Diagramas UML de Clases

Introducción

El *Lenguaje de Modelación Unificado (Unified Modeling Language* o *UML* por sus siglas en inglés) es un lenguaje gráfico para modelar los aspectos estáticos y dinámicos de un sistema. UML es una unificación de las metodologías desarrolladas por Grady Booch, James Rumbaugh e Ivar Jacobsen. El *Grupo de Administración de Objetos (Object Management Group* u *OMG*), una organización sin fines de lucro que promueve el uso de la tecnología orientada a objetos, ha asumido la responsabilidad del futuro desarrollo del estándar UML.

UML es un lenguaje que unifica las mejores prácticas de la ingeniería para modelar sistemas. Define diferentes diagramas gráficos que proveen múltiples perspectivas de un sistema en desarrollo. UML define los siguientes diagramas:

- Diagrama de caso de uso
- Diagrama de clase
- Diagramas de comportamiento:
  - Diagrama de estado
  - Diagrama de actividad
  - Diagramas de interacción:
    - Diagrama de secuencia
    - Diagrama de colaboración
- Diagramas de implementación:
  - Diagrama de componentes
  - Diagrama de despliegue

A lo largo de este curso, utilizaremos *diagramas de clase* para discutir los diferentes elementos del diseño orientado a objetos. Los diagramas de clase muestran la estructura interna de las clases y sus relaciones con otros elementos del sistema.

Notación de Clase

El elemento fundamental de un diagrama de clase es la notación de una clase. Como se muestra en la Figura 1, una clase está representada por un rectángulo con tres secciones. La primera sección contiene el nombre de la clase; la segunda sección describe los atributos de la clase y la tercera sección describe los métodos de la clase.

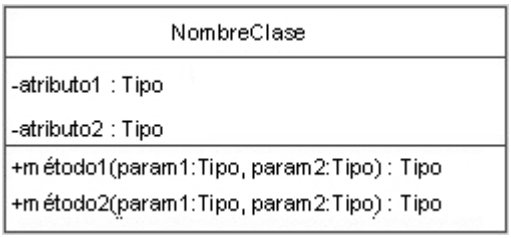

Figura 1 *Notación de Clase*

El tipo de una variable y el tipo de retorno de un método están especificados por dos puntos (:) seguidos por el nombre de un tipo. La especificación de un atributo o método comienza con un símbolo que indica la visibilidad de dicho atributo o método:

- \- indica visibilidad privada
- \+ indica visibilidad pública

La *visibilidad privada* no permite que otras clases tengan acceso a este atributo o método, mientras que la *visibilidad pública* permite que otras clases tengan acceso a este atributo o método.

Comúnmente, los atributos se definen como privados; de esta manera el objeto puede esconder su información para mantener el control de su estado. Un método se define como público si se espera que sea llamado por otras clases y se define como privado si no se pretende que sea llamado por clases externas.

Cuando una clase necesita brindar acceso a sus atributos privados, la convención indica definir los métodos públicos para acceder el valor de un atributo o modificar un atributo. Estos métodos son conocidos como *selectores* (*accessors*) o *modificadores (mutators)*, respectivamente.

A continuación se muestra la representación UML de la clase `Employee` (*empleado*). Esta clase contiene tres atributos privados: `name` (*nombre*),`hourlyWage` (*salario por hora*) y `hoursWorked` (*horas trabajadas*). También contiene los siguientes métodos públicos:

- Métodos para obtener los valores de los atributos (*accessors*): `getName`, `getHourlyWage`, y `getHoursWorked`
- Métodos para modificar los valores de los atributos (*mutators*): `setName`, `setHourlyWage`, y `setHoursWorked`
- Un método para obtener los ingresos del empleado: `getEarnings`

