2. Clases y Objetos

2.1. Conceptos Básicos

2.1.1. Clases, sus Características y Comportamiento

Una clase es una agrupación de objetos de un mismo tipo, que tienen un conjunto de **características** y/o un **comportamiento** común. Una clase es una representación abstracta de algo.

Ejemplo de clases:

- Asignatura
- Rectángulo
- Producto
- Medicamento
- Automóvil
- Persona
- Libro
- Electrodoméstico
- Estrella
- Préstamo (en una biblioteca, p.e.)

Las **características** de una clase corresponden a los atributos comunes que poseen todos los objetos que pertenecen a ella. Estos atributos asumen valores dentro de un conjunto dependiendo de la naturaleza de los mismos (por ejemplo: numérico, string, etc.)

Ejemplos:

- Una clase Asignatura podría tener los siguientes atributos: código, nombre, número de créditos SCT.
- Una clase Rectángulo podría tener los siguientes atributos: largo, alto.
- Una clase Producto podría tener los siguientes atributos: código, nombre, stock actual, stock mínimo.
- Una clase Libro podría tener los siguientes atributos: nombre, número de páginas y género. La Figura 1 muestra la representación en UML de la clase Libro y sus atributos.

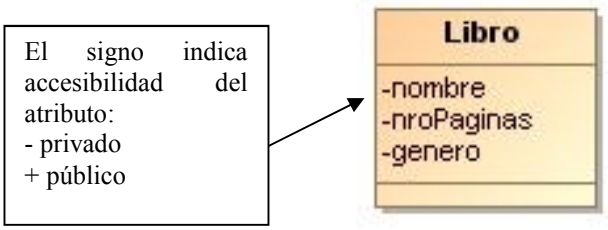

Figura 1. Representación UML de la clase Libro y sus atributos.

Los atributos nombre, nroPaginas y genero son todos privados, ello se representa mediante el signo menos que antecede el nombre del atributo. La accesibilidad de los atributos es relevante al momento de definir el comportamiento de una clase.

El **comportamiento** de una clase son las operaciones que la clase tiene disponible para otras clases que interactúan con ella. Estas operaciones se llaman métodos y, en general, utilizan los atributos de la clase.

Ejemplos:

- La clase Asignatura podría tener las siguientes operaciones: obtiene código, obtiene nombre, modifica nombre, entre otros.
- La clase Rectángulo podría tener las siguientes operaciones: calcula área, calcula perímetro, obtiene largo, cambia tamaño.
- La clase Producto podría tener las siguientes operaciones: obtiene nombre, aumenta stock, disminuye stock.
- La clase Libro que podría tener las siguientes operaciones: obtiene nombre, cambia género. La figura 2, muestra la representación en UML de la clase Libro, incluyendo los atributos y el comportamiento indicado.

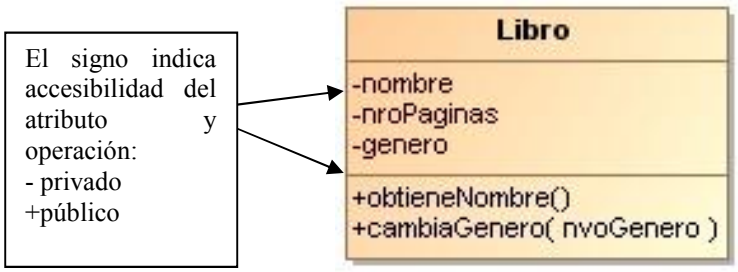

Figura 2. Representación UML de la clase Libro, sus atributos y operaciones.

En la Figura 2 las operaciones obtieneNombre y cambiaGenero son públicas, ello se representa mediante el signo más que antecede el nombre de la operación.

2.1.2. Objetos, sus Características y Comportamiento

Un **objeto** es una instancia o ejemplar de una clase, pudiendo ser una persona, lugar o cosa (objetos tangibles), o una organización, rol, incidente, interacción (objetos conceptuales).

Ejemplo de objetos:
- Programación Orientada a Objetos, código 634172, objeto de la clase Asignatura.
- Largo: 12, ancho: 4, es un objeto de la clase Rectángulo.
- Cereales Cuadritos de Quaker, objeto de la clase Producto.

- Aspirina objeto de la clase Medicamento.
- Chevrolet corsa, color gris, año 2012, objeto de la clase Automóvil.
- Juan Oliva, rut: 20.089.067-0, objeto de la clase Persona.
- Cómo Programar en Java, 8va. Edición, Deitel & Deitel, objeto de la clase Libro.
- Plancha de vapor Phillips Ph250e, objeto de la clase Electrodoméstico.
- Sol objeto de la clase Estrella.
- Préstamo a <usuario de la biblioteca> de <material bibliográfico> realizado el 30/07/2013 para ser devuelto el 05/08/2013, objeto de la clase Préstamo.

Dado que un objeto es la instancia de una clase, dicho objeto poseerá las **características y el comportamiento de la clase.** Los atributos que representan las características de una clase poseen valores específicos y concretos. Debe tenerse presente que un objeto es una instancia de una única clase.

Ejemplo:
La Figura 3 presenta la representación UML de la clase Producto.

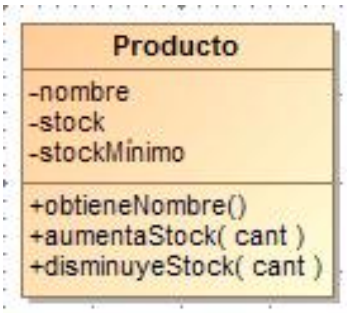

Figura 3. Representación UML de la clase Producto.

