Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Diseño de Software

Profesora Ing Ericka Solano Fernández

Proyecto 1

Justificación de los Principios Utilizados

Integrantes

Sebastián Salas Garcia 2015183511

Camila Víquez Alpízar 2014152891

Leonardo Mata Mora 2016081321

14 de Agosto del 2018

TEC | Tecnológico de Costa Rica

Tabla de Contenidos

Resumen del propósito de cada Principio utilizado.

El presente documento describe cada uno de los principios de diseño de software implementados para el desarrollo de la primera etapa de este proyecto. No se incluirá la explicación de ningún patrón de diseño dentro de este resumen ya que estos no fueron implementados para el desarrollo de esta primer etapa. Los principios con los que analizaremos nuestro código serán con SOLID. A continuación, se describen cada uno con más detalle.

- **Responsabilidad Única:** Este principio se cumple con las clases implementadas para el desarrollo del proyecto, cada clase tiene una única funcionalidad, por ejemplo.

Clase	Implementación
Alfabeto	Encargada de todos los objetos alfabetos, en ella se almacenan los datos propios de un alfabeto tales como el identificador, el nombre y los símbolos que lo conforma. Y dentro de sus métodos podemos validar si un alfabeto es correcto o no según lo que el usuario seleccione. Es importante mencionar que para esta fase el usuario solo puede seleccionar el alfabeto por default, pero se diseñó para que haya más multiplicidad.
gestorAlfabeto	Encargada de realizar todas las funcionalidades relacionadas al CRUD de cada alfabeto. Dentro de

	esta podemos crear, consultar, eliminar o editar más alfabetos.
gestorAlgoritmo	Tiene la funcionalidad de manejar los métodos abstractos correspondientes a la codificación, a la decodificación y nada más.
DTOTraduccion	Cumple con la única funcionalidad de almacenar las variables necesarias para la gestión de la traducción
Guardar archivos	Cada clase asignada para el almacenamiento de las traducciones en archivos cumple con un una única responsabilidad que consiste en guardar la información necesaria.

En cuanto a la clase Controlador está no cumple con el principio de responsabilidad única ya que lleva a cabo varias funcionalidades. Se podría pensar en separar las funcionalidades en otra clase y así no sobrecargar el controlador por lo que se deja abierto los cambios a futuro.

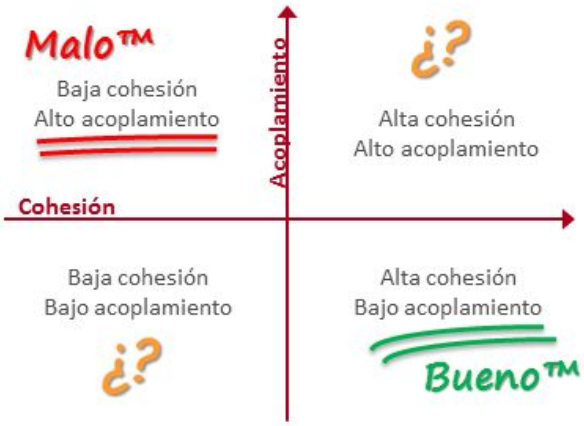
- **Abierto cerrado:** Este principio se cumple mediante la implementación de herencia. Se tiene la superclase llamada “Gestor Algoritmo” la cual extiende a las clases hijas que representan una forma de codificación o decodificación. Esta herencia permite que el código sea abierto a la extensibilidad, por ejemplo si en un futuro se quiere implementar un nuevo método de codificación o decodificación simplemente se debe agregar como una clase hija e implementar los métodos abstractos que tiene esta. Además, con el uso de interfaces en los mecanismos de guardado de la información nos da

esa flexibilidad de poder agregar otras manera de guardado como por ejemplo .xlsx. Cumpliéndose que está abierto a la extensión y en cierta medida con clases básicas cerradas a la modificación pues con las clases grandes ocuparían aún se modificadas en caso de querer más funcionalidades, a sabiendas de que no sería tanto problema.