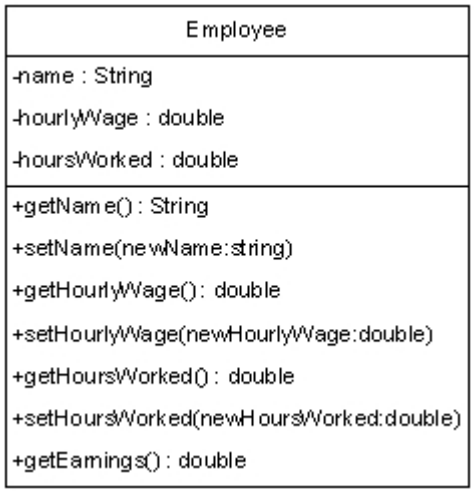

Figura 2 *Representación de la clase Employee*

La notación de clase puede ser abreviada para resaltar otras partes del sistema que se está modelando. Por ejemplo, la sección de métodos únicamente debe especificar los nombres de los métodos:

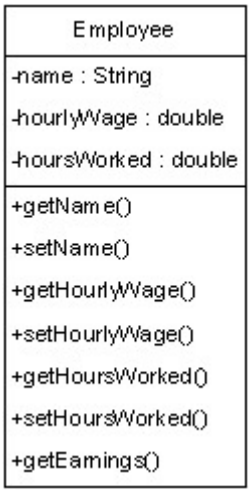

Figura 3 *Representación abreviada de la clase Employee*

La forma más pequeña de una clase consiste en un rectángulo con un nombre de clase:

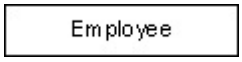

Figura 4 *La forma más pequeña de la clase Employee*

Un diagrama de clase contiene muchas clases, atributos y métodos que pueden ser difíciles de leer. En tal caso, pueden omitirse algunos atributos y métodos de forma segura. El siguiente diagrama de clase incluye solamente los atributos y métodos que son más importantes para describir la funcionalidad de la clase:

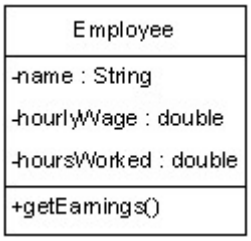

Figura 5 *Representación esencial de la clase Employee*

Relaciones Entre Clases

- Asociaciones
- Asociaciones de Una Vía (One-Way) y Dos Vías (Two-Way)
- Multiplicidad
- Agregación
- Especialización/Generalización

Asociaciones

Una *asociación* representa la relación entre dos o más clases. Una *asociación binaria* es una relación entre dos clases. Existe una asociación binaria si un objeto de una clase requiere un objeto de otra clase para hacer su trabajo. En UML, una asociación binaria está representada por una línea sólida que conecta dos clases.

Por ejemplo, en un salón de clases, un profesor (`Professor`) enseña a los estudiantes (`Student`). Esta asociación está expresada en el siguiente diagrama de clase:

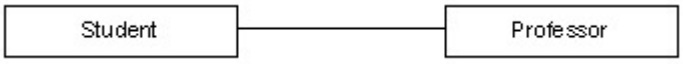

Figura 1 *Asociación entre las clases que representan estudiantes y profesores*

En un sistema bancario, los clientes (`Client`) tienen cuentas de banco (`BankAccount`). Esta asociación se encuentra expresada en el siguiente diagrama de clase:

Figura 2 *Asociación entre las clases que representan clientes y cuentas bancarias*

Asociaciones de Una Vía (One-Way) y Dos Vías (Two-Way)

Una asociación puede ser una relación de una vía (*one-way*) o de dos vías (*two-way*). Una *asociación de una vía* indica la dirección en la que se puede navegar de un objeto de una clase a un objeto de otra clase. Una *asociación de dos vías* indica una navegación bidireccional entre objetos de dos clases.

En una asociación de una vía, la primera clase tiene una referencia a un objeto de la segunda clase, pero la segunda clase no tiene una referencia a un objeto de la primera clase. En una asociación de dos vías, cada clase contiene una referencia a un objeto de la otra clase.

UML indica una asociación de una vía con una flecha al final de la línea de asociación. El atributo de la primera clase que contiene una referencia a un objeto de la segunda clase también está escrito al final de la línea.

