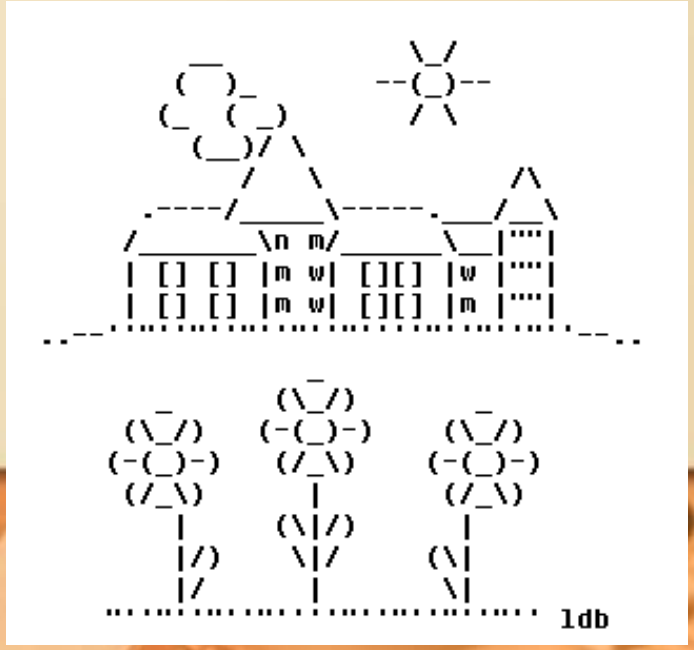
Lecture 3-2

Handling Characters and Text



Textual data

"When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume among the powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should decla
to the sepa

10月27日、マイクロソフトの定番ビジネススイート「マイクロソフト・オフィス」の最新版となる「オフィス2011（Office for Mac 2011）」がついに発売されました。

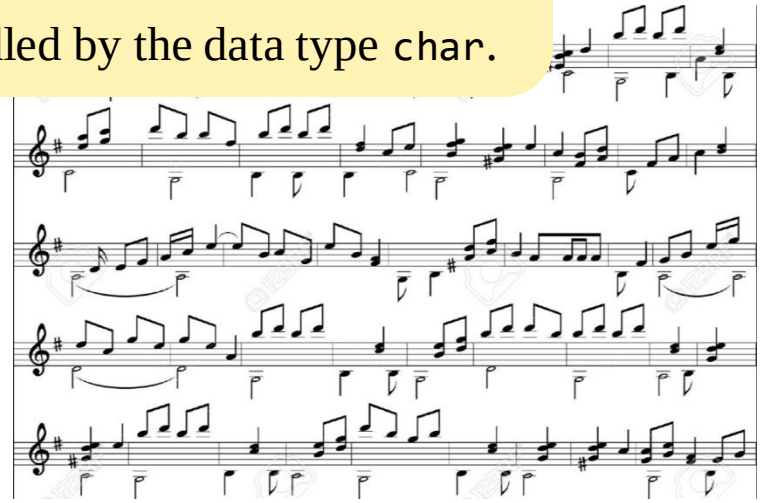
2009年8月、マイクロソフトは新しいMac版オフィスについてのロプを示しました。当時、2010デーシーズンに発売する予定であ新たに統合メールソフトの「アク」を搭載することなどが発表さが、詳細はあまり明らかにされずユーザは長い間その詳細の発表をしていたわけです。

Data = a stream of *characters*;

Every character is represented by a numeric code;

