Índice de contenido

1. ¿Qué es una dependencia?

En Java, la dependencia se refiere a la relación entre clases o componentes donde un componente (clase, método, interfaz, etc) utiliza o depende de otro componente para su funcionamiento.

La dependencia implica que un componente requiere el uso de otro componente para realizar ciertas acciones o cumplir con ciertas funcionalidades.

Cuando un componente depende de otro, cualquier cambio en el segundo componente puede afectar al primero.

Por ejemplo: Un automóvil esta conformado por diferentes partes: Llantas, motor, puertas, asientos, frenos, etc. Un automóvil tiene dependencia con todas sus partes para poder funcionar correctamente.

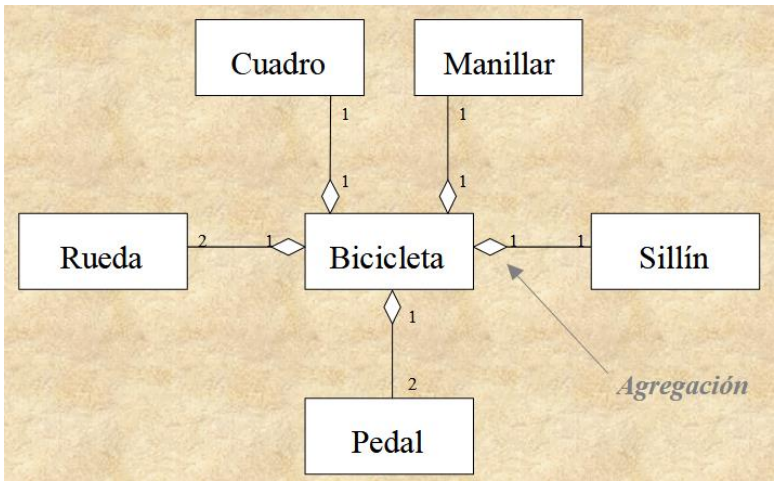
2. Tipos de dependencia

Ya que el concepto de dependencia es muy amplio, podemos encontrar diferentes escenarios de dependencia entre nuestras clases.

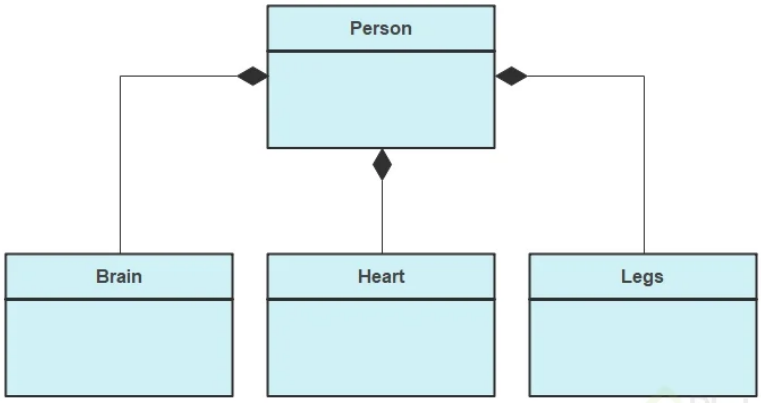
Podemos definir los siguientes tipos de dependencia entre clases:

- **Dependencia de asociación**
  - Se da cuando una clase utiliza otra clase.
  - Ambas clases interactúan entre si y una clase requiere la otra para realizar su trabajo.
  - Por ejemplo: Una clase Sistema tiene un método llamado mostrarFormulario() que utiliza un objeto de la clase Formulario

- **Dependencia de agregación**
  - Es una relación parte-de entre dos clases.
  - Una clase esta conformada por un conjunto de objetos de otras clases.
  - La existencia de los objetos que conforman la clase dependiente son independientes a la existencia de la clase dependiente. Es decir, Si la clase dependiente no existe, los objetos que la conforman pueden seguir existiendo y tener relación con otras clases.
  - Los objetos que conforman la clase dependiente ya pueden existir o crearse aparte para luego formar la clase dependiente.
  - Por ejemplo: La clase Bicicleta esta formada por objetos de la clase Rueda, Cuadro, Manillar, Sillin y Pedal. Si la bicicleta se destruye o se desarma, los objetos que la componen podrían seguir existiendo y formar parte de otra bicicleta o de otro aparato.

