The char Data Type (Outsource: 3-27)

The char data type is used to hold any single character. You place constant characters values within single quotation marks because the computer stores characters and integers differently. For example the statement `char aCharValue = 'A';` assigns the letter **A** to the char variable `aCharValue`. A number can be represented as a character by enclosing it in single quotation marks, I.E. `char aCharValue = '9';`.

The String Class (Outsource: 3-32 – 3-35)

A variable of type char can hold only one character. To store more than one character, such as a person's name, you must use the `String` class. Characters assigned to a String are placed inside double quotation marks.

The String class stores data (sequences of characters) and it also has numerous *methods* to act on the data. A String can be declared two different ways. The statements `String str = new String("Hello World");` and `String str = "Hello World";` both create a String and assign it the value "Hello World"; The first method is the normal way of creating an instance of a class.

Two or more Strings can be joined together (concatenated) using the plus (+) operator. The statement `String str = "Hello " + "World";` concatenates the two string literals "Hello " and "World" into one String and stores it in the String str. You can perform the same assignment using multiple Strings:

```
String a = "Hello ";
String b = "World";
String c = a + b;
```

Getting The Length Of A String

The length method returns the number of characters contained in the string. After the following two lines of code have been executed, len equals 17:

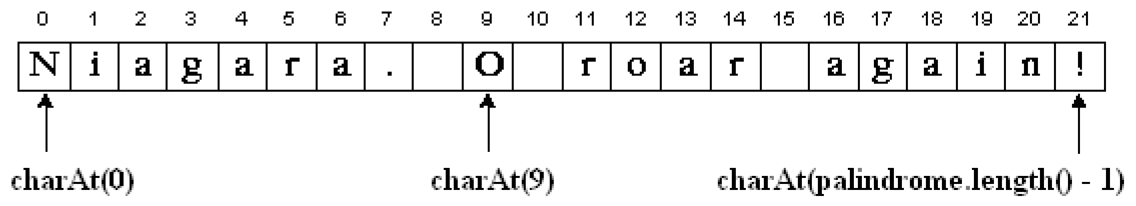
```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();            // len is 17
```

Getting Characters By Index From A String

You can get the character at a particular index within a string by using the charAt method. The index of the first character is 0; the index of the last is length()-1. For example, the following code gets the character at index 9 in a string:

```
String palindrome = "Niagara. O roar again!";
char aChar = palindrome.charAt(9);
```

Indices begin at 0, so the character at index 9 is 'O', as illustrated in the following figure:



ASCII Codes

ASCII stands for *American Standard Code for Information Interchange.* Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. Below is the ASCII character table and this includes descriptions of the first 32 IBM PC Extended ASCII Display Characters.

ASCII	CHARACTER (CODE)	ASCII	CHARACTER (CODE)
0	NULL	16	►
1	☺	17	◄
2	☻	18	↕
3	♥	19	‼
4	♦	20	¶
5	♣	21	§
6	♠	22	▬
7	[Bell]	23	↕
8	[Backspace]	24	↑
9	[Horizontal Tab]	25	↓
10	[Linefeed]	26	→
11	♂	27	← [Escape]
12	♀	28	∟
13	[Carriage Return]	29	↔
14	♫	30	▲
15	☼	31	▼

ASCII	CHARACTER	ASCII	CHARACTER	ASCII	CHARACTER
32	<SPACE>	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l

45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	<DEL>

The Scanner Class

The Scanner class provides methods to read integers, doubles, and strings. There are no methods in the Scanner class that read characters.

```
Scanner reader = new Scanner(System.in);
String name = reader.next ();
```

Method Summary – **Scanner class**	
String	`next()` Finds and returns the next complete token from this scanner.
double	`nextDouble()` Scans the next token of the input as a `double`.
int	`nextInt()` Scans the next token of the input as an `int`.
String	`nextLine()` Advances this scanner past the current line and returns the input that was skipped.
Scanner	`useDelimiter(String pattern)` Sets this scanner's delimiting pattern to a pattern constructed from the specified `String`.

LAB 03 - ASSIGNMENT

Lab 03A - 60 points

OBJECTIVE

WAP that reads three words from the keyboard then displays them in reverse order.

