Estructuras de Datos

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Primer Cuatrimestre de 2019

Proyecto N° 1
Proyecto de programación en Java

1. Propósito

El presente proyecto tiene por propósito la implementación de diferentes tipos de datos abstractos (TDA) en Java, en función de interfaces previamente definidas y considerando diferentes técnicas y estructuras oportunamente indicadas. Luego, con los TDAs implementados, se espera el desarrollo de una aplicación que permita un conjunto de operaciones por sobre un manejador de archivos virtual. Para la aplicación, se espera la programación tanto de una componente lógica (que implementa toda la funcionalidad), como de una componente gráfica (interfaz) a través de la cual el usuario podrá realizar las operaciones solicitadas por sobre la aplicación. Además, la misma deberá contar con un manual de usuario destinado a la especificación de cómo es que la aplicación debe ser utilizada.

2. Implementación de TDAs

A continuación, se listan los TDAs requeridos. En todos los casos, se debe documentar su implementación utilizando comentarios *Javadoc*.

- ***TDA Pila:*** Implementar el *TDA Pila*, para elementos de tipo genérico E, utilizando un arreglo. Para ello, respete la interfaz *Stack* provista por la cátedra.

- ***TDA Cola:*** Implementar el *TDA Cola*, para elementos de tipo genérico E, utilizando un arreglo circular. Para ello, respete la interfaz *Queue* provista por la cátedra.

- ***TDA Lista:*** Implementar el *TDA Lista*, para elementos de tipo genérico E, utilizando una estructura doblemente enlazada con centinelas. Para ello, respete la interfaz *PositionList* provista por la cátedra.

- ***TDA Mapeo:*** Implementar el *TDA Mapeo* para almacenar pares (clave, valor), donde el tipo K de las claves y el tipo V de los valores son genéricos. Utilizar para la implementación del mapeo una Tabla Hash Cerrada, adoptando la política de resolución lineal de colisiones. Se debe respetar la interfaz *Map* provista por la cátedra.

- ***TDA Diccionario:*** Implementar el *TDA Diccionario* para almacenar pares (clave, valor), donde el tipo K de las claves y el tipo V de los valores son genéricos. Utilizar para la implementación del diccionario una Tabla Hash Abierta. Se debe respetar la interfaz *Dictionary* provista por la cátedra.

- ***TDA Árbol:*** Implementar el *TDA Árbol* para representar un árbol general donde los rótulos del mismo son de tipo genérico E. Utilizar para la implementación del árbol una estructura nodo que mantiene una colección de hijos y referencia al padre. La colección de hijos debe modelarse haciendo uso del *TDA Lista* solicitado anteriormente. Se debe respetar la interfaz *Tree* provista por la cátedra.

3. Implementación de la aplicación *ED-Drive*

Desarrollar una aplicación denominada *ED-Drive* que permita un conjunto de operaciones por sobre un sistema de archivos virtual. La aplicación manipula dicha información a partir de un jerarquía de tipo árbol (ver secciones 3.4 y 3.5), generada en función de un archivo de texto que, siguiendo el formato enunciado en la sección 3.1, indica la composición del sistema respecto a los directorios y archivos que contiene.

Para la aplicación, se debe desarrollar una componente lógica y una componente gráfica (interfaz). La componente lógica debe permitir la siguiente funcionalidad:

1. **Generar jerarquía:** dada la ruta de acceso a un archivo de texto A con extensión *.ed19*, parsea el contenido de A (ver secciones 3.1 y 3.4) creando una jerarquía de tipo árbol (ver sección 3.4) que representa el sistema de archivos virtual. Si el archivo A no puede ser procesado o no existe, la jerarquía no debe ser generada.

2. **Agregar archivo:** dada la ruta completa dentro del sistema virtual de archivos de un directorio D (ver sección 3.2), y el nombre de un archivo A, agrega el archivo A al directorio D. En caso de que D no exista, la jerarquía no debe ser modificada.

3. **Eliminar archivo:** dada la ruta completa dentro del sistema virtual de archivos de un archivo A (ver sección 3.2), elimina el archivo A del directorio que lo contiene. En caso de que A no exista, la jerarquía no debe ser modificada.

