Java 8 NullPointerException

En Java 8, la **NullPointerException** es una excepción de tiempo de ejecución que ocurre al intentar utilizar un objeto nulo, como llamar a sus métodos o acceder a sus campos. Para mitigar este error común, Java 8 introdujo la clase **Optional**. que permite manejar valores ausentes de manera más elegante y segura.

Soluciones en Java 8

- **Uso de Optional**: Envuelve el objeto potencialmente nulo en `Optional.ofNullable()` y utiliza métodos como `.orElse()` para proporcionar un valor predeterminado si el objeto es nulo, evitando así el lanzamiento de la excepción.
- **Verificación explícita**: Se recomienda verificar si el objeto es nulo antes de usarlo, o devolver colecciones vacías en lugar de `null` desde los métodos para reducir la necesidad de comprobaciones constantes.
- **Herramientas de validación**: Se puede utilizar `Objects.requireNonNull()` para lanzar una excepción con un mensaje personalizado si un parámetro es nulo, asegurando que el código no continúe con referencias inválidas.

Una `NullPointerException` en Java es una `RuntimeException`. Ocurre cuando un programa intenta utilizar una referencia de objeto que tiene el valor nulo. En Java, `null` es un valor especial que se puede asignar a referencias de objetos para indicar la ausencia de un valor.

Razones de la excepción del puntero nulo

Se produce una `NullPointerException` debido a los siguientes motivos:

- Invocar un método desde un objeto nulo.
- Acceder o modificar el campo de un objeto nulo.
- Tomando la longitud de `null`, como si fuera un array.
- Accediendo o modificando los slots de objetos nulos, como si de un array se tratara.
- Lanzando nulo, como si fuera un valor `Throwable`.
- Cuando intentas sincronizar sobre un objeto nulo.

Ejemplo:

```
public class Geeks {

    public static void main(String[] args) {

        // Reference set to null
        String s = null;

        System.out.println(s.length());
    }
}
```

Salida

```
Hangup (SIGHUP)
Exception in thread "main" java.lang.NullPointerException
at Geeks.main(Geeks.java:10)
```

Explicación:

En este ejemplo, la referencia de cadena `s` es nula. Cuando el programa intenta llamar al método `length()`, genera una excepción `NullPointerException` porque no hay ningún objeto real.

Caso de uso de nulo

El valor nulo sirve como marcador de posición e indica que no se asigna ningún valor a una variable de referencia. Las aplicaciones comunes incluyen:

- **Estructuras de datos vinculados**: Representa el final de una lista o rama de árbol.
- **Patrones de diseño**: se utiliza en patrones como el patrón de objeto nulo o el patrón singleton.

Cómo evitar la excepción `NullPointerException`

Para evitar `NullPointerException`, debemos asegurarnos de que todos los objetos se inicialicen correctamente antes de usarlos. Cuando declaramos una variable de referencia, debemos verificar que el objeto no sea nulo, antes de solicitar un método o un campo de los objetos.

Ejemplos

1. **Usar literales de cadena en iguales ()**

   Un caso de problema muy común implica la comparación entre una variable `String` y un literal. El literal puede ser una cadena o un elemento de una enumeración. En lugar de invocar el método desde el objeto nulo, considere invocarlo desde el literal.

   Ejemplo:

```
import java.io.*;

class Geeks {
    public static void main (String[] args) {

        // Initializing String variable with null value
        String s = null;

        // Checking if s.equals null
        try
        {
            // This line of code throws NullPointerException because s is null
            if (s.equals("gfg"))
                System.out.print("Same");
            else
                System.out.print("Not Same");
        }
        catch(NullPointerException e)
        {
            System.out.print("NullPointerException Caught");
        }
    }
}
```

   Salida

```
NullPointerException Caught
```

   Podemos evitar `NullPointerException` llamando a iguales en un literal en lugar de en un objeto.

```
import java.io.*;

class Geeks {
    public static void main (String[] args) {

        // Initializing String variable with null value
        String s = null;

        // Checking if s is null using try catch
        try
        {
            if ("gfg".equals(s))
                System.out.print("Same");
            else
```

```
                System.out.print("Not Same");
        }
        catch(NullPointerException e)
        {
            System.out.print("Caught NullPointerException");
        }
    }
}
```

Salida

```
Not Same
```

Nota: Invoque siempre iguales en el literal para evitar llamar a un método en una referencia nula.

