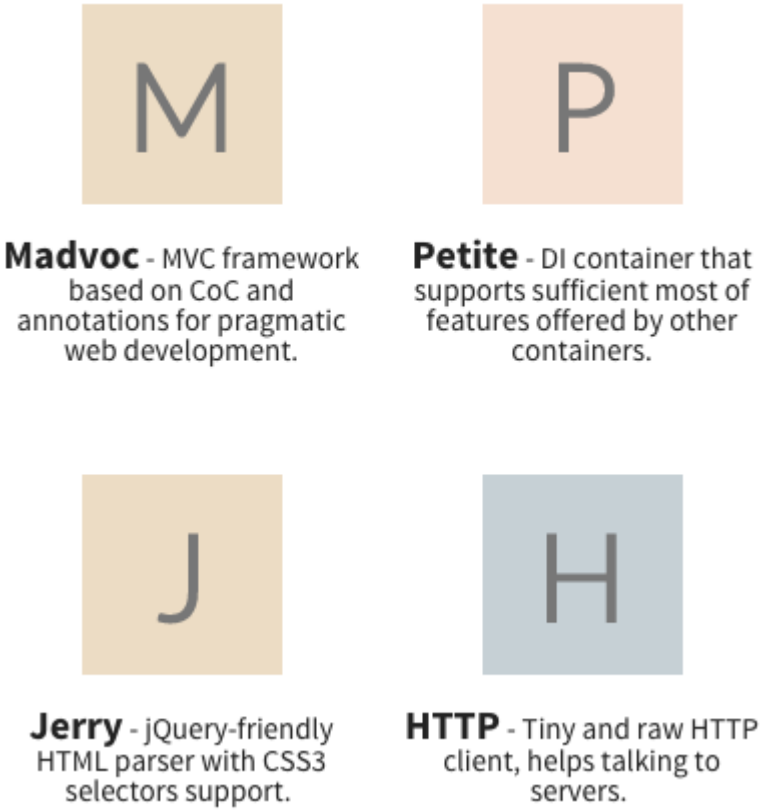
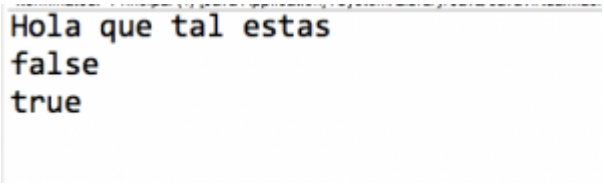
Java Jodd MicroFramework es un conjunto de frameworks y utilidades que en muchos proyectos pueden realizar un buen aporte. Una de las características que más destaca es que solo ocupa 1.5 megas. Algo que hoy es cada vez más importante por la necesidad de modularidad.



Java Jodd Microframework esta bien diseñado y aporta clases que añaden funcionalidad al JDK en varias areas que son interesantes. Por ejemplo dispone de una clase StringUtils que potencia la funcionalidad de la clase String.

```
package com.arquitecturajava;

import jodd.util.StringUtil;

public class Principal {

public static void main(String[] args) {

String cadena="hola que tal estas";

System.out.println(StringUtil.capitalize(cadena));
String cadena2=" ";
System.out.println(cadena2.isEmpty());
System.out.println(StringUtil.isBlank(cadena2));

}

}
```

En este ejemplo se han utilizado dos métodos de StringUtils, capitalize() que nos pasa a mayúscula la primera letra de un texto, por otro lado isBlank() que comprueba si una cadena contiene únicamente espacios en blanco. El resultado por consola del código es :



Otra clase útil y que simplifica el uso de expresiones regulares es WildCard que permite comparar una cadena con los clásicos WildCards de sistema operativo.

```
package com.arquitecturajava;

import jodd.util.Wildcard;

public class Principal2 {

public static void main(String[] args) {

System.out.println(Wildcard.match("hola", "h??a"));

};

}
```

Java Jodd MicroFrameworks

Estas utilidades están muy bien pero en lo que realmente destaca Jood , es en su conjunto de microframeworks que permiten realizar tareas complejas con poco esfuerzo. El siguiente ejemplo muestra las capacidades de inyección de dependencias que tiene Jood a través del Micro framework Petite.

```
package com.arquitecturajava;

public class Motor {

        public void arrancar() {

                System.out.println("arranca");
        }
```

```
}
```

```
package com.arquitecturajava;

import jodd.petite.meta.PetiteBean;
import jodd.petite.meta.PetiteInject;

@PetiteBean
public class Coche {

        private String marca;

        @PetiteInject("motor")
        private Motor motor;

        public String getMarca() {
                return marca;
        }

        public void setMarca(String marca) {
                this.marca = marca;
        }

        public Motor getMotor() {
                return motor;
        }

        public void setMotor(Motor motor) {
                this.motor = motor;
```

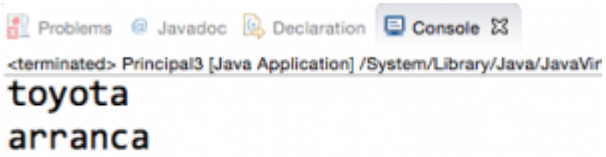
```
	}

	public void arrancar() {

		motor.arrancar();
	}
}
```

En el código se pueden observar dos clases y como el uso de anotaciones permite la inyección de dependencia entre ambas. A continuación queda mostrar el programa Main.

```
package com.arquitecturajava;

import jodd.petite.PetiteContainer;

public class Principal3 {

	public static void main(String[] args) {

		PetiteContainer contenedor = new PetiteContainer();

		contenedor.registerPetiteBean(Coche.class, null, null,
null, false);
		contenedor.registerPetiteBean(Motor.class, null, null,
null, false);

		Coche coche = (Coche) contenedor.getBean("coche");
		coche.setMarca("toyota");

		System.out.println(coche.getMarca());
```

```
                coche.arrancar();
        }

}
```

Al invocar al método arrancar() de Coche , se invocará el método arrancar() del Motor que ya sido inyectado a través del framework.



Otros artículos relacionados: Spark Web Microframework , Introducción a Spring MVC