

## Índice de contenido

1. ¿Qué es la sobrecarga de constructores?
2. Definición de múltiples constructores
  - 2.1. Sobrecarga de constructores con distinta cantidad de parámetros
  - 2.2. Sobrecarga de constructores con misma cantidad de parámetros
3. Ejemplo sobrecarga de constructores: Clase CalculoTriangulo
4. ¿Qué es la sobrecarga de métodos?
5. Ejemplo sobrecarga de métodos: Clase CalculoBono

## 1. ¿Qué es la sobrecarga de constructores?

La sobrecarga de constructores nos permite declarar múltiples constructores con el mismo nombre para una misma clase.

Se utiliza la sintaxis propia del constructor de una clase pero se asignan diferentes cantidades y/o tipos de parámetro en cada constructor.

Java determina cuál constructor utilizar basándose en la cantidad y los tipos de argumentos proporcionados en el momento en que se llama al método constructor en la instanciación de un objeto.

## 2. Definición de múltiples constructores

Para definir múltiples constructores se utiliza la sintaxis:

```
public NombreClase( ) { };
```

↑  
**Parámetros**

- Donde podrá tener 0 a N cantidad de parámetros.
- En el caso de que dos constructores de la misma clase tengan la misma cantidad de parámetros, deberán tener por lo menos un (1) parámetro de tipo de dato distinto para que Java pueda determinar cuál método constructor utilizar. De lo contrario se producirá una ambigüedad ya que Java no podrá determinar cuál constructor utilizar y se producirá un error de compilación.

### 2.1. Sobrecarga de constructores con distinta cantidad de parámetros

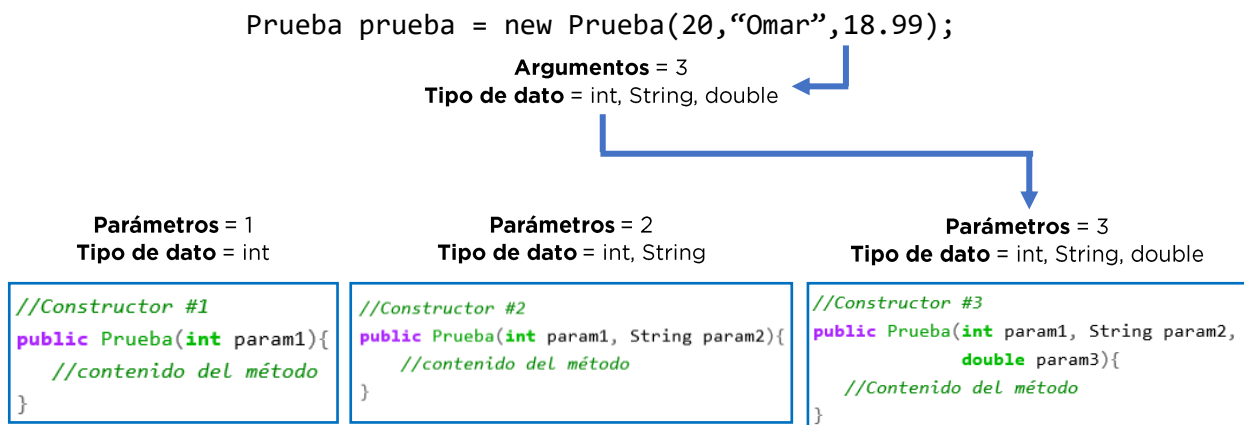
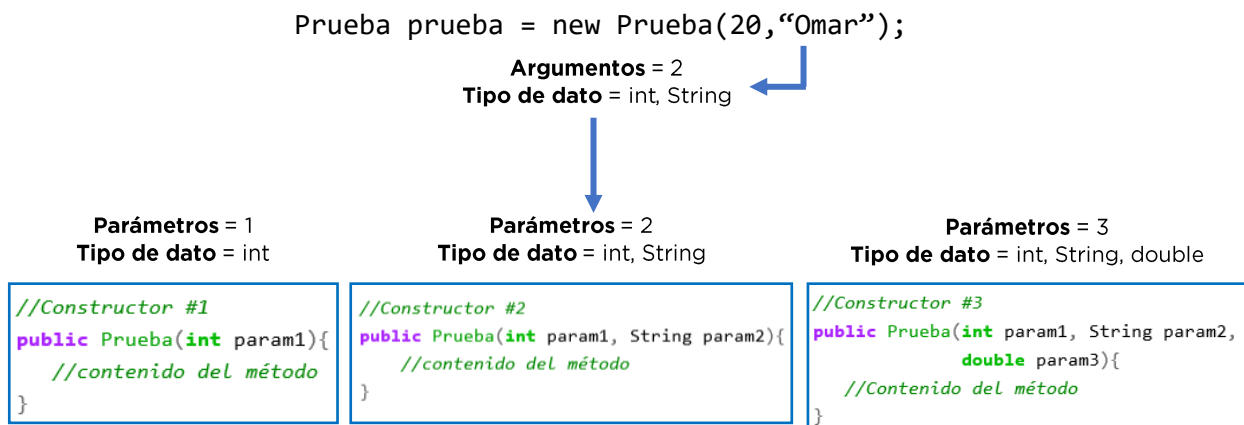
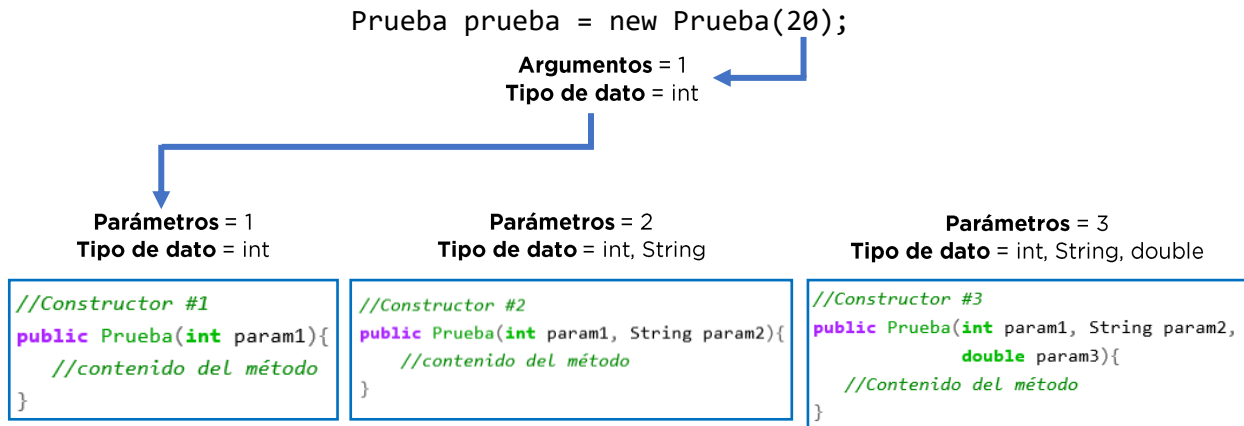
La clase Prueba tiene tres métodos constructores con diferente cantidad de parámetros:

```
1 public class Prueba{
2
3     //Constructor #1
4     public Prueba(int param1){
5         //contenido del método
6     }
7
8     //Constructor #2
9     public Prueba(int param1, String param2){
10        //contenido del método
11    }
12
13    //Constructor #3
14    public Prueba(int param1, String param2,
15                  double param3){
16        //Contenido del método
17    }
18 }
```

