Tabla de Contenidos

- Java Metodo de instancia
  - Metodos de instancia y su uso
- Static Method vs instance method
  - El principio DRY
- Static Method vs instance method delegación
  - Métodos estáticos y mantenimiento de estado
- Metodos Estáticos resumen
- Metodos de instancia resumen
  - Otros artículos relacionados

La pregunta de static method vs instance method es muy muy común en el mundo de la programación orientada a objeto no tanto por cómo diferenciarlos que también sino sobre cuál es su uso correcto y cuando debemos utilizar uno u otro método . Vamos a construir un ejemplo sencillo que nos ayude a entenderlo . Para ello vamos a empezar creando una clase de Factura muy sencilla que contenga un instance method o método de instancia que nos calcule el IVA de la Factura que recordemos que ya es un 21 % ,vamos con ello :

CURSO SPRING BOOT

GRATIS

APUNTATE!!


```
package com.arquitecturajava;

public class Factura {

	private int numero;
```

```
private String concepto;
private double importe;
public int getNumero() {
        return numero;
}
public void setNumero(int numero) {
        this.numero = numero;
}
public String getConcepto() {
        return concepto;
}
public void setConcepto(String concepto) {
        this.concepto = concepto;
}
public double getImporte() {
        return importe;
}
public void setImporte(double importe) {
        this.importe = importe;
}
public Factura(int numero, String concepto, double importe) {
        super();
        this.numero = numero;
        this.concepto = concepto;
        this.importe = importe;
}

public double getImporteIVA() {

        return getImporte()*1.21;
}
```

```
}
```

```
package com.arquitecturajava;

public class Factura {

	private int numero;
	private String concepto;
	private double importe;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
	}
	public String getConcepto() {
		return concepto;
	}
	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}
	public double getImporte() {
		return importe;
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}
	public Factura(int numero, String concepto, double importe) {
		super();
		this.numero = numero;
		this.concepto = concepto;
		this.importe = importe;
```

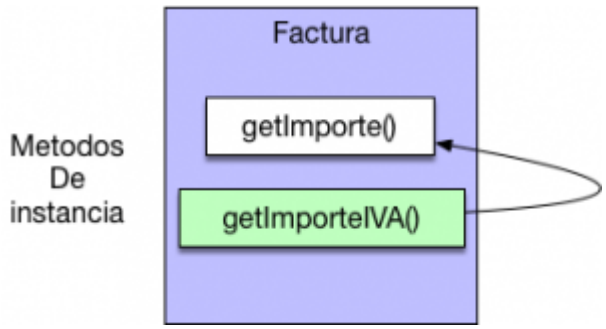
```
	}

	public double getImporteIVA() {

		return getImporte()*1.21;
	}
}
```

Java Metodo de instancia

Se trata de un ejemplo muy sencillo la Factura contiene sus propiedades set/get y su constructor . Hemos añadido un método que nos calcula el IVA .



Se trata de un método de instancia (instance method) . Un método de instancia esta relacionado con el estado que almacena un objeto que creamos a partir de una clase. En este caso queremos acceder al importe y calcular su IVA. Por lo tanto nosotros en nuestro programa main construimos una factura y calculamos su importe con el IVA usando el método de instancia.

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {

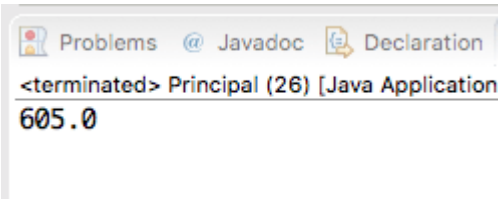
		Factura f= new Factura(1, "ordenador", 500);
```

```
                System.out.println(f.getImporteIVA());

        }

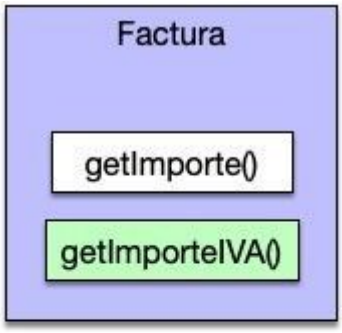
}
```

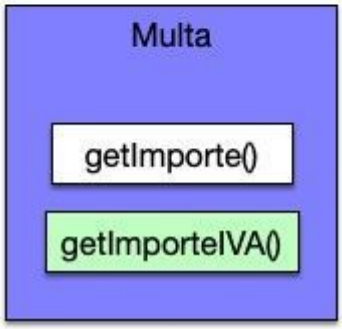
El resultado le veremos aparecer en la consola :



Metodos de instancia y su uso

¿Es correcto el código qué hemos construido? . Esta es una pregunta que parece trivial en un punto de partida . Claro que es correcto tenemos la factura y calculamos su importe con IVA a través de un método de instancia . Por lo tanto SI es totalmente correcto , pero ese SI tiene una puntualización es correcto “ahora” ,es correcto en este momento con este código . Imaginemonos que a futuro nuestro proyecto incluye otro concepto el concepto de multa.





Una multa tiene también un importe y un IVA que aplicar. Vamos a ver la clase:

