1 Relación de ejercicios. *Tema 1*

1. Aplicar el patrón `Abstracción-caso` en las siguientes situaciones. Para cada una de estas situaciones, representar las dos clases ligadas, la asociación entre las clases y los atributos de cada clase.
   (a) Los números de una revista.
   (b) Las copias de los números de una revista.
   (c) Las repeticiones y re-emisiones de una misma serie de televisión.
   (d) Modelos de electrodomésticos y cada electrodoméstico.

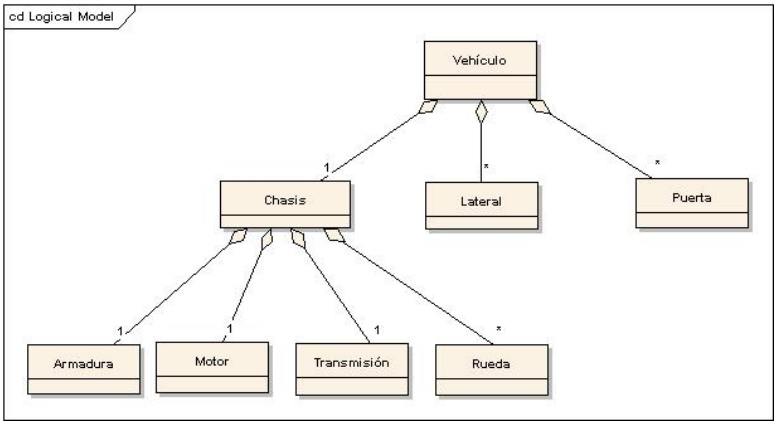

Figura 1: Jerarquía de partes de un coche

2. La figura 1 muestra una jerarquía de partes de un vehículo. Mostrar cómo se podría representar mejor esta jerarquía utilizando el patrón `Jerarquía General`, o de manera más precisa, con el patrón `Composite` de [Gamma et al. 2009].[1]

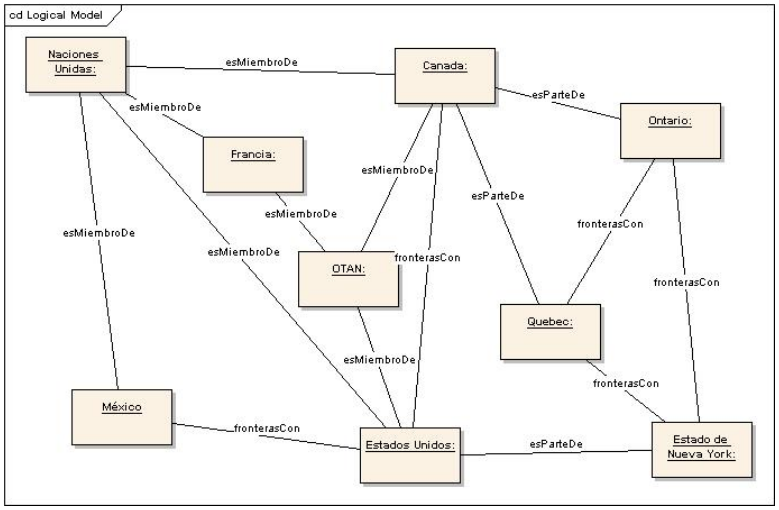

Figura 2: Relaciones entre organismos estatales

3. Representar un diagrama de clases que pueda generar el diagrama de objetos de la figura 2 eligiendo las multiplicidades de forma razonable. ¿Piensas utilizar el patrón `Jerarquía`

[1] Este patrón compone objetos en estructuras con forma de árbol para representar jerarquías *parte–todo* Permite a las objetos–cliente tratar a los objetos individuales y a las composiciones de objetos de manera uniforme

`General` para realizar el diagrama de clases pedido? En caso contrario, rehacer el ejercicio, mostrando una jerarquía que incluya a admistraciones de varios niveles. Suponer que los ayuntamientos son el nivel más bajo de gobierno.

4. Una expresión en Java se puede descomponer en una jerarquía de subexpresiones. Por ejemplo, `(a/(b+c))+(b-func(d)*/e/(f+g))` `tiene a` `(a/(b+c))` como una de sus subexpresiones de nivel más alto.

   (a) Utilizando uno de los patrones `Jerarquía General` o `Composite`, crear un diagrama de clases para representar expresiones en Java. Ayuda: (a) una expresión de alto nivel podría tener más de dos partes, (b) las partes de una expresión de alto nivel están conectadas mediante operadores ¿cómo se puede representar esto? (c)pensar lo que serían ahora los elementos "`NodoNoSuperior`" de la plantilla del patrón `Jerarquía General` visto en clase (d) Algunas expresiones están encerradas entre paréntesis, mientras que otras no lo necesitan, ¿Cómo se puede representar esto?