El constructor de la línea 4 tiene 1 parámetro. El constructor de la línea 9 tiene 2 parámetros.

El constructor de la línea 14 tiene 3 parámetros.

Si llamamos al método constructor cuando instanciamos un objeto de la clase Prueba, Java decidirá cuál constructor usar primero en base a la cantidad de argumentos que tenga el llamado al constructor. Si los argumentos coinciden con los tipos de datos del constructor, entonces ejecutará ese constructor:



## 2.2. Sobrecarga de constructores con misma cantidad de parámetros

La clase Prueba2 declara tres métodos constructores. Todos tienen solo 1 parámetro.

En estos casos en que se sobrecarga el constructor utilizando la misma cantidad de parámetros, cada parámetro debe ser de diferente tipo de dato:

```
1 public class Prueba2{
2
3     //Constructor #1
4     public Prueba(int param1){
5         //contenido del método
6     }
7
8     //Constructor #2
9     public Prueba(String param1){
10        //contenido del método
11    }
12
13    //Constructor #3
14    public Prueba(double param1){
15        //Contenido del método
16    }
17 }
```

Si llamamos al método constructor cuando instanciamos un objeto de la clase Prueba2, Java decidirá cuál constructor usar primero en base a la cantidad de argumentos que tenga el llamado al constructor.

En este caso todos los constructores tienen la misma cantidad de parámetros (1), por lo que Java utilizará el constructor cuyo tipo de dato del parámetro coincida con el tipo de dato del argumento del llamado al método constructor:

Prueba2 prueba2 = new Prueba2(20);

Argumentos = 1  
Tipo de dato = int

Parámetros = 1  
Tipo de dato = int

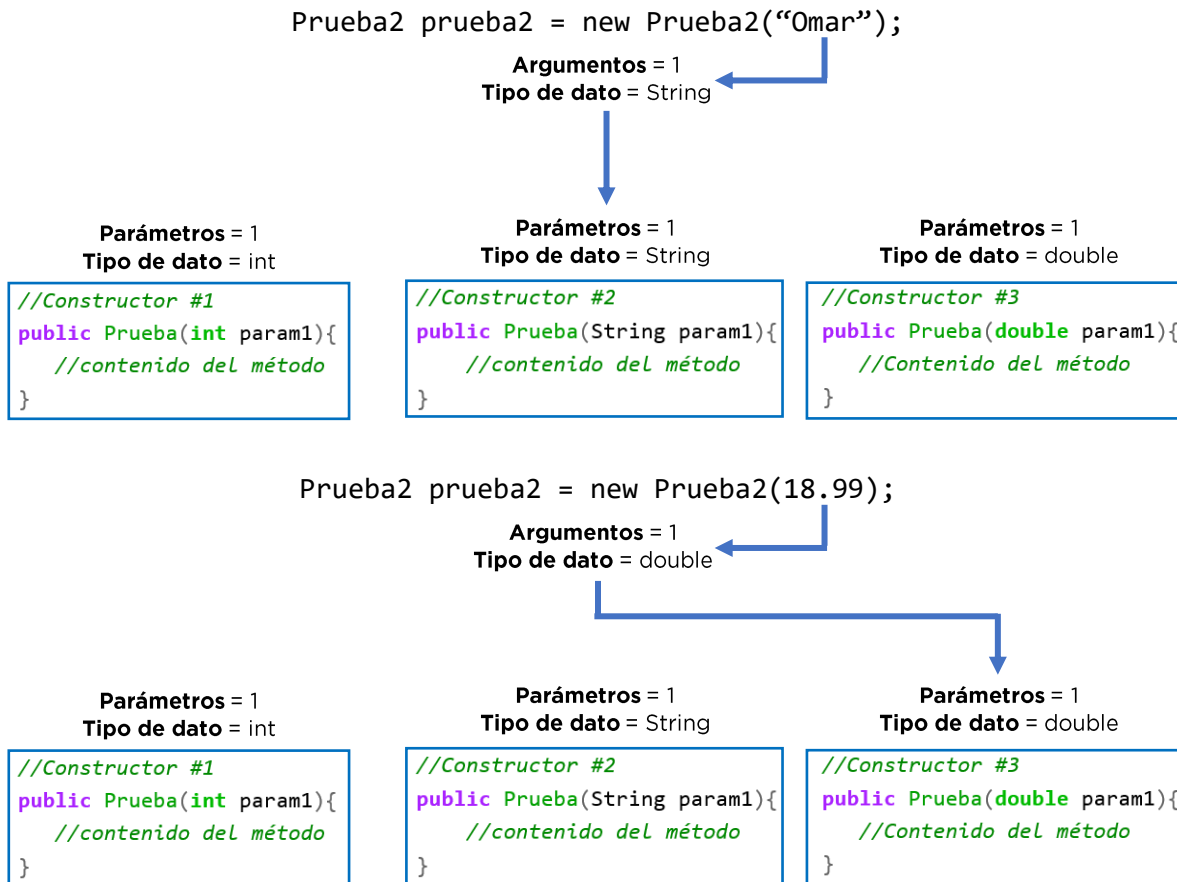
```
//Constructor #1
public Prueba(int param1){
    //contenido del método
}
```