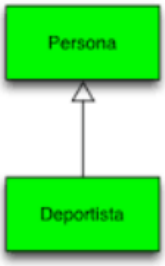


- **Dependencia de composición**
  - Al igual que la dependencia de agregación, es una relación parte-de entre dos clases.
  - Una clase esta conformada por un conjunto de objetos de otras clases.
  - La diferencia radica en que los objetos que conforman la clase dependiente solo pueden pertenecer a una sola clase.
  - Además, si la clase dependiente es destruida, todos los objetos que la conforman también son destruidos; es decir, los objetos no pueden existir sin la clase depentiente.
  - La clase dependiente es la encargada de crear e inicializar los objetos que la componen. No pueden ser creados aparte.
  - Por ejemplo: La clase Persona esta conformada por los objetos de la clase Cerebro, Corazón y Piernas. Si la persona dejase de existir, los objetos tambien dejarían de existir y no podrían ser utilizados por otra persona (metafóricamente hablando).

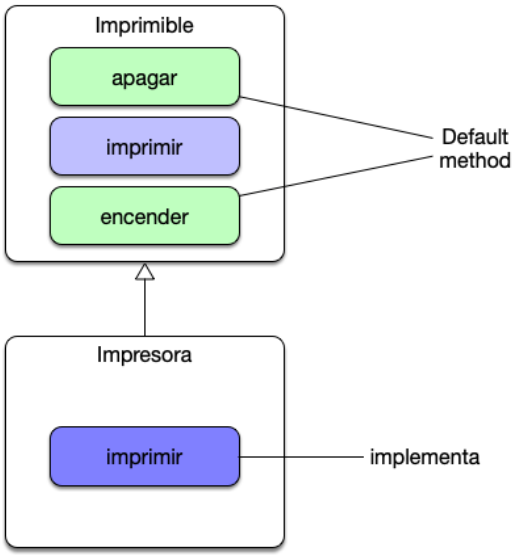


- **Dependencia de herencia**

- Es una relación es-un entre dos clases.
- Se da cuando una clase extiende (*extends*) de otra clase.
- Se tendrá una clase hija (llamada subclase) y una clase padre (llamada superclase).
- La clase hija hereda los atributos y métodos declarados en la clase padre.
- Por ejemplo: La clase Deportista tiene una relación de herencia con la clase Persona, debido a que un deportista es-una persona.



- **Dependencia de implementación**
  - Se da cuando una clase implementa (*implements*) una interfaz.
  - Una interfaz es un contrato que obliga a una clase a implementar sus métodos.
  - La clase que implementa la interfaz debe definir el comportamiento de los métodos declarados en la interfaz.
  - Por ejemplo: La interfaz Imprimible define los métodos (comportamientos) que toda clase Impresora debe tener. Cuando una clase Impresora implementa la interfaz Imprimible, nos aseguramos de que esa impresora cuenta con los métodos mínimos para que funcione de la manera esperada.



3. Definición y manejo de dependencia entre dos clases

Si construyes un programa complejo en java lo mas probable es que termines utilizando una de estas dependencias en tu código.

La dependencia de herencia y dependencia de implementación es bien específica y se explicará mas adelante en clases siguientes.

La dependencia de asociación sería la que hemos estado utilizando en las clases anteriores con el método main: Declaramos un objeto de la clase que queremos testear y utilizamos ese objeto para imprimir sus datos por consola. La relación que existe entre la clase Main y el objeto instanciado en ella es meramente temporal para poder imprimir su información, mas no implica una relación parte-de entre ambas.

Para esta clase crearemos una dependencia de agregación entre dos clases.

