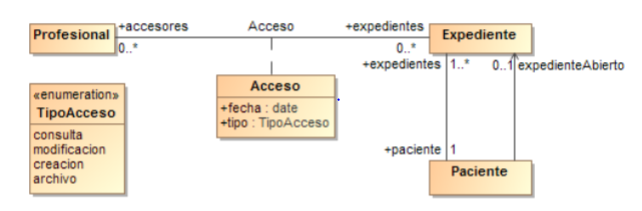
Pedro Díaz Gutiérrez

Ejercicio 1

Para implementar un esquema del código de andamiaje Java para este modelo, primero vamos a analizar las clases que obtendremos del sistema.

Por un lado, tendremos 3 clases (Profesional, Expediente y Paciente), una clase de asociacion (Acceso) y una clase enumerada (Tipo Acceso).



-Relación entre Expediente y Paciente

Observamos que existe una relación direccionada , “expedienteAbierto”, la cual introduciría un nuevo atributo en la clase Paciente de tipo Expediente , llamado “expedienteAbierto”.

Ambas relaciones públicas, debido al indicador “+” que los acompaña junto con su cardinalidad en la relación.

-Relación de asociación Acceso, entre Profesional y Expediente

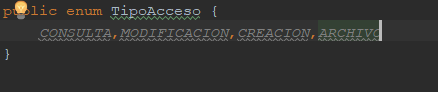
Para implementar esta parte del modelo, tendremos que crear las clases Profesional, Expediente y para la asociación que hay entre ellos, Acceso, tendremos que crear otra clase en donde “podamos mantener la consistencia de ambas”, conociendo así los diferentes accesos a cada expediente y los expedientes a los que ha accedido un determinado Profesional.

-Clase enumerada “Tipo Acceso”

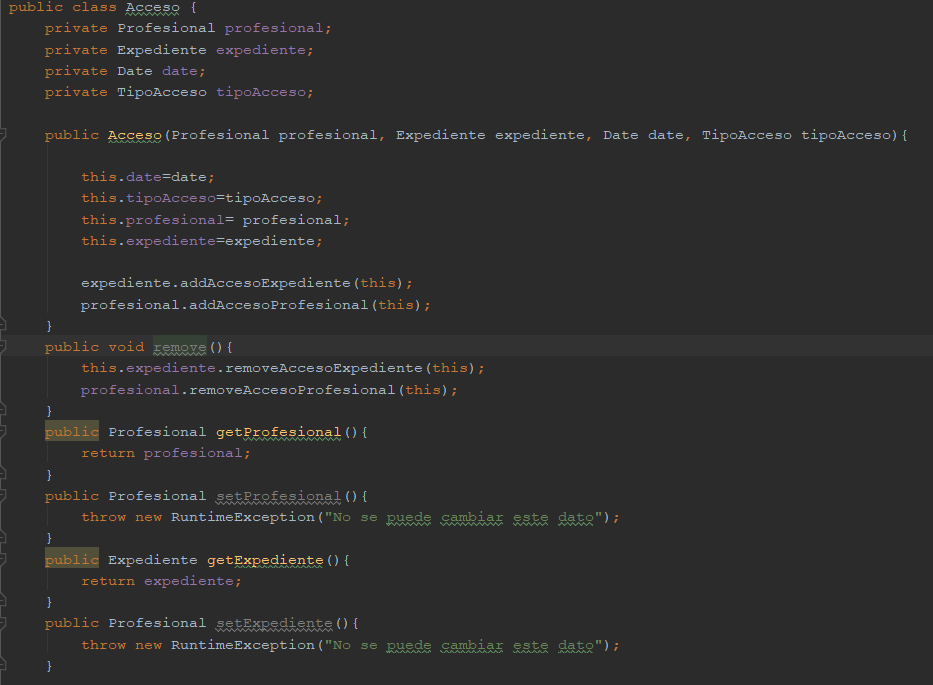
Clase enum de Java en la que introduciremos los tipos de acceso.

