OBSERVACIONES DE LA PRACTICA

Adriana Sofía Rozo Cepeda 1 Cod 202211498
Diego Fernando Galván Cruz 2 Cod 202213709
Nicolás Bedoya Figueroa 3 Cod 202212100

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

En view.py encontramos la función PrintMenu (Imagen 1) que muestra al usuario el menú disponible. Esta función es ejecutada en el ciclo encontrado al final de view.py (Imagen 2) siempre y cuando se cumpla la condición de avanzar (while True), a su vez, se pide al usuario un Input para seleccionar la opción para ejecutar. Como Output (Imagen 3) el usuario recibe la información de la opción solicitada. Por ejemplo, la ejecución de la opción dos da como resultado (Output) el top de libros por promedio dado un número (Input).

```
def printMenu():
    """
    Menu de usuario
    """
    print("Bienvenido")
    print("1- Cargar información en el catálogo")
    print("2- Consultar los Top x libros por promedio")
    print("3- Consultar los libros de un autor")
    print("4- Libros por género")
    print("0- Salir")
```

Imagen 1. Función que imprime el menú de usuario.

```
while True:
    printMenu()
    inputs = input('Seleccione una opción para continuar\n')
    if int(inputs[0]) == 1:
        print("Cargando información de los archivos ....")
        bk, at, tg, bktg = loadData(control)
        print('Libros cargados: ' + str(bk))
        print('Autores cargados: ' + str(at))
        print('Géneros cargados: ' + str(tg))
        print('Asociación de Géneros a Libros cargados: ' +
              str(bktg))

    elif int(inputs[0]) == 2:
        number = input("Buscando los TOP ?: ")
        books = controller.getBestBooks(control, int(number))
        printBestBooks(books)

    elif int(inputs[0]) == 3:
        authorname = input("Nombre del autor a buscar: ")
        author = controller.getBooksByAuthor(control, authorname)
        printAuthorData(author)

    elif int(inputs[0]) == 4:
        label = input("Etiqueta a buscar: ")
        book_count = controller.countBooksByTag(control, label)
        print('Se encontraron: ', book_count, ' Libros')

    elif int(inputs[0]) == 0:
        sys.exit(0)

    else:
        continue
sys.exit(0)
```

Imagen 2. Ciclo que ejecuta la opción solicitada.

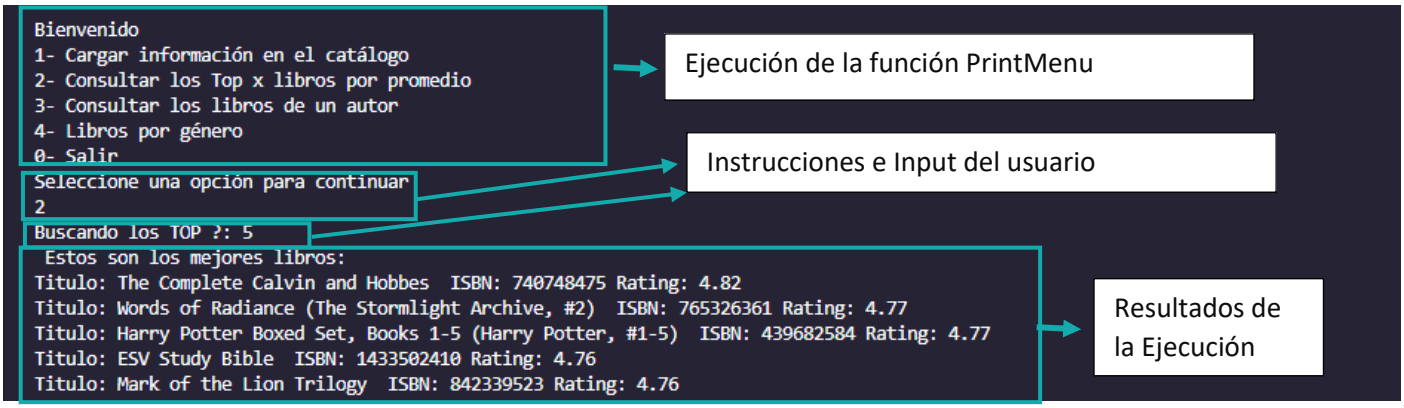

Imagen 3. Ejemplo de ejecución de view.py.