Parámetros = 1  
Tipo de dato = String

```
//Constructor #2
public Prueba(String param1){
    //contenido del método
}
```

Parámetros = 1  
Tipo de dato = double

```
//Constructor #3
public Prueba(double param1){
    //Contenido del método
}
```



### 3. Ejemplo sobrecarga de constructores: Clase CalculoTriangulo

En la “clase 9 – Dependencia de agregación entre clases” creamos una clase POJO llamada Triangulo.

Esta clase declara 5 atributos de tipo double para guardar los datos de cada triangulo.

Se definen dos métodos constructores: Constructor por defecto sin parámetros y un constructor con todos los parámetros. Aquí se estaría aplicando sobrecarga de constructores.

Los demás métodos son los Getters y Setters de cada atributo junto con el método toString de la clase.

```
1 package com.codewithomarm.clase10.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(){} ← Sobrecarga de constructores
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
25     //Getter atributo lado1
26     public double getLado1() {
27         return lado1;
28     }
29
30     //Setter atributo lado1
31     public void setLado1(double lado1) {
32         this.lado1 = lado1;
33     }
34
35     //Getter atributo lado2
36     public double getLado2() {
37         return lado2;
38     }
39
40     //Setter atributo lado2
41     public void setLado2(double lado2) {
42         this.lado2 = lado2;
43     }
44
45     //Getter atributo lado3
46     public double getLado3() {
47         return lado3;
48     }
49
50     //Setter atributo lado3
51     public void setLado3(double lado3) {
52         this.lado3 = lado3;
```

```
53     }
54
55     //Getter atributo altura
56     public double getAltura() {
57         return altura;
58     }
59
60     //Setter atributo latura
61     public void setAltura(double altura) {
62         this.altura = altura;
63     }
64
65     //Getter atributo base
66     public double getBase() {
67         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 }
```

Para manipular los datos de la clase POJO y declarar nuestros procesos para calcular las propiedades de los distintos triángulos, declaramos una clase CalculoTriangulo.

Esta clase define una dependencia de agregación con respecto a la clase Triangulo, declarando un objeto de la clase Triangulo como un atributo privado.

La clase CalculoTriangulo también hace uso de sobrecarga de constructores definiendo tres tipos distintos de constructores:

**1. Constructor por defecto sin parámetros.**

- a. Se utilizará para crear un objeto de tipo CalculoTriangulo sin proveer un objeto triangulo (dependencia) al momento de instanciar el objeto.

**2. Constructor con 1 parámetro de tipo Triángulo.**

- a. Se utilizará para crear un objeto de tipo CalculoTriangulo y a la misma vez asignarle un objeto triangulo (dependencia).
  - i. El objeto triangulo (dependencia) pudo haberse instanciado anteriormente y pasárselo como argumento al constructor.
  - ii. El objeto triangulo (dependencia) también podría instanciarse directamente en el parámetro del constructor del objeto CalculoTriangulo.

**3. Constructor con 5 parámetros de tipo double.**

- a. Se utilizará para crear un objeto de tipo CalculoTriangulo y a la misma vez asignarle un objeto triangulo (dependencia) al pasarle los 5 argumentos para que el método constructor instancia el objeto triangulo (dependencia).

