

Parte II Modelado y Programación Orientada a Objetos

En esta segunda parte se describirá la programación orientada a objetos desde dos perspectivas distintas. La primera es el *modelado* (Capítulo 4), una descripción teórica de los conceptos básicos de la orientación a objetos, en nuestro caso utilizando la notación UML (*Unified Modeling Language*). La segunda es la *programación* (Capítulo 5), una descripción práctica basada en el modelado orientado a objetos, en nuestro caso utilizando el lenguaje Java.

4 Modelado con UML

El modelado, o *modelo de objetos*, describe los conceptos principales de la orientación a objetos: las estructuras estáticas y sus relaciones. Las principales estructuras estáticas son los *objetos* y *clases*, los cuales están compuestos de *atributos* y *operaciones*, mientras que las principales relaciones entre objetos y entre clases corresponden a las *ligas* y *asociaciones*, respectivamente. Estos temas y otros serán descritos en este capítulo, en término de los *objetos*, *clases*, *atributos*, *operaciones*, *asociaciones*, *composición*, *herencia* y *módulos*.

4.1 Objetos

Los *objetos* son las entidades básicas del modelo de objeto. La palabra objeto proviene del latín *objectus*, donde *ob* significa *hacia*, y *jacere* significa *arrojar*; o sea que teóricamente un *objeto* es cualquier cosa que se pueda arrojar.

Ejemplo: Una *pelota* o un *libro* se pueden arrojar, por lo tanto estos son objetos. Por otro lado, un *avión* o un *elefante* también se consideran objetos, aunque sean bastante pesados para ser arrojados.

Los objetos son más que simples cosas que se puedan arrojar, son conceptos pudiendo ser abstractos o concretos.

Ejemplo: Una *mesa* es un objeto concreto, mientras que un *viaje* es un objeto abstracto.

Los objetos corresponden por lo general a sustantivos, pero no a gerundios.

Ejemplo: *Mesa* y *viaje* son ambos sustantivos y por lo tanto objetos. *Trabajando* y *estudiando* son gerundios por lo cual no se consideran objetos.

Cualquier cosa que incorpore una estructura y un comportamiento se le puede considerar como un objeto.

Ejemplo: Una *pelota* es sólida y redonda y se le puede arrojar o atrapar. Un *libro* es rectangular y sólido y se le puede abrir, cerrar, y leer.

Un objeto debe tener una identidad coherente, al que se le puede asignar un nombre razonable y conciso.

Ejemplo: Se consideran *manzanas* todas las frutas con un sabor, textura, y forma similar.

La existencia de un objeto depende del contexto del problema. Lo que puede ser un objeto apropiado en una aplicación puede no ser apropiado en otra, y al revés. Por lo general, existen muchos objetos en una aplicación, y parte del desafío es encontrarlos.

Ejemplo: La *temperatura* se puede considerar un objeto abstracto, teniendo propiedades tales como el valor de la temperatura y el tipo de la escala en que se mide (Celsius o Fahrenheit). Por otro lado, si hablamos de un *termómetro*, la temperatura pasa a ser una propiedad del termómetro.

Los objetos se definen según el contexto de la aplicación.

Ejemplo: Una persona llamada *Juan Pérez* se considera un objeto para una compañía, mientras que para un laboratorio el *hígado* de Juan Pérez es un objeto. Una universidad como la *ITAM* se considera un objeto, mientras que dentro de la *ITAM* los objetos serían las *aulas*, los *estudiantes* y los *profesores*.

Los objetos deben ser entidades que existen de forma independiente. Se debe distinguir entre los objetos, los cuales contienen características o propiedades, y las propias características.

Ejemplo: El *color* y la *forma* de una *manzana* no se consideran propiamente objetos, sino propiedades del objeto *manzana*. El *nombre* de una *persona* se considera una propiedad de la *persona*.

Un grupo de cosas puede ser un objeto si existe como una entidad independiente.

Ejemplo: Un *automóvil* se considera un objeto el cual consiste de varias partes, como el *motor* y la *carrocería*.

Los objetos deben tener nombres en singular, y no en plural.

Ejemplo: Un *automóvil* es un objeto, *automóviles* son simplemente muchos objetos y no un solo objeto.

Parte de una cosa puede considerarse un objeto.

Ejemplo: La *rueda*, la cual es parte del *automóvil*, se puede considerar un objeto. Por otro lado, el *lado izquierdo del automóvil* sería un mal objeto.

Los objetos deben tener nombres razonables y concisos para evitar la construcción de objetos que no tengan una identidad coherente.

Ejemplo: Datos o información no son nombres concisos de objetos. Por otro lado, un estudiante es un objeto, ya que contiene propiedades como el número de matrícula y nombre del estudiante, además incluye un comportamiento tal

como ir a clases, presentar exámenes, y graduarse. Todos los estudiantes cuyos apellidos comiencen con "A" sería un mal objeto ya que el nombre del objeto no es conciso.

El objeto integra una estructura de datos (*atributos*) y un comportamiento (*operaciones*).

4.1.1 Diagramas de Objetos

Los objetos se describen gráficamente por medio de un *diagrama de objetos* o *diagrama de instancias*.

La notación general para una objeto es una caja rectangular conteniendo el nombre del objeto subrayado, el cual sirve para identificar al objeto, como se muestra en la Figura 4.1.

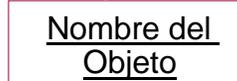


Figura 4.1. Notación para un objeto.

Ejemplo: Los objetos *Juan Pérez* y *ITAM* se muestran en la Figura 4.2.



Figura 4.2. Notación para los objetos *Juan Pérez* y *ITAM*.

4.1.2 Identidad

Los objetos se distinguen por su propia existencia, su *identidad*, aunque internamente los valores para todos sus datos sean iguales.. Todos los objetos se consideran diferentes.

Ejemplo: Si tenemos una biblioteca llena de libros, cada uno de esos libros, como *La Ilíada*, *Hamlet*, *La Casa de los Espíritus*, etc., se consideran e identifican como objetos diferentes. Dos *manzanas* aunque sean exactamente del mismo color y forma, son diferentes objetos.

Los objetos tienen un nombre que puede no ser único.

Ejemplo: Pueden existir múltiples copias de un solo libro, lo cual requiere identificadores especiales para distinguir entre diferentes objetos con propiedades similares, como el código del libro en la biblioteca.

Los objetos necesitan un *identificador interno único* cuando son implementados en un sistema de computación para acceder y distinguir entre los objetos. Estos identificadores no deben incluirse como una propiedad del objeto, ya que solo son importantes en el momento de la implementación.

Ejemplo: Los diferentes *personas* se distinguirían internamente dentro de una computadora por los identificadores *Persona1*, *Persona2*, *Persona3*, etc. Por otro lado, el *número del seguro social* de la *persona* es un identificador externo válido, ya que existe fuera de la implementación en una computadora.

4.2 Clases

Una *clase* describe un grupo de objetos con estructura y comportamiento común. (*Clase* y *tipo* no son necesariamente equivalentes, *tipo* se define por las manipulaciones que se le puede dar a un objeto dentro de un lenguaje y *clase* involucra una estructura, pudiendo corresponder a una implementación particular de un tipo. En el capítulo 5 se hablará más de esto.)

Las estructuras o propiedades de la clase se conocen como *atributos* y el comportamiento como *operaciones*. Una clase define uno o más objetos, donde los objetos pertenecen a la clase, teniendo características comunes.

Ejemplo: *Juan Pérez* y *María López* se consideran miembros de la clase *persona*, donde todas las personas tienen una edad y un nombre. El *ITAM* y la *UNAM* pertenecen a la clase *universidad*, donde todas las universidades tienen una dirección y un grado máximo. *Chrysler* y *Microsoft* pertenecen a la clase *compañía*, donde todas las compañías tienen una dirección, un número de empleados, y una ganancia al año.

Una clase se considera un "molde" del cual se crean múltiples objetos.

Ejemplo: La clase es como un molde de una cerámica de la cual se pueden crear múltiples cerámicas, todas con exactamente las mismas características. Para modificar las cerámicas hay que primero construir un nuevo molde.

Al definir múltiples objetos en clases se logra una abstracción del problema. Se generaliza de los casos específicos definiciones comunes, como nombres de la clase, atributos, y operaciones.

Ejemplo: Los objetos impresora a láser, impresora de burbuja, e impresora de punto son todos objetos que pertenecen a la clase impresora.

Una clase como se ha definido en esta sección se conoce también como *clase básica*.

4.2.1 Diagramas de Clases

Las clases se describen por medio del *diagrama de clases*. La notación para una clase es una caja rectangular conteniendo el nombre de la clase con letras **negritas**, como se muestra en la Figura 4.3.

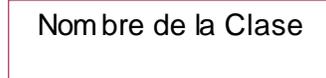


Figura 4.3. Notación para una clase.

Ejemplo: Las clases *Persona* y *Universidad* se muestran en la Figura 4.4.



Figura 4.4. Notación para las clases *Persona* y *Universidad*.

La notación general para el objeto se extiende mediante el nombre de la clase subrayado seguido al nombre del objeto, como se muestra en la Figura 4.5.

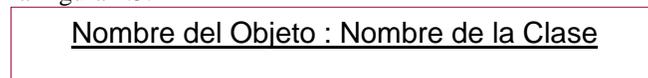


Figura 4.5. Notación para un objeto incluyendo el nombre de la clase.

Ejemplo: Los objetos *Juan Pérez* y *ITAM* se muestran en la Figura 4.6 incluyendo el nombre de sus respectivas clases *Persona* y *Universidad*.



Figura 4.6. Notación para los objetos *Juan Pérez* y *ITAM* incluyendo el nombre de la clase.

Por lo general, se utilizan más los diagramas de clases que los diagramas de objetos, ya que los diagramas de clases son más generales, y corresponde a varios diagramas de objetos.

4.2.2 Instanciación

El proceso de crear objetos pertenecientes a una clase se conoce como *instanciación*, donde los objetos son las *instancias* de la clase. El objeto es la instancia de la clase a la que pertenece.

Se utiliza un flecha punteada para mostrar los objetos como instancias de las clases, como se muestra en la Figura 4.7.

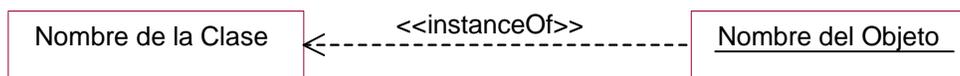


Figura 4.7. Notación para instanciación de objetos.

Ejemplo: *Juan Pérez* y *María López* son instancias de la clase *Persona*, como se muestra en la Figura 4.8.

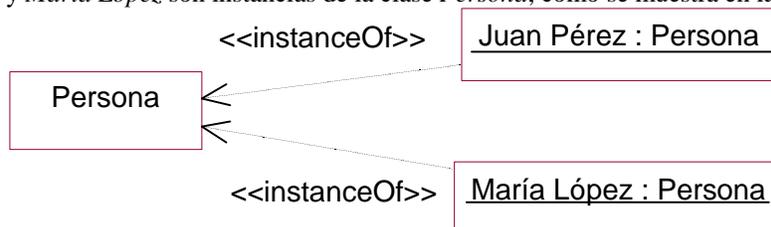


Figura 4.8. Notación para la instanciación de objetos de la clase *Persona*.

Pueden ser instanciados un número indefinido de objetos de cierta clase.

4.3 Atributos

Los *atributos* definen la estructura de una clase y de sus correspondientes objetos. El atributo define el valor de un dato para todos los objetos pertenecientes a una clase.

Ejemplo: *Nombre, edad, peso*, son atributos de la clase *persona*. *Color, precio, modelo*, son atributos de la clase *automóvil*.

Los atributos corresponden a sustantivos y sus valores pueden ser sustantivos o adjetivos.

Ejemplo: *Nombre, edad, color*, son sustantivos. *Juan, 24*, son sustantivos, y *verde* es un adjetivo.

Se debe definir un valor para cada atributo de una clase. Los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase.

Ejemplo: el valor del atributo *edad* puede ser "24" para los objetos Juan Pérez y María López, y "15" para Ramón Martínez.

Dentro de una clase, los nombre de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases).

Ejemplo: Las clases *persona* y *compañía* pueden tener ambas un atributo *dirección*, en cambio no pueden existir dos atributos llamados *dirección* dentro de la clase *persona*.

Los atributos no tienen ninguna identidad, al contrario de los objetos.

Ejemplo: Los atributos *nombre* y *edad* de la clase *persona* tienen valores simples. El valor para *nombre* puede ser "Juan" o "María", mientras que el valor para *edad* puede ser "17" o "25". (Nótese que pudieran existir dos objetos distintos con exactamente el mismo *nombre* y *edad*, donde estos identificarían dos personas distintas.)

Un atributo como se ha definido en esta sección se conoce también como *atributo básico*. Los atributos se listan en el diagrama de clases a continuación del nombre de la clase, en una segunda sección, como se muestra en la Figura 4.9.

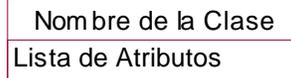


Figura 4.9. Diagrama de clases conteniendo atributos.

Ejemplo: En la Figura 4.10 se muestran dos atributos, *Nombre* y *Edad*, para la clase *Persona*.

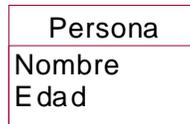


Figura 4.10. Diagrama de clases, para la clase *Persona*, conteniendo los atributos, *Nombre* y *Edad*.

La notación para el diagrama de objetos incluye los valores para los atributos que se ubican en el centro de la caja en letra normal a continuación del nombre de la clase. Existen dos notaciones alternas:

?? La notación extendida se muestra en la Figura 4.11.

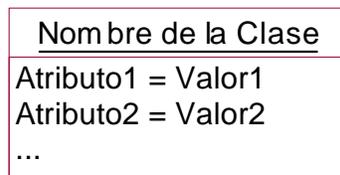


Figura 4.11. Notación extendida del diagrama de instancias, para objetos conteniendo valores de atributos.

Ejemplo: En la Figura 4.12 se muestran dos objetos de tipo *Persona*, utilizando la notación extendida. Los valores de los dos objetos para el atributo *Nombre*, son *María López* y *Juan Pérez*, y para el atributo *Edad*, 24 y 21, respectivamente. (Los valores para los atributos pueden ser o no iguales.)



Figura 4.12. Diagrama de instancias, para dos objetos de tipo *Persona*, conteniendo valores de atributos, usando la notación extendida.

?? La notación compacta se muestra en la Figura 4.13.

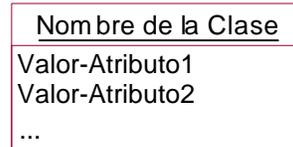


Figura 4.13. Notación compacta del diagrama de instancias, para objetos conteniendo valores de atributos.

Ejemplo: En la Figura 4.14 se muestran dos objetos de tipo *Persona*, utilizando la notación compacta. Los valores de los dos objetos para el atributo *Nombre*, son *María López* y *Juan Pérez*, y para el atributo *Edad*, *21* y *24*, respectivamente. En esta notación, es muy importante mantener el mismo orden de como han sido definido los atributos en el diagrama de clases.



Figura 4.14. Diagrama de instancias, para dos objetos de tipo *Persona*, conteniendo valores de atributos, usando la notación compacta.

Se puede asociar con cada atributo un *tipo* de dato, por ejemplo, entero, cadena, etc., para restringir sus posibles valores. El tipo se añade, separado por dos puntos, al diagrama de clases inmediatamente después del nombre del atributo. También se puede definir un valor de *omisión* para cada atributo, o sea un valor que se asigna en caso de no haberse especificado uno, el cual se añade separado por un signo de "igual", a continuación del tipo de dato. La notación se muestra en la Figura 4.15. (El diagrama de objetos no varía con respecto a esta información adicional.)

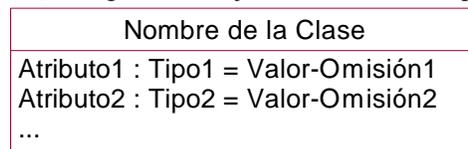


Figura 4.15. Notación extendida para diagrama de clases conteniendo atributos.

Ejemplo: En la Figura 4.16 se muestran los dos atributos de la clase *persona*, *nombre* y *edad*, donde *nombre* está definido como una *cadena* de caracteres, con valor de omisión " ", mientras que *edad* está definido como un *entero*, con valor de omisión "0".

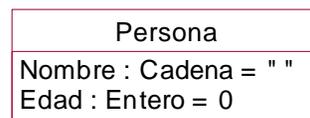


Figura 4.16. Diagrama de clases, para la clase *persona*, conteniendo atributos con definición de tipo de datos y valores de omisión.

La notación compacta sería análoga a la anterior. (El detalle que se muestra en cualquier diagrama de puede variar.)
Ejemplo: Los diagramas de clases que se muestran en la Figura 4.17 son todos correctos, variando según el detalle deseado en la descripción.



Figura 4.17. Diagrama de clases, para la clase *Persona*, conteniendo diferente nivel de detalle.

4.3.1 Identificadores

En el momento de incluir atributos en la descripción de una clase se debe distinguir entre los atributos los cuales reflejan las características de los objetos en el mundo real, y los identificadores los cuales son utilizados exclusivamente por razones de implementación. Estos identificadores internos del sistema no deben ser incluidos como atributos.

Ejemplo: *Número del Seguro Social*, o *número de la licencia de conducir*, son identificadores válidos del mundo real, en cambio un *identificador* para distinguir entre objetos de tipo persona no se debe incluir en el diagrama. En la Figura 4.18 se muestra la forma incorrecta de incluir un identificador ("identificador : ID") en la clase del objeto, seguido por la forma correcta (omitido).

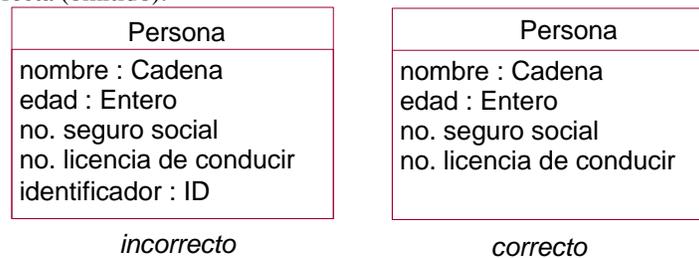


Figura 4.18. Diagrama de clases mostrando de forma incorrecta la inclusión de un atributo *identificador*, seguido por la forma correcta.

4.3.2 Atributos Derivados

Los *atributos básicos* son atributos independientes dentro del objeto. En contraste, los *atributos derivados* son atributos que dependen de otros atributos. Los atributos derivados dependen de otros atributos del objeto, los cuales pueden ser *básicos* o *derivados*. La notación es una diagonal como prefijo del atributo, como se muestra en la Figura 4.19.

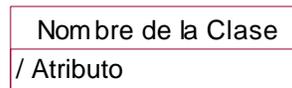


Figura 4.19. Notación para atributos derivados.

Ejemplo: El *Area* de un *Rectángulo* se puede calcular conociendo su *Ancho* y *Largo*, por lo cual no se define como un atributo básico de la caja, sino como un atributo derivado, como se muestra en la Figura 4.20.



Figura 4.20. Diagrama mostrando *Area* como un atributo derivado de los atributos básicos *Ancho* y *Largo*.

4.3.3 Restricciones de Atributos

Los valores de los atributos de una clase pueden restringirse. La notación para una restricción (en inglés "constraint") es incluir, por debajo de la clase y entre corchetes, la restricción para los valores del atributo, como se muestra en la Figura 4.21.

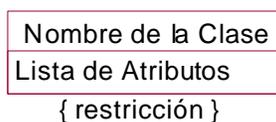


Figura 4.21. Notación para restricción en un diagrama de clases.

Ejemplo: Un *Rectángulo* puede restringir que su *Ancho* y *Largo* sean siempre iguales, lo que es equivalente a un *Cuadrado*. Así mismo, el *Area* del *Rectángulo* está definida como el *Ancho* por el *Largo*. Las dos restricciones se muestran en la Figura 4.22.

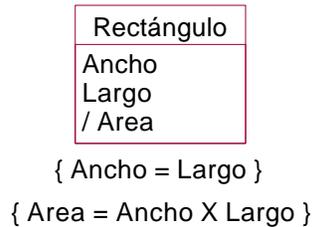


Figura 4.22. Diagrama mostrando dos restricciones para un *Rectángulo*: la restricción que el *Largo* sea igual al *Ancho* para el caso de cuadrados, y la restricción que el *Area* es igual al *Ancho* por el *Largo*.

