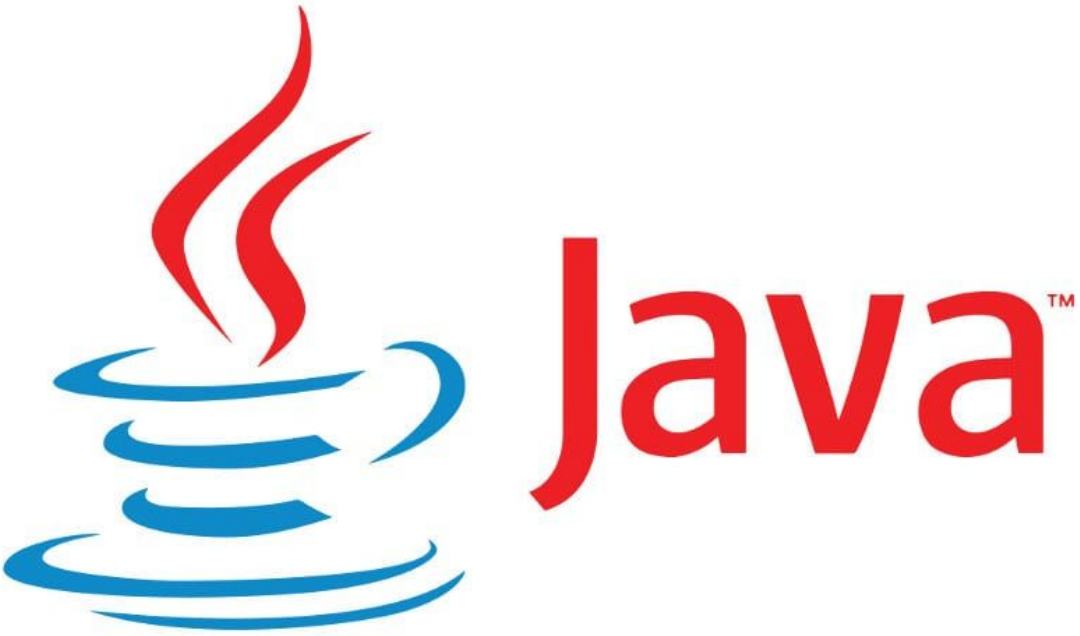
Práctica 1

Iniciación a Java



Carlos Revuelto Quero - i82requc@uco.es
José Antonio González Aguilera - i82goagj@uco.es

Para la realización del pdf primero se tendrá en cuenta las dificultades que se han ido encontrando en cada uno de los ejercicios y, posteriormente, se procederá a justificar y explicar las decisiones tomadas.

Ejercicio 1

El primer problema que se nos presentó fue en la aplicación del patrón de diseño. El problema no fue notorio hasta haber llegado a la primera función en la que se tenían que crear contactos, configurar sus atributos y luego añadirlos al gestor.

Lo que se intentó realizar al principio fue como se hubiese hecho en cualquier otra función que recibe un objeto. Crear un objeto e ir modificando sus atributos para luego poder añadirlo a la lista. Sin embargo, al intentarlo apareció el primer error y, tras indagar en diferentes páginas y dar varias vueltas al código de ejemplo, se llegó a la conclusión de que solamente se puede crear un único objeto y trabajar con él a través de la instancia. Por tanto, para poder añadir y crear una lista de contactos, se debería estar sobreescribiendo todo el rato la misma instancia.

Tras resolver el problema mencionado se procedió a implementar los métodos para trabajar con el fichero properties. La idea era tenerlo actualizado en todo momento y para ello cada vez que se hacía una operación en la que se modificase algo del gestor, se volcaba directamente en el fichero y viceversa, si se añadía algo al fichero cada vez que se iba a operar con el gestor, este estaba actualizado.

Además, se quería implementar la idea de tener el array actualizado directamente en el constructor, es decir, que cada vez que se crease un contacto directamente se pretendía llamar a la función FileToArray() para así poder trabajar con la versión más actualizada del gestor. Sin embargo, al realizar el main y probar la clase, esa idea empezó a dar problemas y se tuvo que llegar a la decisión de que el propio main se encargara de la tarea de empezar con el gestor actualizado.