In Java, these codes are handled by the data type `char`.

```
>ARPM2ref|NC_000001.10|:29
reference assembly
TGGAAGAGGCCTCAGCAGGCCCAGGC
CGGGCACGGTGCTAGCCCTGCCTTGA
ATCACAAAGCGGCCCTGGAGGGCTGGTC
TAATCCGCACGCTTTAGACTCCCCGGCTGTGATTTTTGACAATGGCTCGGGGTTCTGCAAAGCGGGCCTG
TCTGGGGAGTTTGGACCCCGGCACATGGTCAGCTCCATCGTGGGGCACCTGAAATTCCAGGCTCCCTCAG
CAGAGGCCAACCAGAAGAAGTACTTTGTGGGGGAGGAGGCCCTGTACAAGCAGGAGGCCCTGCAGCTGCA
CTCCCCTTTCGAGCGTGGCCTGATCACAGGGTGGGATGACGTGGAGAGACTCTGGAAGCACCTCTTTGAG
TGGGAGCTAGGCGTGAAACCCAGCGACCAGCCCCTGCTTGCAACGGAGCCCTCCCTGAACCCCAGGGAGA
ACCGTGAGAAGATGGCAGAAGTCATGTTCGAGAACTTCGGCGTGCCCGCTTTCTACCTGTCGGACCAGGC
GGTGCTGGCTCTCTACGCCTCTGCCTGTGTCACGGGCCTGGTGGTGGACAGCGGGGATGCGGTCACCTGC
ACTGTCCCCATCTTTGAGGGTTACTCCCTGCCCCACGCAGTCACCAAGCTCCACGTGGCGGGCAGGGACA
TCACGGAGCTCCTCATGCAGCTGCTCCTGGCCAGCGGCCACACCTTCCCCTGCCAGCTGGACAAGGGTCT
CGTGGACGACATCAAAAAGAAGCTGTGCTACGTGGCCTTGGAGCCCGAGAAGGAGCTTTCCCGGAGGCCG
GAGGAGGTCCTGAGGGAGTACAAGCTGCCCGACGGGAACATCATCAGCCTCGGGGACCCGCTGCACCAGG
CGCCCGAGGCCCTGTTCGTGCCCCAGCAGCTGGGCAGCCAGAGCCCCGGGCTCTCGAATATGGTCTCCAG
CAGCATCACCAAGTGTGATACCGACATCCAGAAGATCCTCTTTGGGGAGATTGTGCTGTCGGGGGGCACT
ACCCTGTTCCACGGGCTGGATGACCGGCTTCTCAAGGAGCTGGAGCAGCTGGCCTCCAAGGACACCCCCA
TCAAGATCACGGCTCCCCCCGACCGGTGGTTCTCCACCTGGATTGGAGCCTCCATCGTCACCTCTCTGAG
TAGCTTCAAGCAGATGTGGGTCACCGCCGCAGACTTCAAGGAGTTTGGGACCTCCGTGGTGCAGAGAAGA
TGCTTCTGAAGGCCGCTTCTCGTTGGGTACCGTGGGGGGTGAACCCTAGCCCCAGCTTTGGGAGGATGTT
CAATAAAGGACCAATGCCGGAA
```



The `char` data type

type	bits	range of values
`byte`	8	`-128 ... 127`
`short`	16	`-32,768 ... 32,767`
`char`	16	`0 ... 65,535`
`int`	32	`-2,147,483,648 ... 2,147,483,647`
`long`	64	`< -9 x` 10^{18} `...` `> 9 x` 10^{18}

Java data types for representing integers

`char` values are integers

- When we say `print('k')`, the computer displays an image that looks like k
- But, the letter `'k'` is actually represented by the *character code* `107`.

Character codes: ASCII

Dec	Char		Dec	Char	Dec	Char	Dec	Char
0	NULL	null	32	Space	64	@	96	`
1	SOH	Start of heading	33	!	65	A	97	a
2	STX	Start of text	34	"	66	B	98	b
3	ETX	End of text	35	#	67	C	99	c
4	EOT	End of transmission	36	$	68	D	100	d
5	ENQ	Enquiry	37	%	69	E	101	e
6	ACK	Acknowledge	38	&	70	F	102	f
7	BELL	Bell	39	'	71	G	103	g
8	BS	Backspace	40	(	72	H	104	h
9	TAB	Horizontal tab	41	)	73	I	105	i
10	LF	New line	42	*	74	J	106	j
11	VT	Vertical tab	43	+	75	K	107	k
12	FF	Form Feed	44	,	76	L	108	l
13	CR	Carriage return	45	-	77	M	109	m
14	SO	Shift out	46	.	78	N	110	n
15	SI	Shift in	47	/	79	O	111	o
16	DLE	Data link escape	48	0	80	P	112	p
17	DC1	Device control 1	49	1	81	Q	113	q
18	DC2	Device control 2	50	2	82	R	114	r
19	DC3	Device control 3	51	3	83	S	115	s
20	DC4	Device control 4	52	4	84	T	116	t
21	NAK	Negative ack	53	5	85	U	117	u
22	SYN	Synchronous idle	54	6	86	V	118	v
23	ETB	End transmission bloc	55	7	87	W	119	w
24	CAN	Cancel	56	8	88	X	120	x
25	EM	End of medium	57	9	89	Y	121	y
26	SUB	Substitute	58	:	90	Z	122	z
27	FSC	Escape	59	;	91	[	123	{
28	FS	File separator	60	<	92	\	124	|
29	GS	Group separator	61	=	93	]	125	}
30	RS	Record separator	62	>	94	^	126	~
31	US	Unit separator	63	?	95	_	127	DEL

Character set

A set of characters, and their codes;

Each character is represented by an agreed-upon numeric code.

