Programación

Objetivos a evaluar

Este trabajo evaluará su dominio de los siguientes objetivos:

1. Implementar una clase Java bien diseñada para cumplir con una especificación dada.
2. Aplicar los principios de *herencia* y *polimorfismo*, uno de los pilares de la POO
3. Mantener una abstracción adecuada entre el cliente y la implementación de una clase.
4. Seguir las convenciones prescritas para la calidad del Código, la documentación y lalegibilidad.

1. Inventario de letras

En este trabajo de programación POO practicar el uso de Arrays y clases. Debe implementar una clase llamada *InventarioLetras* que puede ser utilizada para mantener un inventario de letras del alfabeto Ingles o español. El constructor de la clase tomar como parámetro un*String* y calcular cuantas letras hay en ese *String*. Esta es la información que guarda el objeto (cuantas a's, , cuantas *b's*, etc.). *InventarioLetras* ignora cualquier carácter que nosea un carácter alfabético de idioma inglés (como caracteres de puntuación o dígitos, tildesy la ñ) o española (como caracteres de puntuación o dígitos); y trata las letras mayúsculas y minúsculas como si fueran iguales.

1.1. Clases Constructores y Métodos

Su clase *InventarioLetras* debe incluir el siguiente método constructor:

public InventarioLetras(String data)

Construye un inventario (un recuento) de las letras alfabéticas en el *String*, ignorando las mayúsculas y minúsculas y los caracteres no alfabéticos.

Su clase también debe incluir los siguientes métodos públicos.

`public char encriptarCesar(char letra)`

Método que encripta el *char* dado utilizando la estrategia del cifrado de Cesar.

El cifrado Cesar es un tipo de encriptación por sustitución en el que los caracteres de un alfabeto se sustituyen por caracteres generados al desplazar cada carácter del texto por un número fijo de posiciones hacia delante en el alfabeto (3 por defecto). El alfabeto por defecto es el conjunto de todas las letras minúsculas, pero debería poder modificar el alfabeto.

La siguiente tabla muestra cómo se cifra la palabra "play" utilizando este tipo de cifrado.

texto	**p**	**l**	**a**	**y**
+Shift	3	3	3	3
=Encriptado	s	o	d	b

Observe en que el cifrado "envuelve" el alfabeto según sea necesario. Por ejemplo, si se desplaza la "y" tres posiciones, se obtiene la "b". Aquí está el mapa completo de un cifrado Cesar con un desplazamiento de tres, con las letras en "play" resaltada:

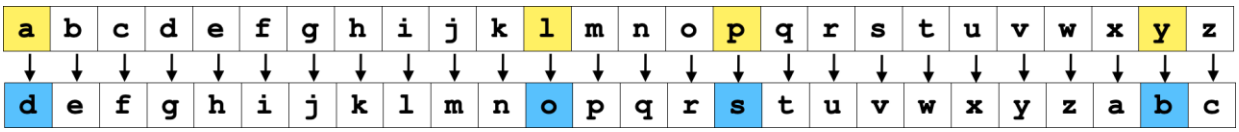

Figura 1: Cifrado de Cesar

`public void encriptarPalabra(Char letra, int valor)`

Método que utiliza el método *encriptarCesar*(*char letra*), que puede encriptar una sola letra, para implementar un método que pueda encriptar un *String* carácter a carácter.

`public void desencriptarPalabra(Char letra, int valor)`

Método que utiliza el método *desencriptarCesar*(*char letra*), que puede desencriptar una sola letra, para implementar un método que pueda desencriptar un *String* carácter a carácter.

public int get(char letra)

Devuelve un recuento de la cantidad de esta letra (que no distingue entre mayúsculas y minúsculas) que hay en el inventario. La letra puede ser minúscula o mayúscula (a su método no debería importarle). Si pasa un carácter que no está en el alfabeto (Ingles o español), su método debería lanzar una excepción ***IllegalArgumentException*.**

public void set(char letra, int valor)

Fija el recuento de la letra dada. La letra puede ser minúscula o mayúscula. Si se pasa un carácter no alfabético o si el valor es negativo, el método debe lanzar una excepción ***IllegalArgumentException*.**

public int size()

Devuelve la suma de todos los recuentos de este inventario. Esta operación debe ser "rápida" en el sentido de que debe almacenar el tamaño en algún lugar, en vez de tener que calcularlo cada vez que se llama a este método.

