Trabajo Práctico Obligatorio No. 1

El objetivo de este trabajo práctico es evaluar la comprensión y el manejo de construcciones fundamentales de la programación orientada a objetos, incluyendo tipos primitivos, objetos y clases, sentencias de programación que abarcan ciclos, condicionales y composición secuencial, especificación de funcionalidad con invariantes, precondiciones y postcondiciones, y manejo de colecciones básicas (listas y arreglos).

El Proyecto

Este trabajo práctico requiere completar la implementación de un proyecto Java denominado `CifradoCesar`. Corresponde a una implementación de tres componentes básicas:

- Mensajes. Los mensajes representan texto en formato ASCII (formato de 128 símbolos, que incluye letras y caracteres numéricos, espacios y caracteres básicos de control, etc.). Un mensaje está organizado en líneas (los renglones del mensaje) de hasta 80 caracteres, y almacenado internamente en una lista.
- Codificador de mensajes. El codificador de mensajes es la componente responsable de tomar un mensaje, y cifrarlo o codificarlo. Consta de dos partes principales: *(i)* el cómputo del código de encripción, y *(ii)* la encripción del mensaje de entrada utilizando el código generado. El código es un arreglo de enteros no negativos (dígitos, para ser precisos), y se calcula en base al contenido de la primera línea del mensaje a codificar. La encripción utiliza el código para sustituir caracteres del mensaje, por desplazamiento.
- Decodificador de mensajes. El decodificador de mensajes permite, dado un mensaje cifrado y el código usado para el cifrado, recomponer el mensaje original. Es decir, funciona de manera inversa al codificador de mensajes, aunque esta componente es más simple, dado que no computa el código de encripción, que es geneado por el codificador de mensajes.

Este proyecto requiere la implementación de algunos métodos específicos, concretamente dos métodos de la clase `Mensaje` (`agregarLinea(int pos, String linea)` y `boolean equals(Mensaje otro)`), un método de la clase `CodificadorMensajes` (`int[] generarCodigoEncripcion(String str)`), y finalmente dos métodos de la clase `DecodificadorMensajes` (`void decodificarMensaje()` y `String desencriptarCadena(String str, int[] codigo)`).

Estudie las clases y aproveche la estructura e ideas algorítmicas de los métodos ya implementados para implementar los restantes (algunos son similares). Sugerimos además comenzar con la clase `Mensaje`, seguir con `CodificadorMensajes`, y finalmente `DecofidicadorMensajes`, pero por supuesto puede elegir el orden de implementación que le resulte más conveniente.

Sobre el cifrado

El Cifrado César de clave múltiple suele denominarse *Cifrado de Vigenère*. Consiste en sustituir caracteres de un texto por los caracteres correspondientes a desplazarse la cantidad de caracteres (en orden lexicográfico) que indica el código respectivo. Tomemos por ejemplo la cadena `Viterbo`, y el código {`1, 2`}. El primer caracter se codificará con `1`, el segundo con `2`, el tercero con `1`, el cuarto con `2`, y así sucesivamente hasta agotar la cadena. Esto significa, por ejemplo, que el primer caracter será reemplazado por el que le sigue en la codificación ASCII (el siguiente a la 'V' es la 'W'), mientras que el segundo caracter se reemplazará por el caracter ubicado 2 lugares hacia la derecha (reemplazando 'i' por 'k'). La cadena encriptada será, entonces, `Wkugsdp`.

Puede analizar el código de los métodos ya implementados para ver cómo operar con caracteres y su respectivo código. Tenga en cuenta que el avance es *circular*; por ejemplo, el caracter siguiente al caracter con código 127 será el caracter con código 0.

Pruebas El proyecto contiene una clase diferente a las que hemos visto previamente. Esta clase se llama `MensajeTest` y BlueJ la destaca con un color diferente. Puede explorar esta clase, es fácil de entender. Contiene *pruebas*, o *tests*, que analizan si la implementación de los métodos de la clase `Mensaje` es correcta (al menos en esos casos de prueba). Por ejemplo, la clase `MensajeTest` contiene un método `agregarLineaValida()`, que crea un mensaje vacío, agrega una primera línea (`‘‘Hola, que tal’’`), y luego comprueba que el mensaje obtenido tiene una línea, y que satisface el invariante de `Mensaje` (`repOK()` retorna true). Para ejecutar las pruebas simplemente presione el botón *Ejecutar pruebas*, y BlueJ le informará cuántas y cuáles pruebas pasaron, y cuáles y cuántas fallaron. Al completar la implementación de `Mensaje`, todas las pruebas deberían pasar.

El uso de pruebas ayuda a analizar o probar nuestras implementaciones de manera más eficiente y efectiva (apretando sólo un botón), en comparación con hacerlo manualmente creando e invocando métodos de objetos interactivamente.