4.4 Operaciones

Las *operaciones* son funciones o transformaciones que se aplican a todos los objetos de una clase particular. La operación puede ser una acción ejecutada por el objeto o sobre el objeto.

Ejemplo: *Arrojar*, *atrapar*, *inflar*, y *patear*, son operaciones para la clase *pelota*. *Abrir*, *cerrar*, *ocultar*, y *dibujar*, son operaciones para la clase *ventana*.

Las operaciones deben ser únicas dentro de una misma clases, aunque no necesariamente para diferentes clases.

Ejemplo: Las clases *pelota* y *libro* pueden las dos tener operaciones de *comprar*, pero no pueden tener cada una dos operaciones *comprar*.

No se debe utilizar el mismo nombre para operaciones que tengan un significado totalmente diferente.

Ejemplo: No se debe utilizar el mismo nombre *invertir* para la operación de invertir una figura y para la operación de invertir una matriz, ya que son operaciones totalmente diferentes. Invertir una figura es rotarla por 180 grados, mientras que invertir una matriz M es encontrar su inverso N , para que $MxN = 1$. Se deben usar nombres diferentes, como *invertir-figura* e *invertir-matriz*.

Las operaciones pueden tener *argumentos*, o sea, una lista de *parámetros*, cada uno con un tipo, y pueden también devolver resultados, cada uno con un tipo. Las operaciones se incorporan en la tercera sección de la clase, como se muestra en la Figura 4.23.

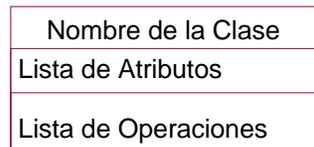


Figura 4.23. Notación para diagrama de clases conteniendo atributos y operaciones.

Ejemplo: En la Figura 4.24 se muestra tres clases, *Persona*, *Universidad* y *Rectángulo*, conteniendo atributos y operaciones. *Trabajar* y *Votar* son operaciones en *Persona*; *Enseñar* y *Graduar* son operaciones en *Universidad*; mientras que *Dibujar* y *Borrar* son operaciones en *Rectángulo*.



Figura 4.24. Diagrama de clases conteniendo atributos y operaciones para las clases *Persona*, *Universidad* y *Rectángulo*.

En la Figura 4.25 se muestra la notación extendida para una clase conteniendo atributos y operaciones, donde las operaciones pueden incluir una lista de tipos de argumentos, además de un lista de tipos de resultados.

Nombre de la Clase
Atributo1 : Tipo1 = Valor-Omisión1 Atributo2 : Tipo2 = Valor-Omisión2 ...
Operación1(Lista-Tipo-Arg1) : Tipo-Result1 Operación2(Lista-Tipo-Arg2) : Tipo-Result2 ...

Figura 4.25. Notación extendida para una clase, conteniendo atributos y operaciones.

Ejemplo: En la Figura 4.26 se muestra dos clases, *Figura* y *Archivo*, conteniendo atributos y operaciones. *Mover* y *Rotar* son operaciones de *Figura* conteniendo los argumentos *V* de tipo *vector* y *Angulo*, respectivamente. Ambas operaciones devuelven un resultado de tipo *Booleano* el cual devuelve un valor de *Cierto* o *Falso* (*True* o *False*). *Imprimir* es una operación de *Archivo* conteniendo un argumento *D* de tipo *dispositivo* que puede ser el nombre de una impresora, y el número *N* de copias a imprimir. El resultado puede ser un valor booleano.

Figura	Archivo
Posición Color	Nombre
Mover(v:Vector()) : Booleano Rotar(Angulo()) : Booleano	Imprimir(d:Dispositivo,n:Entero) : Booleano

Figura 4.26. Diagrama de clases de *Figura* y *Archivo* conteniendo atributos y operaciones con notación extendida.

Nótese que los objetos no incluyen ninguna información sobre sus operaciones, a diferencia de las clases, ya que las operaciones son idénticas para todos los objetos de una misma clase, a diferencia de los atributos que varían entre objetos en relación a su valor.

A continuación se describen otros conceptos relacionados con operaciones: *consultas*, *accesos*, *métodos*, *polimorfismo*, *parametrización* y *firmas*:

?? **Consultas.** Las operaciones que no tienen efectos secundarios, las cuales no cambian los valores de los atributos en el objeto, se les llama *consultas* (*query*). Una consulta es una operación que no modifica al objeto. Las consultas por lo general devuelven valores de atributos, básicos o derivados, y pueden o no tener argumentos. Ejemplo: Para una clase *Rectángulo* conteniendo los atributos *Ancho* y *Largo* pueden existir operaciones de consulta para leer los valores del *Ancho* y *Largo* sin afectarlos.

Una consulta se puede definir para la lectura de atributos derivados, ya que los atributos derivados no cambian los valores de los atributos del objeto.

Ejemplo: En la clase *Rectángulo* el atributo *Area* puede ser leído por medio de una consulta implementada como la multiplicación de los valores de los atributos *Largo* y *Ancho*.

La notación para las operaciones de consulta es la misma que para las operaciones, la diferencia es sólo en su comportamiento. Por regla general estas consultas no se incluyen de forma explícita en el modelo de objetos. Estas se agregan durante la etapa de diseño ya que son operaciones bastante básicas.

?? **Accesos.** Se puede definir *operaciones de acceso* para leer o escribir los atributos de un objeto. Si el acceso es hecho para leer solamente, sin afectar los valores de los atributos, entonces el acceso se considera también una operación de consulta. Se utiliza la notación de punto para indicar, dentro de la operación de acceso, el acceso a un atributo: "*objeto.atributo*". Ejemplo: para acceder el *nombre* de una *persona* se utiliza: *persona.nombre*

La notación para las operaciones de acceso es la misma que para las operaciones, la diferencia es sólo en su comportamiento. Por regla general estas operaciones de acceso no se incluyen de forma explícita en el modelo de objetos. Estas se agregan durante la etapa de diseño ya que son operaciones bastante básicas.

?? **Métodos.** El término *método* se utiliza para distinguir entre la operación, como un concepto de alto nivel, de la propia implementación de la operación, algo correspondiente a un concepto de más bajo nivel. Ejemplo: La operación *Imprimir* de la clase *Archivo* se implementa mediante un método *Imprimir* conteniendo un argumento llamado *dispositivo*, conteniendo el código para la implementación de la operación *Imprimir*.

En ciertos lenguajes de programación orientados a objetos se permite incluir más de un método para implementar una misma operación, en otras palabras múltiples descripciones de bajo nivel o implementaciones

para un mismo concepto a alto nivel. En estos casos los métodos varían según el número y tipo de argumentos, debiendo ser todos los métodos deben ser consistentes entre sí.

Ejemplo: La operación *Imprimir* de la clase *Archivo* se puede implementar con diferentes métodos. Un método *Imprimir* contiene el argumento *dispositivo* el cual manda el archivo a la impresora correspondiente. Otro método *Imprimir*, con exactamente el mismo nombre, pero sin argumentos, mandaría el archivo a la pantalla, en lugar de la impresora.

?? **Polimorfismo.** El *polimorfismo* se define como una misma operación tomando diferentes formas. Una operación se considera polimórfica si ésta se implementa en diferentes clases de forma diferente. Ejemplo: Para las clases *Archivo-ASCII* y *Archivo-PostScript* pueden existir diferentes métodos para implementar la operación *Imprimir*. Estos métodos corresponden a la misma operación *Imprimir*, pero se implementan de diferentes formas. El *Archivo-ASCII* imprime el texto en *ASCII*, mientras que el *Archivo-PostScript* requiere un interpretador *PostScript*.

Una operación también se considera polimórfica si ésta se implementa en una misma clase por diferentes métodos, con diferente número y tipo de argumentos.

Ejemplo: La operación *Imprimir* de la clase *Archivo* implementada con diferentes métodos *Imprimir* conteniendo diferente número de argumentos se considera polimórfica.

Con polimorfismo el transmisor de un mensaje no necesita saber la clase del objeto receptor.

Usando terminología más técnica, el vínculo (“binding”) entre el mensaje recibido y la operación apropiada se hace mediante: (i) un *vínculo estático* durante la compilación del lenguaje o (ii) un *vínculo dinámico* durante la ejecución, el cual también se conoce como *vínculo virtual*.

?? **Parametrización.** La *parametrización* de una operación está definida por el número y tipo de argumentos de cada método. Ejemplo: La parametrización de la operación *Imprimir* de la clase *Archivo* está definida por su argumento de tipo *dispositivo*.

?? **Firmas.** La *firma* de una operación se define por el tipo y número de argumentos y el tipo de resultados que devuelve. Ejemplo: La firma de la operación *Imprimir* en la clase *Archivo* está definida por su argumento de tipo *dispositivo*.

En operaciones polimórficas a través de diferentes clases las operaciones deben mantener la misma firma.

Ejemplo: La operación *Imprimir* para las clases *Archivo-ASCII* y *Archivo-PostScript* deben tener la misma firma a través de sus respectivos métodos.

4.5 Ligas y Asociación

La relación entre objetos se conoce como *liga*. Una *asociación* describe la relación entre clases de objetos, y describe posibles ligas, donde una liga es una *instancia* de una asociación, al igual que un objeto es una instancia de una clase.

Tipos de asociaciones entre clases:

?? Una *asociación de conocimiento* (“acquaintance associations”) es una asociación estática entre instancias y significa que una instancia conoce de la existencia de otra instancia. Denota conocimiento entre clases durante largos periodos.

?? Una *asociación de comunicación* es una asociación dinámica que modela la comunicación entre dos objetos, sirviendo para intercambiar información con entre objetos. Denota relación entre clases cuando existe una comunicación ellos A través de estas asociaciones, un objeto envía y recibe eventos.

Ejemplo: Los objetos *Juan Pérez* e *ITAM* están relacionadas por la liga *estudia-en* que describe que "Juan Pérez estudia en la ITAM".

Ejemplo: Las clases *Estudiante* y *Universidad* están relacionadas por la asociación *estudia-en* que describe que un "estudiante estudia en la universidad".

El nombre de una liga debe ser igual al nombre de la correspondiente asociación.

Ejemplo: *Juan Pérez* es una instancia de la clase *Estudiante*, y *ITAM* es una instancia de la clase *Universidad*. Por lo tanto, la liga *estudia-en* entre *Juan Pérez* y *ITAM* lleva el mismo nombre que la asociación *estudia-en* entre *Estudiante* y *Universidad*.

La asociación, al igual que la liga, es por naturaleza bidireccional. Por lo general, el nombre de la liga o asociación implica una dirección, pero puede ser invertida para mostrar la dirección opuesta. Cualquiera de las dos direcciones es igualmente correcta, aunque por lo general se acostumbra a leer de izquierda a derecha.

Ejemplo: El opuesto de "estudiante estudia en la universidad" sería "universidad da estudios a estudiante". Las dos direcciones se refieren a la misma asociación.

La notación describiendo una liga es una línea conectando los dos objetos, conteniendo el nombre de la liga en letras cursivas, como se muestra en la Figura 4.27.

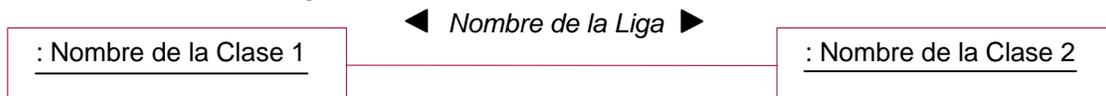


Figura 4.27. Notación para diagrama de objetos conteniendo una liga.

En la Figura 4.28 se muestra la liga *estudia-en* entre objetos de tipo *Estudiante* y *Universidad*.



Figura 4.28. Diagrama de objetos conteniendo la liga *estudia-en* entre los objetos *Juan Pérez*, de tipo *Estudiante*, y *ITAM*, de tipo *Universidad*.

Ejemplo: En la Figura 4.29 se muestra la liga *trabaja-para* entre los objetos *Raúl González*, de tipo *Persona*, y *IBM*, de tipo *Compañía*.



Figura 4.29. Diagrama de objetos conteniendo la liga *trabaja-para* entre los objetos *Raúl González*, de tipo *Persona*, y *IBM*, de tipo *Compañía*.

La notación describiendo una asociación es una línea conectando las dos clases, conteniendo el nombre de la asociación en letras cursivas, como se muestra en la Figura 4.30.

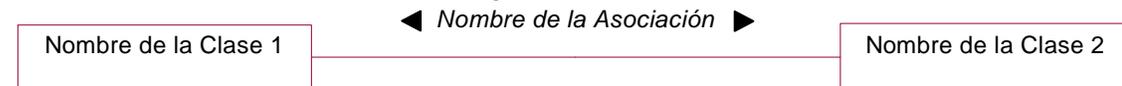


Figura 4.30. Notación para diagrama de clases conteniendo una asociación.

Ejemplo: En la Figura 4.31 se muestra la asociación *estudia-en* entre *Estudiante* y *Universidad*.

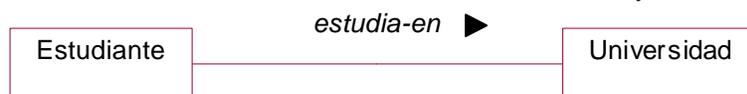


Figura 4.31. Diagrama de clases conteniendo la asociación *estudia-en* entre *Estudiante* y *Universidad*.

Ejemplo: En la Figura 4.32 se muestra la asociación *trabaja-para* entre *Persona* y *Compañía*.

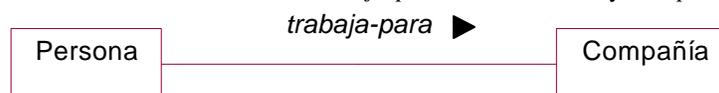


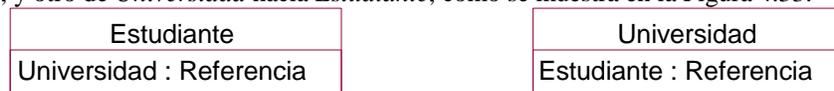
Figura 4.32. Diagrama de clases conteniendo la asociación *trabaja-para* entre *Persona* y *Compañía*.

4.5.1 Implementación

Vale la pena mencionar algo acerca de la implementación de ligas y asociaciones, ya que es un aspecto que la gran mayoría de los lenguajes orientados a objetos no apoyan de forma directa. El mecanismo principal que se utiliza en la mayoría de los lenguajes es la *referencia* o *apuntador* que permite a una entidad referirse a otra de manera primitiva. Por tal motivo, la referencia o apuntador se guardaría como un atributo adicional de la clase, algo que veremos con mayor detalle en el capítulo 8 y 9 donde discutiremos aspectos de diseño e implementación, respectivamente. Sin embargo, esta solución casi obligatoria, oculta el hecho de que la asociación no es parte de ninguna clase, sino depende de la combinación de varias clases. Como las asociaciones son bidireccionales, sería necesario el uso de un par de atributos, lo cual más aún ocultaría el hecho de que las dos direcciones de la asociación son dependientes. Por tales razones, es un error conceptual modelar asociaciones por medio de referencia o

apuntadores, pero como se mencionó antes, no tenemos alternativa. (Durante el diseño se puede decidir implementar la asociación por medio de uno o más referencias o apuntadores.)

Ejemplo: Sería incorrecto modelar la asociación *estudia-en* por medio de un *apuntador* apuntando de *Estudiante* hacia *Universidad*, y otro de *Universidad* hacia *Estudiante*, como se muestra en la Figura 4.33.



incorrecto

Figura 4.33. Diagrama de clases describiendo de forma incorrecta la asociación *estudia-en* por medio de una referencia entre *Estudiante* y *Universidad*.

Ejemplo: Sería también incorrecto modelar la liga *estudia-en* por medio de una *referencia* apuntando de *Juan Pérez* hacia *ITAM*, y otro de *ITAM* hacia *Juan Pérez*, como se muestra en la Figura 4.34. ("&" se usa como notación para referencias.)



incorrecto

Figura 4.34. Diagrama de objetos describiendo de forma incorrecta la liga *estudia-en* por medio de una referencia entre *Juan Pérez* y *ITAM*.

4.5.2 Grado de la Asociación

El *grado* de una asociación se determina por el número de clases conectadas por la misma asociación. Las asociaciones pueden ser binarias, ternarias, o de mayor grado. Las asociaciones se consideran binarias si relacionan solo dos clases.

Ejemplo: La asociación entre *Persona* e *Instituto* es una asociación binaria.

Las asociaciones pueden ser de mayor grado si relacionan a la misma vez más de dos clases. Aparte de relaciones binarias, lo más común son relaciones *ternarias* (entre tres clases), relaciones de más alto nivel son mucho menos comunes. Mientras el grado de una relación aumenta, su comprensión se dificulta, y se debe considerar partir las relaciones en varias relaciones binarias.

Ejemplo: Puede existir una relación *ternaria* entre *Estudiante*, *Profesor*, y *Universidad* donde "un estudiante estudia con un profesor en una universidad".

El grado de las ligas corresponden al de las asociaciones.

Ejemplo: Para una asociación binaria entre las clases *estudiante* y *universidad*, la liga correspondiente también es binaria ya que relaciona exactamente dos objetos, un objeto de tipo *estudiante* y otro de tipo *universidad*, como *Juan Pérez* y *ITAM*.

La notación para una relación ternaria entre objetos se muestra en la Figura 4.35. La relación no se etiqueta.

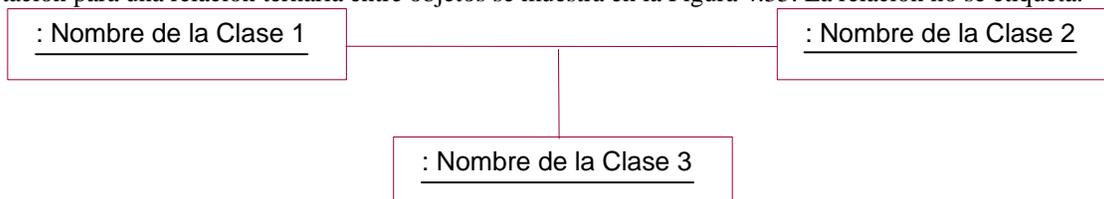


Figura 4.35. Notación para diagrama de instancias describiendo una asociación ternaria.

Ejemplo: En la Figura 4.36 se muestra posibles relaciones entre objetos correspondiendo a esta relación ternaria. El *Estudiante Juan Pérez* estudia con el *Profesor Alfredo Weitzenfeld* en la *Universidad ITAM*.



Figura 4.36. Diagrama de instancias describiendo una relación ternaria entre objetos de tipo *Estudiante*, *Profesor* y *Universidad*.

La notación para una relación ternaria entre clases se muestra en la Figura 4.37. La relación no se etiqueta.

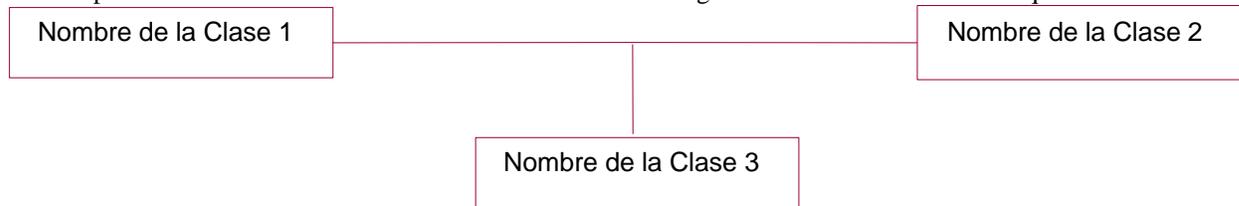


Figura 4.37. Notación para diagrama de clases describiendo una asociación ternaria.

Ejemplo: En la Figura 4.38 se muestra una asociación ternaria entre las clases *Estudiante*, *Profesor* y *Universidad*, describiendo a diferentes estudiantes que estudian con diferentes profesores en diferentes institutos.



Figura 4.38. Diagrama de clases describiendo una asociación ternaria entre *Estudiante*, *Profesor* y *Universidad*.

4.5.3 Asociaciones Reflexivas

Las asociaciones pueden ser *reflexivas*, relacionando distintos objetos de una misma clase.

Ejemplo: Para una clase *persona* puede existir una asociación *pariente* que describe que dos objetos de tipo *persona*, como *Juan Pérez* y *Laura Pérez* son *parientes*.

El grado de una asociación reflexiva puede ser binario, ternario, o de mayor grado, dependiendo del número de objetos involucrados.

Ejemplo: Para la clase *persona* puede existir una asociación ternaria entre tres personas donde uno es el *abuelo*, el otro es el *hijo del abuelo*, y el tercero es el *nieto del abuelo*.

