Programación 2
Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas
Práctica Nº 10

Para cada uno de los siguientes problemas Implementarlos en Java.

1. Listas

Definir una interfaz similar a la interfaz List de Java, llamada SimpleList, que defina los siguientes métodos:

- `int size();`
  Retorna la cantidad de elementos de la lista
- `boolean isEmpty();`
  Retorna true si la lista no contiene elementos y false si contiene al menos un elemento
- `boolean contains(Object element);`
  Retorna true si la lista contiene al objeto o pasado como parámetro
- `void add(Object element);`
  Agrega el objeto o al final de la lista
- `Object add(int index, Object element);`
  Agrega el objeto element en la posición index pasada como parámetro. Se realizan los corrimientos necesarios con los elementos posteriores a dicha posición. Si la posición index no existe retorna null, caso contrario retorna el mismo objeto agregado.
- `Object set(int index, Object element);`
  Similar al método anterior, pero se sobreescribe el elemento existente en la posición index. El elemento que anteriormente ocupaba la posición index es retornado al usuario. Si la posición index no existe retorna null.
- `boolean remove(Object element);`
  Elimina la primera ocurrencia del objeto element pasado como parámetro. Si el objeto no se encontró, retorna false. Caso contrario retorna true.
- `Object remove(int index);`
  Elimina el objeto que ocupa la posición index pasada como parámetro y lo retorna al usuario. Si no existe, retorna null.
- `void addAll(List otherList);`
  Agrega todos los elementos de la lista otherList pasada como parámetro al final de la lista que recibe el mensaje
- `void clear();`
  Elimina todos los elementos de la lista
- `Object get(int index);`
  Retorna el objeto almacenado en la posición index. Si no existe la posición retorna null;
- `int indexOf(Object element);`
  Retorna la posición (índice) en la que se encuentra el objeto element pasado como parámetro. Si el objeto no existe en la lista, retorna -1

Codificar dos implementaciones de esta interfaz:

1. Una lista vinculada, en la cual cada objeto se almacena en un Nodo que conoce el siguiente elemento de la lista. La lista almacena únicamente una referencia al primer y último elemento de la lista. Los demás elementos quedan referenciados a partir del primero.

2. Una lista basada en arreglos. Cuando se crea la lista, se inicializa un arreglo de tamaño predefinido que se va utilizando para almacenar los elementos que se insertan en la lista. Cuando este arreglo se completa, la estructura automáticamente crea un nuevo arreglo del doble del tamaño actual, conteniendo los elementos existentes. De esta forma tiene espacio disponible para agregar más elementos.

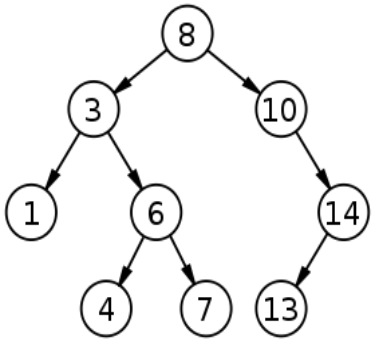
2. Procesador de Texto con plugins

Se debe diseñar e implementar un procesador de texto basado en plugins. La clase principal que modela el procesador de textos debe tener un método ejecutarPlugin(Plugin plugin) que permita ejecutar cualquier plugin pasado como parámetro y el mismo se aplica al texto del documento actual. Crear una interfaz llamada Plugin que tenga un método ejecutar(). Luego, implementar esta interfaz en los siguientes plugins concretos:

- ContadorPalabrasPlugin: Cuenta las palabras en el texto actual y guarda el resultado para que pueda luego ser consultado.
- ContadorOcurrenciasPlugin: Cuenta la cantidad de veces que una palabra dada por el usuario aparece en el texto actual.
- ReemplazoTextoPlugin: Reemplaza una palabra específica en el texto actual por otra palabra ingresada por el usuario.
- EliminarPalabraPlugin: Elimina del texto todas las palabras que coincidan con una dada por el usuario.
- GuardarArchivoPlugin: Guarda el contenido actual del documento en un archivo en el sistema de archivos. Una forma rápida de escribir en un archivo es la siguiente:

```
String textoAGuardar = "Hello File!";
String path = "C:\\Users\\marce\\Downloads\\archivoDestino.txt";
try {
    Files.write(Paths.get(path), textoAGuardar.getBytes());
} catch (IOException e) {
    System.out.println(e);
}
```

3. Árbol binario de búsqueda



Un árbol binario es una estructura de datos formada por nodos que contienen un determinado valor (en principio un número entero). El primer elemento agregado a la estructura se conoce con el nombre de “raíz” y es el único punto de acceso a la misma. Cada nodo, puede tener un nodo “hijo” a su izquierda y un nodo hijo a su derecha cumpliendo con la restricción que los valores a su izquierda son valores menores que su propio valor, y los valores a su derecha son

valores mayores (no se almacenan valores repetidos). Los nodos sin hijos se conocen como "hojas".