A parte de los constructores, la clase CalculoTriangulo define un método publico que imprime en consola el resultado de todos los cálculos de las propiedades del triángulo. Para esto define subrutinas privadas para hacer los distintos cálculos.

```
1 package com.codewithomarm.clase10.calculos;
2
3 import com.codewithomarm.clase10.figuras.Triangulo;
4
5 /*Clase que contiene los procesos para calcular
6 * las propiedades del triángulo*/
7 public class CalculoTriangulo {
8     //Declaración de atributos
9     private Triangulo triangulo;
10    private static final double pi = 3.14;
11
12    //Constructor vacío por defecto
13    public CalculoTriangulo(){
14
15    //Constructor con parámetros
16    public CalculoTriangulo(Triangulo triangulo){
17        this.triangulo = triangulo;
18    }
```



```
19
20 //Constructor con parámetros de los atributos del triángulo
21 public CalculoTriangulo(double lado1, double lado2, double lado3,
22                         double altura, double base){
23     this.triangulo = new Triangulo(lado1, lado2, lado3, altura, base);
24 }
25
26 //Setter atributo triangulo
27 public void setTriangulo(Triangulo triangulo) {
28     this.triangulo = triangulo;
29 }
30
31 //Mostrar Todos los calculos del triangulo
32 public String mostrarCalculos(){
33     return triangulo.toString() + "\n" +
34         "Calculos del Triángulo {" + "\n" +
35         "\t" + "Area = " + calcularArea() + "\n" +
36         "\t" + "Perímetro = " + calcularPerimetro() + "\n" +
37         "\t" + "Semiperímetro = " + calcularSemiperimetro() + "\n" +
38         "\t" + "Circunferencia Circunscrita = " + calcularCircunfCirc() + "\n" +
39         "\t" + "Circunferencia Inscrita = " + calcularCircunfInsc() + "\n" +
40         '}';
41 }
42
43 //Calcular el area del triángulo
44 private double calcularArea(){
45     double area;
46     area = (triangulo.getBase() * triangulo.getAltura()) / 2;
47     return area;
48 }
49
50 //Calcular el perímetro del triángulo
51 private double calcularPerimetro(){
52     double perimetro;
53     perimetro = triangulo.getLado1() + triangulo.getLado2() +
54         triangulo.getLado3();
55     return perimetro;
56 }
57
58 //Calcular el semiperímetro del triángulo
59 private double calcularSemiperimetro() {
60     double semi;
61     semi = calcularPerimetro() / 2;
62     return semi;
63 }
64
65 //Calcular la circunferencia circunscrita del triángulo
66 private double calcularCircunfCirc(){
```

```

67     double radio;
68     radio = (triangulo.getLado1() * triangulo.getLado2() *
69             triangulo.getLado3()) / (4 * calcularArea());
70     double circunferencia;
71     circunferencia = pi * (radio * 2);
72     return circunferencia;
73 }
74
75 //Calcular la circunferencia Inscrita del triángulo
76 private double calcularCircunfInsc(){
77     double radio;
78     radio = calcularArea() / calcularSemiperimetro();
79     double circunferencia;
80     circunferencia = pi * (radio * 2);
81     return circunferencia;
82 }
83 }

```

En el método main utilizaremos 4 objetos de tipo CalculoTriangulo para utilizar los distintos métodos constructores de esta clase:

1. Primero, utilizaremos el método constructor por defecto sin parámetros.
  - a. Declaramos e instanciamos un objeto triangulo1 (Línea 10).
  - b. Definimos los valores de los atributos del objeto triangulo1 (Línea 12-16).
  - c. Declaramos e instanciamos un objeto calculo1 (Línea 19)
  - d. Asignamos la dependencia utilizando el método setTriangulo( ) de la clase CalculoTriangulo (Línea 22).
  - e. Dentro de un método System.out.print(), llamamos al método public del objeto calculo1: mostrarCalculos() para que se hagan los cálculos de las propiedades de triangulo1 e imprima por consola los datos de triangulo1 y sus propiedades (Línea 26).

```

1  package com.codewithomarm.clase10.main;
2
3  import com.codewithomarm.clase10.calculos.CalculoTriangulo;
4  import com.codewithomarm.clase10.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 triangulo1 = new Triangulo();
11         //Definición de los valores de los atributos del objeto triangulo1
12         triangulo1.setLado1(15.00);
13         triangulo1.setLado2(25.00);
14         triangulo1.setLado3(35.00);
15         triangulo1.setAltura(28.00);
16         triangulo1.setBase(25.00);
17
18         //Declaración e instanciación de objeto de tipo CalculoTriangulo

```

```

19     CalculoTriangulo calculo1 = new CalculoTriangulo();
20
21     //Asignar objeto triangulo1 al objeto calculo1 para satisfacer dependencia
22     calculo1.setTriangulo(triangulo1);
23
24     //Impresión de los calculos de las propiedades del calculo1 con el triangulo1
25     System.out.println("-----Objeto triangulo1-----");
26     System.out.println(calculo1.mostrarCalculos());
27 }
28 }

```

### Output por consola

```

"C:\Program Files\Java\jdk-21\bin\java.exe" [...]
-----Objeto triangulo1-----
Triangulo{lado1=15.0, lado2=25.0, lado3=35.0, altura=28.0, base=25.0}
Calculos del Triángulo {
    Area = 350.0
    Perímetro = 75.0
    Semiperímetro = 37.5
    Circunferencia Circunscrita = 58.875
    Circunferencia Inscrita = 58.61333333333334
}

Process finished with exit code 0

```

2. Segundo, utilizaremos el método constructor con 1 parámetro de tipo Triangulo
  - a. Declaramos e instanciamos un objeto triangulo2 (Línea 29).
  - b. Definimos los valores de los atributos del objeto triangulo2 (Línea 31-35).
  - c. Declaramos e instanciamos un objeto calculo2 y a la misma vez, asignamos la dependencia a través del argumento del método constructor. (Línea 39).

```

37     /*Declaración e instanciación de objeto de tipo CalculoTriangulo
38     *a la misma vez que se le asigna el objeto triangulo2 */
39     CalculoTriangulo calculo2 = new CalculoTriangulo(triangulo2);

```

```

//Constructor con parámetros
public CalculoTriangulo(Triangulo triangulo){
    this.triangulo = triangulo;
}

```

- d. Dentro de un método System.out.print(), llamamos al método public del objeto calculo2: mostrarCalculos() para que se hagan los cálculos de las propiedades de triangulo2 e imprima por consola los datos de triangulo2 y sus propiedades (Línea 43).

```

1 package com.codewithomarm.clase10.main;
2
3 import com.codewithomarm.clase10.calculos.CalculoTriangulo;
4 import com.codewithomarm.clase10.figuras.Triangulo;

