Definiciones:

**Ingeniería del software:** es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.

**Modelo:** una representación o especificación desde un determinado punto de vista y con un objetivo en concreto.

**Diseño:** conjunto de planes y decisiones para definir un producto con los suficientes detalles como para permitir su realización física de acuerdo a unos requisitos.

**Patrón de diseño:** solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software.

**Patrón Estado y patrón Estrategia**:

Ambos patrones son patrones de comportamiento de clases. Ambos patrones definen distintos comportamientos en subclases de una interfaz a la que la clase principal hace referencia, sin embargo, el patrón estrategia define y encapsula una familia de algoritmos intercambiables para el cliente, mientras que en el patrón estado, representa los estados en los que se encuentra y, dependiendo del estado en el que esté, tendrá un comportamiento u otro. La principal diferencia entre ambos es que el patrón estado no puede decidirse el estado en el que quieres que esté el objeto, puesto que eso depende de sus transiciones, sin embargo, el patrón estrategia te permite decidir el comportamiento que quieres que tenga tu clase.

**Refactoring:**

El refactor es el proceso de modificar la estructura interna de un código fuente sin modificar su comportamiento. La principal característica del refactor es aumentar la compresibilidad de un código mediante reestructuraciones y eliminación de código muerto para facilitar su mantenimiento en el futuro.

**Ejercicio 1:**

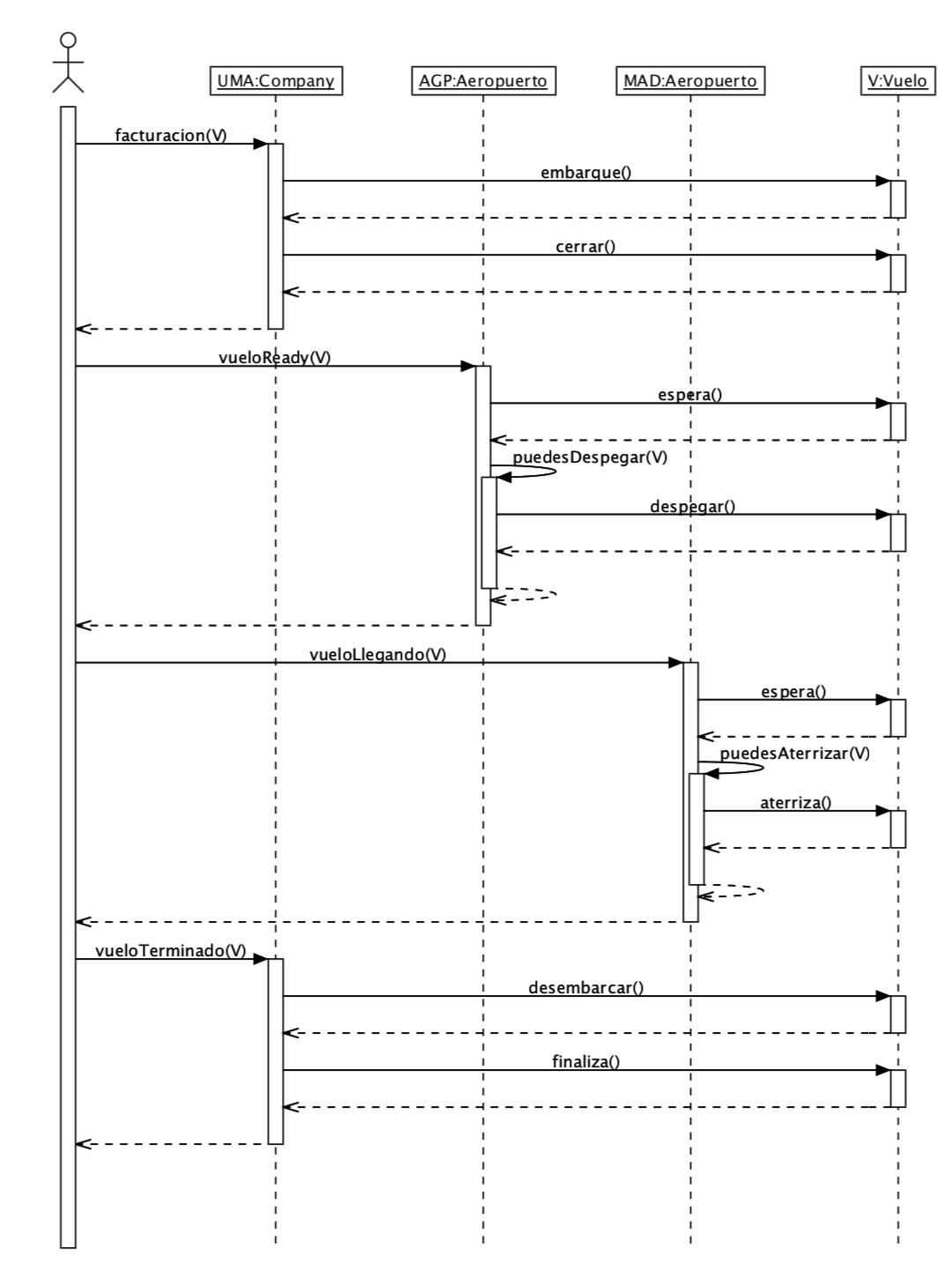
1. En el ejercicio se nos pide modelar un sistema de navegación aérea con unos requisitos. Hemos decidido crear una clase para el modelo del avión puesto que se nos especifica que tendrá un nombre y número de asientos, y para poder almacenar esa información, lo introduciremos en una clase. En cuanto a compañía aérea, aunque no contenga información como para considerarla clase, aparece repetidas veces a lo largo del sistema y para simplificar lo trataremos como una clase. Puesto que en el enunciado se nos dice que una etapa es un desplazamiento entre dos aeropuertos, crearemos 2 relaciones entre la clase Etapa y Aeropuerto para saber de qué aeropuerto sale y a qué aeropuerto va. Una etapa podrá salir sólo de un único aeropuerto y tendrá como destino otro aeropuerto. En el enunciado se menciona que el vuelo es una ocurrencia de un destino en una fecha y hora determinada, por tanto, se puede entender que un vuelo es una materialización del destino. Ocurre lo mismo con Trayecto, que es una materialización de una etapa para un avión en concreto y una compañía, sin embargo, no añadiremos una relación entre trayecto y compañía ya que hemos supuesto que un avión pertenece a una única compañía, y a través del avión, podemos saber qué compañía realiza dicho trayecto.