3.1. Dependencia de agregación

3.1.1. Definición de la clase dependencia

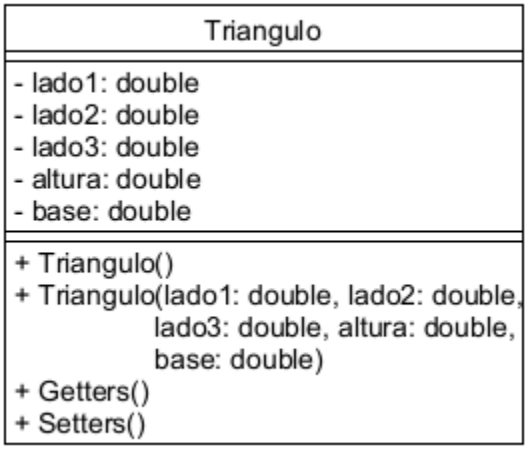
Para nuestro proyecto definiremos una clase POJO Triangulo. Esta clase no contendrá más que los valores de un triángulo común, constructor, getters, setters y toString.

```
1  package com.codewithomarm.clase9.figuras;
2
3  //Clase POJO que almacena los datos de un triángulo
4  public class Triangulo {
5      //Declaración de atributos
6      private double lado1;
7      private double lado2;
8      private double lado3;
9      private double altura;
10     private double base;
11
12     //Constructor vacío por defecto
13     public Triangulo(){}
14
15     //Constructor con parámetros
16     public Triangulo(double lado1, double lado2, double lado3,
17                      double altura, double base) {
18         this.lado1 = lado1;
19         this.lado2 = lado2;
20         this.lado3 = lado3;
21         this.altura = altura;
22         this.base = base;
23     }
24
```

```java
    //Getter atributo lado1
    public double getLado1() {
        return lado1;
    }

    //Setter atributo lado1
    public void setLado1(double lado1) {
        this.lado1 = lado1;
    }

    //Getter atributo lado2
    public double getLado2() {
        return lado2;
    }

    //Setter atributo lado2
    public void setLado2(double lado2) {
        this.lado2 = lado2;
    }

    //Getter atributo lado3
    public double getLado3() {
        return lado3;
    }

    //Setter atributo lado3
    public void setLado3(double lado3) {
        this.lado3 = lado3;
    }

    //Getter atributo altura
    public double getAltura() {
        return altura;
    }

    //Setter atributo latura
    public void setAltura(double altura) {
        this.altura = altura;
    }

    //Getter atributo base
    public double getBase() {
        return base;
```

```
68      }
69
70      //Setter atributo base
71      public void setBase(double base) {
72          this.base = base;
73      }
74
75      @Override
76      public String toString() {
77          return "Triangulo{" +
78                  "lado1=" + lado1 +
79                  ", lado2=" + lado2 +
80                  ", lado3=" + lado3 +
81                  ", altura=" + altura +
82                  ", base=" + base +
83                  '}';
84      }
85  }
```

En diagrama UML, nuestra clase Triangulo se podría ver de la siguiente manera:



3.1.2. Definición de la clase dependiente

Ahora, declararemos una clase llamada `CalculoTriangulo`. Esta clase tendrá como atributo un objeto de la clase `Triangulo`, por lo que habrá una dependencia con la clase `Triangulo`.

Por ahora solo nos enfocaremos en ver como podemos definir y manejar el atributo de una clase cuando este atributo es el objeto de otra clase en vez de un tipo de dato primitivo o wrapper.

En la clase siguiente: Sobrecarga de métodos, veremos la implementación completa de los métodos para calcular las propiedades del triángulo.