La Figura 4 presenta tres objetos correspondientes a instancias de la clase Producto.

Obsérvese cómo los objetos de la Figura 4 se diferencian en sus atributos, particularmente en el valor del atributo nombre (el cual se espera sea diferente para cada objeto de esta clase), en cambio las operaciones son siempre las mismas.

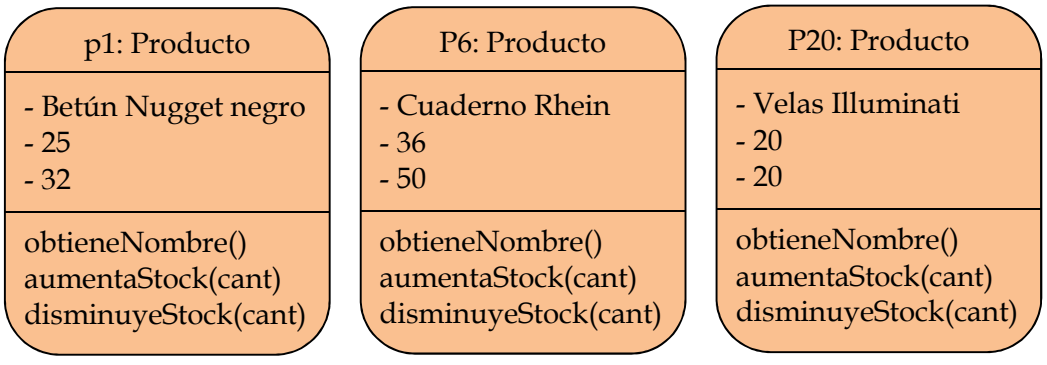

Figura 4. Tres objetos de la clase Producto.

Para implementar clases en un lenguaje de programación se agrega una operación adicional que permite crear objetos de dichas clases, a esa operación se le denomina constructor. A través de esta operación, usualmente, se inicializan los atributos de los objetos que se crean. En las figuras que siguen se han incorporado constructores en las clases que, por lo general, aparecerán como primera operación antecediéndole el elemento <<constructor>> (ver Figura 5). Dicho elemento se conoce como estereotipo en la notación UML.

2.2. Relaciones entre Clases / Objetos

Una aplicación en ejecución (más aún un sistema de software conformado por múltiples aplicaciones) se puede describir como un conjunto de objetos colaborando para desarrollar una tarea dada. Dichas interacciones se producen debido a las relaciones que existen entre tales objetos. Las relaciones entre objetos son: Asociación, Agregación/composición y Herencia. A continuación se describen dichas relaciones haciendo uso de diagramas de clases UML.

> ***¡Importante!***
> Si dos objetos tienen una relación, entonces las clases que los definen tienen tal relación y, corresponde, al definir tales clases, indicar la relación.

2.2.1. Asociación

La asociación se produce entre dos o más objetos. La más simple es una asociación entre dos objetos, denominada asociación binaria. Una asociación es la más general de las relaciones y se produce cuando un objeto interactúa con otro debido a una transacción propia del ámbito del problema, por lo tanto, el nombre de la asociación será el verbo que represente tal transacción. A continuación se presenta un ejemplo[1].

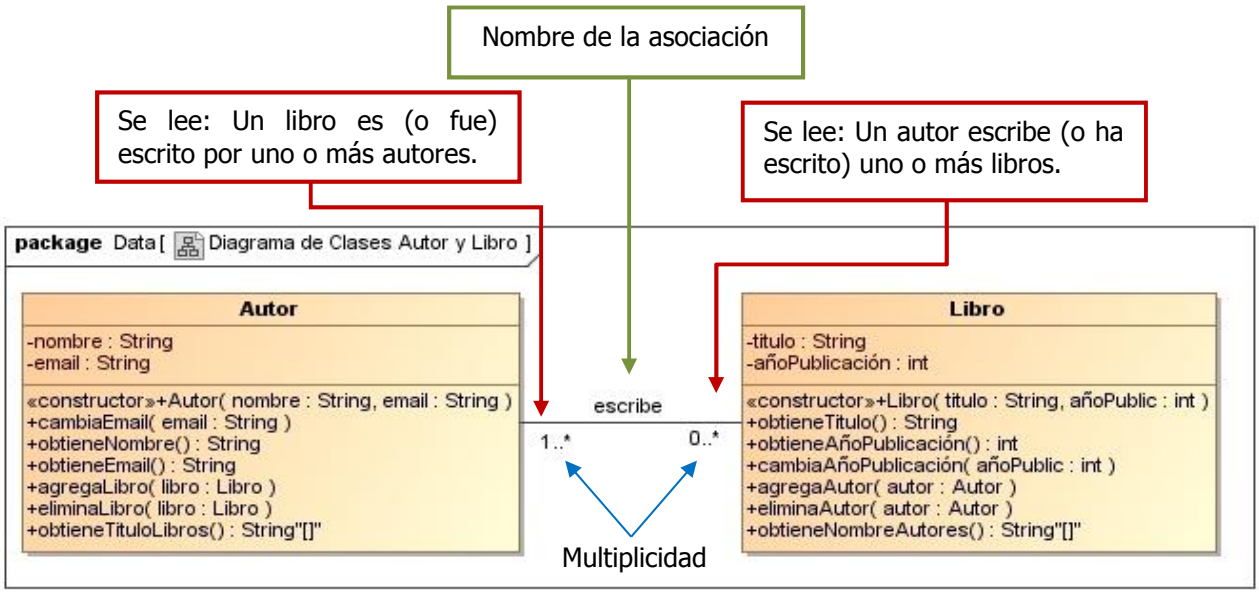

Figura 5. Asociación entre las clases Autor y Libro.

