Informe de Proyecto Dictionary 2020

SOLID

SRP (Principio de responsabilidad única):

-La clase mainWindow tiene más de una responsabilidad, más de una razón de cambio. Se encarga de procesar y gestionar los eventos de entrada, de conectarse a Wikipedia para buscar el texto deseado, de transformar el texto en html para poder guardarlo en la base de datos, y además tampoco se trabaja en el mismo nivel de abstracción.

Por lo tanto, si se desea hacer algún cambio habrá que modificar la clase, esto está mal, para solucionarlo delegamos las responsabilidades a otras clases.

OCP (Principio de abierto cerrado):

Si se quiere agregar una funcionalidad nueva como las que se piden en el enunciado, se debe modificar las clases, lo cual está mal, solo se debería poder extender sin tener que modificarlas.

DIP (Inversión de dependencias):

-Hay clases concretas que tienen dependencia directa. Las clases de más alto nivel no deberían depender de los detalles.

Similarmente al Open Closed, se deberían agregar interfaces para eliminar estas dependencias directas.

Por lo tanto, para solucionarlo se agregan interfaces para cumplir con DIP.

SMELL CODES:

- Código en el nivel de abstracción incorrecto: ocurre en clase mainWindow
- Las funciones y clases deben hacer una sola cosa: en clase mainWindow, y en database
- Hay comentarios innecesarios en las clases y funciones.
- No se usan nombres descriptivos-> nombres de clases y de variables no representativos

MVC

-Uso de la vista:

- En el main se renderiza el modelo.
- Se solicitan actualizaciones a los modelos con SearchItemViewActivity en el método updateText(String definition)
- Se envían los gestos del usuario al controlador con SearchItemViewActivity en el metodo initListeners()
- Permite al controlador seleccionar la vista con setSearchItemView(SearchItemView searchItemView) en la clase SearchItemControllerImp

-Uso del modelo:

- Encapsula el estado de la aplicación mediante Repository.
- Responder a consultas de estado mediante updateTerm(String name)
- Notifica a la vista de cambios.

-Uso del controlador:

- Mapea acciones de usuario a actualizaciones de modelo, mediante onEventSearch(String term).
- Hay un controlador por cada funcionalidad (un solo controlador searchItem, buscar ítem)
- Seleccionar vistas para las respuestas, mediante setSearchItemView

MEJORAS REALIZADAS:

-Reportar al usuario por pantalla que no se encontró la palabra que buscaba cuando Wikipedia devuelve una entrada vacía y además no debe guardarse en la cache.

Si al ingresar una palabra, Wikipedia no encuentra ninguna definición correspondiente, a la definición se le asigna el String igual a “No Results” y luego en la clase SearchItemViewActivity al detectar esto muestra un cartel por pantalla que no se encontraron resultados correspondientes al termino que se buscó. Además de mostrar el cartel tampoco se modifica la cache y se actualiza el panel para que quede en blanco sin mostrar ninguna definición.

-Considerar entradas temporales en la cache: cuando se recupera de la cache una entrada que tiene más de un día de antigüedad, debe ser descartada y debe utilizarse la funcionalidad básica.

Para realizar esta nueva funcionalidad, en primer lugar, agregue un nuevo atributo a la tabla llamado fecha del tipo datetime, entonces cuando se busca una palabra en Wikipedia y se guarda en la cache, ya la guardo con la fecha actual de ese momento. Luego, para detectar si la entrada buscada tiene más de un día de antigüedad selecciono la entrada de la cache que la fecha sea mayor a la fecha actual menos un día y la elimino de la cache. Por último, se busca la palabra en Wikipedia que fue eliminada anteriormente y la agrego a la cache nuevamente.

TESTING CON JUNIT

Para realizar el testeo utilizo testing unitario con Junit, para esto se creó la clase DictionaryTest y el método test() con la anotación @test.

En primer lugar, seteo el ambiente adecuado para la unidad creando los objetos correspondientes para realizar los tests. Luego, proveo los datos de entrada y hago la invocación de la unidad y por último hago la verificación usando los assertEquals y assertNotEquals.

Test realizados:

- Buscar una palabra en Wikipedia. Para este caso busco la palabra “audi” la cual debe buscar su definición y guardarla en la cache. Por lo tanto, como el resultado de la definición será distinto de null el testeo es correcto.
- No se encuentra la palabra buscada. Para este caso intento buscar "estaPalabraNoExiste" la cual como no existe su definición, no la guarda en la cache y al hacer el test como la definición es “No Results” este termina exitosamente.
- Buscar una entrada con más de un día de antigüedad. Para testear este caso hice el método testAntiguedad() en la clase DataBase y luego la clase test() en Repository, en este método ingreso una entrada manualmente con una fecha mayor a un día de antigüedad para poder realizar el test necesario. Ingreso la entrada "entrada con más de un día de antiguedad" con la fecha '2020-05-01 10:10:10’, entonces al buscar esa entrada detecta que tiene más de un día de antigüedad, la elimina de la cache y la busca nuevamente. Como esa entrada no existe en Wikipedia retorna que no existe esa definición y el test termina satisfactoriamente.
- Buscar una entrada con menos de un dia de antigüedad. Para testear este caso, en el método testAntiguedad() en la clase DataBase ingrese la entrada: 'entrada con menos de 1 dia de antiguedad' con definición: 'definicion de entrada con menos de 1 dia de antiguedad' y la fecha actual al ejecutar el método test() de la clase DictionaryTest. Por lo tanto, al hacer el test compruebo que la definición es 'definicion de entrada con menos de 1 dia de antiguedad' y el test termina exitosamente.