(/)

Java 10 LocalVariable Type-Inference

Last modified: September 10, 2018

by Ganesh Pagade (/author/ganesh-pagade/)

Java (/category/java/) +

Java 10 (/tag/java-10/)

This article is part of a series:

1. Overview

One of the most visible enhancements in JDK 10 is type inference of local variables with initializers.

This tutorial provides the details of this feature with examples.

2. Introduction

Until Java 9, we had to mention the type of the local variable explicitly and ensure it was compatible with the initializer used to initialize it:

```
String message = "Good bye, Java 9";
```

In Java 10, this is how we could declare a local variable:

```
@Test
public void whenVarInitWithString_thenGetStringTypeVar() {
    var message = "Hello, Java 10";
    assertTrue(message instanceof String);
}
```

We don't provide the data type of *message*. Instead, we mark *the message* as a *var*, and the compiler infers the type of *message* from the type of the initializer present on the right-hand side.

In above example, the type of *message* would be *String*.

Note that this feature is available only for local variables with the initializer. It cannot be used for member variables, method parameters, return types, etc – the initializer is required as without which compiler won't be able to infer the type.

This enhancement helps in reducing the boilerplate code; for example:

```
Map<Integer, String> map = new HashMap<>();
```

This can now be rewritten as:

```
var idToNameMap = new HashMap<Integer, String>();
```

This also helps to focus on the variable name rather than on the variable type.

Another thing to note is that ***var* is not a keyword** – this ensures backward compatibility for programs using *var* say, as a function or variable name. *var* is a reserved type name, just like *int*.

Finally, note that there is **no runtime overhead in using *var* nor does it make Java a dynamically typed language.** The type of the variable is still inferred at compile time and cannot be changed later.

3. Illegal Use of *var*

As mentioned earlier, *var* won't work without the initializer:

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

Nor would it work if initialized with *null*:

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

It won't work for non-local variables:

```
public var = "hello"; // error: 'var' is not allowed here
```

Lambda expression needs explicit target type, and hence *var* cannot be used:

```
var p = (String s) -> s.length() > 10; // error: lambda expression nee
```

Same is the case with the array initializer:

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

4. Guidelines for Using *var*

There are situations where *var* can be used legally, but may not be a good idea to do so.

For example, in situations where the code could become less readable:

```
var result = obj.prcoess();
```

Here, although a legal use of *var*, it becomes difficult to understand the type returned by the *process()*making the code less readable.

java.net (http://openjdk.java.net/)has a dedicated article on Style Guidelines for Local Variable Type Inference in Java (http://openjdk.java.net/projects/amber/LVTIstyle.html)which talks about how we should use judgment while using this feature.

Another situation where it's best to avoid *var* is in streams with long pipeline:

```
var x = emp.getProjects.stream()
  .findFirst()
  .map(String::length)
  .orElse(0);
```

Usage of *var* may also give unexpected result.

For example, if we use it with the diamond operator introduced in Java 7:

```
var empList = new ArrayList<>();
```

The type of *empList*will be *ArrayList<Object>*and not *List<Object>*. If we want it to be *ArrayList<Employee>*, we will have to be explicit:

```
var empList = new ArrayList<Employee>();
```

Using *var* with non-denotable types could cause unexpected error.

For example, if we use *var* with the anonymous class instance:

```
@Test
public void whenVarInitWithAnonymous_thenGetAnonymousType() {
    var obj = new Object() {};
    assertFalse(obj.getClass().equals(Object.class));
}
```

Now, if we try to assign another *Object*to *obj*, we would get a compilation error:

```
obj = new Object(); // error: Object cannot be converted to <anonymous
```

This is because the inferred type of *obj* isn't *Object*.

5. Conclusion

In this article, we saw the new Java 10 local variable type inference feature with examples.

As usual, code snippets can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-10).

Next »
Java 10 Performance Improvements (https://www.baeldung.com/java-10-performance-improvements)

▲ newest ▲ oldest ▲ most voted

Guest

Alonso Isidoro Roman (http://aironman2k.wordpress.com)

It's funny that precisely what makes it nice for some people to use python or scala instead of java to not have to declare the types, now we seem to realize that it's bad practice not to force the typing with version 10. It has always been a bad idea not to force typing, both because of performance issues and because of code maintenance and readability issues.

✚ 0 ━

3 months ago

(/author/grzegorz-author/)

Editor

Grzegorz Piwowarek (http://4comprehension.com)

Everything is contextual, it's just a bit easier to write unreadable code if you're not forced to provide types explicitly.

+ 0 −

3 months ago

CATEGORIES

SPRING (/CATEGORY/SPRING/)
REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JACKSON/)
HTTPCLIENT (/CATEGORY/HTTP/)
KOTLIN (/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (HTTPS://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTPS://WWW.BAELDUNG.COM/JACKSON)
HTTPCLIENT 4 TUTORIAL (HTTPS://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTPS://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT/)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)

META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
CONTACT (/CONTACT)
EDITORS (/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)