Por ejemplo, la siguiente asociación indica que un motor (`Engine`) es parte de un carro (`Car`):

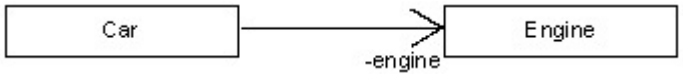

Figura 3 *Asociación de una vía entre las clases que representan carros y motores*

Observa el atributo `engine` al final de la línea. La clase `Car` tiene un atributo `engine` que contiene una referencia a un objeto de la clase `Engine`.

El siguiente ejemplo muestra la relación entre las clases `Country` (*país*), `Government` (*gobierno*) y `Capital`. Cada país tiene un gobierno y una capital. Los atributos al final de las líneas indican que la clase `Country` tiene una referencia a un objeto de la clase `Government` así como una referencia a un objeto de la clase `Capital`:

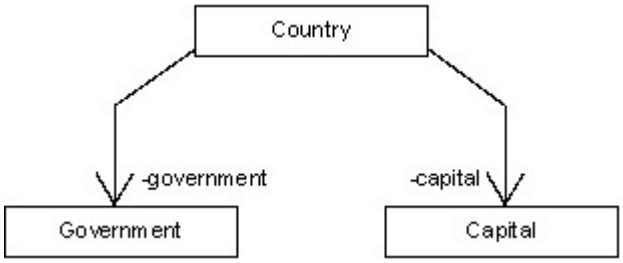

Figura 4 *Asociaciones de una vía entre las clases que representan países, gobiernos y capitales*

Una clase puede contener una o más asociaciones con otra clase. Por ejemplo, el siguiente diagrama de clase muestra dos asociaciones entre las clases `Flight` (*vuelo*) y `Pilot` (*piloto*), una asociación con el atributo `pilot` (*piloto*) y otra con el atributo `coPilot` (*copiloto*):

Figura 5 *Dos asociaciones entre las clases que representan vuelos y pilotos*

Multiplicidad

La *multiplicidad* indica el número de instancias de una clase que pueden ser asociadas a una sola instancia de otra clase. Por ejemplo, un carro tiene cuatro llantas y un motor, también un cliente tiene una o más cuentas de banco. La multiplicidad puede especificarse con un solo entero o como un rango *n..m*, donde *n* es el límite inferior y *m* es el límite superior. Podemos utilizar un asterisco (*) para denotar que no existe un límite superior. A continuación se presentan las multiplicidades más comunes:

0..1	Cero o una instancia
*0..** ó *	Cero o más instancias
1	Exactamente una instancia
1..*	Una o más instancias

Tabla 1 *Multiplicidades comunes*

Las asociaciones pueden clasificarse de acuerdo a su multiplicidad. En este curso, discutiremos tres tipos: uno a uno, uno a muchos y muchos a muchos.

Asociación uno a uno

En una *asociación uno a uno*, exactamente una instancia de cada clase está relacionada solamente con una instancia de otra clase. Por ejemplo, la asociación entre las clases `Car` (*carro*) y `Engine` (*motor*) puede definirse como una asociación uno a uno, indicando que cada carro contiene solo un motor y que cada motor está instalado solamente en un carro:

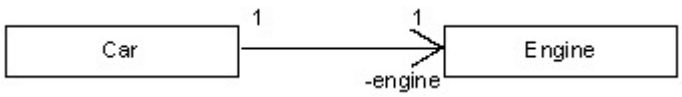

Figura 6 *Asociación uno a uno entre las clases que representan carros y motores*

El siguiente diagrama de clase muestra las asociaciones uno a uno entre las clases `Country` (*país*), `Government` (*gobierno*) y `Capital`. Cada país contiene sólo un gobierno y una capital. Cada gobierno está asociado con un solo país y cada capital está asociada con solo un país:

