Tabla de Contenidos

- Filtrando Personas
- java == null vs Objects.isNull
- Otros artículos relacionados

java == null vs Objects.isNull es una pregunta bastante habitual a partir de Java 8 . Todos estamos bastante acostumbrados a comprobar si un objeto es nulo con el operador == o con el operador != . Java 8 añade una opción adicional que nos permite comprobar si un objeto es null con la clase de utilidad Objects.isNull. En un primer momento nos puede parecer extraño y no entendemos el porqué de su uso pero tiene su razón de ser . Vamos a verlo con un ejemplo sencillo y una lista de Personas de las cuales imprimimos su nombre en el caso de que el objeto no sea nulo.

```
package com.arquitecturajava;

public class Persona {

	private String nombre;

	public Persona(String nombre) {
		super();
		this.nombre = nombre;
	}

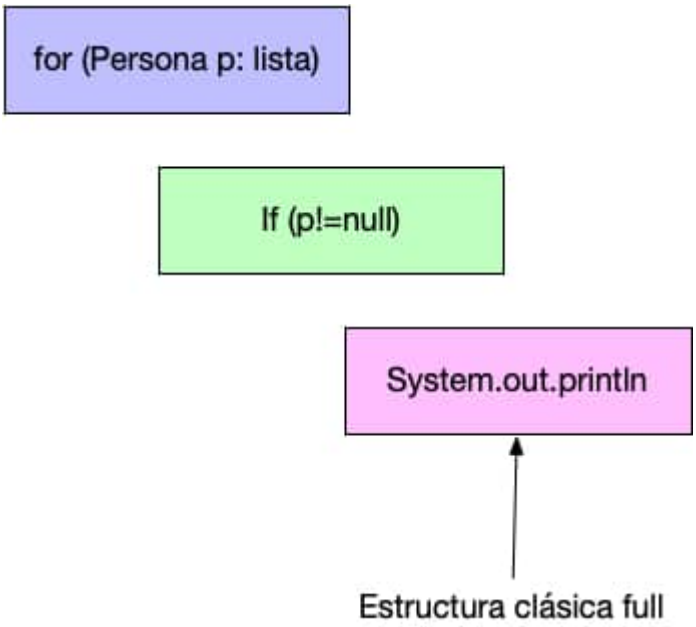
	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
```

```
        }
}
```

Filtrando Personas

La clase Persona únicamente incluye el nombre . Es momento de generar una lista que incluya valores nulos para varios objetos Persona y recorrerla para eliminar estas situaciones apoyándonos con el operador != en cuando a valores null se refiere.



Veamos el código:

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

public class Principal {

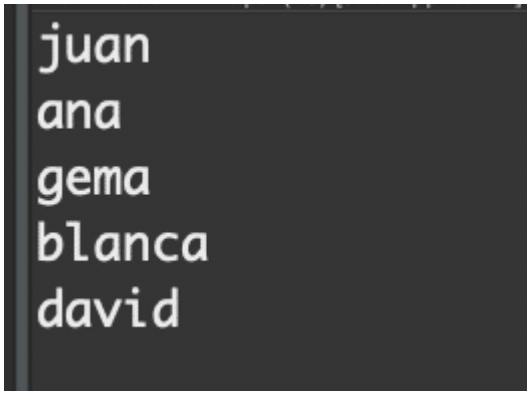
        public static void main(String[] args) {
                List<Persona> lista= new ArrayList<Persona>();
```

```
                lista.add(new Persona("juan"));
                lista.add(new Persona("ana"));
                lista.add(new Persona("gema"));
                lista.add(null);
                lista.add(new Persona("blanca"));
                lista.add(null);
                lista.add(new Persona("david"));
                for (Persona p:lista) {
                        if (p!=null) {
                                System.out.println(p.getNombre());
                        }
                }

        }

}
```

En este caso el código es sencillo y el resultado por la consola el esperado:



Simplemente hemos usado un bucle for con una sentencia if para filtrar.

java == null vs Objects.isNull

¿Podemos realizar otro enfoque con Java 8? . La realidad es que sí , pero el código es muy similar al anterior y utilizaríamos la clase de utilidad Objects.

