ADSI

Conceptos de POO

Instructor: Gustavo Adolfo Rodríguez Q.

garodriguez335@misena.edu.co

ADSI

1. CONCEPTOS DE PROGRAMACION ORIENTADA A OBJETOS

Anteriormente hemos hablado de las bases principales de la programación orientada a objetos, discutimos sobre conceptos tales como objetos, clases, tipos de acceso, mensajes y el concepto de herencia.

El paradigma de programación orientada a objetos también incorpora otros conceptos tales como:

- Relaciones entre clases
  - Relaciones de Generalización
  - Relaciones de Agregación
  - Relaciones de Asociación

- Propiedades de los objetos

  - Encapsulación
  - Modularidad
  - Polimorfismo

- Ciclo de vida de los objetos

- Constructor de Objetos.

1.1. Relaciones entre clases

Las relaciones entre clases juegan un papel importante en el paradigma de programación orientada a objetos. Las clases, al igual que los objetos, no existen de modo aislado, sino que poseen relaciones entre clases y entre objetos.

Los tres grandes tipos de relaciones entre clases son:

- **Generalización** (*es un*)
- **Agregación** (*todo/parte*)
- **Asociación**

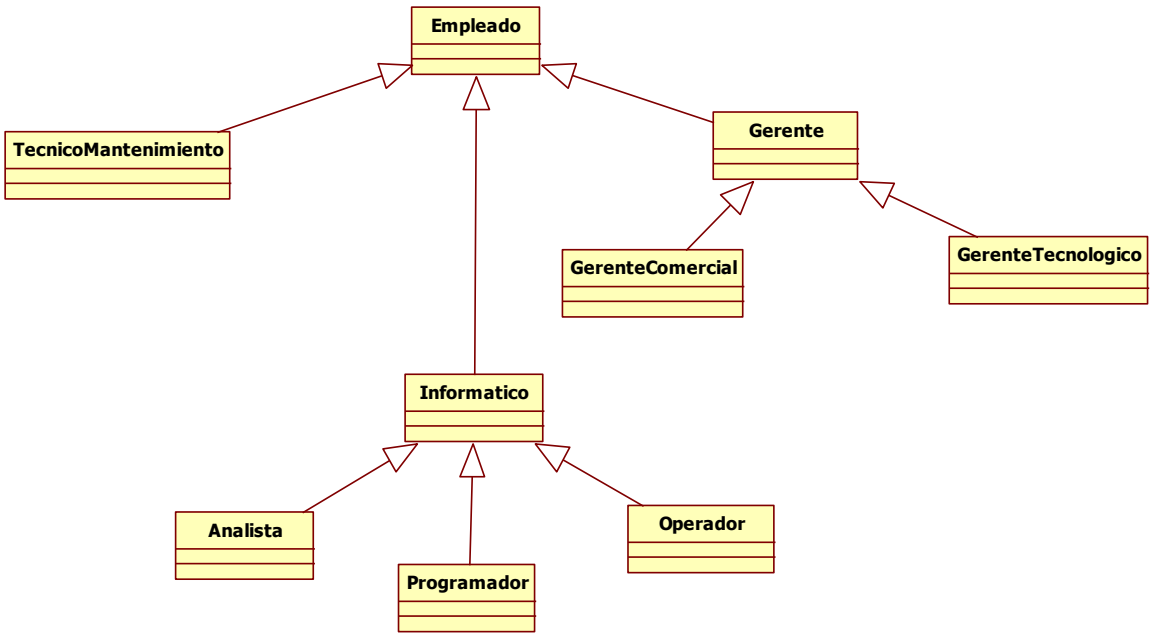
1.1.1. Generalización

La generalización representa las relaciones *"es un tipo de"*. Una rosa **es un tipo de** flor, un triángulo **es un tipo de** figura geométrica. Esto significa que la clase Rosa en una subclase o clase hija de una clase más general llamada Flor. Así mismo, significa que una clase llamada Triángulo es una subclase de una clase más general llamada Figura.

