DISEÑO ORIENTADO A OBJETOS

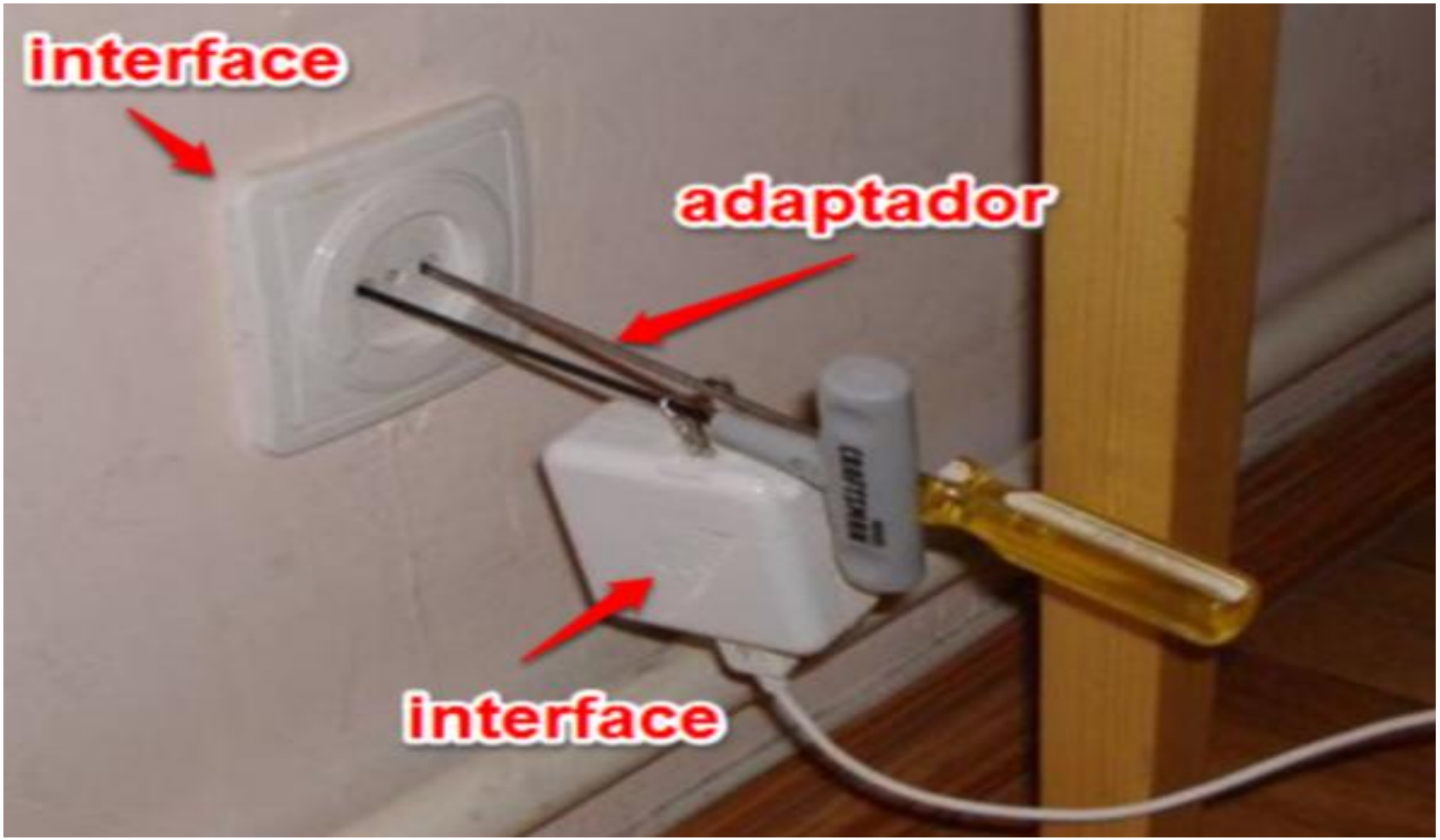
Prof. Esp. Ing. Agustín Fernandez

Patrones de Diseño (estructura)

- Patrón Adapter o Wrapper (Envoltorio)
  - Video

Patrones de Diseño (estructura)

