PRÁCTICA 2ª OPORTUNIDAD (2019-2020)

Íker García Calviño

INTRODUCCIÓN:

Queremos crear un algoritmo para cifrar en nuestro sistema las contraseñas de los usuarios. Por simplicidad supondremos que las contraseñas son, como mucho, de 10 letras pertenecientes alfabeto inglés en minúsculas (de la 'a' a la 'z') y que siempre se introducen de forma correcta (no es necesario comprobarlo).

PRINCIPIOS DE DISEÑO:

PRINCIPIO DE RESPONSABILIDAD ÚNICA:

Una clase debe tener solo una razón para cambiar.

- Se aplica en todas las clases porque cada una de ellas no debería tener varias responsabilidades que puedan afectar unas a otras.

PRINCIPIO ABIERTO-CERRADO:

Una entidad de software (clase, módulo, función, etc.) debe quedarse abierta para su extensión, pero cerrada para su modificación.

- No aplica.

PRINCIPIO DE SUSTITUCIÓN DE LISKOV:

Sea $\phi(x)$ una propiedad comprobable acerca de los objetos x de tipo T. Entonces $\phi(y)$ debe ser verdad para los objetos y del tipo S donde S, es un subtipo de T.

- No aplica.

PRINCIPIO DE SEGREGACIÓN DE INTERFACES:

Los clientes de un programa dado sólo deberían conocer de éste aquellos métodos que realmente usan, y no aquellos que no necesitan usar.

- No aplica.

PRINCIPIO DE INVERSIÓN DE LA DEPENDENCIA:

Los módulos de alto nivel no deberían depender de los módulos de bajo nivel. Ambos deberían depender de abstracciones (p.ej., interfaces).

Las abstracciones no deberían depender de los detalles. Los detalles (implementaciones concretas) deben depender de abstracciones.

- La clase *Scrambler* con la interfaz *Operation*.

PATRONES DE DISEÑO:

Los patrones usados en este proyecto son el Factoría y el Estrategia, se han usado estos dos debido a que uno de ellos aporta una independencia muy grande y el otro nos ayuda con la generación de un interfaz para llevar a cabo distintas ejecuciones del algoritmo.

PATRÓN FACTORÍA:

El patrón Factoría es un patrón que nos permite crear distintas clases que generarán objetos concretos lo que permite implementar de manera fácil las operaciones requeridas de manera independiente y que en un futuro podemos añadir una mayor cantidad de ellas simplemente añadiendo una clase que la ejecute y una condición para poder llevarla a cabo.

- Nos aporta la independencia de cada uno de los algoritmos para poder crear otros sin tener que editar clases que ya funcionan correctamente y de la misma manera que si necesitamos realizar una modificación en alguna podamos modificarla sin miedo a que toda la clase deje de funcionar.

PATRÓN ESTRATEGIA:

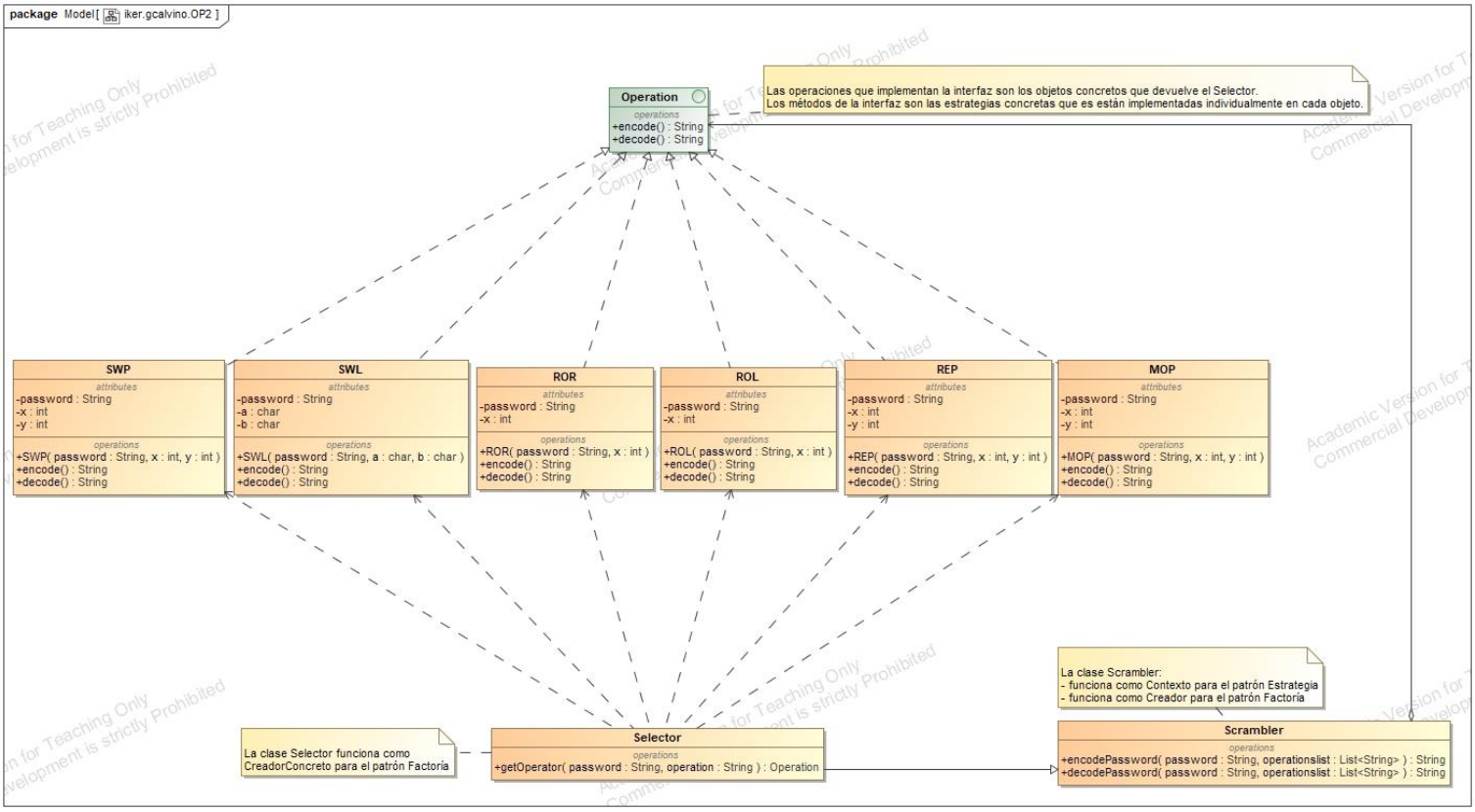
El patrón Estrategia es un patrón que nos permite crear métodos a partir de una interfaz la cual tal y como se nos presenta en el enunciado no va a modificarse porque sobre una contraseña solo podemos realizar esas dos operaciones (codificación y descodificación) y por ello nos permite poder llevar a cabo las dos estrategias de manera individual sobre cada algoritmo.

- Nos aporta poder elegir de qué manera podemos ejecutar la operación; si hay que añadir un nuevo método para cada operación con implementarlo en el interfaz (si hay que usarlo para todas las operaciones) o solo en dicha operación en específico poder hacerlo sin modificar las ya creadas anteriormente y que si su ejecución de codificación es muy distinta de la de descodificación, poniendo como ejemplo el ejercicio que se nos pide, podemos implementarlas independientemente (se repite alguna parte de código en este ejercicio, pero porque con las operaciones requeridas coincide en la mayoría el algoritmo *encode()* y *decode()*, pero en futuras operaciones puede que esto no sea posible).