La relación de generalización y el concepto de herencia están fuertemente ligados. Decir que la clase Flor es una generalización de la clase Rosa, es similar a decir que la

clase Rosa hereda de la clase Flor y también equivalente a decir que la clase Rosa es una subclase de la clase Flor.

Existe la posibilidad de tener varios niveles de herencia, donde una clase hija se convierte a su vez en clase padre de una nueva clase. Esto da lugar a las jerarquías de clases. A continuación se muestra un diagrama de clases donde se puede ver la jerarquía de clases originada al modelar la organización interna de una empresa.



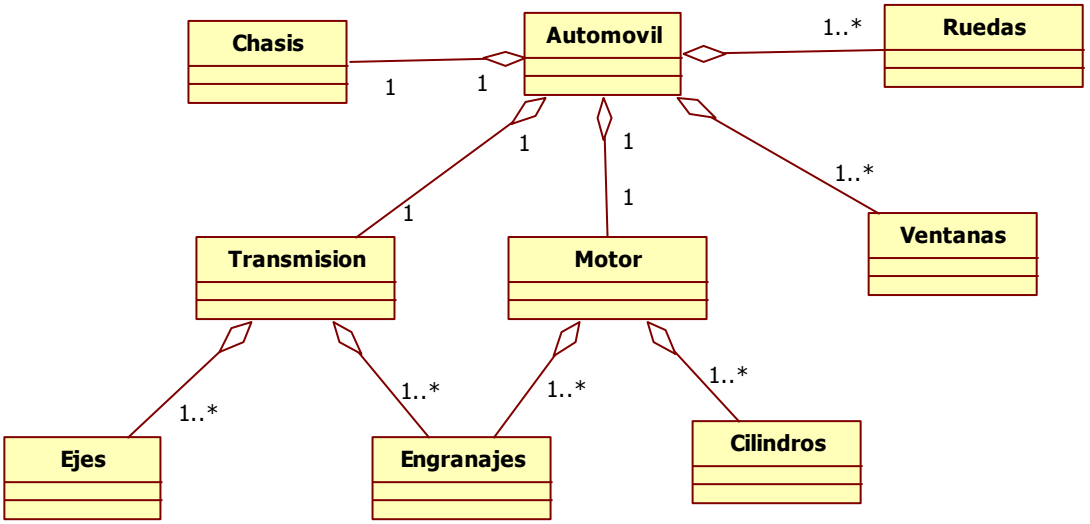
1.1.2. Agregación

Una agregación es una relación que representa a las clases compuestas. Una clase se considera compuesta si dentro de sus atributos posee objetos de otras clases. Por ejemplo, una clase Casa está compuesta por objetos de la clase Habitación, por objetos de la clase Ventana, etc.

La agregación de objetos permite describir modelos del mundo real que se componen de otros modelos, que a su vez se componen de otros modelos. La agregación es un concepto que se utiliza para expresar relaciones ***parte-de o tiene-un*** entre objetos.

Las relaciones de agregación generalmente incluyen el valor de multiplicidad de la agregación, estos valores pueden ser de uno a uno, de uno a muchos o de muchos a uno.

A continuación se presenta un diagrama de clases del modelado de un automóvil donde se evidencia la relación de agregación.