[1] Los diagramas que se presentan fueron generados con la herramienta MagicDraw UML Personal Edition 16.6 (demo de evaluación).

En este caso la asociación entre Autor y Libro se produce porque un autor "escribe" libros. En el diagrama se indica que un autor escribe uno o más libros y que un libro puede haber sido escrito por uno o más autores.

La multiplicidad, como se desprende de la Figura 5, indica cuántas veces puede relacionarse un objeto de una clase con otros de otra clase. Las relaciones de asociación pueden presentar diversas multiplicidades: uno y sólo uno (1..1), cero o uno (0..1), uno o más (1..*), cero ó más (0..*).

Otro aspecto relevante es que pueden darse asociaciones múltiples, esto es, dos o más asociaciones entre dos objetos (ver Figura 6). Por otra parte, un objeto puede estar asociado consigo mismo (ver Figura 7).

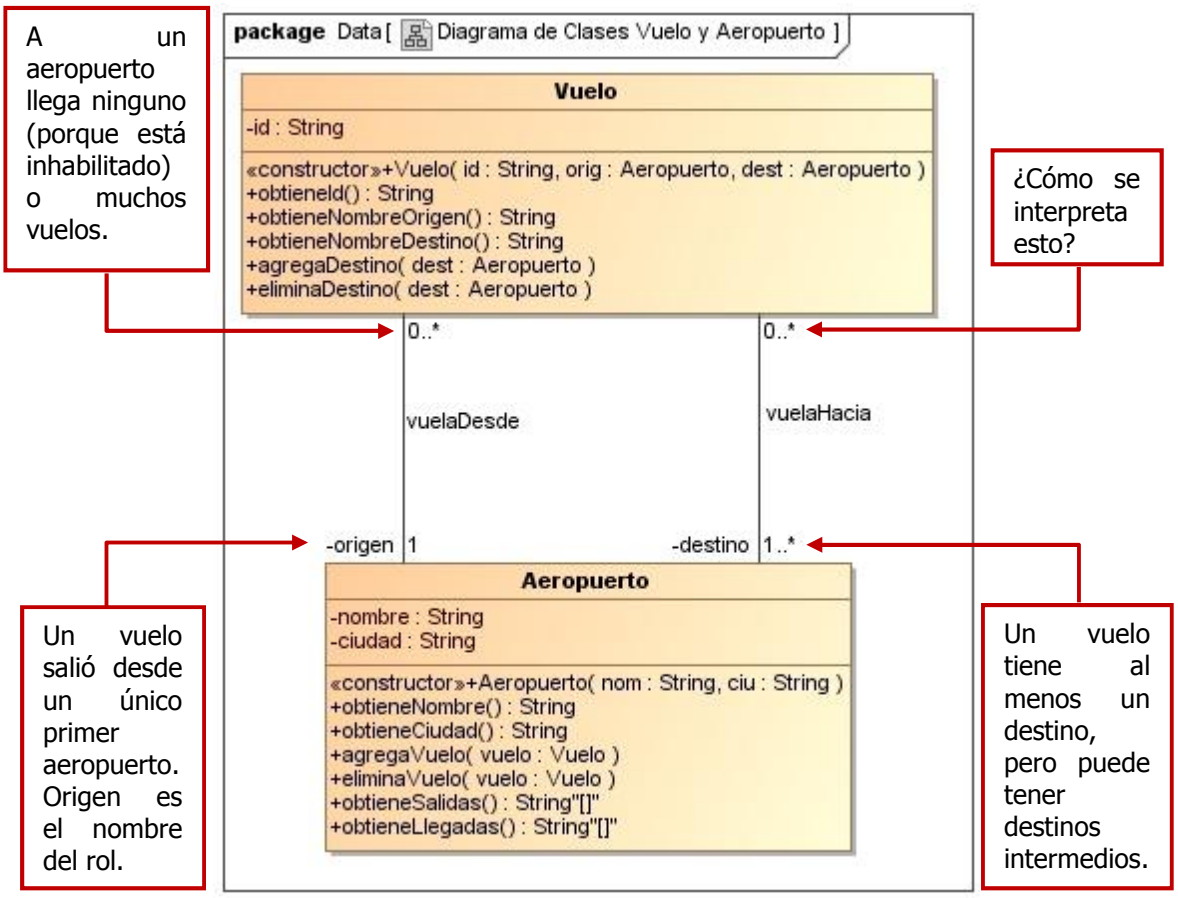

Figura 6. Doble relación entre dos clases.

> ***¡Importante!***
> En los diagramas, cuando en una relación no aparece expresamente la multiplicidad, ella ha de interpretarse como 1..1 (uno y sólo uno).

