Legal Notice

Notice

The Specification is protected by copyright and the information described therein may be protected by one or more U.S. patents, foreign patents, or pending applications. Except as provided under the following license, no part of the Specification may be reproduced in any form by any means without the prior written authorization of Oracle America, Inc. ("Oracle") and its licensors, if any. Any use of the Specification and the information described therein will be governed by the terms and conditions of this Agreement.

Subject to the terms and conditions of this license, including your compliance with Paragraphs 1 and 2 below, Oracle hereby grants you a fully-paid, non-exclusive, non-transferable, limited license (without the right to sublicense) under Oracle's intellectual property rights to:

1. *Review the Specification for the purposes of evaluation. This includes: (i) developing implementations of the Specification for your internal, non-commercial use; (ii) discussing the Specification with any third party; and (iii) excerpting brief portions of the Specification in oral or written communications which discuss the Specification provided that such excerpts do not in the aggregate constitute a significant portion of the Technology.*

2. *Distribute implementations of the Specification to third parties for their testing and evaluation use, provided that any such implementation:*

    i. *does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented;*

    ii. *is clearly and prominently marked with the word "UNTESTED" or "EARLY ACCESS" or "INCOMPATIBLE" or "UNSTABLE" or "BETA" in any list of available builds and in proximity to every link initiating its download, where the list or link is under Licensee's control; and*

    iii. *includes the following notice: "This is an implementation of an early-draft specification developed under the Java Community Process (JCP) and is made available for testing and evaluation purposes only. The code is not compatible with any specification of the JCP."*

The grant set forth above concerning your distribution of implementations of the specification is contingent upon your agreement to terminate development and distribution of your "early draft" implementation as soon as feasible following final completion of the specification. If you fail to do so, the foregoing grant shall be considered null and void.

No provision of this Agreement shall be understood to restrict your ability to make and distribute to third parties applications written to the Specification.

Other than this limited license, you acquire no right, title or interest in or to the Specification or any other Oracle intellectual property, and the Specification may only be used in accordance with the license terms set forth herein. This license will expire on the earlier of: (a) two (2) years from the date of Release listed above; (b) the date on which the final version of the Specification is publicly released; or (c) the date on which the Java Specification Request (JSR) to which the Specification corresponds is withdrawn. In addition, this license will terminate immediately without notice from Oracle if you fail to comply with any provision of this license. Upon termination, you must cease use of or destroy the Specification.

"Licensor Name Space" means the public class or interface declarations whose names begin with "java", "javax", "com.oracle" or their equivalents in any subsequent naming convention adopted by Oracle through the Java Community Process, or any recognized successors or replacements thereof.

Trademarks

No right, title, or interest in or to any trademarks, service marks, or trade names of Oracle or Oracle's licensors is granted hereunder. Oracle, the Oracle logo, Java are trademarks or registered trademarks of Oracle USA, Inc. in the U.S. and other countries.

Disclaimer of Warranties

THE SPECIFICATION IS PROVIDED "AS IS" AND IS EXPERIMENTAL AND MAY CONTAIN DEFECTS OR DEFICIENCIES WHICH CANNOT OR WILL NOT BE CORRECTED BY ORACLE. ORACLE MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. ORACLE MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

Limitation of Liability

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ORACLE OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF ORACLE AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will hold Oracle (and its licensors) harmless from any claims based on your use of the Specification for any purposes other than the limited right of evaluation as described above, and from any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

Restricted Rights Legend

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

Report

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide Oracle with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Oracle a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

General Terms

Any action related to this Agreement will be governed by California law and controlling U.S. federal law. The U.N. Convention for the International Sale of Goods and the choice of law rules of any jurisdiction will not apply.

The Specification is subject to U.S. export control laws and may be subject to export or import regulations in other countries. Licensee agrees to comply strictly with all such laws and regulations and acknowledges that it has the responsibility to obtain such licenses to export, re-export or import as may be required after delivery to Licensee.

This Agreement is the parties' entire agreement relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, conditions, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification to this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

For example, the raw input `"\\u2122=\u2122"` results in the eleven characters `" \ \ u 2 1 2 2 = ™ "` (`\u2122` is the Unicode encoding of the character ™).

If an eligible `\` is not followed by `u`, then it is treated as a *RawInputCharacter* and remains part of the escaped Unicode stream.