2. **Verificación de argumentos del método**

   Antes de ejecutar un método, se deben validar sus argumentos, como verificar si hay valores nulos. El método debe continuar sólo cuando las entradas sean válidas y estén debidamente verificadas. De lo contrario, debería generar una `IllegalArgumentException` para notificar a la persona que llama sobre argumentos no válidos.

   Ejemplo:

```
import java.io.*;

class Geeks {
    public static void main(String[] args) {

        // String s set an empty string and calling getLength()
        String s = "";

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }

        // String s set to a value and calling getLength()
        s = "GeeksforGeeks";

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }

        // Setting s as null and calling getLength()
        s = null;

        try {
            System.out.println(getLength(s));
        }
        catch (IllegalArgumentException e) {
            System.out.println(
                "IllegalArgumentException caught");
        }
    }

    public static int getLength(String s)
    {
        if (s == null)
            throw new IllegalArgumentException(
                "The argument cannot be null");

        return s.length();
    }
}
```

Salida

```
0
13
IllegalArgumentException caught
```

3. **Utilice el operador ternario**

   El operador ternario se puede utilizar para evitar `NullPointerException`. Primero, se evalúa la expresión booleana. Si la expresión es verdadera, se devuelve el `valor1`; de lo contrario, se devuelve el `valor2`. Podemos utilizar el operador ternario para manejar punteros nulos.

   Ejemplo:

```
import java.io.*;

class Geeks {
    public static void main(String[] args)
    {
        String s = null;
        String m = (s == null) ? "" : s.substring(0, 5);

        System.out.println(m);

        s = "Geeksforgeeks";
        m = (s == null) ? "" : s.substring(0, 5);
        System.out.println(m);
    }
}
```

   Salida

```
Geeks
```

   Explicación: El operador ternario ayuda a comprobar si hay nulos y a evitar operaciones con referencias nulas.

4. **Uso de clases opcionales (Java 8+)**

   En Java 8, la clase `Opcional` se introdujo como un objeto contenedor que puede contener o no un valor no nulo. Ayuda a evitar `NullPointerException` al obligar a manejar explícitamente el caso cuando falta un valor.

   Ejemplo:

```
import java.util.Optional;

public class OptionalExample {
    public static void main(String[] args) {
        Optional<String> name = Optional.ofNullable(null);

        // Safe way to access
        System.out.println(name.orElse("Default Name")); // prints: Default Name
    }
}
```

   Salida

```
Default Name
```

   Explicación: `Opcional.ofNullable(valor)` envuelve el valor que podría ser nulo. `orElse()` proporciona un recurso alternativo si el valor no está presente.

Test

Marque la alternativa correcta

1. **¿Qué pasará en el siguiente código?**

```
public class Geeks {
    public static void main(String[] args) {
        try {
            throw new NullPointerException("Demo");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception");
        } finally {
            System.out.println("Finally block executed");
        }
    }
}
```

a) `Arithmetic Exception Finally block executed`
b) `NullPointerException Finally block executed`
c) `Finally block executed`
d) `Compilation Error`

2. **¿Qué causa una `NullPointerException` en Java?**

a) Usando una variable primitiva no inicializada
b) Accediendo a una referencia de objeto que apunta a nulo
c) Llamar a un método estático
d) Creando un objeto con nueva palabra clave

3. **¿Cuál de los siguientes arrojará una `NullPointerException`?**

a) Accediendo a la longitud de una matriz válida
b) Comprobando si una cadena está vacía
c) Llamar a un método en una referencia de objeto nulo
d) Declarar una variable como nula

4. **¿Cuándo ocurre una NullPointerException en Java?**

a) Al acceder a una matriz fuera de los límites
b) Cuando se llama a un método en una referencia de objeto nulo
c) Al dividir un número por cero
d) Cuando no se encuentra un archivo

Referencias

- https://www.geeksforgeeks.org/java/null-pointer-exception-in-java/