****

Tecnológico de Costa Rica

Área Académica de Administración de Tecnologías de Información

Diseño de software

Documentación del diseño del sistema

**Profesor:** Luis Javier Chavarría Sánchez

**Estudiantes:**

Óscar Trejos Esquivel               2018163987

Daniel Quirós Picado                 2018080151

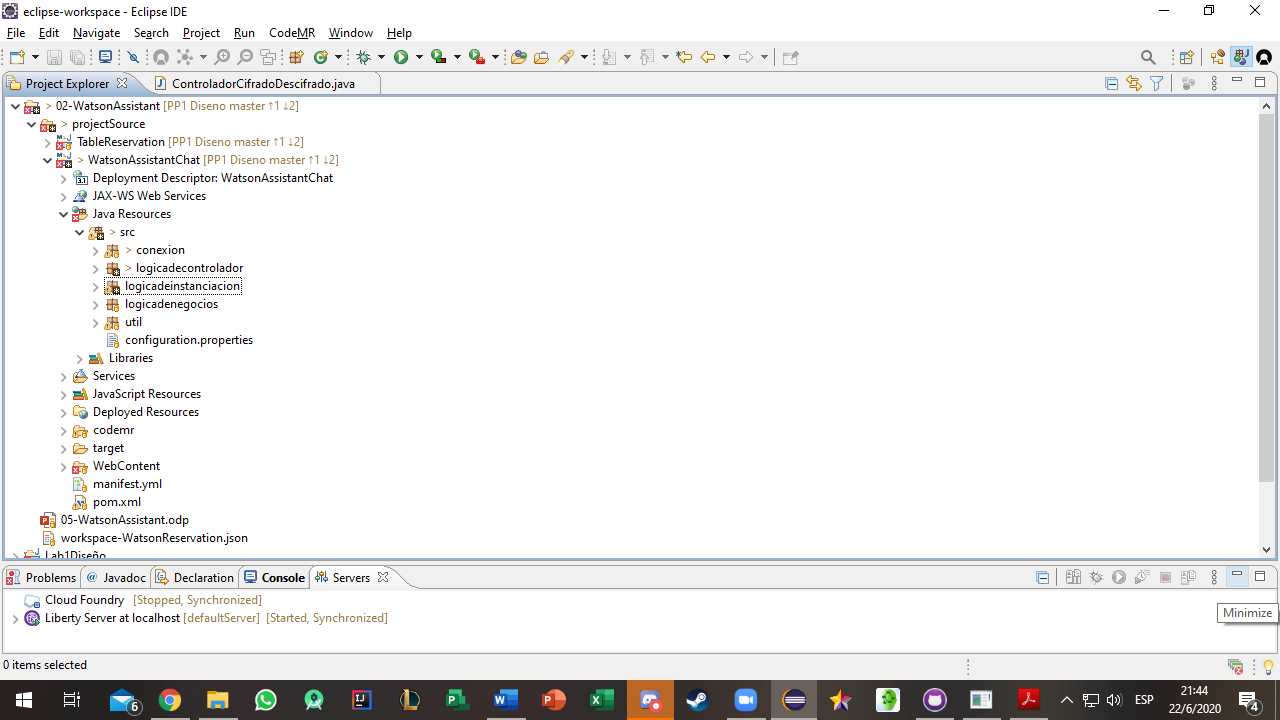
I Semestre 2020

**Documentación de diseño del sistema**

Durante la realización y diseño del proyecto en todo momento se trabaja bajos las mejores prácticas, bajo estándares previamente estudiados y establecidos, así como una serie de pasos que forman parte de un proceso disciplinado. A continuación, se muestran las evidencias respecto a su cumplimiento.

**Separación de responsabilidades**

Esto se refiere a separación de diferentes elementos y clases contenidos en el proyecto, agrupados por su comportamiento u objetivo. En el proyecto se denota una separación de responsabilidad por, lógica de negocios, lógica de instanciación, clases utilitarias, conexión al asistente y la lógica de controlador. En la siguiente imagen se evidencia lo mencionado.