2) ¿Cómo se almacenan los datos de GoodReads en el model.py?

Los datos se almacenan en un diccionario de listas. El diccionario se llama catalog. Las llaves de este diccionario contienen de valor listas vacías. Gracias a las funciones que encontramos en el model.py estas van obteniendo sus elementos.

```
42  def newCatalog():
43      """
44      Inicializa el catálogo de libros. Crea una lista vacia para guardar
45      todos los libros, adicionalmente, crea una lista vacia para los autores,
46      una lista vacia para los generos y una lista vacia para la asociación
47      generos y libros. Retorna el catalogo inicializado.
48      """
49      catalog = {'books': None,
50                 'authors': None,
51                 'tags': None,
52                 'book_tags': None}
53
54      catalog['books'] = lt.newList('ARRAY_LIST')
55      catalog['authors'] = lt.newList('SINGLE_LINKED',
56                                      cmpfunction=compareauthors)
57      catalog['tags'] = lt.newList('SINGLE_LINKED',
58                                   cmpfunction=comparetagnames)
59      catalog['book_tags'] = lt.newList('ARRAY_LIST')
60
61      return catalog
62
```

Diccionario catalog con llaves y valor

1.La llave ‘books’ tiene asociado un valor que es un TAD que acopla una Array List.

2. La llave ‘authors’ tiene asociado un valor que es un TAD que acopla una Linked List.

3. La llave ‘tags’ tiene asociado un valor que es un TAD que acopla una Linked List.

4. La llave ‘book_tags’ tiene asociado un valor que es un TAD que acopla un Array List.

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

No hay funciones que conecten el view.py y el model.py directamente, sin embargo, el archivo controller.py conecta ambos archivos con funciones como:

```python
# Funciones de consulta sobre el catálogo

def getBooksByAuthor(control, authorname):
    """
    Retrona los libros de un autor
    """
    author = model.getBooksByAuthor(control['model'], authorname)
    return author


def getBestBooks(control, number):
    """
    Retorna los mejores libros
    """
    bestbooks = model.getBestBooks(control['model'], number)
    return bestbooks


def countBooksByTag(control, tag):
    """
    Retorna los libros que fueron etiquetados con el tag
    """
    return model.countBooksByTag(control['model'], tag)

```

Imagen 4. Funciones en controller.py que conectan view.py y el model.py.

4) ¿Cuál es la función que permite crear una lista?, ¿Qué datos son necesarios?
La función se llama newList. Se necesita especificar obligatoriamente el Datastructure que puede ser: ARRAY_LIST, SINGLE_LINKED, DOUBLE_LINKED. Esta función también puede recibir otros parámetros como: **cpmfunction**: Si se usara función de comparación**, key**: comparación (None) por defecto**, filename**: Archivo CSV UTF8 **o delimiter:** Separación de campos, por defecto es (,).

```python
def newList(datastructure='SINGLE_LINKED',
            cmpfunction=None,
            key=None,
            filename=None,
            delimiter=","):
    """Crea una lista vacia

    Args:
        datastructure:  Tipo de estructura de datos a utilizar para implementar
        la lista. Los tipos posibles pueden ser: ARRAY_LIST,
        SINGLE_LINKED y DOUBLE_LINKED.

        cmpfunction: Función de comparación para los elementos de la lista.
        Si no se provee función de comparación se utiliza la función
        por defecto pero se debe proveer un valor para key.
        Si se provee una función de comparación el valor de Key debe ser None.

        Key:  Identificador utilizado para comparar dos elementos de la lista
        con la función de comaparación por defecto.

        filename: Si se provee este valor, se crea una lista a partir
        de los elementos encontrados en el archivo.
        Se espera que sea un archivo CSV UTF8.

        delimiter: Si se pasa un archivo en el parámetro filename, se utiliza
        este valor para separar los campos. El valor por defecto es una coma.

    Returns:
        Una nueva lista
    Raises:
        Exception
    """
    try:
        module = listSelector(datastructure)
```

```python
    try:
        module = listSelector(datastructure)
        lst = module.newList(
            cmpfunction,
            module,
            key,
            filename,
            delimiter
        )
        return lst
    except Exception as exp:
        error.reraise(exp, 'TADList->newList: ')
```

5) **¿Para qué sirve el parámetro `datastructure` en la función `newList()`?, ¿Cuáles son los posibles valores para este parámetro?**

   Sirve para determinar la estructura de datos de la newList y los posibles tipos son: ARRAY_LIST, SINGLE_LINKED y DOUBLE_LINKED.

