UPM ETSIINF. Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software.

Examen Parcial de Programación2 (GMI)
Ejercicio Práctico
23 de mayo de 2022

Duración: La duración total del ejercicio será de **90 minutos**.
Calificaciones: Las notas se publicarán como muy tarde el día **6 de junio del 2022**.
Revisión: Las revisiones será como muy tarde el **8 de junio del 2022** previa petición por correo electrónico al profesor que se indique por el foro de la asignatura.
El ejercicio se puntúa sobre 10 y tiene un peso del 65% sobre la nota total del examen

Se dispone de un tipo de editor que implementa una serie de operaciones. Este editor se caracteriza por ser un editor secuencial. El cursor solo puede ser movido por las operaciones de escribir texto, borrar texto, insertar línea y borrar línea.

En el siguiente diagrama se muestran las clases e interfaces que participan. No se muestran las excepciones.

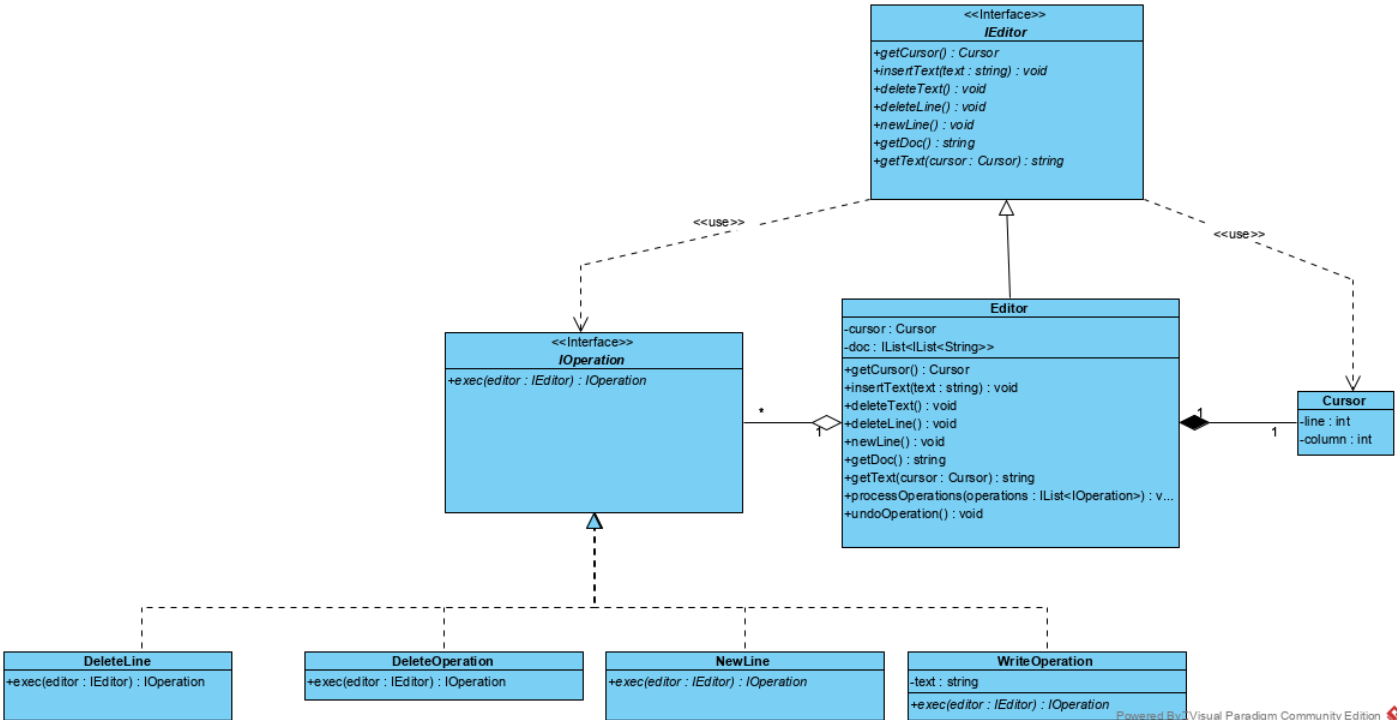

Figura 1 diagrama de clases del editor

La clase Editor implementa la interfaz IEditor que se usa para desacoplar las operaciones del editor. Descripción de la clase Editor:

- Atributos: se enumeran todos los atributos que aparecen en el diagrama, falta un atributo para poder implementar el método undoOperation.
    - doc: Es una lista de listas de String. Donde cada lista de la lista representa una línea y cada elemento de la lista de listas es un texto (String).
    - cursor: Instancia u objeto de la clase Cursor que indica la posición en el atributo doc en la que se realizaría la siguiente operación. Inicialmente el cursor debe estar en la línea 0 y columna 0. El cursor indicará la posición de un texto completo en el atributo doc, **y no solo un carácter**.