```

```

5
6 //Clase Main para probar La clase CalculoTriangulo
7 public class Main {
8     public static void main(String[] args) {

        [...]

28     //Declaración e instanciación de objeto de tipo Triangulo
29     Triangulo triangulo2 = new Triangulo();
30     //Definición de los valores de los atributos del objeto triangulo1
31     triangulo2.setLado1(40.00);
32     triangulo2.setLado2(40.00);
33     triangulo2.setLado3(30.00);
34     triangulo2.setAltura(40.00);
35     triangulo2.setBase(40.00);
36
37     /*Declaración e instanciación de objeto de tipo CalculoTriangulo
38     *a la misma vez que se le asigna el objeto triangulo2 */
39     CalculoTriangulo calculo2 = new CalculoTriangulo(triangulo2);
40
41     //Impresión de los calculos de las propiedades del calculo2 con el triangulo2
42     System.out.println("-----Objeto triangulo2-----");
43     System.out.println(calculo2.mostrarCalculos());
44     }
45 }

```

### Output por consola

```

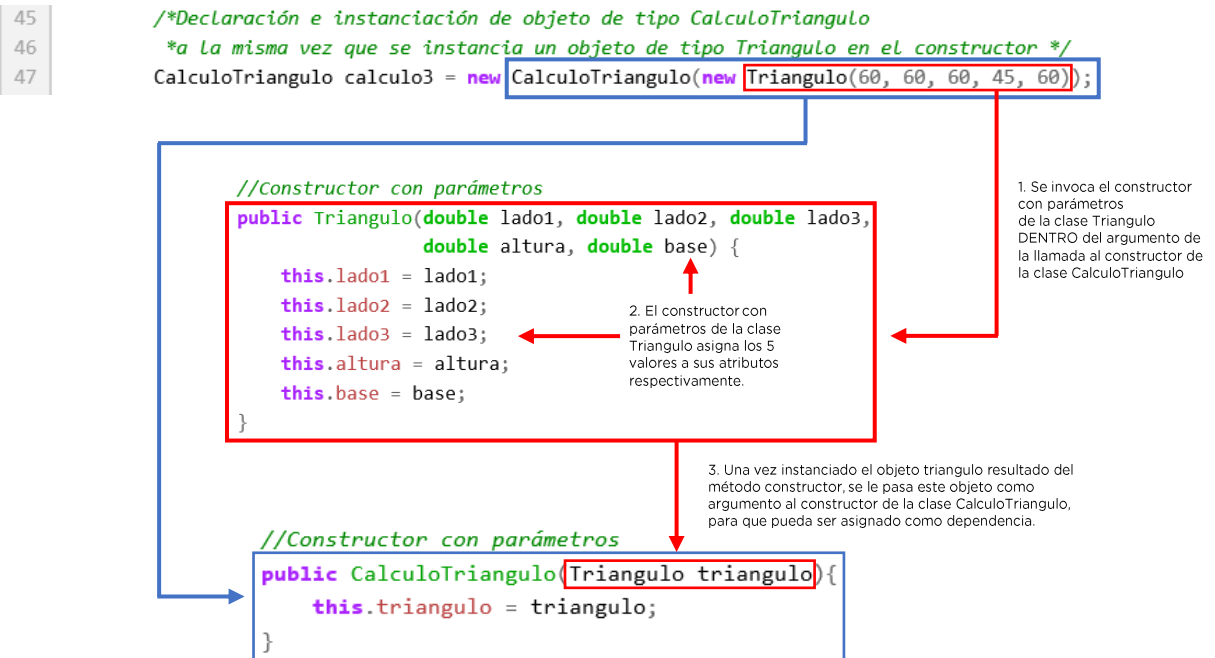
"C:\Program Files\Java\jdk-21\bin\java.exe"[...]
-----Objeto triangulo1-----
Triangulo{lado1=15.0, lado2=25.0, lado3=35.0, altura=28.0, base=25.0}
Calculos del Triángulo {
    Area = 350.0
    Perímetro = 75.0
    Semiperímetro = 37.5
    Circunferencia Circunscrita = 58.875
    Circunferencia Inscrita = 58.61333333333334
}
-----Objeto triangulo2-----
Triangulo{lado1=40.0, lado2=40.0, lado3=30.0, altura=40.0, base=40.0}
Calculos del Triángulo {
    Area = 800.0
    Perímetro = 110.0
    Semiperímetro = 55.0
    Circunferencia Circunscrita = 94.2
    Circunferencia Inscrita = 91.34545454545454
}

Process finished with exit code 0

```

3. Tercero, utilizaremos nuevamente el constructor con 1 parámetro de tipo Triangulo.
- a. Declaramos e instanciamos un objeto calculo3 (Línea 45).

- En este caso nos saltamos la parte de declarar e instanciar un objeto de tipo Triangulo y también asignarle valores al objeto triangulo antes de declarar e instanciar el objeto calculo3.
- En reemplazo, lo que hacemos es instanciar (**no declarar**) un objeto triangulo **directamente** en el argumento del método constructor.
  - Esto es posible ya que el parámetro es un objeto de tipo Triangulo. El parámetro no le importa en que momento se instancie ese objeto.
  - Este objeto puede estar ya declarado e instanciado o puede instanciarse directamente en el argumento del método.



