OBSERVACIONES DEL LA PRACTICA

Mateo Oviedo Reyes Cod 202028312
Angie Catalina Campos Perilla Cod 202115094

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

El view, es el modulo de nuestro código que se encarga de interactuar directamente con el usuario. Para esto, se utilizo un sistema de interacción que muestra al usuario la información utilizando las funciones print e input, las cuales permiten imprimir la información que se desea en la consola, y recibir información del usuario que se almacena como una cadena de caracteres en una variable.

La primera función que hace uso de la consola es la función “print menu”:

```
36
37
38  def printMenu():
39      print("Bienvenido")
40      print("1- Cargar información en el catálogo")
41      print("2- Consultar los Top x libros por promedio")
42      print("3- Consultar los libros de un autor")
43      print("4- Libros por género")
44      print("0- Salir")
45
```

Esta función le muestra al usuario las funcionalidades del programa, y que numero representa cada una, utilizando la función print().

la función se ejecuta de primeras cada vez que corremos el programa.

```
"""
Menu principal
"""
while True:
    printMenu()
    inputs = input('Seleccione una opción para continuar\n')
    if int(inputs[0]) == 1:
        print("Cargando información de los archivos ....")
        catalog = initCatalog()
        loadData(catalog)
        print('Libros cargados: ' + str(lt.size(catalog['books'])))
        print('Autores cargados: ' + str(lt.size(catalog['authors'])))
        print('Géneros cargados: ' + str(lt.size(catalog['tags'])))
        print('Asociación de Géneros a Libros cargados: ' +
              str(lt.size(catalog['book_tags'])))

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

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

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

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

Después se le pregunta al usuario que opción desea seleccionar, utilizando un input y al ejecutarlo luce así:

```
Bienvenido
1- Cargar información en el catálogo
2- Consultar los Top x libros por promedio
3- Consultar los libros de un autor
4- Libros por género
0- Salir
Seleccione una opción para continuar
```

Utilizando estas funciones el programa tiene una comunicación directa con el usuario para intercambiar información.

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

Los datos del archivo “GoodReads” son almacenados en el modulo “model.py”, en un Tipo de Dato Abstracto (TDA) haciendo uso de la librería “DISClib.ADT”, en particular se utiliza un tipo de dato abstracto lista. Para los datos de los libros “Se define la estructura de un catálogo de libros. El catálogo tendrá tres listas, una para libros, otra para autores y otra para géneros”. Esto se logra con la función “newCatalog()”:

```
def newCatalog():
    """
    Inicializa el catálogo de libros. Crea una lista vacia para guardar
    todos los libros, adicionalmente, crea una lista vacia para los autores,
    una lista vacia para los generos y una lista vacia para la asociación
    generos y libros. Retorna el catalogo inicializado.
    """
    catalog = {'books': None,
               'authors': None,
               'tags': None,
               'book_tags': None}

    catalog['books'] = lt.newList()
    catalog['authors'] = lt.newList('ARRAY_LIST',
                                    cmpfunction=compareauthors)
    catalog['tags'] = lt.newList('ARRAY_LIST',
                                 cmpfunction=comparetagnames)
    catalog['book_tags'] = lt.newList('ARRAY_LIST')

    return catalog
```

En esta función, se crea una variable que contiene un diccionario que contiene listas TDA, para cada información de los libros, en particular el tipo de dato utilizado por estas listas es una lista de arreglo.

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

Las funciones del modulo “Controller.py” comunican el modulo “view.py” con el modulo “model.py”, cada una de ellas esta asociada con una función en el view, y lo que hace es hacer uso de las funciones en el model para obtener la informacion que el usuario solicito.

4) ¿Cómo se crea una lista?

Una lista se define como una colección de elementos; para la creación de la misma se puede realizar de dos maneras:

a. Se puede crear una variable con una lista vacía (o directamente se pueden agregar los elementos)

```
1    list = []
```

b. Se puede crear una lista vacía con una función (como se puede evidenciar en el archivo del Laboratorio 3 → DISClib → ADT → list.py)

```
1   def newList(datastructure='SINGLE_LINKED',
2               cmpfunction=None,
3               key=None,
4               filename=None,
5               delimiter=","):
6       """Crea una lista vacia
7       """
8       try:
9           lst = lt.newList(datastructure, cmpfunction, key, filename, delimiter)
10          return lst
11      except Exception as exp:
12          error.reraise(exp, 'TADList->newList: ')
```

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

El parámetro ***cmpfunction*** es una función que compara los elementos de la lista. El valor de la Key debe ser None siempre y cuando se provee esta funcion, en caso contrario, se pone un valor por defecto (como se puede evidenciar en el archivo del Laboratorio 3 → DISClib → ADT → list.py).

```
40   def newList(datastructure='SINGLE_LINKED',
41               cmpfunction=None,
42               key=None,
43               filename=None,
44               delimiter=","):
```

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

Esta función aparte de agregar un elemento a la última posición de la lista, también incrementa el tamaño de la misma y actualiza el apuntador a la última posición (como se puede evidenciar en el archivo del Laboratorio 3 → DISClib → ADT → list.py).

```
def addLast(lst, element):
    """ Agrega un elemento en la última posición de la lista.
    """
    try:
        lt.addLast(lst, element)
    except Exception as exp:
        error.reraise(exp, 'TADList->addLast: ')
```

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

Recorre toda la lista en busca del elemento (este debe ser mayor a cero por lo que la posición 0 no existe; además, debe ser menor o igual al tamaño de la lista dado que si es mayor no estaría en la lista), finalmente retorna el elemento en dicha posición sin eliminarlo (como se puede evidenciar en el archivo del Laboratorio 3 → DISClib → ADT → list.py).

```
def getElement(lst, pos):
    """ Retorna el elemento en la posición pos de la lista.
    """
    try:
        return lt.getElement(lst, pos)
    except Exception as exp:
        error.reraise(exp, 'List->getElement: ')
```

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

Recorre la lista en busca de la posición, después crea copia de los elementos a partir de la posición y con la longitud del número de elementos a copiar, por último, retorna una nueva lista (como se puede evidenciar en el archivo del Laboratorio 3 → DISClib → ADT → list.py).

```
def subList(lst, pos, numelem):
    """ Retorna una sublista de la lista lst.
    """
    try:
        return lt.subList(lst, pos, numelem)
    except Exception as exp:
        error.reraise(exp, 'List->subList: ')
```

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

Se evidenció un cambio en la estructura de datos, ***"ARRAY_LIST"*** es una lista basada en arreglo, en cuanto a ***"SINGLE_LINKED"*** es una lista encadenada. Por lo tanto, el comportamiento del comportamiento cambia considerablemente aunque no se ve a simple vista, ya que no se ejecuta de la misma manera. Además, notamos que el ***"ARRAY_LIST"*** se demoraba menos que el ***"SINGLE_LINKED"***, dado que el ***"SINGLE_LINKED"*** guardar los datos en un contenedor, también tiene que generar el nodo para saber en donde se encuentra el siguiente dato, en cambio en el ***"ARRAY_LIST"*** no tiene problema, puesto que cada dato ya tiene una posición predispuesta.