

Java Language Changes for Java SE 10

Java SE 10 introduces support for inferring the type of local variables from the context, which makes code more readable and reduces the amount of required boilerplate code.

Local-Variable Type Inference

Code such as that shown in the following example seems redundant and makes the code harder to read.

```
URL url = new URL("http://www.oracle.com/");
URLConnection conn = url.openConnection();
Reader reader = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
```

A new identifier named `var` is now available for local variables with non-null initializers. Using this identifier, the type of the variable is inferred from the context. The code from the previous example can be rewritten as shown in the following example:

```
var url = new URL("http://www.oracle.com/");
var conn = url.openConnection();
var reader = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
```

`var` is a reserved type name, not a keyword, which means that existing code that uses `var` as a variable, method, or package name is not affected. However, code that uses `var` as a class or interface name is affected and the class or interface needs to be renamed.

`var` can be used for the following types of variables:

- Local variable declarations with initializers
- Enhanced for-loop indexes
- Index variables declared in traditional for loops
- Try-with-resources variable

The following code shows examples using `var`:

```

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

var path = Paths.get(fileName);       // infers Path
var bytes = Files.readAllBytes(path);  // infers bytes[]

for (var counter=0; counter<10; counter++) {...}    // infers int

try (var input =
    new FileInputStream ("validation.txt")) {...}    // FileInputStream

```

Java Language Changes for Java SE 9

The major change to Java Platform, Standard Edition (Java SE) 9 is the introduction of the Java Platform module system.

The Java Platform module system introduces a new kind of Java programming component, the module, which is a named, self-describing collection of code and data. Its code is organized as a set of packages containing types, i.e., Java classes and interfaces; its data includes resources and other kinds of static information. Modules can either export or encapsulate packages, and they express dependencies on other modules explicitly.

To learn more about the Java Platform module system, see [Project Jigsaw](#) on OpenJDK.

Apart from the new module system, a few changes have been made to the Java language. The rest of this guide describes those changes.

More Concise try-with-resources Statements

If you already have a resource as a `final` or effectively `final` variable, you can use that variable in a `try-with-resources` statement without declaring a new variable. An "effectively final" variable is one whose value is never changed after it is initialized.

For example, you declared these two resources:

```

// A final resource
final Resource resource1 = new Resource("resource1");
// An effectively final resource
Resource resource2 = new Resource("resource2");

```

In Java SE 7 or 8, you would declare new variables, like this:

```

try (Resource r1 = resource1;
    Resource r2 = resource2) {
    ...
}

```

In Java SE 9, you don't need to declare `r1` and `r2`:

```

// New and improved try-with-resources statement in Java SE 9
try (resource1;
    resource2) {

```

```
    ...  
}
```

There is a more complete description of [the try-with-resources statement](#) in The Java Tutorials (Java SE 8 and earlier).

Small Language Changes in Java SE 9

There are several small language changes in Java SE 9.

@SafeVarargs annotation is allowed on private instance methods.

The **@SafeVarargs** annotation can be applied only to methods that cannot be overridden. These include static methods, final instance methods, and, new in Java SE 9, private instance methods.

You can use diamond syntax in conjunction with anonymous inner classes.

Types that can be written in a Java program, such as `int` or `String`, are called denotable types. The compiler-internal types that cannot be written in a Java program are called non-denotable types.

Non-denotable types can occur as the result of the inference used by the diamond operator. Because the inferred type using diamond with an anonymous class constructor could be outside of the set of types supported by the signature attribute in class files, using the diamond with anonymous classes was not allowed in Java SE 7.

In Java SE 9, as long as the inferred type is denotable, you can use the diamond operator when you create an anonymous inner class.

The underscore character is not a legal name.

If you use the underscore character ("_") an identifier, your source code can no longer be compiled.

Private interface methods are supported.

Private interface methods are supported. This support allows nonabstract methods of an interface to share code between them.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Java Platform, Standard Edition Java Language Updates, Release 10

E91239-01

Copyright © 2017, 2018, Oracle and/or its affiliates. All rights reserved.

Describes the updated language features in Java SE 9 and subsequent releases.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.