Otros problemas que se encontraron a la hora de realizar pruebas fueron:

-Al cargar el fichero vacío, solamente con las KEYS que lo caracterizan, el programa dejaba de funcionar. La solución elegida fue inicializar las variables en las que se guardan los datos del fichero para que si no hay nada que leer, funcione correctamente.

-Al cargar el fichero con los campos de fecha vacíos, el programa dejaba de funcionar. Es por ello que se decidió inicializar las fechas a 00/00/0000 para solucionar dicho problema.

-Solo se podrá admitir los intereses siguientes:moda ,fitness, videojuegos, deporte, cine, series,arquitectura,baile,música,deportes.

Por último, para agilizar y facilitar el acceso al fichero properties que simula la base de datos, se decidió localizar a este en la ruta: \P1\contactos.properties.

Ejercicio 2

Sin duda alguna este patrón es el que más difícil ha sido de implementar. La idea básica del patrón en sí se tenía en mente, pero a la hora de implementarlo es cuando han aparecido los problemas. Primero se decidió implementarlo de tal manera que se recogiese todo en un arrayList que guardase objetos anuncios, ya que los demás tipos de anuncios eran, en parte, como los objetos de la clase que heredan. Sin embargo, al llegar y empezar a realizar las funciones donde se tenían que acceder a los métodos específicos, se mostraba el principal problema de nuestra idea, no se podía realizar dichas operaciones.

Tras este primer intento, se intentó realizar de una segunda forma creando por cada clase un arrayList que guardase su respectivo tipo de anuncio y posteriormente, identificar y trabajar con ellos a través de un arraylist de enteros implementado en la clase común y que sirviese de conexión entre todos los arrayList a través del atributo id.

De nuevo, se volvió a topar con otro problema, cómo hacer que ese arrayList pudiese trabajar y comparar con las demás estructuras definidas en los otros tipos de anuncios. No se encontró ninguna solución y se tuvo que volver a pensar acerca de cómo implementar y conectar todo.

¿Qué es lo que se decidió?, se decidió estructurar y configurar la clase abstracta y las clases que heredan de ésta con los métodos justos y necesarios y juntar todas las funciones y variables de nexo en la clase tablón. Ya que se decidió que esta fuese la encargada de mantener los arrays de cada tipo de anuncio así como de las funciones comunes.

Otra idea que se quiso llevar a cabo es que los anuncios estuvieran caracterizados por un entero, de tal forma que cada este entero se inicializase a 0 y cada vez que se llamase al constructor se incrementara en uno. De nuevo apareció otro problema. Cada vez que creabas un objeto, el id se inicialazaba a 0 y se incrementaba en el constructor de tal manera que todos los anuncios que se creasen tendrían por id 1 no pudiendo así ser identificados.

La solución que se implementó fue que el id fuera un entero generado aleatoriamente. El id puede ser un número comprendido entre 0 - 1000, al ser un abanico de posibilidades amplio, la probabilidad de que se repitiese un id es pequeña aunque no imposible.

Una idea de diseño que se llevó a cabo en este ejercicio fue la de solamente definir los destinatarios para los tipos de anuncios individualizados, ya que se asumió que para todo los demás tipos de anuncios, se iban a dirigir a todos aquellos usuarios registrados en el sistema.

Debido a un error que apareció al tratar de publicar un anuncio flash sin haber llegado su fecha inicial, se decidió que, hasta que llegase a la fecha inicial, se estableciera la fase de este en espera.

Cosas a tener en cuenta de la entrega realizada: se entregan dos versiones del programa debido a la incompatibilidad que se mencionó en el foro de la asignatura.

Una de ellas es con la librerías de la versión 8 de java, realizada en casa y totalmente funcional mientras que la otra, es la probada en thingstation con la versión 7 de java y con las librerías necesarias para que haya compatibilidad. Esta última también es funcional pero a la hora de ejecutar sale un aviso que no se ha podido solventar.

Fuentes consultadas:

https://informaticapc.com/patrones-de-diseno/singleton.php
https://refactoring.guru/es/design-patterns/singleton
http://lineadecodigo.com/patrones/patron-abstract-factory/
https://informaticapc.com/patrones-de-diseno/factory-method.php
https://refactoring.guru/es/design-patterns/abstract-factory