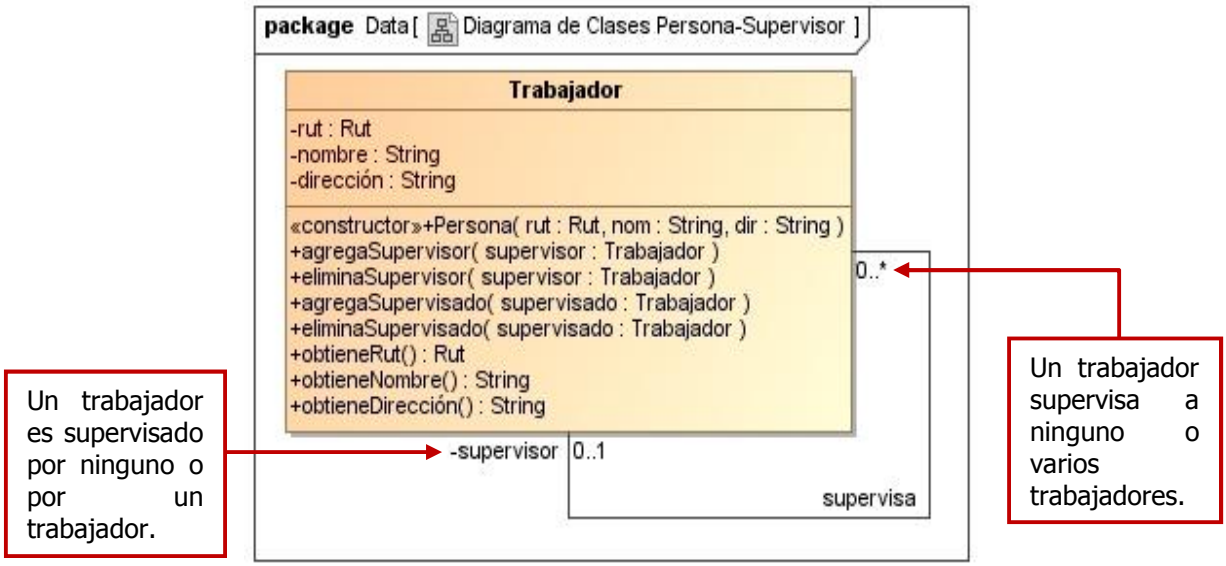

Figura 7. Asociación de una clase consigo misma.

2.2.2. Agregación

Una *agregación* es una relación que permite indicar si un objeto (el todo) está conformado por otros objetos (sus partes). Por lo general, esta relación no lleva nombre en un diagrama. En la Figura 8 se presenta un ejemplo que señala que los objetos de la clase LineaRecta se definen por dos objetos de la clase Punto.

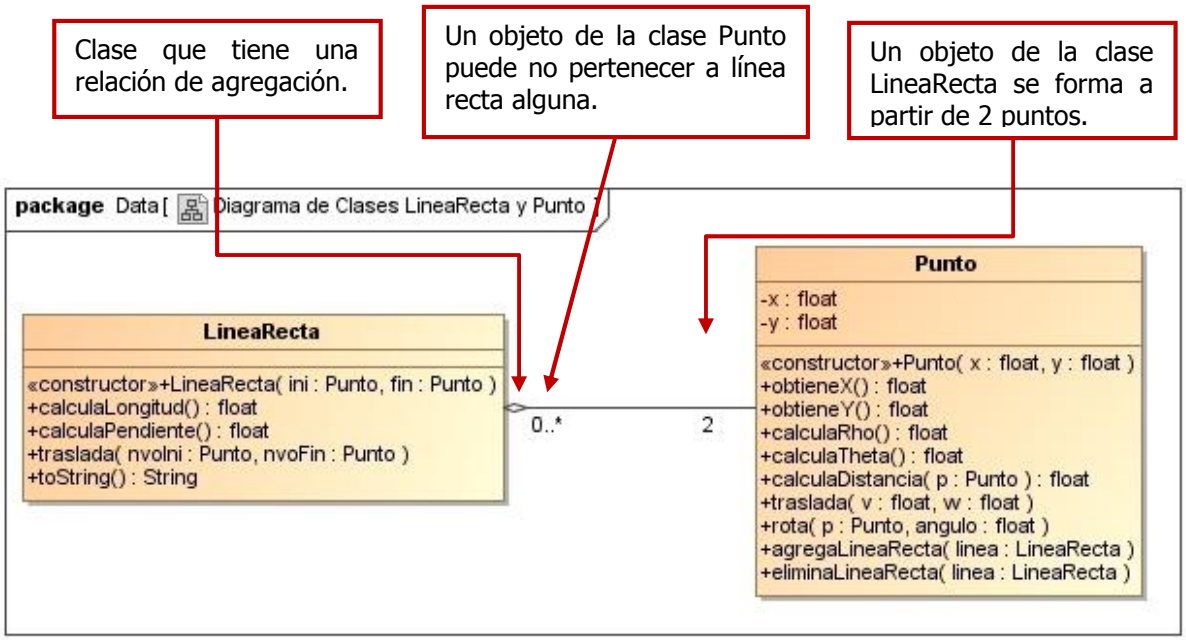

Figura 8. Relación de agregación entre LineaRecta y Punto.

Un objeto puede ser parte de la definición de más de un objeto de otra clase. Se aclarará esto mediante el ejemplo de la Figura 8, allí se observa que los objeto de tipo Punto pueden formar parte de ninguna o varias líneas, ello queda de manifiesto porque la multiplicidad en el lado de la clase LineaRecta es 0..*.

Por lo demás, los objetos que son parte de otros tienen un ciclo de vida independiente del ciclo de vida del o los objetos en que aparecen agregados. En el ejemplo de la Figura 8, se refleja en que si una cierta línea desaparece, ello no implica que desaparezcan también los puntos que la conforman.

2.2.3. Composición

Una *composición* es una relación semejante a la agregación, pero más estricta. En la Figura 9 se presenta un ejemplo que señala que los objetos de la clase Carrera se componen de uno o más objetos de la clase Asignatura, y que una Asignatura pertenece a una y sólo a una Carrera.

