RELACIONES DE PROGRAMACIÓN ORIENTADA A OBJETOS EN UML

Jesús Reyes Carvajal

Diagrama de Clases UML: Asociación

Definición

Es una relación de estructura entre clases, es decir, una entidad se construye a partir de otra u otras. Aunque este tipo de relación es mas fuerte que la Dependencia es más débil que la Agregación, ya que el tiempo de vida de un objeto no depende de otro.

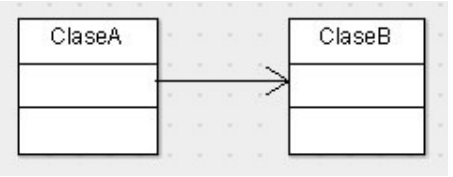
Representación UML

Se representa con una flecha continua que parte desde una clase y apunta a otra. El sentido de la flecha nos indica la clase que se compone (base de la flecha) y sus componentes (punta de la flecha).

Se puede observar que:
La *ClaseA* depende de la *ClaseB*.

La *ClaseA* está asociada a la *ClaseB*.

La *ClaseA* conoce la existencia de la *ClaseB,* pero la *ClaseB* desconoce que existe la *ClaseA*.
Todo cambio en la *ClaseB* podrá afectar a la *ClaseA*.

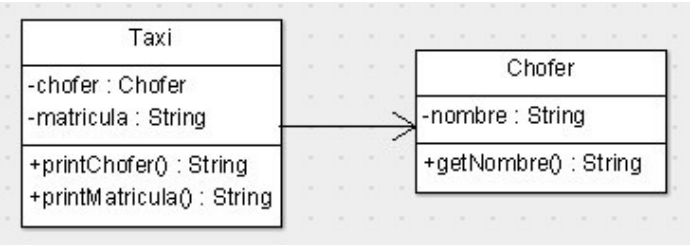


Esto significa que la *ClaseA* tendrá como atributo un objeto o instancia de la *ClaseB* (su componente). La *ClaseA* podrá acceder a las funcionalidades o atributos de su componente usando sus métodos.

Ejemplo Práctico

Tenemos una clase *Taxi* con un atributo *matricula*.
Tenemos una clase *Chofer* con un atributo *nombre*.
Cada *Taxi* necesita ser conducido por un *Chofer.*
Taxi necesita acceder a algunos de los atributos de su *Chofer* (por ejemplo, su nombre).

```
/* Clase Chofer */
class Chofer {
    private String nombre;
    public Chofer(String nombre) {
        this.nombre = nombre;
    }
    public String getNombre() {
        return this.nombre;
    }
}
```




```
Chofer miChofer = new Chofer("Pedro");
Taxi miTaxi = new Taxi(miChofer, "AHJ-1050");
miTaxi.printChofer();
miTaxi.printMatricula();
```

```
/* Clase Taxi */
class Taxi {
    private Chofer chofer;
    private String matricula;
    public Taxi(Chofer chofer, String matricula) {
        this.chofer = chofer;
        this.matricula = matricula;
    }
    public void printMatricula() {
        System.out.println(this.matricula);
    }
    public void printChofer() {
        String nombreChofer = this.chofer.getNombre();
        System.out.println(nombreChofer);
    }
}
```

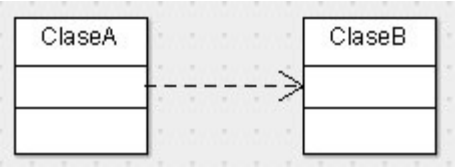
Diagrama de Clases UML: Dependencia

Definición

Es una relación de uso entre dos clases (una usa a la otra). Esta relación es la más básica entre clases y comparada con los demás tipos de relación, la mas débil.

Representación UML

Se representa con una flecha discontinua que parte desde una clase y apunta a otra. El sentido de la flecha nos indica quien usa a quien.



Del diagrama se puede observar que:
La *ClaseA* usa a la *ClaseB*.
La *ClaseA* depende de la *ClaseB*.
Dada la dependencia, todo cambio en la *ClaseB* podrá afectar a la *ClaseA*.
La *ClaseA* conoce la existencia de la *ClaseB* pero la *ClaseB* desconoce que existe la *ClaseA*.

En la practica este tipo de relación se interpreta como que la *ClaseA* hace uso de la *ClaseB* ya sea instanciandola directamente, o bien, recibiéndola como parámetro de entrada en uno de sus métodos.

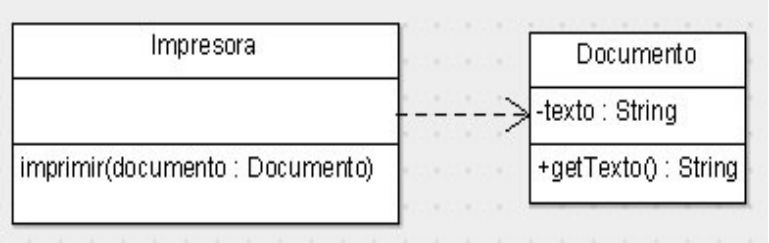
Ejemplo practico

Supongamos lo siguiente:

- Tenemos una clase *Impresora.*.
- Tenemos una clase *Documento* con un atributo *texto*.
- La clase *Impresora* se encarga de *imprimir* los *Documentos.*

```
/* Clase Documento */
class Documento {
    private String texto;
     public Documento(String texto) {this.texto = texto; }
     public String getTexto() {return this.texto;}
}
```

```
/* Clase Impresora */
class Impresora {
     public Impresora() {     }
     public void imprimir(Documento documento) {
         String texto = documento.getTexto();
         System.out.println(texto);
     }
}
```




```
Documento miDocumento = new Documento("Hello World!");
Impresora miImpresora = new Impresora();
miImpresora.imprimir(miDocumento);
```

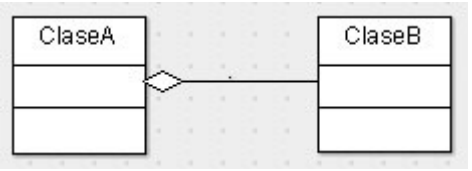
Agregación

Definición

Es muy similar a la relación de Asociación solo varía en la multiplicidad ya que en lugar de ser una relación "uno a uno" es de "uno a muchos".

