	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): _____

Asignatura: _____

Grupo: _____

No de Práctica(s): _____

Integrante(s): 321172974

_____

_____

_____

No. de lista o brigada: _____

Semestre: _____

Fecha de entrega: _____

Observaciones: _____

_____

CALIFICACIÓN: _____

ÍNDICE

Introducción

Planteamiento del problema

Diseñar e implementar un programa que reciba una cadena como entrada y lance una excepción si la cadena no contiene letras vocales. Dicha excepción debe ser personalizada y definida por el programador, con las herramientas que le proporciona el lenguaje Java.

Motivación

Al momento de generar programas, soluciones a problemas, se deben considerar los casos en los que el usuario que utiliza dicha aplicación (cliente) comete acciones inesperadas para el sistema, esto es, acciones que no siguen la lógica del programa pero que naturalmente surgen por la naturaleza curiosa del humano. Para poder atrapar la curiosidad desesperada del cliente por romper el programa, el diseñador del sistema y el programador deben encargarse de manejar toda acción inesperada y manejarla como error controlado, para así no romper con la secuencia del programas y al contrario, tener un registro de dichas excepciones.

Objetivos

Para tener los conocimientos necesarios para dar solución, se debe practicar el diseño e implementación de excepciones personalizadas en java, mediante la realización de 3 ejercicios.

Para el **problema planteado**:

- ☑ Se deben usar excepciones personalizadas, además de encapsular cada parte del programa, definirlo en un paquete, generar jar y javadocs
- ☑ Adicionalmente, se deben generar **diagramas estáticos** y **dinámicos UML** para la documentación del sistema

Marco Teórico

Excepciones

Excepción. "Una excepción es la indicación de un problema que ocurre durante la ejecución de un programa. [...] El manejo de excepciones le permite crear aplicaciones que puedan resolver (o manejar) las excepciones. En muchos casos, el manejo de una excepción permite que el programa continúe su ejecución como si no se hubiera encontrado el problema." **[1, p.579]**

La palabra **excepción** hace alusión a la situación opuesta al funcionamiento correcto y usual, por lo que una excepción se espera sea algo que ocurre con poca frecuencia.

El manejo de una excepción se puede entender como:

```
Realizar una tarea
Si la tarea anterior no se ejecutó correctamente
     Realizar un procesamiento de errores
Si la tarea anterior se ejecutó correctamente, sigue la secuencia normal
```

En términos simples, se intenta una porción de código, si esta produce algún error, entonces se manejan los errores y ya no se hace lo que iba en la secuencia original, pues el error puede representar que alguna acción produzca más errores.
En el manejo de errores, se decide que hacer de acuerdo al tipo de error.

La clase base de todas las excepciones en Java es `Throwable`, que tiene dos subclases principales: **[1]**

`Error`: Representa problemas graves, como fallos del sistema, que normalmente no pueden ser manejados por el programa.
`Exception`: Representa problemas que el programa puede manejar, como errores de entrada/salida o divisiones por cero

Bloque `try-catch`

El bloque try-catch es la estructura principal para manejar excepciones en Java.
Consiste en:

`try`: Contiene el código que puede lanzar una excepción.

`catch`: Permite capturar y manejar la excepción lanzada en el bloque try.
`finally` (opcional): Contiene código que se ejecuta siempre, independientemente de si ocurrió una excepción o no

La **jerarquía de excepciones** permite manejar múltiples tipos de excepciones en un solo bloque `try-catch` con diferentes cláusulas catch. Java selecciona la primera cláusula compatible con el tipo de excepción lanzada. Es importante manejar las excepciones más específicas primero, ya que las más generales podrían capturarlas y evitar que las específicas sean evaluadas **[1]**

Diagramas UML

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) es un estándar ampliamente utilizado para especificar, visualizar, construir y documentar los componentes de un sistema de software. UML se divide en diagramas estáticos y dinámicos, cada uno ofreciendo perspectivas específicas de un sistema. **[2, p. 6]**

Diagramas Estáticos

Los diagramas estáticos describen los aspectos estructurales de un sistema, enfocándose en los elementos y sus relaciones. **[2, p. 4]**

Diagramas de Casos de Uso

Los diagramas de casos de uso modelan las interacciones entre los actores (usuarios u otros sistemas) y el sistema bajo análisis. Representan los requisitos funcionales del sistema, detallando cómo los actores interactúan con el sistema para lograr un objetivo específico