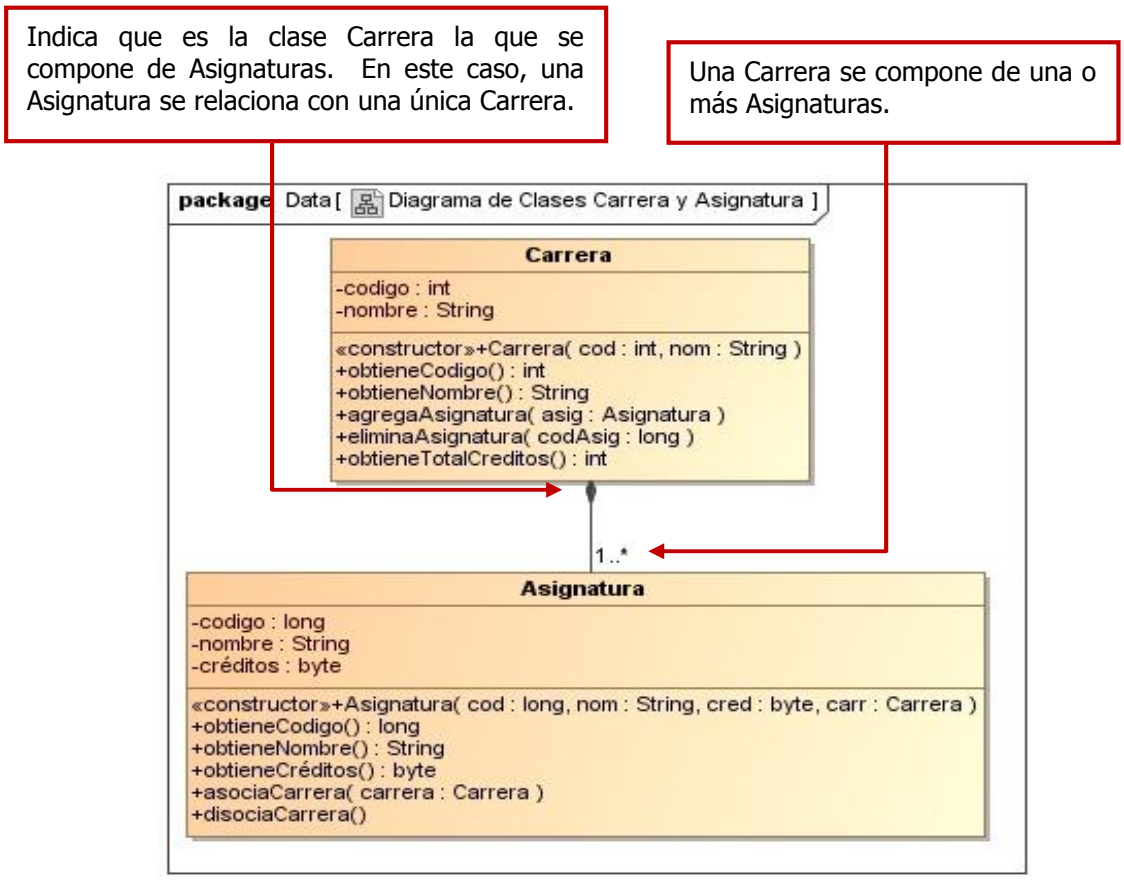

Figura 9. Relación de composición entre las clases Carrera y Asignatura.

En el ejemplo anterior, si una carrera desaparece (se destruye el objeto respectivo), las asignaturas que componían dicha asignatura (los objetos respectivos) también desaparecen. De hecho, una cierta asignatura se creará sólo para pertenecer a una cierta carrera.

2.2.4. Direccionalidad de las relaciones

En el ejemplo de la Figura 8 la relación de agregación entre LineaRecta y Punto es bidireccional, es decir, es posible, desde un objeto obtener los objetos relacionados con éste, y ello en ambos sentidos. Ahora, si el problema a resolver sólo requiere obtener los puntos que forman parte de una recta, y no las rectas en que participa un cierto punto, entonces sería aconsejable indicar tal consideración definiendo una relación unidireccional. En este caso el diagrama de clases UML correspondiente sería el que muestra la Figura 10.

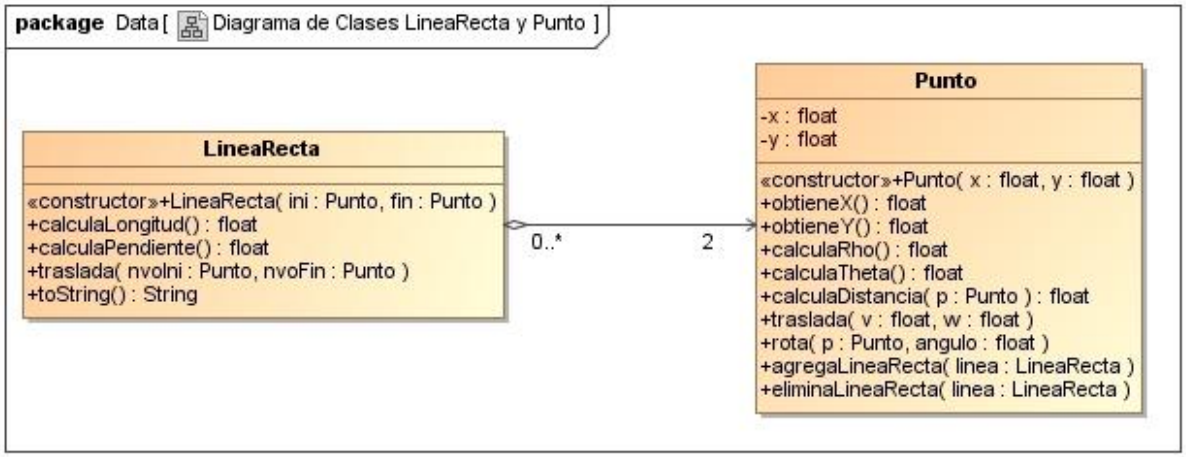

Figura 10. Ejemplo de una relación unidireccional entre LineaRecta y Punto.