```
package com.arquitecturajava;

public class Multa {

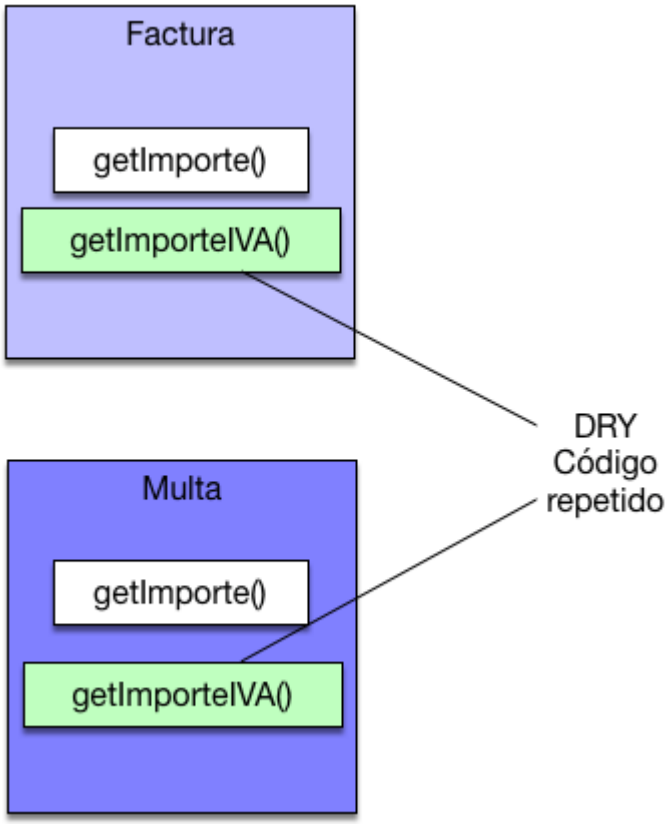
	private String tipoInfraccion;
	private double importe;
	public String getTipoInfraccion() {
		return tipoInfraccion;
	}
	public void setTipoInfraccion(String tipoInfraccion) {
		this.tipoInfraccion = tipoInfraccion;
	}
	public double getImporte() {
		return importe;
```

```
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}
	public Multa(String tipoInfraccion, double importe) {
		super();
		this.tipoInfraccion = tipoInfraccion;
		this.importe = importe;
	}