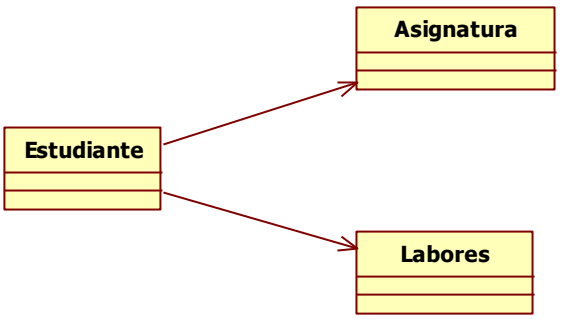
1.1.3. Asociación

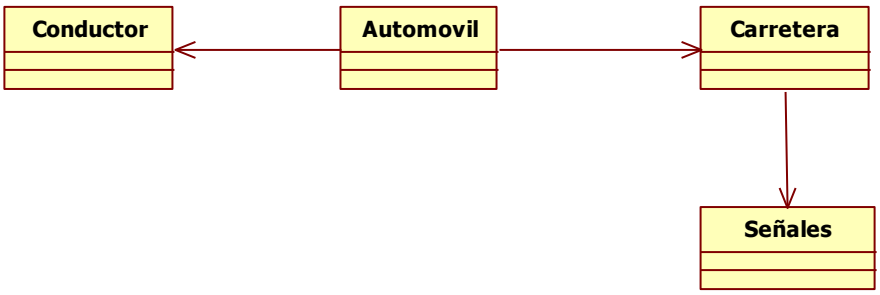
Una relación de asociación representa una dependencia semántica entre clases e implica la dirección de esta dependencia. En general, las asociaciones son bidireccionales, aunque pudieran ser unidireccionales si así se indican expresamente. Para poder identificar las relaciones de asociación es necesario identificar frases tales como ***es miembro de***, ***está asociado con*** *o* ***trabaja para.***

Por ejemplo, Jaime Caicedo trabaja para el SENA, aquí podemos identificar una relación de asociación.

Otro ejemplo puede ser los estudiantes de ADSI son miembros de la clase de programación orientada a objetos.

A continuación se muestran los diagramas de clases de algunos ejemplos de asociación de clases.





1.2. Propiedades de los objetos

1.2.1. Encapsulación

La encapsulación o encapsulamiento es la propiedad que le permite a un objeto asegurar que el contenido de su información está oculto al mundo exterior (otros objetos). El objeto A no conoce lo que hace el objeto B, y viceversa. La encapsulación o encapsulamiento es en esencia un proceso mediante el cual se ocultan los detalles de la abstracción de datos de un objeto.

El encapsulamiento permite la división de un programa en módulos, estos módulos se implementan mediante clases. Si utilizamos el encapsulamiento, podemos decidir cuáles acciones y propiedades de cada módulo será visibles para otros módulos.

1.2.2. Modularidad

La modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas denominadas módulos, cada una de estas partes debe ser tan independiente como sea posible de la aplicación e sí y de las restantes partes.

1.2.3. Polimorfismo

Esta propiedad no suele ser considerada como fundamental en el paradigma de programación orientada a objetos pero dada su importancia siempre es considerada como parte del paradigma.

Polimorfismo es la propiedad que indica, literalmente, la posibilidad de que una entidad tome *varias formas.* En términos prácticos, el polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas según sea el objeto que se referencie en un instante determinado.

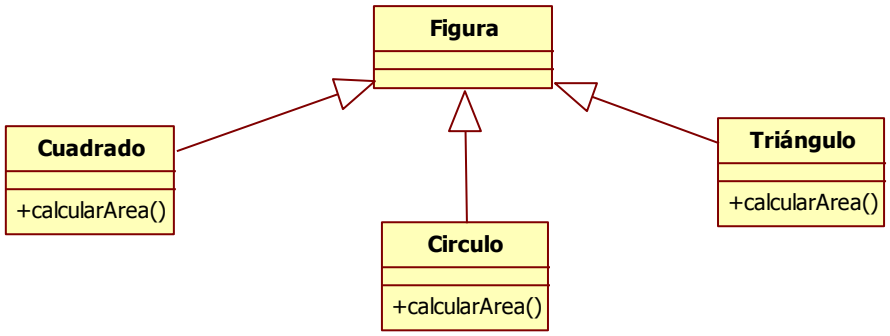
Por ejemplo, cuando se describe la clase *Mamifero* se puede observar que la operación *comer* es una operación fundamental en la vida de los mamíferos, de modo que cada tipo de mamífero debe poder realizar la operación o función comer. Por otra parte, una vaca o una cabra comen pasto, un niño se come un bombón o caramelo, y un león devora a otros animales, las anteriores son formas diferentes que utilizan los mamíferos para realizar la misma función *comer.*

Existen dos clases de polimorfismo en la programación orientada a objetos:

- **Polimorfismo de generalización:** Esta clase de polimorfismo se presenta cuando dos o más objetos de diferentes clases pero que tienen un padre común, responden de manera distinta al mismo mensaje.