Al definir que la relación entre estas dos clases es unidireccional en la forma en que aparece en el diagrama, se está indicando que no se desea, dado un punto, obtener las líneas rectas a las que pertenece.

La importancia de definir direccionalidad en las relaciones entre clases (objetos) es la capacidad de especificar claramente lo que requiere el problema a resolver. Las relaciones unidireccionales demandan menos esfuerzo de desarrollo y menos recursos.

2.2.5. Herencia

La herencia relaciona dos objetos indicando que uno se deriva o es una extensión del otro, de tal forma que uno se puede entender como una generalización y el otro como una especialización de un concepto común. El verbo asociado a esta relación es: "**es un**" o "**es una**". La Figura 11 muestra un ejemplo.

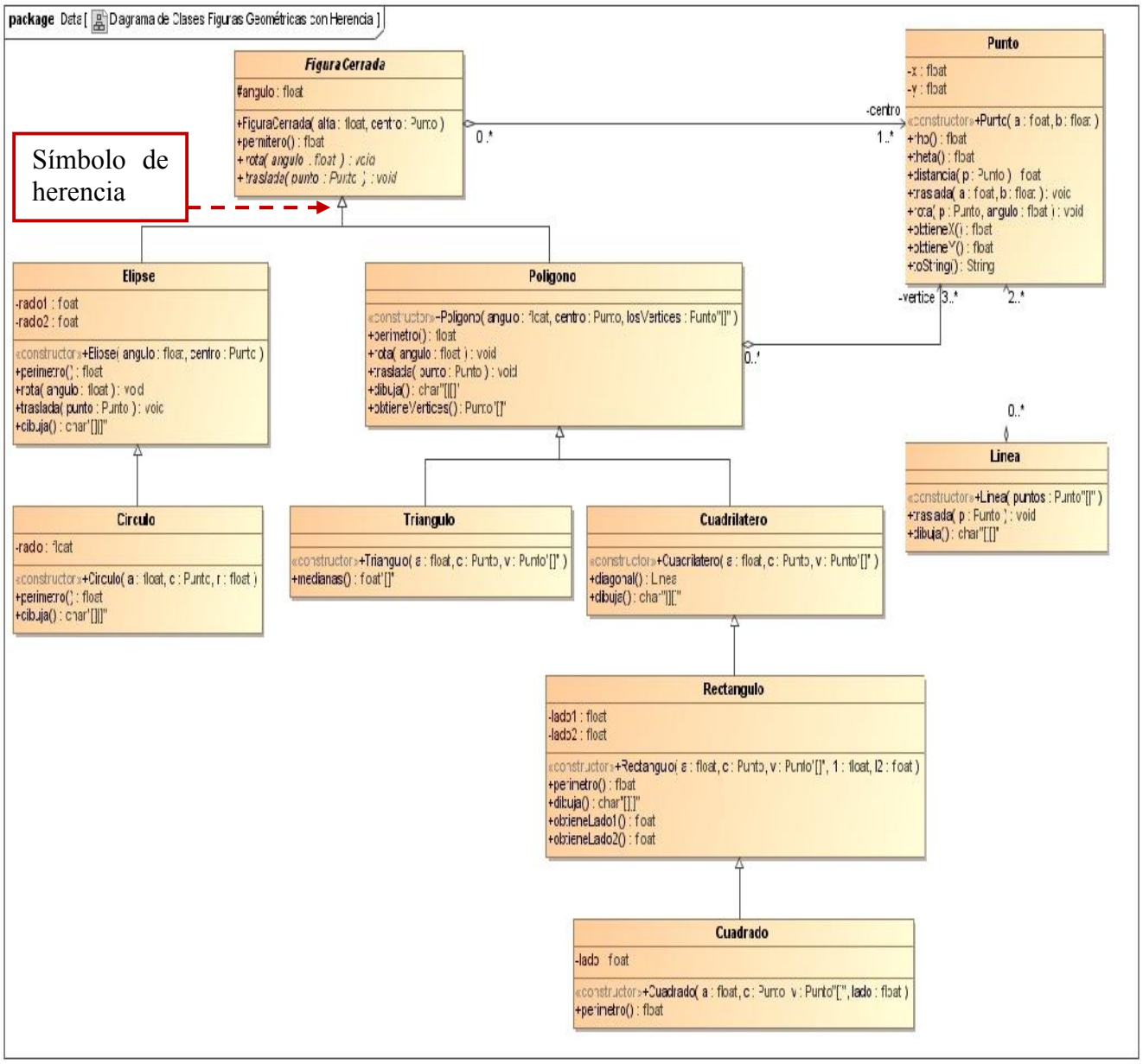

Figura 11. Relación de herencia entre clases.

En este caso, las clases Elipse y Poligono heredan atributos y métodos de la clase FiguraCerrada. Por otro lado, las clases Cuadrilatero y Triangulo heredan de la clase Poligono (y a través de esta clase heredan desde la clase FiguraCerrada). La clase Cuadrado hereda de Rectangulo y ésta de Cuadrilatero.

