Utilizar un Java Fluent Interface suele ser muy práctico pero lamentablemente no todas las APIs lo soportan. Recordemos que los Fluent Interfaces o interfaces fluidos nos permiten trabajar de una forma más cómoda cuando programamos. Vamos a ver un ejemplo de una clase Java que no soporta un enfoque fluido y como podemos adaptarla para que lo soporte . La clase que vamos a utilizar es la de Properties vamos a ver un ejemplo :


```
package com.arquitecturajava.fluent;

import java.util.Properties;

public class ProgramaMain {

public static void main(String[] args) {
 Properties p = new Properties();

p.setProperty("correo", "prueba@prueba.com");
 p.setProperty("titulo", "hola");
 p.setProperty("mensaje", "hola que tal");

System.out.println(p.getProperty("correo"));
 System.out.println(p.getProperty("titulo"));
 System.out.println(p.getProperty("mensaje"));
```

```
}

}
```

Se trata de una clase que contiene un HashMap para almacenar pares claves, valor. En muchas ocasiones se usa en Java alguna Fachada para limitar las claves y valores a los que nosotros necesitamos. Un ejemplo suele ser una clase que parametriza la conexión a la base de datos .Nosotros vamos a hacer algo similar pero parametrizando un correo.

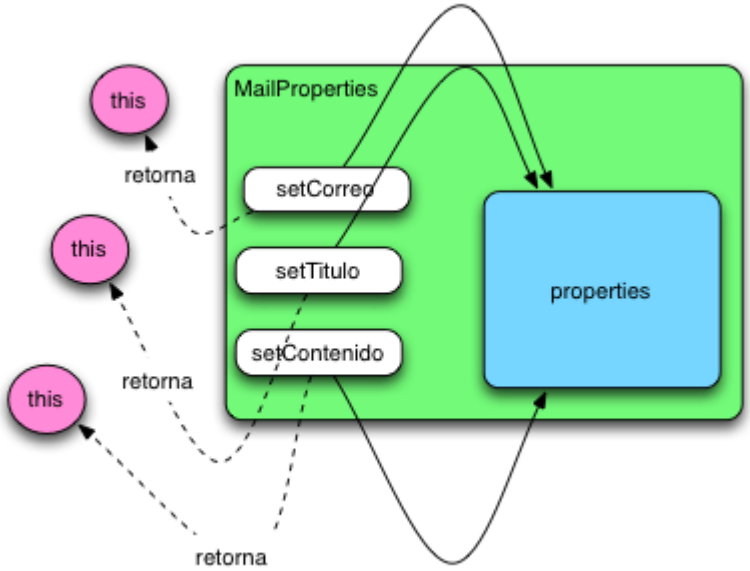
La diferencia va a ser que lo diseñaremos utilizando interfaces fluidos.

```
package com.arquitecturajava.fluent;

import java.util.Properties;

public class MailProperties {

 private Properties propiedades = new Properties();

 public MailProperties setCorreo(String valor) {
```

```
 propiedades.setProperty("correo", valor);
 return this;
 }

 public String getCorreo() {
 return propiedades.getProperty("correo");
 }

 public MailProperties setTitulo(String valor) {

 propiedades.setProperty("titulo", valor);
 return this;
 }

 public String getTitulo() {
 return propiedades.getProperty("titulo");
 }

 public MailProperties setMensaje(String valor) {

 propiedades.setProperty("mensaje", valor);
 return this;
 }

 public String getMensaje() {
 return propiedades.getProperty("mensaje");
 }

}
```

Como podemos observar cada una de las propiedades cuando asigna valor devuelve una

referencia al propio objeto utilizando this.



El resultado es que es mucho más natural trabajar con la clase

```
package com.arquitecturajava.fluent;

public class ProgramaMain2 {

 public static void main(String[] args) {
 MailProperties p = new MailProperties();

 p.setCorreo("prueba@prueba.com").setTitulo("hola").setMensaje("hola
que tal");
 System.out.println(p.getCorreo());
 System.out.println(p.getTitulo());
 System.out.println(p.getMensaje());

 }
```

```
}
```

Los Java Fluent Interface nos simplifican de forma importante el trabajo con clases de todo tipo pero quizás las clases de objetos de negocio y los parametrizadores son los que más se benefician de este enfoque.

Otros artículos relacionados:

Java Singleton

Java this vs this

Java Lambda