1. Implementar la funcionalidad para agregar un nuevo valor a la estructura.
2. Implementar un método que permita recorrer la estructura en orden, es decir, todos los elementos a la izquierda, luego la raíz y después todos los elementos a la derecha. Al recorrer los elementos es necesario que se defina una acción la cual se va a ejecutar. Para poder trabajar de forma transparente y que se pueda extender la funcionalidad definir una interfaz *AccionEjecutable*.
   La misma posee un método public void ejecutarNodo(Object nodo) que realiza alguna acción sobre el nodo recibido. Por ejemplo, una posible implementación sería:

```
public class ImprimirEnPantalla implements AccionEjecutable {

    public void ejecutarNodo(Object nodo) {
        System.out.println(nodo.toString());
    }
}
```

3. Crear una acción que permita incorporar los elementos de forma ordenada ascendentemente a una lista.
4. Crear una acción que permita incorporar los elementos de forma ordenada descendentemente a una lista.
5. Crear una acción que cuente la cantidad de elementos visitados.

4. Sistema de Notificaciones

Una empresa de desarrollo de software utiliza para sus aplicaciones una biblioteca que compró y que le permite enviar y recibir notificaciones por email, sms o por internet. Esta biblioteca cuenta con una interface **Notificador** y 3 subclases que implementan dicha interface que a continuación se detallan:

```
public interface Notificador {
    void enviarNotificacion(String s);  //Envía una notificación
    String getNotificacion(); // Recupera una notificación

}

public class NotificadorEmail implements Notificador {
    //implementa la interface Notificador basada en emails
    NotificadorEmail (String emailEnvio, String emailRecepcion){
        ...
    }
    ...
}
public class NotificadorSms implements Notificador {
    //implementa la interface  Notificador basada en sms
    NotificadorSms (String numeroTelefono){
        ...
    }
```

```
. . .
}
```

La empresa quiere agregar nuevas funcionalidades al soporte de notificaciones, pero no dispone del código fuente y no puede modificar el comportamiento de estas clases.

Defina las clases (nombre, superclase, atributos y métodos) para implementar una solución orientada a objetos en base a la interface y las clases dadas **sin modificarlas,** que agregue el soporte necesario para poder extender el comportamiento los notificadores con las siguientes funcionalidades: ():

1. Comprimir la información que se envía de manera de minimizar el volumen de transferencia. Para comprimir/descomprimir usar los métodos de clase de la clase **Gzip**:
   - String comprimir(String original)
   - String descomprimir(String comprimido)
2. Encriptar la información de manera de aumentar la seguridad. Para encriptar/desencriptar se cuenta con la clase **Encriptador** que provee los siguientes métodos:
   - String encriptar(String original)
   - String desencriptar(String)

   y el constructor
   - public Encriptador(String clave) que crea instancia del encriptador con una clave en particular que se utiliza luego internamente en el proceso de compresión.
3. Estadísticas de la cantidad de mensaje enviados y recibidos

La solución que proponga debe separar estas funcionalidades y permitir combinarlas en tiempo de ejecución, así como permitir agregar nuevas funcionalidades.

Tenga en cuenta además que la empresa tiene varias aplicaciones ya desarrolladas en base a las interfaces definidas en esta biblioteca y la incorporación de la nueva funcionalidad no debe impactar con modificaciones al código de las mismas. Se debe poder utilizar la solución aportada (que combina las nuevas funcionalidades con las existentes) en donde antes se usaban notificadores de sms, red o email notificadores. Por ejemplo, un notificador por email que calcula estadísticas y comprime la información, o un notificador por sms que comprime, encripta y calcula estadísticas, o cualquier otra combinación de notificadores y funcionalidades.