La clase FiguraCerrada tiene o se define a partir de un punto que constituye el centro de la figura, en el plano XY. Asimismo Poligono está conformado por un conjunto de vértices (al menos 3), que no son otra cosa que puntos en el plano XY. En este caso puede observarse que Poligono no es lo mismo que FiguraCerrada, de hecho cuando una clase hereda de otra, la primera debe agregar alguna nueva característica a la definición de la segunda (redefinir una operación, agregar atributos u operaciones).

En cambio, una Linea "no es una" FiguraCerrada, pero sí se compone de 2 o más puntos.

Veamos un ejemplo más simple de herencia, tomando como base la clase Libro presentada antes. Supongamos que se desea construir una nueva clase más general que permita considerar distintos tipos de materiales bibliográficos, como videos por ejemplo (a este proceso de definir clases más generales a partir de clases más especializadas se le llama generalización).

Como todo material bibliográfico debe tener un título y un año de publicación o edición asociado, se podría crear una clase MaterialBibliografico que contemplara estos atributos. Luego se podría contar con una clase Libro y una clase Video que heredaran dichos atributos y agregaran sus propios y específicos atributos. En la Figura 12 se presenta un diagrama que representa la relación de herencia en UML entre las clases.

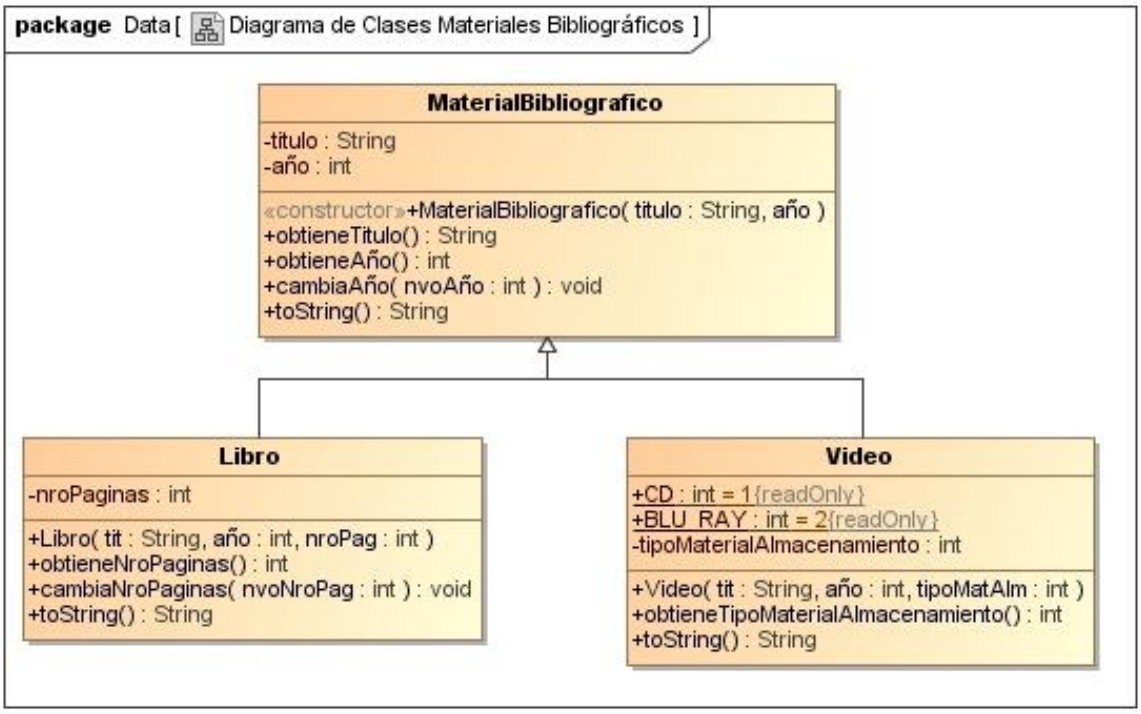

Figura 12. Relación de herencia entre MaterialBibliografico, Libro y Video.

2.2.6. Clases Abstractas

Se puede observar en la Figura 11 que la clase FiguraCerrada contempla 3 métodos (entre ellos perímetro), pero dichos métodos no debieran definirse en esta clase, sino en sus clases descendientes, pues cada uno depende de la figura cerrada particular sobre la que el método debe actuar. Si dichos métodos no pueden definirse en la clase FiguraCerrada, ¿por qué se declaran como métodos de dicha clase? Bueno, ello permite:

- Dejar claro que estos métodos son propios de todas las clases herederas de FiguraCerrada.

- Crear objetos de la clase FiguraCerrada en un programa que, durante su ejecución, puedan especializarse en alguna figura cualquiera e invocar la definición adecuada de uno de los métodos.

La lógica indica que no deben existir objetos de la clase FiguraCerrada, puesto que esta clase es sólo una noción conceptual y no una realidad geométrica, distinto es el caso de las clases Elipse y Circulo, entre otras, que representan figuras geométricas concretas y, por lo tanto, se debiera permitir la existencia de objetos de las mismas.

Las clases como FiguraCerrada, que no contemplan la existencia de objetos, se denominan abstractas. Ellas aparecen al crear generalizaciones de otras clases, es decir, se derivan a partir de relaciones de herencia.

2.2.7. Polimorfismo

Debido a la herencia, si un objeto es una instancia de una cierta subclase, entonces el objeto también es una instancia de la superclase correspondiente. Por ejemplo, en la Figura 11, si se tiene un objeto **c** de la clase Cuadrado, **c** también es un Cuadrilatero, un Poligono y una FiguraCerrada.