- **Segregación de interfaces:** Este principio se implementa mediante el uso de dos interfaces, una cumple con la función de validar los alfabetos y la otra cumple con la función de escribir los resultados de las traducciones en los archivos. Así los métodos correspondientes a estas interfaces no necesitan estar dentro de las clases, esto se hace con el fin de que las clases no tengan métodos innecesarios. Es importante resaltar que cada interfaz cumple su función específica y evitamos “ensanchar” sus tareas con el fin de cumplir con el bajo acoplamiento y alta cohesión.

- **Inversión de dependencias:** Este principio se cumple al implementar una clase intermedia que cumple con la comunicación entre clases, el DTO no es propiamente una clase de cierta manera, es un objeto, que se encarga de comunicar los datos de las clases pertenecientes a la capa de vista con los clases del modelo y del negocio encapsulando los datos para un mejor traslado de la información.

- **Alta cohesión y bajo acople**: Con la implementación del patrón de arquitectura MVC se logra parte de este principio. Este patrón tiene el fin de separar los datos de la aplicación de la interfaz y la lógica de control. Así se logra modelar la aplicación es tres grandes componentes. Al usar este patrón se tiene una alta cohesión y un bajo acople ya que cada paquete contiene clases que están relacionadas y a su vez existe una comunicación entre los paquetes.

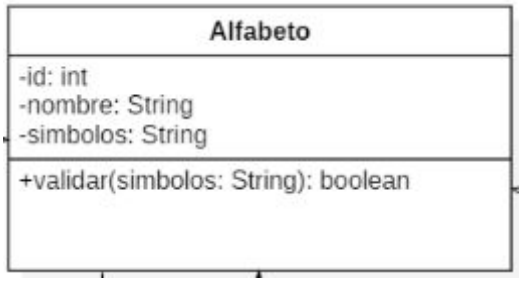


Relación Diagrama Principio

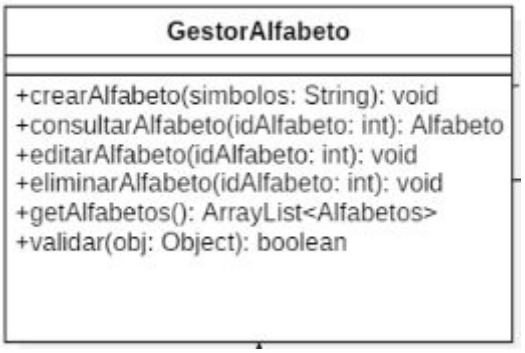
A continuación, se presenta una serie de tomas sobre partes específicas del diagrama UML, esto con el fin de representar cada una de las implementaciones de las clases que cumplen con un principio SOLID mencionados en la sección anterior.

- **Principio de responsabilidad unica:**

  Alfabeto



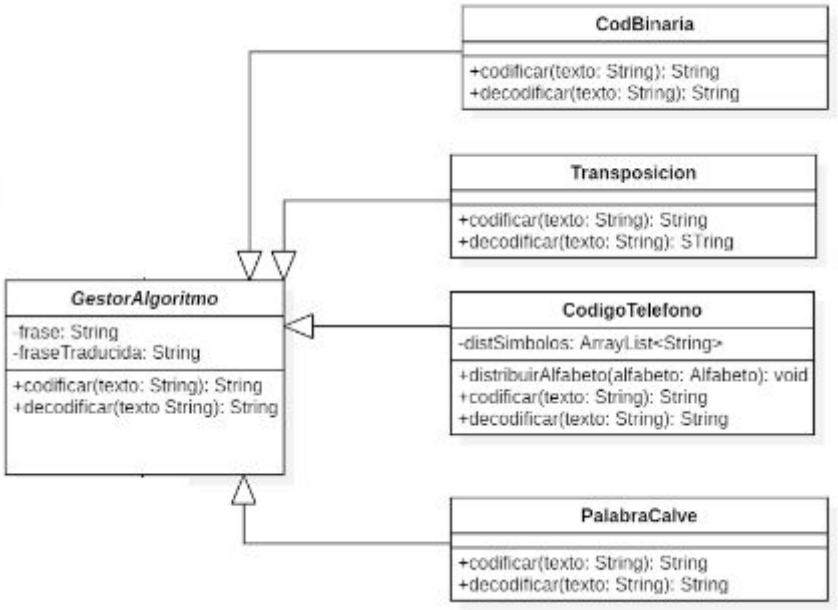
  GestorAlfabeto.



