Local Variable Type-Inference: Friend or Foe?

Simon Ritter, Azul Systems

Stuart Marks, Oracle Corporation

@speakjava

@stuartmarks

Local Variable Type Inference

- Why?
  - JavaScript has it so it must be good
  - Streams hide intermediate types

```
List l = names.stream()              // Stream<String>
  .filter(s -> s.length() > 0)       // Stream<String>
  .map(s -> getRecord(s))            // Stream<Record>
  .collect(toList());                // ArrayList<Record>
```

  - Why not expand this elsewhere?
  - var can make code more concise
    - Without sacrificing readability

Simple Uses of Local Variables

```
ArrayList<String> userList = new ArrayList<>();
Stream<String> stream = userList.stream();

for (String name : userList) {
}

for (int i = 0; i < 10; i++) {
}
```

Simple Uses of var

```
var userList = new ArrayList<String>();  // infers ArrayList<String>
var stream = userList.stream();          // infers Stream<String>


for (var name : userList) {              // infers String
}

for (var i = 0; i < 10; i++) {               // infers int
}
```

Simple Uses of var

- Clearer try-with-resources

```
try (InputStream inputStream = socket.getInputStream();
     InputStreamReader inputStreamReader =
       new InputStreamReader(inputStream, UTF_8);
     BufferedReader bufferedReader =
       new BufferedReader(inputStreamReader)) {
     // Use bufferedReader
}
```

Simple Uses of var

- Clearer try-with-resources

```
try (var inputStream = socket.getInputStream();
     var inputStreamReader = new InputStreamReader(inputStream, UTF_8);
     var bufferedReader = new BufferedReader(inputStreamReader)) {
     // Use bufferedReader
}
```

More Typing With Less Typing

- Java is still statically typed

```
var name = "Simon";      // Infers String, so name is String

name = "Dylan";          // Still String

name = 42;               // Not a String, error
```

Final and Non-Final

- var simply indicates the compiler infers the type
- Use of var does not make a variable final

```
var name = "Simon";
name = "Dylan";       // name is not final
```

- Still need final

```
final var name = "Simon";
```

- No shortcut for `final var` (like `val`)
  - How often do you make local-variables final?

Action At A Distance

```
var l = new ArrayList<String>();
var s = l.stream();


// Lots of lines of complex code


var n = l.get(0);   // Err, what is n?
```

Jumping Variable Names

```
No no = new No();
AmountIncrease<BigDecimal> more =
  new BigDecimalAmountIncrease<>(5.1);
HorizontalConnection<Line, Line> jumping =
  new HorizontalLineConnection<>();
Constant variable = new Constant(42);
List<String> names = List.of("Stuart", "Simon");
```

Jumping Variable Names

```
var no = new No();
var more = new BigDecimalAmountIncrease<BigDecimal>(5.1);
var jumping = new HorizontalLineConnection<Line, Line>();
var variable = new Constant(42);
var names = List.of("Stuart", "Simon");
```

Not Everything Can Be Inferred

```
int[] firstSixPrimes = {2, 3, 5, 7, 11, 13};
```

```
var firstSixPrimes = {2, 3, 5, 7, 11, 13};
```

```
error: cannot infer type for local variable firstSixPrimes
    var firstSixPrimes = {2, 3, 5, 7, 11, 13};
        ^
  (array initializer needs an explicit target-type)
```

Initialisation is Required

```
var list = null;
```

```
error: cannot infer type for local variable list
    var list = null;
        ^
  (variable initializer is 'null')
```

Initialisation is Required

```
var list;

list = new ArrayList<String>();


error: cannot infer type for local variable list
    var list;
        ^
  (cannot use 'var' on variable without initializer)
```

Initialisation is Required

```
var list;

list = new ArrayList<String>();
...

list = 42;
```

What type to choose?

```
ArrayList<String> ?
int ?
Integer ?
Object ?
```

Literals with var (Good)

- Original
  - `boolean ready = true;`
  - `char ch = 'x';`
  - `long sum = 0L;`
  - `String label = "Foo";`

Literals with var (Good)

- With var
  - `var ready = true;`
  - `var ch = 'x';`
  - `var sum = 0L;`
  - `var label = "Foo";`

Literals with var (Dangerous)

- Original
  - `byte flags = 0;`
  - `short mask = 0x7fff;`
  - `long base = 10;`

Literals with var (Dangerous)