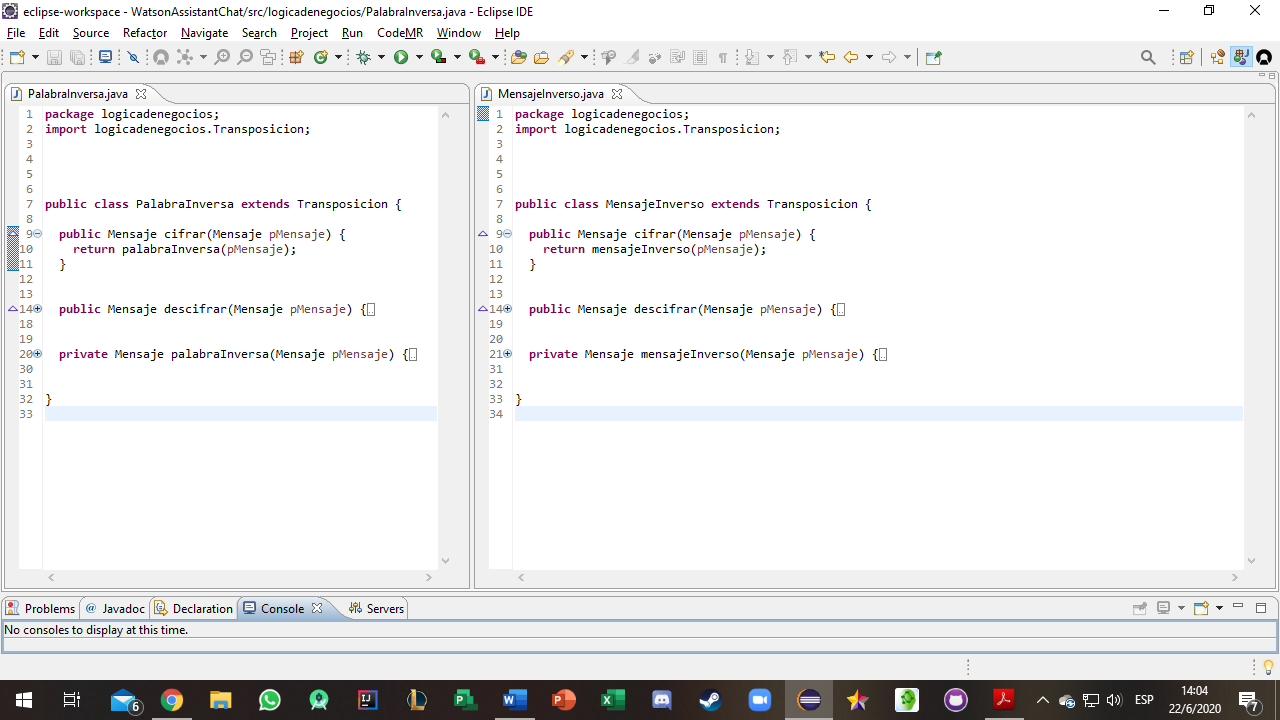
**Principios S.O.L.I.D.**

**S (SRP): Single responsability principle**

Cada clase que forma parte de los diferentes módulos, tienen una sola razón para cambiar, es decir, su cohesión es alta y responden a solamente a una necesidad de negocio o de funcionabilidad. A continuación, un ejemplo de su cumplimiento. Ambas clases son especializaciones de un tipo de cifrado y cada una de ella tiene una sola razón para cambiar, su respectiva funcionabilidad de cifrar o descifrar que poseen diferentes implementaciones entre sí.

Figura 1Captura de pantalla de un computador

Descripción generada automáticamente

Figura 2****

**O (O.C.P): Open and close principle**

Tanto la jerarquía como la lógica de negocios, que forman parte de los servicios de cifrado y descifrado cumple en su totalidad con este principio, ya que, es posible añadir funcionabilidad sin cambiar el código existente. Como se muestra en el siguiente ejemplo. El siguiente método es el encargado de instanciar los objetos de ICifrado, en caso de extensión de los tipos de cifrado el siguiente código no necesitará algún cambio debido que cumple OCP figura 3. Y en la figura 4 se muestra la jerarquía abierta para cualquier extensión.

Captura de pantalla de computadora

Descripción generada automáticamenteFigura 3

Imagen que contiene captura de pantalla, computadora

Descripción generada automáticamente  
Figura 4

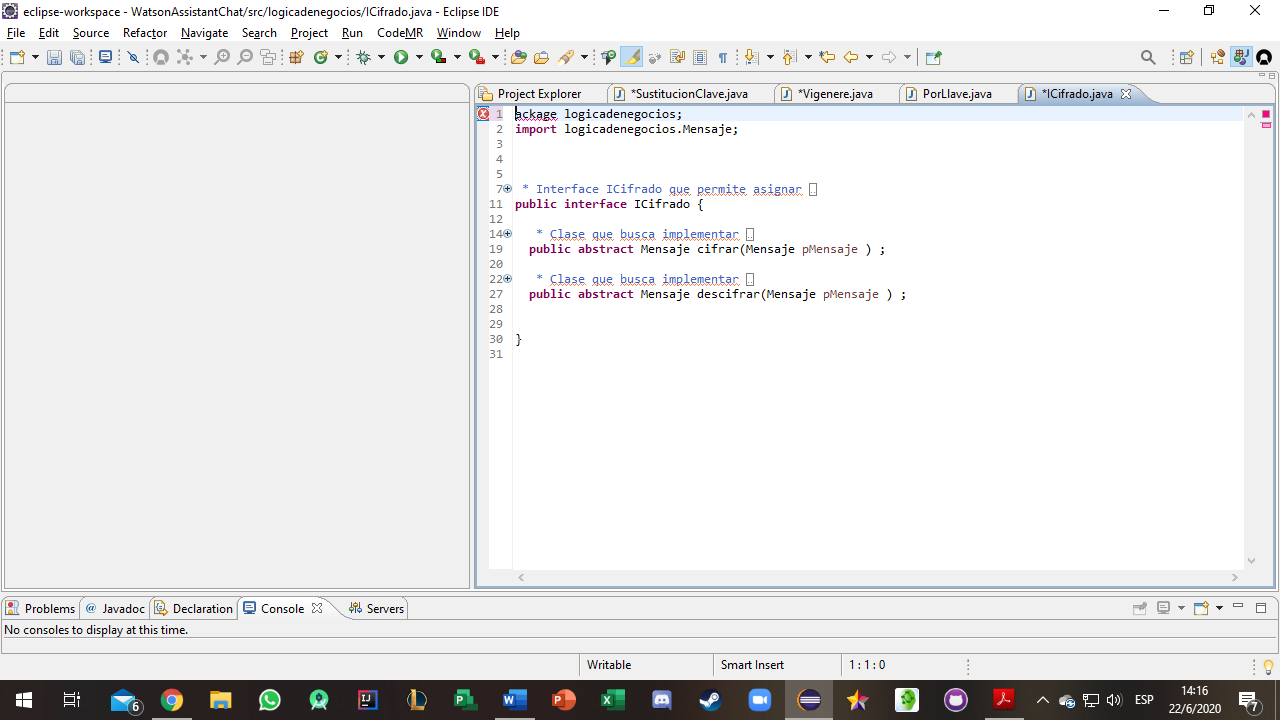
**L (L.S.P.): Liskov segregation principle**

