Observaciones_lab3

Grupo 08

1. ¿Cuáles son los mecanismos de interacción (Input / Output) que tiene el **view.py** con el usuario?

Los mecanismos de interacción que tiene el **view.py** con el usuario son los siguientes. En primer lugar, el programa despliega un menú que muestra las opciones existentes. Este presenta un mecanismo de input que le solicita al usuario seleccionar una de las anteriores opciones. En segundo lugar, los mecanismos de output se pueden dividir entre los outputs dirigidos al *controller* y los outputs dirigidos nuevamente al usuario. De hecho, ambos están relacionados en la medida en que los comandos dirigidos al *controller*, y en último lugar al *model*, retornan como el output dirigido al usuario. Por ejemplo, la función printBestBooks() es un output de tipo *string* dirigido al usuario, cuyo parámetro *books* se obtiene en la función del *model* getBestBooks() y se comunica mediante la función del *controller* getBestBooks().

2. ¿Cómo se almacena los datos de **GoodReads** en el **model.py**?

Primero, el *model* tiene la función newCatalog() que retorna un diccionario *catalog*. Cada llave de este diccionario tiene asociada una estructura de datos, la cual es implementada mediante la función newList del TAD Lista. Esta función, por defecto, implementa la estructura de datos SINGLE_LINKED y, por tanto, si se desea implementar otra estructura de datos como el ARRAY_LIST se debe especificar en el parámetro de esta función. Entonces, en este diccionario, cada llave tiene asociada una estructura de datos vacía, que puede ser un arreglo o una lista encadenada. Segundo, el programa lee los archivos csv en el *controller*, mediante funciones como loadBooks() o loadTags(). Estas funciones invocan funciones del *model*, tal como addBooks() o addTags(). Por último, estas últimas funciones almacenan la información en las estructuras de datos, vacías hasta entonces, del diccionario *catalog*.

3. ¿Cuáles son las funciones que comunican el **view.py** y el **model.py**?

Las funciones que comunican el *view* y el *model* están ubicadas en el c*ontroller*, y son las siguientes. La función initCatalog() comunica el *view* con el *model* para crear el diccionario *catalog*. Asimismo, la función getBestBooks() comunica a la función del *view* printBestBooks() con la función del *model* que retorna los mejores libros. Además, la función getBooksByAuthor() comunica a la función del *view* printBooksByAuthor () con la función del *model* que retorna un autor con sus libros. Por último, el valor de la variable book_count es la función countBooksByTag() del *controller*, que en últimas es el retorno de la función countBooksByTag() del *model*.