Primero declaramos nuestra clase `CalculoTriangulo`:

```
1  package com.codewithomarm.clase9.calculos;
2
3  /*Clase que contiene los procesos para calcular
4  * las propiedas del triángulo*/
5  public class CalculoTriangulo {
6
7  }
```

Luego, declaramos (No instanciamos) un objeto de la clase `Triangulo`. Como nuestro objeto `triangulo` será un atributo dentro de nuestra clase `CalculoTriangulo`, vamos a declararlo con el modificador de acceso `private`:

```
1  package com.codewithomarm.clase9.calculos;
2
3  import com.codewithomarm.clase9.figuras.Triangulo;
4
5  /*Clase que contiene los procesos para calcular
6  * las propiedas del triángulo*/
7  public class CalculoTriangulo {
8      //Declaración de atributos
9      private Triangulo triangulo;
10
11 }
```

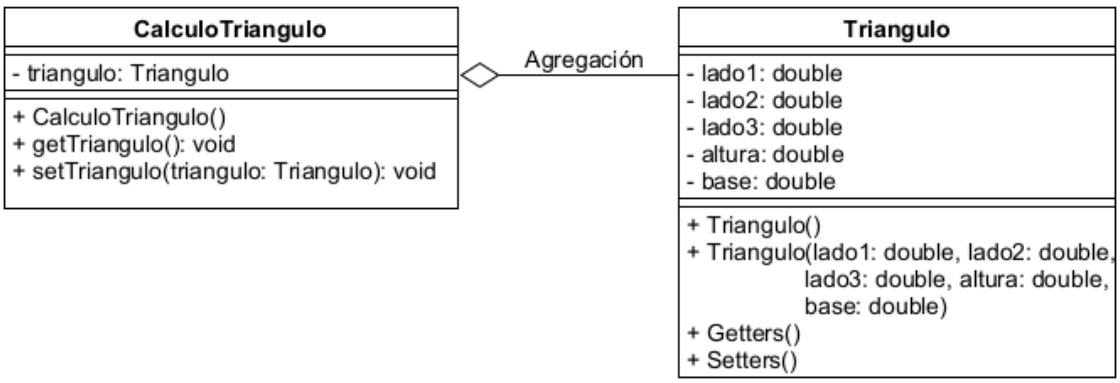
Al igual que cuando declaramos nuestros atributos en clases pasadas, lo hacemos definiendo el modificador de acceso, tipo de dato y nombre del atributo.

En este caso utilizamos el modificador de acceso `private`, el tipo de dato es `Triangulo` (clase `Triangulo`) y el nombre del atributo es el nombre de nuestro objeto `triangulo`.

Al igual que lo haríamos en clases anteriores, tenemos que definir método constructor y getters y setters para nuestro atributo:

```java
package com.codewithomarm.clase9.calculos;

import com.codewithomarm.clase9.figuras.Triangulo;

/*Clase que contiene los procesos para calcular
* las propiedas del triángulo*/
public class CalculoTriangulo {
    //Declaración de atributos
    private Triangulo triangulo;

    //Constructor vacío por defecto
    public CalculoTriangulo(){}

    //Getter atributo triangulo
    public Triangulo getTriangulo() {
        return triangulo;
    }

    //Setter atributo triangulo
    public void setTriangulo(Triangulo triangulo) {
        this.triangulo = triangulo;
    }
}
```

En diagrama UML nuestra clase `CalculoTriangulo`, junto con la clase `Triangulo` quedarían de la siguiente manera:



De esta manera, la clase `CalculoTriangulo` tiene definida una instancia de la clase `Triangulo` como un atributo. Declaramos un método Setter y un método Getter para la variable de instancia `triangulo`.

3.1.3. Manejo del objeto de la clase dependiente

Al igual que en las clases anteriores, declararemos una clase `Main` junto con el método `main(String[] args)` para probar nuestro programa.

En esta ocasión en vez de imprimir los datos de la clase POJO directamente desde su objeto, lo haremos a través de un objeto de la clase dependiente `CalculoTriangulo`.