Diagramas de Clases

Estos diagramas muestran las clases del sistema y las relaciones entre ellas. Representan atributos, métodos y visibilidad, y son fundamentales para la programación orientada a objetos. Las relaciones incluyen asociaciones, generalizaciones y agregaciones

Diagramas de Objetos

Similares a los diagramas de clases, los diagramas de objetos se centran en instancias específicas de las clases y sus relaciones en un momento dado. Son útiles para visualizar el estado del sistema en un punto específico de tiempo

[2, pp. 14-28], [2, pp. 103-111]

Diagramas Dinámicos

Los diagramas dinámicos describen el comportamiento del sistema a lo largo del tiempo, mostrando cómo interactúan los elementos en respuesta a eventos.

Diagramas de Estados

Modelan el ciclo de vida de un objeto, detallando los estados por los que pasa y las transiciones entre ellos en respuesta a eventos específicos. Los diagramas de estados son fundamentales en sistemas que dependen de eventos y condiciones

Diagramas de Actividades

Se centran en el flujo de control y la lógica de procesos, ilustrando cómo se ejecutan las actividades dentro del sistema. Son útiles para modelar procesos secuenciales y paralelos

Diagramas de Interacción

Se dividen en dos tipos principales:

Diagramas de Secuencia: Muestran cómo los objetos interactúan en una secuencia temporal específica. Son útiles para modelar escenarios detallados basados en casos de uso

Diagramas de Comunicación: Enfatizan las relaciones estructurales entre objetos más que el orden temporal de los mensajes. Complementan a los diagramas de secuencia al resaltar las dependencias de comunicación

[2, pp. 133-139]

[2, pp. 71-89]

Objeto String y sus métodos

`String` es un objeto, por lo que al ser un objeto tiene algunos métodos útiles que nos sirvieron durante la práctica.

`equalsToIgnoreCase(String otraCadena)`	Compara dos cadenas ignorando las diferencias entre mayúsculas y minúsculas. Utilizado únicamente para saber si una cadena era igual a "-1" ***[3]***

Desarrollo

Ejercicios en clase

Ejercicio0	
Planteamiento	**Solución**
Programa para definir una excepción creada por el programador para poder capturar una división entre cero	Se crea una clase que extiende de Exception, aquí su inicialización como constructor invoca con super mandandole el mensaje de impresión de la excepción. Se crea la clase OperacionMatematica, en ella el método dividir donde se específica mediante la palabra reservada throws que dicho método arrojará una excepción del tipo DivisionPorCeroException. Dentro del método se controla cuando se arroja dicha excepción. Usando el bloque try-catch en el método principal, se intenta hacer la división con el método dividir, de tal forma que el bloque try intentará hacer dicha acción, en caso de encontrar que se arroja una excepción la manejará con catch, en donde se especifica el tipo y se solicita que se imprima el mensaje del error.

Ejercicio1	
Planteamiento	**Solución**
Programa para definir una excepción	Se crea una clase RaizNegativaException

personalizada que permita capturar una raíz cuadrada de un valor negativo	que extiende de Exception. En su constructor, se invoca al constructor de la superclase con super, pasando el mensaje de error que se imprimirá cuando la excepción sea lanzada. En la clase OperacionMatematica, se define el método raizCuadrada, que utiliza la palabra reservada throws para indicar que puede lanzar una excepción de tipo RaizNegativaException. Dentro del método, se valida si el número recibido como argumento es negativo; en caso de serlo, se lanza una excepción con un mensaje indicando que no se puede calcular la raíz cuadrada de un número negativo. Si el número es válido, el método retorna el resultado de la operación matemática. En el método principal se crea una instancia de OperacionMatematica y se llama al método raizCuadrada dentro de un bloque try. Si se lanza una excepción, el bloque catch la captura y maneja, imprimiendo el mensaje de error utilizando System.err.

