JEP 323: Local-Variable Syntax for Lambda Parameters

Author	Brian Goetz
Owner	Vicente Arturo Romero Zaldivar
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	11
Component	tools
Discussion	amber dash dev at openjdk dot java dot net
Effort	XS
Duration	XS
Relates to	JEP 286: Local-Variable Type Inference
Reviewed by	Alex Buckley
Created	2017/12/08 15:15
Updated	2018/08/23 15:44
Issue	8193259

Summary

Allow `var` to be used when declaring the formal parameters of implicitly typed lambda expressions.

Goals

- Align the syntax of a formal parameter declaration in an implicitly typed lambda expression with the syntax of a local variable declaration.

Non-goals

- Align the syntax of any other kind of variable declaration, e.g., a formal parameter of a method, with the syntax of a local variable declaration.

Motivation

A lambda expression may be *implicitly typed*, where the types of all its formal parameters are inferred:

```
(x, y) -> x.process(y)    // implicitly typed lambda expression
```

Java SE 10 makes implicit typing available for local variables:

```
var x = new Foo();
for (var x : xs) { ... }
try (var x = ...) { ... } catch ...
```

For uniformity with local variables, we wish to allow 'var' for the formal parameters of an implicitly typed lambda expression:

```
(var x, var y) -> x.process(y)   // implicit typed lambda expression
```

One benefit of uniformity is that modifiers, notably annotations, can be applied to local variables and lambda formals without losing brevity:

```
@Nonnull var x = new Foo();
(@Nonnull var x, @Nullable var y) -> x.process(y)
```

Description

For formal parameters of implicitly typed lambda expressions, allow the reserved type name `var` to be used, so that:

```
(var x, var y) -> x.process(y)
```

is equivalent to:

```
(x, y) -> x.process(y)
```

An implicitly typed lambda expression must use `var` for all its formal parameters or for none of them. In addition, `var` is permitted only for the formal parameters of implicitly typed lambda expressions --- explicitly typed lambda expressions continue to specify manifest types for *all* their formal parameters, so it is not permitted for some formal parameters to have manifest types while others use `var`. The following examples are illegal:

```
(var x, y) -> x.process(y)         // Cannot mix 'var' and 'no var' in implicitly typed lambda expression
(var x, int y) -> x.process(y)     // Cannot mix 'var' and manifest types in explicitly typed lambda expression
```

In theory, it would be possible to have a lambda expression like the last line above, which is semi-explicitly typed (or semi-implicitly typed, depending on your point of view). However, it is outside the scope of this JEP because it deeply affects type inference and overload resolution. This is the main reason for keeping the restriction that a lambda expression must specify all manifest parameter types or none. We also want to enforce that the type inferred for a parameter of an implicitly typed lambda expression is the same whether `var` is used or not. We may return to the problem of partial inference in a future JEP. Also, we do not wish to