- Original
  - `var flags = 0;`
  - `var mask = 0x7fff;`
  - `var base = 10;`

All these cases will infer `int`

Beware of Multiple Type Inference

Diamond operator, primary type inference

```
var itemQueue = new PriorityQueue<>();
```

Secondary type inference

`itemQueue` infered as `PriorityQueue<Object>();`

Reserved Type, Not Keyword

```
var var = new ValueAddedReseller();
```

```
public class var {
  public var(String x) {
    ...
  }
}
```

```
public class Var {
  public Var(String x) {
    ...
  }
}
```

Programming To An Interface

- Common to use interface rather than concrete type

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

- Using var always infers concrete type

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

- Polymorphism still works
  - `ArrayList<String>` is still a `List`
  - You can use `myList` wherever a `List` is required

Lambdas and var

```
Predicate<String> blankLine = s -> s.isBlank();
```

```
var blankLine = s -> s.isBlank();
```

```
error: cannot infer type for local variable blankLine
    var blankLine = s -> s.isBlank();
        ^
  (lambda expression needs an explicit target-type)
```

Puzzler 1

```
static boolean calc1(int mask) {
  long temp = 0x12345678;
  return (((temp << 6) | temp) & mask) > 0;
}
```

Puzzler 1

```
static boolean calc1(int mask) {
  var temp = 0x12345678;
  return (((temp << 6) | temp) & mask) > 0;
}
```

- Should we use var? Yes? No?

Puzzler 1

- 0x12345678 fits in an int
  - Using var will infer an int, not a long
- Passing a mask < 0 will give different results
  - Using long, mask < 0 returns true
  - Using var (infer int), mask < 0 returns false

Puzzler 2

```
static List<String> create1(boolean foo, boolean bar) {
  List<String> list = new ArrayList<>();

  if (foo)
    list.add("foo");

  if (bar)
    list.add("bar");

  return list;
}
```

Puzzler 2

```
static List<String> create1(boolean foo, boolean bar) {
  var list = new ArrayList<>();

  if (foo)
    list.add("foo");

  if (bar)
    list.add("bar");

  return list;
}
```

- Should we use var? Yes? No?

Puzzler 2

- Remember, "Beware of multiple type inference"
  - `var list = new ArrayList<>();`
  - Diamond operator infers `Object`

```
error: incompatible types: ArrayList<Object> cannot be
converted to List<String>
    return list;
```

Puzzler 3

```
static List<String> create1(boolean foo, boolean bar) {
  List<String> list = new ArrayList<>(List.of("x", "y", "z"));

  if (foo)
    list.add("foo");

  if (bar)
    list.add("bar");

  return list;
}
```

Puzzler 3

```
static List<String> create1(boolean foo, boolean bar) {
  var list = new ArrayList<>(List.of("x", "y", "z"));

  if (foo)
    list.add("foo");

  if (bar)
    list.add("bar");

  return list;
}
```

- Should we use var? Yes? No?

Puzzler 3

- This time multiple type inference is not a problem
- The compiler can infer the generic type correctly
  - From the argument passed to the constructor

Puzzler 4

```
static List<Integer> listRemoval() {
  List<Integer> list =
    new ArrayList<>(Arrays.asList(1, 3, 5, 7, 9));
  var v = valueToRemove();
  list.remove(v);
  return list;
}

// Separate class, package or module (in a galaxy far, far away)
static Integer valueToRemove() {
  return 3;
}
```

- Good use of var? Yes? No?

Puzzler 4

- New intern arrives and changes code...

```
static int valueToRemove() {
  return 3;
}
```

- Unexpected change in behaviour

```
List.remove(int)      // Remove element at given index
List.remove(Object)   // Remove first instance of object

{1,3,7,9}   // Using var with changed valueToRemove
{1,5,7,9}   // Using var with unchanged valueToRemove
```

Guidelines For Use of var

1. Reading code is more important that writing it
2. Code should be clear from local reasoning
3. Code readability shouldn't depend on an IDE
4. Choose variable names that provide useful information
5. Minimise the scope of local variables
6. Consider var when the initialiser provides sufficient information
7. Use var to break up chained or nested expressions
8. Take care when using var with generics
9. Take care when using var with literals

Conclusions

- Local variable type inference is a useful feature
  - With great power comes great responsibility

- Consider implications of using `var`

Thank you!

Simon Ritter, Azul Systems

Stuart Marks, Oracle Corporation

@speakjava

@stuartmarks