciones (nº tipos de anuncios) que recorren los ArrayList y buscan el anuncio a partir de su ID.
- Para las funciones publicar y archivar se solicita el tipo de anuncio requerido y así saber en qué ArrayList buscarlo. En el caso de la función para archivar ha sido necesaria una función complementaria que elimine el anuncio del ArrayList tablón una vez este ha sido archivado.
- Finalmente, para las funciones de mostrar los anuncios se han implementado dos funciones más. Una ordena el ArrayList tablón según la fecha de publicación y la otra alfabéticamente según el nombre del usuario propietario.
- En los métodos de editar anuncios no se puede modificar el ID.

EJ2.java (main)

- El usuario se loguea a través de su email al ejecutar el programa.
- Dispone de un menú en el que tiene a su disposición todas las funcionalidades del tablón.
- Para que el usuario se suscriba a temas de interés se reutiliza el código del Ej1.
- Antes de crear o editar un anuncio, es necesario especificar el tipo.
- Las opciones de buscar los anuncios según una serie de criterios devuelven un ArrayList que muestra por pantalla los anuncios que cumplen dichos criterios.

Complicaciones:

En general, una de las mayores dificultades que tuvimos con esta práctica, fue con la entrada por teclado. Es importante que al usar java.util.Scanner, solo debemos crear un único scanner, y siempre utilizar el método nextLine(), ya que si lo mezclamos con otros métodos como next() o nextInt(), puede dar problemas.

Para el resto del código pudimos solventar rápidamente los problemas que nos surgieron, es algo normal cuando nos iniciamos en un lenguaje nuevo.

Fuentes consultadas (Bibliografía):

https://es.stackoverflow.com/questions/39804/ordenar-una-lista-en-orden-alfab%C3%A9tico

https://es.stackoverflow.com/questions/234308/como-eliminar-un-elemento-de-un-array-por-el-contenido-del-mismo

https://es.stackoverflow.com/questions/203747/cerrar-un-programa-java-en-cualquier-momento/203749

https://docs.oracle.com/javase/8/docs/api/

https://stackoverflow.com/questions/16040601/why-is-nextline-returning-an-empty-string

https://www.journaldev.com/709/java-read-file-line-by-line#java-read-file-line-by-line-using-bufferedreader

https://stackoverflow.com/questions/10631715/how-to-split-a-comma-separated-string

https://www.javatpoint.com/java-string-to-date