- b. Dentro de un método System.out.print(), llamamos al método public del objeto calculo3: mostrarCalculos() para que se hagan los cálculos de las propiedades de su atributo de tipo Triangulo e imprima por consola los datos de triangulo3 y sus propiedades (Línea 51).

```

1  package com.codewithomarm.clase10.main;
2
3  import com.codewithomarm.clase10.calculos.CalculoTriangulo;
4  import com.codewithomarm.clase10.figuras.Triangulo;
5
6  //Clase Main para probar la clase CalculoTriangulo
7  public class Main {
8      public static void main(String[] args) {
9
10         [...]
11
12         /*Declaración e instanciación de objeto de tipo CalculoTriangulo
  
```

```
46      *a la misma vez que se instancia un objeto de tipo Triangulo en el constructor */
47      CalculoTriangulo calculo3 = new CalculoTriangulo(new Triangulo(60, 60, 60, 45, 60));
48
49      //Impresión de los calculos de las propiedades del calculo3
50      System.out.println("-----Objeto triangulo3-----");
51      System.out.println(calculo3.mostrarCalculos());
52  }
53 }
```

### Output por consola

```
"C:\Program Files\Java\jdk-21\bin\java.exe"[...]
-----Objeto triangulo1-----
Triangulo{lado1=15.0, lado2=25.0, lado3=35.0, altura=28.0, base=25.0}
Calculos del Triángulo {
    Area = 350.0
    Perímetro = 75.0
    Semiperímetro = 37.5
    Circunferencia Circunscrita = 58.875
    Circunferencia Inscrita = 58.61333333333334
}
-----Objeto triangulo2-----
Triangulo{lado1=40.0, lado2=40.0, lado3=30.0, altura=40.0, base=40.0}
Calculos del Triángulo {
    Area = 800.0
    Perímetro = 110.0
    Semiperímetro = 55.0
    Circunferencia Circunscrita = 94.2
    Circunferencia Inscrita = 91.34545454545454
}
-----Objeto triangulo3-----
Triangulo{lado1=60.0, lado2=60.0, lado3=60.0, altura=45.0, base=60.0}
Calculos del Triángulo {
    Area = 1350.0
    Perímetro = 180.0
    Semiperímetro = 90.0
    Circunferencia Circunscrita = 251.20000000000002
    Circunferencia Inscrita = 94.2
}

Process finished with exit code 0
```

4. Últimamente, utilizamos el constructor con 5 parámetros de tipo double.
- Declaramos e instanciamos un objeto calculo4 (Línea 55).
    - Los 5 argumentos que utiliza este constructor, los utiliza para instanciar un objeto de tipo triangulo.
    - A diferencia del caso anterior, la instanciación del objeto triangulo no ocurre directamente en el argumento del constructor de calculo, sino dentro del cuerpo del método constructor.

```

53  /*Declaración e instanciación de objeto de tipo CalculoTriangulo
54  *pasandole los valores de los atributos de su dependencia triangulo */
55  CalculoTriangulo calculo4 = new CalculoTriangulo(5, 3, 2, 1.5, 5);

```

```

//Constructor con parámetros de los atributos del triángulo
public CalculoTriangulo(double lado1, double lado2, double lado3,
                        double altura, double base){
    this.triangulo = new Triangulo(lado1, lado2, lado3, altura, base);
}

```

- Dentro de un método System.out.print(), llamamos al método public del objeto calculo3: mostrarCalculos() para que se hagan los cálculos de las propiedades de su atributo de tipo Triangulo e imprima por consola los datos de triangulo3 y sus propiedades (Línea 59).

```

1  package com.codewithomarm.clase10.main;
2
3  import com.codewithomarm.clase10.calculos.CalculoTriangulo;
4  import com.codewithomarm.clase10.figuras.Triangulo;
5
6  //Clase Main para probar la clase CalculoTriangulo
7  public class Main {
8      public static void main(String[] args) {
9
10         [...]
11
12         /*Declaración e instanciación de objeto de tipo CalculoTriangulo
13         *pasandole los valores de los atributos de su dependencia triangulo */
14         CalculoTriangulo calculo4 = new CalculoTriangulo(5, 3, 2, 1.5, 5);
15
16         //Impresión de los calculos de las propiedades del calculo3
17         System.out.println("-----Objeto triangulo4-----");
18         System.out.println(calculo4.mostrarCalculos());
19     }
20 }

```

### Output por consola

```

"C:\Program Files\Java\jdk-21\bin\java.exe"[...]
-----Objeto triangulo1-----
Triangulo{lado1=15.0, lado2=25.0, lado3=35.0, altura=28.0, base=25.0}
Calculos del Triángulo {
    Area = 350.0
    Perímetro = 75.0
}

```

```
Semiperímetro = 37.5
Circunferencia Circunscrita = 58.875
Circunferencia Inscrita = 58.61333333333334
}
-----Objeto triangulo2-----
Triangulo{lado1=40.0, lado2=40.0, lado3=30.0, altura=40.0, base=40.0}
Calculos del Triángulo {
    Area = 800.0
    Perímetro = 110.0
    Semiperímetro = 55.0
    Circunferencia Circunscrita = 94.2
    Circunferencia Inscrita = 91.34545454545454
}
-----Objeto triangulo3-----
Triangulo{lado1=60.0, lado2=60.0, lado3=60.0, altura=45.0, base=60.0}
Calculos del Triángulo {
    Area = 1350.0
    Perímetro = 180.0
    Semiperímetro = 90.0
    Circunferencia Circunscrita = 251.20000000000002
    Circunferencia Inscrita = 94.2
}
-----Objeto triangulo4-----
Triangulo{lado1=5.0, lado2=3.0, lado3=2.0, altura=1.5, base=5.0}
Calculos del Triángulo {
    Area = 3.75
    Perímetro = 10.0
    Semiperímetro = 5.0
    Circunferencia Circunscrita = 12.56
    Circunferencia Inscrita = 4.71
}

Process finished with exit code 0
```

Podemos observar cómo utilizar sobrecarga de constructores nos permite tener diferentes maneras de instanciar nuestros objetos, dándonos bastante libertad y posibilidades a la hora de crear nuestros proyectos.