ASCII

8-bit code character set for representing western alphabets (a standard formed in 1963):

- `0-31`:
  Control characters
- `32-127`:
  Common western characters

Character codes: ASCII

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Character set

A set of characters, and their codes;

Each character is represented by an agreed-upon numeric code.

ASCII

8-bit code character set for representing western alphabets (a standard formed in 1963):

- 0-31:
  Control characters
- 32-127:
  Common western characters
- 128-255:
  More characters, mostly graphics primitives

Character codes: Unicode

ASCII (1963): an 8-bit code that represents $2^8 = 256$ characters

Unicode (1991): a 16-bit code that can represent $2^{16} = 65536$ characters

Java uses Unicode.

ASCII is embedded within Unicode. For example:

`'a'` in ASCII: `0110001` (97 in decimal)

`'a'` in Unicode: `000000000110001` (97 in decimal).

Plan

- ✓ ASCII / Unicode
- ➡ Characters
- Strings
- String processing examples

Characters in action

Dec	Char	Dec	Char	Dec	Char
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

Characters in action

We'll show some examples, focusing on this arbitrary subset of selected characters.

Dec	Char
32	Space
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?

Dec	Char
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_

Dec	Char
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~
127	DEL

Characters in action

```
// CharDemo.java
...
System.out.println(3);
System.out.println('3');
System.out.println(3 + 1);
System.out.println('3' + 1); // casts to int

char c = 8096; // sets c to some arbitrary value
System.out.println(c);

System.out.println('h');
System.out.println('h' - 1);

c = 'd';
// Converts to uppercase:
System.out.println(c - 32); // casts to int
System.out.println((char) (c - 32));

c = 'K';
// Converts to lowercase:
// You do it

c = 't';
// Checks if c is a lowercase letter:
System.out.println((c >= 97) && (c <= 122));
System.out.println((c >= 'a') && (c <= 'z')); // more readable

c = '7';
// Checks if c is a digit:
// You do it
...
```

output

```
3
3
4
52
ẘ
h
103
68
D
true
true
```

Dec	Char	Dec	Char	Dec	Char
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

A peek into how computers work (sneak preview)

```
int x = 97;

System.out.println(x);
```

Internal view:

In memory, x stores the 32-bit, binary value of 97, which happens to be:
`00000000000000000000000001100001`

Anatomy of `System.out.println(x)`

- Java converts the code `00000000000000000000000001100001` to decimal; The result is 97
- Java enters a loop that iterates through all the digits of 97, left to right: first the 9, then the 7
  - Java looks up the ASCII / Unicode table, and finds that `'9'` is represented as `57`
  - Java tells the OS "display the image of ASCII code `57`"
  - The OS looks up `57` in the current font table, and sees that it is represented by a matrix of pixels that creates the image 9; it displays this image on the screen
  - Continues to do the same with `'7'`, etc.
- The user sees 97 on the screen, completely unaware of the underlying drama.

Special characters

example:

```
System.out.println("I don\'t like\nthe word \"no\"");
```

```
I don't like
the word "no"
```

Escape Sequences

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

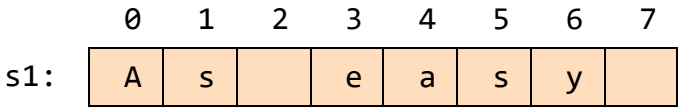
Patterns that begin with a backslash (\) are called *escape sequences*

Plan

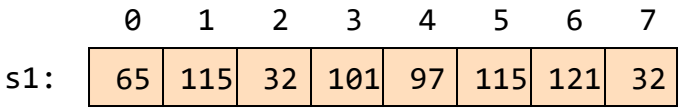
- ✓ ASCII / Unicode
- ✓ Characters
- ➡ Strings
  - String processing examples