`Ejercicio2`	
Planteamiento	**Solución**
Programa para definir una excepción personalizada que permita identificar dada una lista si hay números duplicados	Se implementa una clase NumeroDuplicadoException, la cual extiende de Exception. El constructor de esta clase recibe un mensaje que es enviado al constructor de la superclase con super, permitiendo personalizar el mensaje de error que se mostrará al lanzar la excepción. La clase IngresaUsuario contiene el método leerNumerosUsuario, que solicita al usuario ingresar una cantidad determinada de

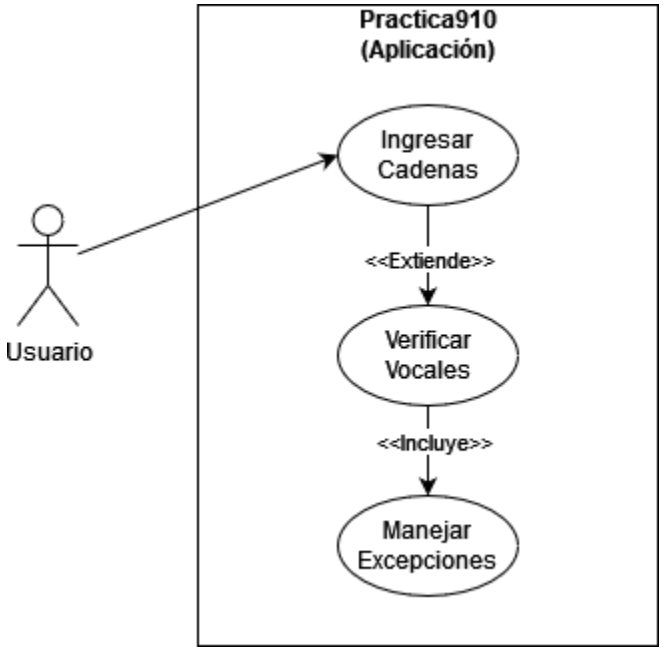
	números enteros a través de la consola. Estos números son almacenados en una lista que se retorna al final del método. La clase RevisionDuplicado tiene el método checarDuplicado, que verifica si existen números duplicados en una lista. Internamente, utiliza un conjunto (Set) para identificar duplicados. Si encuentra un número repetido, lanza una excepción NumeroDuplicadoException con un mensaje personalizado. En el método principal primero se llama al método leerNumerosUsuario para obtener una lista de números ingresados por el usuario. Después, se pasa esta lista al método checarDuplicado, que valida si existen duplicados. Si no se encuentran números repetidos, se imprime un mensaje de éxito. En caso de que ocurra una excepción, el bloque catch la captura e imprime el mensaje de error correspondiente

Ejercicios de Tarea

Diagramas UML[1]

