Section Seven – Java Strings

In this section the Developer will learn…

- How to manipulate text in Java

Java Strings - Introduction

This section will discuss developing Strings in Java, moving on from the simple "Hello" used in previous examples.

Remember that Strings are always enclosed by double quotes.

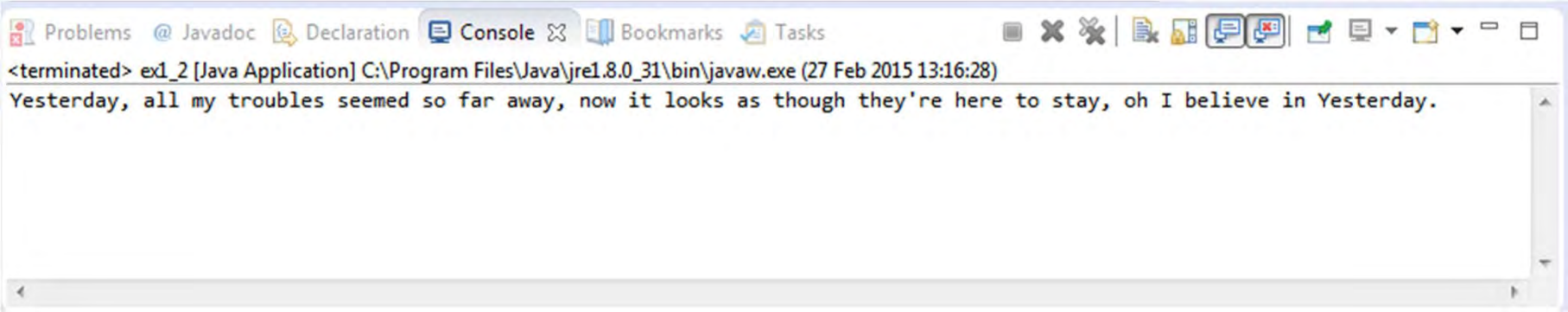
The text used in these examples are termed 'literals', later in the course we will use the String datatype which is an object.

Both types, literal and object can be manipulated using Java functions and these will be demonstrated in this section.

Java Strings – Using Escape Characters

The following output doesn't use escape characters and the Developer cannot predict how the line will appear …

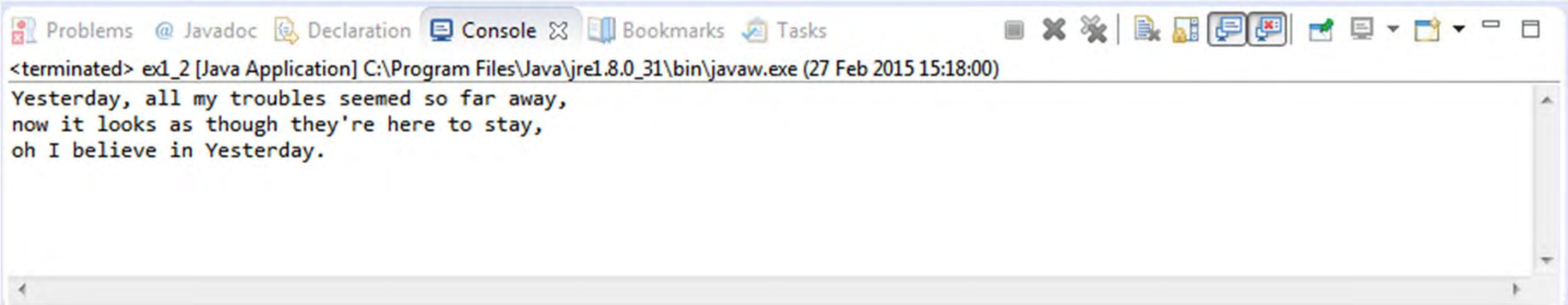
System.out.println("Yesterday, all my troubles seemed so far away, now it looks as though they're here to stay, oh I believe in Yesterday.");



Java Strings – Using Escape Characters

By using the escape character 'n' (newline) preceded by a backslash to signify it is an escape character, the Developer can force a line break …

*System.out.println("Yesterday, all my troubles seemed so far away,**n***now it looks as though they're here to stay,**n***oh I believe in Yesterday.");*



Java Strings – Using Escape Characters

The following Escape characters are available …

\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage return
\s	Space
\t	Tab
\'	Single quote
\"	Double quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

Java Strings – Concatenation of Strings

The previous example although it worked perfectly well, did have one problem in that the literal was very long therefore very untidy, (we recommend a limit of 80 characters per line in the code) …

```
6    {
7
8        System.out.println("Yesterday, all my troubles seemed so fa
9
10   }
11}
```