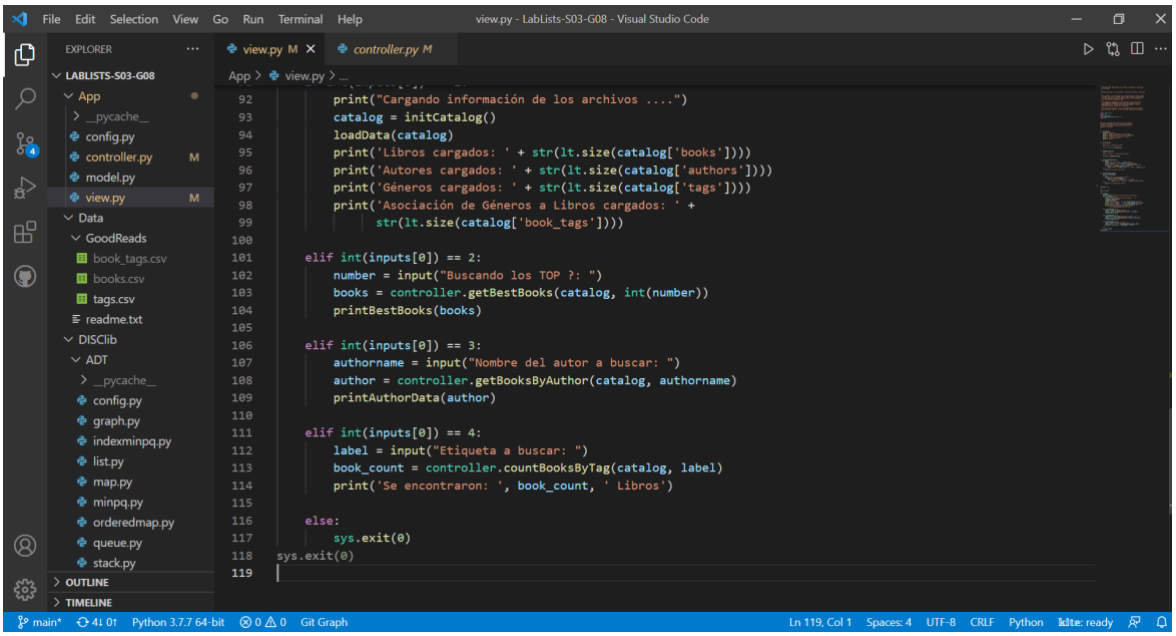

Imagen.

La anterior imagen muestra que en las líneas 103 y 108 se define una variable a partir de la función del *controller*. Esta variable es después utilizada como parámetro de la función del *view*. En el caso de la línea 113, el valor de la variable es una función del *controller*, pero esta no es utilizada como parámetro posteriormente, sino que directamente se imprime.

4. ¿Cómo se crea una lista?

Una lista se crea utilizando la función newList del TAD lista. Esta función permite escoger la estructura de datos para la lista, que puede ser una lista encadenada simple (SINGLE_LINKED) o un arreglo (ARRAY_LIST), y la forma en la que se comparan los elementos de la lista. Estas elecciones se hacen a través de los argumentos. Por defecto, una lista se crea con la estructura de lista encadenada simple. Hay dos argumentos que controlan la comparación de elementos de la lista. Uno permite especificar la función de comparación y otro permite usar identificadores. No se pueden usar ambos al tiempo. La función que crea una lista (como tipo abstracto de datos) lo hace llamando una función que selecciona la estructura de datos y esta función a su vez llama a las funciones de creación de las estructuras de datos ‘single-linked’ o ‘array list’, de acuerdo a la elección del programador.

5. ¿Qué hace el parámetro **cmpfunction=None** en la función **newList( )**?

Este parámetro es una función que permite comparar los elementos de la lista creada. Si este parámetro se deja por defecto, se usa una función por defecto que depende de la estructura de datos de la lista (single-linked o array-list).

6. ¿Qué hace la función **addLast( )**?

Esta función tiene dos parámetros: uno es una lista y el otro es un elemento. Esta función agrega el elemento pasado por parámetro a la lista en la última posición de esta.

7. ¿Qué hace la función **getElement( )**?

Esta función recibe dos parámetros: una lista y un entero. La función retorna el elemento en la posición de la lista marcada por el entero dado por parámetro.

8. ¿Qué hace la función **subList( )**?

Esta función recibe tres parámetros: una lista y dos enteros. La función retorna una lista nueva cuyo primer elemento es aquel en la posición del primer entero de la lista pasada por parámetro. El tamaño de la lista está dado por el segundo entero y los elementos restantes son los subsiguientes al elemento en la posición escogida en la lista pasada por parámetro.

9. ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro **ARRAY_LIST** a **SINGLE_LINKED**?

Creemos que con este cambio el programa tardó menos tiempo en cargar los datos. Esto se puede deber a que las listas encadenadas utilizan de forma más eficiente la memoria principal que los arreglos y, dado que utilizamos el archivo csv que contiene todos los datos, esta diferencia entre estructuras de datos marca una diferencia en el tiempo de ejecución. Por otro lado, el programa se ejecutó igual que antes. Esto se debe a dos razones, el programa está divido entre *view*, *controller* y *model*, y el programa implementa el TAD Lista. Primero, si el programa está divido entre estos tres archivos, un cambio en el *model* no implica un cambio en el *view*, ya que estos dos están mediados por el *controller*. Segundo, si se usa un Tipo Abstracto de Dato como la Lista, se puede cambiar la estructura de dato sin que eso afecte la solución al problema.