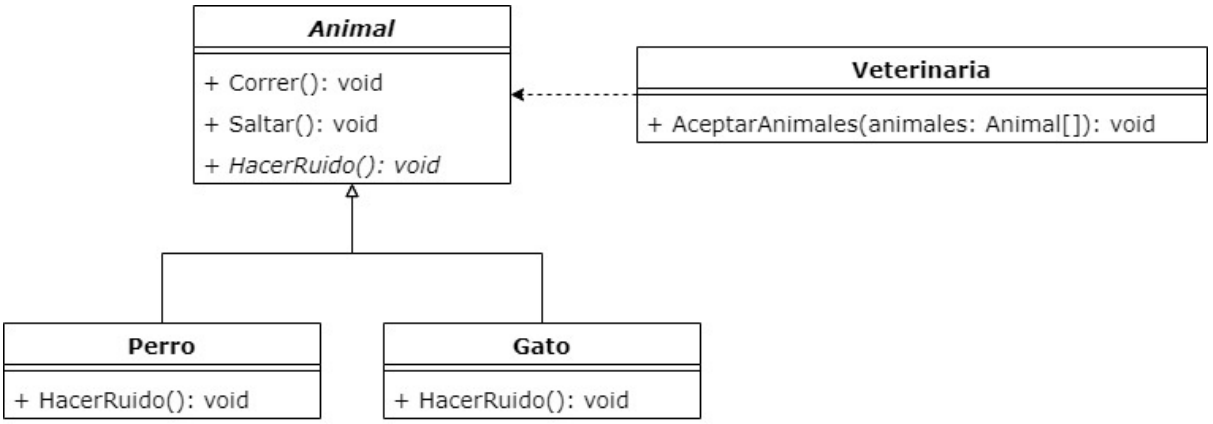
TRABAJO PRÁCTICO 4

Abstracción, Herencia y Polimorfismo, Patrones de diseño

Ejercicio 1

Implemente el diagrama de clases que está al final del enunciado del ejercicio. Los métodos Correr y Saltar de la clase abstracta Animal deben imprimir por pantalla las cadenas de texto “corriendo” y “saltando” respectivamente. Cada instancia particular de las subclases de la clase Animal implementa el método HacerRuido con la impresión por consola de texto de su sonido característico.
Al desarrollar este ejercicio utilice un arreglo con los dos tipos de animales, e inlcuya el arreglo a una instancia de la clase Veterinaria. Dicha clase en su único método itera el arreglo pasándole el mensaje HacerRuido a todas las instancias de la clase Animal.
Nota: El objetivo de este ejercicio es la demostración de uso de la característica polimorfismo, por ello no es necesario realizar una fachada. Este ejercicio incurre en una técnica no recomendable que es la mezcla de código de presentación (la impresión por consola del sonido) y de código de dominio, pero se realiza por fines educativos y para mostrar más claramente el concepto.



Ejercicio 2

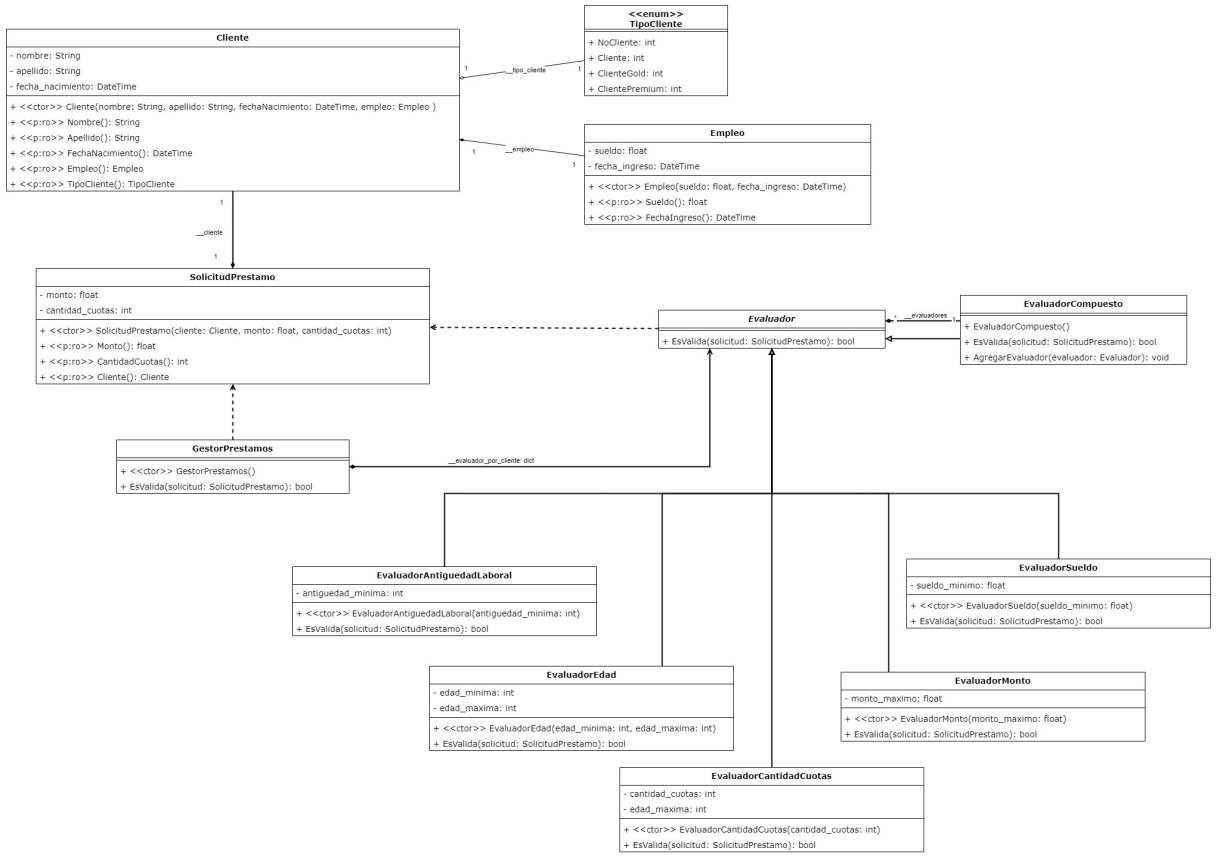
Se debe desarrollar una aplicación que permita validar solicitudes de préstamos bancarios,
siendo los requerimientos funcionales los siguientes:

- Existen cuatro tipos de clientes, siendo los mismos de menor a mayor categoría *No Cliente*, *Cliente*, *Cliente Gold* y *Cliente Platinum*.
- Los requisitos para la solicitud de préstamos son:
    - Tener entre 18 y 75 años, como mínimo 6 meses de antigüedad laboral y un sueldo mínimo de $5.000.
    - Los clientes del tipo *No cliente* pueden solicitar hasta $20.000 en un máximo de 12 cuotas.
    - Los clientes del tipo *Cliente* pueden solicitar hasta $100.000 en un máximo de 32 cuotas.

- Los clientes del tipo *Cliente Gold* pueden solicitar hasta $150.000 en un máximo de 60 cuotas.
- Los clientes del tipo *Cliente Premium* pueden solicitar hasta $200.000 en un máximo de 60 cuotas.

- Para que una solicitud de préstamo sea válida, se deben cumplir todos los requisitos anteriormente enumerados acorde al tipo de cliente que realiza la solicitud.

El diagrama de clases que se recibe del diseñador es el siguiente, en el cual se hace uso del patrón de diseño **Composite** para evaluar las reglas por tipo de cliente.



Adicionalmente el diseñador brinda las siguientes indicaciones:

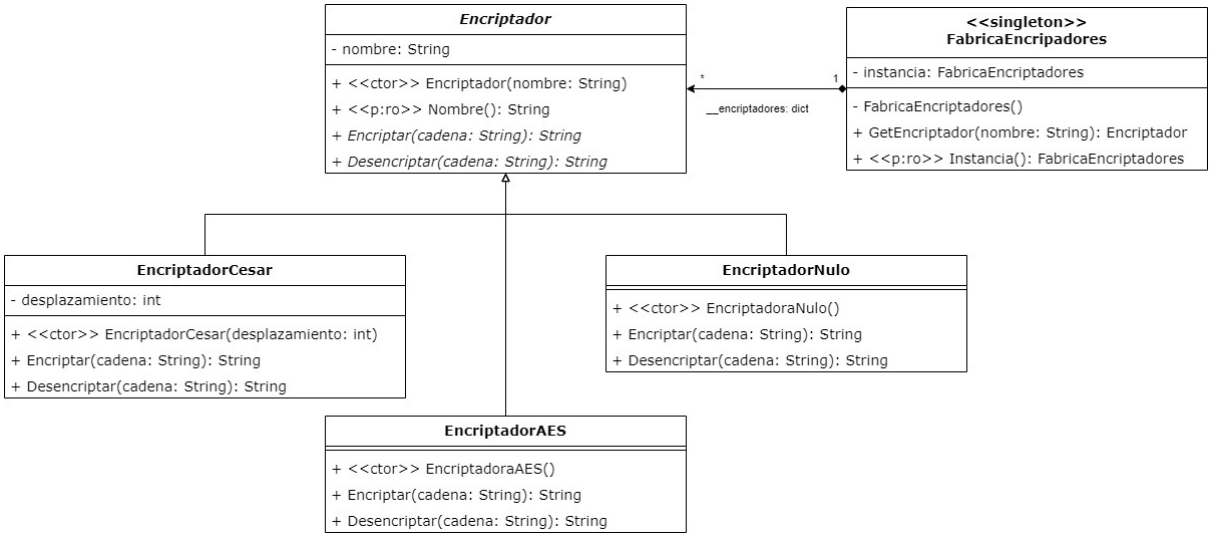
- La clase *Cliente* representa al cliente que realiza la solicitud. Por defecto todos los clientes son del tipo *No Cliente* exceptuando que explícitamente se indique lo contrario.
- La clase *Empleo* contiene información laboral acerca del cliente que realiza la solicitud
- La clase *SolicitudPrestamo* representa a la solicitud del préstamo que realiza el cliente.
- La responsabilidad de la interface *Evaluador* es evaluar una regla particular de la solicitud de préstamo, e indicar si la misma se cumple o no.