Cada super clase presente en la lógica de negocios posee la misma interfaz que sus subclases en este caso solamente hay una jerarquía y se cumple este principio como se muestra en la siguiente imagen de la jerarquía SustituciónClave. Figura 6

Captura de pantalla de un computador

Descripción generada automáticamenteFigura 5

**Una captura de pantalla de una computadora

Descripción generada automáticamente**Figura 6 Figura 7 **Captura de pantalla de computadora

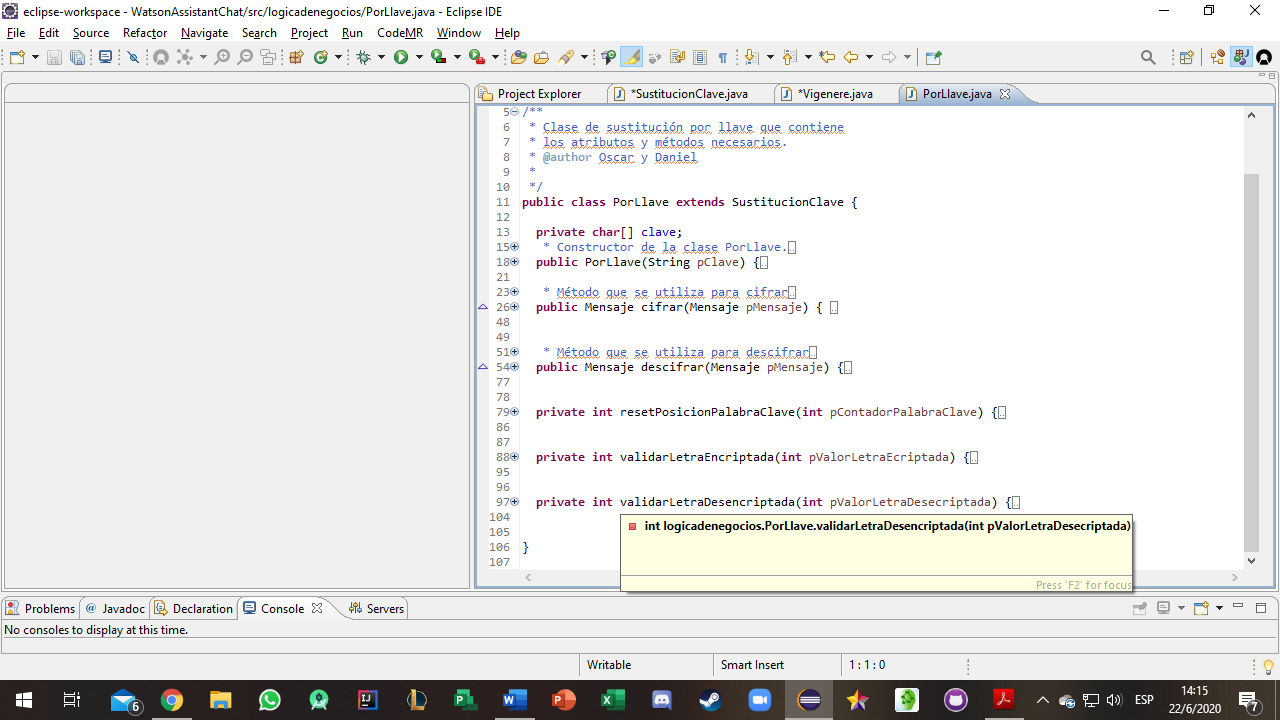
Descripción generada automáticamente**

Figura 8 Figura 9

**I (I.S.P): Interface Segregation Principle**

Cada interface agregada al proyecto debe contar con alta cohesión, es decir, cada elemento de la interface debe estar relacionado con las clases que lo implementan. Como se muestra a continuación la única interfaz en el proyecto ICifrado, posee alta cohesión respecto a las clases que la implementan. Por asuntos de legibilidad se muestra una sola implementación.