Las asociaciones reflexivas relacionan distintos objetos de una misma clase.

Ejemplo: *Juan Pérez* es *pariente-de* *Laura Pérez*, donde ambos son objetos de tipo *Persona*, como se muestra en la Figura 4.39.

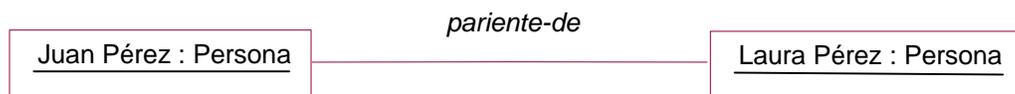


Figura 4.39. Diagrama de instancias describiendo una asociación reflexiva objetos de la clase *Persona*.

Ejemplo: La asociación reflexiva *pariente-de* para la clase *Persona* se muestra en la Figura 4.40.

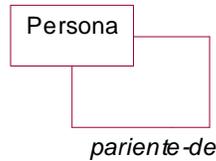


Figura 4.40. Diagrama de clases describiendo una asociación reflexiva para la clase *Persona*.

4.5.4 Multiplicidad

La *multiplicidad* (**cardinalidad**) de una asociación especifica cuantas instancias de una clase se pueden relacionar a una sola instancia de otra clase.

Ejemplo: En el caso de *Estudiante* y *Universidad*, la multiplicidad está dada por el número de estudiantes que puedan estudiar en una sola universidad. En otras palabras, muchos objetos de tipo *Estudiante* se conectan a un solo objeto de tipo *Universidad*.

Es necesario decidir la multiplicidad para cada clase en una asociación, o sea dos multiplicidades por cada relación binaria, una para cada extremo de la relación.

Ejemplo: En la relación *estudia-en* es necesario definir la multiplicidad para el *Estudiante* y para la *Universidad*.

La multiplicidad restringe una asociación limitando el número de objetos que pueden relacionarse a un objeto particular.

Ejemplo: En la asociación *estudia-en* se puede restringir el número de estudiantes que pueden estudiar en una universidad.

La multiplicidad depende del contexto de la aplicación. Existen, distintos tipos de multiplicades para las asociaciones de las cuales las más relevantes son las siguientes:

?? "uno-uno": donde dos objetos se relacionan de forma exclusiva, uno con el otro. Ejemplo: Cada *Universidad* tiene un *Rector*, y cada *Rector* rige una *Universidad*.

?? "uno-muchos": donde uno de los objetos pueden estar ligado a muchos otros objetos. Ejemplo: Muchos *Estudiantes* pueden estudiar en una *Universidad*, y una sola *Universidad* da estudios a cada *Estudiante*.

?? "muchos-muchos": donde cada objeto de cada clase puede estar ligados a muchos otros objetos. Ejemplo: Muchos *Estudiantes* pueden estudiar en varias *Universidades*.

La multiplicidad se incluye en el diagrama de clases únicamente. La multiplicidad para relaciones de mayor grado es más compleja, volviéndose esta notación un poco ambigua para relaciones de mayor orden ya que no sabría cómo leerse la relación. Se incorpora la siguiente notación en cada uno de los extremos de una asociación binaria. La notación para relaciones "uno-uno", donde dos objetos solo pueden tener una liga entre ellos, es la notación básica de asociación hasta ahora dada, como se muestra en la Figura 4.41.



Figura 4.41. Diagrama de clases describiendo una multiplicidad de "uno-uno".

La notación para relaciones "uno-muchos", donde uno de los objetos pueden estar ligado a muchos otros objetos, está dada por una "bolita negra" representando el lado de "muchos", el cual corresponde a cero o más ligas, como se muestra en la Figura 4.42.



Figura 4.42. Diagrama de clases describiendo una multiplicidad de "uno-muchos".

Ejemplo: En el caso de una *Universidad* donde pueden atender muchos *Estudiantes*, el diagrama se muestra en la Figura 4.43, donde la relación de "muchos" se incorpora del lado de *Estudiantes*. (El contrario significaría que un estudiante puede atender muchas universidades.)



Figura 4.43. Diagrama de clases describiendo una multiplicidad de "uno-muchos" entre *Estudiantes* y *Universidad*.

La notación para relaciones "muchos-muchos", donde los dos objetos pueden estar ligados a muchos otros objetos, está dada por dos "bolitas negras" correspondiendo cada una a una multiplicidad de "muchos", como se muestra en la Figura 4.44.



Figura 4.44. Diagrama de clases describiendo una multiplicidad de "muchos-muchos".

Ejemplo: En el caso de muchas *Universidades* donde pueden atender muchos *Estudiantes*, el diagrama se muestra en la Figura 4.45.



Figura 4.45. Diagrama de clases describiendo una multiplicidad de "muchos-muchos" entre *Estudiantes* y *Universidad*.

La notación para representar una relación opcional, donde la multiplicidad es "uno" o "cero", describiendo una relación opcional, 0 o 1. Esto significa que dos objetos pueden o no estar conectados, y si lo están corresponden a una multiplicidad de 1. La notación se muestra en la Figura 4.46.



Figura 4.46. Diagrama de clases describiendo una multiplicidad "opcional".

Ejemplo: El caso de muchos *Estudiantes* que pueden o no atender a una sola *Universidad* se muestra en la Figura 4.47. (Esto es a diferencia del ejemplo anterior donde los estudiantes tienen que atender a una universidad.)



Figura 4.47. Diagrama de clases describiendo una multiplicidad de "opcional-muchos" entre *Estudiantes* y *Universidad*.

La notación de "asterisco", correspondiendo a una relación de muchos, se puede restringir con un número, conjunto de números, de objetos que deben estar conectados entre sí.

Ejemplo: En la Figura 4.48 se muestra una relación con multiplicidad de cero o más personas que trabajan para una compañía.



Figura 4.48. Diagrama de clases incluyendo multiplicidad en la asociación.

Ejemplo: En la Figura 4.49 se muestra una relación donde exactamente dos personas trabajan en una compañía.



Figura 4.49. Diagrama de clases incluyendo multiplicidad de "2" en la asociación.

Ejemplo: En la Figura 4.50 se muestra una relación donde por lo menos diez personas trabajan para una compañía.



Figura 4.50. Diagrama de clases incluyendo multiplicidad de "10" o más en la asociación.

Ejemplo: En la Figura 4.51 se muestra una relación donde una o dos personas trabajan para una compañía.



Figura 4.51. Diagrama de clases incluyendo multiplicidad de "1" o "2" en la asociación.

Ejemplo: En la Figura 4.52 se muestra una relación, de tipo opcional, donde cero o una persona trabajan para una compañía.



Figura 4.52. Diagrama de clases incluyendo multiplicidad de tipo *opcional*, donde "0" o "1" *personas* se relacionan con *compañía*.

4.5.5 Rol

El *rol* describe el papel que juega cada extremo de una asociación. Una asociación binaria tiene dos roles, uno en cada extremo, los cuales pueden tener un nombre diferente cada uno. Una relación de *n* clases tendría *n* roles. El nombre del rol provee una forma de atravesar la asociación de un objeto en un extremo sin mencionar explícitamente el nombre de la asociación.

Ejemplo: Una *Persona* asume el rol de *Empleado* con respecto a la *Compañía*; y la *Compañía* asume el rol de *Empleador* con respecto a la *Persona*.

Cuando hay solamente una asociación conectando dos clases, a menudo el nombre de la clase sirve como nombre de rol, y no es necesario agregar un nombre de rol de forma explícita.

Ejemplo: *Estudiante* y *Universidad* son bastante descriptivos que no es necesario agregar un nombre de rol. Si la clase fuera *Persona*, el nombre de rol *Estudiante* sería más descriptivo.

Los nombres de los roles no deben duplicar los atributos de la clase a la cual éstos describen. (Esto se hace por razones de implementación.)

Ejemplo: El rol *empleado* no debe ser un atributo de *Persona*.

Se pueden incorporar nombres de los roles y de la asociación a la misma vez, o uno de los dos solamente, lo cual es más que suficiente para describir la relación. La Figura 4.53 muestra la notación para un rol.



Figura 4.53. Notación para diagrama de clases conteniendo nombres de los roles.

Ejemplo: *Persona* tiene el rol de *empleado* con respecto a la *Compañía*, la cual tiene el rol de *empleador* con respecto a *Persona*, como se muestra en la Figura 4.54.

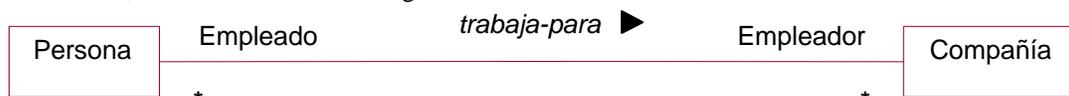


Figura 4.54. Diagrama de clases para una asociación con los nombres de los roles, nombre de la asociación y multiplicidad.

Los nombres del rol son necesarios para asociaciones reflexivas (asociaciones entre objetos de la misma clase), ya que sólo sabiendo el nombre de la asociación no es suficiente para distinguir el papel que juegan los diferentes objetos en la asociación.

Ejemplo: Si una *Persona* puede ser *jefe* o *empleado* en una *Compañía*, entonces la única forma de distinguir el papel que la *Persona* juega es por medio de un nombre de rol, como se muestra en la Figura 4.55.

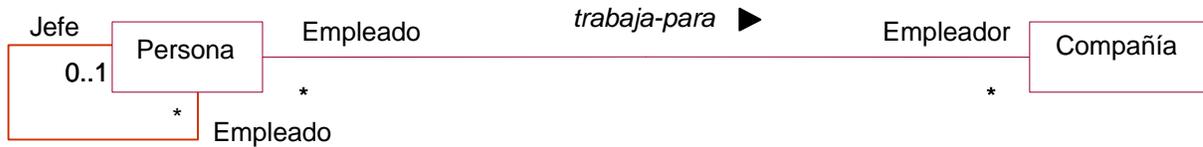


Figura 4.55. Diagrama de clases para una asociación reflexiva con roles.

Es importante utilizar nombres de rol para distinguir entre dos asociaciones diferentes las cuales relacionan a un mismo par de clases. Los roles deben ser diferentes según la asociación para el mismo extremo de la relación. Ejemplo: Una *Persona* además de ser *empleado* de una *Compañía* también puede ser el *dueño* de ella. Por lo tanto existen dos asociaciones entre *Persona* y *Compañía*, la primera es *trabaja-para* y la segunda es *dueño-de*. Ambas relaciones se pueden describir por medio de roles, como se muestra en la Figura 4.56.

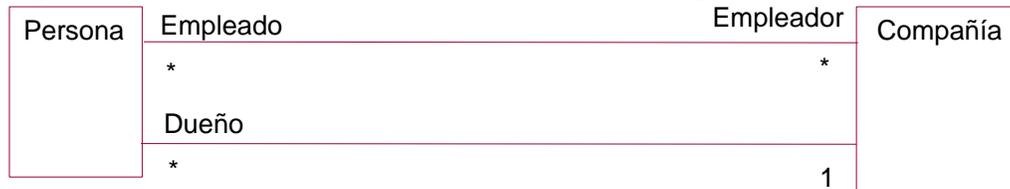


Figura 4.56. Diagrama de clases para asociaciones entre dos clases distinguidas por los nombres de rol.

4.5.6 Restricciones de Ligas y Asociaciones

Las *restricciones* especifican una relación particular entre las diferentes asociaciones o ligas. Las restricciones restringen los valores que estas entidades pueden asumir. Las restricciones sencillas se pueden añadir al modelo de objeto, mientras que las más complejas deben ser especificadas en el modelo funcional. Por lo general las restricciones se describen de forma declarativa, aunque luego deban convertirse a un procedimiento para ser implementadas. Las restricciones pueden ser expresadas en lenguaje natural o por medio de ecuaciones. Las restricciones son limitadas por corchetes y son ubicadas cerca de la entidad restringida, como se muestra en la Figura 4.57.



Figura 4.57. Notación para un diagrama de clases mostrando una restricción de asociación. La restricción debe ubicarse cerca de la entidad que afecte, y no necesariamente en el centro como se muestra en la figura.

?? **Subconjunto.** Las posibles ligas de una asociación pueden ser un subconjunto de las posibles ligas de otra asociación. La multiplicidad de la asociación del subconjunto debe ser igual o menor que la multiplicidad de la asociación del superconjunto. Una flecha se utiliza para relacionar el subconjunto con la entidad de la cual depende, como se muestra en la Figura 4.58.

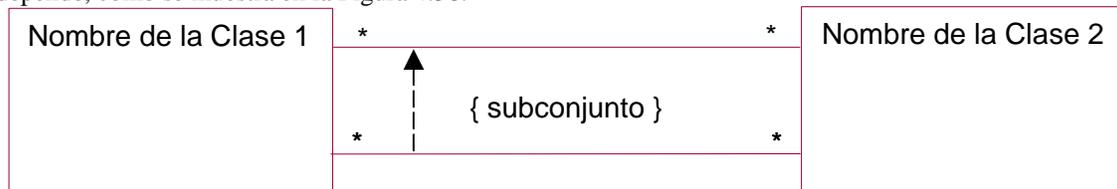


Figura 4.58. Diagrama de clases mostrando una restricción de subconjunto entre dos asociaciones.

Ejemplo: El *presidente* de un *País* debe ser un *habitante* del *País*. La asociación *presidente-de* es un subconjunto de la asociación *habitante-de*, como se muestra en la Figura 4.59.

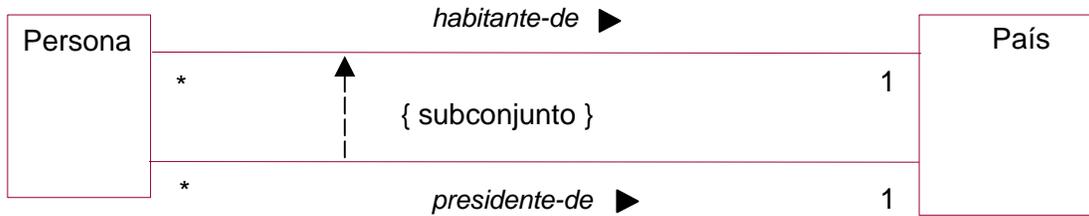


Figura 4.59. Diagrama de clases mostrando a *presidente-de* como un subconjunto de *habitante-de*.

?? **Orden.** La multiplicidad "muchos" indica que un conjunto de objetos puede estar relacionados a un mismo objeto. Estas relaciones pudieran estar ordenadas, como se muestra con la notación en la Figura 4.60.



Figura 4.60. Notación para un diagrama de clases mostrando una restricción de orden.

Ejemplo: Una ventana en una estación de trabajo está superpuesta por varias otras ventanas. Las ventanas son ordenadas para que se desplieguen en el orden correcto y la de más arriba se despliegue por último. Para indicar esta situación se incluye una restricción especial de orden, como se muestra en la Figura 4.61.



Figura 4.61. Diagrama de clases mostrando una restricción de orden para las *ventanas* en una *estación de trabajo*.

4.5.7 Asociaciones Derivadas

Las *asociaciones derivadas* se consideran *asociaciones dependientes* o *redundantes*, las cuales están determinadas directa o indirectamente a través de otras asociaciones y que se agregan para facilitar la búsqueda de información.. La notación es una diagonal atravesando la asociación, como se muestra en la Figura 4.62.

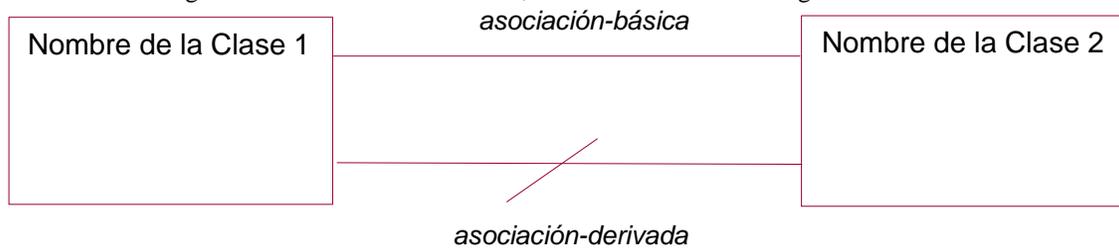


Figura 4.62. Notación para un diagrama de clases mostrando asociaciones derivadas.

Ejemplo: Para la clase *Persona* puede existir una asociación *padre-de* que define que una persona es el padre y la otra el hijo. Se puede crear una asociación derivada *abuelo-de* que depende de que existan dos ligas *padre-de* definiendo a uno de los objetos como abuelo y el otro como nieto. La relación *abuelo-de* es una asociación derivada, como se muestra en la Figura 4.63. También se podría definir las relaciones derivadas *tío*, *suegra*, *primo*, las cuales se pueden deducir de otras relaciones básicas como *padre-de*, *esposo*, y *hermana*.

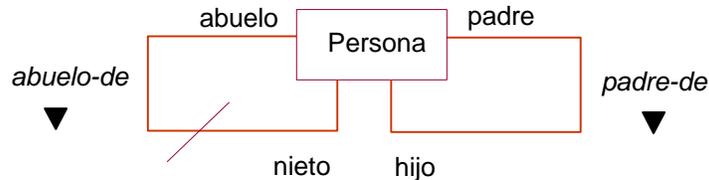


Figura 4.63. Diagrama mostrando la clase *Persona* conteniendo *abuelo-de* como una asociación derivada de *padre-de*.

Ejemplo: Un *Profesor* enseña una sola *Materia* a muchos *Estudiantes*, como muestra la Figura 4.64. La relación *Estudiante* estudia *Materia* es una asociación derivada ya que conociendo al *Profesor*, se puede conocer la *Materia*, y por la tanto deducir que *Materia* se le es enseñada a los *Estudiantes*.



Figura 4.64. Diagrama de clases con asociaciones derivadas redundantes.

No todas las asociaciones que forman múltiples conexiones entre clases indican redundancia. A veces la existencia de una asociación se deriva de dos o más asociaciones primitivas, pero la multiplicidad no. Se debe mantener la asociación adicional, ya que puede ser importante para una restricción adicional de multiplicidad.

Ejemplo: Los *Profesores* enseñan muchas *Materias* a muchos *Estudiantes*, como muestra la Figura 4.65. La relación *Estudiante* estudia *Materia* no es en este caso una asociación derivada ya que conociendo al *Profesor*, no se puede deducir la *Materia*, ya que cada profesor da muchas materias, y por la tanto es necesario agregar la relación adicional para saber que *Materia* se le es enseñada a cada *Estudiante*.



Figura 4.65. Diagrama de clases con asociaciones no redundantes.

4.5.8 Accesos

Las *operaciones de acceso* son operaciones que además de leer o escribir atributos, pueden servir para acceder las ligas relacionadas con un objeto. Se utiliza la notación de punto para indicar el acceso a una liga: "*objeto1.objeto2*". Ejemplo: Se puede acceder la *cuenta* la cual esta ligada a una *persona* por medio de la instrucción *persona.cuenta*, la cual se incluiría dentro de la operación *accesar-cuenta*, como se muestra en la Figura 4.66.



Figura 4.66. Diagrama mostrando una *Persona* relacionada con una *Cuenta*.

Se puede acceder los objetos por medio de "pseudo-atributos" de los roles de las asociaciones.

Ejemplo: Se puede acceder la *universidad* a la cual asiste una *persona* por medio de su rol de *estudiante* utilizando la instrucción *estudiante.universidad*, la cual se incluiría dentro de la operación *accesar-universidad*, como se muestra en la Figura 4.67.

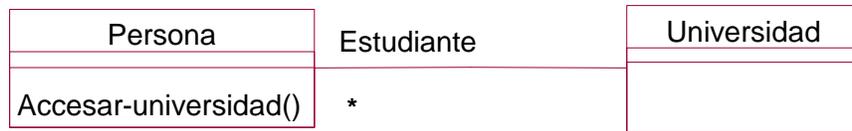


Figura 4.67. Diagrama mostrando una *Universidad* relacionada con varios *Personas*, según su rol de *Estudiantes*.

4.5.9 Atributos de Liga y Asociación

Al igual que un atributo de clase es propiedad de la clase, un *atributo de asociación* (o *atributo de liga*) es propiedad de una asociación. La notación es similar a la usada para los atributos de clases, excepto que se añade a la asociación, y no se incorpora un nombre de clase, como se muestra en la Figura 4.68.

