Explicad mínimamente el código de vuestra solución, especialmente en los métodos más complejos.

Problema 1 (7 puntos)

Apartado a: La clase `CharPair` (2 puntos)

Diseña e implementa una clase denominada `CharPair` para representar una pareja de caracteres.

- `new CharPair(char from, char to)`
  - constructor que crea una instancia de `CharPair` a partir de los dos caracteres de la pareja
- `int encode(char c)`
  - si el carácter `c` coincide con `from`, devuelve `to`; en otro caso, devuelve `-1`
- `boolean conflictsWith(CharPair other)`
  - una pareja tiene conflicto con otra si coinciden sus caracteres `from` o sus caracteres `to`
  - esta función devuelve `true` si la pareja referenciada por `other` tiene conflicto con el objeto receptor y `false` en caso contrario
- `CharPair invert()`
  - devuelve la pareja de caracteres consistente en intercambiar `from` y `to` en la pareja correspondiente al objeto receptor
  - se devuelve una instancia nueva; **en ningún momento se modifica el objeto receptor**

Fijaos en que las instancias de la clase `CharPair` son inmutables: una vez creadas, no hay ningún método de la clase que las modifique.

Apartado b: La clase `Cypher` (5 puntos)

La clase `Cypher` servirá para definir una clave de cifrado que se construirá en base a añadir parejas de caracteres.

Dicha colección de parejas será extensible, es decir, el número de parejas que se podrán almacenar no estará limitado.

- `new Cypher(int initialSize)`
  - tenemos garantizado que `initialSize` es positivo
  - se crea una clave de cifrado con tamaño para `initialSize` parejas de caracteres

- `boolean addPair(char from, char to)`
    - si la pareja formada por los caracteres dados causa conflicto con alguna de las parejas de la clave, devuelve `false` y no hace nada
    - si no hay conflictos, añade la pareja formada por esos caracteres a la clave de cifrado y devolverá `true`
    - en caso de necesitar más espacio para añadir la nueva pareja, doblará el tamaño actual
- `Cypher invert()`
    - si la clave de cifrado actual no tiene claves, devolverá `null`
    - devuelve la clave de cifrado invertida, es decir, la que servirá para decodificar un mensaje previamente codificado
    - el tamaño de dicha clave (su número de parejas) será el mínimo necesario para guardar todas las parejas de la clave del objeto receptor
    - se devuelve una nueva instancia de `Cypher`, no se modifica la instancia correspondiente al objeto receptor
- `String encode(String input)`
    - codifica la cadena usando las parejas guardadas en la clave de cifrado
    - un espacio en la entrada se codifica siempre como un espacio en la salida sin tener que usar la clave de cifrado
    - si alguno de los caracteres de `input` no puede cifrarse, devuelve `null`
    - `PISTA:` definid una función auxiliar para codificar un sólo carácter.

Como siempre, definid funciones auxiliares para que el código sea entendible y manejable.

Problema 2 (3 puntos)

El programa principal pedirá la clave de cifrado y la frase a cifrar de la siguiente manera:

```
Clave: a>b b>z b>u i>z p>o r>t t>z t>c v>f
Frase: patata brava
```

El programa retornará el número de parejas que han dado error al crear la clave (las que creaban conflictos con parejas anteriores) y, en caso de que la frase pueda codificarse, su codificación (o el hecho de que no ha podido codificarse). Podéis suponer, como siempre, que los elementos de la clave están bien formados (tenemos garantizado que siempre son dos caracteres separados por `>`).

En el ejemplo anterior se han resaltado en negrita las parejas que provocan errores y se ha subrayado el carácter de la pareja responsable de ello.

Para la ejecución anterior:

```
Errores: 3
Cifrado: obcbcb ztbfb
```

Recordad que, para simplificar el programa principal, haremos que tanto la clave de cifrado como el número de errores producidos sean variables de instancia de la clase del programa principal (y así serán accesibles desde cualquier método del mismo), es decir:

```
public class CypherProgram extended CommandLineProgram {

    private static final int INITIAL_SIZE = 28;
    private Cypher cypher;
    private int numErrors;

    private void initializeResults() {
        this.cypher = new Cypher(INITIAL_SIZE);
        this.numErrors = 0;
    }

    public void run() {
        this.initializeResults();
        ???
    }

    ???
}
```

Como siempre, definid funciones auxiliares para que el código sea entendible y manejable.

- class **CommandLineProgram**
  - String **readLine**(String message)
  - void **println**(String str)/ void **println**(int n) / void **println**(double d)
  - void **print**(String str) / void **print**(int n) / void **print**(double d)
- class **String**:
  - new **String**(char[] chars, int offset, int length)
  - char **charAt**(int index)
  - int **length**()
  - boolean **equals**(String other) / boolean **equalsIgnoreCase**(String other)
  - int **compareTo**(String other)
  - int **indexOf**(char c) / int **indexOf**(String s)
  - String **substring**(int p1, int p2) / String **substring**(int p1)
  - String **concat**(String s) / o usar **+** para concatenar Strings
  - static String **valueOf**(int n)
  - static String **valueOf**(double d)
- class **StringTokenizer**:
  - new **StringTokenizer**(String str)
  - new **StringTokenizer**(String str, String delims)
  - new **StringTokenizer**(String str, String delims, boolean returnDelims)
  - boolean **hasMoreTokens**()
  - String **nextToken**()
- class **Math**
  - static int **max**(int n1, int n2) / static int **min**(int n1, int n2)
  - static double **max**(double d1, double d2)
  - static double **min**(double d1, double d2)
- char c = (char) n; // para int n