Sin embargo, se debe tener claro que un objeto que es una instancia de una superclase no implica que sea un objeto de una subclase particular de esa superclase. Para el ejemplo de las figuras, si un objeto **p** es una instancia de la clase Poligono, no implica que dicho objeto sea un Cuadrado, de hecho podría ser una instancia de Triangulo.

La propiedad señalada en el primer párrafo de esta sección permite tener objetos de distintas subclases en una colección que se define del tipo de la superclase común y, de este modo, invocar, **durante ejecución**, los métodos correctos dependiendo del tipo concreto del objeto. Esto se conoce como polimorfismo.

El polimorfismo resulta posible de implementar en los lenguajes de programación como Java cuando estos cuentan con una característica conocida como binding dinámico, es decir cuando son capaces de asignarles tipos (o clases) concretos a las variables (que referencian objetos que se han creado) de manera tardía, no durante compilación, sino durante ejecución.

Ejemplo:

Se podría definir una colección de rectángulos, cuadrados y triángulos y luego crear objetos de las distintas clases e incorporarlos como elementos de la colección.

Lo anterior permitiría invocar el método perimetro, sin tener preocupación de la clase a la que pertenece el objeto al cual se le está invocando dicho método, sabiendo que este se encuentra definido para todo objeto de la clase Poligono y, por lo tanto, de sus subclases.

Finalmente, el diagrama de clases de la Figura 13 presenta los diversos tipos de relación existentes entre un conjunto de clases derivadas de un mismo problema.

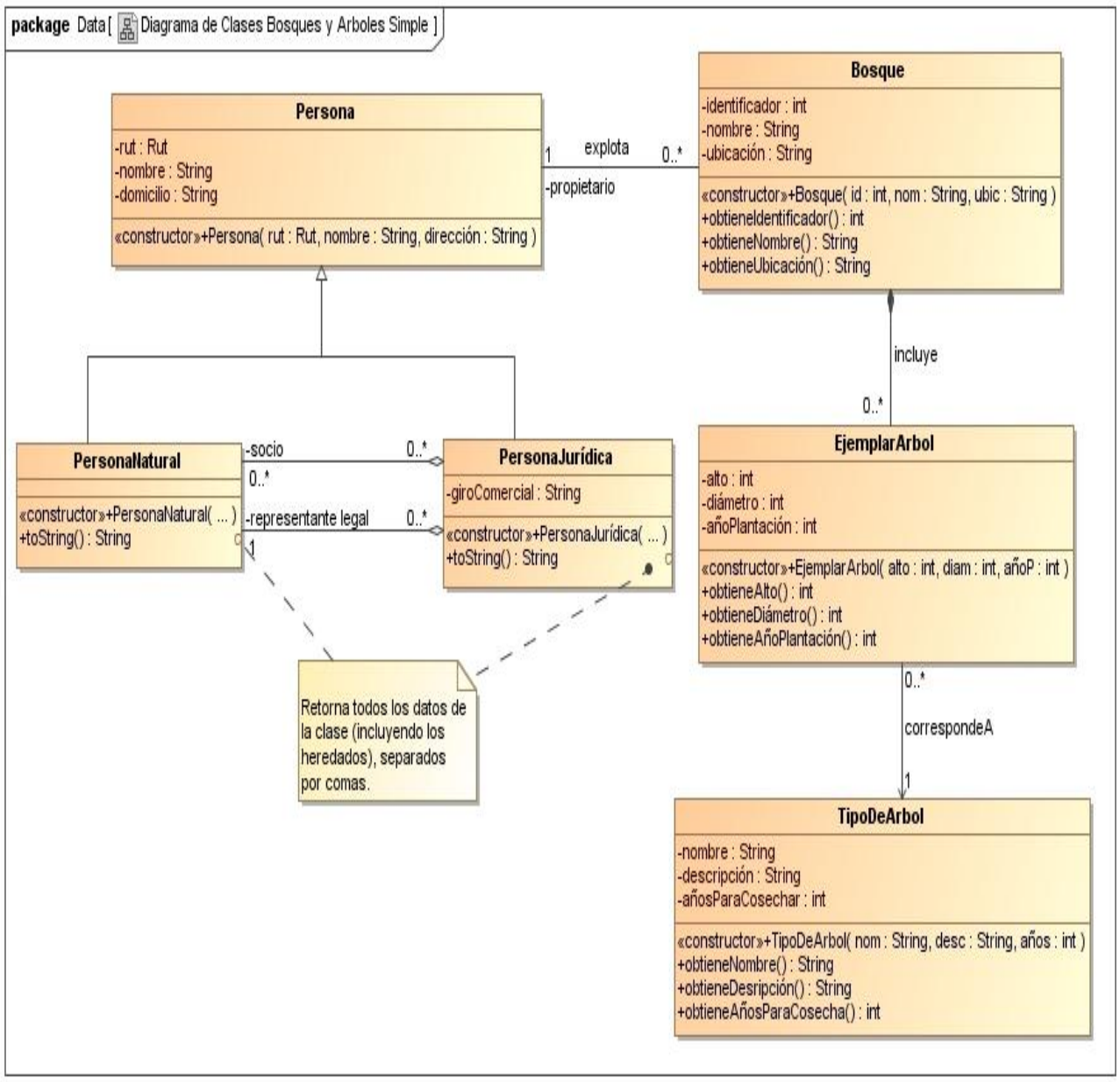

Figura 13. Diversas relaciones entre clases.