   (b) Utilizar el diagrama de clases realizado anteriormente para crear ahora un diagrama de objetos del mismo ejemplo.

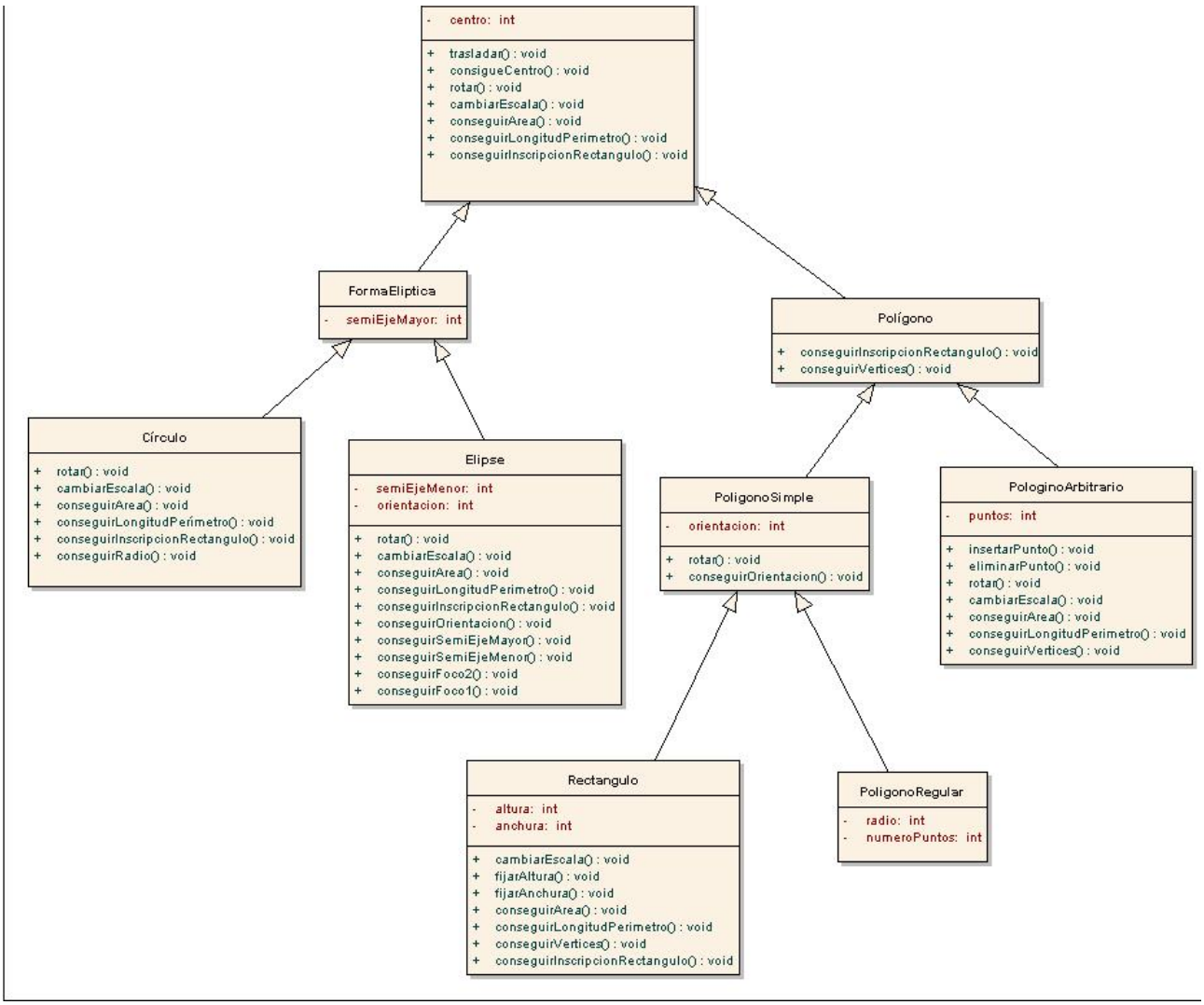

Figura 3: Jerarquía de formas con polimorfismo y sobrecarga

5. Representar un diagrama de clases aplicando el patrón `Actor-papel` en las siguientes representaciones de:
   (a) Los usuarios con distintos privilegios de acceso a un sistema.
   (b) Directivos a los que se les da una o más responsabilidades.
   (c) Los jugadores de tu deporte favorito (piensa uno en el que puedan ocupar los jugadores diferentes posiciones durante un partido, o en diferentes partidos).

