Análisis y Diseño de Software

Práctica 5



Tabla de contenido

[Apartado 1. Modelo de datos 3](#_Toc5551603)

[Apartado 2. Recomendador de vecinos 3](#_Toc5551604)

[Apartado 3. Otros recomendadores 3](#_Toc5551605)

[Apartado 4. Evaluar recomendaciones 3](#_Toc5551606)

[Apartado 5. Comprobar que todo funciona correctamente 3](#_Toc5551607)

[Diagrama de clases 4](#_Toc5551608)

[Nota 4](#_Toc5551609)

# Apartado 1. Genericidad y Colecciones

En este primer apartado, hemos realizado la implementación de las dos interfaces que nos daban. La primera decisión de diseño importante que hemos llevado a cabo es usar una lista enlazada para almacenar los elementos de la matriz, utilizándola como una matriz dispersa. Evidentemente de esta manera evitamos que en matrices grandes las búsquedas sean muy largas y que se reduzca al mínimo el tamaño de la matriz, dado que no almacenamos los elementos vacíos. Además, en el elemento almacenamos su posición para saber dónde se encuentra. Hemos implementado *MatrixElement* de manera que sea genérico, y pueda almacenar cualquier objeto en su interior, en este caso serán celdas, en el siguiente apartado.

# Apartado 2. Comparadores

En el segundo apartado, hemos implementado un *Comparator* de *IMatrixElement*, que vamos a usar en un nuevo método (asListSortedBy), que se encarga de ordenar nuestra lista de la matriz, y retornar la misma. En nuestro método compare hemos ordenado primero por columnas, y si se encontraban en la misma por filas.

# Apartado 3. Simulación básica

En este ejercicio, hemos comenzado a usar Simuladores, usando en este, *BasicSimulator*, el más básico de los que vamos a usar y que tan solo muestra el número de elementos que hay en cada *IMatrixElement*, pero permanecerán impasibles dado que aún no tenemos ningún comportamiento que pueda modificarlos. También hemos creado una nueva clase *BasicAgent*, que implementa la interfaz que nos dan y una nueva clase, Cell; que básicamente contendrá una lista de agentes. Para hacer las copias, en un principio usamos clone, pero definitivamente decidimos sencillamente crear un nuevo *BasicAgent*.

# Apartado 4. Lambdas e Interfaces Funcionales

En el cuarto apartado hemos desarrollado los agentes, añadiéndoles comportamiento. Ahora tenemos una nueva interfaz (*IAgent*) la cual extiende de *IBasicAgent* de manera, que la clase que implementa la primera (*Agent*). Por comodidad también hemos creado la clase Comportamiento para almacenar la tupla de comportamiento y su condición necesaria para ocurrir. Si a un comportamiento no le damos condición, esta será siempre verdadera.

A su vez, como íbamos a implementar un segundo simulador *Simulator*, para mantener una estructura más clara, como se puede ver en el diagrama de clases, hemos creado la clase abstracta *GeneralSimulator*, de la que ahora extienden *Simulator y BasicSimulator*, que almacena la matriz sobre la cual hacemos la simulación, el tiempo de la simulación y declara el método abstracto *run (int pasos).*

# Apartado 5. Patrones de diseño

En este último apartado hemos creado estados para los agentes implementado *IAgentState* y agentes con estados *IAgentWithState*, mediante las clases *AgentState* y *AgentWithState* respectivamente.

Los estados como los agentes y los agentes con estado guardan una lista de *Comportamiento* con el comportamiento y su condición y para los cambios de estado, en toState(…), introducimos el target y su trigger en un map <String, Predicate> que a la hora de hacer changeState() comprueba y hace el cambia su owner (*AgentWithState*, que tiene dicho estado) a ese estado si se cumple el trigger.

Por último, *AgentWithState* tiene una lista de todos sus posibles estado, que son añadidos en su constructor, por lo que tras crearlo no se le pueden crear más estados y almacena el estado actual en el que está. Para que se ejecutase correctamente, hemos sobrescrito el método exec() de *Agent* para que ejecutase primero todos los comportamientos del agente y tras eso ejecutara los comportamientos del estado actual.

# Diagrama de clases

El .png de la imagen se encuentra en el .zip si se quiere ver más grande y a mejor calidad.

# Nota

Documentación en ./doc, código fuente en ./src y ficheros utilizados en Moodle.

Compilar y ejecutar desde el directorio ./src