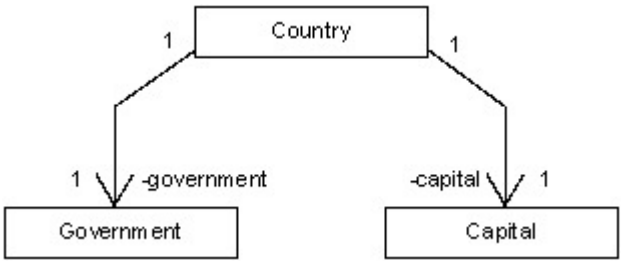

Figura 7 *Asociaciones uno a uno entre las clases que representan países, gobiernos y capitales*

Asociación uno a muchos

En una *asociación uno a muchos* entre las clases `A` y `B`, una instancia de la clase `A` puede estar relacionada con muchas instancias de la clase `B`, pero una instancia de la clase `B` está relacionada solamente con una instancia de la clase `A`. Por ejemplo, en la asociación entre las clases `Client` (*cliente*) y`BankAccount` (*cuenta bancaria*), cada cliente puede tener muchas cuentas de banco, pero cada cuenta de banco tiene solamente un propietario:

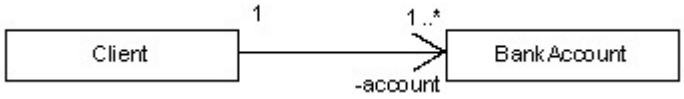

Figura 8 *Asociación uno a muchos entre las clases que representan clientes y cuentas bancarias*

El siguiente diagrama muestra la relación entre las clases que representan un calendario:

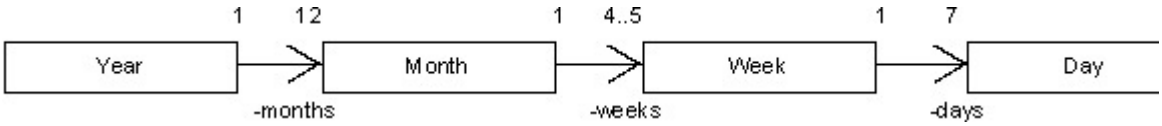

Figura 9 *Asociación entre las clases de un calendario*

En este diagrama, un año (`Year`) contiene doce meses (`Month`) y cada mes está asociado con solo un año. (Toma en cuenta que los meses*Enero1962* y *Enero2029* son instancias diferentes de la clase `Month`.) De la misma forma, un mes contiene cuatro o cinco semanas (`Week`) y cada semana está asociada con solo un mes. Finalmente, una semana contiene siete días (`Day`) y cada día está asociado con una sola semana.

Asociación muchos a muchos

En una *asociación muchos a muchos* entre las clases `A` y `B`, una instancia de la clase `A` puede estar relacionada con muchas instancias de la clase `B` y una instancia de la clase `B` puede estar relacionada con muchas instancias de la clase `A`. Por ejemplo, en la asociación entre las clases `Student`(*estudiante*) y `Course` (*curso*), cada estudiante toma muchos cursos y cada curso está siendo impartido para muchos estudiantes:

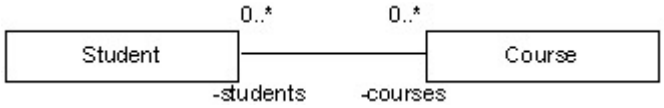

Figura 10 *Asociación entre las clases que representan estudiantes y cursos*

El siguiente diagrama representa una compañía donde un empleado (`Employee`) puede trabajar en muchos departamentos (`Department`) y puede estar involucrado en muchos proyectos (`Project`); cada departamento puede tener muchos empleados y a su vez, cada proyecto puede involucrar muchos empleados:

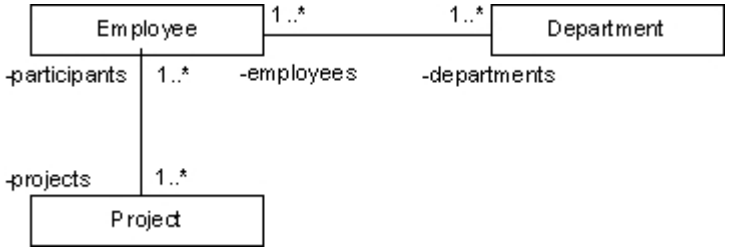

Figura 11 *Asociación entre las clases que representan empleados, departamentos y proyectos*