	public double getImporteIVA() {

		return getImporte()* 1.21;
	}
}
```

Ok todo funciona correctamente y el comportamiento es correcto . Sin embargo tenemos un problema que podemos ver de una forma muy clara con el nuevo código.



Tenemos código repetido y no estamos cumpliendo con uno de los principios de ingeniería de software más clásicos el principio DRY (Dont Repeat YourSelf) En este caso el método getImporteConIVA(). Así que tenemos un problema y necesitamos abordar una solución para él.

Static Method vs instance method

Es aquí donde nos podemos dar cuenta que para calcular el IVA de algo no necesitamos construir un objeto previamente sino que nos es suficiente con disponer del importe inicial. Por lo tanto para nuestras dos clases podría ser útil disponer de una clase adicional que se denomine por ejemplo CalculosFinancieros y que tenga un método que reciba un importe y nos calcule el importe con IVA .Este método será estático porque no necesita acceder a ningún estado concreto de ningún objeto, así de sencillo . Por lo tanto vamos a construirla y ver como nos puede ayudar en nuestro programa:

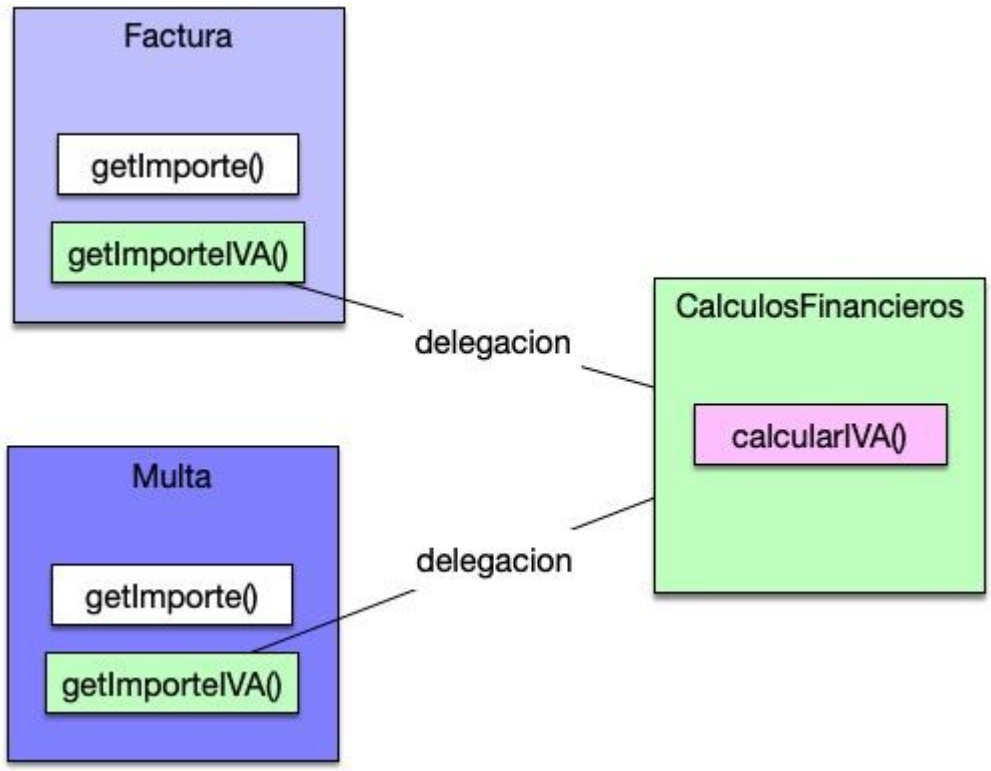
```
package com.arquitecturajava;

public class CalculosFinancieros {

        public static double calcularIVA(double importe) {

                return importe* 1.21;
        }
}
```

Esta nueva clase nos permitirá realizar el calculo del IVA sin disponer de código repetido tanto en Factura como en Multa para ello usaremos el concepto de delegación y haremos que ambas clases deleguen la funcionalidad en ella .



Vamos a ver el nuevo código:

```
package com.arquitecturajava;

public class Multa {

	private String tipoInfraccion;
	private double importe;
	public String getTipoInfraccion() {
		return tipoInfraccion;
	}
	public void setTipoInfraccion(String tipoInfraccion) {
		this.tipoInfraccion = tipoInfraccion;
	}
	public double getImporte() {
		return importe;
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}
	public Multa(String tipoInfraccion, double importe) {
		super();
		this.tipoInfraccion = tipoInfraccion;
		this.importe = importe;
	}

	public double getImporteIVA() {

		//return getImporte()* 1.21;
		return
CalculosFinancieros.calcularIVA(this.getImporte());
	}
}
```

```
package com.arquitecturajava;

public class Factura {

	private int numero;
	private String concepto;
	private double importe;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
	}
	public String getConcepto() {
		return concepto;
	}
	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}
	public double getImporte() {
		return importe;
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}
	public Factura(int numero, String concepto, double importe) {
		super();
		this.numero = numero;
		this.concepto = concepto;
		this.importe = importe;
	}
```

```
        public double getImporteIVA() {

                //return getImporte()*1.21;
                return
CalculosFinancieros.calcularIVA(this.getImporte());
        }
}
```

El principio DRY

Ahora ya no tenemos código repetido ya la funcionalidad de calcular el importe con IVA se encuentra aislada en la clase Cálculos financieros en el método calcularIVA.

```
public static double calcularIVA(double importe) {

                return importe* 1.21;
        }
```

Static Method vs instance method delegación

Este método no necesita que se construya ningún objeto en concreto para funcionar de forma correcta y por lo tanto es un método estatico (static method). Por el otro lado la clase Factura y la clase Multa disponen de un método de instancia cada una de ellas que delega en este método estático.

