

TAREA 6

Programación

Para la realización de la tarea necesito crear una clase, a la que llamo **Cliente**, que me ofrezca los datos de forma serializada para poder grabarlos en disco, por lo que ha de implementar la clase **Serializable**. Con ella lo que conseguimos es convertir los datos introducidos, en una cadena de bytes para poder realizar su grabación, y cuando vayamos a realizar la lectura de dicha información se realizará el proceso inverso, se convierte la cadena de bytes en una serie de datos de igual formato a como fueron serializados. Para asegurarnos que el formato es el mismo tanto en la clase como en la aplicación, utilizo la variable **serialVersionUID** como valor constante para que sirva como canal de datos.

En la clase Cliente declaramos las variables a utilizar como privadas, siendo **teléfono** de tipo **long** y **deuda** como **float**. El resto son de tipo **String**.

Creamos el constructor con parámetros y sus correspondientes **setters** y **getters**.

También decido sobrescribir el método **toString** con el fin que nos sea más fácil la visualización de todos los datos únicamente utilizando **elObjeto.toString**

A la clase contenedora del método **main** le llamo **Tarea6** y la creo dentro del paquete **tarea6** igual que la clase **Cliente**. Dentro de dicho main pongo el menú principal compuesto de las seis opciones solicitadas, y la única manera de abandonar el menú es seleccionando la opción 6 de finalizar. El resto de opciones nos lleva a los distintos métodos que he creado para cada una de las distintas elecciones, y que describo a continuación:

Todos los métodos de Tarea6 son privados y estáticos ya que van a ser llamados desde otro estático, ya sea desde main o desde otro método del mismo tipo dentro de la propia clase.

```
private static void introducirDatos()
```

- ✓ Llamamos al método abrir que se encarga de comprobar si existe el fichero y si es así le introduce sus valores al arreglo ArrayList definido como variable de clase personas.
- ✓ Solicitamos el NIF y comprobamos que sea correcto. En caso contrario mostraremos un aviso y volveremos a solicitarlo.
- ✓ Pedimos el nombre del cliente, y el teléfono, el cual comprobamos que se encuentre entre unos valores válidos.
- ✓ Solicitamos la dirección del cliente y la deuda de éste, que comprobamos haya sido tecleada utilizando valores numéricos (*no controlo la posibilidad de valores negativos por si se necesita introducir clientes que hayan pagado más de lo adeudado y entonces tengan una deuda negativa*)
- ✓ Cuando se terminan de introducir todos los datos en sus correspondientes variables, se añaden dichos valores al ArrayList **personas**, pero antes se comprueba si dicho arreglo está vacío y se ha de volver a crear (*cuando se elimina el fichero en la opción 5*).
- ✓ Se llama al método encargado de realizar la grabación de los datos del arreglo en el fichero y se muestra el número de registro almacenado (*utilizo el parámetro **size** que nos indica cuál es el tamaño del arreglo **personas***)

```
private static void verDatos()
```

- ✓ Método utilizado para ver todos los registros almacenados. Primero compruebo que el fichero exista, y en caso contrario nos muestra un aviso y vuelve al menú principal.
- ✓ En caso de que el fichero exista, llamo al método abrir que vuelca todos los registros en **personas**.
- ✓ si el volcado de datos es de un fichero existente pero vació se muestra un mensaje alusivo y volvemos al menú principal.
- ✓ Si existe el fichero y contiene datos, se recorren todos utilizando un bucle **for** que se encarga de mostrar todos los datos de cada registro haciendo uso del método **toString** sobrescrito en la clase **Cliente**

```
private static void buscarDato()
```

- ✓ Método para visualizar los datos de un cliente en base a su NIF, el cual solicitamos al principio.
- ✓ Cuando sabemos el dato a buscar llamamos al método **abrir** que se encarga de volcar los datos del fichero al ArrayList **personas**
- ✓ Si encontramos el registro (*hacemos un bucle for que recorra personas comprobando su NIF*) mostramos sus datos haciendo uso de nuevo del método **toString**, y marcamos que lo hemos encontrado.
- ✓ Si termina el bucle sin que se haya encontrado ninguna coincidencia lo avisaremos y ofrecemos la posibilidad de repetir otra búsqueda o volver al menú principal.

```
private static void eliminarUno()
```

- ✓ Método similar al anterior pero por cada registro que encuentre nos ofrece la posibilidad de eliminarlo o simplemente volver al menú principal.

```
private static void eliminarTodos()
```

- ✓ Primero nos aseguramos que se conteste que sí se desea eliminar el fichero
- ✓ Si contestamos afirmativamente se elimina el fichero mediante la orden **delete**, y si se ha conseguido su eliminación, se procede a borrar el ArrayList **persona** mediante su método **clear**

```
private static void abrir()
```

- ✓ Se comprueba que el fichero existe, y si no es así, se llama al método encargado de crearlo.
- ✓ Si el fichero existe y se puede leer (*tiene registros*) se vuelcan todos los datos en el ArrayList **personas** y se cierran los métodos de entrada de ficheros y objetos abiertos
- ✓ Si el fichero existe pero no se pueden leer datos es porque está vacío, por lo que se muestra el mensaje y se vuelve al menú principal.

```
private static void crearArchivo()
```

- ✓ Nos permite restablecer nuevamente el archivo de datos cuando se va a introducir información y el fichero no está creado (*cuando desde el método abrir se intenta abrir el fichero de datos y éste no existe*)

```
private static void escribirArchivo()
```

- ✓ Cuando se tienen datos almacenados en el ArrayList debemos grabarlos en el fichero de datos, para lo cual hemos de comprobar primero que dicho fichero existe, y en caso contrario lo creamos.

- ✓ Se crea un objeto de la clase `ObjectOutputStream` vinculándolo a un objeto `FileOutputStream` para escribir en el archivo en disco ***clientes.dat***.
- ✓ Escribimos el `ArrayList` en el objeto (***personas*** en el fichero ***clientes.dat***)
- ✓ Cerramos el objeto

```
private static boolean compruebaNIF(String ni)
```

- ✓ Función que comprueba la validez del NIF introducido (valor numérico entre 1000000 y 99999999, letra final válida)

```
private static boolean compruebaTel(long tel)
```

- ✓ Comprueba que el número de teléfono introducido tenga unos valores numéricos de nueve cifras y que empiezan por 6 ó por 9.