- Patrón Adapter o Wrapper (Envoltorio)



Patrones de Diseño (estructura)

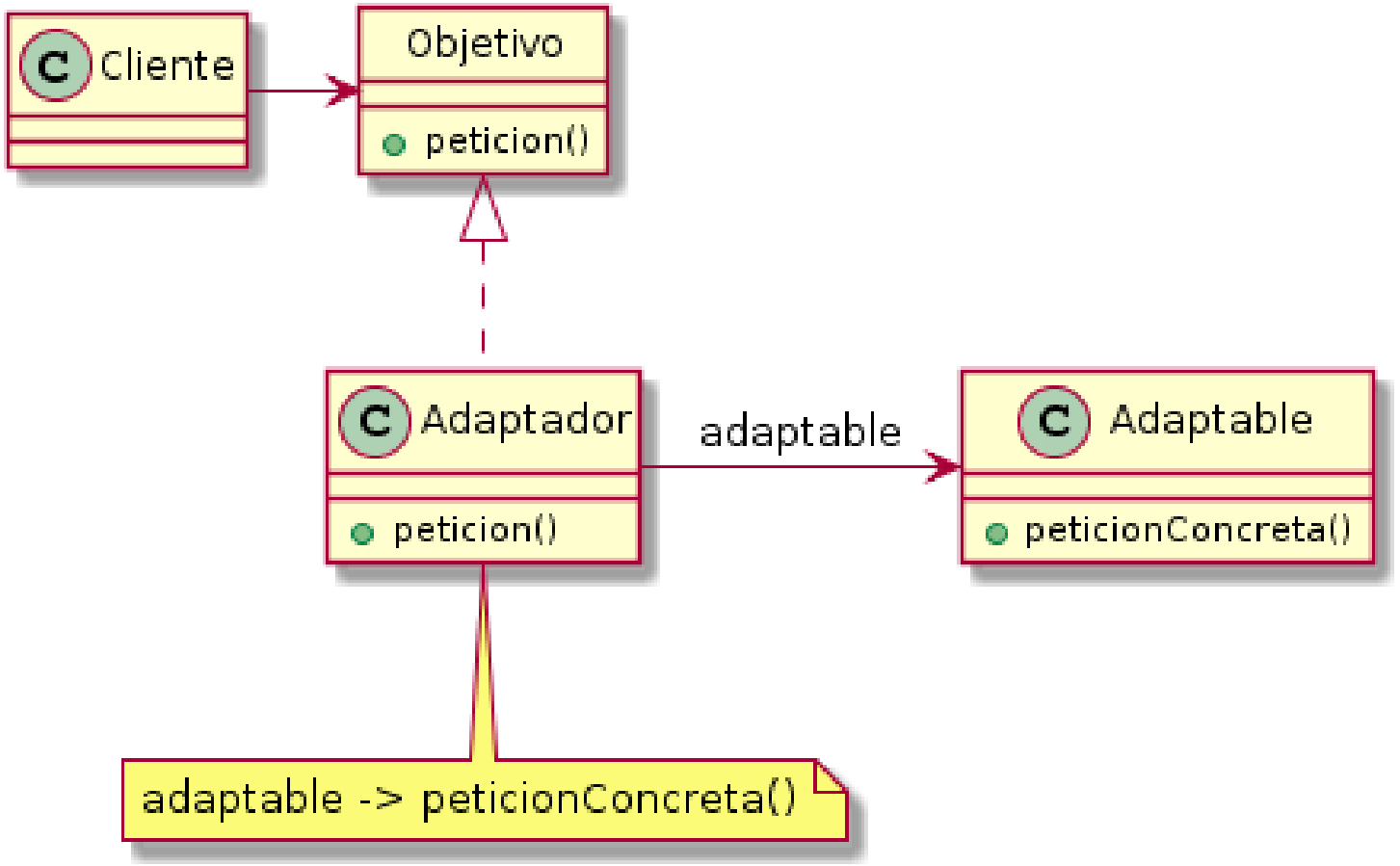
- Patrón Adapter o Wrapper (Envoltorio)
  - Convierte la interfaz de una clase en otra interfaz que es la que esperan los clientes.
  - Permiten que cooperen clases que de otra forma no podrían por tener interfaces diferentes.
  - Busca una manera estandarizada de adaptar un objeto a otro.
  - Muchas veces el adaptador es responsable de las funcionalidades que la clase adaptada no proporciona.
  - Se lo suele llamar Wrapper. Cabe aclarar que al patrón Decorator también es llamado Wrapper, por algunos, con lo cual el nombre Wrapper puede prestarse a confusión.

