CSE 21
Intro to Computing II

Lecture 11 – Objects: String & Scanner

Announcement

- Lab#10 due before start of next lab
  - Type your answers in a text file and submit it as an attachment
- Project #2 out this Friday (11/11)
  - Due: Monday (11/28) at 11:59pm
- Reading assignment
  - Chapter 9.1 to 9.5, 12.1 to 12.6 of textbook

String Class

- A string is an object containing one or more characters, treated as a unit.
- Strings are objects, like almost everything else in Java.
- The simplest way to create a string is a string literal or anonymous **String** object, which is a series of characters between quotation marks.
  - Example:

```
        "This is a string literal."
```

Creating Strings

- To create a String:
  - First, declare a pointer to a **String.**
  - Then, create the **String** with a string literal or the new operator.
- Examples:

```
String s1, s2;               // Create pointers
s1 = "This is a test.";   // Create String object
s2 = new String();          // Create String object
```

- These steps can be combined on a single line:

```
String s3 = "String 3."; // All together
```

Strings are …

- Very much like arrays!
- Examples:

```
String s1 = "This is a test.";
int [] a = {1, 2, 3};
```

- Yes, because everything in Java is an Object and we always access objects with pointers!
  - Remember all the exercises you have been doing to understand references to arrays!

Just like other Objects

- **new** operator allows us to create objects:

```
String s1 = new String();
int [] a = new int[];
```

- We manipulate objects with variables which are pointers to the objects!

s1 and ***a*** are pointers to a String and an array!

- We access methods of objects using the . operator

```
sharp.getName();
sharp.setAmount(input.nextInt());
```

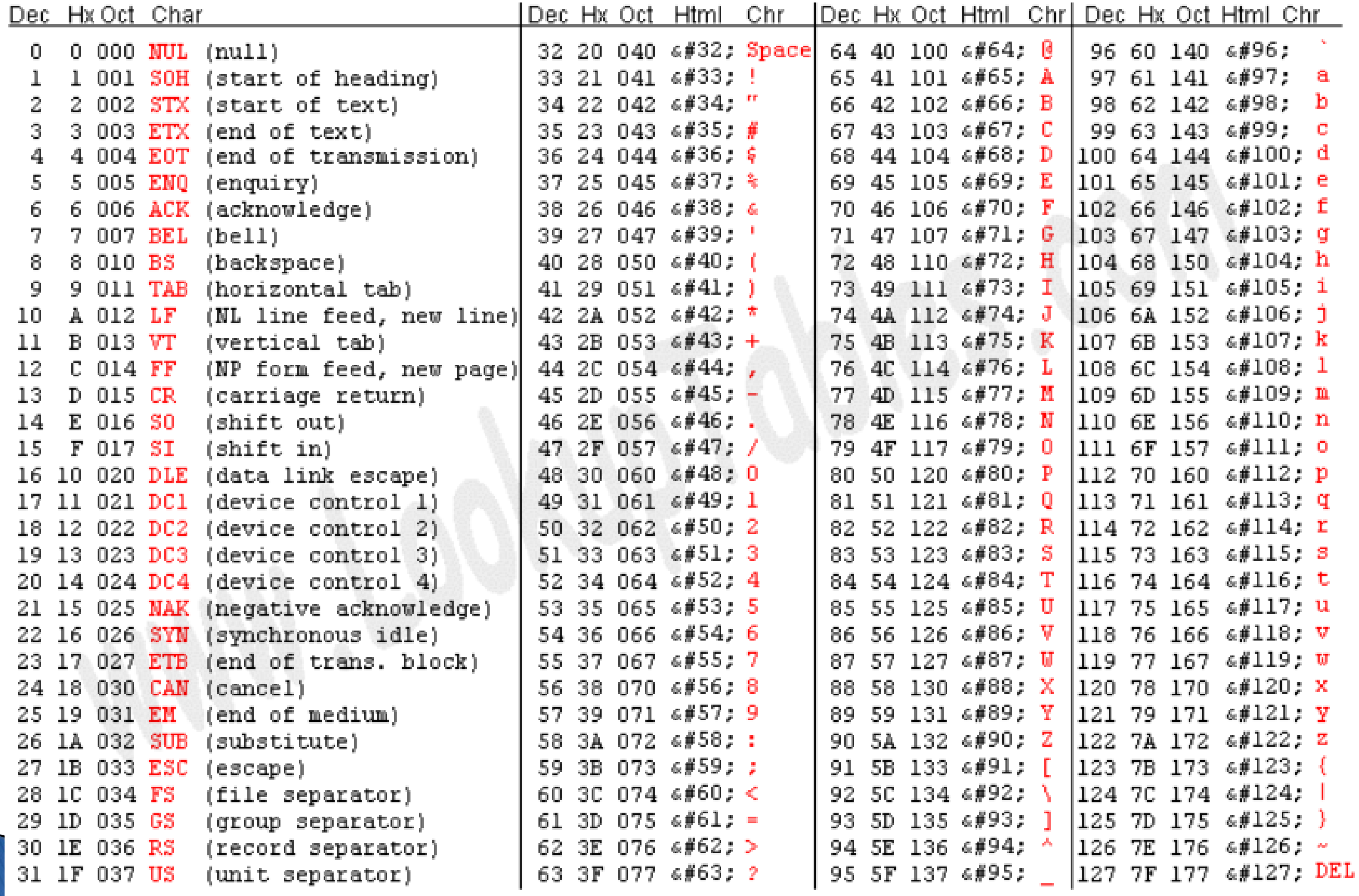
- We can also access methods of Strings:

```
String s1 = “my string”;
s1.substring( … );
```

