SchoolNova

IT101

Characters: from ASCII to Unicode

Java Primitives

Name	Data	Range	Default Value	Size
byte	signed integer	[-128, 127]	0	8 bits
short	signed integer	[-32768, 32767]	0	16 bits
int	signed integer	[-2147483648, 2147483647]	0	32 bits
long	signed integer	[-9223372036854775808, 9223372036854775807]	0	64 bits
float	floating-point	**MIN:** ±1.4E-45 **MAX:** ±3.4028235E+38	0.0	32 bits
double	floating-point	**MIN:** ±4.9E-324 **MAX:** ±1.79769313486231570E+308	0.0	64 bits
char	Unicode	['\u0000', '\uFFFF']	'\u0000'	16 bits
boolean	logical value	{false, true}	false	≥ 1 bit

- Note the “char” (character) primitive. How does it represent the alphabet letters?
- What is the difference between “char” and “String”? Does a “String” consist of “char”s?

Characters: Historical Perspective

- Back in the early 1960s there was no character representation standard. Hence computer manufacturers were doing things pretty much any way they saw fit to do. This resulted in the following situation:
  - Different computers had no way of communicating with each other.
  - Each manufacturer had their own ways of representing the letters of the alphabet, numbers and the likes.
  - There were over 60 different ways, at the time, to represent these letters, numbers and any specific symbols.
  - I.B.M. with it's different equipment had 9(!) different character sets.
- Bob Bemer worked for IBM and played an important role in the establishment of the first character representation standard: ASCII.

ASCII

- American Standard Code for Information Interchange
- ASCII was able to represent every English character using a number between 32 and 127. Space was 32, the capital letter "A" was 65, capital letter “S” was 83 (binary representation: 1010011) etc.
- This could conveniently be stored in 7 bits. Most computers in those days were using 8-bit bytes, so not only could they store every possible ASCII character, but they had a whole one bit to spare, which they could use for their own devious purposes.

USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Column / Row →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

http://en.wikipedia.org/wiki/ASCII

Problem with ASCII

- All was good, assuming you were an English speaker, but ASCII started becoming popular with the advent of first IBM Personal Computer in 1981.
- Because bytes have room for up to eight bits, lots of people got to thinking, "gosh, we can use the codes 128-255 for our own purposes." The trouble was, lots of people had this idea at the same time, and they had their own ideas of what should go where in the space from 128 to 255.
- As soon as people started buying PCs outside of America all kinds of different character sets were dreamed up, which all used the top 128 characters for their own purposes. For example on some PCs the character code 130 would display as é, but on computers sold in Israel it was the Hebrew letter Gimel (ג), so when Americans would send their résumés to Israel they would arrive as rגsumגs. In many cases, such as Russian, there were lots of different ideas of what to do with the upper-128 characters, so you couldn't even reliably interchange Russian documents.
- Eventually this free-for-all got codified in the ANSI (American National Standards Institute) standard. In that standard, everybody agreed on what to do below 128, which was pretty much the same as ASCII, but there were lots of different ways to handle the characters from 128 and on up, depending on where you lived. These different systems were called code pages or character encoding.
- Until 1990-ies a byte (8 bits) was a character and as long as you never moved a string from one computer to another, or spoke more than one language, it would sort of always work. But as soon as the Internet happened, it became quite commonplace to move strings from one computer to another, and the whole mess came tumbling down. Luckily, Unicode was invented.

Unicode

- Unicode was a brave effort to create a single character set that included every reasonable writing system on the planet.
- Unicode has a different way of thinking about characters. Until now, we've assumed that a letter maps to some bits which you can store on disk or in memory: A -> 0100 0001 (65)
- In Unicode, a letter maps to something called a code point which is still just a theoretical concept. How that code point is represented in memory or on disk is another story.
- Every conceivable letter in every alphabet is assigned a magic number by the Unicode consortium which is written like this: U+0639. The U+ means "Unicode" and the numbers are hexadecimal. U+05E2 is the Hebrew letter Ayin. The English letter A would be U+0041. You can find them all using the charmap utility on Windows 2000/XP or visiting the Unicode web site (http://www.unicode.org/charts/).
- There is no real limit on the number of letters that Unicode can define.
- Various contemporary character encodings implement the Unicode standard. The most popular of them is UTF-8.