Patrones de Diseño (estructura)

- Patrón Adapter o Wrapper (Envoltorio)
- **Debe usarse este patrón cuando:**
  - Se requiere usar una clase existente y su interfaz no concuerda con la que se necesita.
  - Se requiere utilizar una clase para que coopere con clases no relacionadas, es decir, clases que no tienen por qué tener interfaces compatible.
  - Se requiere utilizar varias subclases existentes, pero como es poco práctico adaptar cada interfaz, se crea un objeto que adapte la interfaz de la clase padre.

Patrones de Diseño (estructura)

- Patrón Adapter o Wrapper (Envoltorio)
- **Estructura UML:**



Patrones de Diseño (estructura)

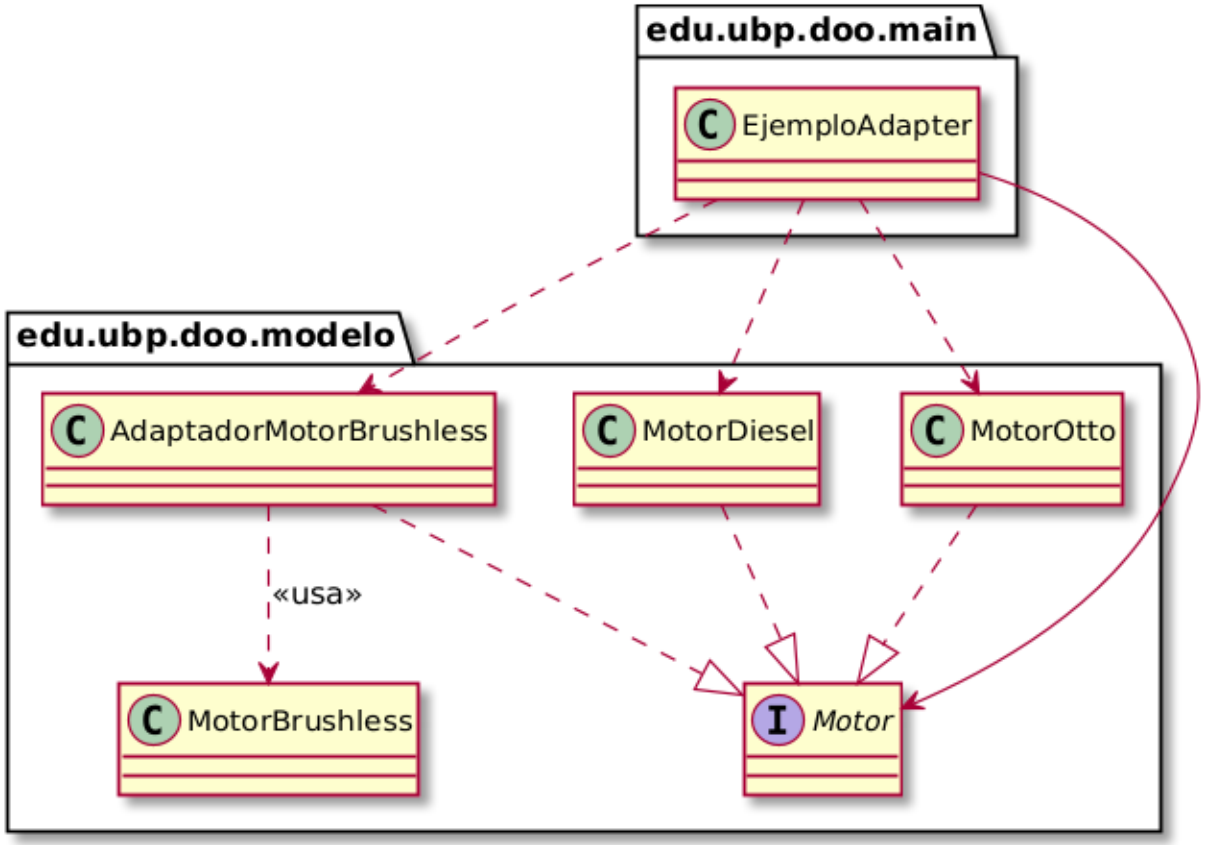
- Patrón Adapter o Wrapper (Envoltorio)
- **Estructura UML (Cont.):**
  - **Objetivo**: define la interfaz específica del dominio que Cliente usa.
  - **Cliente**: colabora con la conformación de objetos para la interfaz Objetivo.
  - **Adaptable**: define una interfaz existente que necesita adaptarse
  - **Adaptador o Adapter**: adapta la interfaz de Adaptable a la interfaz Objetivo