**Imagen que contiene captura de pantalla, computadora

Descripción generada automáticamente**Figura 10

**Una captura de pantalla de una computadora

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamente**

Figura 11 Figura 12

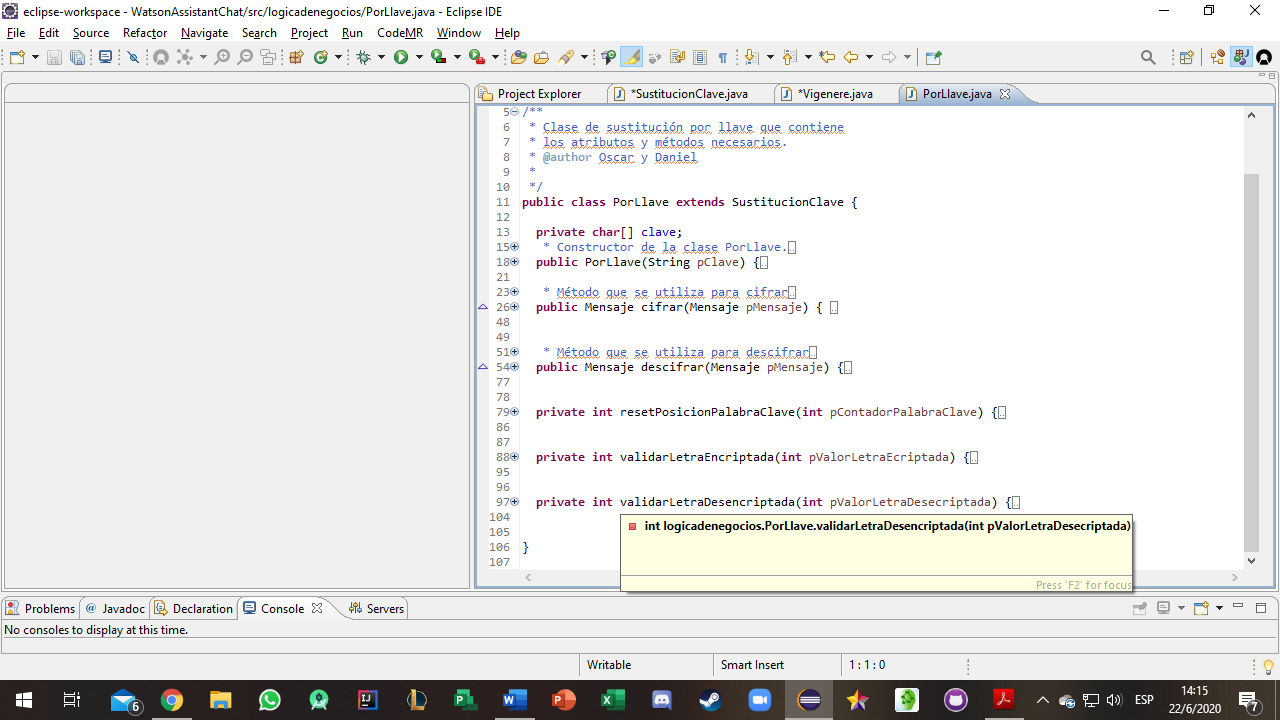
****

Figura 13

**D (D.I.P): Dependency Inversión Principle**

Este principio indica que se debe procurar que entre clases concretas exista al menos una capa de abstracción. En el siguiente ejemplo se muestra la aplicación de este principio entre las clases concretas Mensaje y las subclases Vigenere y PorLLave con una super clase y una interface de por medio.