Strings

```
String s1 = "As easy ";
```



high-level view

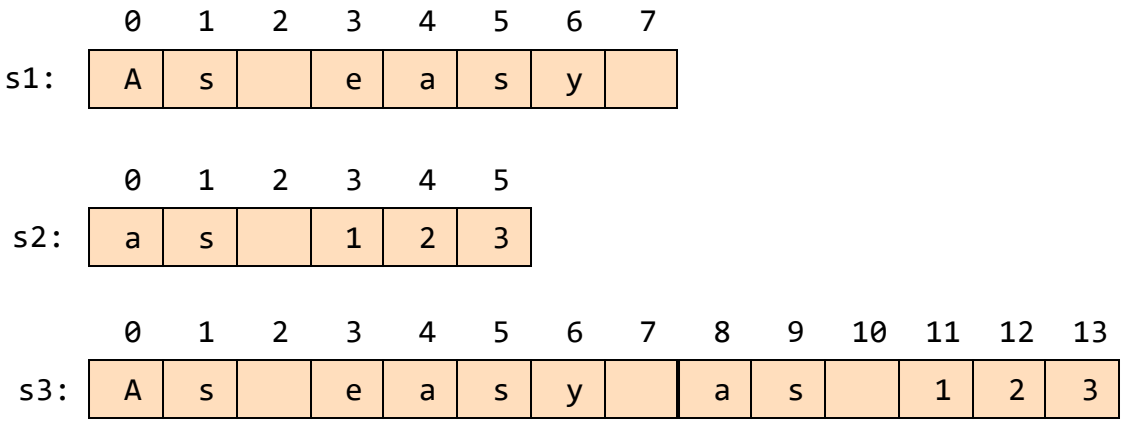


low-level view

Dec	Char	Dec	Char
32	Space	96	`
33	!	97	a
34	"	98	b
35	#	99	c
36	$	100	d
37	%	101	e
38	&	102	f
39	'	103	g
40	(	104	h
41	)	105	i
42	*	106	j
43	+	107	k
44	,	108	l
45	-	109	m
46	.	110	n
47	/	111	o
48	0	112	p
49	1	113	q
50	2	114	r
51	3	115	s
52	4	116	t
53	5	117	u
54	6	118	v
55	7	119	w
56	8	120	x
57	9	121	y
58	:		
59	;		
...		...	

Strings

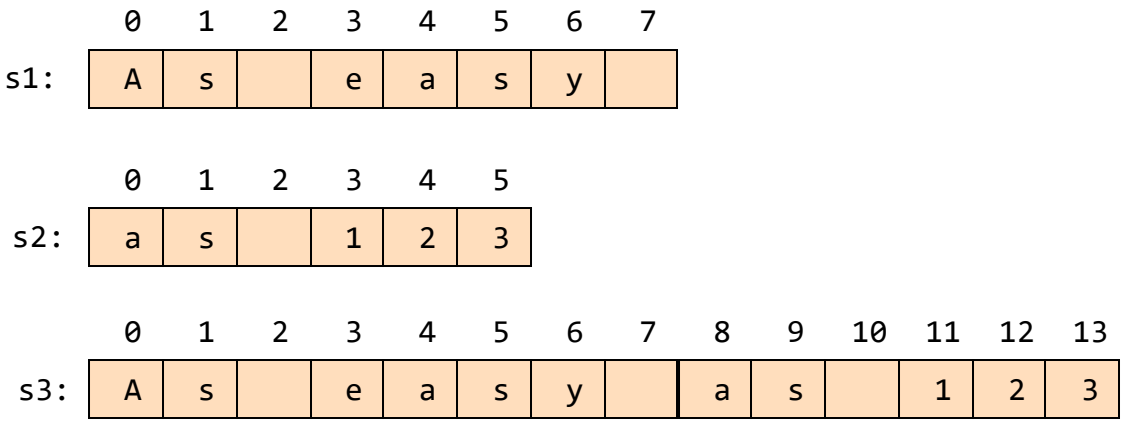
```
String s1 = "As easy ";
String s2 = "as 123";
String s3 = s1 + s2;
```



Strings

```
String s1 = "As easy ";
String s2 = "as 123";
String s3 = s1 + s2;
System.out.println(s1.length());
System.out.println(s1.charAt(3));
System.out.println(s1.substring(1,5));
System.out.println(s3.indexOf('a'));
```

```
8
e
s ea
4
```



