



**UNIVERSIDAD AUTÓNOMA DE
CHIAPAS**

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

LIDTS

6 M

COMPILADORES

SUB3

COMPROBACIONES DE TIPOS EN EXPRESIONES

ALUMNO: MARCO ANTONIO ZÚÑIGA MORALES

DOCENTE: DR. LUIS GUTIÉRREZ ALFARO

TUXTLA GTZ, CHIAPAS

NOVIEMBRE, 2023



COMPROBACION ES DE TIPOS EN EXPRESIONES



Comprobaciones de tipos en expresiones



¿Qué es?

Las comprobaciones de tipos en expresiones son una parte importante de la programación, especialmente en lenguajes de programación estáticamente tipados.

Estas comprobaciones se realizan para asegurarse de que los tipos de datos utilizados en una expresión son compatibles y no causarán errores en tiempo de ejecución.

Comprobaciones de tipos en expresiones

Conversión de tipos (Casting)

A veces, es necesario convertir un tipo de dato a otro para que una expresión sea válida. Esto se hace mediante operaciones de conversión de tipos.

Por ejemplo, en muchos lenguajes de programación, puedes convertir un entero en un número decimal usando un casting, como (double) entero.

Comprobación de tipos estática

En lenguajes de programación estáticamente tipados, como Java, C++ o C#, las comprobaciones de tipos se realizan en tiempo de compilación. El compilador verifica que las expresiones sean consistentes en cuanto a tipos antes de generar el código ejecutable. Esto ayuda a atrapar errores de tipo antes de que el programa se ejecute.

Comprobaciones de tipos en expresiones

Comprobación de tipos dinámica

En lenguajes de programación dinámicamente tipados, como JavaScript o Python, las comprobaciones de tipos se realizan en tiempo de ejecución. Esto significa que los errores de tipo pueden surgir durante la ejecución del programa si se utiliza un tipo de dato incorrecto en una expresión.

Operadores de comprobación de tipos

Algunos lenguajes de programación proporcionan operadores o funciones específicas para comprobar tipos de datos. Por ejemplo, en JavaScript, puedes usar “typeof” para verificar el tipo de una variable.

Comprobaciones de tipos en expresiones

Manejo de excepciones

En algunos lenguajes, como Java o C#, se pueden usar excepciones para manejar errores de tipo. Cuando una expresión no cumple con los requisitos de tipo, se puede lanzar una excepción y manejarla en un bloque de manejo de excepciones.

Inferencia de tipos

En lenguajes de programación modernos, como TypeScript o Kotlin, se puede aprovechar la inferencia de tipos para reducir la necesidad de anotar explícitamente los tipos en las expresiones. El compilador puede deducir los tipos de datos basándose en el contexto.

Comprobaciones de tipos en expresiones

Pruebas unitarias

Las pruebas unitarias son una parte importante del proceso de desarrollo de software para comprobar que las expresiones y funciones se comportan correctamente en términos de tipos. Las pruebas unitarias permiten identificar y solucionar problemas de tipo de manera efectiva.


Análisis estático de código

Herramientas de análisis estático de código, como linters o analizadores estáticos, pueden ayudar a identificar problemas de tipo en el código fuente antes de la ejecución. Estas herramientas son especialmente útiles en lenguajes dinámicos.

Ventajas


Detección temprana de errores

Las comprobaciones de tipos en tiempo de compilación (tipado estático) permiten detectar errores de tipo antes de que el programa se ejecute, lo que facilita la corrección y reduce la probabilidad de errores en tiempo de ejecución.



Mejora de la seguridad y la robustez

El tipado estático ayuda a evitar errores de tipo que podrían llevar a comportamientos inesperados o fallos en tiempo de ejecución. Esto hace que los programas sean más seguros y robustos.



Documentación

Las anotaciones de tipos en el código fuente proporcionan documentación clara y explícita sobre qué tipos de datos se esperan y se deben usar en una expresión, lo que facilita la comprensión del código.

Ventajas

Optimización de rendimiento

Los compiladores pueden realizar optimizaciones basadas en el conocimiento de los tipos de datos utilizados en una expresión, lo que puede conducir a un mejor rendimiento del programa.

Mantenimiento más sencillo

Las comprobaciones de tipos en tiempo de compilación ayudan a evitar errores de tipo sutiles y facilitan el mantenimiento del código, especialmente en proyectos grandes y complejos.

Desventajas

Rigidez y verbosity

El tipado estático puede requerir más trabajo y código adicional para definir los tipos

Curva de aprendizaje

Los programadores pueden necesitar tiempo adicional para aprender y aplicar correctamente las anotaciones de tipos, lo que puede ralentizar el desarrollo inicial.

Menos expresividad

Esto puede ser menos adecuado para tareas que requieren flexibilidad dinámica.

Dificultad con tipos genéricos

En algunos lenguajes con tipado estático, trabajar con tipos genéricos o paramétricos puede ser más complejo y requerir una sintaxis más elaborada.

Costo computacional adicional

Las comprobaciones de tipos en tiempo de compilación y las anotaciones de tipos pueden aumentar el tiempo de desarrollo y el tamaño del binario del programa.

Python (Tipado dinámico)

En Python, el tipado es dinámico, lo que significa que las comprobaciones de tipos se realizan en tiempo de ejecución.

En este ejemplo, se utiliza la función **"isinstance"** para verificar si **"x"** es de tipo **"int"** y si **"y"** es de tipo **"str"**. Solo si ambos tipos son compatibles, se realiza la concatenación y se muestra el resultado.

EJEMPLO



```
x = 5  
y = "Hola"
```

```
# Comprobación de tipos antes de realizar una operación  
if isinstance(x, int) and isinstance(y, str):  
    resultado = str(x) + y  
    print(resultado)  
else:  
    print("Los tipos no son compatibles.")
```

Java (Tipado estático)

En Java, el tipado es estático, lo que significa que las comprobaciones de tipos se realizan en tiempo de compilación.

En este ejemplo, se utiliza el operador **"instanceof"** para verificar si **"x"** es una instancia de la clase **"Integer"** (que es un tipo de dato primitivo) y si **"y"** es una instancia de la clase **"String"**. Las comprobaciones de tipos se realizan en tiempo de compilación, lo que significa que el código no se compilará si los tipos no son compatibles.

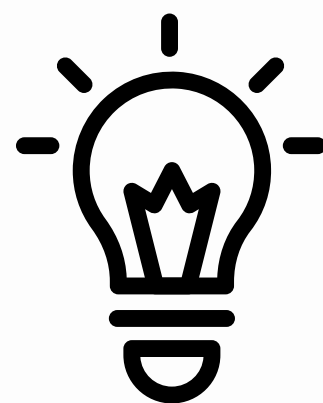
EJEMPLO

```
public class ComprobacionTiposJava {  
    public static void main(String[] args) {  
        int x = 5;  
        String y = "Hola";  
  
        // Comprobación de tipos en tiempo de compilación  
        if (x instanceof Integer && y instanceof String) {  
            String resultado = x + y;  
            System.out.println(resultado);  
        } else {  
            System.out.println("Los tipos no son compatibles.");  
        }  
    }  
}
```

CONCLUSIÓN

En conclusión, las comprobaciones de tipos en expresiones son una parte fundamental de la programación para garantizar la integridad y la corrección de los programas. La forma en que se realizan estas comprobaciones varía según el lenguaje de programación que estés utilizando y el enfoque de tipado del mismo.





MUCHAS GRACIAS

POR VER ESTA PRESENTACIÓN