One method to get round this problem is to concatenate smaller literals together using the plus (+) sign, this will allow the value to run over more than one line …

```
System.out.println("Yesterday, all my troubles seemed so far away,\n"
                   +"now it looks as though they're here to stay,\n"
                   +"oh I believe in Yesterday.");
```

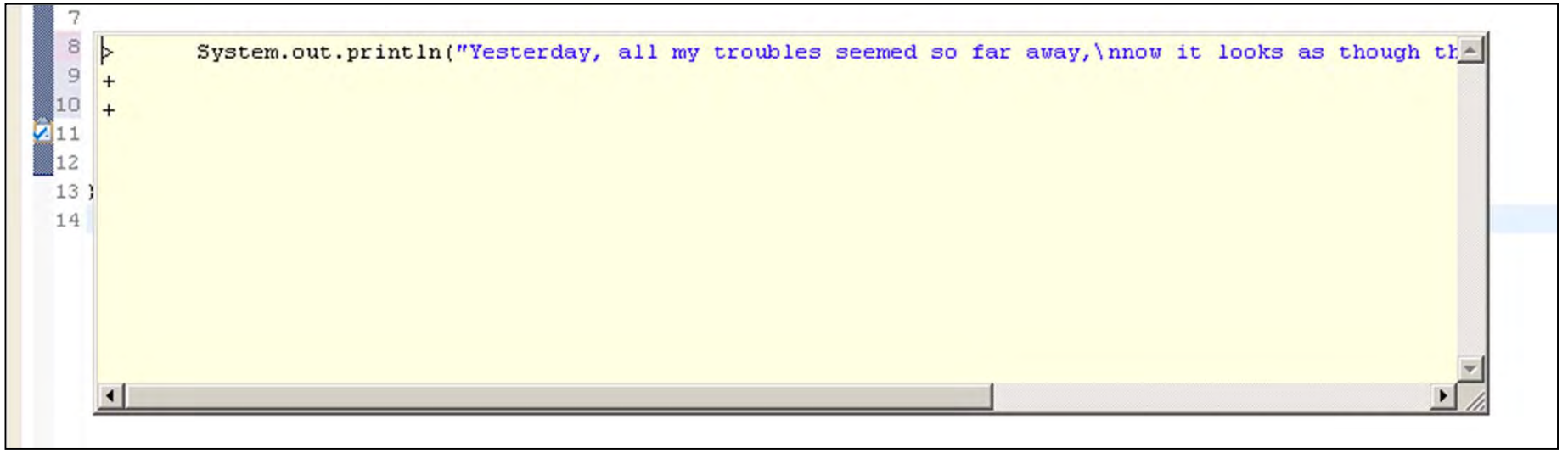
Java Strings – Concatenation of Strings

System.out.println("Yesterday, all my troubles seemed so far away,\n"
+"now it looks as though they're here to stay,\n"
+"oh I believe in Yesterday.");

Although the output will remain the same, because the program itself is spread over several lines it makes the code far easier to read. Remember to put in spaces if joining literals but not putting in line returns etc ….

```
7
8       System.out.println("Yesterday, all my troubles seemed so far away,\n"
9                          +"now it looks as though they're here to stay,\n"
10                         +"oh I believe in Yesterday.");
11
```

Java Strings – Concatenation of Strings



Interestingly, Eclipse views the concatenated lines as one line and will group them together when the Developer runs the mouse to the left hand side, in addition the Editor colours the left hand edge to signify they belong together …

```
7
8       System.out.println("Yesterday, all my troubles seemed so far away,\nnow it looks as though th
9   +
10  +
11
12
13  }
14
```

Java Strings – Using Methods with Strings

The following examples deal with literals and manipulating their values, the examples given can just as easily been applied to String objects or other appropriate objects.

When using a method on an object, dot notation is employed, dot notation is used to signify different levels of methods, in these first examples it is used to separate the two components of the process, on the left the literal or object to be manipulated, on the right the method which does the manipulation …