#### 4. ¿Qué es la sobrecarga de métodos?

La sobrecarga de métodos cumple con el mismo principio de que la sobrecarga de constructores: Definir un método y sobrecargarlo con distintos comportamientos en base a los parámetros que se definen.

La sobrecarga de constructores y métodos funcionan exactamente igual ya que al final los métodos constructores son eso, métodos.

De igual manera, Java decidirá cuál método utilizar en base a la cantidad de argumentos y los tipos de datos de esos argumentos que coincidan con alguno de los métodos declarados con ese nombre, cantidad de parámetros y tipos de datos.

#### 5. Ejemplo sobrecarga de métodos: Clase CalculoBono

Para este ejemplo construiremos un programa Java por consola que calcule el bono de final de año para cada tipo de empleado: Agente, Supervisor y Gerente. Cada tipo de empleado tiene un porcentaje distinto en base a su categoría.

Para ellos hemos definido tres clase POJO: Agente, Supervisor y Gerente:

##### Clase Agente

```
1 package com.codewithomarm.clase10.trabajadores;
2
3 public class Agente {
4     //Declaración de atributos
5     private String nombre;
6     private String apellido;
7     private String departamento;
8     private double salario;
9
10    //Constructor con parámetros
11    public Agente(String nombre, String apellido, String departamento, double salario) {
12        this.nombre = nombre;
13        this.apellido = apellido;
14        this.departamento = departamento;
15        this.salario = salario;
16    }
17
18    //Getter atributo nombre
19    public String getNombre() {
20        return nombre;
21    }
22
23    //Setter atributo nombre
24    public void setNombre(String nombre) {
25        this.nombre = nombre;
26    }
27
28    //Getter atributo apellido
29    public String getApellido() {
30        return apellido;
31    }
```

```
32
33 //Setter atributo apellido
34 public void setApellido(String apellido) {
35     this.apellido = apellido;
36 }
37
38 //Getter atributo departamento
39 public String getDepartamento() {
40     return departamento;
41 }
42
43 //Setter atributo departamento
44 public void setDepartamento(String departamento) {
45     this.departamento = departamento;
46 }
47
48 //Getter atributo salario
49 public double getSalario() {
50     return salario;
51 }
52
53 //Setter atributo salario
54 public void setSalario(double salario) {
55     this.salario = salario;
56 }
57
58 @Override
59 public String toString() {
60     return "Gerente{" +
61         "nombre='" + nombre + '\'' +
62         ", apellido='" + apellido + '\'' +
63         ", departamento='" + departamento + '\'' +
64         ", salario=" + salario +
65         '}';
66 }
67 }
```

### Clase Supervisor

```
1 package com.codewithomarm.clase10.trabajadores;
2
3 public class Supervisor {
4     //Declaración de atributos
5     private String nombre;
6     private String apellido;
7     private String departamento;
8     private double salario;
9
10    //Constructor con parámetros
11    public Supervisor(String nombre, String apellido, String departamento, double salario) {
12        this.nombre = nombre;
13        this.apellido = apellido;
14        this.departamento = departamento;
15        this.salario = salario;
16    }
17 }
```

```
16     }
17
18     //Getter atributo nombre
19     public String getNombre() {
20         return nombre;
21     }
22
23     //Setter atributo nombre
24     public void setNombre(String nombre) {
25         this.nombre = nombre;
26     }
27
28     //Getter atributo apellido
29     public String getApellido() {
30         return apellido;
31     }
32
33     //Setter atributo apellido
34     public void setApellido(String apellido) {
35         this.apellido = apellido;
36     }
37
38     //Getter atributo departamento
39     public String getDepartamento() {
40         return departamento;
41     }
42
43     //Setter atributo departamento
44     public void setDepartamento(String departamento) {
45         this.departamento = departamento;
46     }
47
48     //Getter atributo salario
49     public double getSalario() {
50         return salario;
51     }
52
53     //Setter atributo salario
54     public void setSalario(double salario) {
55         this.salario = salario;
56     }
57
58     @Override
59     public String toString() {
60         return "Gerente{" +
61             "nombre='" + nombre + '\'' +
62             ", apellido='" + apellido + '\'' +
63             ", departamento='" + departamento + '\'' +
64             ", salario=" + salario +
65             '}';
66     }
67 }
```

## Clase Gerente

```
1 package com.codewithomarm.clase10.trabajadores;
2
3 public class Gerente {
4     //Declaración de atributos
5     private String nombre;
6     private String apellido;
7     private String departamento;
8     private double salario;
9
10    //Constructor con parámetros
11    public Gerente(String nombre, String apellido, String departamento, double salario) {
12        this.nombre = nombre;
13        this.apellido = apellido;
14        this.departamento = departamento;
15        this.salario = salario;
16    }
17
18    //Getter atributo nombre
19    public String getNombre() {
20        return nombre;
21    }
22
23    //Setter atributo nombre
24    public void setNombre(String nombre) {
25        this.nombre = nombre;
26    }
27
28    //Getter atributo apellido
29    public String getApellido() {
30        return apellido;
31    }
32
33    //Setter atributo apellido
34    public void setApellido(String apellido) {
35        this.apellido = apellido;
36    }
37
38    //Getter atributo departamento
39    public String getDepartamento() {
40        return departamento;
41    }
42
43    //Setter atributo departamento
44    public void setDepartamento(String departamento) {
45        this.departamento = departamento;
46    }
47
48    //Getter atributo salario
49    public double getSalario() {
50        return salario;
51    }
52
53    //Setter atributo salario
```

```

54 public void setSalario(double salario) {
55     this.salario = salario;
56 }
57
58 @Override
59 public String toString() {
60     return "Gerente{" +
61         "nombre='" + nombre + '\'' +
62         ", apellido='" + apellido + '\'' +
63         ", departamento='" + departamento + '\'' +
64         ", salario=" + salario +
65         '}';
66 }
67 }

