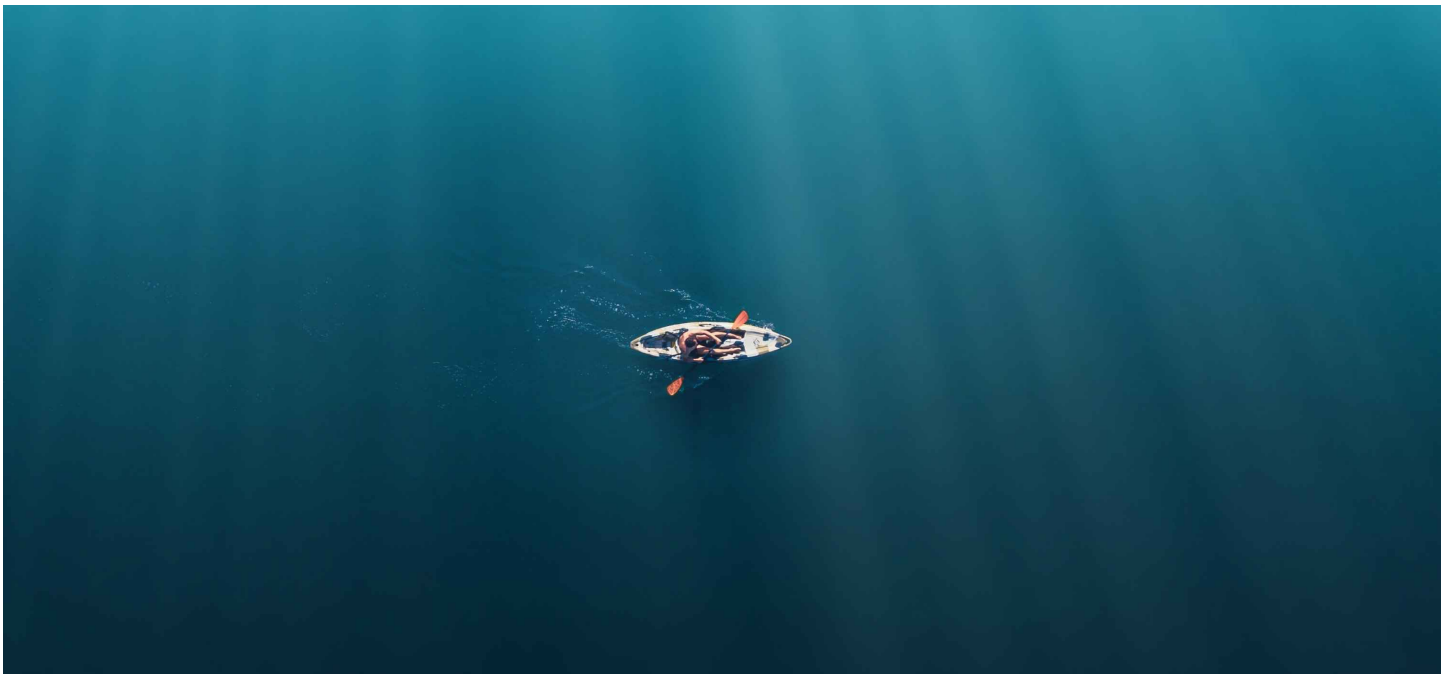

35-Java 10: Type Inference

Author: Vincent Lau

Learning Objectives

> Understand how Java implements Local Variable Type Inference by var
>
> Understand what var can do, and cannot do
>
> Understand what exactly compile-time type inference means

Introduction

The `var` keyword was introduced in Java 10 as a local variable type inference feature. It allows you to declare variables without explicitly specifying their types, and the compiler infers the type from the initialization expression. Here's how you can use the `var` keyword with some code examples:

Compile-time Type Inference

The `var` keyword in Java is related to **compile-time type inference**. When you use `var` to declare a variable, the compiler determines the actual type of the variable based on the expression on the right-hand side of the assignment during compilation. Once the code is compiled, the type is explicitly known and fixed.

Here's a breakdown of how it works:

1. You declare a variable using `var` and provide an initialization expression.
2. During compilation, the compiler analyzes the initialization expression to determine its type.
3. The determined type becomes the actual type of the variable.
4. The compiled code uses the determined type for type checking and execution.

For example:

```
var number = 42; // Compiler infers type as int
var name = "John"; // Compiler infers type as String
```

In this code, after compilation, the `number` variable will be treated as an `int` and the `name` variable as a `String`. This approach reduces the need for you to explicitly declare the types while still maintaining the benefits of strong typing and compile-time checking.

You Can Do

Simple Variable Declaration

```
var age = 25; // Compiler infers type as int
var name = "John"; // Compiler infers type as String
```

Enhanced For Loop

```
List<String> names = List.of("Alice", "Bob", "Charlie");
for (var name : names) {
    System.out.println(name);
}
```

Lambda Expressions

```java
Runnable runnable = () -> System.out.println("Hello, World!");
var runnable2 = (Runnable) () -> System.out.println("Hello, World!"); // Type can also be inferred
```

Method Return Types

```java
var result = calculateResult(); // Compiler infers type based on the method return type
```

Complex Initialization

```java
var point = new Point(); // Compiler infers type as Point
point.setX(10);
point.setY(20);
```

You Can't Do

Re-assign value

```java
var number = 42; // Compiler infers type as int
number = "John"; // compile-error
```

Method Parameters

```java
public static void print(var str) { // compile-error
    System.out.println(str);
}
```

Return Types

```java
public static var print(String str) { // compile-error
    return 1;
}
```

Instance Variables

```
1 public class Person {
2     var age; // compile-error
3 }
```

Without the initializer

```
1 var n; // error: cannot use 'var' on variable without initializer
```

Initialized with null

```
1 var emptyList = null; // error: variable initializer is 'null'
```

Lambda expression

```
1 var p = (String s) -> s.length() > 10; // error: lambda expression needs an
  explicit target-type
```

Array initializer

```
1 var arr = { 1, 2, 3 }; // error: array initializer needs an explicit target-
  type
```

Summary & Key Notes

- The type inference is done by the compiler **at compile-time**, and the type of the variable is determined by the expression on the right-hand side of the assignment.
- The `var` keyword can only be used for local variables with initializers. **It cannot be used for method parameters, fields, or return types**.

Note that while using `var` can make code more concise, **overuse of it can lead to less readable** and more confusing code. It's important to strike a balance between using `var` and providing explicit type declarations to improve code clarity.