El uso de Java Assert y sus variantes es el punto de entrada al desarrollo con Test Driven Development. Ahora bien existen varias librerías que nos permiten modificar la forma en la cual trabajamos con los java assert . Vamos a ver varios ejemplos que nos pueden ayudar a clarificar cosas . Supongamos que tenemos la clase Ordenador.

```
package com.arquitecturajava;

public class Ordenador {

	private String modelo;
	private int memoria;
	private double precio;

	public String getModelo() {
		return modelo;
	}

	public void setModelo(String modelo) {
		this.modelo = modelo;
	}

	public int getMemoria() {
		return memoria;
	}

	public void setMemoria(int memoria) {
		this.memoria = memoria;
	}
```

```
	public double getPrecio() {
		return precio;
	}

	public void setPrecio(double precio) {
		this.precio = precio;
	}

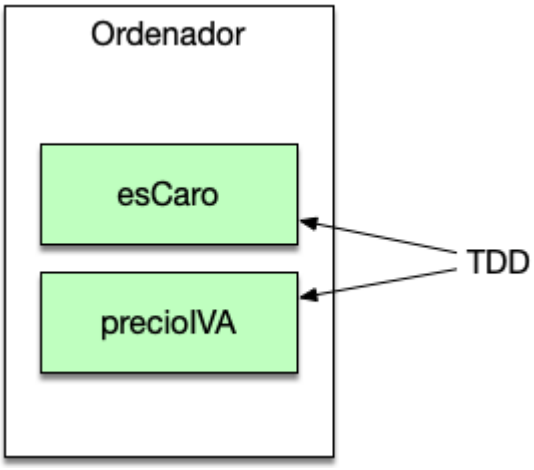
	public Ordenador(String modelo, int memoria, double precio) {
		super();
		this.modelo = modelo;
		this.memoria = memoria;
		this.precio = precio;
	}

	public boolean esCaro() {

		if (this.getPrecio() &gt; 1500) {

			return true;
		} else {
			return false;
		}
	}
	public double getPrecioIVA() {
		return this.getPrecio()*1.21;
	}
}
```

Esta clase dispone de dos métodos fundamentales a testear el primero es si el ordenador es caro y el segundo es el precio con IVA del ordenador.



Lo más sencillo en este caso es usar JUnit a través de dependencias de Maven

```
        <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-api</artifactId>
                        <version>5.3.2</version>
                        <scope>test</scope>
                </dependency>

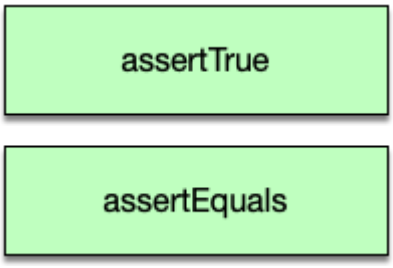
                <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-engine</artifactId>
                        <version>5.3.2</version>
                        <scope>test</scope>
                </dependency>
                <dependency>
                        <groupId>org.junit.platform</groupId>
                        <artifactId>junit-platform-
launcher</artifactId>
```

```
            <version>1.3.2</version>
            <scope>test</scope>
        </dependency>
```

Generando los Test adecuados.

```
@Test
void testOrdenadorEsCaro() {
        Ordenador ordenador= new Ordenador("imac",16,2500);
        assertTrue(ordenador.esCaro());
}
@Test
void testPrecioIvaDelOrdenador() {
        Ordenador ordenador= new Ordenador("macmini",8,1300);
        assertEquals(1573,ordenador.getPrecioIVA(),0.001);
}
```

En este caso hemos usado los métodos assertTrue y assertEquals de JUnit para construir nuestros ejemplos de java assert . Ahora bien no a todo el mundo le gusta este enfoque ya que se encuentra obligado a estar muy pendiente de los tipos de assert que se aplican y es cierto que a veces cuesta leer el código.

