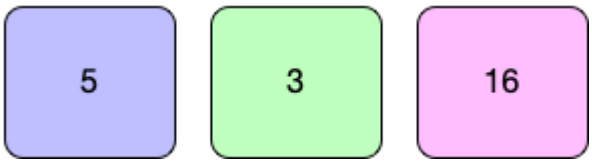
El versionado semántico hace referencia a cómo generar los diferentes números de release para nuestro software . Es decir nosotros tenemos que versionar los programas o librerías que construimos de igual manera que Spring o Hibernate versionan su software. Cuando nos encontramos con una versión de Spring Framework tenemos muy claro que versión deseamos utilizar. Por ejemplo una de las últimas versiones de Spring es 5.3.16 .

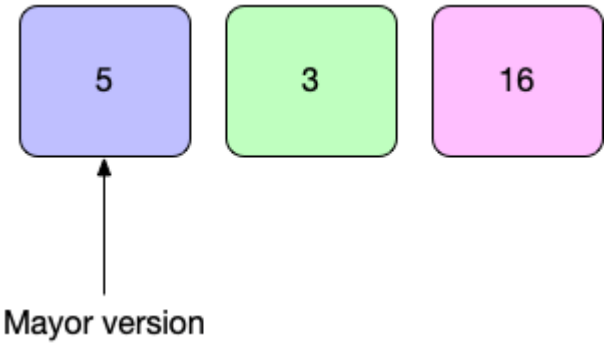


Esto en principio puede que no te diga nada y simplemente lo consideres un numero . Sin embargo esta versión de Spring Framework se apoya en lo que comúnmente se denomina versionado semántico. Vamos a hablar de ello a detalle:

Mayor Version (5.x.y)

Cuando estamos hablando del versionado semántico hablamos de tres números que podemos usar . El primer número de izquierda a derecha es en este caso el 5 . Este número se conoce comúnmente como mayor versión y es el encargado de definir la versión mayor de nuestro software.

En estos momentos en el caso de Spring Framework estamos en la versión 5. Cuando Spring Framework avance a la versión 6 estará dando un salto importante y estará enviando información al desarrollador de que la versión 6 no tiene porque ser compatible con la versión 5 que ha estado usando hasta ahora . Es decir los interfaces de comunicación han cambiado y los clientes se pueden ver afectados. P



Para entenderlo mejor si dispusieramos de una función de sumar en una librería con el siguiente código:

```
package com.arquitecturajava.main;

public class Calculadora {

	public double sumar(double a , double b) {
		return a+b;
	}
}
```

El cambio de mayor versión podría suponer un cambio a nivel de métodos que a nivel cliente se utilizan . Por ejemplo en nuestro caso cambiar el nombre del método.

```
package com.arquitecturajava.main;

public class Calculadora {

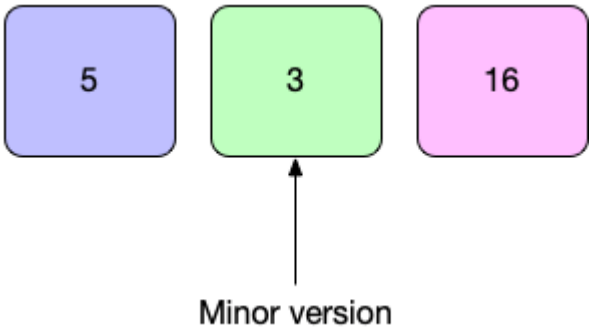
	public double sum(double a , double b) {
		return a+b;
	}
}
```

Este cambio hace incompatible la versión de nuestra calculadora con la nueva versión por lo

tanto si la Calculadora estaba en la versión 1.0.0 pasará a la versión 2.0.0

Minor Version (x.3.y) y versionado semántico

Un cambio de Minor Versión es cuando hacemos cambios que mantienen la compatibilidad del API ya sea añadiendo métodos ya sea cambiando o ampliando la funcionalidad existente.



Por ejemplo en este caso podríamos realizar cualquiera de los dos cambios y deberíamos saltar a la versión 2.1.0

```
package com.arquitecturajava.main;

public class Calculadora {

	public double sum(double a, double b) {

		return a + b;
	}

	public double substract(double a, double b) {

		return a - b;
	}
}
```

En este caso no hemos cambiado el método sum sino que hemos añadido un nuevo método que aumenta la funcionalidad pero no cambia el API existente.

```
package com.arquitecturajava.main;

public class Calculadora {

	public double sum(double a, double b) {

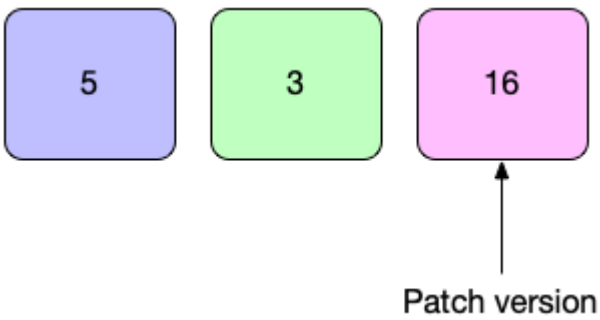
		return Double.sum(a, b);
	}

}
```

Este cambio también se abordará como un Minor Versión estamos modificando el código existente para simplificarlo o refactorizarlo pero no estamos cambiando el interface que el programa cliente va a usar.

Patch Version (x.y.16)

En este caso estamos hablando de un Patch o parche que soluciona problemas existentes , temas de rendimiento etc que permiten un funcionamiento correcto del programa.



Por ejemplo supongamos que tenemos el programa con tres métodos:

```
package com.arquitecturajava.main;
```

```
public class Calculadora {

	public double sum(double a, double b) {

		return a + b;
	}

	public double substract(double a, double b) {

		return a - b;
	}
	public double multiply(double a, double b) {

		return a * b;
	}

}
```

Hemos avanzado en la versión y estamos con la 2.2.0 . Ahora un desarrollador se percata de que sería más óptimo cambiar el método de multiplicar por :

```
public double multiply(double a, double b) {

		if (a==0 || b==0) return 0;
		return a * b;
	}
```

Conclusiones Versionado Semántico

Se trata de una simple mejora por lo tanto estamos parcheando nuestro código y la versión saltará a 2.2.1 . Estos son los conceptos fundamentales que tenemos que conocer a nivel de versionado semántico a la hora de defendernos.

Otros artículos relacionados

- Git Branch Java y Eclipse
- Eclipse GIT commit y como crear un repositorio
- Eclipse Git , Repositorios locales y remotos
- Curso Introducción GIT