Strings == arrays of chars!

- Indices start with 0
- Method **s.length()** returns the number of chars
  - Similar to array.length (method call vs variable)
- Each char encodes a number that represents one character
- The ASCII table defines these codes
  - ASCII stands for ***American Standard Code for Information Interchange***

The ASCII table



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Parse Strings

- So we can iterate over all “ascii codes” in a string as an array:
  - Using the **charAt (int i)** method

will print 32, the ascii code of the space character

```
String s1 = “my string”;
System.out.println ( (int)s1.charAt(2) );

for(int i = 0; i < s1.length() ; i++)
  System.out.print( s1.charAt(i) ); // print 1 character a time

 // System.out.println( s1 ); // print the whole string
```

- We check if a character is numeric, lower/upper case, etc, by checking its ascii code
  - if (s1.charAt(i) == ‘+’) …
  - If (s1.charAt(i) <= ‘Z’ && s1.charAt(i) >= ‘A’)

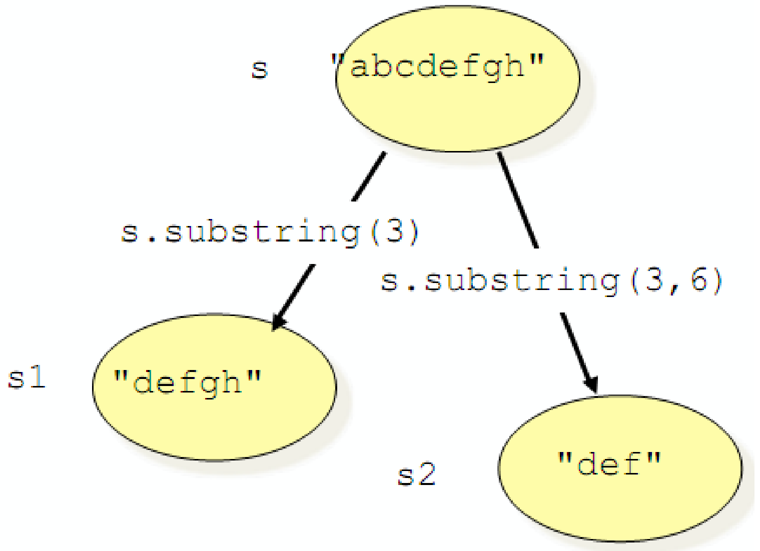
Substrings

- A substring is a portion of a **String**.
- The **String** method **substring** creates a new **String** object containing a portion of another **String**.
- The forms of this method are:
  - `s.substring(int start);            // From [start]`
  - `s.substring(int start, int end); // [start] to [end]`
- This method returns another **String** object containing the characters from ***start*** to ***end*** (or the end of the **string**).