Patrones de Diseño (estructura)

- Patrón Adapter o Wrapper (Envoltorio)
- **Ejemplo:**
  - Supongamos que tenemos una clase “Cliente” que utiliza la Interfaz Motor que le permite Encender, Apagar y Acelerar un determinado motor. Cuando la Interfaz fue creada se la ideo para que pudiera funcionar con cualquier tipo de motor de combustión interna (Diesel u Otto). El inconveniente esta en que se ha adquirido un motor Brushless el cual no implementa la interfaz Motor por lo cual ha quedado “incompatible” con nuestra clase “Cliente” para que esta pueda Encenderlo, Apagarlo y Acelerarlo.
  - ¿Cómo resolvemos esto mediante Adapter para que la clase “Cliente” pueda manipular el motor Brushless?

Patrones de Diseño (estructura)

- Patrón Patrón Adapter o Wrapper (Envoltorio)
- **Ejemplo:**



Patrones de Diseño (estructura)

- Patrón Patrón Adapter o Wrapper (Envoltorio)
- **Ejercicio:** Imaginemos que tenemos una clase ModemUsb la cual implementa la interfaz IUsb para poder leer y enviar (“escribir”) datos. Además nos han prestado un módem un poco antiguo con conexión Rs232 el cual esta representado por la clase ModemRs232 y que a su vez implementa la Interfaz IRs232 para poder leer y enviar datos. Cuando se leen datos con IRs232 el proceso es el siguiente:
  - Cts();
  - Dtr();
  - Reading();
- Cuando se escriben datos usando IRs232 podemos decir que el proceso es:
  - Rts();
  - Dtr();
  - Writing();
- ¿Cómo podremos hacer para construir un adaptador y que nuestra clase “Cliente”, sin modificar la interfaz que usa actualmente (Iusb), pueda hacer uso del módem Rs232?

Patrones de Diseño (estructura)

- Patrón Patrón Adapter o Wrapper (Envoltorio)
- **Ejercicio:** Imaginemos que tenemos una clase Licuadora que para poder funcionar utiliza la clase EnchufeArgentino el cual a su vez implementa la interface IEnchufeEuropeo cuyas características son:
  - Voltaje: 220V
  - Cantidad de patas: 2
- Por nuestro trabajo hemos tenido que mudarnos a EEUU y decidimos llevar con nosotros a nuestra Licuadora, pero cuando llegamos notamos que allí los enchufes implementan otro tipo de interface denomina IEnchufeBritánico cuyas características son:
  - Voltaje: 110V
  - Cantidad de patas: 3
- Además la clase licuadora posee un método encender que no devuelve nada y que al ejecutarse muestra las características del voltaje y la cantidad de patas usadas.
- ¿Cómo podremos hacer para construir un adaptador y que nuestra
- licuadora siga funcionando en EEUU?

Patrones de Diseño (estructura)

- Patrón Patrón Adapter o Wrapper (Envoltorio)
- **Cosas a considerar:**
  - ¿Cuánto se debe adaptar? La cantidad de métodos a adaptar es proporcional a la interface que se debe soportar como Objetivo.
  - ¿El adaptador solo wrappea una clase? No, puede haber casos en que wrapee más de un Adaptable.