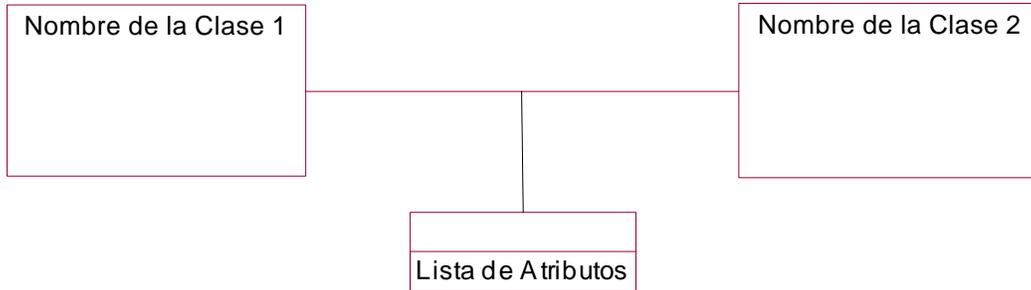


Figura 4.68. Notación para diagrama de clases con asociaciones conteniendo una lista de atributos de asociación.

Ejemplo: Para una asociación entre *Persona* y *Compañía*, se puede definir los atributos *salario* y *puesto* como atributos de la asociación *trabaja-para*, como se muestra en la Figura 4.69.

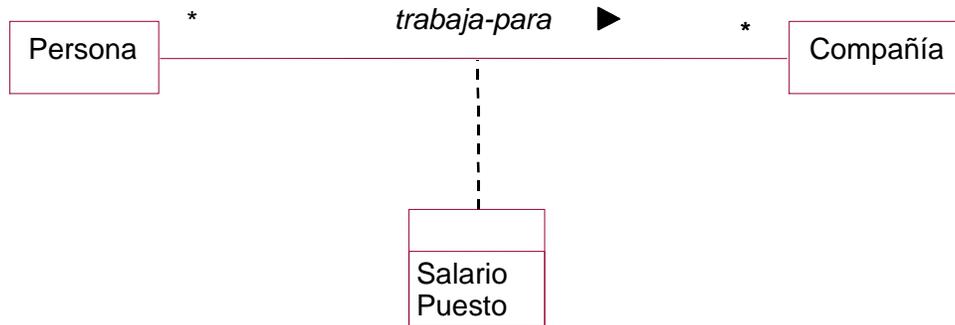


Figura 4.69. Diagrama de clases para *Persona* y *Compañía* conteniendo los atributos de asociación *salario* y *puesto*.

La notación en el diagrama de objetos es similar, como se muestra en la Figura 4.70.

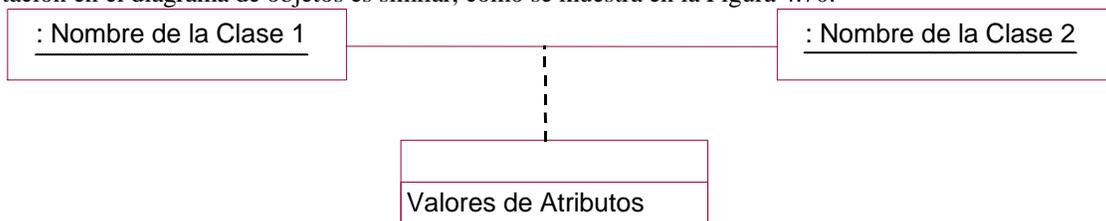


Figura 4.70. Notación para diagrama de objetos con ligas conteniendo valores de atributos de las ligas.

Ejemplo: Para una liga entre *Raúl González* e *IBM*, se da un valor de *\$100,000* al *salario* y un valor de *gerente* al *puesto*, como se muestra en la Figura 4.71.

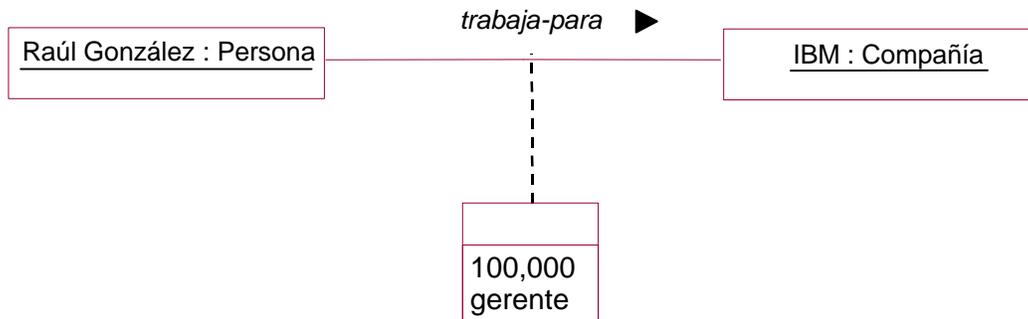


Figura 4.71. Diagrama de objetos para *Raúl González* e *IBM* conteniendo el valor de \$100,000 para el atributo de asociación *salario*, y *gerente* para el atributo de liga *puesto*.

Modelo Alternativo

Las siguientes son diferentes alternativas al modelo como atributo de asociación. Se puede modelar tal atributo como atributo de una de las clases involucradas en la relación, según la multiplicidad de la relación. Aunque las alternativas son posibles, es conceptualmente más correcto describir los atributos los cuales dependen de ambas clases, como atributo de asociación.

Si la multiplicidad de la asociación es de "uno-uno", el atributo de asociación se podría modelar como un atributo en cualquiera de las dos clases.

Ejemplo: En la Figura 4.72 se incluyen *salario* y *puesto* como parte de *Persona* (podría también ser parte de *Compañía*).



Figura 4.72. Diagrama de clases para *Persona* y *Compañía* conteniendo multiplicidad "uno-uno" y los atributos de asociación *salario* y *puesto* incluidos directamente en la clase *Persona* (o de forma alterna en la clase *Compañía*).

Si la multiplicidad de la asociación es de "uno-muchos", el atributo de asociación se podría modelar como un atributo en el lado de la clase "muchos".

Ejemplo: En la Figura 4.73 se incluyen *salario* y *puesto* como parte de *Persona*, ya que *Persona* está del lado "muchos" de la asociación. Del otro lado sería incorrecto ya que significaría que todas las *Personas* estarían ganando el mismo *salario* y tendrían el mismo *puesto*.



Figura 4.73. Diagrama de clases para *Persona* y *Compañía* conteniendo multiplicidad "uno-muchos" y el atributo de asociación *salario* y *puesto* incluidos directamente en la clase *Persona*.

Si la multiplicidad de la asociación es de "muchos-muchos" no es posible modelarlo como un atributo de clase.

Ejemplo: En la Figura 4.74 se muestra de forma incorrecta la incorporación de *salario* y *puesto* a *Persona* (o de forma alterna a *Compañía*) ya que la relación es de "muchos-muchos", y esto significaría que una *Persona* tendría el mismo *salario* y el mismo *puesto* en todas las *Compañías* en las cuales trabajara. Esto no es necesariamente correcto. La representación correcta será mostrada más adelante.



Figura 4.74. Diagrama de clases incorrecto para *Persona* y *Compañía* conteniendo los atributos de asociación *salario* y *puesto* incluidos directamente en la clase *Persona*.

Asociaciones Ternarias

Los atributos de asociación también pueden existir para las asociaciones ternarias.

Ejemplo: Un *Estudiante* toma una *Materia* en una *Universidad* donde se le da una *Calificación* la cual depende de las tres clases, como se muestra en la Figura 4.75.



Figura 4.75. Diagrama de clases mostrando una asociación ternaria entre las clases *Estudiante*, *Materia* y *Universidad* incluyendo el atributo *Calificación*.

Ejemplo: En la Figura 4.76 se muestra un ejemplo de modelo de objetos para la relación ternaria anterior.

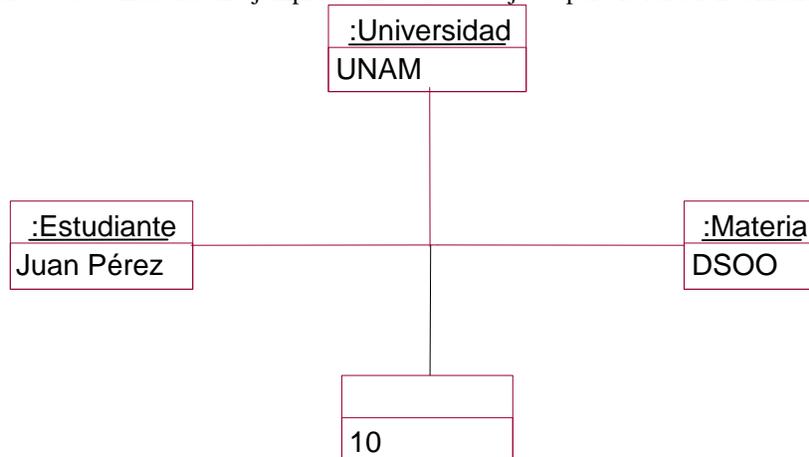


Figura 4.76. Diagrama de instancias para *Estudiante*, *Materia* y *Universidad* incluyendo un valor para el atributo *Calificación* para la asociación ternaria entre estas clases.

4.5.10 Operaciones de Asociación

Se pueden modelar como operaciones de asociación aquellas operaciones que dependan de las clases involucradas en la relación, de forma análoga a los atributos de asociación.

La notación se muestra en la Figura 4.77.



Figura 4.77. Notación para el diagrama de clases conteniendo atributos y operaciones de asociación.

Ejemplo: La operación *Calcular-Ganancia* depende del *Salario* de la *Persona* con respecto a la *Compañía* por lo cual se modela como operación de asociación, como se muestra en la Figura 4.78.

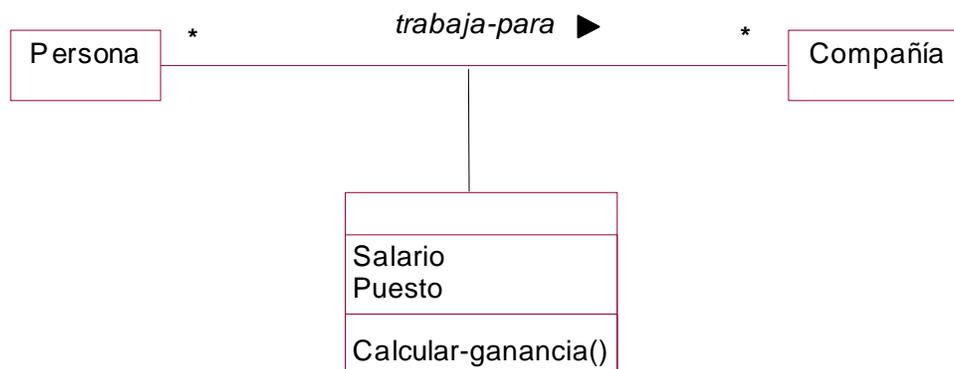


Figura 4.78. Diagrama de clases conteniendo los atributos y operaciones de asociación, *Salario* y *Puesto*, y *Calcular-Ganancia*, respectivamente.

4.5.11 Asociaciones como Clases

Se puede modelar una asociación como si fuera una clase, en particular si se desea asociar la propia asociación con otras clases. Los atributos y operaciones de la asociación pasan a ser miembros de la clase correspondiente a la asociación.

La notación se muestra en la Figura 4.79.

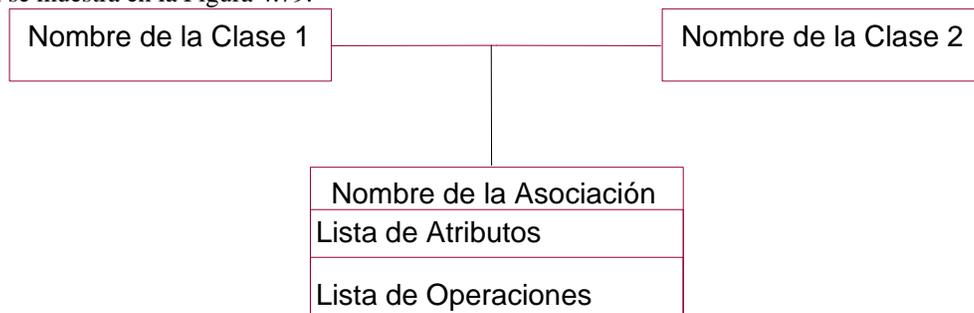


Figura 4.79. Notación para diagrama de clases modelando la asociación como una clase.

Ejemplo: La relación *trabaja-para* entre *Persona* y *Compañía*, se puede convertir en una clase, si pensamos en relaciones con otra clase como *Seguro-de-Trabajo* para cada *Persona* que esté trabajando en una *Compañía*, como se muestra en la Figura 4.80.

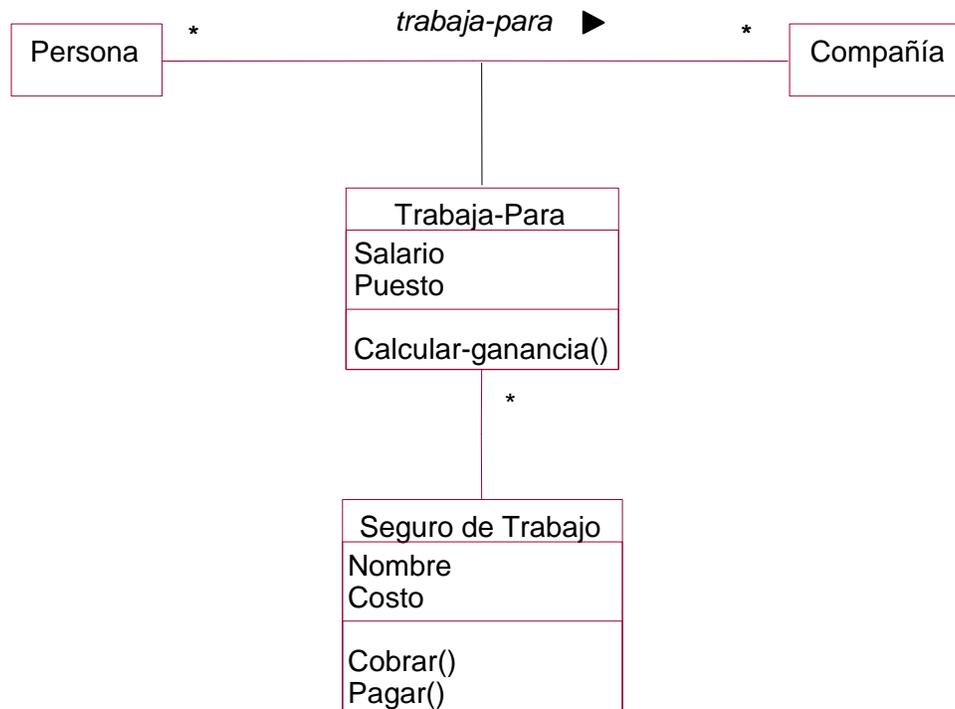


Figura 4.80. Diagrama de clases para una asociación, *trabaja-para*, modelada como una clase.

4.6 Ensamblados: Agregación y Composición

Los *ensamblados*, en particular la *agregación* y *composición*, son formas especiales de asociación entre un todo y sus partes, en donde el *ensamblado* está compuesto por sus componentes. El ensamblado es el objeto **central**, y la estructura completa se describe como una **jerarquía de contenido**. Un ensamblado puede componerse de varias partes, donde cada relación *parte-todo* se considera una relación separada. En el caso de *agregación*, las partes del ensamblado pueden aparecer en múltiples ensamblados. En el caso de *composición*, las partes del ensamblado no pueden ser compartidas entre ensamblados.

Ejemplo: Una *Red de Computadoras* se puede considerar un ensamblado, donde las *Computadoras* son sus componentes. Este también es un ejemplo de agregación, ya que las *Computadoras* pudieran ser partes de múltiples *Redes de Computadoras* a la vez. Adicionalmente, las *Computadoras* pueden existir independientemente de la existencia de la *Red de Computadoras*.

Ejemplo: Un *Automóvil* se puede también considerar un ensamblado, donde el *Motor* y la *Carrocería* son sus componentes. Este también es un ejemplo de composición, ya que el *Motor* y la *Carrocería* son partes del *Automóvil*, y a diferencia de la agregación, no pueden ser compartidos entre distintos *Automóviles* a la vez. Adicionalmente, no tiene mucho sentido que el *Motor* y la *Carrocería* existan de manera independiente al *Automóvil*, por lo cual la composición refleja de manera importante, el concepto de *propiedad*.

El ensamblado tiene propiedades de *transición*: Si A es parte de B y B es parte de C; entonces A es parte de C.

Ejemplo: Si el *Motor* es parte del *Automóvil*, entonces sus propiedades, como su posición y velocidad, están dadas por la posición y velocidad del *Automóvil*.

El ensamblado es *antisimétrico*: Si A es parte de B, entonces B no es parte de A. Estas propiedades se propagan entre el ensamblado y sus componentes.

Ejemplo: Si el *Motor* es parte del *Automóvil*, entonces el *Automóvil* no es parte del *Motor*.

Se considera un ensamblado y no una asociación regular:

?? Si se puede usar la frase "parte-de" o "consiste-de" o "tiene";

?? Si algunas operaciones en el todo se pueden *propagarse* a sus partes;

?? Si algunos atributos en el todo se pueden propagar a sus partes;

El ensamblado es común en los objetos interfaz. En un sistema de ventanas, por ejemplo, una ventana puede consistir de botones, menús, y barras ("scrollbars"), cada una modelada por su propio objeto interfaz. El resultado es una estructura de interfaz en forma de árbol. La decisión de usar ensamblados es un poco arbitraria, y en la práctica no

causa grandes problemas la distinción imprecisa entre agregación, composición y asociación, aunque es bueno ser consistente.

La notación para un ensamblado, en particular para un agregado, es un diamante adherido al lado del objeto correspondiente al ensamblado total, conectado por una línea a sus componentes, como se muestra en la Figura 4.81.



Figura 4.81. Notación en un diagrama de clases para una agregación.

La notación para la composición es similar a una agregación y al ensamblado en general, aunque a diferencia de la agregación el diamante se rellena de color negro, como se muestra en la Figura 4.82.



Figura 4.82. Notación en un diagrama de clases para una composición.

Existe una notación adicional para la composición descrita como un anidamiento gráfico, como se muestra en la Figura 4.83.

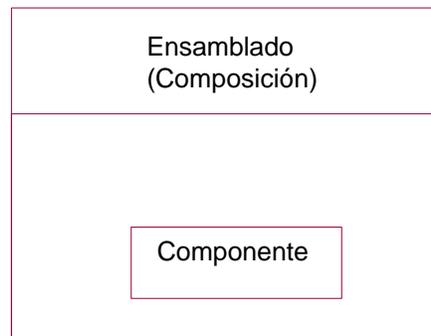


Figura 4.83. Notación en un diagrama de clases para una composición.

Ejemplo: El *Automóvil* con sus componentes, *Motor* y *Carrocería*, se muestran en el diagrama de la Figura 4.84.

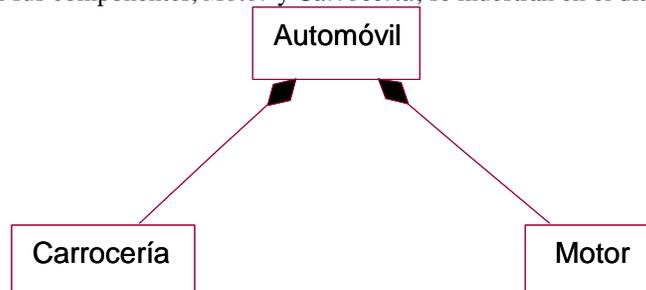


Figura 4.84. Diagrama de clases para la composición de un *Automóvil* que contiene un componente *Motor* y otro componente *Carrocería*.

Ejemplo: Una computadora personal (PC) está compuesta por uno o varios monitores, un sistema, un teclado y opcionalmente un ratón. El sistema tiene un chasis, un procesador central (CPU), varias tarjetas de memoria (RAM), y opcionalmente un ventilador. El diagrama se muestra en la Figura 4.85.