6. Discutir cómo se puede generalizar el patrón `Singleton` al caso en que una clase esté limitada a tener un máximo de N instancias.

7. Revisar la documentación Java y explicar las similaridades o diferencias entre el mecanismo que implican las clases `ActionEvent` y `ActionListener` y el patrón `Observer`.

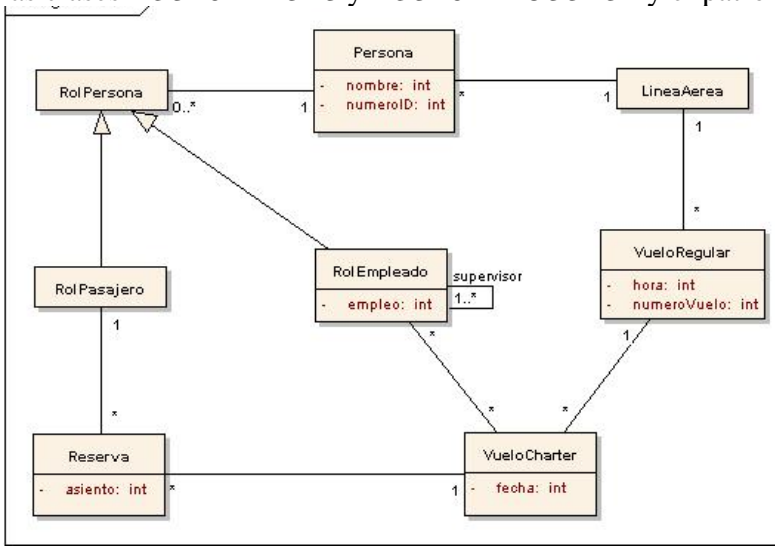

Figura 4: Sistema para Línea Aérea

8. Utilizar el patrón `Observer` para modelar un sistema pequeño en el que varias clases distintas deberían ser notificadas cada vez que se añade o se elimina un item del inventorio.

9. Encontrar todas las situaciones en las que el patrón `Delegación` podría ser aplicado para desarrollar un modelo que representa a un árbol genealógico familiar.

10. Explicar si la clase `MouseAdapter` definida en Java puede ser considerada (o no) una implementación del patrón de diseño `Adaptador`.

11. Suponer que se quiere ser capaz de utilizar distintos sistemas de bases de datos en las diferentes secciones de código de una aplicación. Para facilitar el intercambio de bases de datos, se creará una interfaz `Fachada` específica además de las clases `Fachada` asociadas con cada cada sistema de bases de datos concreto. Dibujar el diagrama de clases correspondiente.

12. Suponer que todas las clases de la figura 3 fuesen *inmutables* ¿Qué otros métodos podrían ser añadidos al sistema para que devolvieran instancias de una clase *distinta* de la clase en la que fueron escritos?

13. Crear una versión de sólo lectura de la clase `VueloCharter` mostrada en las figuras 4, 5 de tal manera que se puedan pasar instancias de dicha clase de forma segura a otros subsistemas.