Imagen que contiene captura de pantalla, computadora

Descripción generada automáticamenteFigura 14

**Una captura de pantalla de una computadora

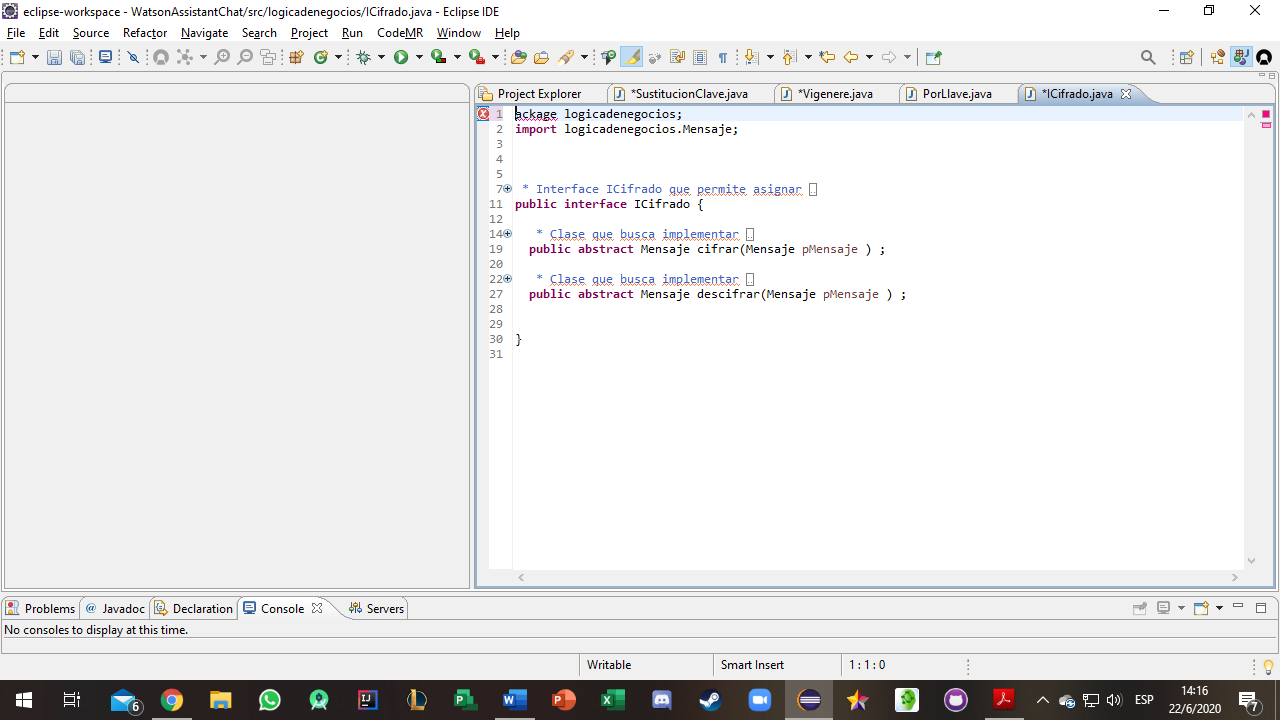
Descripción generada automáticamente**

Figura 15 Figura 16

**Captura de pantalla de computadora

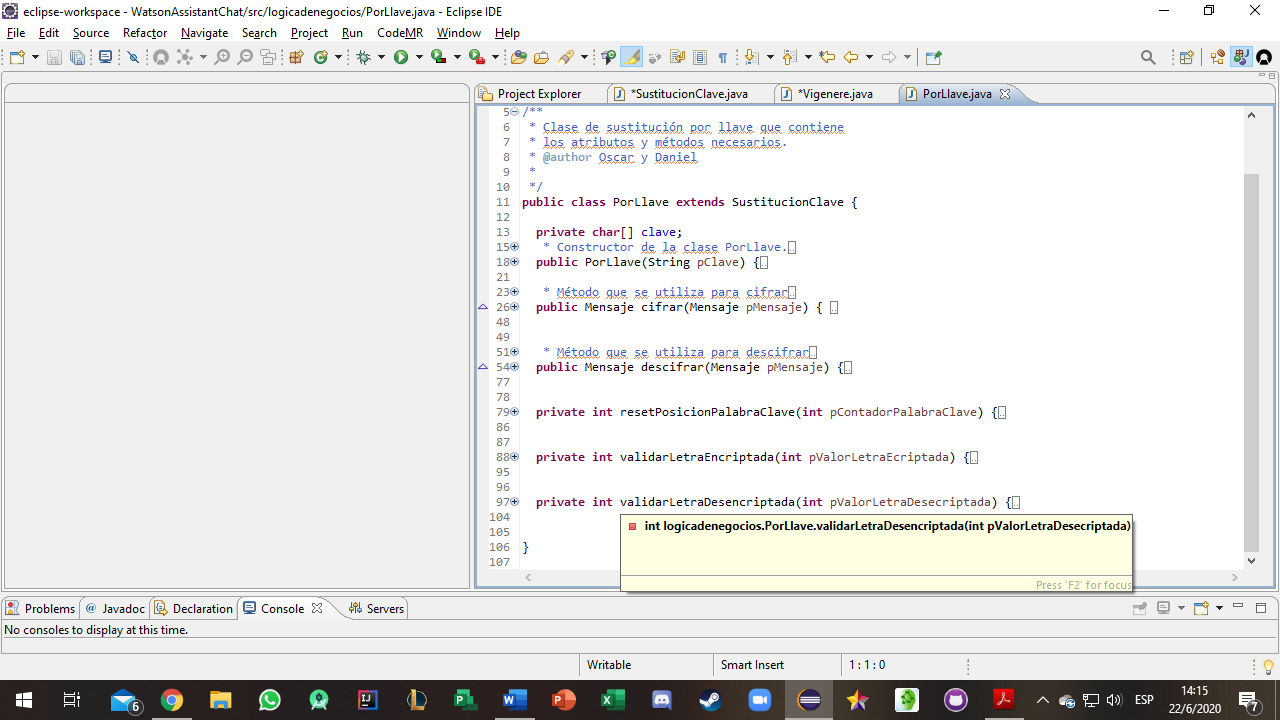
Descripción generada automáticamente**Figura 18

Figura 17

**Patrón creacional**

**SimpleFactory**

El patrón de diseño seleccionado para la realización del proyecto es SimpleFactory. Se decide implementar este patrón debido a la naturaleza de las clases de la lógica de negocios y para asegurar el cumplimiento de O.C.P. a su vez, que se toma en cuenta para no agregar más clases y aumentar la opacidad del diagrama.

El patrón creacional Simple Factory permite encapsular la lógica de instanciación, como en este caso las diferentes clases que implementan la interfaz ICifrado son instanciadas por la clase SimpleCifradoFactory.

El controlador es el encargado de ejecutar los servicios de instanciación de SimpleCifradoFactory cada vez que sea necesario, además se puede observar en su estructura el elemento respectivo para el proceso de instanciación fabrica.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 19