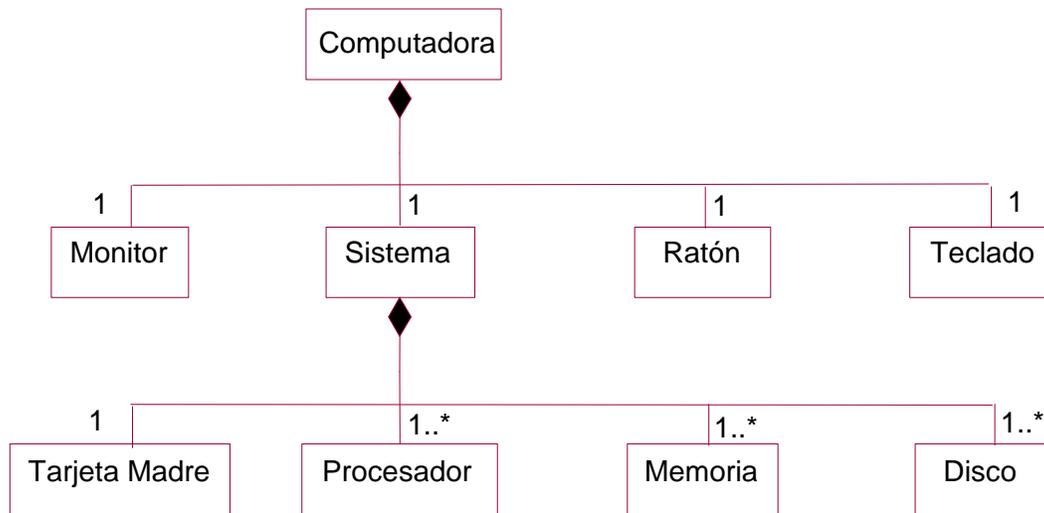


Figura 4.85. Diagrama de clases para la composición de una *computadora personal (PC)* conteniendo diferentes componentes.

Ejemplo: Una *Universidad* está compuesta de sus *Divisiones*, que están a su vez compuestas de sus *Departamentos*. Una *Universidad* es indirectamente una composición de sus *Departamentos*; pero la *Universidad* no es una composición de sus *Estudiantes*, si no más bien una agregación, ya que sus *Estudiantes* son objetos independientes, incluso pudiendo pertenecer a múltiples *Universidades*, como se muestra en la Figura 4.86. (Este es un ejemplo de ensamblado variable, como se describe en la siguiente sección.)

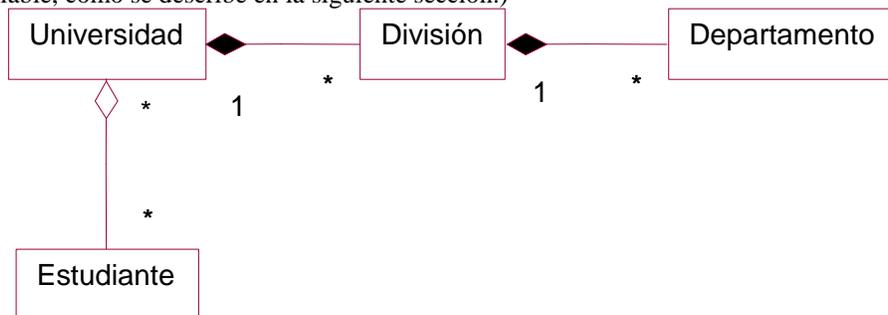


Figura 4.86. Diagrama de clases para la composición de una *Universidad* que contiene componentes de *División* que a su vez contiene componentes de *Departamento*. La relación entre *Universidad* y *Estudiante* es de agregación.

4.6.1 Tipos de Ensamblados

Los tipos de ensamblados pueden ser:

?? **Fijos.** Los ensamblados *fijos* tienen una estructura fija donde el número de componentes está predefinido.

Ejemplo: Un *Automóvil* tiene un *Motor*, una *Carrocería*, y cuatro *Ruedas*, como se muestra en la Figura 4.87.

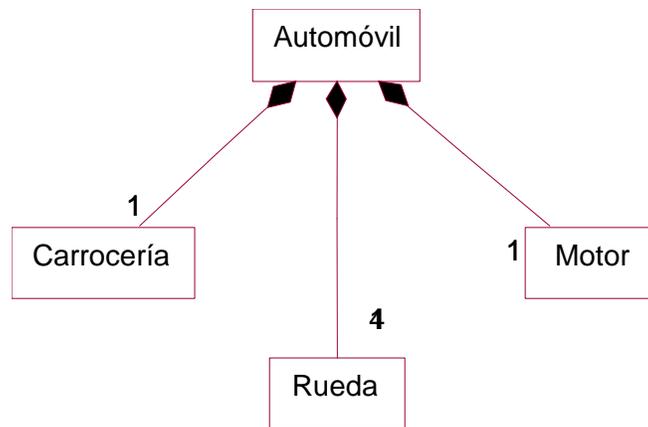


Figura 4.87. Diagrama de clases para un ensamblado fijo describiendo un *Automóvil* que contiene varios componentes: un *Motor*, una *Carrocería*, y cuatro *Ruedas*.

?? **Variables.** Los ensamblados *variables* tienen un número finito de niveles, pero el número de componentes varía. Ejemplo: Un *País* contiene varias *Ciudades*, que contienen a su vez varias *Casas*. No se sabe cuántas ciudades existen, ni tampoco cuántas casas, aunque sí se sabe el nivel del ensamblado, como se muestra en la Figura 4.88.

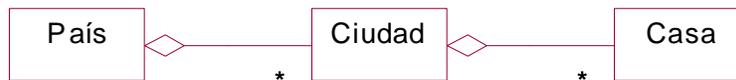


Figura 4.88. Diagrama de clases para un ensamblado variable describiendo un *País* que contiene varias *Ciudades*, que a su vez contienen varias *Casas*.

?? **Recursivos.** Los ensamblados *recursivos* contienen de forma directa o indirecta una instancia del mismo tipo de agregado, donde el número de niveles de ensamblado es potencialmente ilimitado. Ejemplo: Un *Directorio* en un sistema operativo está definido de forma recursiva, pudiendo contener otros directorios que a su vez pueden contener otros directorios, y así sucesivamente de forma indefinida, como se muestra en la Figura 4.89.



Figura 4.89. Diagrama de clases para un ensamblado recursivo describiendo un *Directorio* que puede contener otros varios *Directorios* de forma recursiva.

4.6.2 Propagación de Operaciones

Una de las metas del ensamblado es que las operaciones aplicadas al ensamblado puedan *propagarse* de forma automática a sus objetos componentes. La operación se propaga en una sola dirección, y puede ser aplicada a las partes del ensamblado de forma independiente.

La notación para la propagación de operaciones se muestra en la Figura 4.90. La propagación se indica con una flecha, en el sentido de la propagación, junto al nombre de la operación etiquetando la asociación.

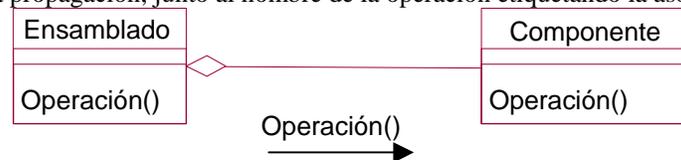


Figura 4.90. Notación en un diagrama de clases para la propagación de operaciones en un ensamblado.

Ejemplo: Cuando el *Automóvil* se mueve, la operación de moverse se propaga por el *Automóvil*, y también se mueven todas sus partes, como el *Motor*, la *Carrocería*, y las *Ruedas*. La operación no se propaga en sentido contrario, ya que, por ejemplo, la *Carrocería* no puede moverse sin que se mueva el *Automóvil* completo, como se muestra en la Figura 4.91. También se pudieran propagar otras operaciones, como *pintar*, *vender*, *comprar*, etc.

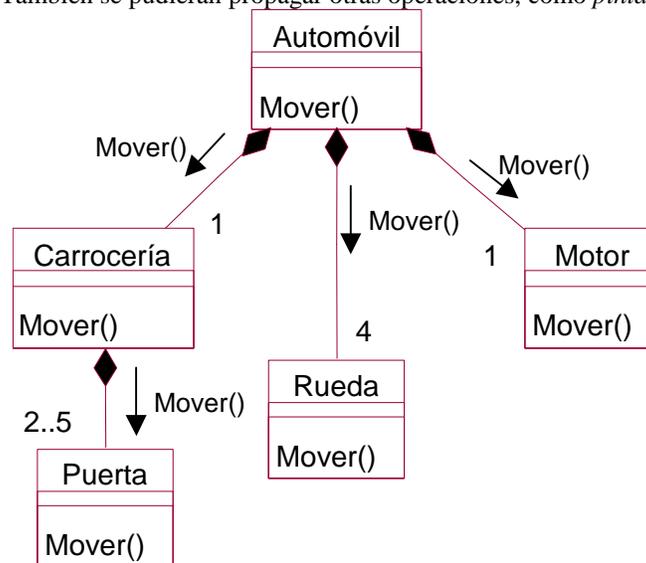


Figura 4.91. Diagrama de clases para la propagación de la operación mover del ensamblado *Automóvil* a *Carrocería*, y luego a *Puerta* la cual es parte a su vez de la *Carrocería*. La operación se propaga a todos los componentes.

4.7 Generalización y Herencia

Las clases con atributos y operaciones comunes se pueden organizar de forma jerárquica, mediante la *herencia*. La herencia es una abstracción importante para compartir similitudes entre clases, donde todos los atributos y operaciones comunes a varias clases se pueden compartir por medio de la *superclase*, una clase más general. Las clases más refinadas se conocen como las *subclases*.

Ejemplo: Las *Impresoras Láser*, de *Burbuja*, y de *Matriz*, son todas subclases de la superclase *Impresora*. Los atributos generales de una *Impresora* son el *Modelo*, *Velocidad*, y *Resolución*, mientras que sus operaciones son *Imprimir* y *Alimentar*.

Herencia es una relación "es-una" entre las clases las más refinadas y más generales.

Ejemplo: *Impresora Láser es una Impresora*.

Herencia es útil para el modelo conceptual al igual que para la implementación. Como modelo conceptual da buena estructuración a las clases. Como modelo de implementación es un buen vehículo para no replicar innecesariamente el código. *Generalización* define una relación entre una clase más *generalizada*, y una o más versiones refinadas de ella.

Ejemplo: La clase *Impresora* es una generalización de las clases *Impresoras Láser*, de *Burbuja*, y de *Matriz*.

Especialización define una relación entre una clase más general, y una o más versiones *especializadas* de ella.

Ejemplo: *Impresoras Láser*, de *Burbuja*, y de *Matriz*, son todas especializaciones de *Impresoras*.

La superclase *generaliza* a sus subclases, y las subclases *especializan* a la superclase. El proceso de especialización es el inverso de generalización. Una instancia de una subclase, o sea un objeto, es también una instancia de su superclase.

Ejemplo: Cuando se crea un objeto de tipo *Impresora Láser*, este objeto incluye toda la información descrita en la subclase *Impresora Láser*, al igual que en la superclase *Impresora*; por lo tanto se considera que el objeto es una instancia de ambas.

La herencia es transitiva a través de un número arbitrario de niveles. Los *ancestros* de una clase son las superclases de una clase en cualquier nivel superior de la jerarquía, y los *descendientes* de una clase son las subclases de una clase en cualquier nivel inferior de la jerarquía.

Ejemplo: Si además de *Impresora de Burbuja*, se define una clase más especializada como *Impresora de Burbuja Portátil*, entonces *Impresora* e *Impresora de Burbuja* son ancestros de la clase *Impresora de Burbuja Portátil*, mientras que *Impresora de Burbuja* e *Impresora de Burbuja Portátil* son descendientes de *Impresora*.

Las siguientes características se aplican a clases en una jerarquía de herencia:

?? Los valores de una instancia incluye valores para cada atributo de cada clase ancestra.

?? Cualquier operación de cualquier clase ancestra, se puede aplicar a una instancia.

?? Cada subclase no solo hereda todas las características de sus ancestros sino también añade sus propios atributos y operaciones.

(Los nombres de atributos y operaciones deben ser únicos en la jerarquía de herencia.)

Ejemplo: Una *Impresora de Burbuja Portátil* incorpora todas las características, primero de una *Impresora*, y luego de una *Impresora de Burbuja*, conteniendo valores para todos los atributos ancestrales y pudiéndose aplicar todas las operaciones ancestrales.

La generalización se puede extender a múltiples niveles de jerarquías, donde una clase hereda de su superclase, que a su vez hereda de otra superclase, hacia arriba en la jerarquía. En otras palabras, las relaciones entre subclases y superclases son relativas. La herencia define una jerarquía de clases donde existen *ancestros* y *descendientes*, que pueden ser directos o no.

Para representar herencia y generalización, se utiliza un triángulo conectando a la superclase con sus subclases. La superclase está del lado superior del vértice del triángulo, mientras que las subclases están en la parte inferior de la base del triángulo, como se muestra en la Figura 4.92.

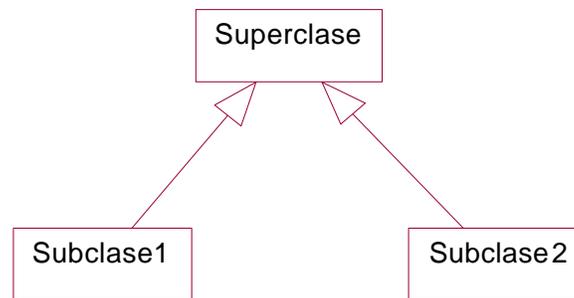


Figura 4.92. Notación en diagrama de clases para generalización y herencia.

Ejemplo: Un *Barco* tiene un *Nombre*, *Fabricante*, y *Costo*. Tipos especiales de *Barco*, como *Velero*, tienen además de estas características básicas, un *Número de Velas*, mientras que otro tipo especial de *Barco*, como *Barco a Motor*, tiene un *Motor*. *Barco* es la clase básica (la superclase) mientras que *Velero* y *Barco a Motor* son las clases refinadas (las subclases). Se debe definir las características básicas de los barcos una sola vez, y luego añadir detalles para veleros, barcos a motor, etc. En la Figura 4.93 se muestra el diagrama de clases describiendo la relación de herencia.

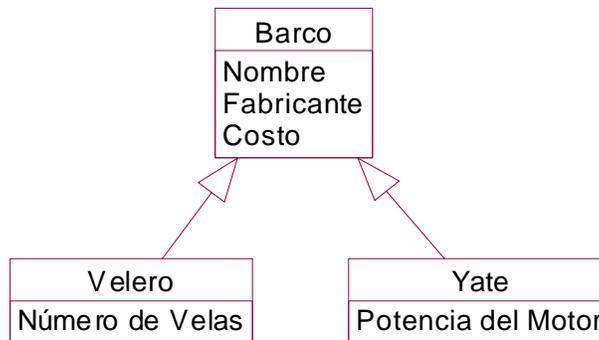


Figura 4.93. Diagrama de clases describiendo herencia de la superclase *Barco* a las subclases *Velero* y *Barco a Motor*. SE incluyen atributos para las diferentes clases.

Ejemplo: Una jerarquía conteniendo una superclase *Mueble*, y varias subclases *Mesa* y *Asiento*, puede ser extendida con nuevas subclases, como *Mesa Circular*, *Mesa Rectangular*, mientras que un *Asiento* puede extenderse con las subclases *Silla*, *Sillón*, y *Taburete*, como se muestra en la Figura 4.94. Cada clase tiene sus propios atributos los cuales se van especializando a medida que las clases son cada vez más especializadas. Nótese que no necesariamente todas las clases tienen que incluir atributos.

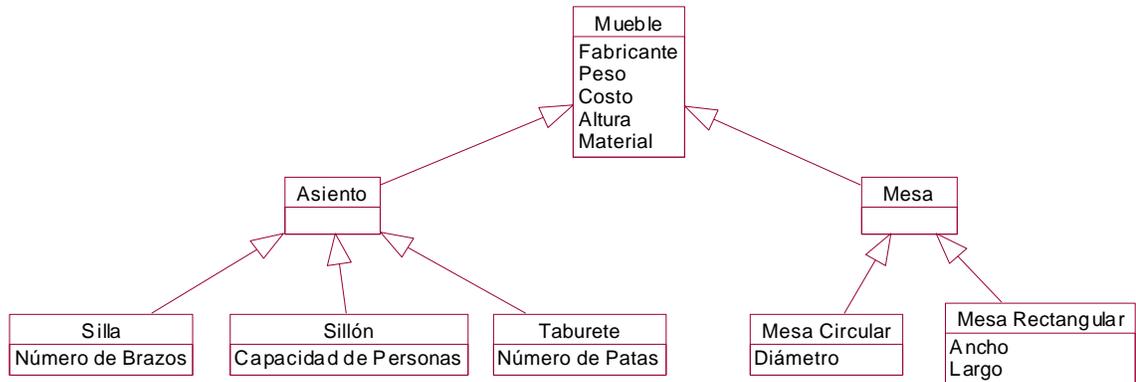


Figura 4.94. Diagrama de clases describiendo diferentes tipos de *Mueble*, *Asiento* y *Mesa*, con sus respectivas subclases.

Ejemplo: En la Figura 4.95 se muestran instancias de *Sillón* y *Mesa Circular* con valores para los distintos atributos.

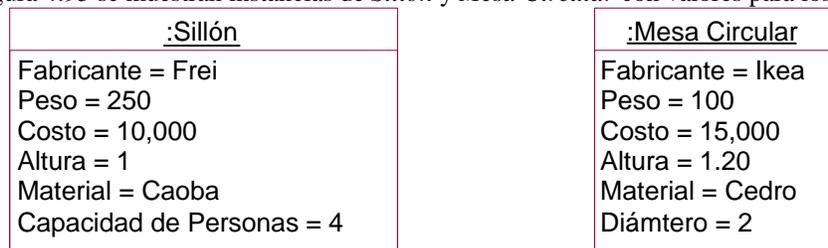


Figura 4.95. Diagrama de instancias para un *Sillón* y una *Mesa Circular*.

Agregación o composición no es lo mismo que generalización, el ensamblado relaciona clases correspondientes a muchos objetos, mientras que generalización relaciona clases las cuales finalmente corresponden a un solo objeto. Agregación o composición es una forma de estructurar la descripción de un solo objeto, mientras que con generalización, un solo objeto es una instancia de la combinación de su superclase y subclases. El ensamblado es una relación *parte-de*, en cambio generalización es una relación *tipo-de* o *es-una*.

Ejemplo: Un *Automóvil* está compuesto de un *Motor*, una *Carrocería*, y cuatro *Ruedas*. El *Automóvil* puede ser clasificado, como *Automóvil Deportivo* y *Automóvil Sedan*. Cada subclase puede tener sus propias partes, como *Rin de Magnesio* o *Parrilla para Maletas*. *Rin de Magnesio* es subclase de *Rin*, el cual es un componente de *Rueda*, al igual que *Llanta*. El diagrama se muestra en la Figura 4.96.

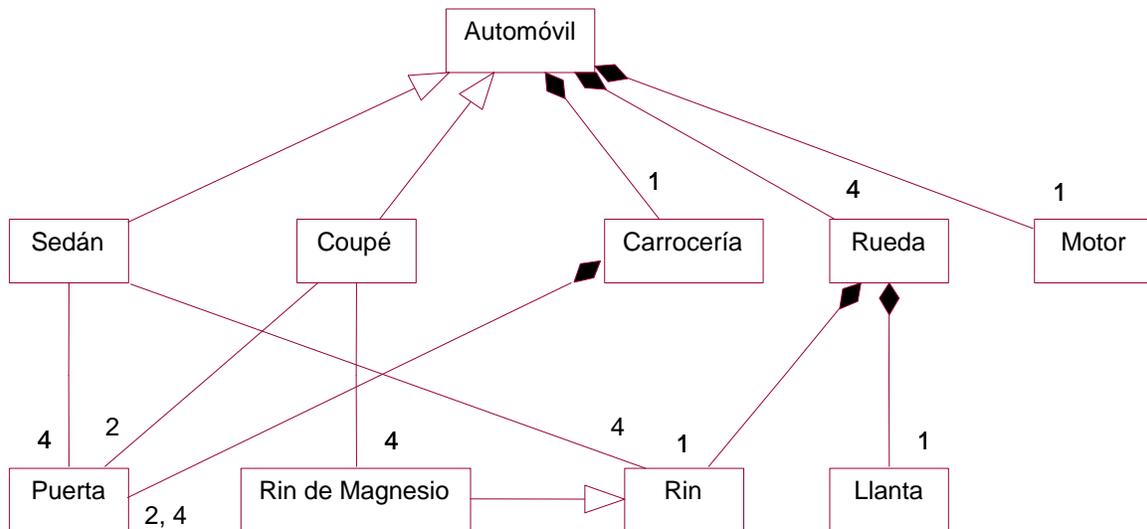


Figura 4.96. Diagrama de clases para un ensamblado describiendo un *Automóvil* que contiene varios componente: un *Motor*, una *Carrocería*, y cuatro *Ruedas*. Pueden haber diferentes tipos de *Automóvil*, *Deportivo* o *Sedan*, conteniendo *Rin de Magnesio* o una *Parrilla para Maletas*, respectivamente. *Rin de Magnesio* es subclase de *Rin*, el cual es componente de *Rueda* al igual que *Llanta*.