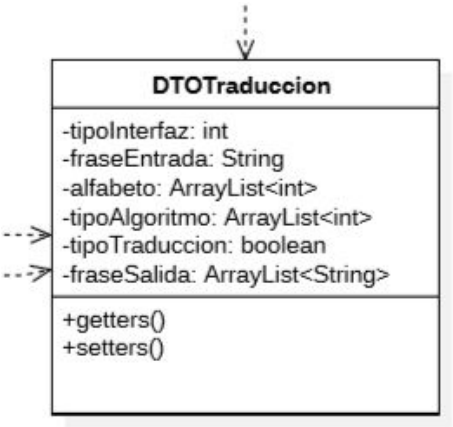
DTOTraduccion

Las clases utilizadas para la implementación de los métodos de guardado también cumplen con este principio.

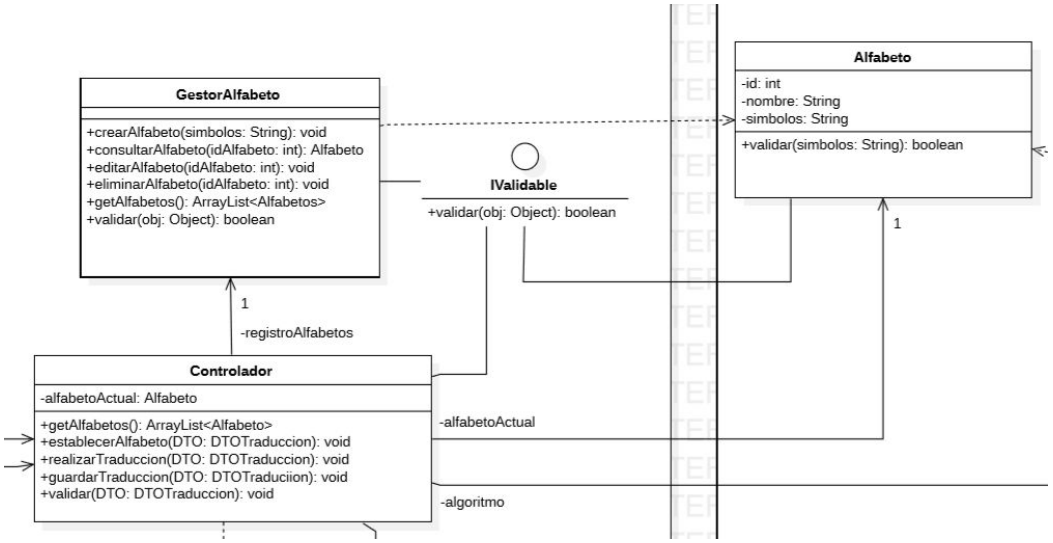
- **Principio abierto cerrado:**



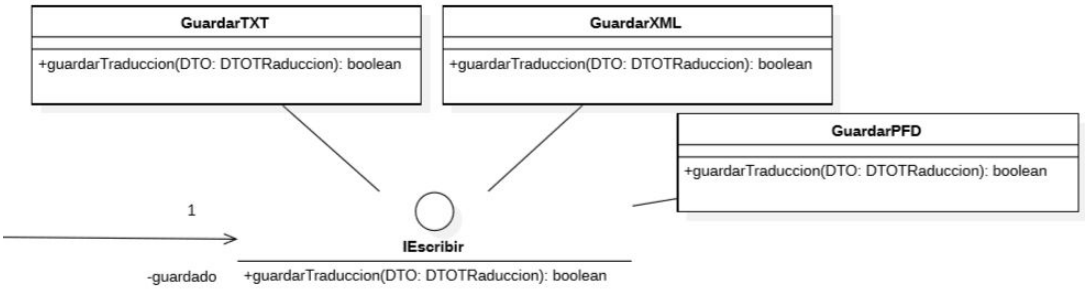
- **Inversión de dependencias:** Principio implementado en el diagrama mediante el objeto: DTO Data transfer object



- **Segregación de interfaces:**
  - Interfaz IValidar



  - Interfaz Escribir



Segmentos de código según Principio

El siguiente segmento se presentan bloques de código de una clase o un conjunto de clases que hacen posible la implementación de algún principio.