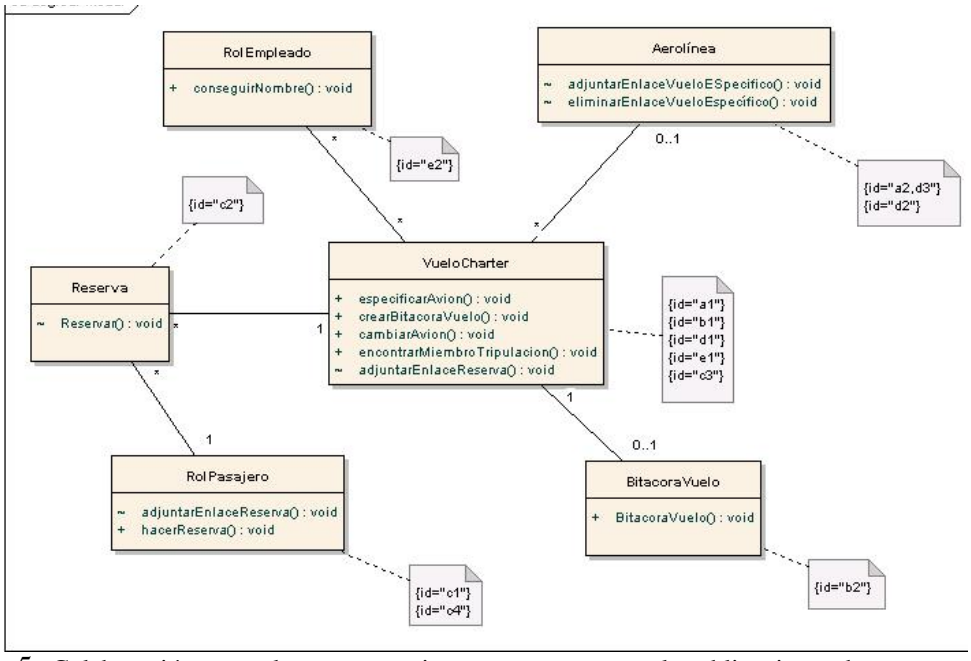

Figura 5: Colaboración entre clases y operaciones para representar las obligaciones de `VueloCharter`

14. Discutir las ventajas de utilizar una *imagen–proxy* cuando se manipulan las fotos en una aplicación que implemente un álbum de fotos digital. ¿Qué operaciones, que no necesiten cargar una imagen *pesada* en memoria, podría pensarse que realizaría el proxy aludido?

15. En un determinado juego, los animales carnívoros y herbívoros se crean en momentos aleatorios por el *motor* del juego. Dependiendo del país seleccionado por el usuario, se cargaría una factoría para crear los animales apropiados para cada uno de estos (por ejemplo, se crearían leones y gacelas en África, pumas y castores en Canada). Dibujar el diagrama de clases que represente este concepto de diseño para la aplicación.

16. Encontrar el patrón de diseño más apropiado para ser aplicado en los siguientes problemas:

    (a) Se está construyendo una jerarquía de herencia de productos que vende la empresa en la que trabajamos; sin embargo, queremos reutilizar algunas clases de alguno de los suministradores. No podemos modificar las clases de los suministradores. ¿Cómo podíamos asegurarnos que las clases suministradoras puedan seguir utilizándose con polimorfismo?

    (b) Queremos permitir operaciones invocadas sobre instancias de `PoligonoRegular` que distorsionen objetos–polígonos, de tal forma que dejen de ser regulares ¿Cómo hacemos para permitir la ejecución de dichas operaciones sin que se levanten excepciones en el programa?

    (c) Nuestro programa manipula imágenes que usan mucho espacio en memoria. ¿Cómo podemos diseñar un programa que sólo cargue las imágenes en memoria cuando las necesite y, en otro caso, que sólo puedan ser recuperadas de archivos?

    (d) Hemos creado un subsistema con 25 clases. Sabemos que la mayoría de los otros subsistemas accederán, en promedio, a sólo 5 métodos en el subsistema aludido ¿Cómo podemos simplificar la vista que poseen los otros subsistemas de nuestro subsistema?

    (e) Estamos desarrollando una marco de trabajo para usarlo en la presentación de las cotizaciones de las acciones en bolsa. Algunas aplicaciones que lo usarán querrán obtener las cotizaciones en una pantalla tan pronto estén disponibles; otras aplicaciones querrán que

nuevas cotizaciones activen determinados cálculos financieros y otras aplicaciones querrán ambas cosas además de hacer que las cotizaciones sean transmitidas sin cables a una red de mensáfonos (o *buscas*). ¿Cómo podemos diseñar el marco de trabajo aludido de tal forma que varias partes diferentes del código de la aplicación pudieran reaccionar de manera independiente a la llegada de nuevas cotizaciones?

17. La interfaz `Iterator`, tal como se define en Java, es la implementación de lo que se denomina un patrón de diseño *Iterador*. Estudiar la documentación Java que describe al patrón aludido para después, utilizando el formato discutido en este capítulo, escribir una descripción del patrón *Iterador*, con secciones que definan su contexto, problemas, fuerzas y solución (tal como se han especificado las plantillas de los patrones).

18. Para mejorar el acceso a la información almacenada en una base de datos, algunas aplicaciones utilizan el concepto de *cache*. El principio básico consiste en mantener objetos en memoria local (que de otro modo serían normalmente destruidos), ya que dichos objetos se espera que sean demandados más tarde. De esta manera, cuando se vuelva a necesitarlos, su acceso se producirá con mucha rapidez.
    (a) Crear un patrón de diseño que materialice esta idea. Utilizar el formato de descripción de patrones que se ha visto en clase, incluyendo los diagramas de clases.
    (b) Buscar en la literatura específica el patrón denominado `Cache Management` y compararlo con la solución encontrada.