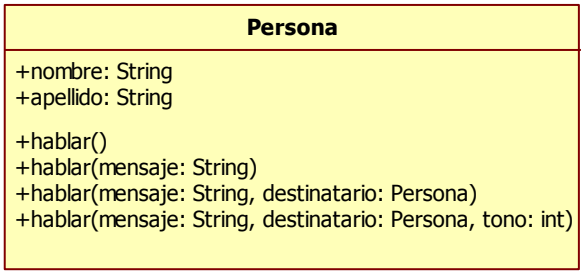
  Por ejemplo, supongamos que se tienen las clases Cuadrado, Triángulo y Circulo cada una de ellas desciende o hereda de la clase Figura. Las clases Cuadrado, Triángulo y Circulo poseen un método ***calcularArea()*** paro la forma de calcular el área es diferente entre ellas.

  A continuación se muestra el diagrama de clases donde se evidencia la sobrecarga de generalización.



- **Polimorfismo de sobrecarga:** Esta clase de polimorfismo se presenta cuando un objeto posee varias veces el mismo método pero cada uno de ellos se diferencia de los demás en los argumentos necesarios para su ejecución.

  Por ejemplo, Supongamos que se tiene la clase Persona, la clase define el método hablar con sobrecarga como se muestra en el siguiente diagrama de clase



1.3. Ciclo de vida de los Objetos

Los objetos tiene un ciclo de vida, este ciclo de vida está compuesto por los diferentes estados que alcanza un objeto a través de su existencia hasta que finalmente muere.

Los estados del ciclo de vida de un objeto son:

- **Declaración:** Este es el estado inicial, en este estado se *declara* la identidad del objeto, es decir, se especifica la clase a la cual va a pertenecer este objeto y el nombre que tendrá para posteriores referencias a el.

  Para la declaración de objetos se utiliza el siguiente esquema:
  tipo_de_accceso tipo_de_dato nombre_objeto.

  Por ejemplo, las siguientes son declaraciones válidas de objetos:
  - public String objetoTexto.
  - private Persona juanito.
  - public Figura miFiguraGeometrica.
  - protected Computador mipc.

- **Instanciación:** Este estado se alcanza después de que el objeto ha sido ***construido.*** Es decir cuando el objeto empieza a tener valores en sus atributos y ocupa en espacio real.

  En la instanciación el objeto entre a hacer parte de la memoria del dispositivo donde se ejecuta el programa y todo gracias a un método especial llamado ***método constructor*** del cual hablaremos más adelante.

  Por ejemplo:

  - objetoTexto = **new String().**
  - juanito = **new Persona()**.
  - miFiguraGeometrica = **new Figura()**.
  - mipc = **new Computador()**.

- **Utilización:** En este estado el objeto está listo para interactuar con otros objetos, recibiendo y pasando mensajes para lograr el comportamiento deseado. En el estado de utilización es donde ocurren los cambios de los valores de las propiedades de los objetos

- **Destrucción:** Este es el estado final del ciclo de vida y es donde el objeto es destruido y libera todos los recursos que haya consumido del dispositivo donde se ejecuta el programa.

1.4. Constructor de Objetos

Las clases definen un método particular que permite la construcción de objetos a partir de ellas. Este método particular se llama ***método constructor.*** Un constructor, como su nombre lo indica, *construye* nuevos objetos de una clase particular.

Una clase puede tener uno o más constructores, si tiene más de un método constructor estos están diferenciados entre sí mediante sus argumentos (polimorfismo de sobrecarga).

El método constructor de una clase debe tener la siguiente estructura:

tipo_de_accceso nombre_de_clase (argumentos1, argumento2, …, argumento n)

Por ejemplo, los siguientes podrían ser los métodos constructores para una clase Tablero:

- public Tablero()
- public Tablero(int ancho)
- public Tablero(int ancho, int alto)

El diagrama de clase para la clase Tablero se presenta a continuación.