4. **Agregar directorio:** dada la ruta completa dentro del sistema virtual de archivos de un directorio D_1 (ver sección 3.2), y el nombre de un directorio D_2, agrega el directorio D_2 dentro del directorio D_1. En caso de que D_1 no exista, la jerarquía no debe ser modificada.

5. **Eliminar directorio:** dada la ruta completa dentro del sistema virtual de archivos de un directorio D (ver sección 3.2), elimina el directorio D y todo archivo y sub-directorio con o sin archivos que contenga. En caso de que D no exista, la jerarquía no debe ser modificada.

6. **Mover directorio:** dada la ruta completa dentro del sistema virtual de archivos de dos directorios D_1 y D_2 (ver sección 3.2), elimina el directorio D_1 del directorio que actualmente lo contiene y lo mueve dentro del directorio D_2. En caso de que D_1 o D_2 no existan, o que D_1 sea el directorio raíz de la jerarquía, esta última no debe ser modificada.

7. **Cantidad de directorios y archivos:** dada la jerarquía del sistema de archivos virtual, computa la cantidad de directorios y la cantidad de archivos que contiene el sistema, retornando estas cantidades en un objeto de tipo *Pair* (ver sección 3.4).

8. **Listado por niveles:** dada la jerarquía del sistema de archivos virtual, genera una lista de *strings* tal que, siguiendo el formato definido en la sección 3.3, lista los directorios y archivos del sistema de archivos por niveles.

9. **Listado por extensión:** dada la jerarquía del sistema de archivos virtual, genera una diccionario que modela como claves las extensiones de los archivos, y como valores los nombres de los archivos correspondientes a dicha extensión.

10. **Listado por profundidad:** dada la jerarquía del sistema de archivos virtual, genera un mapeo que modela como claves las rutas completas de cada uno de los directorios (ver sección 3.2), y como valores la profundidad que dichos directorios poseen en la jeraquía.

Finalmente, la componente gráfica debe implementar una interfaz que le permita al usuario hacer uso de todas las operaciones que implementa la componente lógica. Para esto, por ejemplo, deberá definir funcionalidad en la interfaz que le permitan al usuario indicar la ubicación de un archivo *mi-sistema-archivos.ed19* para generar la jerarquía del sistema de archivos virtual, así como en todo momento deberá permitir visualizar gráficamente cómo es que la jerarquía se compone. En el caso de los listados por nivel, extensión y profundidad, se debe elegir una alternativa para que estos puedan ser visualizados en la interfaz cada vez que sean generados. Adicionalmente, cuando una operación de la componente lógica no finalice adecuadamente, se debe indicar esto mediante algún aviso especificando el origen del error (ver sección 3.5).

3.1. Especificación del formato del archivo de entrada de *ED-Drive*

El formato del archivo de entrada de la aplicación *ED-Drive* es similar al formato de un archivo *.xml*. Considerando la jerarquía del sistema de archivos de ejemplo de la Figura 1, luego, el archivo de entrada para *ED-Drive* especificando dicha estructura sigue el formato indicado en la Figura 2.

3.2. Especificación del formato de rutas en el sistema *ED-Drive*

Considerando la jerarquía del sistema de archivos de ejemplo de la Figura 1, luego, una ruta completa hacia un archivo A o un directorio D en *ED-Drive* será considerada de la siguiente manera:

- La ruta completa al archivo *Stack.java* será *ED-Proyecto\Fuentes\TDAPila\Stack.java*
- La ruta completa al directorio *Javadoc* será *ED-Proyecto\Documentacion\Javadoc*

3.3. Especificación del formato de listado por niveles

Considerando la jerarquía del sistema de archivos de ejemplo de la Figura 1, luego, el listado por niveles en *ED-Drive* será dado mediante una lista de la siguiente manera:

Listado: *<ED-Proyecto, \, Jars, Fuentes, Documentacion, \, ED-Drive-ejecutable.jar, TDA-Pila, Manual-de-usuario.pdf, Javadoc, \, Stack.java, PilaConArreglo.java, EmptyStackException.java, Index.html>*

Cada elemento de la secuencia es un *string* representando un archivo o un directorio, o bien, el símbolo "\" representando el fin de un nivel en la jerarquía.