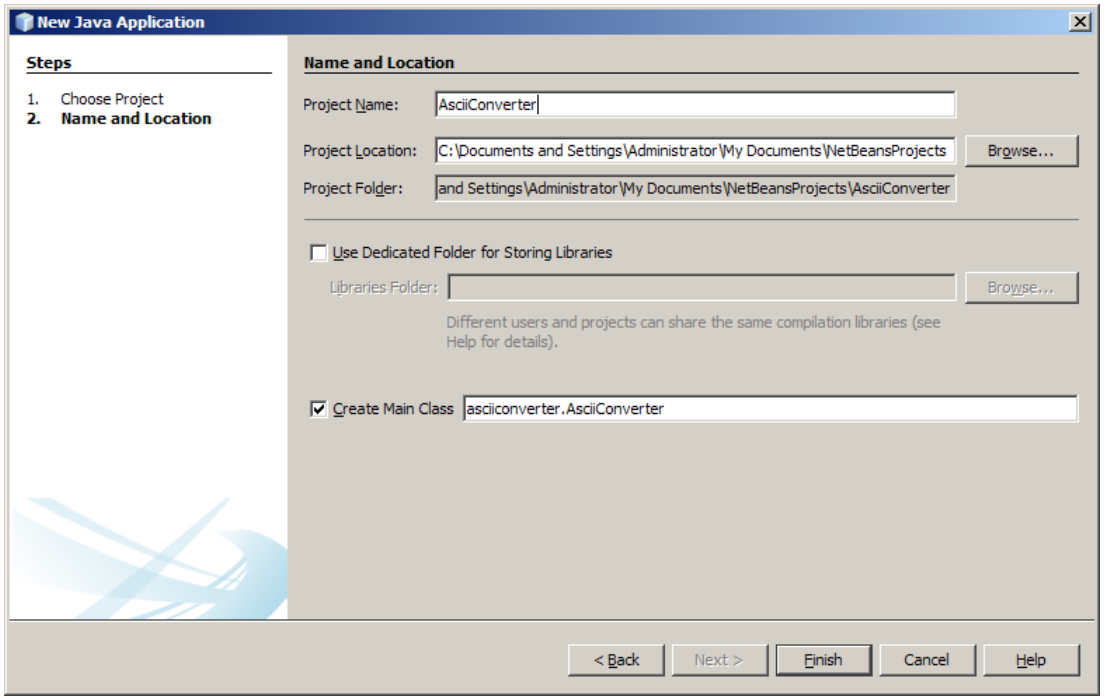
http://www.joelonsoftware.com/articles/Unicode.html

UTF-8

- UTF (Universal True Font) is a variable width encoding which can store any of the Unicode characters.
- In UTF-8, every code point from 0-127 is stored in a single byte. Only code points 128 and above are stored using 2, 3 or 4 bytes. This has the neat side effect that English text looks exactly the same in UTF-8 as it did in ASCII, so Americans don't even notice anything wrong.
  - Hello in Unicode: U+0048 U+0065 U+006C U+006C U+006F;
  - Hello in UTF-8: 48 65 6C 6C 6F
- UTF-8 has become the dominant character encoding for the World Wide Web, accounting for more than half of all Web pages. UTF-8 is also increasingly being used as the default character encoding in operating systems, programming languages, APIs, and software applications.

Homework

- Create a new program in NetBeans. The program shall convert each character of a given word into a decimal equivalent
- Open NetBeans IDE and go to File -> New Project
- Select “Java Application” from Project types.
- Type in project name. Click “Finish” and NetBeans will generate all the class definition and main method code for you.



Homework

- F11 compiles the code;
- F6 runs the code;
- See the output:
  - H is 72
  - e is 101
  - Etc.
  -

- **Optional assignment**: Instead of hard-coding the word in the “main” method, prompt the user for word input. Check the last week's exercise for sample code.