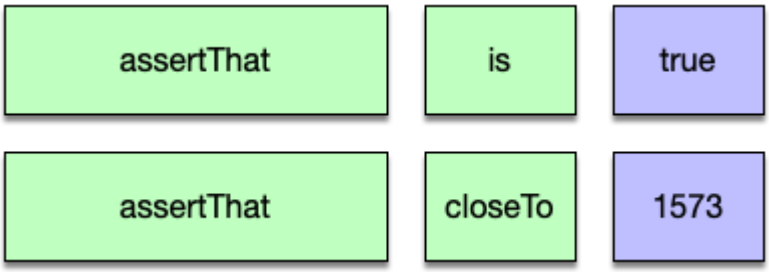


HamCrest y java Assert

En muchos casos prefieren usar el enfoque que aporta hamCrest con sus machers ya que permite una lectura mas natural del código a través del uso de import estáticos.

```
	@Test
	void testOrdenadorEsCaroHamCrest() {
		Ordenador ordenador= new Ordenador("imac",16,2500);
		assertThat(ordenador.esCaro(),is(true));
	}
	@Test
	void testPrecioIvaDelOrdenadorHamCrest() {
		Ordenador ordenador= new Ordenador("macmini",8,1300);
assertThat(ordenador.getPrecioIVA(),closeTo(1573,0.01));
	}
```

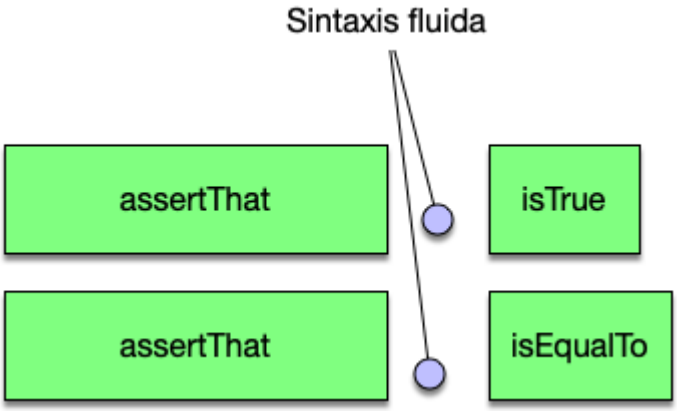
En este caso usamos assertThat como instrucción principal y una serie de matchers que se definen al final del assert . Esto nos permite una lectura más natural del código.



Lo cual es una ventaja a la hora de construir nuestros java assert . Estas son las opciones más habituales . Sin embargo existe una tercera vía ligada a lo que se denomina la programación fluida.

AssertJ y Java Assert

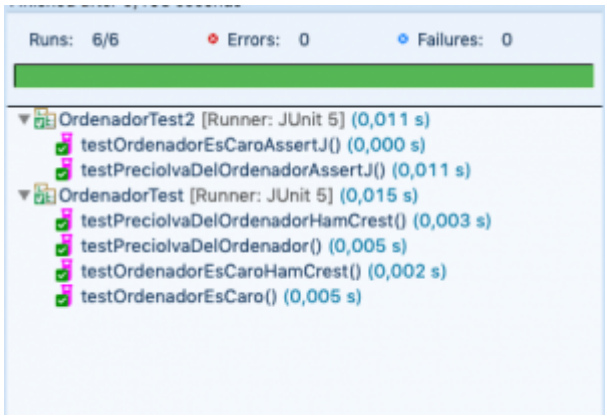
Podemos utilizar AssertJ una librería que nos permitirá realizar los assert de una forma mucho más integrada con el código y soportando sintaxis fluida e intellisense por parte del IDE de desarrollo.



Veamoslo :

```
@Test
void testOrdenadorEsCaroAssertJ() {
        Ordenador ordenador= new Ordenador("imac",16,2500);
        assertThat(ordenador.esCaro()).isTrue();
}
@Test
void testPrecioIvaDelOrdenadorAssertJ() {
        Ordenador ordenador= new Ordenador("macmini",8,1300);
        assertThat(ordenador.getPrecioIVA()).isEqualTo(1573);
}
```

Todos los métodos se ejecutarán sin problemas :



Como siempre para gustos los colores. Podemos usar cualquiera de las opciones pero normalmente las dos últimas son las que mayor consenso generan aunque nada nos impide combinar unas opciones con otras.

1. Java Mockito y los Mock Object
2. Java y fluid interface
3. Spring REST Test utilizando Rest Assured