- Las clases *EvaluadorAntiguedadLaboral*, *EvaluadorEdad*, *EvaluadorMonto*, *EvaluadorCantidadCuotas* y *EvaluadorSueldo* tienen la responsabilidad de evaluar reglas particulares para indicar si una solicitud de préstamo es o no válida.
- Al momento de su creación, la clase *EvaluadorAntiguedadLaboral* recibe como parámetro la antigüedad mínima expresada en meses.
- La clase *EvaluadorCompuesto* permite evaluar si una solicitud de préstamo es válida de acuerdo a una serie de reglas.
- La clase *GestorPrestamos* mantiene las instancias de *Evaluador* que deben evaluarse de acuerdo al tipo de cliente que realiza la solicitud de préstamo.

Se deben identificar las operaciones del sistema utilizando el patrón de diseño GRASP Controlador de Fachada para desacoplar la lógica del sistema de la interfaz de usuario.
Utilice pruebas unitarias para validar el comportamiento de la clase *GestorPrestamos*.

Ejercicio 3

Se recibe un requerimiento para desarrollar un software que permita encriptar y desencriptar cadenas de caracteres utilizando diferentes algoritmos de encriptación, los cuales deben ser fácilmente intercambiables. Para satisfacer al requerimiento, el diseñador del equipo realiza el diagrama de clases que se puede observar a continuación. En el diseño se utiliza el **patrón de diseño GOF Estrategia** para definir los diferentes algoritmos de encriptación, los cuales respetan una interfaz común. También se hace uso de una **Factoría** implementada mediante el patrón **Singleton** para la obtención de los encriptadores particulares.



Adicionalmente, el diseñador brinda los siguientes detalles:

- La clase abstracta *Encriptador* establece el contrato mínimo que deben cumplir todos los algoritmos de encriptación, y sirve como clase de base para los diferentes encriptadores que se deseen implementar, agregando la responsabilidad de nombrar al encriptador.
- La clase *EncriptadorCesar* encapsula la lógica que permite encriptar y desencriptar cadenas utilizando el algoritmo de cifrado César, permitiendo establecer el desplazamiento que se aplicará sobre el alfabeto. El constructor de esta clase invoca al constructor de la superclase proporcionándole el nombre "César".
- La clase *EncriptadorAES* tiene la responsabilidad de realizar la encriptación y desencriptación de cadenas utilizando el algoritmo Advanced Encryption Standard o Rijndael. El constructor de esta clase invoca al constructor de la superclase proporcionándole el nombre "AES".

  Para resolver este tipo de encriptación, utilice la librería *pycryptodome:*
  - instale la librería: `pip install pycryptodome`
  - `ejemplo:` https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html
- La clase *EncriptadorNulo* es la implementación del patrón Null-Object de la interface *Encriptador*, donde los métodos encriptar y esencriptar devuelven la misma cadena proporcionada como parámetro. El constructor de esta clase invoca al constructor de la superclase proporcionándole el nombre "Null". Debe evitarse que esta clase pueda extenderse.
- La clase *FabricaEncriptadores* es un **Singleton** que es responsable de crear y mantener una instancia de cada implementación de la interface *Encriptador*.

El método GetEncriptador devuelve la instancia de *Encriptador* cuyo nombre coincide con el parámetro proporcionado. Si el nombre proporcionado no existe, entonces se debe devolver una instancia de *EncriptadorNulo*.

Se deben identificar las operaciones del sistema utilizando el patrón GRASP **Controlador de Fachada** para desacoplar la lógica del sistema de la interfaz de usuario.

Utilice **pruebas unitarias** para validar el comportamiento de los encriptadores.