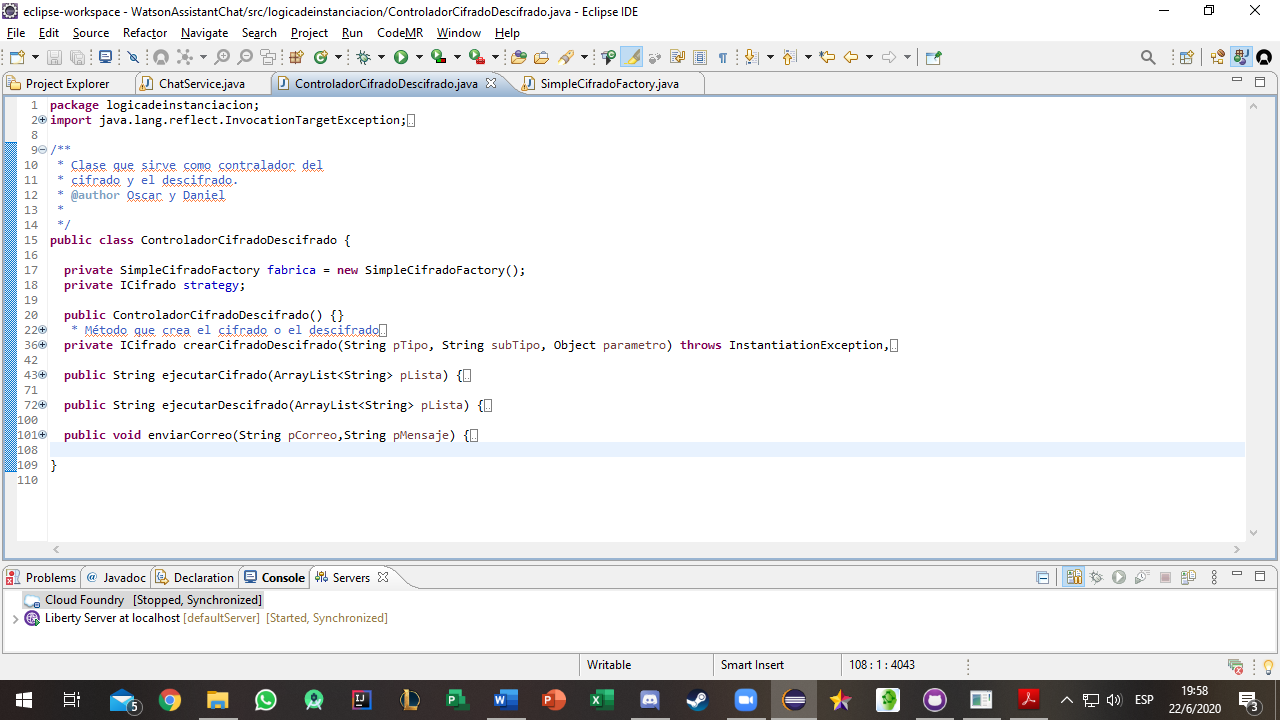
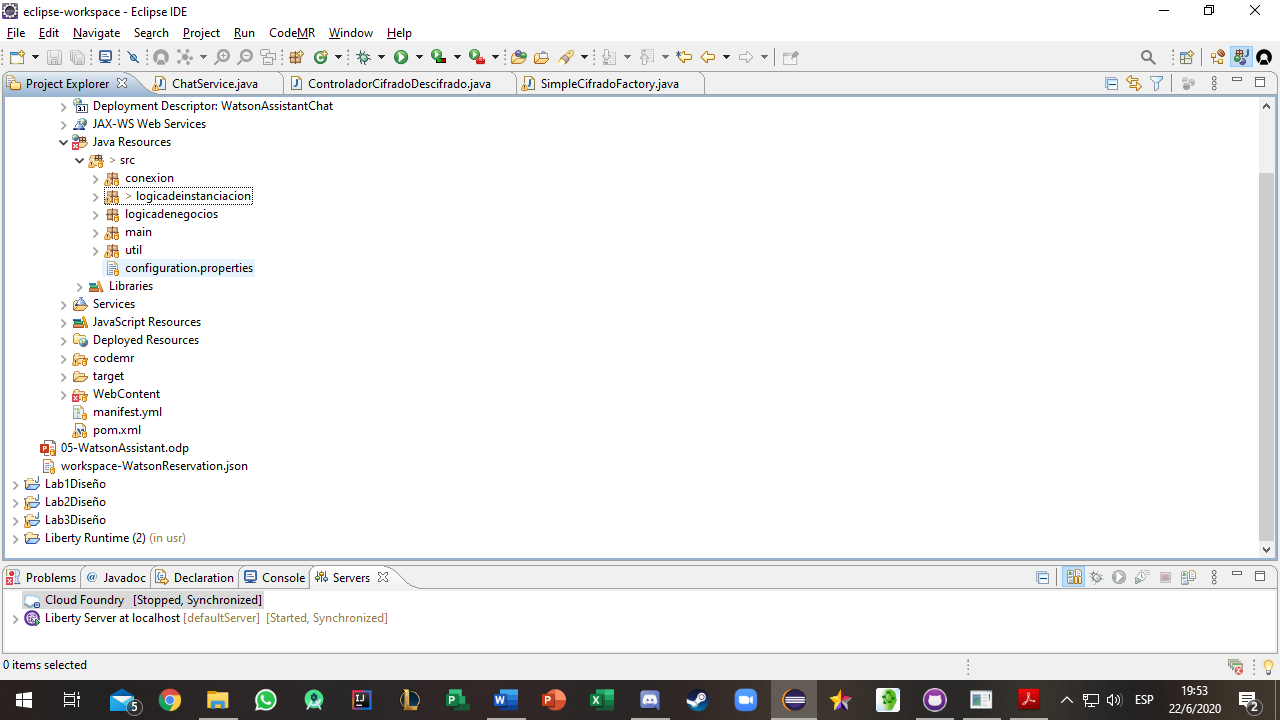


Figura 20 Figura 21

**Patrón de comportamiento**

Estos definen la manera de comunicación entre las clases y objetos. De este modo se definen comportamientos que se especializan de manera encapsulada en las diferentes clases.

Para la implementación en este proyecto se decide utilizar el patrón de comportamiento strategy. Este patrón tiene como finalidad definir diferentes comportamientos, que son encapsulados con lógicas respectivas a la clase, para una utilización poli fórmica. Este patrón ayuda a la escalabilidad del sistema en cuanto a la lógica de negocios

A continuación, se presenta una imagen de este tipo de patrón. En la clase “context”, parte de su estructura está compuesta por el elemento “strategy” que será utilizado en cada ocasión que se necesite algún servicio de las clases concretas. Se presenta con una notación diferente a UML 2.0