```
package com.arquitecturajava;

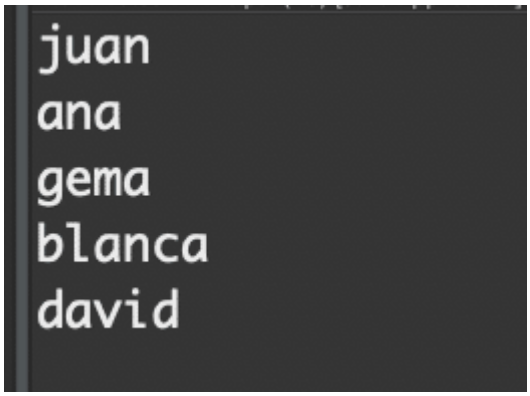
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Principal2 {

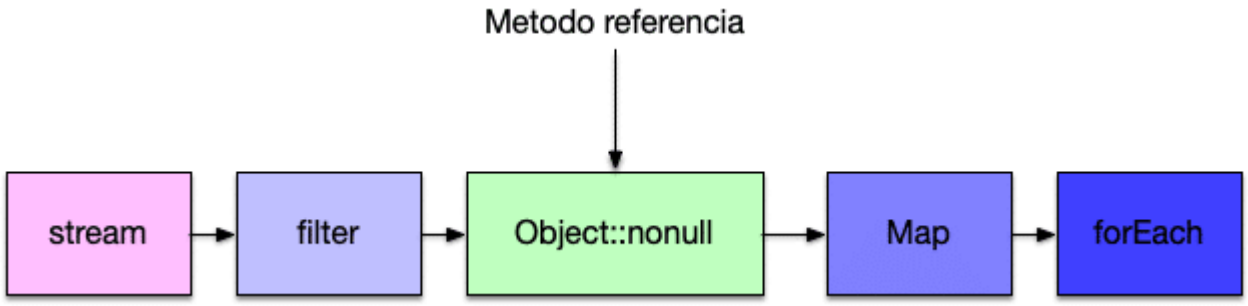
	public static void main(String[] args) {
		List<Persona> lista= new ArrayList<Persona>();
		lista.add(new Persona("juan"));
		lista.add(new Persona("ana"));
		lista.add(new Persona("gema"));
		lista.add(null);
		lista.add(new Persona("blanca"));
		lista.add(null);
		lista.add(new Persona("david"));
		for (Persona p:lista) {
			if (Objects.nonNull(p)) {
				System.out.println(p.getNombre());
			}
		}
	}

}
```

A todo el mundo le suele parecer un sin sentido la nueva versión del código ya que es prácticamente idéntica a la anterior y el resultado no varía.



¿Hemos ganado algo con la nueva versión ?, la realidad es que no lo parece. ¿Entonces porque la inclusión de esta nueva funcionalidad a nivel de Java 8 ¿Qué sentido tiene?. Esta funcionalidad aporta valor cuando trabajamos con programación funcional y Streams y queremos eliminar de una lista los valores nulos.




```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Principal3 {
```

```
    public static void main(String[] args) {
            List<Persona> lista= new ArrayList<Persona>();
            lista.add(new Persona("juan"));
            lista.add(new Persona("ana"));
            lista.add(new Persona("gema"));
            lista.add(null);
            lista.add(new Persona("blanca"));
            lista.add(null);
            lista.add(new Persona("david"));
            lista.stream()
            .filter(Objects::nonNull)
            .map(Persona::getNombre)
            .forEach(System.out::println);
    }

}
```

Como se puede observar nos apoyamos en un método de referencia y apuntamos al método que se encuentra en la clase Objects a nivel de utilidades. De esta manera habremos conseguido realizar el mismo filtrado de una forma más elegante usando programación funcional y apoyándonos en el método Objects.nonNull.

Otros artículos relacionados

- Java Stream for loop y programación funcional
- Java 8 Lambda y forEach (II)
- Java Stream Filter y Predicates
- JDK 8