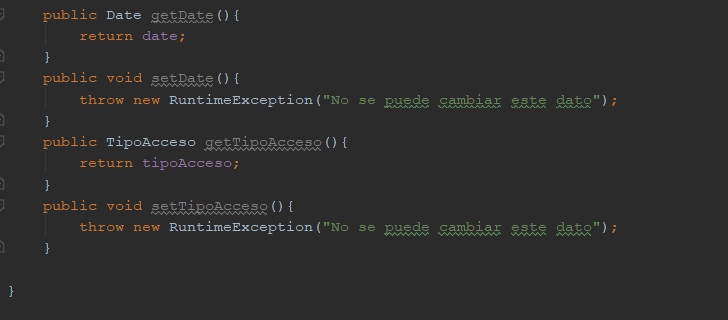
A continuación, se verá el código de andamiaje del modelo:

Clase Tipo Acceso:

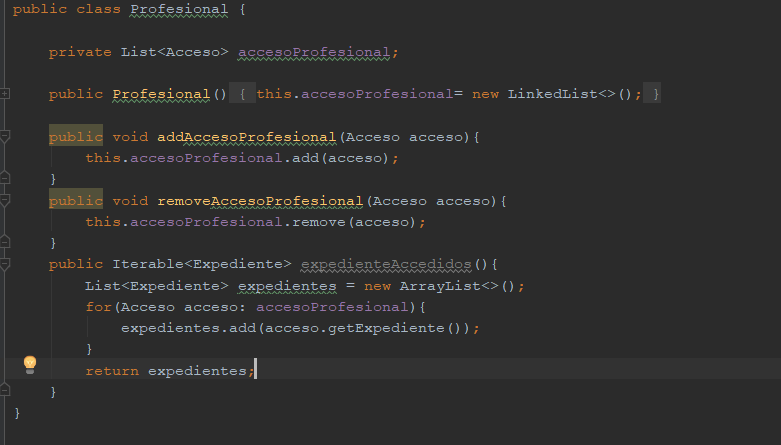


Clase Acceso:

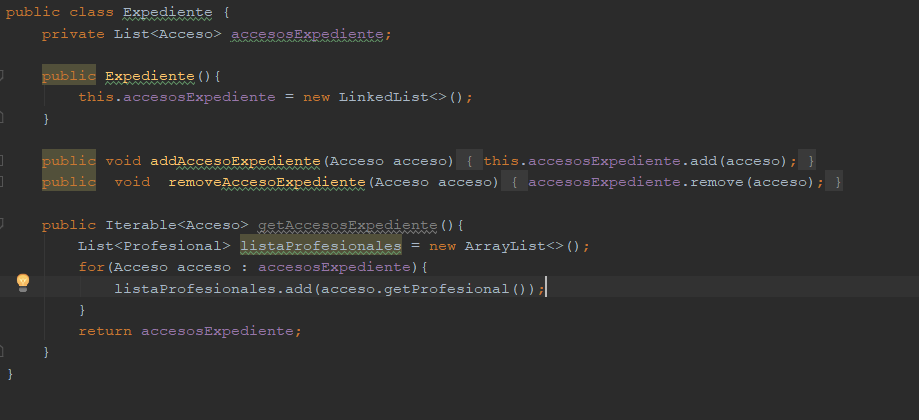




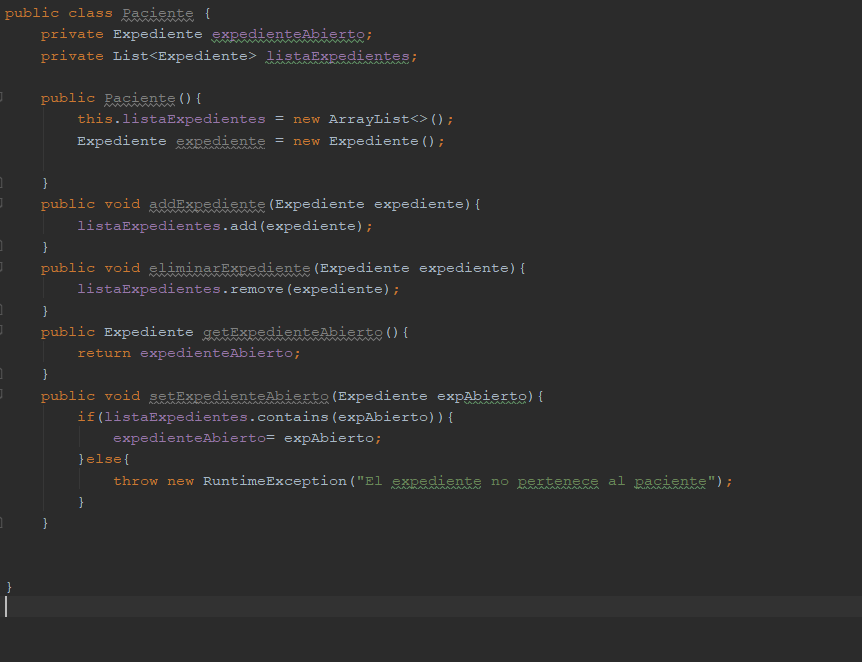
Clase Profesional:



Clase Expediente:



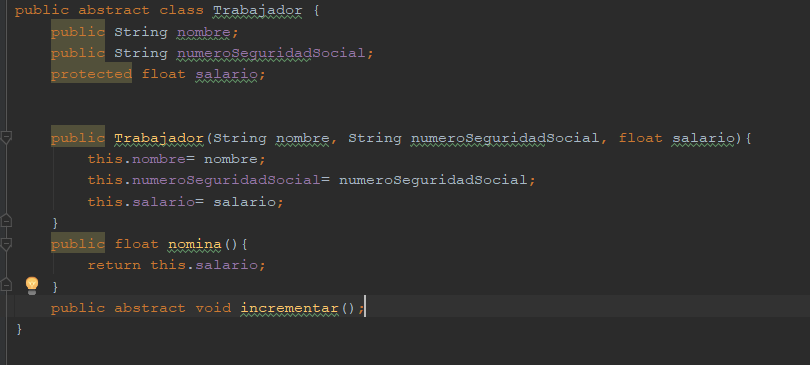
Clase Paciente:



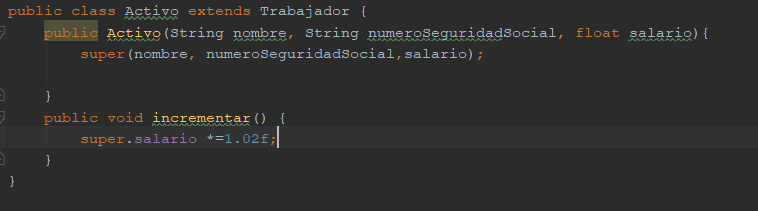
Ejercicio 2

1. Las clases anteriormente descritas en el enunciado del problema no se pueden implementar en Java, debido a que no existe la herencia múltiple de clases en este lenguaje.
2. Para obtener la funcionalidad requerida en el enunciado, implementaremos una clase MedioPensionista que herede(extends) de la clase Trabajador (abstracta) junto con dos atributos privados, Activo y Pensionista, para así distinguir el salario y poder hacer el cálculo correcto de este.
3. A continuación la implementación de la solución propuesta:

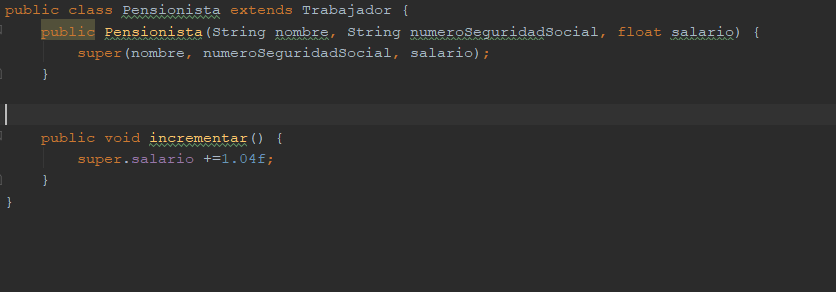
Clase Trabajador:



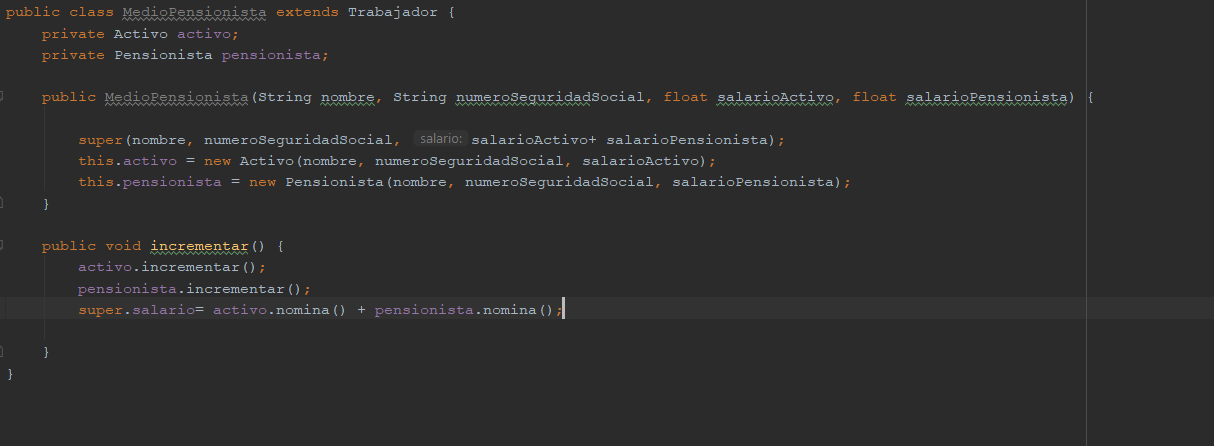
Clase Activo:



Clase Pensionista:



Clase MedioPensionista:



Ejercicio 3

Como dice el enunciado, tenemos que implementar el problema con una de las 3 posibles implementaciones vistas en clase. Estas son :

**-Mediante implementación condicional**: implementación que consiste en realizar comprobaciones de las guardas mediante el uso de bloques if/else/swtichs, cumpliendose asi el estado junto con sus condiciones.

Para realización de la implementación no elegiremos este método debido a la cantidad de if/else que deberíamos crear además de la complejidad que alcanzaría la aplicación.

**-Mediante tabla de estados**: implementación en la cual en la tabla de transición de estados se almacena directamente en la memoria del programa y se usa para actualizar el estado actual del sistema en respuesta a los eventos.

Tampoco utilizaremos este método debido a que volveremos a obtener una tabla de estados compleja.

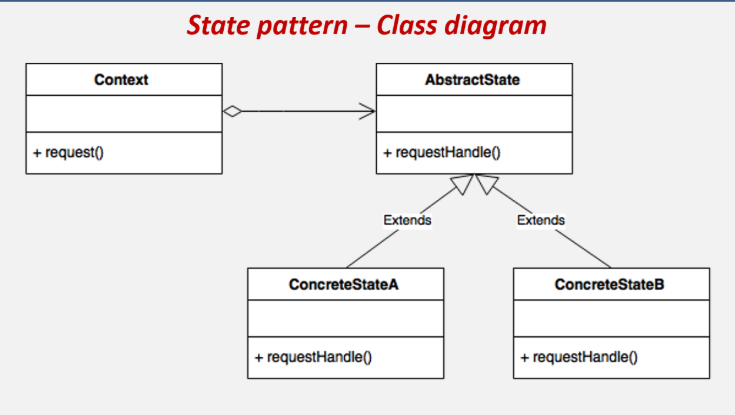
**-Mediante el patrón de diseño de estados:** este tipo de implementación se caracteriza por modificar su comportamiento dependiendo del estado en el que se encuentra la aplicación. Para lograr esto es necesario crear una serie de clases que representarán los distintos estados por los que puede pasar.

Tenemos como ejemplo la siguiente imagen, ejemplo de componentes que forman el patrón de diseño:

\*Context: componente que puede cambiar de estado, el cual tiene las propiedades del estado actual.

\*Abstract state: clase base para la generación de distintos estados.

