

El uso de Java 8 Optional es cada día más común para todos los que desarrollamos sobre Java 8. ¿Para qué sirve un tipo Optional? . Su uso esta centrado en eliminar muchos de los problemas que ocurren con el manejo de excepciones de tipo NullPointerException . Vamos a ver un ejemplo sencillo. Supongamos que partimos de una clase de servicio que tiene una lista de Personas que deseamos filtrar. Vamos a ver una primera versión el código:

```
package com.arquitecturajava;

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

public class ServicioPersonas {

    static List<Persona> lista= new
ArrayList<Persona>();
    static {

        lista.add(new Persona("pedro"));
        lista.add(new Persona("angel"));
        lista.add(new Persona("ana"));
    }

    public Persona buscar(String nombre) {

        for (Persona p:lista) {

            if (p.getNombre().equals(nombre)) {
                return p;
            }
        }
    }
}
```

```

        }

        return null;
    }

}

```

En este caso usamos un bucle `forEach` para buscar la `Persona` que coincide con el nombre y devolverla. Un programa `main` puede hacer uso de nuestra clase de `Servicio`.

```

package com.arquitecturajava;

public class Principal2 {

    public static void main(String[] args) {

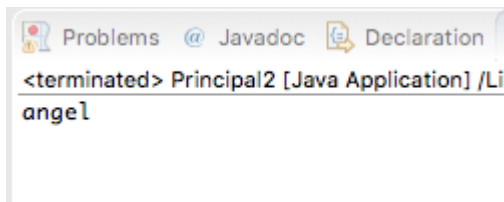
        ServicioPersonas sp= new ServicioPersonas();
        Persona p=sp.buscar("gema");
        System.out.println(p.getNombre());

    }

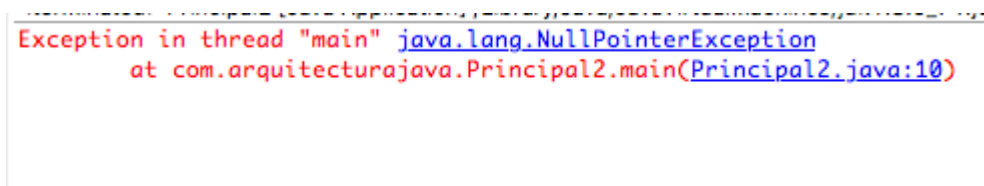
}

```

El programa funciona correctamente e imprime `angel` por la consola:



Sin embargo no sucedería lo mismo si pasáramos como parámetro “gema” ya que esta Persona no existe en la lista y el resultado será un `NullPointerException`:



Una gran parte de nuestros problemas en tiempo de ejecución vienen dados por este tipo de excepciones.

Un ejemplo de Java 8 Optional

Vamos a abordar el mismo ejemplo pero usando un Java Optional. Para ello modificamos el método buscar de la clase de servicio creando una variable de tipo Optional:

```
public Optional<Persona> buscar(String nombre) {  
  
    for (Persona p:lista) {  
  
        if (p.getNombre()==nombre) {  
            return Optional.of(p);  
        }  
    }  
  
    return Optional.empty();  
}
```

Esto nos obligará a modificar el código de nuestro programa main y usar los optionals evitando las excepciones de tipo NullPointerException :

```
ServicioPersonasOptional sp= new ServicioPersonasOptional();
Optional<Persona> op=sp.buscar("gema");
if (op.isPresent()) {
    System.out.println(op.get().getNombre());
}else {
    System.out.println("no hay registros");
}
```

Los tipos optional soportan métodos como isPresent() que fuerzan un chequeo antes de acceder al valor. Las propias APIS de Java hacen un uso fuerte de este concepto . Por ejemplo en vez de usar un bucle for para buscar en la lista podríamos haber usado un Stream, el cual devuelve un optional

```
public Optional<Persona> buscar(String nombre) {

    return lista.stream().filter(p-
    >p.getNombre()==nombre).findFirst();
}
```

El uso de Java 8 Optional es muy práctico y refuerza los conceptos de programación funcional que tenemos en el lenguaje.

Otros artículos relacionados:

[Java Lambda](#)

[Programación Funcional, Java 8 Streams](#)

[Java Functional Interface](#)