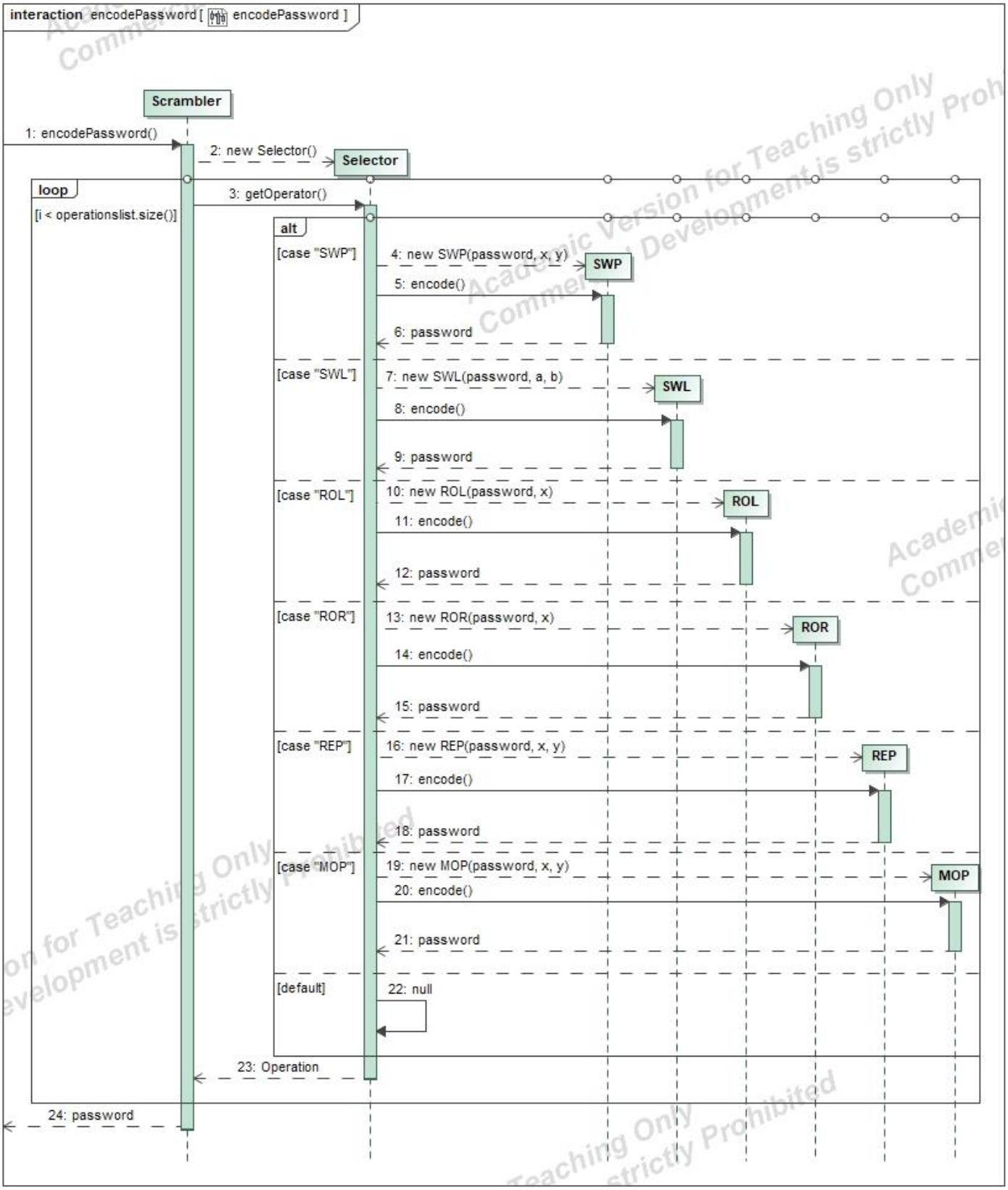
VIDEO:

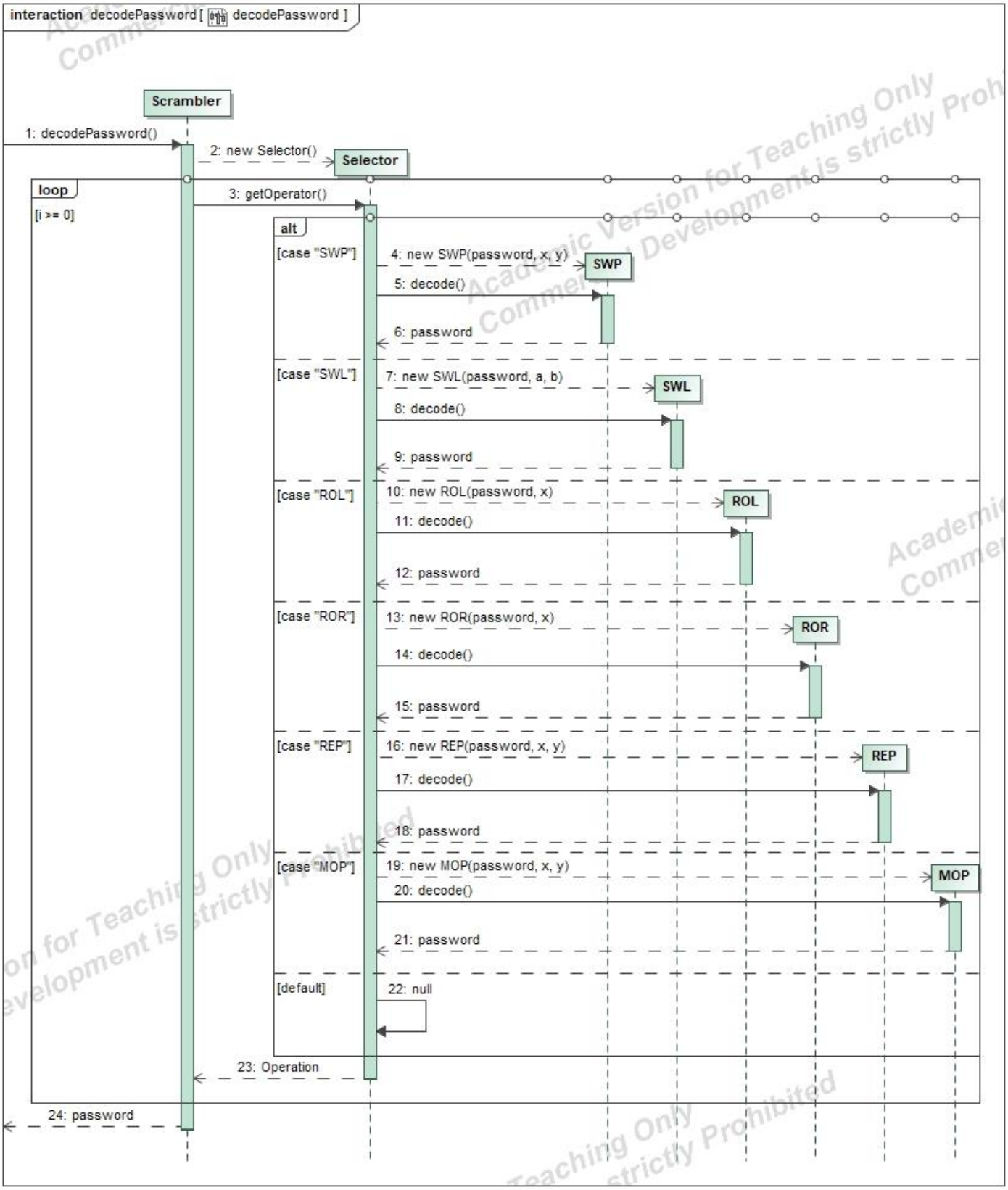
https://web.microsoftstream.com/video/a82ad8f8-bf08-40d0-afa3-ec7c8302ab95

DIAGRAMA DE CLASES



DIAGRAMAS DINÁMICOS




interaction decodePassword [decodePassword]
Scrambler
Selector
SWP
SWL
ROL
ROR
REP
MOP
1: decodePassword()
2: new Selector()
loop
[i >= 0]
3: getOperator()
alt
[case "SWP"]
4: new SWP(password, x, y)
5: decode()
6: password
[case "SWL"]
7: new SWL(password, a, b)
8: decode()
9: password
[case "ROL"]
10: new ROL(password, x)
11: decode()
12: password
[case "ROR"]
13: new ROR(password, x)
14: decode()
15: password
[case "REP"]
16: new REP(password, x, y)
17: decode()
18: password
[case "MOP"]
19: new MOP(password, x, y)
20: decode()
21: password
[default]
22: null
23: Operation
24: password