- Principio de responsabilidad única:

  Clase Alfabeto

```java
public class Alfabeto {

    private int id = 0;
    private String nombre;
    private String simbolos;

    public Alfabeto(int id, String nombre, String simbolos) {
        this.id = id;
        this.nombre = nombre;
        this.simbolos = simbolos;
    }

    //Gettters y Setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getSimbolos() {
        return simbolos;
    }

    public void setSimbolos(String simbolos) {
        this.simbolos = simbolos;
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Alfabeto other = (Alfabeto) obj;
        if (!Objects.equals(this.id, other.id)) {
            return false;
        }
        if (!Objects.equals(this.nombre, other.nombre)) {
            return false;
        }
        if (!Objects.equals(this.simbolos, other.simbolos)) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Alfabeto{" +
                "id=" + id +
                ", nombre='" + nombre + '\'' +
                ", simbolos='" + simbolos + '\'' +
                '}';
    }
}
```

Clase gestorAlfabeto

```
package Negocios;

import Modelo.Alfabeto;

import java.util.ArrayList;
import java.util.List;

public class GestorAlfabeto implements IValidable{

    protected List datos = new ArrayList<Alfabeto>();

    private void crearAlfabeto(Object obj){
        Alfabeto unUsr = (Alfabeto) obj;
        datos.add(unUsr);
    }

    public Object consultarAlfabeto( Object alfabeto){
        if (datos.indexOf(alfabeto)!= -1)
            return datos.get(datos.indexOf(alfabeto));
        else
            return null;
    }

    public boolean editarAlfabeto(Object alfa){
        if (datos.indexOf(alfa)!= -1){
            datos.set( datos.indexOf(alfa), alfa);
            return true;
        }
        else
            return false;
    }

    public boolean eliminarAlfabeto(Object alfa){
        return datos.remove(alfa);
    }
```

```
    @Override
    public boolean validar(Object obj) {
        Alfabeto alfa= (Alfabeto) obj;
        Alfabeto elAlfa = (Alfabeto) consultarAlfabeto(alfa);
        return alfa.equals(elAlfa);
    }
}
```

Clase gestorAlgoritmos

```
package Modelo;
import Negocios.DTOTraduccion;
import java.util.ArrayList;
public abstract class GestorAlgoritmo {
    private String frase;
    private String fraseTraducida;

    abstract public String codificar(String capsula);
    abstract public String decodificar(String capsula);
}
```

Clase DTOTraduccion

```
    }

    public DTOTraduccion(int tipoInterfaz, int tipoTraduccion,
String fraseEntrada,
                         int[] tipoAlgoritmos, int[]
metodoGuardado, int[] tipoAlfabeto) {
        this.tipoInterfaz = tipoInterfaz;
        this.tipoTraduccion = tipoTraduccion;
        this.fraseEntrada = fraseEntrada;
        this.tipoAlgoritmos = tipoAlgoritmos;
        this.metodoGuardado = metodoGuardado;
        this.tipoAlfabeto = tipoAlfabeto;
    }

    //Getters y Setters

    @Override
```

```
    public String toString() {
        return "DTOTraduccion{" +
                "tipoInterfaz=" + tipoInterfaz +
                ", tipoTraduccion=" + tipoTraduccion +
                ", fraseEntrada='" + fraseEntrada + '\'' +
                ", fraseSalida=" + fraseSalida +
                ", tipoAlgoritmos=" +
Arrays.toString(tipoAlgoritmos) +
                ", metodoGuardado=" +
Arrays.toString(metodoGuardado) +
                ", tipoAlfabeto=" + Arrays.toString(tipoAlfabeto)
+
                '}';
    }
```

- Principio abierto cerrado:

Super Clase

```
public abstract class GestorAlgoritmo {
    private String frase;
    private String fraseTraducida;

    abstract public String codificar(String texto);
    abstract public String decodificar( String texto);
}
```