Agregación

La *agregación* es una forma especial de asociación. Una agregación es una asociación entre las clases `A` y `B`, donde cada instancia de la clase `A`contiene, o está compuesta por, instancias de la clase `B`. En este sentido, una instancia de la clase `B` es parte de una instancia de la clase `A`. A la instancia de la clase `A` se le conoce como *agregada (aggregate)* y a la instancia de la clase `B` se le conoce como *componente (component)*. Por ejemplo, un libro (`Book`) está compuesto por una tabla de contenidos (`TableOfContents`), ningún o un prefacio (`Preface`), uno o más capítulos (`Chapter`) y un índice (`Index`). En un libro, cada capítulo está compuesto por una o más secciones (`Section`) y cada sección está compuesta por uno o más párrafos (`Paragraph`) y ninguna o algunas figuras (`Figure`).

Una relación de agregación se representa en UML mediante una línea con un diamante junto a la clase agregada:

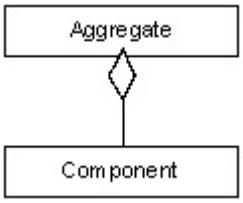

Figura 12 *Notación de agregación*

El siguiente diagrama de clase muestra la estructura de componentes de un libro:

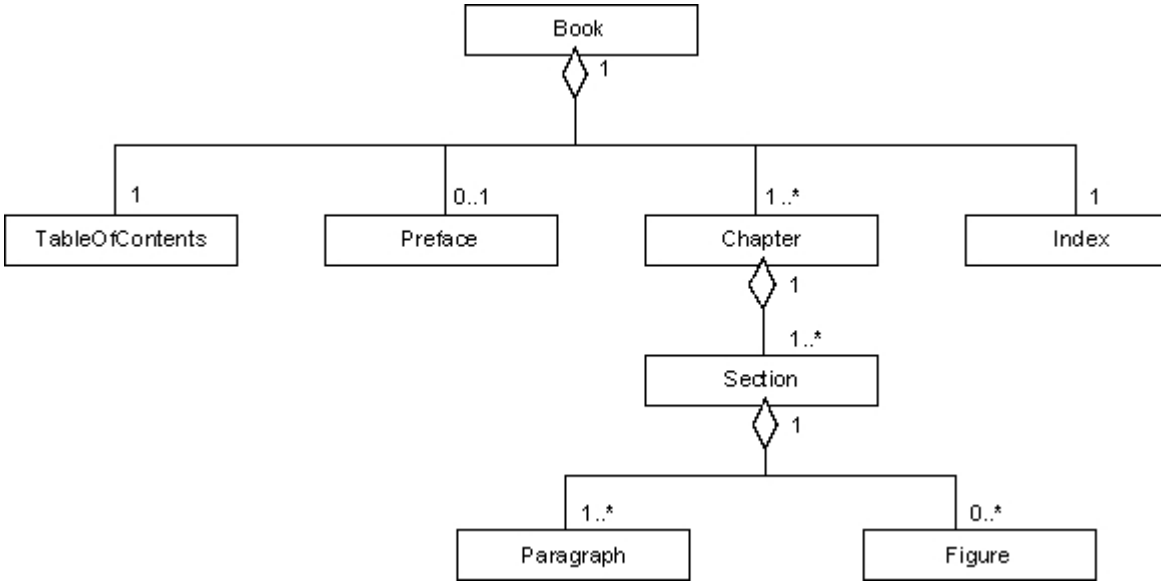

Figura 13 *Agregación jerárquica de un libro*

En este curso, no utilizaremos asociaciones de agregación. Los ejemplos y ejercicios utilizarán únicamente asociaciones de una vía y de dos vías.