FIELD SUMMARY

- **String firstWord** – the first word read from the keyboard.
- **String secondWord** – the second word read from the keyboard.
- **String thirdWord** – the third word read from the keyboard.

METHOD SUMMARY

- **main** – instantiate an object of your class. Call input, process and output.
- **input** – declare a `Scanner` object and read three Strings from the keyboard using an appropriate prompt.
- **output** – display the results on the console screen.

SAMPLE KEYBOARD INPUT

Enter three words: **Able Baker Charlie**

SAMPLE OUTPUT

Charlie Baker Able

Lab 03B - 70 points

OBJECTIVE

WAP that reads your first name and last name from the keyboard and stores them in two separate String variables. Output your name and the total number of characters in your name. Calculate the length of your name by adding together the length of your first name and the length of your last name.

FIELD SUMMARY

- **String firstName** – your first name.
- **String lastName** – your last name.
- **int len** – the length of your first and last names combined.

METHOD SUMMARY

- **main** – instantiate an object of your class. Call input, process and output.
- **input** – declare a `Scanner` object and read three Strings from the keyboard using an appropriate prompt.
- **process** – add together the length of your first and last name and store the result in `len`.
- **output** – display the results on the console screen.

SAMPLE KEYBOARD INPUT

```
Enter your first name: Bob
Enter your last name: Hope
```

OUTPUT

```
My name is Bob Hope.
My name contains 7 characters.
```

Lab 03C - 80 points

OBJECTIVE

WAP that reads a sentence from the keyboard. Output the sentences and the number of characters in the sentence on the console screen Then output the first and last characters in the sentence.

FIELD SUMMARY

- **String sentence** – a sentence to be read from the keyboard.

METHOD SUMMARY

- **main** – instantiate an object of your class. Call input, process and output.
- **input** – declare a `Scanner` object and read a sentences from the keyboard using an appropriate prompt.
- **output** – display the sentences on the console screen, the number of characters in the sentence and the first and last characters in the sentence.

SAMPLE KEYBOARD INPUT

Enter a sentence: **See Spot Run**

SAMPLE OUTPUT

See Spot Run contains 12 characters.
The first character is ‘S’ and the last character is ‘n’.

Lab 03D - 90 points

OBJECTIVE

WAP that reads a name and an age from the keyboard. You must have `input`, `process`, and `output` methods.

SAMPLE KEYBOARD INPUT

```
Enter your name:  Mary Smith
Enter your age: 17
```

SAMPLE OUTPUT

```
Hello, Mary Smith. You were born in 1996.
```

Lab 03E - 100 points

OBJECTIVE

WAP to input a student's first name, last name, age and the math class you are currently enrolled in, I.E. Geometry K, Algebra II, Pre Calculus etc. Input your age as an integer.

FIELD SUMMARY

- **String firstName** – student's first name.
- **String lastName** – student's last name.
- **int age** – student's age.
- **String mathClass** – student's math class.

METHOD SUMMARY

- **main** – instantiate an object of your class. Call input, process and output.
- **input** – declare a `Scanner` object and read a student's name, age, and current math class from the keyboard using appropriate prompts before each input.
- **output** – display the results on the console screen.

SAMPLE KEYBOARD INPUT

Enter Student's First Name: **James**
Enter Student's Last Name: **Smith**
Enter Student's Age: **17**
Enter Student's Math Class: **Calculus AP**

SAMPLE OUTPUT

```
My Name is James Smith.
I am 17 years old.
I am enrolled in Calculus AP.
```

NOTE: The Scanner classes **next()**, **nextInt()** and **nextDouble()** methods do not remove the *Enter* key from the input stream. Therefore, following any of these three methods with a **nextLine()** (which reads until it encounters an *Enter* key) creates a problem. The **nextLine()** method will pick up the *Enter* left by any of the above three methods. This problem only exists when **nextLine()** is preceeded by **next()**, **nextInt()** or **nextDouble()**. The solution to this problem is to follow any of the `next` methods by a dummy read. The following statements demonstrate this technique.

```
Scanner reader = new Scanner(System.in);
number = reader.nextInt();      // Reads an integer but leaves the Enter key in the input stream
```

```
reader.nextLine();              // Dummy read removes the Enter key from the input
stream
sentence = reader.nextLine();   // Reads the next line from the input stream.
```