Representación UML

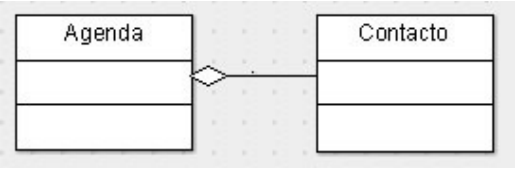
Se representa con una flecha que parte de una clase a otra en cuya base hay un rombo de color blanco.



La *ClaseA* agrupa varios elementos del tipo *ClaseB.*

Ejemplo

- Tenemos una clase *Agenda.*
- Tenemos una clase *Contacto*.
- Una *Agenda* agrupa varios *Contactos*.



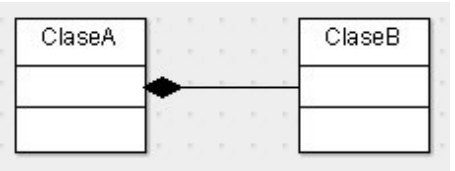
Composición

Definición

Similar a la relación de Agregación solo que la Composición es una relación mas fuerte. Aporta documentación conceptual ya que es una "relación de vida", es decir, el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye.

Representación UML

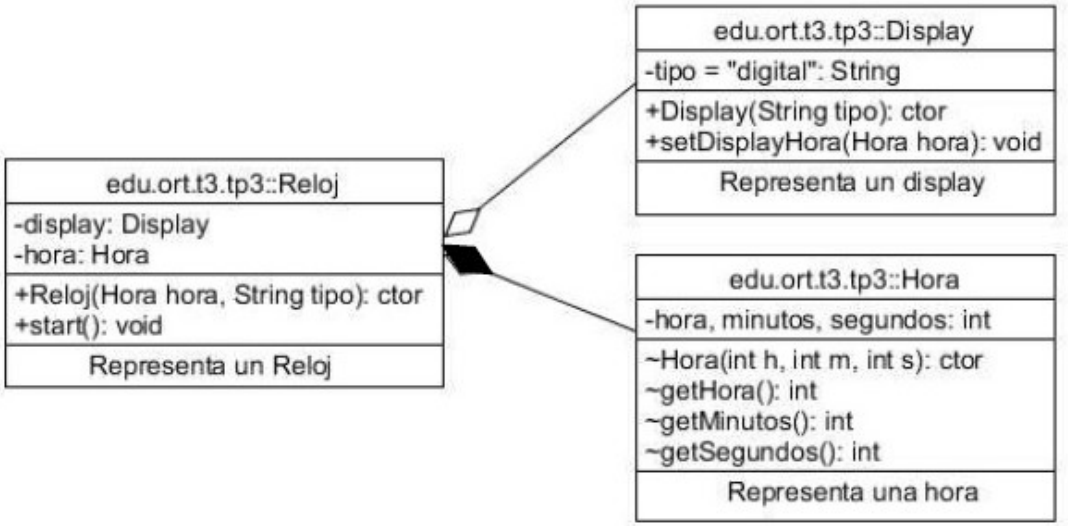
Se representa con una flecha que apunta a la clase mayor y por tanto el rombo va pegado a esta clase.



La *ClaseA* agrupa varios elementos del tipo *ClaseB.*
El tiempo de vida de los objetos de tipo *ClaseB* está condicionado por el tiempo de vida del objeto de tipo *ClaseA*.

Ejemplo
Tenemos una clase *Silla*.
Un objeto *Silla* está a su vez compuesto por cuatro objetos del tipo *Pata*.
El tiempo de vida de los objetos *Pata* depende del tiempo de vida de *Silla*, ya que si no existe una *Silla* no pueden existir sus *Patas*.

Ejercicio




```
public class Reloj {
	private Display display;
	private Hora hora;

	public Reloj(Hora hora, String tipo) {
		this.hora = hora;
		Display display = new Display(tipo);
	}

	public void Start() {
		System.out.println(hora.getHora() + ":" + hora.getMinutos() + ":"
				+ hora.getSegundos());
	 }

  public class Display {
		private String tipo = "digital";
		public Display(String tipo) { this.tipo = tipo; }
		public void setDisplayHora(Hora hora) {
		   if (tipo == "digital")
		   //incompleto
		}
	}

}
```

```
public class Hora {
	private int hora;
	private int minutos;
	private int segundos;
	public Hora(int hora, int minutos, int segundos){
		this.hora=hora;
		this.minutos=minutos;
		this.segundos=segundos;
	}

	public int getHora(){ return hora; }
	public int getMinutos(){ return minutos; }
	public int getSegundos(){ return segundos; }
}
```

Hora hora = new Hora(10,15,55);
Reloj reloj1 = new Reloj(hora, "digital");
Reloj reloj2 = new Reloj(hora, "analogico");
reloj1.Start();
reloj2.Start();