Especialización/Generalización

La *Especialización/Generalización* representa a la relación *es un*. Por ejemplo, una ballena *es un* mamífero y un cliente *es una* persona. La especialización/generalización permite que la clase `A` sea definida como especialización de otra clase `B`, más general. La clase `A` es llamada *clase de especialización (specialization class)* y la clase `B` es llamada *clase de generalización (generalization class)*. La especialización/generalización se representa en UML mediante una línea con un triángulo junto a la clase de generalización:

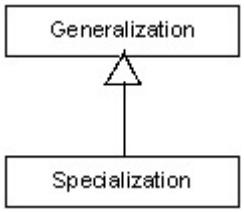

Figura 14 *Notación de Especialización/Generalización*

Si existe una relación de especialización/generalización entre las clases `A` y `B`, es decir, si `A` *es un* `B`, entonces todas las instancias de la clase `A` son también instancias de la clase `B`. Una consecuencia importante de esta relación es que la clase `A` *hereda* todas las características de la clase `B`. Esto significa que todos los atributos y métodos de la

clase `B` son también atributos y métodos de la clase `A`. El siguiente diagrama de clase representa las relaciones *cliente es una persona* (`Client` *is a* `Person`) y un *empleado es una persona* (`Employee` *is a* `Person`):

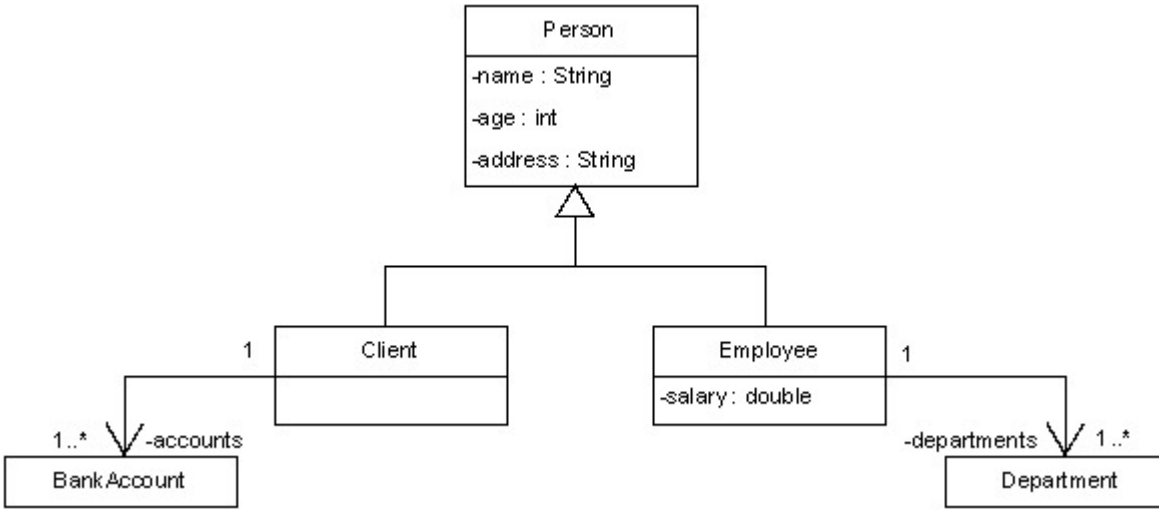

Figura 15 *Relación de especialización/generalización entre clases que representan clientes, empleados y personas*

Debido a las relaciones de especialización/generalización, los atributos de la clase `Person` son también atributos de las clases `Client` y `Employee`. Cada instancia de la clase `Client` contiene los atributos `name` (*nombre*), `age` (*edad*), `address` (*dirección*) y `accounts` (*cuentas*); y cada instancia de la clase `Employee` contiene los atributos `name` (*nombre*), `age` (*edad*), `address` (*dirección*), `salary` (*salario*) y `departament` (*departamento*).