Strings

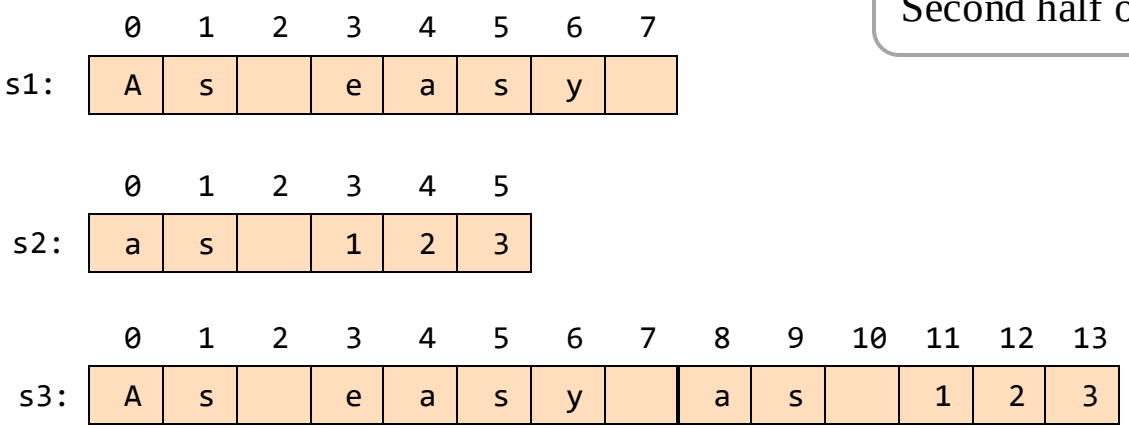
```
String s1 = "As easy ";
String s2 = "as 123";
String s3 = s1 + s2;
System.out.println(s1.length());
System.out.println(s1.charAt(3));
System.out.println(s1.substring(1,5));
System.out.println(s3.indexOf('a'));
```

Method calls

```
8
e
s ea
4
```

- *Methods* are similar to *functions*
- Unlike functions, methods operate on objects
- In particular, string methods operate on string objects
- Instead of writing: `charAt(s1,3)`,
  we write: `s1.charAt(3)`

Much more about objects and methods:
Second half of the course.



Plan

- ✓ ASCII / Unicode
- ✓ Characters
- ✓ Strings
- ➡ String processing examples

Example: Upcase

```
/** A library of string functions. */
public class MyString {
   public static void main(String args[]) {
      System.out.println(upCase(args[0]));
   }

   /** Capitalizes the first letter in every word in the string. */
   public static String upCase(String str) {
```

Dec	Char	Dec	Char	Dec	Char
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

```
% java MyString "it was the best of time"
It Was The Best Of Time
```

Example: Upcase

```
/** A library of string functions. */
public class MyString {
   public static void main(String args[]) {
      System.out.println(upCase(args[0]));
   }

   /** Capitalizes the first letter in every word in the string. */
   public static String upCase(String str) {
      String ans = "" + (char) (str.charAt(0) - 32);
      int i = 1;
      while (i < str.length()) {
         char ch = str.charAt(i);
         if (ch == ' ') {
            ans = ans + ch + (char) (str.charAt(i + 1) - 32);
            i++;
         } else {
            ans = ans + ch;
         }
         i++;
      }
      return ans;
   }
}
```

Dec	Char	Dec	Char	Dec	Char
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

```
% java MyString "it was the best of time"
It Was The Best Of Time

% java MyString "2 apples and 3 pears"
 Apples and  0ears
```

Problems with this version of `upCase`

- Makes naïve assumptions about the input
- Does not handle white space and non-letter characters.

Example: Parsing

Task: Parse a string that has the following semantics:

char int char int char int …

example:

```
% java Parse T5A12C432G3
```

Example: Parsing

Tokenizing

- The process of turning a stream of *characters* into a stream of *tokens*
- The definition of *token* varies from one application to another
- Comes up in numerous applications

Tokenizer

A class that provides tokenizing services.

Task: Parse a string that has the following semantics:

char int char int char int …

example:

```
% java Parse T5A12C432G3
T
5
A
12
C
432
G
3
```

The program "tokenizes" the string, printing each token in a separate line.

Example: Parsing

Tokenizing

- The process of turning a stream of *characters* into a stream of *tokens*
- The definition of *token* varies from one application to another
- Comes up in numerous applications

Tokenizer

A class that provides tokenizing services.