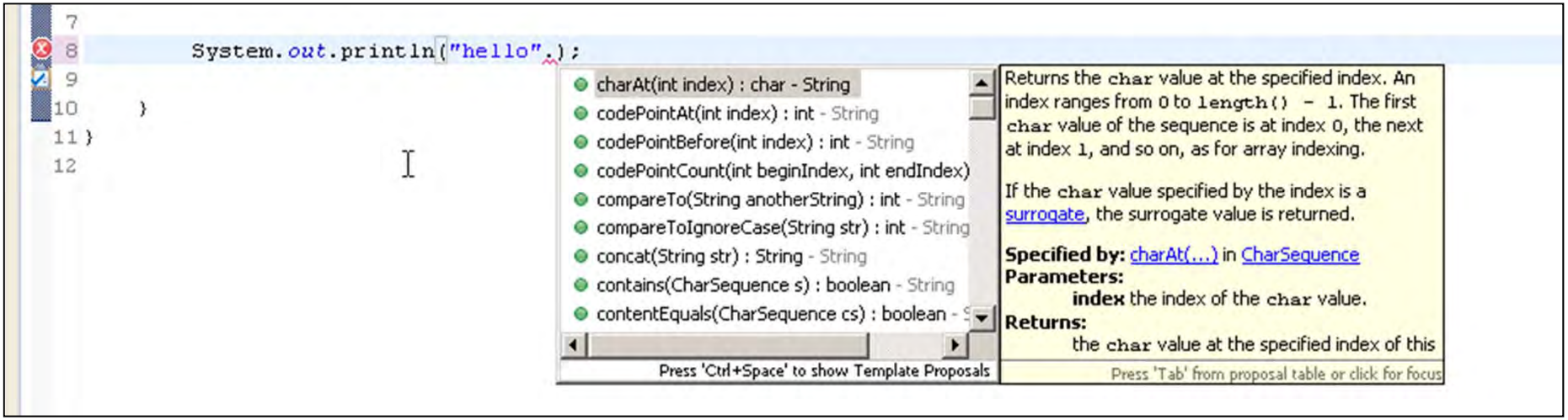
*System.out.println("Hello".**toUpperCase());***

This will output ***HELLO***, the opposite is ***toLowerCase()***

Java Strings – Using the Java Editor with Methods

Eclipse will attempt to 'second guess' what method the Developer requires by listing all available methods in a list, the more letters typed the less displayed, and ignore the red cross for the time being – you are typing !!

One character will display lots of methods, note the description next to the current selected method …



Java Strings – Using the Java Editor with Methods

The more characters the less methods displayed …

```
 7
 8        System.out.println("hello".to);
 9
10    }
11 }
12
```

toCharArray() : char[] - String
toLowerCase() : String - String
toLowerCase(Locale locale) : String - String
toString() : String - String
toUpperCase() : String - String
toUpperCase(Locale locale) : String - String
toarray - convert collection to array

Converts this string to a new character array.
Returns:
a newly allocated character array whose length is the length of this string and whose contents are initialized to contain the character sequence represented by this string.

Press 'Ctrl+Space' to show Template Proposals
Press 'Tab' from proposal table or click for focus

```
 7
 8        System.out.println("hello".tou);
 9
10    }
11 }
12
```

toUpperCase() : String - String
toUpperCase(Locale locale) : String - String

Converts all of the characters in this `String` to upper case using the rules of the default locale. This method is equivalent to `toUpperCase(Locale.getDefault())`.

Note: This method is locale sensitive, and may produce unexpected results if used for strings that are intended to be interpreted locale independently. Examples are programming language identifiers, protocol keys, and HTML tags. For instance, `"title".toUpperCase()` in a Turkish locale returns `"T\u0130TLE"`, where '\u0130' is the LATIN CAPITAL LETTER I WITH DOT ABOVE

Press 'Ctrl+Space' to show Template Proposals
Press 'Tab' from proposal table or click for focus

Java Strings – Using the Java Editor with Methods

As soon as the required method is found the Developer can double click on it and the Editor will complete the spelling including the correct 'case' even if the Developer were typing in lower case, note the empty brackets following the method …

```
 7
 8         System.out.println("hello".toUpperCase());
 9
10     }
11 }
12
```

Java Strings – Methods with Parameters / Arguments

Many methods have been developed to accept arguments, that is, parameters which greatly increase the flexibility of the method, these values are passed to the method within the brackets (which are always present), in this example the Developer has chosen the *replace()* method and when double-clicked the following structure is inserted into the code …

```
7
8          System.out.println("hello".replace(target, replacement));
9
10     }
```

target and *replacement* are hints to the Developer for the required values, separated by commas, they are not accepted values (hence the red cross) and the Developer must change them …

Java Strings – Methods with Parameters / Arguments

This will replace the 'e' with an 'o' …

*System.out.println("hello".replace("**e**", "**o**"));*

hollo

The double quotes in the replace method can be changed to single quotes if required …

*System.out.println("hello".replace('**e**', '**o**'));*

Java Strings – Selecting Substrings

The *substring()* method will allow the Developer to select a portion of the string. Please note that Java starts the count with 0 not 1 as in some other programming languages, the value given represents the start and end position (not character places) …

*System.out.println("Seer Computing Ltd".**substring**(0, 4));*

Seer

Leaving the end position off will display all characters right of the start position.

System.out.println("Seer Computing Ltd".substring(5));

Computing Ltd