Q) Below is the line that builds the path of a directory in java.

Here, File.separator is "\" on windows and "/" on Unix.

String path = System.getProperty("user.home") + File.separator + "workspace" +

File.separator + "JavaCode";

If i hardcode the path, it should look as shown below:

File path = new File("C:\\users\\david\\workspace\\JavaCode");

My question:

Why do we mention \\ in second case?

1. Because, with a string, \ is an escape character: it says to interpret the following character in a special way (which is why \n isn't an n). In your case, you want \ itself to be interpretted specailly by nottreating it special, so you need 2 of them: the first says "treat the next character special", the next gets treated specially for a*\*./ doesn't need to be escaped

All Unicode characters can be used in comments, character and string literals in java. Unicode characters can be expressed through Unicode Escape Sequences.

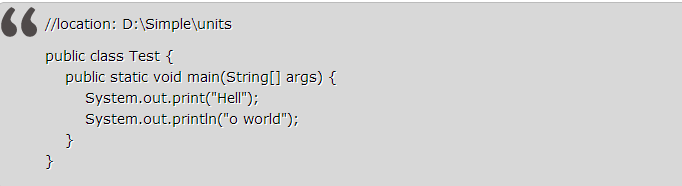
Unicode escape sequences consist of

* a **backslash '\'** (ASCII character 92, hex 0x5c),
* a **'u'** (ASCII 117, hex 0x75)
* optionally one or more additional 'u' characters, and
* **four hexadecimal digits** (the characters '0' through '9' or 'a' through 'f' or 'A' through 'F').

Such sequences represent the UTF-16 encoding of a Unicode character, for example, 'a' is equivalent to '\u0061'. This escape method does not support characters beyond U+FFFF or you have to make use of surrogate pairs.

**Unicode escape sequence may appear anywhere in a Java source file** including inside identifiers, comments, and string literals. **Unicode escapes must be always well formed, even if they appear in comments, else compiler will complain.**It is legal to place a well-formed Unicode escape in a comment. Programmers sometimes use Unicode escapes in Javadoc comments to generate special characters in the documentation.

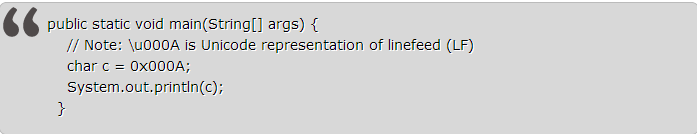
Consider **an example**. Predict the output of the below program:



**Compilation will fail.** Unicode escapes must be well formed, even if they appear in comments**.**Thecomment //location: D:\Simple\apps**\u**nits will throw a compilation error as **\u** is not followed by four hexadecimal digits. To avoid trouble like this, we must not put Windows filenames into comments in generated Java source files without first processing them to eliminate backslashes.

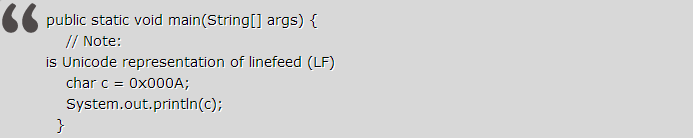
**The compiler translates Unicode escapes into the characters they represent before it parses a program into tokens. It also does so before discarding comments and white space.**

Let us consider **another example:**



**It won't compile.** You will get an error like ';' expected. This program contains a single Unicode escape (\u000A), located in its comment. As the comment tells you, this escape represents the linefeed character, and the compiler translates it before discarding the comment. Unfortunately, this linefeed character is the first line terminator after the two slash characters that begin the comment (//) and so terminates the comment. The words following the escape (is Unicode representation of linefeed (LF)) are therefore not part of the comment; nor are they syntactically valid.

So the above example effectively becomes:



**Any and all characters in a program may be expressed in Unicode escape characters, but such programs are not very readable, except by the Java compiler.** You can write a complete program as Unicode characters. Consider an example .java file contents:

\u0070\u0075\u0062\u006c\u0069\u0063 \u0063\u006c\u0061\u0073\u0073\u0020\u0054\u0065\u0073\u0074  
\u007b\u007d

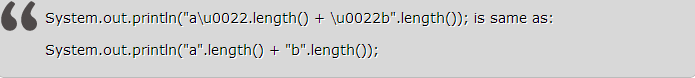
This will compile fine if the file name is Test.java as the above code is same as:  
public   
class Test  
{}

**Java provides no special treatment for Unicode escapes within string literals.**

Consider another example:If \u0022 is the Unicode escape for double quote ("), then what will the below line print:



**It will print 2.** Java provides no special treatment for Unicode escapes within string literals. The compiler translates Unicode escapes into the characters they represent before it parses the program into tokens, such as strings literals. Therefore, the first Unicode escape in the program closes a one-character string literal ("a"), and the second one opens a one-character string literal ("b"). The program prints the value of the expression "a".length() + "b".length(), or 2.



