Programación Avanzada I. Práctica 5.2

Tema 6. Colecciones y Correspondencias

Características de la práctica

En esta práctica se pretende que el alumno se familiarice con las correspondencias. Para ello, se propone un ejercicio donde se deben emplear varios tipos de correspondencias y cómo combinarlas con estructuras ya estudiadas como listas y conjuntos.

Ejercicio 1. (proyecto pa1p52, paquete indices)

Se pretende realizar una aplicación que permita realizar varios listados con las palabras que aparecen en un texto de manera que podamos conocer para cada palabra del texto:

- Cuántas veces aparece.
- Las líneas en las que aparece.
- Las líneas y la posición (o posiciones) dentro de cada línea en las que aparece.

Para ello, vamos a construir tres tipos distintos de índices:

- **`IndiceContador`**, que mostrará cada palabra con el número de veces que aparece.
- **`IndiceLineas`**, que mostrará cada palabra con todas las líneas en las que aparece.
- **`IndicePosicionesEnLineas`**, que mostrará cada palabra con todas las líneas en las que parece, y dentro de cada línea, todas las posiciones en las que aparece.

Por ejemplo, para el texto (se han suprimido las tildes intencionadamente):

> *Guerra tenia una jarra y Parra tenia una perra, pero la perra de Parra rompio la jarra de Guerra.*
>
> *Guerra pego con la porra a la perra de Parra. ¡Oiga usted buen hombre de Parra! Por que ha pegado con la porra a la perra de Parra.*
>
> *Porque si la perra de Parra no hubiera roto la jarra de Guerra, Guerra no hubiera pegado con la porra a la perra de Parra.*

Usando **`IndiceContador`** obtendríamos la siguiente salida, donde se muestra cada palabra seguida del número de veces que aparece:

```
a            3
buen         1
con          3
de           8
guerra       5
ha           1
hombre       1
hubiera      2
jarra        3
...
```

Con **`IndiceLineas`** obtendríamos la siguiente salida, donde se muestra las palabras del texto junto con las líneas donde aparecen:

```
a          <2,3>
buen       <2>
con        <2,3>
de         <1,2,3>
guerra     <1,2,3>
ha         <2>
hombre     <2>
hubiera    <3>
jarra      <1,3>
```

```
...
```

Por último, con **`IndicePosicionesEnLineas`** obtendríamos un índice en el que para cada palabra se muestran las líneas en las que aparece y dentro de cada línea, todas las posiciones en las que aparece:

```
a
                2 <6,24>
                3 <21>
buen
                2 <13>
con
                2 <3,21>
                3 <18>
de
                1 <13,18>
                2 <9,15,27>
                3 <5,12,24>
...
```

Para separar las palabras dentro de cada línea se proporcionan unos delimitadores. En los ejemplos anteriores, la cadena de delimitadores es `"[ .,:;\\-\\!\\¡\\¿\\?]+"` (también son válidos los siguientes delimitadores `"[^a-zA-Z0-9áéíóúüÁÉÍÓÚÜ]+"`). En el siguiente diagrama se muestran las interfaces y clases a desarrollar dentro del paquete **`indices`**.

