OBSERVACIONES DEL LA PRACTICA

Estudiante 1:Juan Felipe García 202014961 jf.garciam1

Estudiante 2: Santiago Rodríguez 202020476 s.rodriguez64

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

El primer mecanismo es el menú, teniendo 5 opciones, cada una de estas activa una función dentro del programa. Dependiendo del input que haga el usuario se va a ejecutar un resultado distinto, las funciones que se tienen son: 1.Cargar información del catalogo, 2.Busca el top de libros dependiendo de los que quiera incluir el usuario, 3. Busca los libros de un autor especifico que el usuario elija, 4. Busca libros con un genero dependiendo del usuario, 0. Salir del programa. Los outputs dependen la opción escogida respondiendo lo que se pregunto con un mensaje y la respuesta.

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)

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

Los datos de GoodReads se almacenan en el model a través de la creación de listas que se guardan en un catálogo. La función de catálogos crea listas vacías (se especifica si es array o single linked) para los libros, autores, tags y booktags. A través de estas listas se permite guardar nueva información de los documentos csv. de GoodReads. También existen funciones para agregar información a las listas del catálogo, crear datos, consultarlos y comparar elementos en el model.py.

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('SINGLE\_LINKED',

                                 cmpfunction=comparetagnames)

   catalog['book\_tags'] = lt.newList('SINGLE\_LINKED')

    return catalog

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

No hay comunicación directa entre el view.py y el model.py, sino que se hace a través del intermediario el controler.py. Esto se evidencia a través de los import que se hacen en view.py y en el controller.py del controller y el model respectivamente. Al ver esto se puede notar que el controller usa las funciones del model para de este modo organizar y darle la información al view que a su vez se la devuelve al usuario, por lo que las funciones que conectan al view y al model son las que están en el controller.

Por ejemplo, el siguiente código está en el view y llama al controler

def initCatalog():

    """

    Inicializa el catalogo de libros

    """

    return controller.initCatalog()

luego el controller invoca al model en el siguiente código

def initCatalog():

    """

    Llama la funcion de inicializacion del catalogo del modelo.

    """

    catalog = model.newCatalog()

    return catalog

este codigo invoca al código en el model que se encuentra en la pregunta 2.

1. ¿Cómo se crea una lista?

La lista se crea usando las funciones que están ubicadas en la librería DISClib. Específicamente en el archivo list.py de la carpeta ADT. Se crea usando la función newList, donde se especifica el tipo de estructura (Single linked o Array list) y se entrega una función para comparar los elementos de la lista. A esto se le suman los parametros key, filename y delimiter. Con estos parámetros se llama a la función newList ubicada en liststructure.py que crea la lista.

def newList(datastructure='SINGLE\_LINKED',

            cmpfunction=None,

            key=None,

            filename=None,

            delimiter=","):

    try:

        lst = lt.newList(datastructure, cmpfunction, key, filename, delimiter)

       return lst

    except Exception as exp:

        error.reraise(exp, 'TADList->newList: ')

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

Determina que función de comparación es la que se va a usar en la lista, es decir cuál es la función mediante la que se van a comparar los elementos de la lista. Si no se entrega ningún valor al crear la lista se va a tomar el valor None por default y este va a poner a la función de comparación por defecto.

        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.

1. ¿Qué hace la funció **addLast()**?

Lo que hace la función addLast es añadir un elemento al final de la lista, al hacer esto se actualiza el apuntador a la última posición. Aparte de esto el tamaño de la lista se aumenta el 1, para llevar a cabo esta función es necesario dar la lista y el elemento a incorporar.

def addLast(lst, element):

        lt.addLast(lst, element)

    except Exception as exp:

        error.reraise(exp, 'TADList->addLast: ')

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

Esta función recibe una lista y un valor pos. Luego la función recorre la lista hasta el elemento pos y retorna el elemento que está en esta posición sin eliminarlo. Para que no haya un error el valor pos debe ser un entero positivo menor a la longitud de la lista.

def getElement(lst, pos):

    try:

        return lt.getElement(lst, pos)

    except Exception as exp:

        error.reraise(exp, 'List->getElement: ')

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

La función subList crea una nueva lista a partir de la lista original. Se ingresa una posición y el número de elementos que se quieran copiar. Desde la posición se revisa la lista dada y se copian los elementos de esta (la cantidad que pidió el usuario) incorporándolos en una nueva lista que se retorna.

def subList(lst, pos, numelem):

    try:

        return lt.subList(lst, pos, numelem)

    except Exception as exp:

        error.reraise(exp, 'List->subList: ')

1. ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro **“ARRAY\_LIST”** a **“SINGLE\_LINKED”**?

En el caso del singled linked se demoró tres minutos menos que el array list en el mismo computador por lo que se puede decir que en este programa el singled linked es más eficaz que el array list.