Ejemplo: Una clase *Ventana* tiene atributos para los vértices de la ventana y operaciones para *Desplegar*, *Ocultar*, *Mover* y *Modificar* la ventana. *Canvas*, *Panel*, y *Ventana de Texto* son tipos diferentes de *Ventanas*. Un *Canvas* se utiliza para diferentes despliegues gráficos, incluyendo atributos como el tamaño del elemento gráfico y operaciones para *Añadir* y *Borrar* tales elementos. El *Canvas* se relaciona con varios *Elementos (Formas)* que son *Líneas* o *Formas Cerradas*, como *Elipses* o *Polígonos*. Un *Polígono* consiste de una lista ordenada de *Puntos*. Un *Panel* contiene diferentes *Artículos de Panel*, los cuales pueden ser de tipo *Botón*, *Selección*, o *Texto*. Todos los *Artículos de Panel* están relacionados con *Eventos* del ratón, y el artículo de tipo *Texto* se relaciona además con un *Evento* del teclado. Cuando un *Artículo de Panel* se escoge, un *Evento* se genera. Una *Selección* se relaciona con diferentes *Posibles Selecciones*, aunque solo una puede escogerse a la vez. El diagrama se muestra en la Figura 4.97.

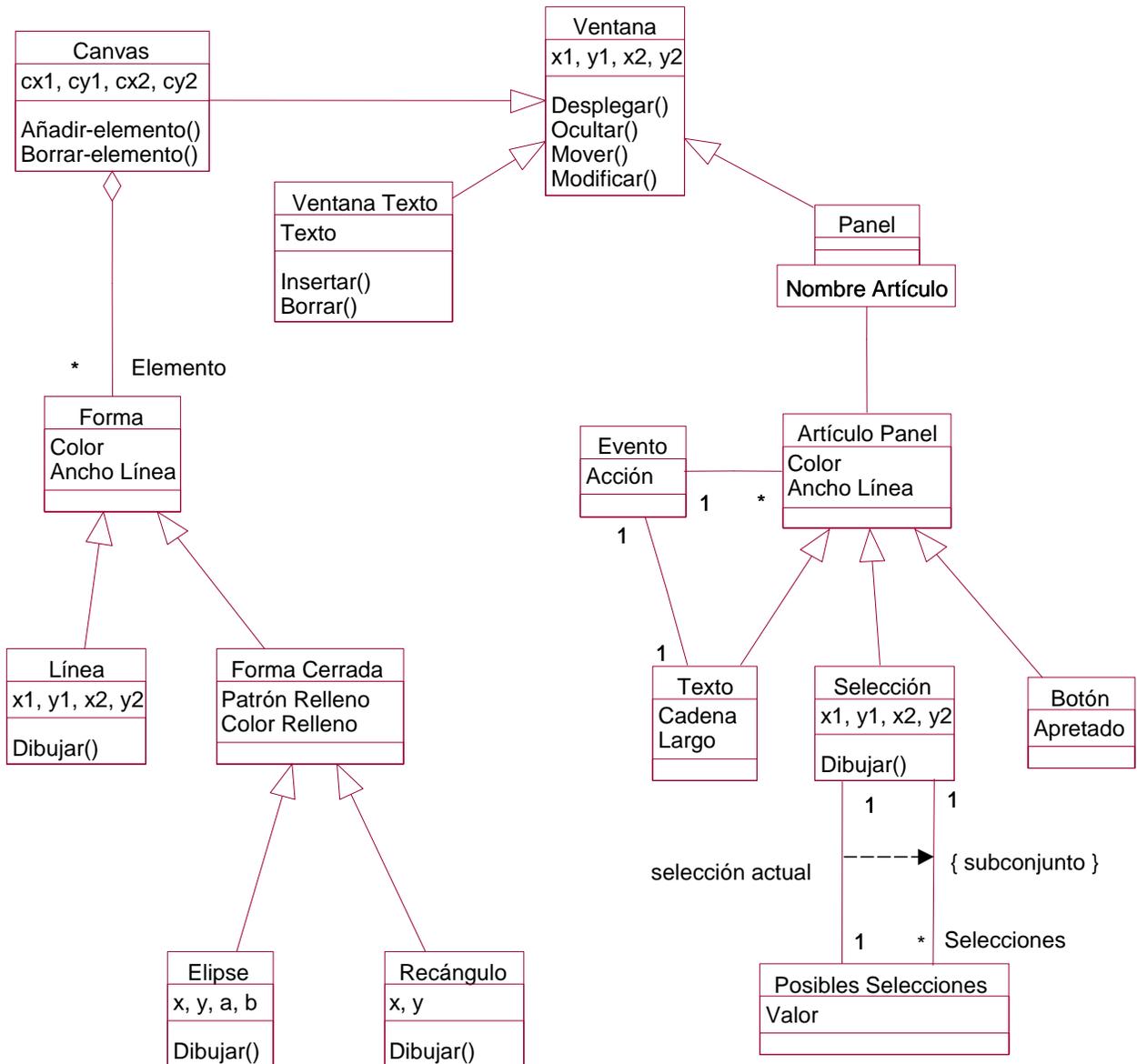


Figura 4.97. Diagrama de clases para un sistema de Ventanas.

Ejemplo: La clase *Persona* tiene un *Nombre*, *Dirección*, y *Número del Seguro Social*. Una persona puede trabajar en algún proyecto y ganar un salario. Una *Compañía* tiene un *Nombre*, *Dirección*, *Número de Teléfono*, y *Producto Primario*. Una *Compañía* contrata y despide *Personas*. *Persona* y *Compañía* tienen una relación "muchos-muchos". El título del trabajo depende de la persona y de la compañía. Hay dos tipos de *Personas*: *Trabajadores* y *Administradores*. Cada *Trabajador* está involucrado en varios *Proyectos*; cada *Administrador* es responsable de varios proyectos. En un proyecto pueden trabajar varios trabajadores y un solo administrador. Cada proyecto tiene un *Nombre*, *Presupuesto*, y una *Prioridad Interna* para asegurar recursos. Además una *Compañía* está compuesta de múltiples *Departamentos*; cada departamento dentro de una compañía se identifica de forma única por su *Nombre*. Un departamento usualmente tiene un administrador. La mayoría de los administradores manejan un departamento; y algunos administradores no están asignados a ningún departamento. Cada departamento manufactura varios productos; mientras que cada producto está hecho por un solo departamento. Un producto tiene *Nombre*, *Costo*, y *Peso*. El diagrama se muestra en la Figura 4.98.

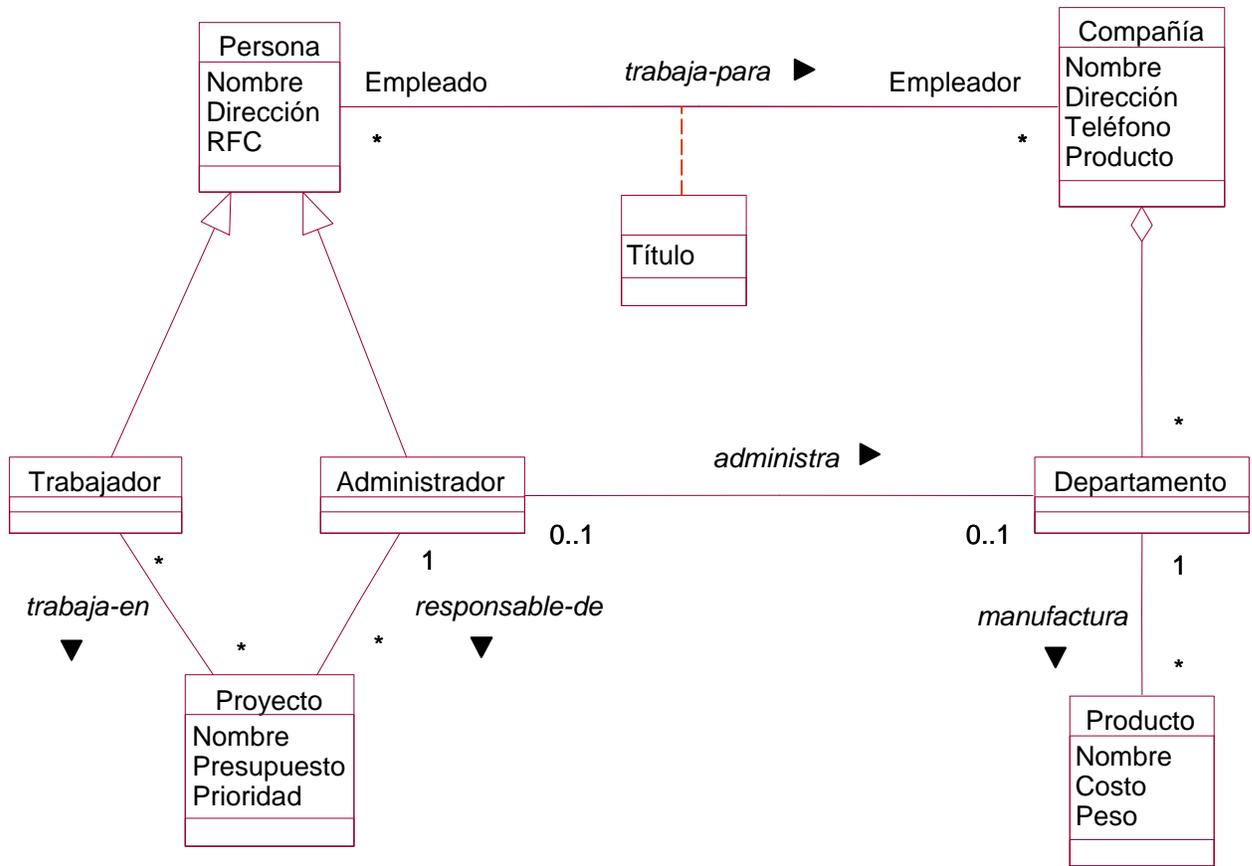


Figura 4.98. Diagrama de clases para *Personas* trabajando en *Compañías*.

La generalización puede hacerse por diferentes razones:

?? **Extensión:** clases definidas por operaciones con estructuras de información.

?? **Restricción:** clases como implementaciones de tipos, especializaciones, subconjunto de todas las instancias de un ancestro.

4.7.1 Extensión de Clase

La *extensión* de clase expresa que una subclase puede añadir nuevos atributos u operaciones a la superclase.

Ejemplo: La clase *Rectángulo* añade los nuevos atributos, *Ancho* y *Largo*, como se muestra en la Figura 4.99.

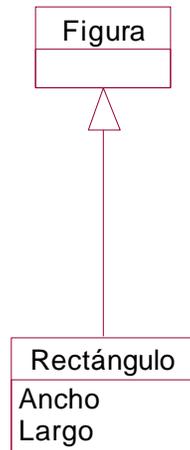


Figura 4.99. Diagrama de clases para una herencia describiendo una superclase *Figura* y una subclase *Rectángulo*.

4.7.2 Restricción de Clase

La *restricción* de clase indica como una subclase puede restringir los atributos heredados de una superclase. Una clase descendiente no puede suprimir los atributos u operaciones de sus ancestros.

Ejemplo: La clase *Cuadrado* restringe los atributos de *Rectángulo*, ya que *Ancho* debe ser igual a *Largo*, como se muestra en la Figura 4.100.

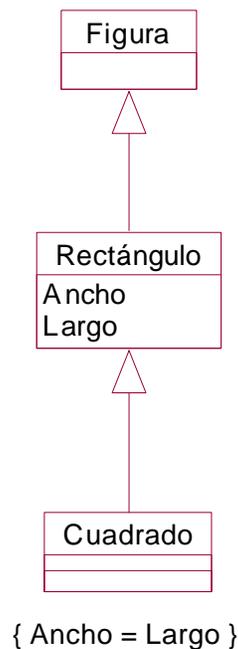


Figura 4.100. Diagrama de clases para una herencia describiendo una superclase *Figura*, una subclase *Rectángulo*, y una nueva subclase *Cuadrado*.

Cambios arbitrarios a los valores de los atributos pueden violar las restricciones de una clase creada por medio de restricciones. La restricción define la condición para la membresía en una clase, donde todos los objetos cuyos valores satisfacen la regla pertenecen a la clase.

Ejemplo: La clase *Cuadrado* debe suprimir la *Escala* desigual, ya que una escala arbitraria en las dimensiones de los ejes puede resultar en un *Rectángulo* y no en un *Cuadrado*. (La clase *Rectángulo* está cerrada bajo la operación de *Escala*, pero no la clase *Cuadrado*.)

4.7.3 Sobrescritura de Operaciones

Una subclase puede sobrescribir atributos y operaciones de una superclase, al definir nuevos atributos y operaciones con el mismo nombre. Se pueden sobrescribir atributos, por ejemplo, para redefinir sus valores por omisión, y se pueden sobrescribir operaciones para mejorar el rendimiento de un algoritmo. (No se puede sobrescribir las *firmas* de los atributos o de las operaciones).

Ejemplo: La operación *Desplegar* se define en la superclase *Figura* y se redefine (sobrescribe) en varias de las subclases, como *Punto* y *Línea*.

En general, sobrescritura va contra el principio de herencia, ya que se deja de heredar ciertos aspectos del ancestro. Sin embargo, la sobrescritura es parte fundamental de la orientación a objetos, en particular del polimorfismo.

Se puede sobrescribir operaciones por varias razones: *extensión*, *restricción*, *optimización* y *implementación*.

?? **Extensión.** En la *sobrescritura por extensión*, una nueva operación es igual a una heredada, excepto que agrega algún comportamiento nuevo afectando algunos de los atributos de la subclase. Ejemplo: La clase *Ventana* tiene una operación *Dibujar* para *Borde* y *Contenido*, mientras que la clase *Ventana-Etiquetada* tiene una operación *Dibujar* donde el método *Dibujar-Ventana-Etiquetada* llama al método *Dibujar* de *Ventana*, agregando código para dibujar la *Etiqueta*, como se muestran en la Figura 4.101.

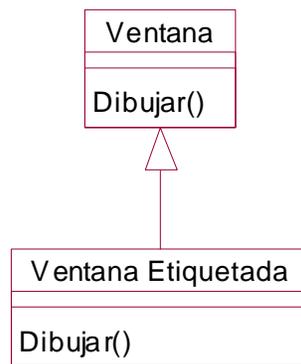


Figura 4.101. Diagrama de clases para una *Ventana* y una *Ventana-Etiquetada*.

?? **Restricción.** En la *sobrescritura por restricción*, una nueva operación restringe el protocolo de la operación (*firma*), como el tipo de argumentos. Ejemplo: Una superclase *Conjunto* puede tener una operación *Sumar(Objeto)*. La subclase *Conjunto-Entero* tiene una operación más restringida *Sumar(Entero)*. El diagrama se muestra en la Figura 4.102.

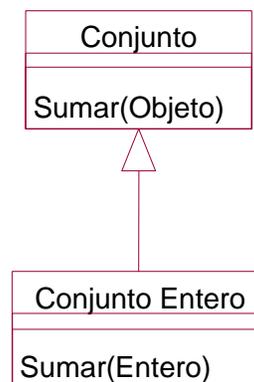


Figura 4.102. Diagrama de clases para un *Conjunto* y un *Conjunto-Entero*, restringiendo la operación *Sumar*.

?? **Optimización.** En la sobrescritura por optimización, la implementación de una clase puede aprovechar las restricciones impuestas para mejorar el código de la operación. El protocolo externo debe ser el mismo para la

nueva operación, aunque internamente sea totalmente diferente. Ejemplo: La superclase Conjunto-Entero puede tener una operación Máximo para encontrar el Entero más grande. El método podría ser implementado usando una búsqueda secuencial. La subclase Conjunto-Entero-Ordenado puede proveer una implementación más eficiente del método Máximo teniendo en cuenta que los números ya están ordenados, como se muestran en la Figura 4.103, optimizando la operación Máximo.

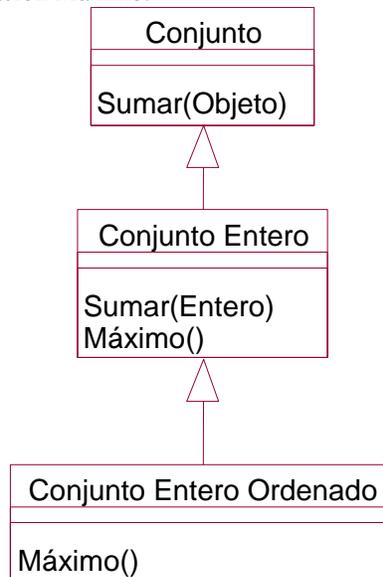


Figura 4.103. Diagrama de clases para un *Conjunto*, un *Conjunto-Entero*, y un *Conjunto-Entero-Ordenado*, optimizando la operación *Máximo*.

?? **Implementación.** Es malo desarrollar nuevas clases sobrescribiendo los métodos de las superclases simplemente por conveniencia de implementación, *sobrescritura por implementación*, sobre las cuales no existe ninguna relación conceptual. Se puede introducir herencia adicional en el sistema durante la etapa de diseño. Ejemplo: Las clases *Persona* y *Compañía* tienen ambos los atributos *Nombre* y *Dirección*, y se podría crear una superclase conteniendo los atributos comunes. Hacer esto durante la etapa del modelo de objetos es incorrecto, ya que las clases *Persona* y *Compañía* son semánticamente diferentes. Crear una superclase, *Dato-Común*, que contenga ambos atributos, como se muestra en la Figura 4.104, sería más bien una decisión hecha durante la etapa del diseño.

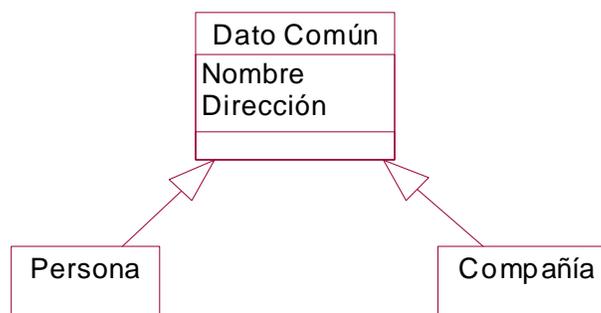


Figura 4.104. Diagrama de clases para una *Persona* y una *Compañía*, compartiendo atributos comunes guardados en la superclase *Dato-Común*.

4.7.4 Discriminador

Un *discriminador* indica el atributo de la superclase cuyo valor distingue a las subclases. El discriminador indica cual propiedad de la superclase es abstraída por las subclases en la herencia. (Sólo una propiedad debe ser discriminada a la vez.) La etiqueta "tipo" representa el discriminador más común, por lo cual a veces no se incluye. Si el discriminador es obvio no es necesario incluirlo. La notación, la cual es opcional, se muestra en la Figura 4.105.

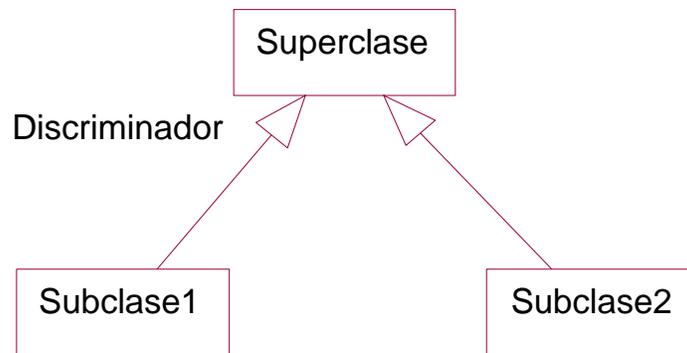


Figura 4.105. Notación para el diagrama de clases con generalización conteniendo discriminadores.

Ejemplo: Un *Asiento* se puede discriminar según su *funcionalidad*, en *Silla*, *Sillón*, o *Taburete*, como se muestra en la Figura 4.106.

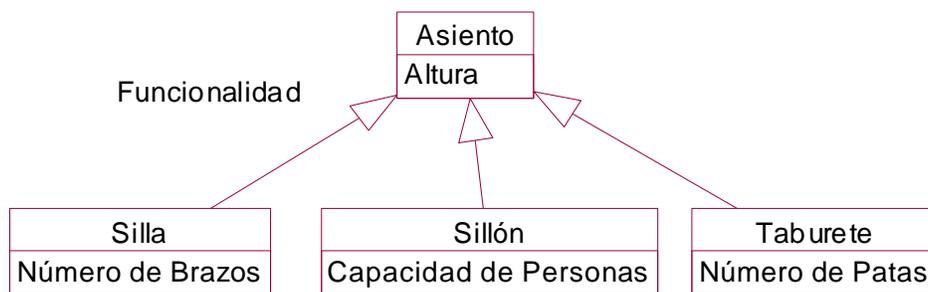


Figura 4.106. Diagrama de clases para diferentes tipos de *Asientos*, según su *funcionalidad*.