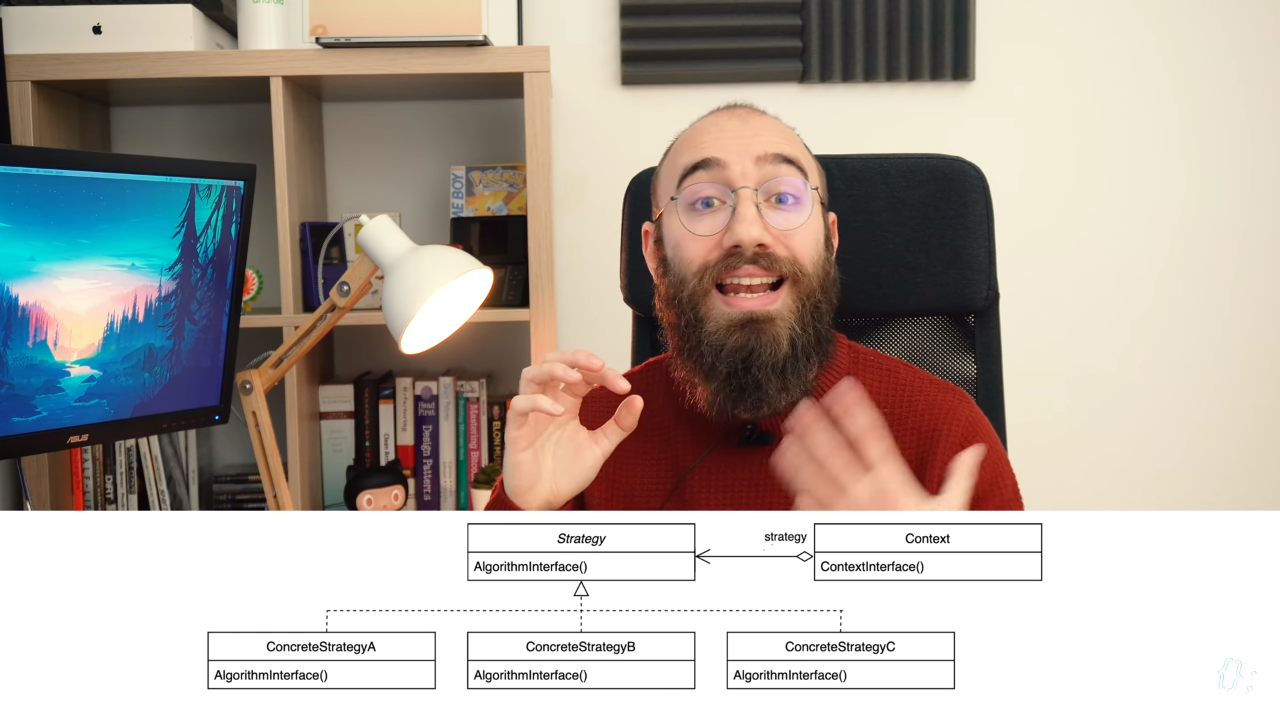
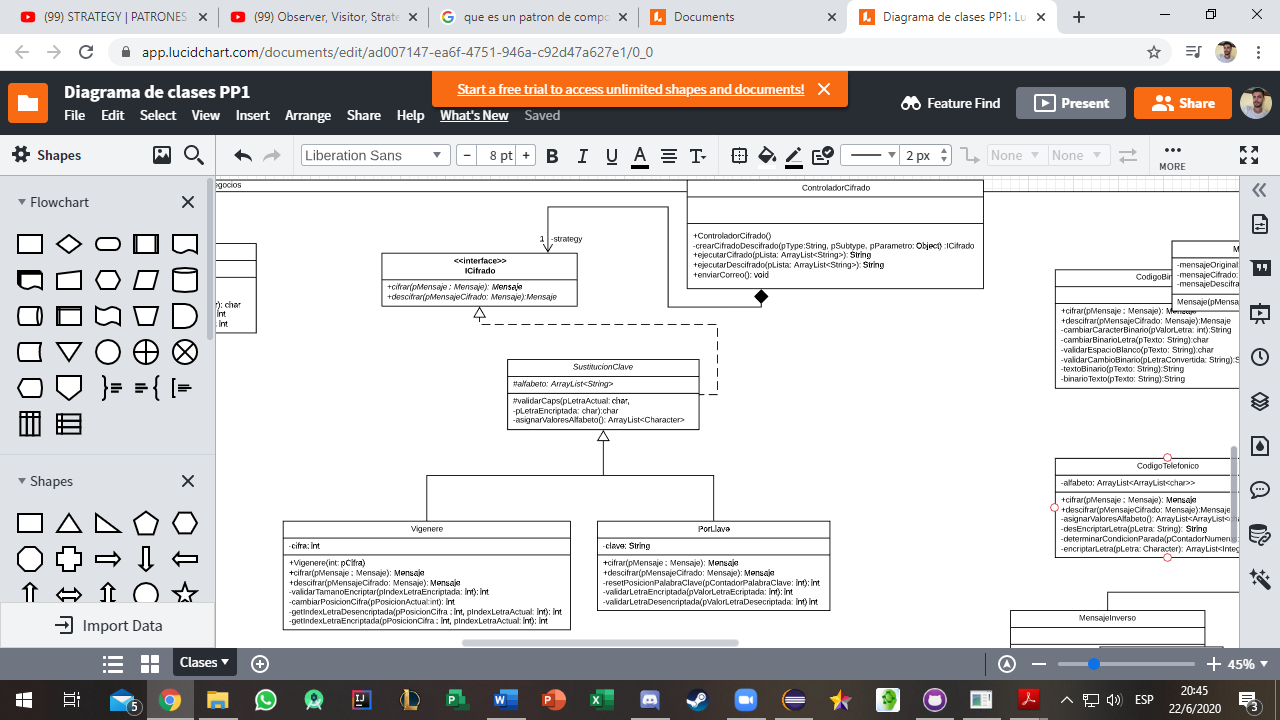


Figura 22

A continuación, se muestra evidencia de la implementación de este patrón de comportamiento tanto el diagrama de clases como su respectivo código. Se utiliza la notación UML 2.0 para el siguiente diagrama.

Figura 23

Equivalente a la clase Context figura 10. Estructura de la clase atributo strategy.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 24

Utilización de los métodos en las clases concretas. Clase ControladorCifradoDescifrado.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 25

Clases concretas e interface ICifrado el equivalente a la clase IStrategy in la figura 10.