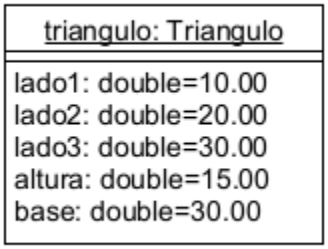
Empezamos declarando e instanciando el objeto de la clase dependencia `Triangulo`:

```
1  package com.codewithomarm.clase9.main;
2
3  import com.codewithomarm.clase9.calculos.CalculoTriangulo;
4  import com.codewithomarm.clase9.figuras.Triangulo;
5
6  //Clase Main para probar la clase CalculoTriangulo
7  public class Main {
8      public static void main(String[] args) {
9          //Declaración e instanciación de objeto de tipo Triangulo
10         Triangulo triangulo = new Triangulo();
11
12     }
13 }
```

Luego le daremos valores a todos los atributos del objeto `triangulo` utilizando los métodos Setter de cada atributo:

```
1  package com.codewithomarm.clase9.main;
2
3  import com.codewithomarm.clase9.figuras.Triangulo;
4
5  //Clase Main para probar la clase CalculoTriangulo
6  public class Main {
7      public static void main(String[] args) {
8          //Declaración e instanciación de objeto de tipo Triangulo
9          Triangulo triangulo = new Triangulo();
10
11         //Definición de los valores de los atributos del objeto triangulo
12         triangulo.setLado1(10.00);
13         triangulo.setLado2(20.00);
14         triangulo.setLado3(30.00);
15         triangulo.setAltura(15.00);
16         triangulo.setBase(30.00);
17
18     }
19 }
```

Tendríamos nuestro diagrama de objetos de la siguiente manera:

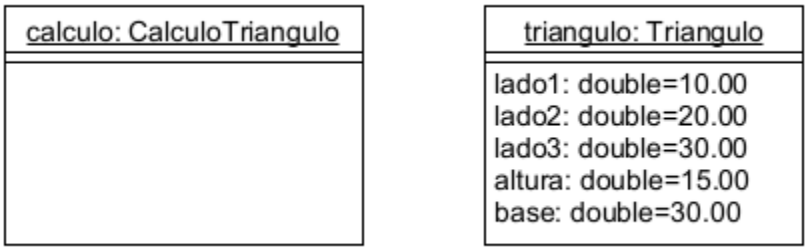


Ahora necesitamos declarar e instanciar el objeto de la clase dependiente `CalculoTriangulo`:

```java
package com.codewithomarm.clase9.main;

import com.codewithomarm.clase9.calculos.CalculoTriangulo;
import com.codewithomarm.clase9.figuras.Triangulo;

//Clase Main para probar la clase CalculoTriangulo
public class Main {
    public static void main(String[] args) {
        //Declaración e instanciación de objeto de tipo Triangulo
        Triangulo triangulo = new Triangulo();

        //Definición de los valores de los atributos del objeto triangulo
        triangulo.setLado1(10.00);
        triangulo.setLado2(20.00);
        triangulo.setLado3(30.00);
        triangulo.setAltura(15.00);
        triangulo.setBase(30.00);

        //Declaración e instanciación de objeto de tipo CalculoTriangulo
        CalculoTriangulo calculo = new CalculoTriangulo();
    }
}
```

De esta manera ya tendríamos los dos objetos instanciados: La clase dependiente y la clase dependencia.

Sin embargo, nuestro objeto dependiente se encuentra vacío, ya que no le hemos asignado ningún objeto de tipo Triangulo; por lo que aún no existe realmente una relación de dependencia entre los dos objetos:

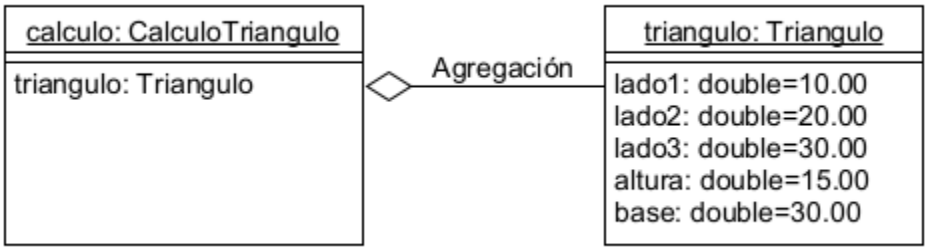


Recordemos que dentro de nuestra clase `CalculoTriangulo`, definimos el método Setter para nuestro atributo triangulo del tipo `Triangulo`. Utilizaremos este método Setter para asignarle el objeto `triangulo` (*dependencia*) a nuestro objeto `calculo` (*dependiente*):

```
1  package com.codewithomarm.clase9.main;
2
3  import com.codewithomarm.clase9.calculos.CalculoTriangulo;
4  import com.codewithomarm.clase9.figuras.Triangulo;
5
6  //Clase Main para probar la clase CalculoTriangulo
7  public class Main {
8      public static void main(String[] args) {
9          //Declaración e instanciación de objeto de tipo Triangulo
10         Triangulo triangulo = new Triangulo();
11
12         //Definición de los valores de los atributos del objeto triangulo
13         triangulo.setLado1(10.00);
14         triangulo.setLado2(20.00);
15         triangulo.setLado3(30.00);
16         triangulo.setAltura(15.00);
17         triangulo.setBase(30.00);
18
19         //Declaración e instanciación de objeto de tipo CalculoTriangulo
20         CalculoTriangulo calculo = new CalculoTriangulo();
21
22         //Definición del objeto triangulo del objeto calculo
23         calculo.setTriangulo(triangulo);
24     }
25 }
```

En la línea 23, invocamos el método `setTriangulo(Triangulo triangulo)` de nuestra clase `CalculoTriangulo`. Le pasamos como argumento nuestro objeto instanciado (Línea 10) y con valores en sus atributos (Línea 13-17) llamado `triangulo`.

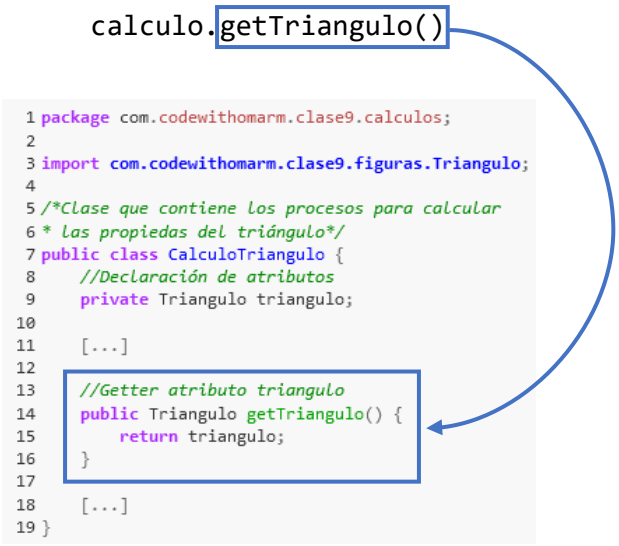
De esta manera, nuestro objeto `calculo` ya tiene su dependencia definida y satisfecha:



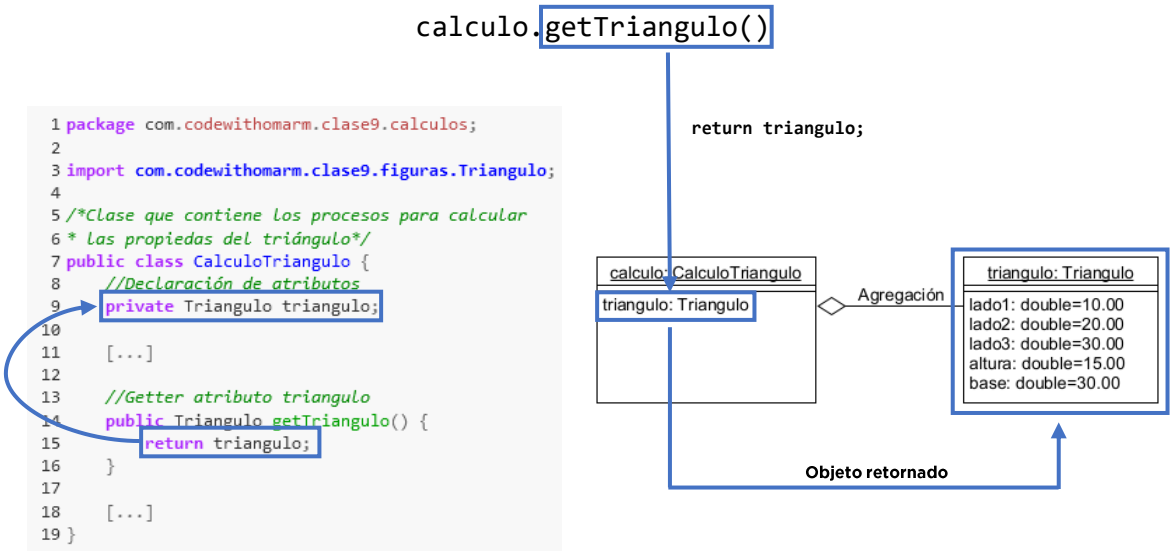
Una vez que nuestro objeto dependiente calculo se encuentra completo con su dependencia satisfecha, podemos entonces mandar a imprimir por consola su atributo triangulo:

```java
package com.codewithomarm.clase9.main;

import com.codewithomarm.clase9.calculos.CalculoTriangulo;
import com.codewithomarm.clase9.figuras.Triangulo;

//Clase Main para probar la clase CalculoTriangulo
public class Main {
    public static void main(String[] args) {
        //Declaración e instanciación de objeto de tipo Triangulo
        Triangulo triangulo = new Triangulo();

        //Definición de los valores de los atributos del objeto triangulo
        triangulo.setLado1(10.00);
        triangulo.setLado2(20.00);
        triangulo.setLado3(30.00);
        triangulo.setAltura(15.00);
        triangulo.setBase(30.00);

        //Declaración e instanciación de objeto de tipo CalculoTriangulo
        CalculoTriangulo calculo = new CalculoTriangulo();

        //Definición del objeto triangulo del objeto calculo
        calculo.setTriangulo(triangulo);

        //Impresión de los datos del atributo triangulo del objeto calculo
        System.out.println(calculo.getTriangulo().toString());
    }
}
```

En la línea 26, hacemos el llamado del método `getTriangulo()` declarado en la clase `CalculoTriangulo`.



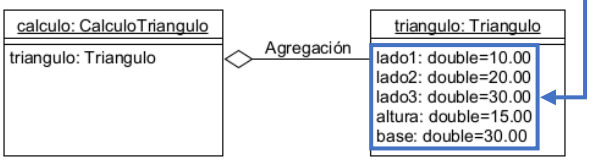
Este método nos retorna el objeto triangulo declarado como atributo privado del tipo Triangulo en nuestra clase como dependencia.



Una vez retornado ese objeto triangulo podemos acceder a los métodos públicos declarados en la clase `Triangulo`. De esta manera, podemos hacer el llamado al método `toString()` de la clase `Triangulo`.

```
calculo.getTriangulo().toString();
```

```
1  package com.codewithomarm.clase9.figuras;
2
3  //Clase POJO que almacena los datos de un triángulo
4  public class Triangulo {
5      //Declaración de atributos
6      private double lado1;
7      private double lado2;
8      private double lado3;
9      private double altura;
10     private double base;
11
12     [...]
13
14     @Override
15     public String toString() {
16         return "Triangulo{" +
17                 "lado1=" + lado1 +
18                 ", lado2=" + lado2 +
19                 ", lado3=" + lado3 +
20                 ", altura=" + altura +
21                 ", base=" + base +
22                 '}';
23     }
24 }
```



Podemos ejecutar el programa y ver el resultado de la consola:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "[...]"

Triangulo{lado1=10.0, lado2=20.0, lado3=30.0, altura=15.0,
base=30.0}

Process finished with exit code 0
```

De esta manera hemos declarado una dependencia de agregación entre dos clases y hemos accedido a los métodos públicos del objeto dependencia a través del objeto dependiente.

Esto es sumamente importante de entender y saber utilizar ya que si tenemos clases POJO donde solo declararemos los datos de nuestro programa, necesitamos crear otras clases con los procesos y lógica del negocio donde realmente manipularemos esos datos y ejecutaremos procesos.