Patrones de Diseño (estructura)

- Patrón Patrón Adapter o Wrapper (Envoltorio)
- **Consecuencias:**
  - El Cliente y las clases Adaptable permanecen independientes unas de las otras.
  - Puede hacer que un programa sea menos entendible.
  - Permite que un único Adaptador trabaje con muchos Adaptables. El Adaptador también puede agregar funcionalidad a todos los Adaptables de una sola vez.

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
  - Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.
  - Define una interfaz de alto nivel que hace que el subsistema sea más fácil de utilizar.
  - Minimiza la comunicación y dependencia entre los participantes.

Patrones de Diseño (estructura)

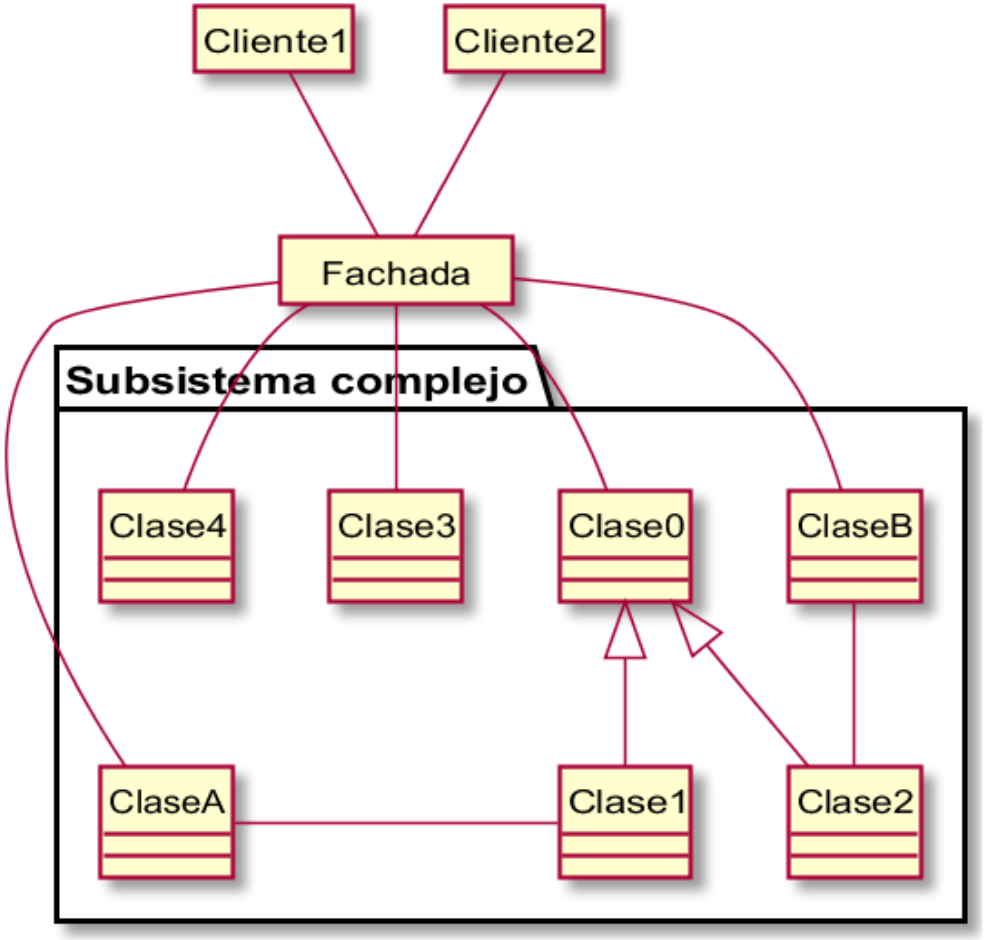
- Patrón Facade (Fachada)
- **Debe usarse este patrón cuando:**
  - Se aplicará el patrón fachada cuando se necesite proporcionar una interfaz simple para un subsistema complejo.
  - Se quiera estructurar varios subsistemas en capas, ya que las fachadas serían el punto de entrada a cada nivel.
  - Se necesita desacoplar un sistema de sus clientes y de otros subsistemas, haciéndolo más independiente, portable y reutilizable (esto es, reduciendo dependencias entre los subsistemas y los clientes).