**Una captura de pantalla de una computadora

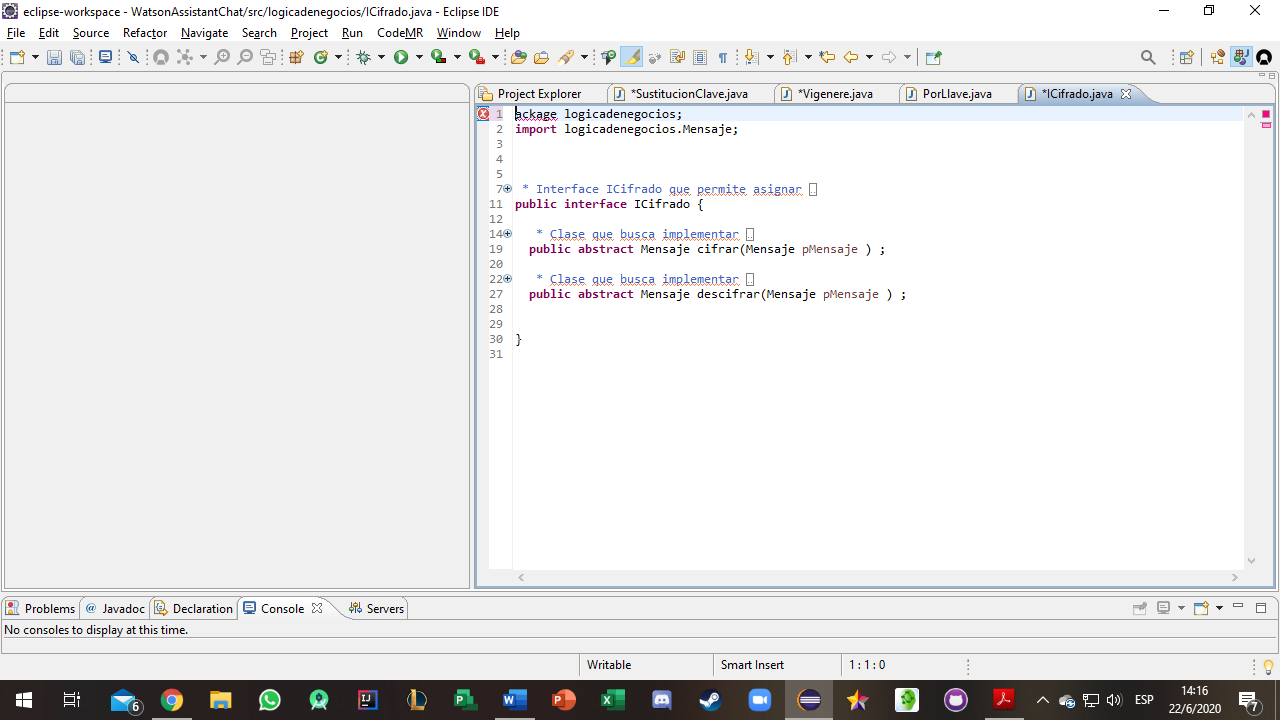
Descripción generada automáticamente**

Figura 26 Figura 27

**Captura de pantalla de computadora

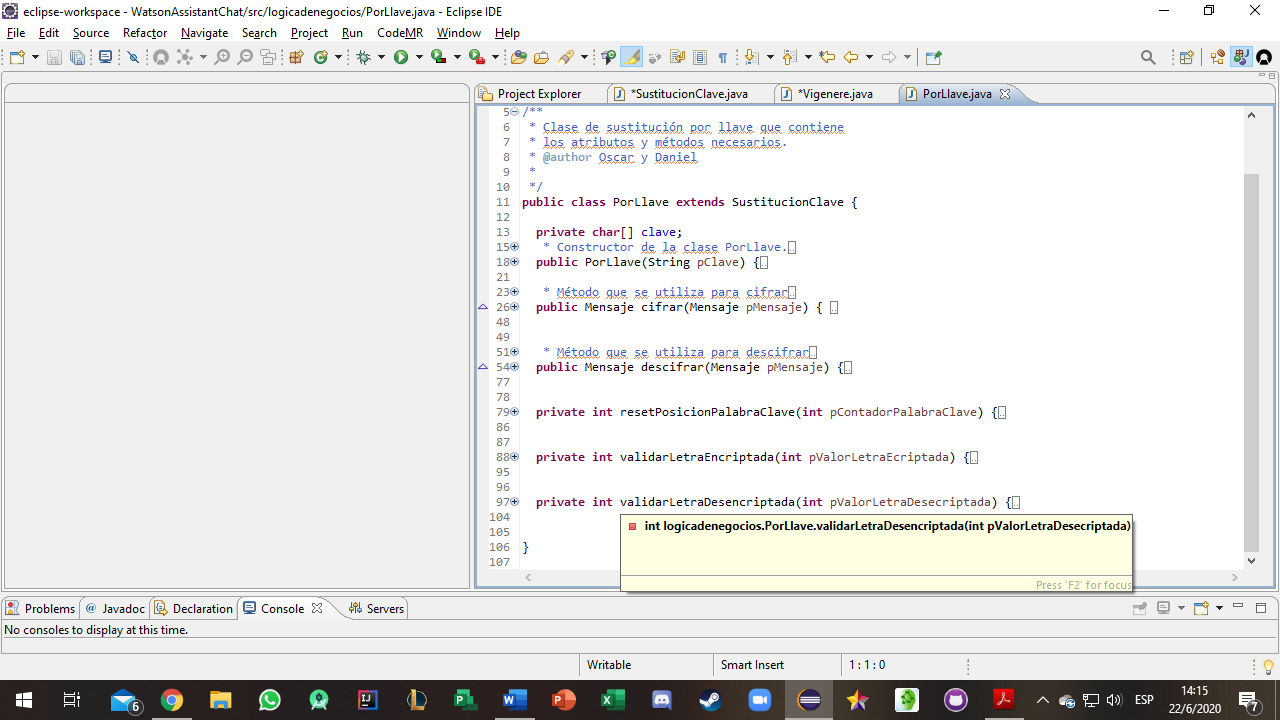
Descripción generada automáticamente**

Figura 28 Figura 29