Tokenizer API:

```
/** A string tokenizer. Returns the next token in a given string. */
public class Tokenizer {

   /** Initializes the Tokenizer to the given string (str). */
   public static void init(String str)

   /** Returns true if the string has more characters to process,
    *  false otherwise. */
   public static boolean hasMoreChars()

   /** Returns the next character in the string. */
   public static char nextChar()

   /** Returns the next int value in the string. */
   public static int nextInt()

   // More tokenizer functions...
}
```

```
% java Parse T5A12C432G3
T
5
A
12
C
432
G
3
```

Example: Parsing

```
// Illustrates using the Tokenizer class. This code can appear in any class.
// Assumes that the data is a string of pairs, each consisting of a letter followed by digits
...
// The data typically comes from a file. Here we assume it comes from the command-line.
// Puts the data in s, then parses and prints each token in s in a separate line.
String s = args[0];
```

Tokenizer API:

```
/** A string tokenizer. Returns the next token in a given string. */
public class Tokenizer {

   /** Initializes the Tokenizer to the given string (str). */
   public static void init(String str)

   /** Returns true if the string has more characters to process,
    *  false otherwise. */
   public static boolean hasMoreChars()

   /** Returns the next character in the string. */
   public static char nextChar()

   /** Returns the next int value in the string. */
   public static int nextInt()

   // More tokenizer functions...
}
```

```
% java Parse T5A12C432G3
T
5
A
12
C
432
G
3
```

Example: Parsing

```
// Illustrates using the Tokenizer class. This code can appear in any class.
// Assumes that the data is a string of pairs, each consisting of a letter followed by digits
...
// The data typically comes from a file. Here we assume it comes from the command-line.
// Puts the data in s, then parses and prints each token in s in a separate line.
String s = args[0];
//  Initializes the Tokenizer to the given string
Tokenizer.init(s);
//  Parses and prints the tokens
while (Tokenizer.hasMoreChars()) {
    System.out.println(Tokenizer.nextChar());
    //  Following the character, there must be an int value
    System.out.println(Tokenizer.nextInt());
}
...
```

This code uses the `Tokenizer` functions as "black box abstractions";
It doesn't know how they are implemented.
It just knows the API.

Tokenizer API:

```
/**  A string tokenizer. Returns the next token in a given string. */
public class Tokenizer {

   /**  Initializes the Tokenizer to the given string (str).  */
   public static void init(String str)

   /**  Returns true if the string has more characters to process,
    *   false otherwise.  */
   public static boolean hasMoreChars()

   /**  Returns the next character in the string.  */
   public static char nextChar()

   /**  Returns the next int value in the string.  */
   public static int nextInt()

   // More tokenizer functions...
}
```

```
% java Parse T5A12C432G3
T
5
A
12
C
432
G
3
```

Example: Tokenizing

```
/**  A string tokenizer. Returns the next token in a given string. */
public class Tokenizer {
```

Let's open the black box

Example: Tokenizing

```
/** A string tokenizer. Returns the next token in a given string. */
public class Tokenizer {
    private static String str; // the string to parse
    private static int N;      // string length (number of characters)
    private static int cursor; // current position in the string

    /** Initializes the Tokenizer to the given string (str). */
    public static void init(String s) {
        str = s;
        N = str.length();
        cursor = 0;
    }

    /** Returns true if the string has more character to process, false otherwise. */
    public static boolean hasMoreChars() {
        // put your code here
    }

    /** Returns the next character in the string.
     *  Should be called only if hasMoreChars() is true. */
    // Side effect: advances the cursor just beyond the character.
    public static char nextChar() {
        // put your code here
    }

    /** Returns the next int value in the string.
     *  Should be called only if hasMoreChars() is true. */
    // Side effect: advances the cursor just beyond the integer.
    public static int nextInt() {
        // put your code here
    }

    // Returns true if the given character is a digit, false otherwise.
    private static boolean isDigit(char c) {
        // put your code here
    }
}
```

Class variables:

Can be accessed by any function in the class

Class members

Class variables and methods are sometimes called “class members”.

`public`: Can be accessed by members of any class; documented by the class API

`private`: Accessed only by members of this class. Called “helpers”, since they help implement other class members

(Notice the different documentation style of private and public members)

Completing the Tokenizer code:

Give it a try.