Veamos otro ejemplo. Considera un sistema de archivos multimedia que incluye archivos de imagen, de audio y de texto. Las clases se definen a continuación:

- La clase `MediaFile` (*archivo de multimedia*) contiene información general sobre los archivos multimedia: `name` (*nombre*), `extension`(*extensión*) y `size` (*tamaño*) en bytes.
- La clase `ImageFile` (*archivo de imagen*) especializa la clase `MediaFile`, añadiendo los atributos `width` (*ancho de la imagen*) y `height`(*altura de la imagen*).
- La clase `AudioFile` (*archivo de audio*) especializa la clase `MediaFile`, añadiendo el atributo`length` *(longitud en minutos*).
- La clase `TextFile` especializa la clase `MediaFile`, añadiendo el atributo `summary` (*sumario*) que contiene todo el texto.
- Las clases `MidiFile` y `Mp3File` especializan la clase `AudioFile`, añadiendo información específica a cada formato.
- Las clases `JpgFile` y `GifFile` especializan la clase `ImageFile`, añadiendo información específica a cada formato.

El siguiente diagrama de clase muestra las relaciones de especialización/generalización entre las clases:

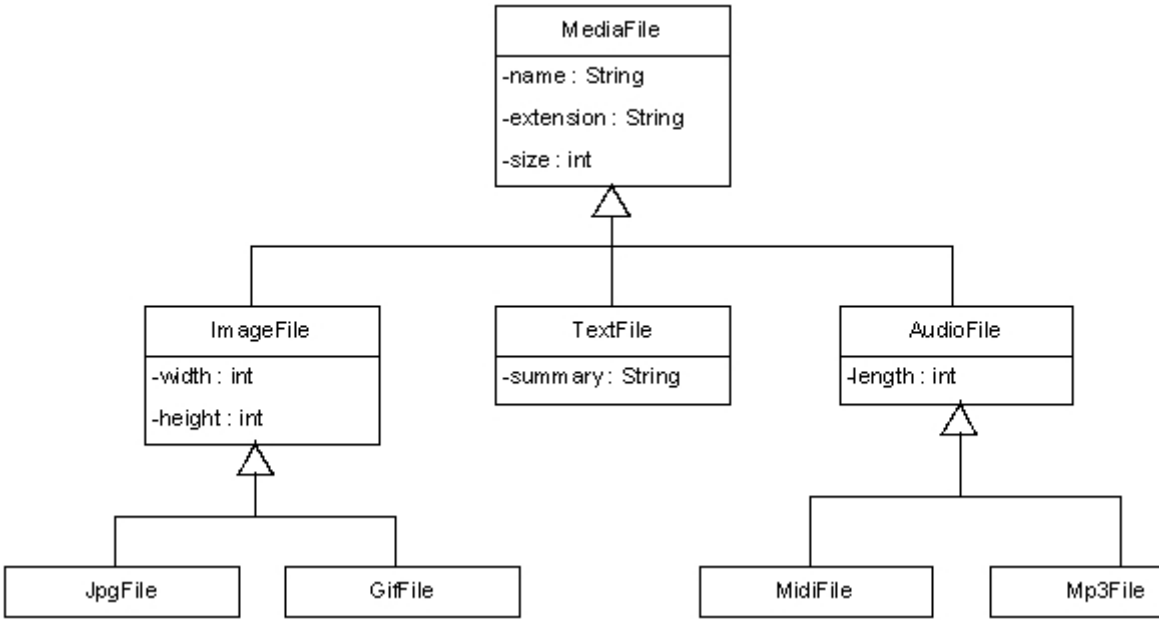

Figura 16 *Jerarquía de especialización/generalización de los archivos multimedia*

En este ejemplo:

- Una instancia de la clase `JpgFile` contiene los atributos específicos de la clase `JpgFile`, así como los atributos `width`, `height`, `name`,`extension` y `size`.
- Una instancia de la clase `TextFile` contiene los atributos `summary`, `name`, `extension` y `size`.
- Una instancia de la clase `Mp3File` contiene los atributos específicos de la clase `Mp3File`, así como los atributos `length`, `name`, `extension` y`size`.