Local-variable Type Inference

Sander Mak
FELLOW & SOFTWARE ARCHITECT
@Sander_Mak

Local-variable Type Inference

```
public void aMethod() {
   var name = "Sander";
}
```

Like C#, Scala, Kotlin

Why?

```
URL url = new URL("http://javamodularity.com");
URLConnection connection = url.openConnection();
BufferedInputStream inputStream =
    new BufferedInputStream(connection.getInputStream());
```

```
var bookurl = new URL("http://javamodularity.com");
var connection = bookurl.openConnection();
var bookStream = new BufferedInputStream(connection.getInputStream());
```

Focus on variable names

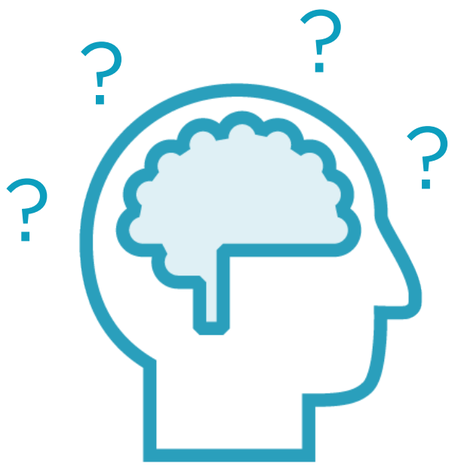
Why?

```
var ugly     var name = "Sander"     eBest();
```

Why Not?

Code is typically written **once**, but read **many** times by many people

```
var result = aService.findTheThing();
```



Reserved Type Name

Is var a new keyword? No!

```
var var = "var";
```

```
package com.myproject.var;
```

Backward compatibility for existing identifiers

Type Inference

"Does this make Java a dynamic language?"

No!

Type is still there, only now *inferred* by compiler

Type Inference

```
int counter = 0;
counter += 1;
```

```
0: iconst_0
1: istore_1
2: iinc          1, 1
```

```
var counter = 0;
counter += 1;
```

```
0: iconst_0
1: istore_1
2: iinc          1, 1
```

Type Inference

Not new in Java:

```
List<String> myList = new ArrayList<>()
```

```
List<String> myList = Collections.<String>emptyList()
```

```
Predicate<String> p = s -> s.length() > 3
```

Limitations of Type Inference

```
var p = s -> s.length() > 3
```

X

```
var p = (String s) -> s.length() > 3
```

X

Lambdas **must have** an explicit **target type**

Limitations of Type Inference

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

ArrayList<Object>

```
var myList = new ArrayList<String>();
```

Local Type Inference Only

Method parameters

Fields

Return types

Catch blocks

```
public void aMethod(var input) {
   ...
}
```

Demo

Using `var`

Non-denotable Types

Null

Anonymous class instances

```
new Object() {}
```

Intersection types

```
(Comparable & Serializable)
```

Non-denotable Types

Null

```
var empty = null;
```

X

Non-denotable Types

Anonymous class instances

```
var obj = new Object() {}
```

Object$1 extends Object

```
obj = new Object()
```

X

Non-denotable Types

Intersection types

```
var list = List.of(1, 2.0, "3");
```

```
List<? extends Serializable & Comparable<..>>
```

All of these types may show up in errors

Summary

Type inference: local variables with **var**

Prevent abuse, use **var** responsibly

Inferred types may be non-denotable