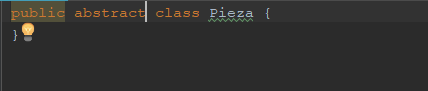
\*ConcreteState: cada uno de estos estados representa un posible estado por el cual la aplicación puede pasar. Esta clase debe de heredar de Abstract state



Para nuestro caso, a la hora de implementar el ejercicio de las bandejas utilizaremos este último método de implementación, ya que podremos distinguir cada uno de los estados(Empty, Normal, Full) en clases separadas, así como sus propias operaciones(put, get) en cada estado.

Implementación en Java:

Clase Pieza:

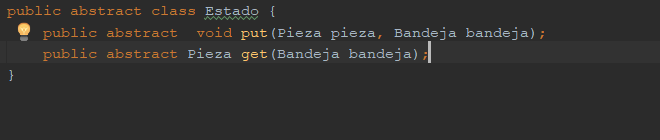


No tenemos más atributos de ella

Clase Bandeja:

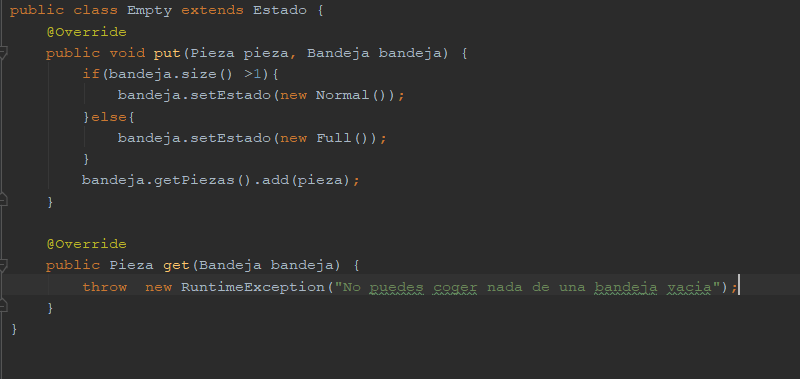


Clase abstract Estado:

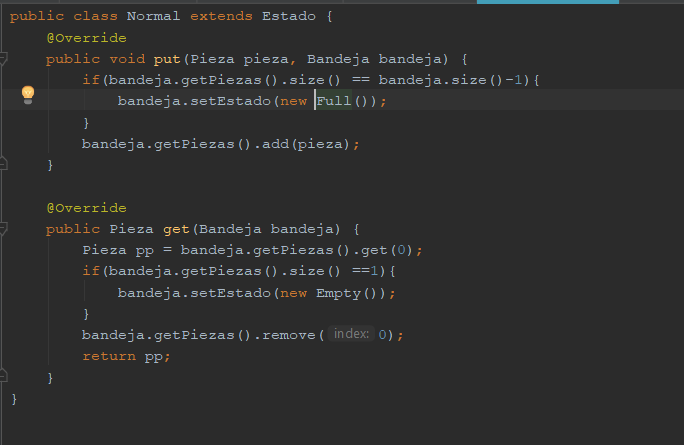


Clase base abstracta, con la imagen anterior se corresponde con AbstractState.

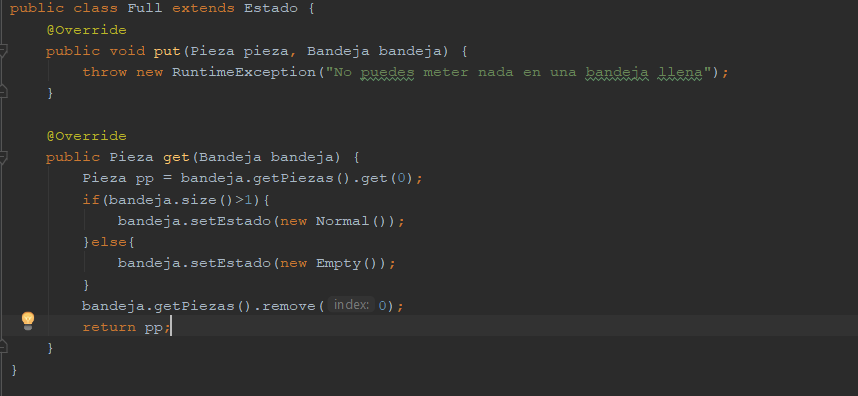
Clase Empty, estado:



Clase Normal, estado:



Clase Full, estado :



Todos estos estados, se corresponde con ConcreteState.