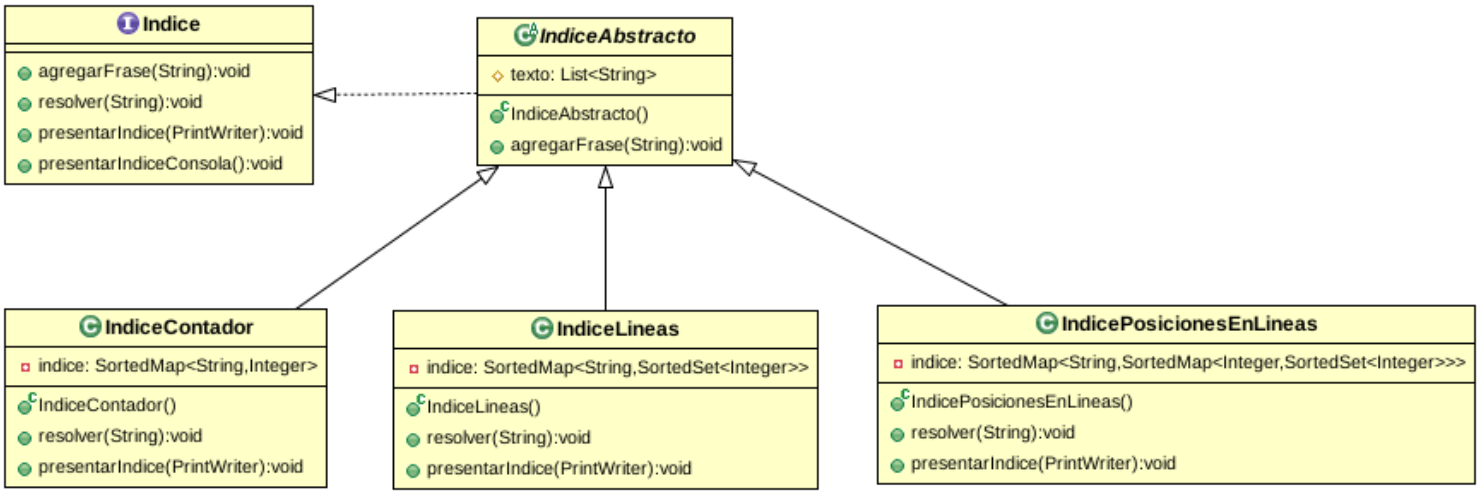

Figura 1: Diagrama de clases UML

Interfaz `Indice`

Dado que, a pesar de sus diferencias, los tres *índices* tienen una funcionalidad similar, se va a definir una interfaz **`Indice`** que después implementarán los tres tipos de índices. La interfaz especifica los siguientes métodos:

- `void agregarFrase(String frase);`

  Agrega una frase o línea de texto a las que ya tenga almacenadas el índice. Estas líneas serán las que formarán el texto a analizar.

- `void resolver(String delimitadores);`

  Recibe los delimitadores de las palabras para cada línea, y debe construir el índice. Por ejemplo, los delimitadores pueden ser `"[ .,:;\\-\\!\\¡\\¿\\?]+"`, o también `"[^a-zA-Z0-9áéíóúüÁÉÍÓÚÜ]+"`.

- `void presentarIndice(PrintWriter pw);`

  Muestra el resultado (en el formato indicado arriba para cada caso) en `pw` .

- `deafult void presentarIndiceConsola();`

Muestra el resultado (en el formato indicado arriba para cada caso) en la consola. Es un método **por defecto** de esta interfaz, por lo que debe proporcionar una implementación *por defecto*, invocando para ello al método `presentarIndice(PrintWriter pw)` con un `PrintWriter` asociado a la consola.

Clase Abstracta `IndiceAbstracto`

Esta clase define las características comunes que tienen los tres índices e **implementa** la interfaz `Indice`.

Dispone de una lista *protegida* de `String` (**frases**) que almacenará las frases. Además:

- Define un constructor que crea adecuadamente la estructura.
- Redefine el método `void agregarFrase(String frase)` que agrega una frase o línea de texto a las que ya tenga almacenadas en `frases`. Estas líneas serán las que formarán el texto a analizar. La última línea agregada será la última línea del texto.

NOTA: Como se puede apreciar, las variables y métodos definidos en esta clase, son comunes a los tres tipos de índices.

Clase `IndiceContador`

La clase `IndiceContador` **hereda** de la clase abstracta `IndiceAbstracto`, y tendrá, además de la variable **`frases`** que hereda de `IndiceAbstracto`, una variable **`indice`** donde se almacenará el índice que se va a construir a partir del texto disponible en la variable de instancia `frases` y los delimitadores que se especifiquen. El índice a construir se guardará en una **correspondencia** en la que a cada palabra del texto se le asocia el número de apariciones de dicha palabra (véase la salida en el caso del ejemplo anterior). Las palabras se deben mantener ordenadas lexicográficamente.