Ejemplo: Una *Mesa* se puede discriminar según su *forma*, en *Mesa Circular* o *Mesa Rectangular*, como se muestra en la Figura 4.107.

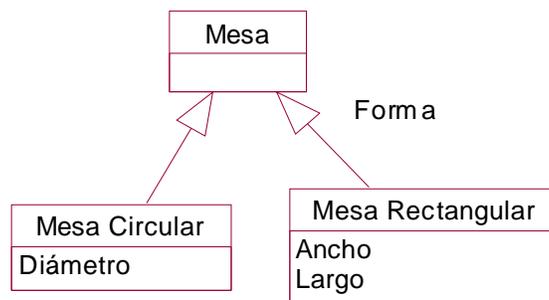


Figura 4.107. Diagrama de clases para diferente tipo de *Mesa* según su *forma*.

Ejemplo: En la Figura 4.108 se muestra un diagrama de clases para figuras gráficas geométricas. Las *Figuras* se discrimina según sus *dimensiones*, 0, 1, y 2. La operación *desplegar* se ha sobrescrito en todas las subclases de último nivel.

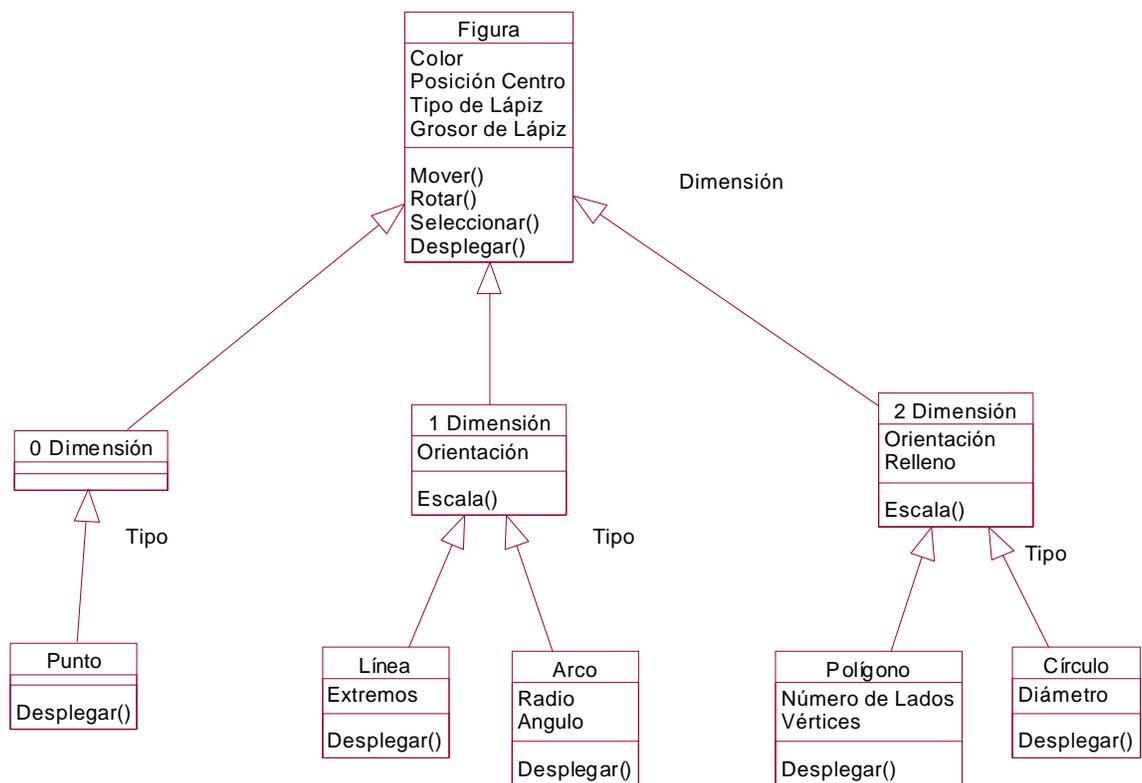


Figura 4.108. Diagrama de clases para diferente tipo de *Figuras*, según sus *dimensiones*.

4.7.5 Clases Abstractas

Hasta ahora todas las clases descritas anteriormente se conocían como *clases concretas* o simplemente *clases*. Donde cualquiera de estas clases tenía la cualidad de ser instanciable. En contraste, una *clase abstracta* es una clase que no tiene directamente instancias, pero que sus clases descendientes sí las tienen. Las clases abstractas organizan características comunes a varias clases; pueden aparecer naturalmente en la aplicación, o pueden ser introducidas artificialmente para promover el reuso de código, para compartir atributos y métodos.

Ejemplo: Para la clase *Mueble*, la clase *Asiento* es abstracta, ya que para crear un asiento se debe instanciar una de sus tres subclases, *Silla*, *Sillón*, o *Taburete*.

Cuando una superclase es dividida en subclases por un discriminador y existe una subclase para cada valor posible del discriminador, entonces la superclase se considera una *clase abstracta*. La clase abstracta puede definir métodos para ser usados por la subclase, o puede definir el protocolo de la operación, o sea el tipo y número de argumentos, el resultado, y la intención semántica, sin dar el correspondiente método (operación abstracta). Cada clase *concreta* debe proveer su propia implementación, y no debe tener operaciones abstractas.

Ejemplo: Ocupaciones, como *Ingeniero*, *Arquitecto*, y *Doctor*, son clases concretas. Una superclase *Trabajador* guardando aspectos comunes a todas las ocupaciones sería una clase abstracta, ya que se debe instanciar un trabajador con una ocupación particular, como se muestra en el diagrama de la Figura 4.109.

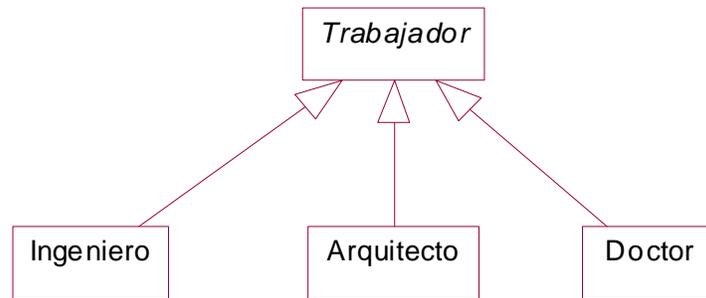


Figura 4.109. Diagrama de clases para una clase abstracta *Trabajador* y diferentes clases concretas *Ingeniero*, *Arquitecto*, y *Doctor*. (Nótese que el nombre de la clase *Trabajador* es en letras itálicas, correspondiente a una clase abstracta.)

Ejemplo: En el diagrama de la Figura 4.110, se muestra una variante del problema anterior donde *Trabajador* es una clase concreta, ya que para instanciar un *Trabajador* de tipo no especificado en el diagrama, se debe hacer una instancia de la superclase *Trabajador*.

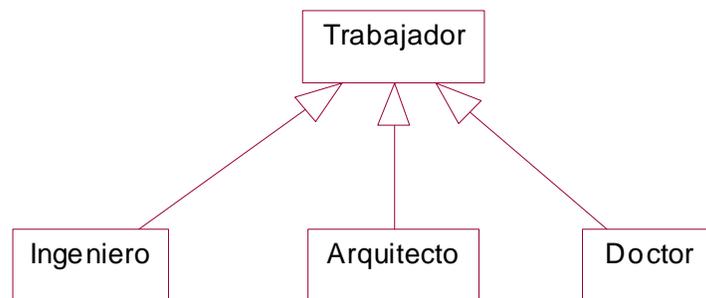


Figura 4.110. Diagrama de clases donde *Trabajador* es una clase concreta ya que además de poder instanciar objeto de las clases concretas *Ingeniero*, *Arquitecto*, y *Doctor*, también se pueden instanciar trabajadores no especificados en el diagrama. (Nótese que el nombre de la clase *Trabajador* es en letras normales, correspondiente a una clase concreta.)

Ejemplo: La clase *Empleado* es una clase abstracta, ya que los empleados deben especificarse si son *Honorarios* o *Nómina*, como se muestra en la Figura 4.111. La operación *Computar Pago* es una operación abstracta en la clase abstracta *Empleado*, requiriendo su implementación en las subclases de *Empleado*. Nótese que el nombre de la clase abstracta es en letras *cursivas*.

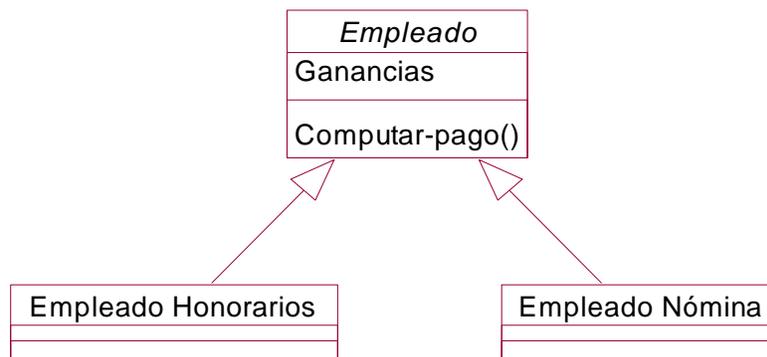


Figura 4.111. Diagrama de clases donde *Empleado* es una clase abstracta, mientras que *Empleado Honorarios*, y *Empleado Nómina* son clases concretas. La operación *Computar Pago* debe ser definida como una operación abstracta en la clase abstracta *Empleado*.

La jerarquía general de clases, para clases abstractas y concretas, se muestra en la Figura 4.112. El diagrama es muy conciso aunque un poco complicado ya que el concepto de *herencia* se describe utilizando un diagrama con herencia. Comenzando desde arriba y llenando hacia abajo en la jerarquía, podemos ver que la clase “Clase” puede especializarse en una “Clase Concreta” o en una “Clase Abstracta”. La “Clase Concreta”, que es una clase instanciable, se vuelve una “Clase Hoja” si no hay ninguna otra clase que herede de ella, mientras que la “Clase Nodo” contiene subclases adicionales. En otras palabras, la distinción entre una clase hoja y una clase nodo es su posición dentro del árbol de herencia, siendo ambas son instanciables. Por otro lado, la asociación “*Tiene-Subclase*” entre “Clase Nodo” y “Clase” muestra las posibles subclases de la jerarquía, comenzando nuevamente con “Clase”. De tal manera el árbol puede continuar indefinidamente mientras sigan existiendo una nueva “Clase Nodo” o una “Clase Abstracta”, que también es un nodo. Dado que la “Clase Abstracta” por definición no es instanciable, no se agrega un nivel de herencia adicional como se hizo con la “Clase Nodo”.

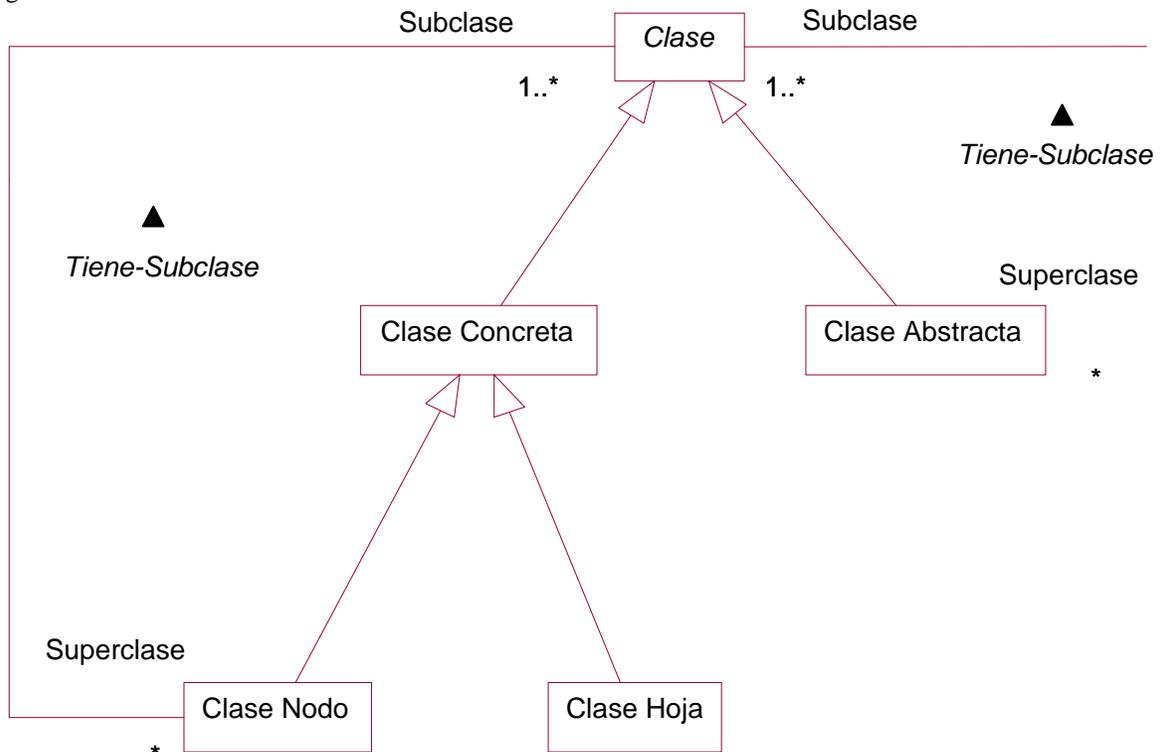


Figura 4.112. Jerarquía general de clases, describiendo la relación entre clases abstractas y concretas.

4.7.6 Herencia Múltiple

La herencia múltiple permite a una clase tener más de una superclase y heredar aspectos de todos sus ancestros. Ejemplo: Una jerarquía de clases conteniendo una superclase *Vehículo*, con dos subclases *Vehículo de Agua* y *Vehículo de Tierra*, y una subclase común a ambas llamado *Vehículo Anfibio*. El diagrama de muestra en la Figura 4.113.

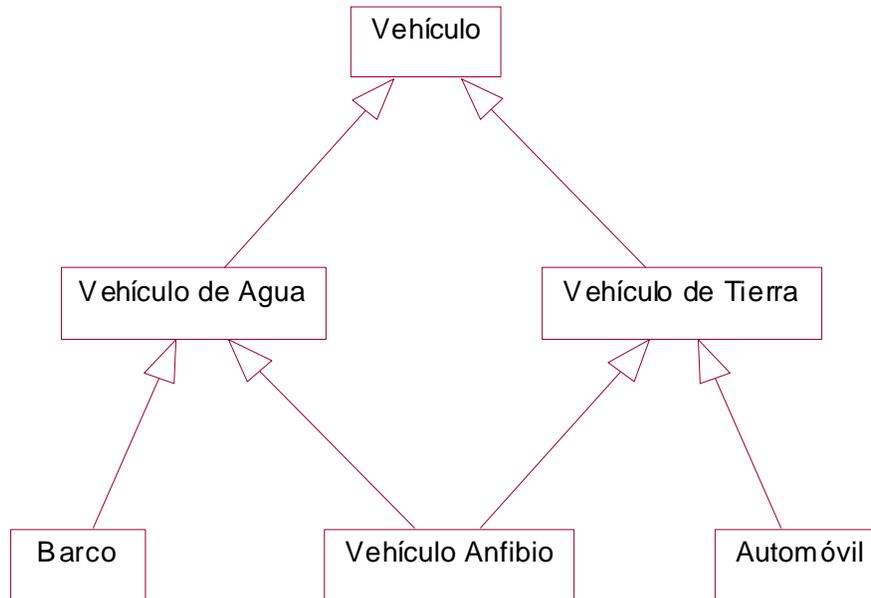


Figura 4.113. Diagrama de clases de herencia múltiple para *Vehículo*, conteniendo la clase *Vehículo Anfibio*, la cual hereda a la vez de *Vehículo de Agua* y *Vehículo de Tierra*.

En herencia sencilla no existen conflictos entre superclases. Cuando se hereda la misma operación de múltiples ancestros, como opción, se debe sobrescribir la operación en la nueva clase para evitar conflictos. Verificar que no se herede más de una vez aspectos de una clase (superclase común a varias jerarquías).

La ventaja de incorporar herencia múltiple, es que permite integrar información de dos o más superclases a la vez, dando más poder en la especificación de clases y más oportunidad de reuso, siendo por lo general más natural para el modelo de objetos que la herencia sencilla.

Por otro lado, la desventaja de incorporar herencia múltiple, es que es más complicada que herencia sencilla, ya que se pierde la simplicidad conceptual y de implementación. El mayor problema resultante es que se podría estar heredando varias veces una misma característica de diferentes clases ancestrales. En general, se pueden definir reglas para resolver ambigüedades y evitar conflictos entre las características heredadas por varios recorridos en la jerarquía de herencia.

Ejemplo: Un *Vehículo Anfibio* estaría heredando características comunes a todos los *Vehículo*, como *Velocidad*, de forma repetida, ya que heredaría tales características a través de *Vehículo de Agua* y *Vehículo de Tierra*.

Clases *disjuntas* son clases que semánticamente describen características diferentes, de diferente jerarquía de herencia (discriminadores diferentes).

Clases *no disjuntas* son clases que tienen aspectos comunes (mismo discriminador) por lo cual una clase que herede características de ambas estaría compartiendo tales propiedades. Se puede incorporar herencia múltiple de una misma generalización, si las clases dentro de la generalización son *no disjuntas*. (No se debe heredar de dos clases perteneciendo a una misma generalización si las clases dentro de la generalización son *disjuntas*.)

Ejemplo: La clase *Vehículo Anfibio* está heredando de una sola jerarquía de generalización, *Tipo de Vehículo*, por lo cual *Vehículo de Tierra* y *Vehículo de Agua* deben ser no disjuntos para que la herencia múltiple sea correcta, o sea, que exista la posibilidad de tener un vehículo que funcione en agua y tierra a la vez. Por otro lado, una clase *Empleado Honorario Asalariado* estaría heredando a la vez de las clases *Empleado Honorario* y *Empleado Asalariado*, siendo esto incorrecto ya que estas son clases disjuntas dentro de una misma jerarquía de generalización, o sea que es incorrecto que un empleado se le considere que gana por honorarios y también es asalariado. La decisión si dos clases en una misma generalización son disjuntas o no disjuntas depende de su interpretación.

Se puede incorporar herencia múltiple de dos generalizaciones distintas, donde las clases son *disjuntas*. Cada generalización debe cubrir una sola propiedad, por lo cual una nueva clase creada por medio de la herencia múltiple estaría siendo refinada sobre dimensiones de generalización diferentes e independientes.

Ejemplo: De la generalización de *Empleado* se puede crear dos jerarquías diferentes de herencia, una jerarquía se define según el *Tipo de Pago* del *Empleado*, dando lugar a *Empleado Honorario* y *Empleado Asalariado*. Otra jerarquía se define según el *Tipo de Pensión*, dando lugar a *Empleado Con Pensión* y *Empleado Sin Pensión*. Se puede definir una nueva clase *Empleado Honorario Con Pensión* el cual hereda características de dos clases disjuntas, *Empleado Honorario* y *Empleado Con Pensión*, de dos jerarquías de generalización independientes, y la herencia múltiple es correcta.

Ejemplo: En la Figura 4.114, se muestra la clase *Empleado Honorario Con Pensión* que hereda de dos generalizaciones diferentes, *Tipo de Pago* y *Tipo de Pensión*. Por lo tanto, *Empleado Honorario* y *Empleado Con Pensión* definen clases disjuntas de dos jerarquías de generalización independientes, y la herencia múltiple de *Empleado Honorario Con Pensión* es correcta.