Clases Hijas

```
public class Binario extends GestorAlgoritmo {
    public String codificar (String texto )
    {
        return "Falta implementacion algoritmo Codigo Binario ";
    }


    public String decodificar(String texto)
    {
        return "Falta implementacion algoritmo Codigo Binario  ";
    }
}
```

- Principio de segregación de interfaces:

```
public interface IEscribir {
   public boolean guardarTraduccion(DTOTraduccion texto);

}

public interface IValidable {
    public boolean validar();

}
```

- Principio de inversión de dependencias:

```
public class DTOTraduccion {
    private int tipoInterfaz ;
    private int tipoTraduccion ;
    private String fraseEntrada ;
    private ArrayList<String> fraseSalida ;
    private int[] tipoAlgoritmos ;
    private int[] metodoGuardado ;
    private int[] tipoAlfabeto ;

public DTOTraduccion(int tipoInterfaz, int tipoTraduccion, String
fraseEntrada,ArrayList<String> fraseSalida, int[]
tipoAlgoritmos,int[] metodoGuardado, int[] tipoAlfabeto)
{
        this.tipoInterfaz = tipoInterfaz;
        this.tipoTraduccion = tipoTraduccion;
        this.fraseEntrada = fraseEntrada;
        this.fraseSalida = fraseSalida;
        this.tipoAlgoritmos = tipoAlgoritmos;
        this.metodoGuardado = metodoGuardado;
        this.tipoAlfabeto = tipoAlfabeto;
    }

public DTOTraduccion(int tipoInterfaz, int tipoTraduccion, String
fraseEntrada,int[] tipoAlgoritmos, int[] metodoGuardado, int[]
tipoAlfabeto)
{
        this.tipoInterfaz = tipoInterfaz;
        this.tipoTraduccion = tipoTraduccion;
        this.fraseEntrada = fraseEntrada;
```

```
        this.tipoAlgoritmos = tipoAlgoritmos;
        this.metodoGuardado = metodoGuardado;
        this.tipoAlfabeto = tipoAlfabeto;
```

Estrategias para el Cumplimiento del Principio

A continuación se presenta un cuadro donde se explican los principios SOLID implementados en la primera fase de desarrollo de este proyecto, en la tabla también se describe la estrategia que el equipo implementará para lograr el cumpliminto de dicho principio.

Principio	Estrategias
Responsabilidad única	Seguir el modelo UML, creado por el equipo, donde cada clase fue diseñada con el propósito de tener una responsabilidad única.
Abierto cerrado	Las clases están diseñadas de manera que cuando se consulte un objeto solo se pueda extraer información y no se permita que el objeto sea modificado por otras clases. La herencia de la clase gestor algoritmo, se realizó pensando en que el futuro se puedan incluir otros métodos de codificación, entonces estos métodos heredarán de la clase gestor algoritmo y no será necesario modificar las demás clases.
Segregación de la interfaz	Se pretende seguir el modelo en el cual se han implementado las interfaces con los métodos necesarios, de esta

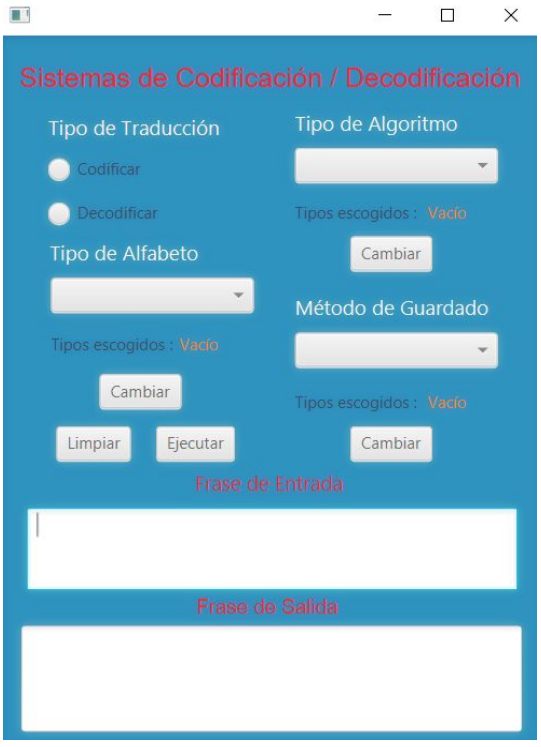
	forma al crear las clases que las implementan se crearán las clases con los únicos métodos que verdaderamente usan.
Inversión de la dependencia	Se pretende implementar una clase que transfiera los datos obtenidos en la vista y los transporte hasta las clases que necesitan hacer uso de estos, talos como la clase controlado.