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Usos comunes:**
  - Un cliente necesita acceder por separado a la funcionalidad de un sistema más complejo, por lo tanto nos conviene definir una interfaz que permita acceder solamente a esa funcionalidad
  - Por ejemplo si hay dependencia entre el código del cliente y la parte interna de una biblioteca, nos conviene crear un intermediario (fachada) y realizar llamadas a la biblioteca a través de él.
  - Necesitamos acceder a un conjunto de APIs que pueden además tener un diseño no muy bueno, por lo cual nos conviene crear una API intermedia, bien diseñada, que permita acceder a la funcionalidad de las demás.

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Estructura UML:**



Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Estructura UML (Cont.):**
  - **Fachada**: Sabe que clases del subsistema son las responsables ante una petición. Delega las peticiones de los clientes a los objetos apropiados del subsistema complejo.
  - **Clases del subsistema**: Implementan la funcionalidad del subsistema. Realizan las labores encomendadas por el objeto fachada. No conocen la fachada, es decir, no tienen referencia a ella.

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Colaboraciones:**
  - Los clientes se comunican con el subsistema enviando peticiones al objeto Fachada, el cual las reenvía a los objetos apropiados del subsistema.
  - Los objetos del subsistema realizan el trabajo final, y la fachada hace algo de trabajo para pasar de su interfaz a las del subsistema.
  - Los clientes que usan la fachada no tienen que acceder directamente a los objetos del subsistema.

Patrones de Diseño (estructura)

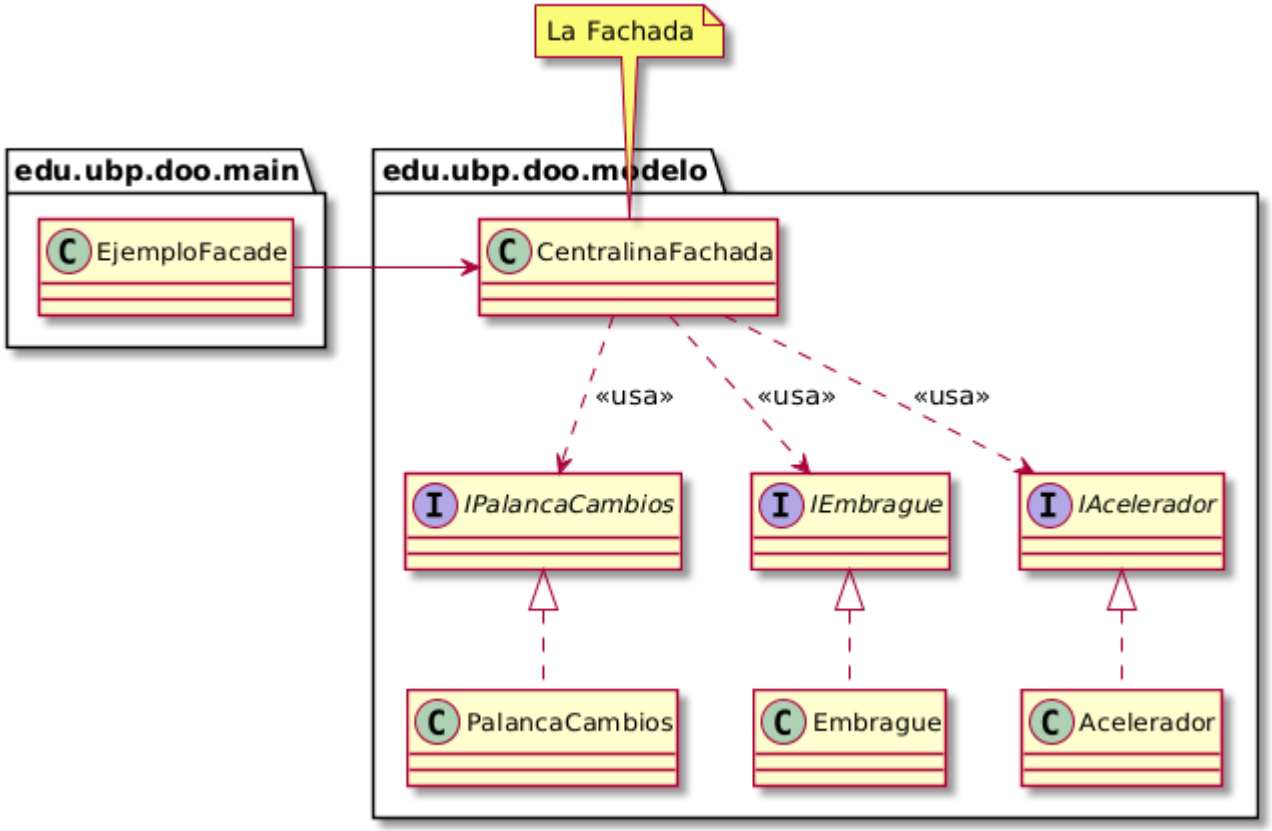
- Patrón Facade (Fachada)
- **Implementación:**
  - **Reducción del acoplamiento cliente-subsistema:** podemos reducir más el acoplamiento entre el cliente y el subsistema definiendo a Fachada como una clase abstracta con subclases concretas para las diferentes implementaciones del subsistema. Este acoplamiento abstracto evita que el cliente tenga que saber qué implementación de un subsistema esta usando.
  - **Clases del subsistema públicas o privadas:** La interfaz pública de un subsistema esta dada por la serie de clases a las que acceden los clientes; y la interfaz privada es sólo para quienes vayan a ampliar el subsistema. Por supuesto la Fachada es parte de la interface pública.