Substrings (2)

- Examples:

```
String s = "abcdefgh";
String s1 = s.substring(3);
String s2 = s.substring(3,6);
```



- Substring **s1** contains "**defgh",** and substring **s2** contains **"def".**
- Again: indices start at 0
- The substring will contain the values from **start** to **end-1.**

Adding Strings

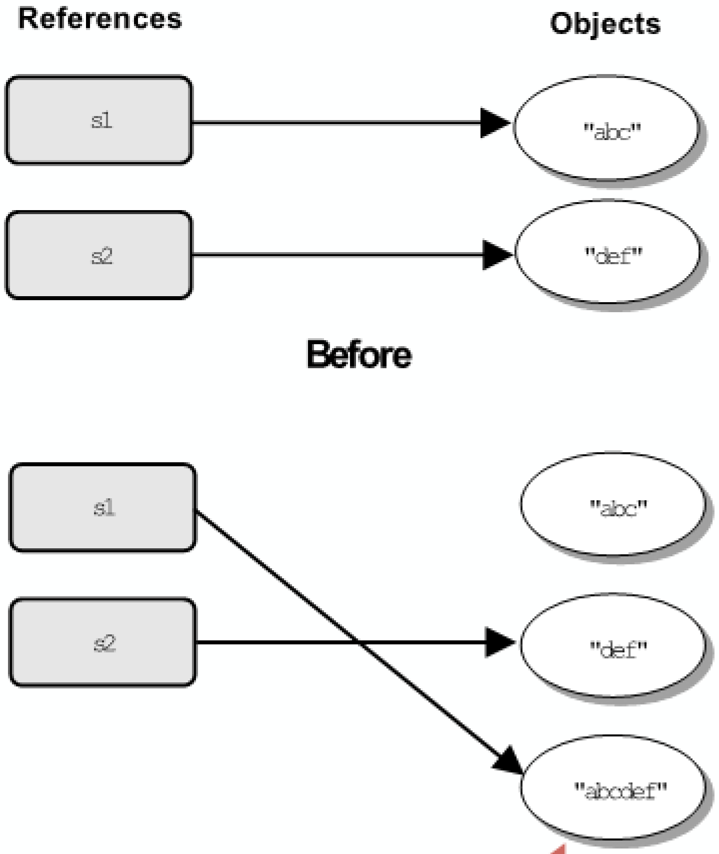
- The **String** method ***concat*** creates a new **String** object containing the contents of two other strings.
- The form of this method is:
  - `s1.concat(String s2);      // Combine s1 and s2`
- This method returns a **String** object containing the contents of **s1** followed by the contents of **s2**.

Concatenating Strings (2)

```
String s1 = "abc";
String s2 = "def";

// Watch what happens here!
System.out.println("\nBefore assignment: ");
System.out.println("s1 = " + s1);
System.out.println("s2 = " + s2);
s1 = s1.concat(s2);
System.out.println("\nAfter assignment: ");
System.out.println("s1 = " + s1);
System.out.println("s2 = " + s2);
```

Output:

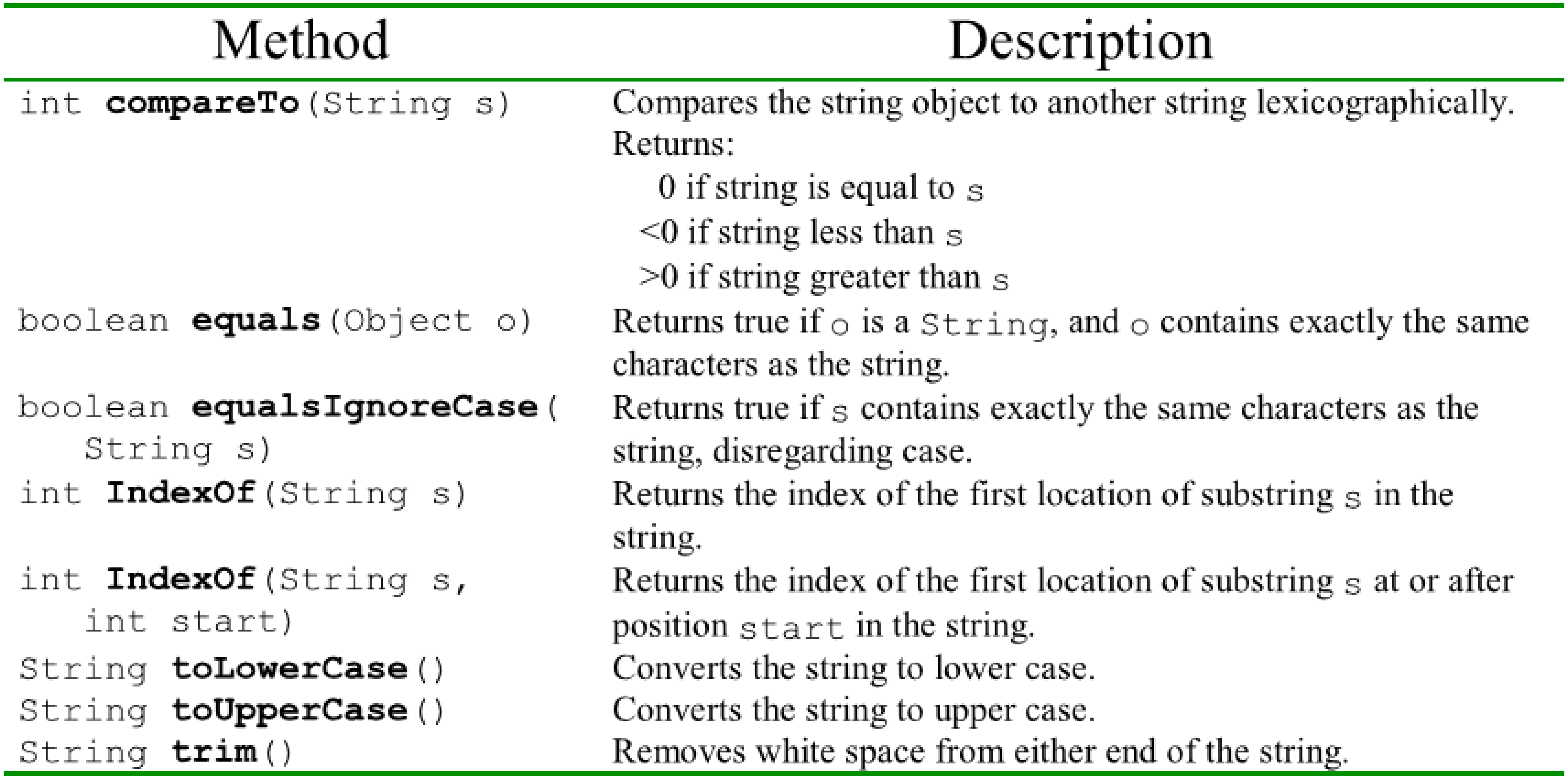
```
Before assignment:
s1 = abc
s2 = def

After assignment:
s1 = abcdef
s2 = def
```



After

New object created.

Selected Additional String Methods



Method	Description
int **compareTo**(String s)	Compares the string object to another string lexicographically. Returns: 0 if string is equal to s <0 if string less than s >0 if string greater than s
boolean **equals**(Object o)	Returns true if o is a String, and o contains exactly the same characters as the string.
boolean **equalsIgnoreCase**(String s)	Returns true if s contains exactly the same characters as the string, disregarding case.
int **IndexOf**(String s)	Returns the index of the first location of substring s in the string.
int **IndexOf**(String s, int start)	Returns the index of the first location of substring s at or after position start in the string.
String **toLowerCase**()	Converts the string to lower case.
String **toUpperCase**()	Converts the string to upper case.
String **trim**()	Removes white space from either end of the string.