Diagrama UML en USE

1. Para poder realizar el diagrama de secuencia con todas las llamadas a operaciones para el comportamiento vamos a necesitar las siguientes operaciones:

en la clase Company añadir la operación *facturacion()* que representa el inicio del vuelo. Cuando la facturación termine, comienza el embarque y se llama a la operación *embarque()*de la clase vuelo, una vez terminado, la clase Company llama a la función cerrar() del vuelo. A continuación, el CV llama a la operación vueloReady(v: Vuelo) para que la torre de control le dé el permiso para despegar, mientras, le dirá al vuelo que espere con la operación espera() de la clase vuelo. Una vez se le dé la confirmación para volar, se llama a la operación puedesDespegar(v: Vuelo) que le permite al vuelo despegar con su operación despegar(). Una vez se esté acercando al otro aeropuerto, se llama al método vueloLlegando(v: Vuelo) que le dice al vuelo que espere, con espera(), y acto seguido se le da permiso para aterrizar con puedesAterrizar(v: Vuelo) que llama a v.aterriza(). una vez aterrice, el CV llama a la operación vuelo terminado, que hará que desembarquen los pasajeros, y acto seguido, finalizará el vuelo.

Diagrama de secuencia



1. Para el diagrama de estados hemos decidido poner los siguientes estados para modelar el proceso del vuelo. Parado (que es el estado inicial), Embarcando, Listo, Esperando, Volando, Esperando, Aterrizado, Desembarcando, Finalizado. Con las operaciones anteriores realizamos las transiciones. Las transiciones entre estados son tal y como los he mencionado, exceptuando al llegar a Volando, que vuelve a Esperando (para aterrizar) y luego aterriza.

**Ejercicio 2:**

1. Para solucionar este apartado, lo primero que vamos a hacer es quitar el enumerado de la asociación y reemplazarlo por una clase llamada *Participación*. Luego, se relaciona la clase Participación con el enumerado de manera que la direccionalidad vaya hacia el enumerado. Una vez realizado esto, reificamos la asociación N-aria. Una vez terminado esto, tendríamos las clases listas para pasar fácilmente a Java.
2. Para añadir el porcentaje de horas que le dedica un empleado a un proyecto en un cargo en concreto, lo añadiría en la clase Participación, puesto que tiene toda esta información de manera inmediata, ya que tiene una referencia al enumerado con el cargo que tiene un único empleado en un proyecto en concreto.
3. Esta operación dependerá de la fórmula de los cargos que tenga. Símplemente se recorre la lista de todas las participaciones de un empleado y para todos los proyectos se coge el porcentaje de participación, el cargo que ocupó y el bonus del proyecto. Un posible esquema de la solución sería el siguiente:

Hay que tener en cuenta que calcularParteCorrespondiente() es una función privada definida dentro de la clase para calcular la parte que le corresponde a este empleado, puesto que no conocemos como se calcula el bonus, se deja para que sea implementado en un futuro.

Euro bonus() {

double bonus;

for(Participacion p: participaciones) {

bonus += calcularParteCorrespondiente(p.proyecto.bonus, p.cargo, p.porcentaje);

}

return new Euro(bonus);

}