

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
<div><div></div>Optionals</div>
<div><div></div>JDBC CRUD</div>
<div><div></div>Exceptions</div>
<div><div></div>SOLID Principles</div>
<div><div></div>Single Responsibility</div>
<div><div></div>Open/Closed</div>
<div><div></div>Liskov Substituiton</div>
<div><div></div>Interface Segregation</div>
<div><div></div>Dependency Inversion</div>
<div><div></div>Best Practice</div>
<div><div></div>Design Patterns</div>
<div><div></div>Creational Design Patterns</div>
<div><div></div>Structural Design Patterns</div>
<div><div></div>Behavioural Design Patterns</div>
<div><div></div>Collection & Map</div>
<div><div></div>HashSets</div>
<div><div></div>HashMaps</div>
<div><div></div>Enums</div>
<div><div></div>Logging</div>
<div><div></div>Generics</div>
<div><div></div>Lambda Expressions</div>
<div><div></div>Streams</div>
<div><div></div>Complexity</div>
<div><div></div>Input and Output</div>
<div><div></div>Local Type Inference</div>
HTML
CSS

Local Type Inference

Contents

- [Overview](#)
- [Tutorial](#)
- [Exercises](#)

Overview

Introduced in Java 10; **local type inference** allows developers to substitute the **var** keyword for the type of *any local variable*.

The basic idea behind **local type inference** is to improve code readability by removing the need to write out long or complex class names multiple times when creating a variable.

Tutorial

Consider this example:

```
public void readFile(String path) {
    try (BufferedFileInputStream bfis = new BufferedFileInputStream(path)) {
        // file stuff
    }
}
```

Here's the same code with **local type inference**:

```
public void readFile(String path) {
    try (var bfis = new BufferedFileInputStream()) {
        // file stuff
    }
}
```

Notice how much shorter the second one is even with such a trivial example? By replacing long or unreadable types with **local type inference** it is possible to make your code much more concise *and* easier to read.

Now you've seen how you *can* use **var** let's look at the ways you *can't* use it:

- No value to infer a type from:

```
var a = null;
a = 37; // adding a value afterwards doesn't help
var b;  // still doesn't work
```

- Trying to change the type after variable initialisation:

```
var num = 27;
// Type is inferred to be Integer
num = "Bonjour";
// Can't save a String to an Integer variable
```

- Class members:

```
class SimpleClass {
    var classMember = new Object(); // compilation error
}
```

- Method parameters:

Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
class SimpleClass {  
  
    void method(var param) {  
        // another compilation error  
    }  
}
```

When using `var` it's important to remember that it should *not* be used at all times; if replacing a type with `var` doesn't make your code more concise or if it makes it harder to read then *don't use it*.

The people behind the OpenJDK wrote up a handy style guide for **local type inference** that you can find [here](#).

Exercises

Copy [this](#) class into your IDE.

Try using **local type inference** with this class - think about which variables *can* use `var` and, particularly, which variables *should* use `var`.