Set 12: Unicode and the char Data Type

Skill 12.01: Explain the difference between ASCII and Unicode
Skill 12.02: Declare a *char* data type
Skill 12.03 Explain the variable compatibility between chars, ints, and String
Skill 12.04 Convert between *char* and *String* data types
Skill 12.05 Convert *char* variables to lower and upper case

Skill 12.01: Explain the difference between ASCII and Unicode

Skill 12.01 Concepts

Unicode

Unicode is a computer encoding methodology that assigns a unique number for every character. It doesn't matter what language, or computer platform it's on. This is important in a global, networked world, and for computer systems that must accommodate multiple languages and special characters. Unicode truly unifies all of these into a single standard.
In Unicode, all characters are represented by numeric values. For example, 65 is the letter A. 66 is B and so on. The lower-case letters start at 97. There are even special system-only characters in the list: carriage returns, tabs, etc. These can be useful in displaying text; while others are leftovers from older systems.

ASCII

ASCII, like unicode, uses numbers to represent symbols and letters. The main difference between the two is in the way they encode the character and the number of bits that they use for each. ASCII originally used seven bits to encode each character. This was later increased to eight with Extended ASCII to address the apparent inadequacy of the original.
With the exansion of ASCII combined many non-standard extended ASCII programs began to emerge. To reconcile the different systems, unicode was adpoted.

Advantages of Unicode over ASCII

Unicode uses a variable bit encoding program where you can choose between 32, 16, and 8-bit encodings. Using more bits lets you use more characters at the expense of larger files while fewer bits give you a limited choice but you save a lot of space. Using fewer bits (i.e. UTF-8 or ASCII) would probably be best if you are encoding a large document in English.

Unicode offers many advantages over ASCII. Variable bit encoding allows for the accommodation of a huge number of characters. Because of this, Unicode currently contains most written languages and still has room for even more. This includes typical left-to-right scripts like English and even right-to-left scripts like Arabic. Chinese, Japanese, and the many other variants are also represented within Unicode. So Unicode won't be replaced anytime soon.

Skill 12.02: Declare a *char* data type

Skill 12.02 Concepts

In Java, *char* is short for character. It is 16 bits in size - that is it can only take up 16 places in memory.

Let's review the place values in binary to understand the maximum value allowed for the *char* data type. Below, are the place values for the first 16 binary places. According to the table the largest number value we can store in a *char* data type is 32767.

2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Most of the time, the actual data stored in the *char* data type doesn't take up more than 8 bits; the reason Java allows 16 bits is so that all characters in all languages can be represented. This representation is in the Unicode format described previously.

Declaring *char* variable types

You'd think that a *char* could be any value from a to Z, and all the numbers. That is true, except for one key item. You can either declare a char variable with single quotes, e.g., setting a *char*'s value to the letter 'a'. Or, you could omit the quotes, and set the Unicode representation of the value. Take a look at the following code for declaring a *char* variable equal to 77.

Code	**Output**	**Explanation**

`char seventySeven = 77;`
`System.out.println(seventySeven);`
	`M`	We can determine the Unicode equivalent of a number by assigning it to a char datatype and printing it.

The *char* data type can also be assigned the value of *M* using single quotes,

Code	**Output**	**Explanation**

`char seventySeven = 'M';`
`System.out.println(seventySeven);`
	`M`	Only a single character can be assigned to a char data type and must be enclosed in single quotes.

Skill 12.02 Exercise 1

Skill 12.03 Explain the variable compatibility between chars, ints, and String

Skill 12.03 Concepts

Compatibility between *char* and *String* data types

char data types and String types cannot be stored into each other.

In the example above, if we had tried to declare the *char* as the actual "77", with quotes, there would be an error - *incompatible types: String cannot be converted to char*

Code	Output	Explanation
`char seventySeven = "77";`	`Error`	Double quotes are reserved for String data types and cannot be used to store char values

Recall, that char variables must be declared with single quotes, but the following code also produces an error, because there is no unicode equivalent for the character '77'.

Code	Output	Explanation
`char seventySeven = '77';`	`Error`	Only a single character is allowed in single quotes. There is no Unicode equivalent for 77

<u>Compatibility between *char* and *int* data types</u>

char data types can be assigned to *int* data types. This is possible because *char* data types have a Unicode (*int*) equivalent.

The example below illustrates how the Unicode (*int*) equivalent of any symbol can be found using the *char* and *int* data types.

Code	Output	Explanation

`char seventySeven = 'M';`
`int M = seventySeven;`
`System.out.println(M);`
	`77`	`Prints the Unicode equivalent of M, which is 77`

The above illustrates how the Unicode equivalent of a symbol or letter can be found by assigning the *char* data type to an *int*. The reverse process however is not allowed. Consider the following example which results in a *possible lossy conversion from int to char* error.

Code	Output	Explanation

`int unicode = 12345678;`
`char symbol = unicode;`
`System.out.println(symbol);`
	`Error`	An *int* data type cannot be stored in a *char* data type. *int* data types are allowed up to 4 bytes of storage, whereas *char* data types are only allowed 2 bytes.

The reason why the above is illegal is because *char* variables can take on Unicode values up to 2^{16} - 1 while *int* variables can go 2^{32} - 1. The compiler justly complains about “possible loss of precision” and refuses to do it.

The process can be overwritten by casting the int as a char,

Code	Output	Explanation

`int unicode = 12345678;`
`char symbol = (char)unicode;`
`System.out.println(symbol);`
	慎	We can force an int to be stored as char by casting it.

<u>Skill 12.03 Exercise 1</u>

Skill 12.04 Convert between *char* and *String* data types

Skill 12.04 Concepts

A String can be converted to a char using the *charAt* method. This is illustrated below,

Code	Output	Explanation
`String wString = "W";`	W	The charAt method gets the character at a specific index. Assigning the character to an int gets the Unicode equivalent of the character.
`char wChar = wString.charAt(0);`	87	
`System.out.println(wChar);`		
`int wUnicode = wChar;`		
`System.out.println(wUnicode);`		

Skill 12.04 Exercise 1

Skill 12.05 Convert *char* variables to lower and upper case

Skill 12.05 Concepts

The symbols for the upper case letters in our alphabet begin at number 65 in the Unicode systems. The symbols for the lower case letters begin at number 97. The letters of our alphabet and the corresponding Unicode values are shown below.

char	Unicode		char	Unicode
A	65		a	97
B	66		b	98
C	67		c	99
D	68		d	100
E	69		e	101
F	70		f	102
G	71		g	103
H	72		h	104
I	73		i	105
J	74		j	106
L	76		l	108
M	77		m	109
N	78		n	110
O	79		o	111
P	80		p	112
Q	81		q	113
R	82		r	114
S	83		s	115
T	84		t	116
U	85		u	117
V	86		v	118
W	87		w	119

X	88		x	120
Y	89		y	121
Z	90		z	122

Notice that there is numerical difference of 32 between all the uppercase and lowercase equivalents. This enables for the easy conversion between uppercase and lower case *char* values,

Code	**Output**	**Explanation**

`char bigLetter = 'H';`
`char smallLetter = (char)(bigLetter + 32);`
`System.out.println(smallLetter);`
	h	Adding 32 to H converts the char to its lower case equivalent, h

Skill 12.05 Exercise 1