Java 10: Local Variable Type Inference

We can use ***var*** for local variables instead of a typed name.

What is Type Inference?

Type inference is Java compiler's ability to look at each method invocation and corresponding declaration to determine the type argument (or arguments) that make the invocation applicable.

Let's have examples.

```
int a=43;  // Pre Java 10
var a = 43;  // Java 10 onwards
double  a=12.3;  // Pre Java 10
var a = 12.3;  // Java 10 onwards
String  a= "Rahul";  // Pre Java 10
var a= "Rahul";  // Java 10 onwards
```

How does Local Variable Type Inference work?

Parsing a ***var*** statement, the compiler looks at the right hand side of the declaration, and it infers the type from the right hand side (RHS) expression.

Does this mean that now Java is a dynamically typed language?

Not really, it's still a statically typed language.

Let's take an example:

```
class  Demo{
        public static void main(String s[]){
                var a=15;
                var b= "Rahul";
                var c=Double.parse Double ("12");
                System.out.println(a+b+c);
        }
}
```

Now, let's look at the decompiled code taken decompiler.

```
class  Demo{
        public static void main(String s[]){
                int a=15;
                String b= "Rahul";
                double c=Double.parse Double ("12");
                System.out.println(a+b+c);
        }
}
```

Here, compiler infers the type of the variable from the right hand side expression and adds that to the bytecode.

var is a reserved type name, but ***var*** is not a keyword, It's a reserved type name. What does it mean?

We can create a variable named "var".

```
var var = 5; // syntactically correct
// var is the name of the variable
```

"var" as a method name is allowed.

```
public static void var() {  } // syntactically correct
```

"var" as a package name is allowed.

```
package var;  // syntactically correct
```

var for Anonymous class is also allowed.

```
class A{
        void m(){
                //body
        }
}
var a=new A(){
        void m(){
                //body
         }
};
```

var Usage Scenarios

Local type inference can be used in the following scenarios:

- Limited only to Local Variable with initialization
- Indexes of enhanced for loop or indexes
- Local declared in for loop

Let's walk through the examples for these scenarios:

```
var numbers = List.of(1, 2, 3, 4, 5); // inferred value ArrayList<String>
// Index of Enhanced For Loop
for (var number : numbers) {
        System.out.println(number);
}
// Local variable declared in a loop
for (var i = 0; i < numbers.size(); i++) {
        System.out.println(numbers.get(i));
}
```

var Limitations

There are certain limitations of using var, let's take a look.

"var" cannot be used as the name of a class or interface.

```
class var{ }  // CompileTime  Error
interface  var{ }  // CompileTime  Error
```

as of release java 10, ***var*** is a restricted local variable type and cannot be used for type declarations

Cannot use 'var' on variables without initialization.

```
var x;  // error: cannot infer type for local variable x
```

Cannot be used for multiple variable definition.

```
var x = 5, y = 10;  // error: var is not allowed in a compound declaration
```

Null cannot be used as an initializer for var.

```
var a = null; // error: Null cannot be inferred to a type
```

Cannot have extra array dimension brackets.

```
var a[] = new int[10];  // error: var is not allowed as an element type of an array
```

Cannot be used for Array initializer.

```
var  a = {1,2,3,4,5};  //error: array initializer needs an explicit target-type
```

Cannot be used for Lambda expressions.

```
var  a = () > { } ;  //error
```

In Java 10, Now, we can do things like this:
Test divide = (***var*** x, ***var*** y) > x / y; //allowed

But, It is illegal to mix explicit and implicit styles. Lambda Expression formals should either be all implicitly or all explicitly typed. This example will not compile.

```
/* Semi-Explicit/Semi-Implicit Mix =>  Illegal */
Test  subtract = (var x, double y) -> x-y;   //error
```

It is illegal to mix inferred styles of declaring Lambda Expression formals. Implicitly typed Lambda Expression formals should take one syntax or the other but not both. This example will not compile.

```
/* Mixing Implicit Styles =>  Illegal */
Test  subtract = (var x, y) -> x-y;   //error
```

Omission of Parentheses in single argument
You must include a parentheses enclosing a single argument. This example will not compile.

```
/* Omission of Parentheses => Illegal */
Test  upper = var x -> { return x.toUpperCase(); }; ;   //error
// Legal
Test  upper = (var x) -> { return x.toUpperCase(); }; ;   // allowed
// Legal
Test  upper = x -> { return x.toUpperCase(); };  //allowed
```