Para ello, la clase `IndiceContador` proporcionará un constructor e implementará los métodos heredados de `Indice`:

- Define un constructor que inicialice adecuadamente las estructuras que sean necesarias para desarrollar la aplicación. Inicialmente no habrá ningún texto sobre el que operar; tanto el texto como los delimitadores se introducirán con algunos métodos.
- Redefine el método `void resolver(String delimitadores)` que debe construir el índice, calculando el número de apariciones de cada palabra significativa en el texto `frases` y completando la estructura `indice` para mantener esta información.

  Obsérvese que:

  - Debe reiniciar primero el índice a vacío (`clear()`).
  - Las palabras se almacenan en minúsculas.
  - Para separar las palabras dentro de cada línea del texto utilizaremos una instancia de la clase `Scanner` con los delimitadores proporcionados.
  - El algoritmo que se utilice debe dar una sola pasada por las frases para ir completando el índice.
- Redefine el método `presentarIndice(PrintWriter pw)` para producir la salida en el formato esperado.

Podemos probar el funcionamiento de esta clase con la aplicación `EjIndice1` presentada al final del enunciado.

Clase `IndiceLineas`

La clase `IndiceLineas` hereda también de `IndiceAbstracto` y sigue un patrón muy similar al de `IndiceContador`. Las principales diferencias con esta son:

- La variable **`indice`** ahora será una correspondencia en la que a cada palabra del texto se le asocia un conjunto con los números de líneas en donde aparece (véase el ejemplo anterior). Las palabras estarán ordenadas lexicográficamente y los números de línea de menor a mayor.
- Redefine el método `void resolver(String delimitadores)` para construir el índice como se indica arriba.
- Redefine el método `void presentarIndice(PrintWriter pw)` para que genere el listado que se muestra en el ejemplo.

Podemos probar el funcionamiento de esta clase con la aplicación `EjIndice1` presentada al final del enunciado.

Clase `IndicePosicionesEnLineas`

Por último, la clase `IndicePosicionesEnLineas` hereda también de `IndiceAbstracto` y sigue un patrón muy similar al de `IndiceContador` e `IndiceLineas`. En este caso **`indice`** será una correspondencia en la que a cada palabra se le asocia una segunda correspondencia en la que a cada número de línea se le asocia un conjunto con las posiciones de la palabra en cada número de línea. Las palabras estarán ordenadas lexicográficamente y las líneas y posiciones de menor a mayor.

Podemos probar el funcionamiento de esta clase con la aplicación `EjIndice1` presentada al final del enunciado.

Aplicación `EjIndice`

Se puede usar el siguiente programa principal para probar las clases anteriores (debe seleccionar la clase a utilizar):

```
import indices.*;

public class EjIndice {
    public static void main(String args[]) {
        // String delimitadores = "[^a-zA-Z0-9áéíóúüÁÉÍÓÚÜ]+";
        String delimitadores = "[ .,:;\\-\\!\\¡\\¿\\?]+";
        Indice cp = new IndiceContador();               // seleccione la clase a utilizar
        // Indice cp = new IndiceLineas();              // seleccione la clase a utilizar
        // Indice cp = new IndicePosicionesEnLineas();  // seleccione la clase a utilizar
        cp.agregarFrase("Guerra tenia una jarra y Parra tenia una perra, "
                + "pero la perra de Parra rompio la jarra de Guerra.");
        cp.agregarFrase("Guerra pego con la porra a la perra de Parra. "
                + "¡Oiga usted buen hombre de Parra! "
                + "Por que ha pegado con la porra a la perra de Parra.");
        cp.agregarFrase("Porque si la perra de Parra no hubiera roto "
                + "la jarra de Guerra, "
                + "Guerra no hubiera pegado con la porra "
                + "a la perra de Parra.");
        cp.resolver(delimitadores);
        cp.presentarIndiceConsola();
    }
}
```