Diagramas Estáticos

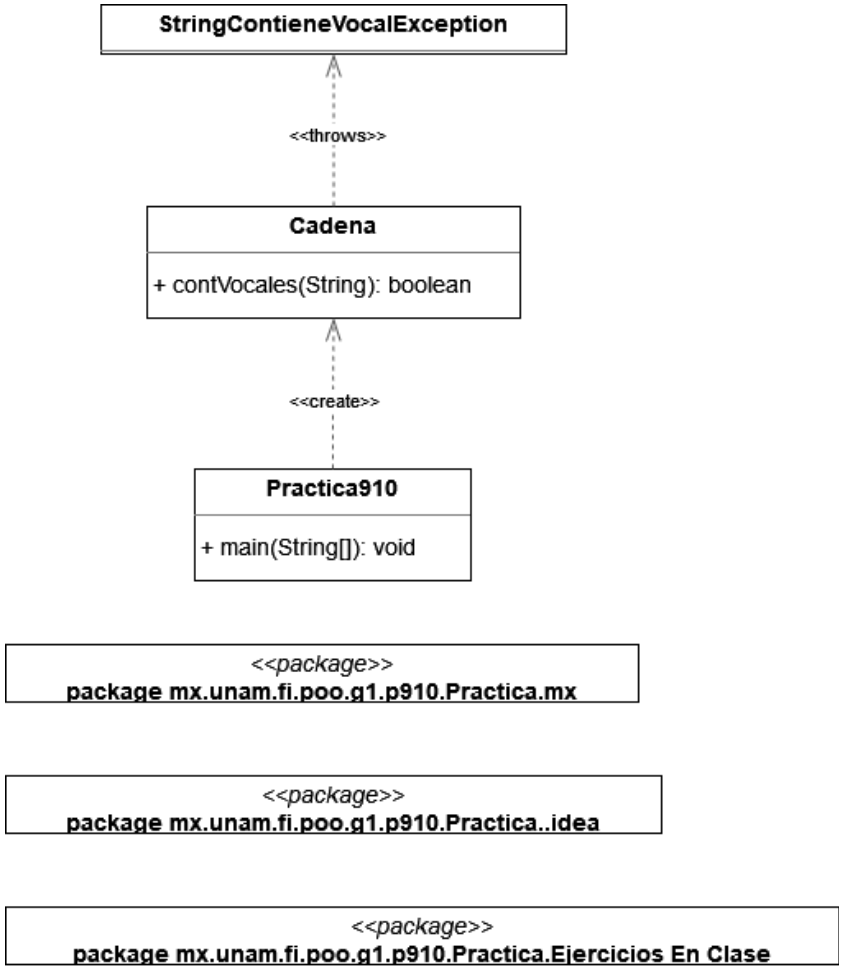
Diagrama de Casos de Uso



Se puede apreciar que el único caso es que el usuario ingrese una cadena, dicha acción verifica las vocales, lo cuál implica manejar excepciones

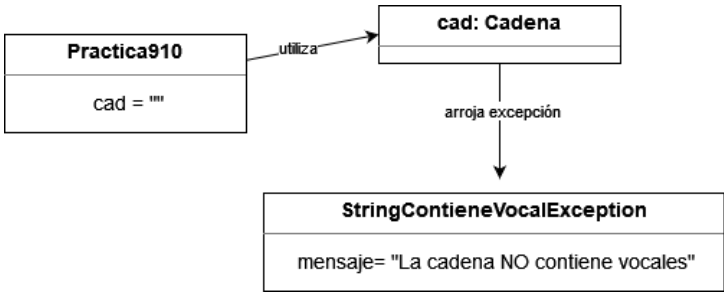
[1] Para la elaboración de los Diagramas se basó en **[3]**

Diagrama de Clases



Como se había estado haciendo desde antes, se pueden apreciar los nombres de las clases así como los métodos, en este particular caso, ninguna con algún atributo

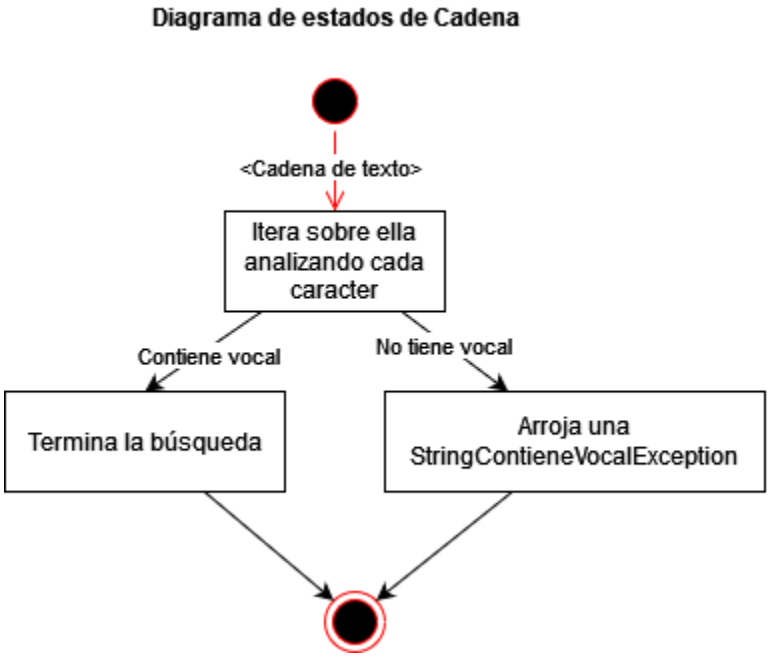
Diagrama de Objetos



Se puede apreciar que solo se tratan los atributos, mismos que se encuentran debajo del nombre de la clase del objeto