6) **¿Para qué sirve el parámetro `cmpfunction` en la función `newList()`?**
   Este parámetro sirve para determinar qué función se utilizará para comparar los datos de la lista. Si no se provee función de comparación se utiliza la función por defecto, pero se debe proveer un valor para key. Si se provee una función de comparación el valor de key debe ser None.

7) ¿Qué hace la función addLast()?
   La función addLast adiciona un elemento a la lista en la última posición, actualiza el apuntador a esta última posición. Es similar a la lógica del método. append() de phyton.

```
def addLast(lst, element):
    """ Agrega un elemento en la última posición de la lista.

    Se adiciona un elemento en la última posición de la lista y se actualiza
    el apuntador a la útima posición. Se incrementa el tamaño de la lista en 1

    Args:
        lst: La lista en la que se inserta el elemento
        element: El elemento a insertar

    Raises:
        Exception
    """
    try:
        lst['datastructure'].addLast(lst, element)
    except Exception as exp:
        error.reraise(exp, 'TADList->addLast: ')
```

8) ¿Qué hace la función **`getElement()`**?

La función getElement recorre la lista (dada por parámetro) hasta el elemento que esta en la posición (pos) dada por parámetro. Dicha posición debe ser mayor que cero y menor o igual al tamaño de la lista. Se retorna el elemento en dicha posición sin eliminarlo. La lista no puede estar vacía.

El funcionamiento es similar a lista[posición] y contrario a el método .pop(posición) de phyton que sí elimina el elemento dada la posición pero devuelve el elemento eliminado.

9) ¿Qué hace la función **`subList()`**?
Crea una copia de la lista inicial y retorna una nueva lista empezando en un elemento con posición n de la lista original y agregando m elementos, todo lo anterior lo recibe en sus parámetros.

```
354    def subList(lst, pos, numelem):
355        """ Retorna una sublista de la lista lst.
356
357        Se retorna una lista que contiene los elementos a partir de la
358        posicion pos, con una longitud de numelem elementos.
359        Se crea una copia de dichos elementos y se retorna una lista nueva.
360
361        Args:
362            lst: La lista a examinar
363            pos: Posición a partir de la que se desea obtener la sublista
364            numelem: Numero de elementos a copiar en la sublista
365
366        Raises:
367            Exception
368        """
369        try:
370            return lst['datastructure'].subList(lst, pos, numelem)
371        except Exception as exp:
372            error.reraise(exp, 'List->subList: ')
373
```

10) Revise el uso de la función **`iterator()`** en las funciones **`printAuthorData(author)`** y **`printBestBooks(books)`** en la Vista que aplican a una lista de libros. ¿Qué hace la función **`iterator()`**?
La función iterator es un iterador en una lista que permite “recorrerla”. En la función printAuthorData dan como parámetro author el cual entra a books en donde se iterará hasta encontrar los libros cuyo autor sea el ingresado por parámetro. Al final con esta información la función imprime únicamente el título, y el ISBN de todos los libros del autor ingresado por parámetro.

```
def printAuthorData(author):
    """
    Recorre la lista de libros de un autor, imprimiendo
    la informacin solicitada.
    """
    if author:
        print('Autor encontrado: ' + author['name'])
        print('Promedio: ' + str(author['average_rating']))
        print('Total de libros: ' + str(lt.size(author['books'])))
        for book in lt.iterator(author['books']):
            print('Titulo: ' + book['title'] + '  ISBN: ' + book['isbn'])
    else:
        print('No se encontro el autor')
```

11) ¿Observó algún cambio en el comportamiento del programa al cambiar el valor del parámetro 'datastructure' en la creación de las listas?

En contraste con la configuración de datastructure inicial, el comportamiento cambió en cuanto a tiempo de ejecución (realmente se demoró mucho más). Los outputs del view.py no sufrieron cambios por lo que la experiencia con el usuario no se vio afectada en cuanto a la información.