If an eligible `\` is followed by `u`, or more than one `u`, and the last `u` is not followed by four hexadecimal digits, then the eligible `\`, and all the `u` characters that follow, are treated as *RawInputCharacter*s and remain part of the escaped Unicode stream. Unless these characters later become part of a raw string literal (§3.10.8), the syntactic grammar implies that a compile-time error must occur.

The character produced by a Unicode escape does not participate in further Unicode escapes.

For example, the raw input `\u005cu005a` results in the six characters `\ u 0 0 5 a`, because `005c` is the Unicode value for `\`. It does not result in the character Z, which is Unicode character `005a`, because the `\` that resulted from the `\u005c` is not interpreted as the start of a further Unicode escape.

Every token produced by the three lexical translation steps has a uniquely defined *raw preimage* which consists of exactly that contiguous sequence of raw Unicode characters which were the input to the first lexical translation step, giving rise to the token. The raw preimage of a token is significant only if that token is a raw string literal (§3.10.8).

In the previous example, the input `\u005cu005a` resulted in the six characters `\ u 0 0 5 a`, but its raw preimage is the original eleven characters `\ u 0 0 5 c u 0 0 5 a`.

The input `\u000D\u000A` will translate to a single line terminator (§3.4), but its raw preimage is the twelve characters `\ u 0 0 0 D \ u 0 0 0 A`.

The Java programming language specifies a standard way of transforming a program written in Unicode into ASCII that changes a program into a form that can be processed by ASCII-based tools. The transformation involves converting any Unicode escapes in the source text of the program to ASCII by adding an extra `u` - for example, `\uxxxx` becomes `\uuxxxx` - while simultaneously converting non-ASCII characters in the source text to Unicode escapes containing a single `u` each.

This transformed version is equally acceptable to a Java compiler and represents the exact same program. The exact Unicode source can later be restored from this ASCII form by converting each escape sequence where multiple `u`'s are present to a sequence of Unicode characters with one fewer `u`, while simultaneously converting each escape sequence with a single `u` to the corresponding single Unicode character.

A Java compiler should use the `\uxxxx` notation as an output format to display Unicode characters when a suitable font is not available.

3.4. Line Terminators

A Java compiler next divides the sequence of Unicode input characters into lines by recognizing *line terminators*.

```
LineTerminator:

  the ASCII LF character, also known as "newline"
  the ASCII CR character, also known as "return"
  the ASCII CR character followed by the ASCII LF character

InputCharacter:

  [UnicodeInputCharacter] [but not CR or LF]{.lineannotation}
```

Lines are terminated by the ASCII characters CR, or LF, or CR LF. The two characters CR immediately followed by LF are counted as one line terminator, not two.

A line terminator specifies the termination of the `//` form of a comment (§3.7).

The lines defined by line terminators may determine the line numbers produced by a Java compiler.

```
abstract    continue    for          new          switch
assert      default     if           package      synchronized
boolean     do          goto         private      this
break       double      implements   protected    throw
byte        else        import       public       throws
case        enum        instanceof   return       transient
catch       extends     int          short        try
char        final       interface    static       void
class       finally     long         strictfp     volatile
const       float       native       super        while
_ (underscore)
```

The keywords `const` and `goto` are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs.

While `true` and `false` might appear to be keywords, they are technically boolean literals (§3.10.3). Similarly, while `null` might appear to be a keyword, it is technically the null literal (§3.10.7).

A further ten character sequences are *restricted keywords*: `open`, `module`, `requires`, `transitive`, `exports`, `opens`, `to`, `uses`, `provides`, and `with`. These character sequences are tokenized as keywords solely where they appear as terminals in the *ModuleDeclaration* and *ModuleDirective* productions (§7.7). They are tokenized as identifiers everywhere else, for compatibility with programs written prior to Java SE 9. There is one exception: immediately to the right of the character sequence `requires` in the *ModuleDirective* production, the character sequence `transitive` is tokenized as a keyword unless it is followed by a separator, in which case it is tokenized as an identifier.