public boolean isEmpty()

Devuelve *True* si este inventario está vacío (es decir, todos los recuentos son 0). Esta operación debe ser "rápida" en el sentido que no deber a tener que examinar cada uno de los 26 o 27 recuentos cuando se llama.

public String toString()

Devuelve una representación de String del inventario con las letras en minúsculas en orden y entre corchetes. El número de apariciones de cada letra debe coincidir con la cuenta en el inventario. Por ejemplo, un inventario de 4 aes, 1 b, 1 l y 1 m se representar a como "[aaaablm]".

public InventarioLetras add(InventarioLetras otro)

Construye y devuelve un nuevo objeto *InventarioLetras* que representa la suma de este *InventarioLetras* y el otro *InventarioLetras* dado. Los recuentos de cada letra deben sumarse. Los dos objetos *InventarioLetras* que se suman (este y el otro) no deben ser modificados por este método.

public InventarioLetras amplifies(int n)

Construye y devuelve un nuevo objeto *InventarioLetras* que representa la multiplicación de este *InventarioLetras* por el valor de *n* dado. Los recuentos de cada letra deben multiplicarse. El objeto *InventarioLetras* que se amplifica no debe ser modificado por este método.

public InventarioLetras subtract(InventarioLetras otro)

Construye y devuelve un nuevo objeto *InventarioLetras* que representa el resultado de restar el otro inventario de este inventario (es decir, restando los recuentos del otro inventario de este objeto). Si el recuento resultante es negativo, este método debe devolver *null*. El segundo objeto *InventarioLetras* que se resta (este y otro) no deben ser modificados por este método.

También puede incluir cualquier otro método privado de ayuda que considere útil.
Como ejemplo, el método *add* podría ser llamado de la siguiente manera:

```
Inventario Letras inventorio1 = new Inventario Letras (" Alan Turing ");
InventarioLetras inventorio2 = new InventarioLetras("Ada Lovelace");
InventarioLetras sum = inventorio1.add(inventorio2);
```

Aquí, el inventario1 contendría [*aagilnnrtu*], el inventario2 contendría [*aaacdeellov*], y la suma contendría [*aaaaacdeegilllnnortuv*].

1. Estrategia para el desarrollo del Trabajo

Una de las técnicas más importantes para los profesionales del software es desarrollar el código por etapas en lugar de intentar escribirlo todo de una vez (el termino técnico es *mejoraiterativa* o *perfeccionamiento por etapas*). También es importante poder probar la corrección de la solución en cada etapa.

He observado que muchos de los estudiantes no desarrollan su código por etapas y no tienen una buena idea de cómo probar sus soluciones. Como resultado, para esta tarea se le proporcionara una estrategia de desarrollo y algo de código de pruebas.

Les sugiero que desarrollen el programa en tres etapas:

1. En esta etapa queremos probar la construcción de *InventarioLetras* y examinar su contenido. Así que los métodos que implementaremos son el constructor, el método *encriptarCesar*(*char letra*), el método *desencriptarCesar*(*char letra*), el método *size*, el método *isEmpty*, el método *get* y el método *toString*. Incluso dentro de esta etapa puedes desarrollar los métodos lentamente. Primero haga los métodos constructor y*size*. Después añada el método *isEmpty*. Después añada el método *get*. Después añadeel método *toString*. El programa de pruebas los probar en este orden, por lo que ser posible implementarlos de uno en uno.

2. En esta etapa queremos añadir el método *set* a la clase que permite al cliente cambiarel número de ocurrencias de una letra individual. El programa de pruebas verificaraque otros métodos funcionan correctamente junto con *set* (los m todos *get*, is*Empty*, *size* y *toString*, ...).
3. En esta etapa queremos incluir los métodos *add*, *subtract* y *amplifies*. Debería escribir primero el método *add* y asegurarse de que funciona. El programa de pruebas primero prueba *add*, así que no se preocupe por el hecho de que las pruebas sobre *subtract* y *amplifies* fallen inicialmente.

3. Restricciones sobre la entrega.

1. El proyecto debe implementarse utilizando el lenguaje Java
2. El código fuente debe utilizar nombres representativos de las variables y los métodos debidamente comentados.
3. Pueden crear métodos adicionales si los necesitan.