Java and Unicode

The confusion about String and char in Java

What is Unicode?

At one level, Unicode is a standard for the encoding, representation and handling of text on computers.

It defines some 136,755 “characters” (and counting) for more than 139 language script systems and a rich symbol set.

The standard is maintained by “ISO/IEC 10646”.

What is a Java char?

> *The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).*
>
> Source: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

How can you store 136,755 characters in a datatype which is only 16 bits large (65,536 values)?

You can't. OK, so how do they do it?

How to encode a Unicode character › char.max

The answer is that you store a large Unicode character value in two chars.

One with a lower value and one with an upper value, a so called *surrogate pair.*

This encoding scheme is called UTF-16.

Difference between charset and encoding

A *coded character set* is simply a numbered ordered set of characters, or a table of characters, each with a unique number. Put differently, a function from a character to a code point (a number).

Examples: ASCII, ISO-8859-1, Unicode

A *character encoding form* is the way a computer represents a code point digitally with a fixed length.

A *character encoding scheme* is the way a computer represents code points as a sequence of octets (bytes), for instance UTF-16.

A Java character

A Java character is represented by a 16 bit number. However, the code points of Unicode is much bigger, so sometimes two 16 bit numbers are needed.

This allows us to represent much more characters (and symbols) than would fit in a 16 bit character set (represented by, e.g. a Java char datatype).

For instance, the character “Bomb”: 💣 can be represented in Java, but not stored in one single char value.

Example: representing 💣

The decimal code for 💣 is 128163. But the largest value which fits in a char is 65535.

In hex, the code for the bomb is 0x1F4A3. So, in Java, one can represent that in a few ways:

```
String bomb = new StringBuilder().appendCodePoint(0x1F4A3).toString();
```

So, the argument to appendCodePoint() in this case is an int:

```
int c = 0x1F36F;
```

Example: representing 💣

Another way to represent 💣 in Java is:

```
String test = "💣";
```

This is because Java supports UTF directly in the source code.

The int value of the bomb doesn’t fit inside the Java char type. So how long is that String?

Example: representing 💣

Another way to represent 💣 in Java is:

```
String test = "💣";
```

This is because Java supports UTF directly in the source code.

The int value of the bomb doesn’t fit inside the Java char type. So how long is that String?

The answer is 2! It takes two chars inside the String to represent it.

The two chars making up the bomb

First part: 0xD83D (high surrogate)

Second part: 0xDCA3 (low surrogate)

So, how did Java turn 0x1F4A3 into those two parts?

UTF-16 encoding a character outside 16 bits

Another fun character is 💩 (the turd). It has value 128169 or 0x1F4A9 in hex.

```
// UTF-16 coding of a large in as two 16 bit values:
//        c is 11111010010101001 (17 bits)

// 1. subtract 0x010000
c -= 0x010000; // Keep 16 bits, get rid of the highest bit!
// We now have: 1111010010101001
// 2. keep the highest 10 bits, and add 0xD800 in the high surrogate
int high = (c >> 10) + 0xD800;
// First:       1111010010101001 >> 10 =     111101
//
// Then:                                     111101
//                                 1101100000000000+ (0xD800)
//                                 -----------------
//                                 1101100000111101  (0xD83D) -> high
```

UTF-16 encoding a character outside 16 bits

```
// 3. keep the lowest 10 bits, and add 0xDC00 in the low surrogate
int low  = (c & 0x3FF) + 0xDC00;

//     1111010010101001
//     0000001111111111 &
//     ----------------
//           0010101001

// Add 0xDC00:
//
//           0010101001
//     1101110000000000 +
// --------------------
//     1101110010101001 (0xDCA9)
```

UTF-16 encoding a character outside 16 bits

```
// Now we have high surrogate 0xD83D and Low surrogate 0xDCA9

// We can construct a String this way:
String turd  = new String(new int[] {0xD83D, 0xDCA9}, 0, 2);

// OR:
String turd2 = new String(new int[] {0x1F4A9},        0, 1);

// It’s the same thing!

// The former is using surrogates - UTF-16 coded 16 bit values
// and the latter is using an int.

// The arguments tells String’s constructor how many code points
// are used.
```

UTF-16 encoding a character outside 16 bits

```
// We can construct a String this way:
String turd  = new String(new int[] {0xD83D, 0xDCA9}, 0, 2);
String turd2 = new String(new int[] {0x1F4A9},        0, 1);
// It’s the same thing!

// What do you think the String class does with the int version?
public static final char MIN_HIGH_SURROGATE = '\uD800';
public static final char MIN_LOW_SURROGATE  = '\uDC00';

public static char highSurrogate(int codePoint) {
  return (char) ((codePoint >>> 10)
    + (MIN_HIGH_SURROGATE - (MIN_SUPPLEMENTARY_CODE_POINT >>> 10)));
}
public static char lowSurrogate(int codePoint) {
  return (char) ((codePoint & 0x3ff) + MIN_LOW_SURROGATE);
}
```

What to take away from all this

- A Java String might contain characters wider than one char
- Java uses UTF-16 to encode such characters as two chars
- It is a standard encoding
- Java allows UTF directly in the source code
- The length of a String is not always the same as number of characters
- You can construct Strings with special characters in various ways
  - `new StringBuilder().appendCodePoint(0x1F4A3).toString();`
  - `String turd  = new String(new int[] {0xD83D, 0xDCA9}, 0, 2);`
  - `String turd2 = new String(new int[] {0x1F4A9},        0, 1);`