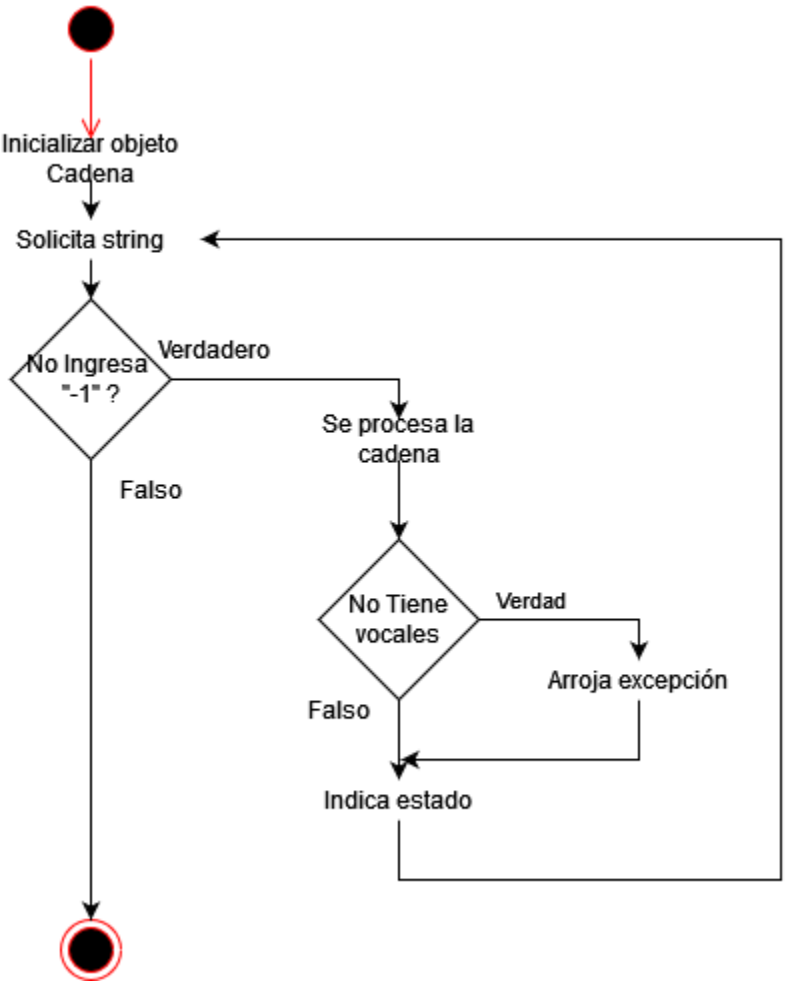
Diagramas Dinámicos

Diagrama de Estados



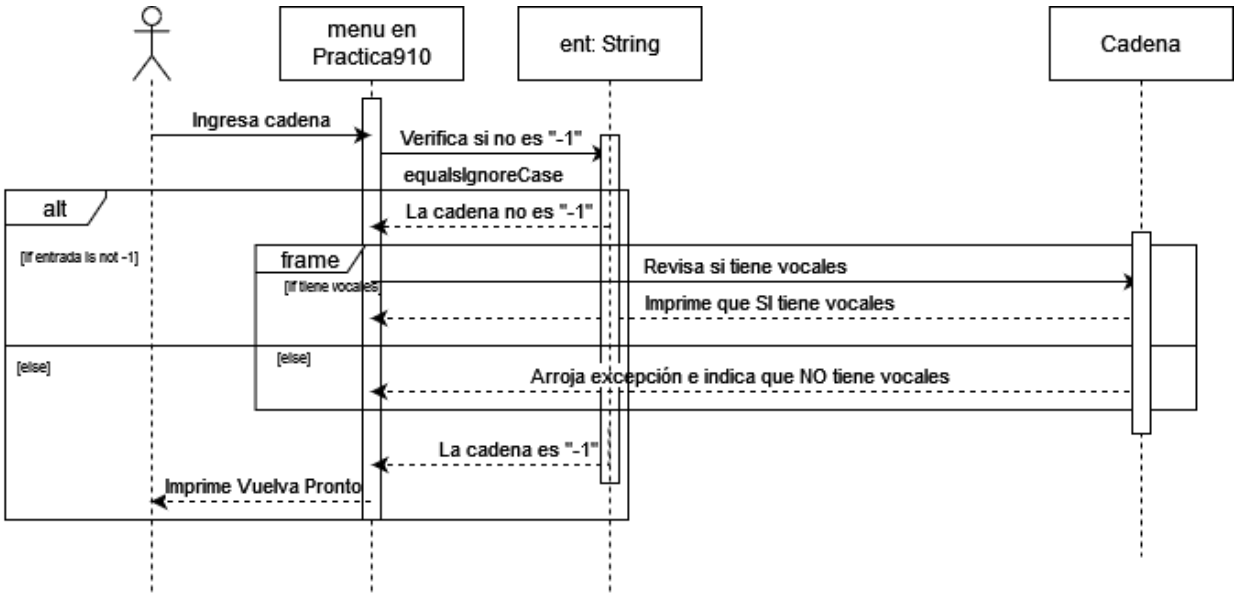
El ciclo de vida del objeto Cadena con dos variantes dependiendo de la entrada; termina la búsqueda si contiene una vocal o arroja una excepción