3.4. Restricciones sobre la implementación de la funcionalidad de *ED-Drive*

- El parseo sobre el archivo de entrada debe realizarse utilizando únicamente los *TDAPila* y *TDACola* solicitados para el presente proyecto. En el mismo se debe validar que el archivo de entrada respete el formato enunciado.
- La jerarquía del sistema de archivos virtual debe representarse haciendo uso del *TDAArbol* solicitado para el presente proyecto, teniendo en cuenta la siguiente definición de tipos: *Tree<Pair<String, PositionList<String>>>*, donde el rótulo de los nodos del árbol son

de tipo *Pair*, y la clave del par (un *string*) identificará el nombre de un directorio A, mientras el valor del par (una *PositionList<String>*) identificará a los archivos que se encuentran dentro del directorio A.

- La operación *Cantidad de directorios y archivos* retornará un *Pair<Integer,Integer>*, donde la clave del par representa la cantidad de directorios mientras el valor del par representa la cantidad de archivos.

3.5. Tips para la implementación de funcionalidad de *ED-Drive*

- Asuma que tanto los nombres de los directorios como de los archivos no contienen espacios en blanco.
- Tenga en cuenta la funcionalidad de las funciones utilizadas en la siguiente porción de código:

```
public static void main (String [] args) {
  String cadena_1 = "Esto|es|una prueba";
  String separador = Pattern.quote("|");
  String [] partes = cadena_1.split(separador);
  for(String s : partes)
    System.out.println(s);

  String cadena_2 = "Hola \n ¿Que    tal? \n \t ¿Todo bien?";
  System.out.println("Cadena 2 original: " + cadena_2);
  System.out.println("Cadena 2 operada: " + cadena_2.replaceAll("\\s",""));
  System.out.println("Cadena 2 luego de ser operada: " + cadena_2);
}
```

- Recuerde que usted ha aprendido a modelar situaciones excepcionales o de falla mediante TDAs del tipo *Exception*. A la hora de que la componente gráfica y lógica intercambien servicios, esto puede ser especialmente útil.

4. Manual de usuario

Confeccionar un manual de usuario dirigido a un usuario no programador. En el mismo, se debe describir claramente las funcionalidades disponibles en la aplicación *ED-Drive*, tanto como la forma en la que dicha funcionalidad debe ser utilizada. Se espera que el manual contenga imágenes ilustrativas de la aplicación, un listado completo de todas las operaciones, y una descripción formal y precisa de cómo debe utilizarse la interfaz: qué opciones tiene el usuario para operar, qué debe ingresar ante el uso de una u otra operación, qué sucederá si presiona un determinado botón, cómo se indican los parámetros, cuáles son las respuestas del sistema, etc. Recuerde que el usuario al que se dirige el manual de usuario no es programador, por lo que no se debe documentar cómo es que se implementan las funcionalidades, sino cómo es que deben utilizarse.