```
public double getImporteIVA() {

                return
CalculosFinancieros.calcularIVA(this.getImporte());
        }
```

Estos métodos si que deben ser de instancia ya que necesitan acceder al importe de cada uno de nuestros objetos Factura o Multa para luego delegar en el método estático.Cuando

hablamos de static method vs instance method no estamos diciendo en ningún momento que uno de estos métodos sea mejor que el otro . Sino que cada uno de ellos tiene una utilidad concreta y el mayor partido se saca cuando somos capaces de combinar ambos conceptos en una solución que aporte sencillez a nuestro código.

Métodos estáticos y mantenimiento de estado

Una de las ideas fundamentales a la hora de usar métodos estáticos es que estos no gestionan ningún estado perteneciente a un objeto y por eso parecen sencillos de localizar. Sin embargo hay que puntualizar que los métodos estáticos tienen también otra limitación y es que no se heredan y por lo tanto no pueden ejecutarse de forma polimórfica tengamos en cuanta esta limitación también a la hora de definirlos .

Esto nos puede parecer en un primer momento curioso o raro. Sin embargo es cierto los métodos estáticos pertenecen a cada clase y se invocan siempre utilizando el nombre de la clase seguido del operador “.”

```
CalculosFinancieros.calcularIVA();
```

Si tuvieramos una clase más especializada que fuera CalculosFinancierosAvanzados los métodos estáticos no serian heredados al usar extends. Esta clase tendría sus propios métodos estáticos que se accederían de la siguiente forma.

```
CalculosFinancierosAvanzados.nuevoCalculo();
```

Estas son al final las diferencias entre static method vs instance method espero que el ejemplo ayude a ver las cosas más claras :). El api de Java dispone de varios ejemplos útiles en ubicaciones como la clase Math o las clases Wrapper del lenguaje como Integer o Double.

Metodos Estáticos resumen

Los métodos estáticos en Java tienen varias ventajas que los hacen útiles en situaciones específicas.

1. No necesitan objetos: Puedes llamarlos usando solo el nombre de la clase, sin crear una instancia.
2. Más eficientes: Al no requerir una instancia, ahorras memoria y recursos, lo que es ideal para tareas comunes o utilidades.
3. Perfectos para funciones generales: Se usan mucho en funciones que no dependen del estado de un objeto, como operaciones matemáticas (Math.sqrt()) o conversiones (Integer.parseInt()).
4. Fáciles de usar desde cualquier parte: Puedes llamarlos desde cualquier clase sin preocuparte por pasar instancias.
   Son ideales cuando no necesitas datos de un objeto específico, pero no sirven para todo, ya que no pueden acceder a variables o métodos no estáticos.

Metodos de instancia resumen

Los métodos de instancia en Java también tienen ventajas clave, especialmente cuando se trabaja con objetos y sus datos. Aquí te explico de forma simple las principales ventajas:

1. Acceso a los datos del objeto: Un método de instancia puede acceder y modificar las variables de instancia (atributos) del objeto al que pertenece, permitiendo que el comportamiento dependa del estado específico del objeto.
2. Soporte para el polimorfismo: Los métodos de instancia permiten aprovechar el polimorfismo, lo que significa que el comportamiento de un método puede variar según la clase específica del objeto que lo llama.
3. Facilitan la encapsulación: Al trabajar con métodos de instancia, puedes encapsular el estado y el comportamiento en el mismo objeto, protegiendo los datos mediante el control de acceso y manteniendo la lógica dentro del objeto.

4. Permiten la sobrecarga de métodos: Los métodos de instancia pueden ser sobrecargados en la misma clase o en clases hijas, lo que permite que un método realice diferentes tareas según los parámetros.
5. Interacción con otros métodos y atributos: Un método de instancia puede llamar a otros métodos no estáticos y acceder a todos los atributos del objeto, lo que le da más flexibilidad para implementar lógica compleja que depende del estado del objeto.
   En resumen, los métodos de instancia son esenciales cuando necesitas que el comportamiento del método dependa de los datos del objeto, y cuando quieres aprovechar características avanzadas como el polimorfismo y la encapsulación.

CURSO SPRING BOOT
GRATIS
APUNTATE!!

Otros artículos relacionados

- ¿Qué es el Patron Factory?
- Java interface private method y como usarlo
- Java Static Method y como usarlo correctamente
- Utilizando un Java reference method

Artículos externos

1. Oracle métodos estáticos