- Operaciones o métodos: la clase editor consta de varias operaciones o métodos.
    - getCursor: método que retorna una copia del cursor del editor.
    - insertText: inserta el texto pasado en la posición en la que se encuentre el cursor. Avanza una posición la columna del cursor**.**
    - deleteText: retrocede el cursor a la posición anterior (columna-1) y borra el texto que hay en la nueva posición del cursor en el doc. La columna del cursor debe ser > 0 en caso contrario se genera la excepción **ErrorIncorrectCursorPosition.**
    - getDoc: Retorna un String con el documento conteniendo todas las líneas del atributo doc.
    - getText: Retorna el texto asociado a la posición del cursor dado.
    - deleteLine: borra la línea actual en la que se encuentra el cursor. La línea del cursor debe ser mayor que cero. En caso contrario se genera la excepción **ErrorIncorrectCursorPosition**
    - newLine: inserta una nueva línea después de la línea a la que hace referencia el cursor.
    - processOperations: Este método recibe una lista de IOperation. Ejecuta las operaciones en el orden en el que aparecen en la lista, y va guardando la información necesaria para deshacer dichas operaciones. Tenga presente que el deshacer en cualquier editor sigue una política LIFO (Last In First Out). Si una operación genera la excepción **ErrorInOperation** debe propagarse al llamante.
    - undoOperation: Deshace la última operación realizada siguiendo el criterio de selección LIFO. Si no hay operaciones para deshacer se genera la excepción **ErrorNoUndo**. **Las operaciones deshechas no se pueden rehacer.** Si una operación genera la excepción **ErrorInOperation** debe propagarse al llamante.

La interfaz IOperation consta de la siguiente operación:

- exec: Esta operación ejecuta una acción sobre el editor que se le pasa como parámetro. Retorna la operación inversa a la que se realiza. Si por algún motivo se genera una excepción al realizar una operación, se renombrará esa excepción por (**ErrorInOperation**). Por ejemplo, la operación inversa de insertText sería deleteText.

La clase Cursor consta de sendos métodos *getters* (sólo lectura) para acceder a la fila y la columna y de setters para modificar la fila y la columna.

El siguiente código demuestra cómo se usaría la clase Editor y las operaciones:

```
public static void main(String[] args) {
  IOperation [] operations = {new WriteOperation("hola"),
      new WriteOperation(" vamos a una nueva línea"),
      new NewLine(), new WriteOperation("en una nueva línea")};
  Editor editor = new Editor();
  try {
    editor.processOperations(operations);
    System.out.print(editor.getDoc());
    System.out.println("<Undo last write>");
    editor.undoOperation();
    System.out.print(editor.getDoc());
  } catch (ErrorInOperation | ErrorNoUndo e) {
    System.err.println("Ubs! Something was wrong");
    e.printStackTrace();
  }
```

}
El resultado que se obtendría por consola sería:

```
hola vamos a una nueva línea
en una nueva línea

<Undo last write>
hola vamos a una nueva línea
```

Se pide completar la clase Editor.

1. Añada los atributos que considere necesarios y complete el constructor para inicializar los atributos con el fin de poder implementar el resto de las operaciones (2 puntos)

```
public class Editor implements IEditor {

  private IList<IList<String>> doc;
  private Cursor cursor;
  //TODO add necessary attributes


  public Editor() {
    this.doc = new ArrayList <IList<String>>();
    this.doc.add(0,new ArrayList <>()); //Line 0
    this.cursor = new Cursor();
    //TODO: initialize added attributtes

  }
```

2. Completar el control de la precondición en deleteText. (1.5 Punto)

```
@Override
public void deleteText()                    {
  //TODO completes preconditions verifications

  this.cursor.setColumn(this.cursor.getColumn() - 1);
  this.doc.get(this.cursor.getLine()).removeElementAt(
            this.cursor.getColumn());
}
```

3. Implementar el método processOperations según lo indicado más arriba y teniendo encuenta que las operaciones pueden ser deshechas llamando a undoOperation siguiendo un criterio LIFO, es decir se deshace primero la última operación realizada (2.5 puntos)

```
/**
 * PRE: True
 * Process all operations in array and adds the corresponding to
 * inverse operations.
 * @param operations
 * @throws ErrorInOperation
 */
public void processOperations(IOperation[] operations)        {
  //TODO: Process operations and keep in mind that these operations
  // may be undone


}
```

4. Implementar el método undoOperation según lo indicado más arriba. (2 Puntos)

```
/**
 * PRE: there is undo operations otherwise raise ErrorNoUndo
      undo the last operation performed on editor
 * @throws ErrorInOperation
 */
public void undoOperation()     {
  //TODO: Process next undo operation


  }
```

5. Implementar el método getDoc según lo indicado más arriba. (2 Puntos)

```
@Override
public String getDoc() {
  String docResult = "";
  // TODO completes this method


  return docResult;
  }
```