```

Para calcular el bono correspondiente a cada empleado definimos una clase CalculoBono. Declaramos un constructor vacío por defecto. Declaramos un método calcularBono() sobrecargado 3 veces:

```

1 package com.codewithomarm.clase10.bono;
2
3 import com.codewithomarm.clase10.trabajadores.Agente;
4 import com.codewithomarm.clase10.trabajadores.Gerente;
5 import com.codewithomarm.clase10.trabajadores.Supervisor;
6
7 public class CalculoBono {
8     //Constructor por defecto
9     public CalculoBono() {
10     }
11
12     //Calcular el bono de un Agente
13     public double calcularBono(Agente agente){
14         return agente.getSalario() * 0.15;
15     }
16
17     //Calcular el bono de un Supervisor
18     public double calcularBono(Supervisor supervisor){
19         return supervisor.getSalario() * 0.30;
20     }
21
22     //Calcular el bono de un Gerente
23     public double calcularBono(Gerente gerente){
24         return gerente.getSalario() * 0.5;
25     }
26 }

```

Sobrecarga  
del método  
calcularBono()

En cada método se calcula el bono de cada empleado, basándose en el tipo de objeto que obtiene como argumento.

Si el objeto que obtiene como argumento es un objeto de tipo Agente, entonces calculará el bono de un agente. Si obtiene como argumento un objeto de tipo Supervisor, entonces calculará el bono de un supervisor. Y si obtiene como argumento un objeto de tipo Gerente, entonces calculará el bono de un Gerente.

Para probar nuestras clases definimos un método main en una clase Main. Aquí instanciaremos objetos de la clase CalculoBono. También declararemos e instanciaremos objetos de las clases Agente, Supervisor y Gerente con sus respectivos datos para sus atributos:

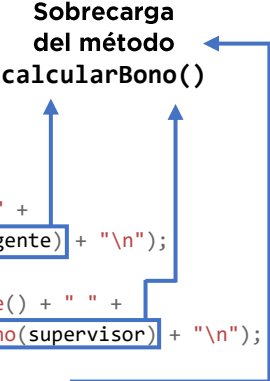
```
1 package com.codewithomarm.clase10.main;
2
3 import com.codewithomarm.clase10.bono.CalculoBono;
4 import com.codewithomarm.clase10.trabajadores.Agente;
5 import com.codewithomarm.clase10.trabajadores.Gerente;
6 import com.codewithomarm.clase10.trabajadores.Supervisor;
7
8 public class Main {
9     public static void main(String[] args) {
10         //Declaración e instanciación de un objeto CalculoBono
11         CalculoBono calculo = new CalculoBono();
12
13         //Declaración e instanciación de un objeto Agente
14         Agente agente = new Agente("Omar", "Montoya",
15             "Dell", 800.00);
16
17         //Declaración e instanciación de un objeto Supervisor
18         Supervisor supervisor = new Supervisor("John", "Mayer",
19             "Under Armour", 1500.00);
20
21         //Declaración e instanciación de un objeto Gerente
22         Gerente gerente = new Gerente("Anahi", "Puente",
23             "Amazon", 3000.00);
24     }
25 }
```

Ahora, hacemos el llamado del método calcularBono() del objeto calculo, para cada uno de los objetos de empleado:

```
1 package com.codewithomarm.clase10.main;
2
3 import com.codewithomarm.clase10.bono.CalculoBono;
4 import com.codewithomarm.clase10.trabajadores.Agente;
5 import com.codewithomarm.clase10.trabajadores.Gerente;
6 import com.codewithomarm.clase10.trabajadores.Supervisor;
7
8 public class Main {
```

```
9 public static void main(String[] args) {
10     //Declaración e instanciación de un objeto CalculoBono
11     CalculoBono calculo = new CalculoBono();
12
13     //Declaración e instanciación de un objeto Agente
14     Agente agente = new Agente("Omar", "Montoya",
15         "Dell", 800.00);
16
17     //Declaración e instanciación de un objeto Supervisor
18     Supervisor supervisor = new Supervisor("John", "Mayer",
19         "Under Armour", 1500.00);
20
21     //Declaración e instanciación de un objeto Gerente
22     Gerente gerente = new Gerente("Anahi", "Puente",
23         "Amazon", 3000.00);
24
25     System.out.println("El bono del agente " + agente.getNombre() + " " +
26         agente.getApellido() + " es de $" + calculo.calcularBono(agente) + "\n");
27
28     System.out.println("El bono del supervisor " + supervisor.getNombre() + " " +
29         supervisor.getApellido() + " es de $" + calculo.calcularBono(supervisor) + "\n");
30
31     System.out.println("El bono del gerente " + gerente.getNombre() + " " +
32         gerente.getApellido() + " es de $" + calculo.calcularBono(gerente));
33 }
34 }
```

Sobrecarga del método calcularBono()



### Output por consola

```
"C:\Program Files\Java\jdk-21\bin\java.exe" [...]  
El bono del agente Omar Montoya es de $120.0  
  
El bono del supervisor John Mayer es de $450.0  
  
El bono del gerente Anahi Puente es de $1500.0  
  
Process finished with exit code 0
```

Podemos notar que efectivamente el método calcularBono calcula los diferentes bonos para cada empleado en base al tipo de objeto que recibe como argumento.