Navegación a través de la Arquitectura

A continuación se presenta una secuencia de capturas de pantalla que tienen la finalidad de representar el flujo dentro del diseño implementado para el proyecto. Esta representación se da a nivel de consola y a nivel de GUI.

- Navegación a través de GUI

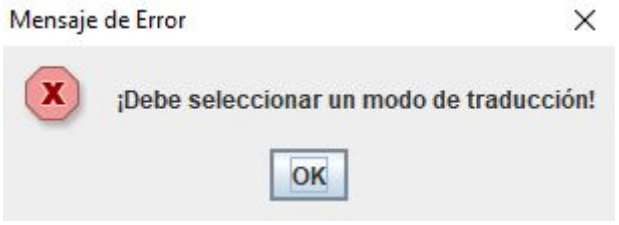
1. Esta captura representa la pantalla principal, en la cual el usuario debe seleccionar uno o varios alfabetos, debe indicar si quiere realizar un codificacion o una decodificación y seguidamente el método a utilizar.

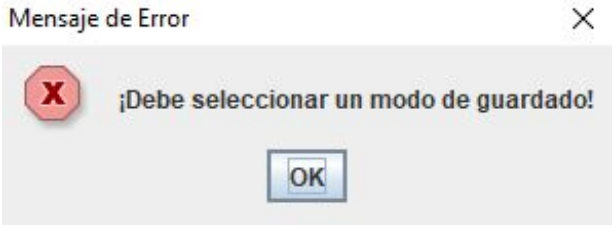


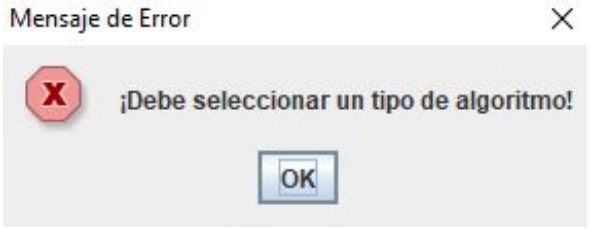
2. En esta captura se muestra la selección realizada por el usuario y el mensaje que desea codificar, con el método llamado código telefónico, desea guardarlo en formato TXT.

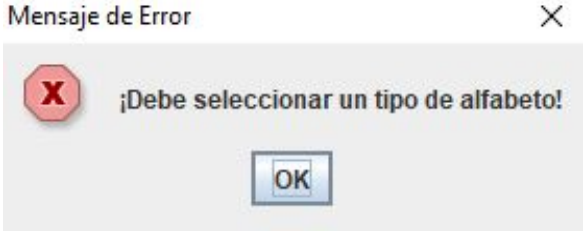
   La frase de salida no es indicada ya que los métodos de codificación y decodificación se encuentran en construcción para esta fase del proyecto.

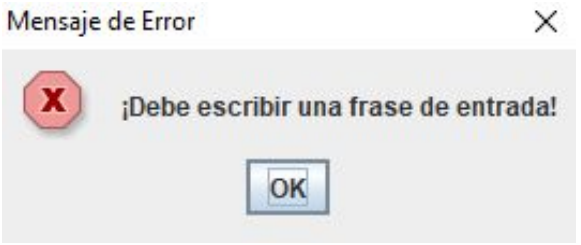
3. Pantallas de error. Si al usuario le faltan datos para poder correr el sistema se presentarán las siguientes ventanas:











- **Navegación a través de Consola**

  La captura representa la consola solicitando los datos necesarios para procesar los algoritmos, en la cual el usuario debe seleccionar uno o varios alfabetos, debe indicar si quiere realizar un codificacion o una decodificación y seguidamente el método a utilizar.

```
Sistemas de Codificación / Decodificación

Ingrese el Tipo de Traducción deseada [ Codificar ~ 0 ,Decodificar ~ 1 ]:
1
Ingrese la frase a traduccir:
Hola como estas
Ingrese los tipos de algoritmos :
1. Sustitución Vigenére
2. Transposición (Letra a letra)
3. Código Telefónico
4. Codificación Binaria
5. Palabra Clave
Nota: Formato para escoger #,#,#,#,#
3
Ingrese los métodos de guardado :
1. TXT
2. XML
3. PDF
Nota: Formato para escoger #,#,#
3
Ingrese el tipo de alfabeto :
1. Default
Nota: Formato para escoger #
1

Inicio de la traducción, por favor espere!
Procesando instrcciones...
```

Además en caso de error saldrán mensajes directamente en la consola que muestran al usuario su error y deben volver a intentarlo.

```
Sistemas de Codificación / Decodificación

Ingrese el Tipo de Traducción deseada [ Codificar ~ 0 ,Decodificar ~ 1 ]
4
¡Error de tipo de traducción, vuelva a digitarlo!
Ingrese el Tipo de Traducción deseada [ Codificar ~ 0 ,Decodificar ~ 1 ]
1
Ingrese la frase a traduccir:

¡Error de la frase de entrada, vuelva a digitarlo!
Ingrese la frase a traduccir:
Hola
Ingrese los tipos de algoritmos :
1. Sustitución Vigenére
2. Transposición (Letra a letra)
3. Código Telefónico
4. Codificación Binaria
5. Palabra Clave
Nota: Formato para escoger #,#,#,#,#
2
Ingrese los métodos de guardado :
1. TXT
2. XML
3. PDF
Nota: Formato para escoger #,#,#
1
Ingrese el tipo de alfabeto :
1. Default
Nota: Formato para escoger #

¡Error, el tipo de alfabeto es incorrectos, vuelva a digitarlo!
```

Por último se presenta unas capturas de pantalla que representa los mensajes encontrados dentro de cada clase, para hacer constar el flujo de datos que se da en el programa. Tanto para consola como para GUI.

```
Creo controlador
Clase controlador, llamada de algoritmos de codificacion y decodificacion
Validacion de frase ingresada por el usuario, gestorAlfabeto
Codificacion con el metodo vigenere:
Se agrega la traduccion a la frase de salida en el DTO
Falta implementacion del metodo para escritura de los archivos
FRASE salida: [Falta implementacion algoritmo Codigo Vigenere ]
Creo controlador
Clase controlador, llamada de algoritmos de codificacion y decodificacion
Validacion de frase ingresada por el usuario, gestorAlfabeto
Codificacion con el metodo Transposicion:
Traigo lo de implementar:Falta implementacion algoritmo  transposicion
Se agrega la traduccion a la frase de salida en el DTO
Falta implementacion del metodo para escritura de los archivos
FRASE salida: [Falta implementacion algoritmo  transposicion ]
Creo controlador
Clase controlador, llamada de algoritmos de codificacion y decodificacion
Validacion de frase ingresada por el usuario, gestorAlfabeto
Codificacion con el metodo Codigo Telefonico:
Se agrega la traduccion a la frase de salida en el DTO
Falta implementacion del metodo para escritura de los archivos
```

```
Creo controlador
Clase controlador, llamada de algoritmos de codificacion y decodificacion
Validacion de frase ingresada por el usuario, gestorAlfabeto
Codificacion con el metodo Codigo binario:
Se agrega la traduccion a la frase de salida en el DTO
Falta implementacion del metodo para escritura de los archivos
FRASE salida: [Falta implementacion algoritmo Codigo Binario ]
Creo controlador
Clase controlador, llamada de algoritmos de codificacion y decodificacion
Validacion de frase ingresada por el usuario, gestorAlfabeto
Codificacion con el metodo Codigo PalabraClave:
Se agrega la traduccion a la frase de salida en el DTO
Falta implementacion del metodo para escritura de los archivos
FRASE salida: [Falta implementacion algoritmo Palabra Clave ]
```