Code point and supplementary characters

Each character of primitive type **char** is implemented as a 16-bit integer in the Unicode system "UTF-16". That integer is called the *code point* for that character. You can see the code point for a **char** by casting it to type **int**. For example, the expression

(**int**) 'a' evaluates to the code point 97.

In Java, you can also write a character in type **char** using the hexadecimal representation of its code point. The hexadecimal representation of decimal 97 is 61, so one can write 'a' as '\u0061':

(**int**) '\u0061' evaluates to the code point 97.

Unicode encodes the characters of almost all the world's writing systems, including Greek, Arabic, Hebrew, and Asian languages like Telugu, Korean, Japanese, and Chinese languages. Most characters are represented in 2 bytes (16 bits) and are values in primitive type **char**. But there are now so many characters that they don't all fit into the 16-bit format, and a some require a 2 16-bit parts. This means that some characters cannot be included in type **char** even though they can be used in Strings, just as all integers are not in type **int**.

There are other complications involving all possible characters, which we don't go into here.

Below, we just want to show one effect of having characters that don't appear in primitive type **char**. However, note that this is a rarity, and in any of the Java programs you will be writing, this shouldn't be an issue.

Many emojis are characters but are not in char

The Java code to the right shows four emojis that are Unicode characters. Execution of this code prints:

four emojis: 😀🙏👍❤️

```
String emos= "😀🙏👍❤️";
System.out.println("four emojis: " + emos);
```

However, the statement

char c= '👍';

is illegal —Java says that the character constant is invalid. In fact, the first three emojis are *not* in type **char**, but the heart *is* in type **char**. The first three emojis are *supplemental characters*, requiring two 16-bit entities. To the right, we show how to write these emojis in a String in terms of the two hexadecimal values and also give the code points for them.

	characters	code point
😀	"\uD83D\uDE00"	128512
🙏	"\uD83D\uDE4F"	128591
👍	"\uD83D\uDC4D"	128007
❤️	"\u2665"	9829

Now, note that execution of the statement to the right prints

emos length: 7

```
System.out.println("emos length: " + emos.length());
```

This is because the first three emojis require 2 char positions and the last one, the heart, requires one. This means that if you are going to use supplemental characters, you have to be extremely careful in manipulating strings: the length is not the number of characters!