If you wanted to put the two double quote chars into the string literal, you can do it with normal escape sequences. But you can't do with Unicode escapes because Java provides no special treatment for Unicode escapes within string literals. Using normal escape sequence we can write the above as **"a\".length() + \"b".length()** which will print 16.

**Avoid Unicode escapes except where they are truly necessary.** They are rarely necessary. Unicode escapes are essential when you need to insert characters that can't be represented in any other way into your program. Avoid them in all other cases. Unicode escapes reduce program clarity and increase the potential for errors.

Most Java program text consists of [ASCII](http://en.wikipedia.org/wiki/ASCII) characters, but any Unicode character can be used as part of identifier names, in comments, and in character and string literals. For example, π (which is the Greek Lowercase Letter **pi**) is a valid Java identifier:

|  |  |
| --- | --- |
| Example | **Code section 3.100: Pi.**   1. **double** π = **Math**.PI; |

and in a string literal:

|  |  |
| --- | --- |
| Example | **Code section 3.101: Pi literal.**   1. **String** pi = "π"; |

Unicode escape sequences[[edit](http://en.wikibooks.org/w/index.php?title=Java_Programming/Unicode&action=edit&section=1" \o "Edit section: Unicode escape sequences)]

Unicode characters can also be expressed through Unicode Escape Sequences. Unicode escape sequence may appear anywhere in a Java source file (including inside identifiers, comments, and [string literals](http://en.wikibooks.org/wiki/Java_Programming/Literals)).

Unicode escape sequences consist of

1. a backslash '\' (ASCII character 92, hex 0x5c),
2. a 'u' (ASCII 117, hex 0x75)
3. optionally one or more additional 'u' characters, and
4. four hexadecimal digits (the characters '0' through '9' or 'a' through 'f' or 'A' through 'F').

Such sequences represent the UTF-16 encoding of a Unicode character. For example, 'a' is equivalent to '\u0061'. This escape method does not support characters beyond U+FFFF or you have to make use of surrogate pairs.[[1]](http://en.wikibooks.org/wiki/Java_Programming/Unicode#cite_note-1)

Any and all characters in a program may be expressed in Unicode escape characters, but such programs are not very readable, except by the Java compiler! They are not compact either!

One can find a full list of the characters [here](http://en.wikibooks.org/wiki/Unicode/Character_reference).

π may also be represented in Java as the *Unicode escape sequence* \u03C0. Thus, the following is a valid, but not very readable, declaration and assignment:

|  |  |
| --- | --- |
| Example | **Code section 3.102: Unicode escape sequences for Pi.**   1. **double** \u03C0 = **Math**.PI; |

The following demonstrates the use of Unicode escape sequences in other Java syntax:

|  |  |
| --- | --- |
| Example | **Code section 3.103: Unicode escape sequences in a string literal.**   1. *// Declare Strings pi and quote which contain \u03C0 and \u0027 respectively:* 2. **String** pi = "**\u**03C0"; 3. **String** quote = "**\u**0027"; |

Note that a Unicode escape sequence functions just like any other character in the source code. E.g., \u0022 (double quote, ") needs to be quoted in a string just like ".

|  |  |
| --- | --- |
| Example | **Code section 3.104: Double quote.**   1. *// Declare Strings doubleQuote1 and doubleQuote2 which both contain " (double quote):* 2. **String** doubleQuote1 = "**\"**"; 3. **String** doubleQuote2 = "**\\**u0022"; *// "\u0022" doesn't work since """ doesn't work.* |

International language support[[edit](http://en.wikibooks.org/w/index.php?title=Java_Programming/Unicode&action=edit&section=2)]

The language distinguishes between bytes and characters. Characters are stored internally using UCS-2, although as of J2SE 5.0, the language also supports using UTF-16 and its surrogates. Java program source may therefore contain any Unicode character.

The following is thus perfectly valid Java code; it contains Chinese characters in the class and [variable](http://en.wikibooks.org/wiki/Java_Programming/Variables) names as well as in a string [literal](http://en.wikibooks.org/wiki/Java_Programming/Literals):

|  |  |
| --- | --- |
| Computer code | **Code listing 3.50: 哈嘍世界.java**   1. **public** **class** 哈嘍世界 { 2. **private** **String** 文本 = "哈嘍世界"; 3. } |