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Ejemplo:**
  - Sabemos que un conductor automovilístico usa cada uno de los subsistemas correspondientes (acelerador, embrague, palanca de cambios) a través de sus interfaces correspondientes para insertar o quitar una determinada velocidad al motor. El conductor manipula por separado a cada uno de estos subsistemas, es decir que para insertar una velocidad realiza lo siguiente:
    - acelerador.soltarAcelerador()
    - embrague.apretarEmbrague()
    - palancaCambios.pasarPuntoMuerto()
    - palancaCambios.cambiarVelocidad(3)
    - embrague.soltarEmbrague()
    - acelerador.apretarAcelerador()
  - ¿Cual sería la solución usando una “Fachada” que le permita al conductor aumentar o disminuir la velocidad sin realizar cada uno de estos pasos? (Centranila electrónica para caja de cambios automática)

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Ejemplo:**



Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Ejercicio:** Suponga que Ud. trabaja en el “Facade International Bank” (FIB) y se le ha pedido verificar si una persona, que ha solicitado un crédito, al FIB no posee deudas en Afip y en Veraz.
- Para poder realizar cada una de las verificaciones correspondientes Ud. debe realizar las comprobaciones del caso utilizando los sistemas de AFIP y VERAZ respectivamente.
- Además para poder otorgarle el crédito, es condición que la persona sea mayor de 18 años, cabe aclarar que esta última verificación sera realizada en el sistema del ANSES.
- ¿Ud. podría construir un “Fachada” de manera que al pasarle el DNI y la edad de la persona esta simplemente devuelva un true o false indicando si la persona es apta para el crédito o no?

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Consecuencias:**
  - Oculta a los clientes los componentes del subsistema, reduciendo así el número de objetos con los que tratan los clientes y haciendo que el subsistema sea más fácil de usar.
  - Proporciona un débil acoplamiento entre el cliente y el subsistema. Esto nos permite modificar los componentes de un subsistema sin que los clientes se vean afectados. La fachada nos ayuda a estructurar en capas a un sistema y las dependencias entre los objetos.
  - No impide que las aplicaciones usen las clases del subsistema en caso de que sea necesario.

Patrones de Diseño (estructura)

- Patrón Facade (Fachada)
- **Cosas a considerar:**
  - Facade define una nueva interfaz, mientras que Adapter utiliza una vieja interfaz. Se debe recordar que Adapter hace que dos interfaces existentes puedan trabajar juntas en lugar de crear una interfaz completamente nueva.
  - Adapter y Facade son ambos Wrappers. Pero ellos son diferentes tipos de Wrappers. La intención de Facade es producir una interfaz más simple y la intención de Adapter es reusar una interfaz existente.
  - Los objetos Facade son a menudo implementados como Singletons porque sólo un objeto Facade es necesario.