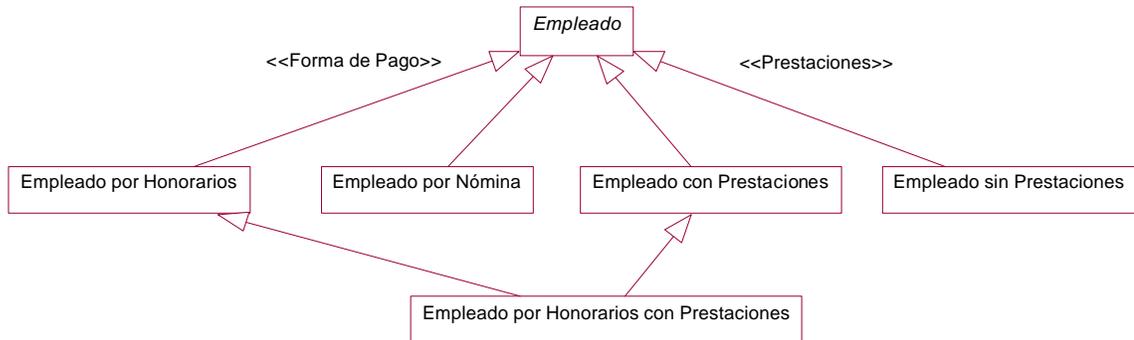


Figura 4.114. Diagrama de clases de herencia múltiple para *Empleado Honorario Con Pensión* heredando a la vez de *Empleado Honorario* y *Empleado Con Pensión*.

4.8 Módulos

Un *módulo* o *paquete* (“package”) es una construcción lógica para agrupar clases, asociaciones y generalizaciones. El módulo captura diferentes perspectivas de un sistema. Los bordes entre los diferentes módulos pueden ser bastante arbitrarios. Un modelo de objetos consiste de uno o más módulos. Los nombres de clases y asociaciones deben ser únicos en cada módulo, y se debe mantener consistencia entre los nombre de varios módulos. La misma clase puede aparecer en diferentes módulos, aunque debe ser definida solamente en uno de los módulos y referido en los otros. Deben haber menos conexiones entre módulos que asociaciones dentro de los módulos. En sistemas grandes la jerarquía de módulos puede ser de múltiples niveles.

Cada módulo debe proveer una visión de alto nivel de las clases más importantes del sistema, mostrando las clases y sus asociaciones sin atributos u operaciones. Cada una de estas clases se asigna a su propio módulo, mostrando su descomposición en clases por generalización y agregación. En la Figura 4.115 se muestra la notación para un módulo o paquete en UML. Nótese que el módulo no tiene ninguna propiedad, a diferencia de la clase. Sirve únicamente como elemento organizacional de las clases.



Figura 4.115. Notación para un módulo en UML.

Por ejemplo, modelos en forma de "estrella" son bastante comunes, donde la estructura de alto nivel está en un módulo, y otros módulos expanden las superclases con generalización y añaden asociaciones a clases de bajo nivel. Para integrar estos y los conceptos anteriores que se han descrito a lo largo del capítulo, presentamos en esta sección un ejemplo mas completo de un *Automóvil* correspondiente al diagrama de la Figura 4.116.



Figura 4.116. Diagrama para un módulo de un *Automóvil*.

Los módulos en los cuales está subdividido el módulo principal del Automóvil son los siguientes: *Carrocería*, *Sistema de Frenos*, *Sistema de Alimentación*, *Sistema Eléctrico*, y *Sistema Mecánico*, como se muestra en la Figura 4.117.

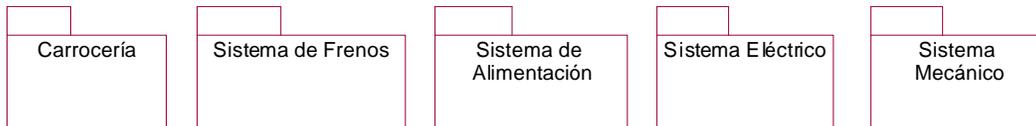


Figura 4.117. Módulos que componen el módulo principal del Automóvil: *Carrocería*, *Sistema de Frenos*, *Sistema de Alimentación*, *Sistema Eléctrico*, y *Sistema Mecánico*.

Estos módulos son descritos con mayor detalle a continuación y cada uno por separado. En el capítulo 6 veremos como se pueden aplicar el concepto del módulo para organizar sistemas de mayor complejidad. Vale la pena notar que esta descripción de un automóvil corresponde al “dominio” de una aplicación, en este caso del automóvil. En el diagrama de la Figura 4.118 se muestra los módulos descritos como clases. Esta descripción puede coexistir con la de los módulos anteriores si así se desea, donde cada una de estas clases se asigna al módulo con su mismo nombre.

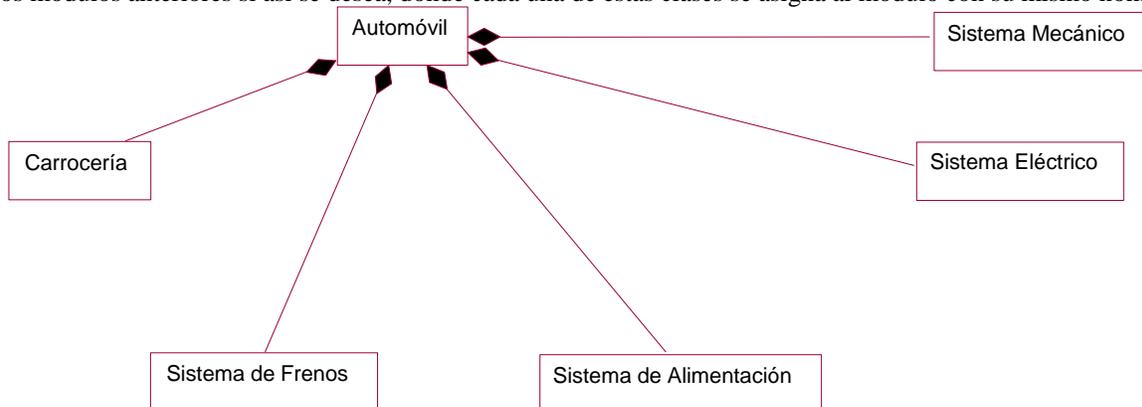


Figura 4.118. Clases para el módulo de un *Automóvil*: *Carrocería*, *Sistema de Frenos*, *Sistema de Alimentación*, *Sistema Eléctrico*, y *Sistema Mecánico*.

El módulo de la *Carrocería* incluye el *Chasis*, la *Cajuela*, el *Cofre*, las *Puertas* y las *Ventanas*, las cuales pueden ser partes de las *Puertas*, como se muestra en la Figura 4.119.

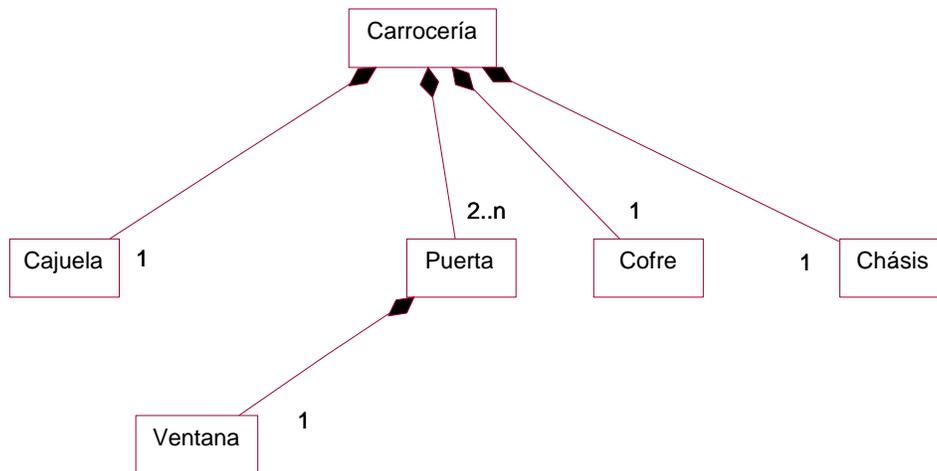


Figura 4.119. Módulo de la *Carrocería*, el cual está compuesto por el *Chasis*, la *Cajuela*, el *Cofre*, las *Puertas*, y las *Ventanas*.

El módulo del *Sistema de Freno* incluye el *Freno de Mano*, el *Pedal de Freno*, los *Frenos de Tambor* y de *Disco*, y las *Balatas*, como se muestra en la Figura 4.120.

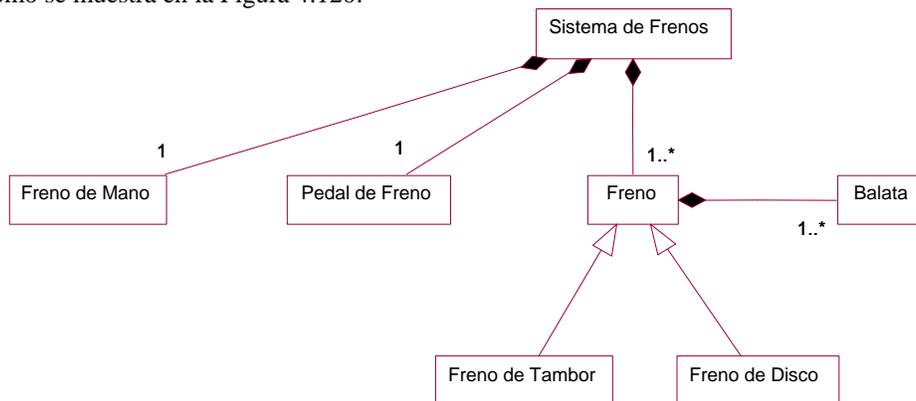


Figura 4.120. Módulo del *Sistema de Freno*, que incluye el *Freno de Mano*, el *Pedal de Freno*, los *Frenos de Tambor* y de *Disco*, y las *Balatas*.

El módulo de *Alimentación* incluye el *Tanque de Gasolina*, la *Bomba de Gasolina*, el *Filtro de Aire* y de *Gasolina*, y el *Carburador* o la *Inyección*, como se muestra en la Figura 4.121.

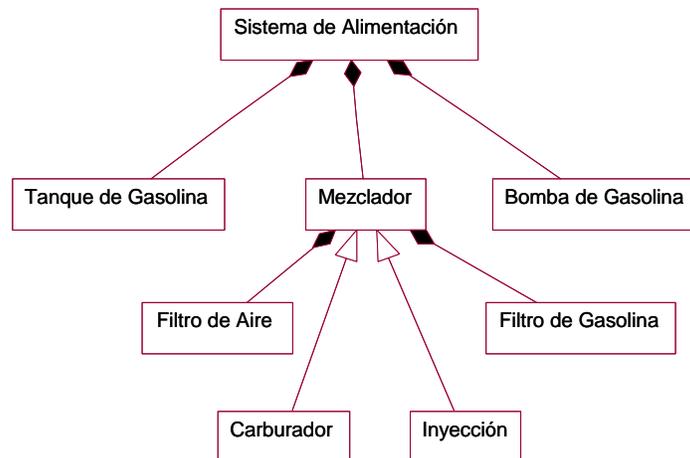


Figura 4.121. Módulo de Alimentación, que incluye el Tanque de Gasolina, la Bomba de Gasolina, el Mezclador conteniendo el Filtro de Aire y de Gasolina, y que se especializa en el Carburador o la Inyección.

El módulo Eléctrico incluye el Distribuidor, la Bobina, la Batería, el Interruptor de Arranque, el Regulador de Voltaje, el Alternador, y las Bujías, como se muestra en la Figura 4.122.

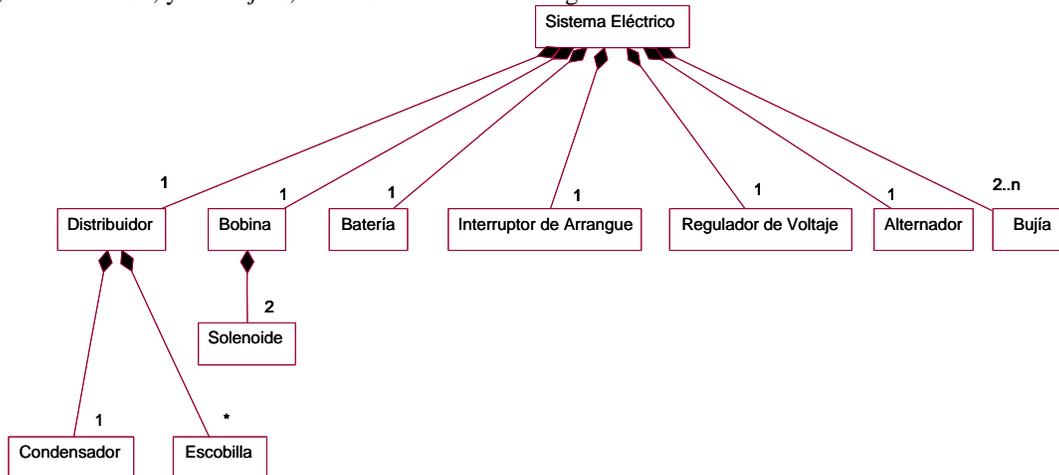


Figura 4.122. Módulo Eléctrico, que incluye el Distribuidor, la Bobina compuesta por dos Solenoides, la Batería, el Interruptor de Arranque, el Regulador de Voltaje, el Alternador, y las Bujías.

El módulo Sistema Mecánico incluye los módulos de Sistema de Transmisión, Sistema de Lubricación, Sistema Motriz, Sistema de Enfriamiento, Sistema de Suspensión, y Sistema de Dirección, como se muestra en la Figura 4.123.



Figura 4.123 Módulo Mecánico que incluye los módulos de Sistema de Transmisión, Sistema de Lubricación, Sistema Motriz, Sistema de Enfriamiento, Sistema de Suspensión, y Sistema de Dirección.

Estos módulos Sistema Mecánico también pueden ser descritos como clases: Sistema de Transmisión, Sistema de Lubricación, Sistema Motriz, Sistema de Enfriamiento, Sistema de Suspensión, y Sistema de Dirección, como se muestra en la Figura 4.124.

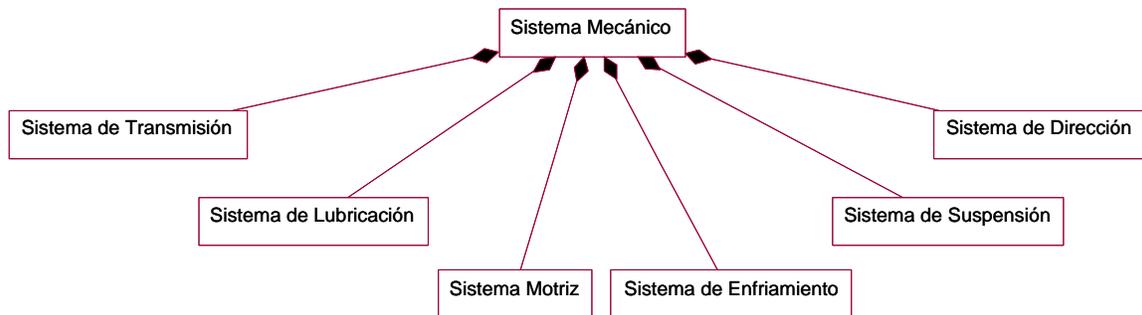


Figura 4.124 El módulo *Sistema Mecánico* que incluye las clases de *Sistema de Transmisión*, *Sistema de Lubricación*, *Sistema Motriz*, *Sistema de Enfriamiento*, *Sistema de Suspensión*, y *Sistema de Dirección*.

El módulo de *Transmisión* incluye la *Palanca de Velocidades*, la *Caja de Velocidades*, el *Overdrive*, la *Transmisión Manual* o *Automática*, y el *Embriague*, como se muestra en la Figura 4.125.

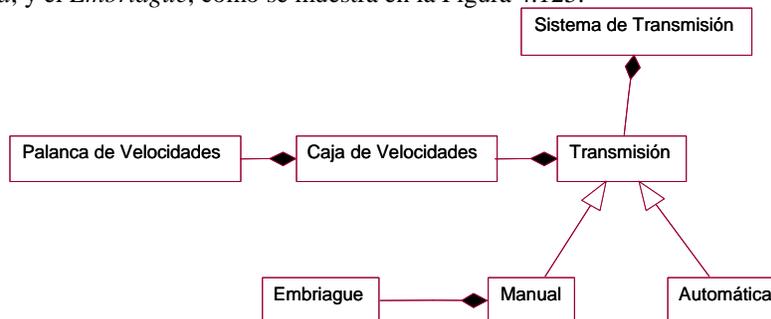


Figura 4.125. Módulo de *Transmisión*, que incluye la *Palanca de Velocidades*, la *Caja de Velocidades*, el *Overdrive*, la *Transmisión Manual* o *Automática*, y el *Embriague*.

El módulo de *Lubricación* incluye la *Bomba de Aceite*, y el *Cárter*, como se muestra en la Figura 4.126.

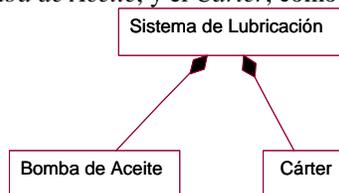


Figura 4.126. Módulo de *Lubricación*, que incluye la *Bomba de Aceite*, y el *Cárter*.

El módulo *Motriz* incluye el *Cigüeñal*, el *Arbol de Levas*, los *Pistones*, y los *Pistones*, como se muestra en la Figura 4.127.

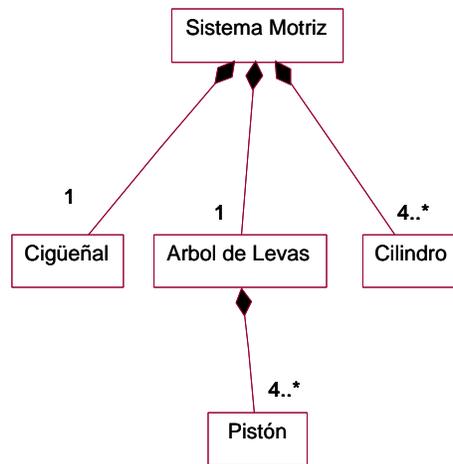


Figura 4.127. Módulo *Motriz*, que incluye el *Cigüeñal*, el *Arbol de Levas*, el *Cilindro*, y los *Pistones*.

El módulo de *Enfriamiento* incluye el *Radiador*, el *Termostato*, el *Ventilador*, y la *Bomba de Agua*, como se muestra en la Figura 4.128.

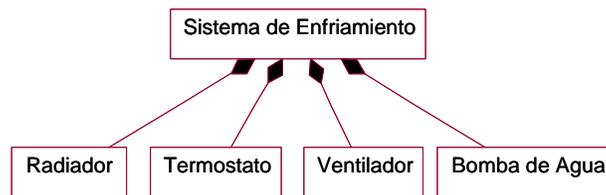


Figura 4.128. Módulo de *Enfriamiento*, que incluye el *Radiador*, el *Termostato*, el *Ventilador*, y la *Bomba de Agua*.

El módulo de *Suspensión* incluye los *Amortiguadores*, la *Barra Estabilizadora*, y los *Resortes*, como se muestra en la Figura 4.129.

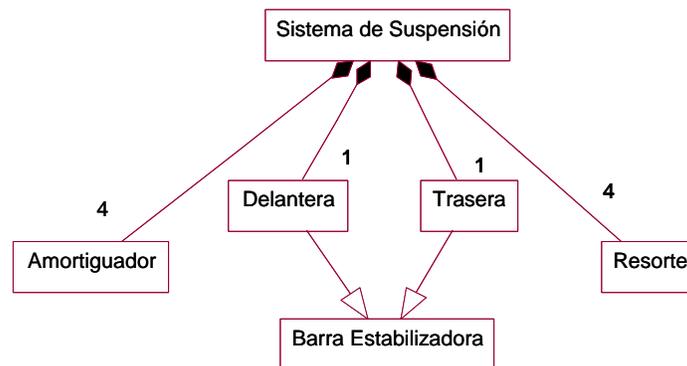


Figura 4.129. Módulo de *Suspensión*, que incluye los *Amortiguadores*, la *Barra Estabilizadora*, y los *Resortes*.

El módulo de *Dirección* incluye la *Caja de Dirección*, el *Eje*, y el *Volante*, como se muestra en la Figura 4.130.

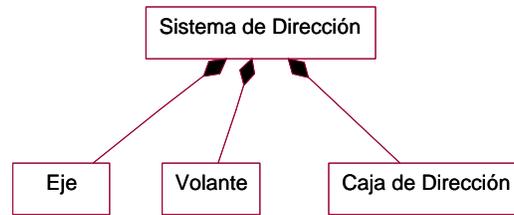


Figura 4.130. Módulo de *Dirección*, que incluye la *Caja de Dirección*, el *Eje*, y el *Volante*.

Ya que el diagrama de un modelo complejo puede ser muy grande, los módulos se deben diagramar en una o más *páginas*. Por lo general no se debe incluir más de un módulo por página, siendo el uso de páginas una conveniencia de notación y no un aspecto de lógica. Las asociaciones y generalizaciones deben aparecer en una sola página, mientras que ciertas clases, pueden aparecer en múltiples páginas para servir de vínculo entre ellas. Se busca minimizar el número de estas clases puente.

