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:

1. 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
```

2. 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
```

3. Class members:

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

4. Method parameters:

```
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`.