5. Sobre la entrega

- Las comisiones estarán conformadas por 3 alumnos, y serán las que oportunamente registró y notificó la cátedra.
- Cada uno de los TDAs solicitados, deben entregarse en paquetes diferentes nombrados *TDAPila*, *TDACola*, *TDALista*, *TDAMapeo*, *TDADiccionario*, *TDAArbol*, que contendrán **únicamente** cada una de las interfaces y clases necesarias para su implementación. Tenga en cuenta que no se aceptarán TDAs nombrados de ninguna otra manera.
- Tanto la componente lógica como la componente gráfica de la aplicación deben entregarse en paquetes diferentes nombrados *Controlador* y *GUI* respectivamente, que contendrán **únicamente** cada una de las interfaces y clases necesarias para su implementación.
- La entrega de todo el *código fuente*, la documentación generada mediante *Javadoc* y el *Manual de usuario* se realizará a través de un archivo comprimido **zip** o **rar**, denominado ***P-COM-XX-Apellido1-Apellido2-Apellido3***, que debe incluir las siguientes carpetas:
  - **Fuentes**, donde se deben incorporar los paquetes *TDAPila*, *TDACola*, *TDALista*, *TDAMapeo*, *TDADiccionario*, *TDAArbol*, *Controlador* y *GUI* que contienen los archivos fuentes (*".java"*, **ningún otro**) de cada TDA solicitado, así como también los fuentes utilizados para la aplicación.
  - **Jar**, donde se debe incorporar el archivo ejecutable *".jar"* desde el cual se ejecutará la aplicación *ED-Drive*.
  - **Documentación**, donde se debe incorporar por un lado el manual de usuario de la aplicación en formato PDF (**ningún otro**), y por otro, una carpeta denominada **Javadoc**, donde se encontrarán todos los archivos generados mediante la herramienta *Javadoc*. Recuerde que deben estar comentadas todas las clases (métodos públicos y privados, constructores, excepciones) utilizadas tanto para la implementación de los TDAs como de las componentes gráfica y lógica.
- El archivo comprimido debe alojarse en algún repositorio online (por ejemplo, Google Drive, Dropbox, etc.), y un enlace a dicho repositorio debe enviarse por e-mail, respetando el siguiente formato:
  - **Para:** *dcic.ed@gmail.com*
  - **Asunto:** *2019 :: COM XX :: Apellido1 - Apellido2 - Apellido3*
- El e-mail debe ser enviado con anterioridad al día **Martes 28 de Mayo de 2019, 23:59 hs**. Se considerará como hora de ingreso, la registrada en el servidor de e-mail de GMail.

6. Sobre la corrección

- La cátedra evaluará tanto el **diseño** e **implementación** como la **documentación** y **presentación** del proyecto, y el cumplimiento de **todas** las condiciones de entrega.
- Tanto para compilar el proyecto, como para verificar su funcionamiento, se utilizarán las interfaces y testers disponibles en el sitio web de la materia (enlace).

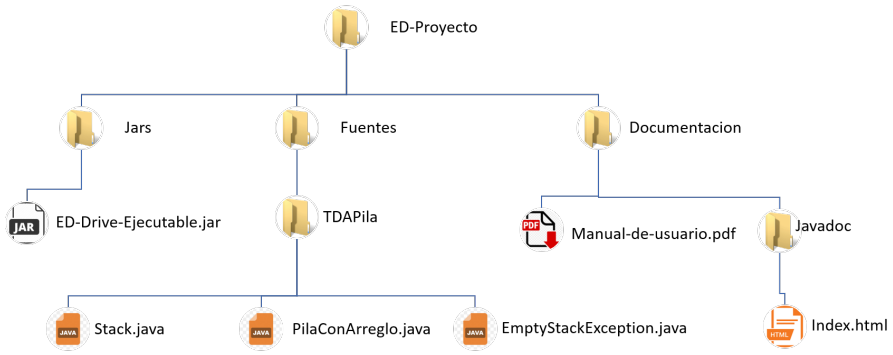

Figura 1: Jerarquía de ejemplo de un sistema de archivos

```
<carpeta>
    <nombre>ED-Proyecto</nombre>
    <lista_sub_carpetas>
        <carpeta>
            <nombre>Jars</nombre>
            <archivo>ED-Drive-Ejecutable.jar</archivo>
        </carpeta>
        <carpeta>
            <nombre>Fuentes</nombre>
            <lista_sub_carpetas>
                <carpeta>
                    <nombre>TDAPila</nombre>
                    <archivo>Stack.java</archivo>
                    <archivo>PilaConArreglo.java</archivo>
                    <archivo>EmptyStackException.java</archivo>
                </carpeta>
            </lista_sub_carpetas>
        </carpeta>
        <carpeta>
            <nombre>Documentacion</nombre>
            <archivo>Manual-de-usuario.pdf</archivo>
            <lista_sub_carpetas>
                <carpeta>
                    <nombre>Javadoc</nombre>
                    <archivo>Index.html</archivo>
                </carpeta>
            </lista_sub_carpetas>
        </carpeta>
    </lista_sub_carpetas>
</carpeta>
```

Figura 2: Formato del archivo entrada de *ED-Drive*, para la jerarquía de ejemplo de la figura 1