Diagrama de Actividades



Actividades realizadas, primero se solicita una string, misma que se evaluará en dado caso de que no tenga una vocal se arrojará una excepción. El ciclo acaba cuando se ingresa un -1

Diagrama de Interacción: Secuencia



Se puede apreciar que la secuencia completa empieza ingresando una cadena, luego dependiendo de si su valor es -1 o no, el programa termina o sigue su ejecución respectivamente.

Se llama al objeto string con el método equalsIgnoreCase para determinar si es -1.

En dado caso de que no lo sea, desde Practica910 se manda a llamar al objeto Cadena para ver si la palabra tiene vocales, en dado caso lo imprime, pero en dado caso de no tener, arroja una excepción.

Pruebas

Las **pruebas** al ser cortas se describen en la sección de resultados directamente.

Resultados

Ejercicios de clase

Ejercicio0
`Error: NO es posible dividir entre 0` Para una entrada de 10/0

Ejercicio1
`` Error: No se puede calcular la raiz ``` Para una entrada de -9
Ejercicio2
Números duplicados ``` Cuantos numeros deseas ingresar 4 Ingresa los enteros: 90 90 16 19 Error: Numero duplicado encontrado: ``` **Números no duplicados** ``` Cuantos numeros deseas ingresar 3 Ingresa los enteros: 1 2 3 NO hay numeros duplicados! ```

Ejercicios de Tarea

Caso en el que se ingresa una palabra con vocal

```
Ingresa una cadena:
Me encanta POO
-> La cadena: "Me encanta POO" SI tiene vocales
```

Caso en el que se ingresa una palabra sin vocales

```
Ingresa una cadena:
L FLDSMDFR
Error: La cadena NO contiene vocales
Ingresa una cadena:
```

Caso de salida

```
Ingresa una cadena:
-1
Vuelva pronto :)
```

Se puede notar que para identificar la cadena -1, se utilizó el método `equalsIgnoreCase()`

Conclusiones

Al momento de generar sistemas, es imprescindible que se haga un análisis detallado de lo que se requiere, así como establecer las convenciones de lo que ocurrirá en el flujo de datos del programa, esto es, definir cuáles serán las entradas y salidas y por consiguiente, cuáles entradas pueden no adecuarse a lo requerido, en dado caso se deben manejar situaciones especiales donde se manejen aquellos datos o situaciones que no eran lo esperado. Dichas situaciones se conocen como excepciones.

Las excepciones en general permiten al sistema en construcción ser capaz de ser autosustentable, continuar con su ejecución sin interrupción pero sin ignorar la situación anormal que representa dicha excepción.

Para poder analizar las entradas, salidas, actividades y situaciones, en general los estados de un sistema, es útil la generación de diagramas UML, que por convención son los elegidos a la hora de elaborar figuras que describan el funcionamiento tanto funcional como no funcional de tu programa.

De esta forma, ambas herramientas, las excepciones para poder manejar a nivel de código, aquellos que no es correcto por convención y saber qué hacer en dichos casos especiales sin afectar la ejecución del programa; y los diagramas UML estáticos y dinámicos son primordiales para fomentar las buenas prácticas de programación, pero también para la documentación del proyecto, para así construir sistemas eficientes, durables y mantenibles.

Referencias

[1] P. Deitel y H. Deitel. *CÓMO PROGRAMAR EN JAVA*. Sétima edición. México: Pearson Educación México, 2008. ISBN: 978-970-26-1190-5.

[2] H. Gomaa, *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge, UK: Cambridge University Press, 2011. ISBN 978-0-521-76414-8

[3] "String (Java SE 11 & JDK 11)". Java API. Accedido el 25 de octubre de 2024. [En línea]. Disponible: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

[4] "Programación Orientada a Objetos" notas de clase de 1323, División de Ingeniería Eléctrica, Facultad de Ingeniería de la Universidad Nacional Autónoma de México, Verano 2024.