3.10. Literals

A *literal* is the source code representation of a value of a primitive type (§4.2), the `String` type (§4.3.3), or the null type (§4.1).

```
Literal:

  IntegerLiteral
  FloatingPointLiteral
  BooleanLiteral
  CharacterLiteral
  StringLiteral
  NullLiteral
  RawStringLiteral
```

3.10.1. Integer Literals

An *integer literal* may be expressed in decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2).

```
IntegerLiteral:

  DecimalIntegerLiteral
  HexIntegerLiteral
  OctalIntegerLiteral
  BinaryIntegerLiteral

DecimalIntegerLiteral:

  DecimalNumeral [IntegerTypeSuffix]

HexIntegerLiteral:

  HexNumeral [IntegerTypeSuffix]

OctalIntegerLiteral:

  OctalNumeral [IntegerTypeSuffix]
```

3.10.7. The Null Literal

The null type has one value, the null reference, represented by the *null literal* `null`, which is formed from ASCII characters.

```
NullLiteral:

  null
```

A null literal is always of the null type (§4.1).

3.10.8. Raw String Literals

The preimage of a *raw string literal* consists of one or more raw Unicode characters enclosed in matching *raw string quotes*, which are also composed of raw Unicode characters. There are no escape sequences or translations of any kind applied to any of these raw Unicode characters. The lexically translated characters of a raw string literal do not affect the value of the literal, although they may affect the tracking of source line boundaries.

```
RawStringLiteral:

  RawQuote RawStringBody RawQuote
  where the two raw-quotes are constrained to be identical

RawQuote:

  ` {`}
  where the preimage is constrained to be unescaped

RawStringBody:

  any non-empty output from the first and second lexical translation
  steps
```

The pre-images of the two matching RawQuotes must be identical, and may not contain escapes. The preimage of the RawStringBody is not allowed to contain any subsequence which is a copy of the RawQuote that introduced the RawStringLiteral. This one constraint is necessary to prevent the raw string token from continuing to the end of the source file.

To prevent ambiguities, a raw string body must not be null, nor may the raw string body begin with a ` ` ``{.literal} (backtick) character, as that would be indistinguishable from a longer RawQuote sequence.

As a consequence of these rules, a RawStringLiteral can contain any non-empty sequence of raw Unicode characters that does not begin with a ` ` ``{.literal} (backtick) character, and that does not contain a matching RawQuote sequence. A raw string body may contain any number of consecutive backtick characters (except at the very beginning), as long as the RawQuote sequence which introduces the raw string contains a *longer* sequence of consecutive backtick characters.

In practice, if the contents of a raw string must begin with a backtick, a padding character must be prepended to the raw string body separate it from RawQuote sequence. Such a padding character may be stripped by a method such as `trim`.

Because raw string literals exactly preserve every UTF-16 character originally in the source file, it follows that additional postprocessing methods, called on the `String` object, may accurately emulate the transformation steps which process unicode escapes and line terminators. They may also introduce other transformation steps, such as stripping vertical or horizontal spaces and other delimiters. The operation of such methods is not defined by the JLS.

The source of the UTF-16 characters present in the raw string body is not defined by this specification. Editing tools, file systems, and language processors may use any appropriate means to convey the original UTF-16 characters to the raw string. Note that UTF-32 characters cannot be directly supported in raw strings, although UTF-16 surrogates are allowed.

The following are examples of raw string literals:

```
`raw`                  // the three letters r a w
`x
```

```
`                       // x and a line terminator preimage
`\n`                    // the two characters \ n (not '\n')
`\u000a`                // the preimage \ u 0 0 0 a (not '\n')
`\u0060`                // the preimage \ u 0 0 6 0 (not `)
`\uvw`                  // the preimage \ u v w
`\(.\)\1`               // the seven characters \ ( . \ ) \ 1
`` ` ``                   // a string containing SP ` SP
``                      // error; string may not be empty
` x `y `                // error; string ends before 'y'
\u0060x`                // error; RawQuote must not be escaped
```

A raw string literal is always of type `String` (§4.3.3). Its value is the preimage of the RawStringBody, which includes the pre-images of any Unicode escapes or line terminators.

3.11. Separators

Twelve tokens, formed from ASCII characters, are the *separators* (punctuators).

Separator:

(one of)

```
(   )   {   }   [   ]   ;   ,   .   ...   @   ::
```

3.12. Operators

38 tokens, formed from ASCII characters, are the *operators*.

Operator:

(one of)

```
=   >   <   !   ~   ?   :   ->
==  >=  <=  !=  &&  ||  ++  --
+   -   *   /   &   |   ^   %   <<   >>   >>>
+=  -=  *=  /=  &=  |=  ^=  %=  <<=  >>=  >>>=
```