Scanners

- Read from User:
  - Scanner kdb = new Scanner (System.in);
  - Pass System.in as parameter to Scanner constructor
- String **s1**= “This is an example”;
- Scanner line = new Scanner (**s1**);
  - Can pass in a String also into Scanner constructor
- kdb.next(); // get next input word
- line.next(); // get next input word also
- line.hasNext() ; // check if there is another word

Parsing Strings

```
String s1 = “This is an example”;
Scanner line = new Scanner (s1);
while (line.hasNext()) {
   System.out.println(line.next());
}
```

- Delimiting character is space : ‘ ’
- OUTPUT
  - This
  - is
  - an
  - example

Parsing Strings with Delimiters

```
String s1 = “This,is,an,example”;
Scanner line = new Scanner (s1);
line.useDelimiter(“[,]”);
while (line.hasNext()) {
   System.out.println(line.next());
}
```

- Delimiting character is comma: ‘,’
- OUTPUT
  - This
  - is
  - an
  - example

Multiple Delimit Characters

```
String s1 = “+This,is+an,example”;
Scanner input = new Scanner (s1);
input.useDelimiter(“[,+]”);
while (input.hasNext()) {
   System.out.println(input.next());
}
```

- Delimiting characters are comma and plus : ‘,’ ‘+’
- OUTPUT:
  - This
  - is
  - an
  - example

Reading Files

```
import java.io.*;
String filename = “nums.txt”;
Scanner input = new Scanner (new FileReader(filename));
input.useDelimiter(“[\t\r]”); // use tab and carriage return
while (input.hasNext()) {
    System.out.println(input.next());
}
input.close();
```

- Import io object library
- Define a file name
- Define a scanner to open a file and read its content
- Close scanner when reading is done
- Exceptions must be handled when reading files:
  - FileNotFoundException (fine does not exist)
  - NoSuchElementException (cannot perform input.next())

Reading line by line

```
System.out.print("Enter the file name: ");
Scanner kdb = new Scanner(System.in);
String filename = kdb.next();

try { // TRY it out
    Scanner input = new Scanner (new FileReader(filename));
    while (input.hasNextLine()) {
        Scanner line = new Scanner(input.nextLine());
        line.useDelimiter(“[\t\r]”); // Tab delimited file
        while (line.hasNext())
            System.out.print(line.next()); // Read each token
        System.out.println();  // Done reading one line
    }
    input.close();
} catch (FileNotFoundException e){ // ERROR : Catch
    System.out.println(e);
} catch (NoSuchElementException e) { // ERROR : Catch
    System.out.println(e);
}
```

2 scanner objects!

1 for reading the whole file, 1 for reading each line.

Example

```
public static void main(String[] args) throws IOException {

    System.out.print("Enter the file name: ");
    Scanner kdb = new Scanner(System.in);
    String filename = kdb.next(); // file name input from user

    Scanner input = new Scanner (new FileReader(filename));
    while (input.hasNextLine()) {
        Scanner line = new Scanner(input.nextLine());
        line.useDelimiter(“[\t\r]”); // Tab delimited file
        while (line.hasNext())
            System.out.print(line.next()); // Read each token
        System.out.println();  // Done reading one line
    }
    input.close();
}
```

Different Scanner Methods

```
while (input.hasNextLine()) {
    Scanner line = new  Scanner(input.nextLine());
    line.useDelimiter(“[\t\r]”);
    short s = line.nextShort();
    int i = line.nextInt();
    double d = line.nextDouble();
    float f = line.nextFloat();
    String str = line.next();
    char c = line.next().charAt(0);
    String rest = line.nextLine();
}
```

Filenames and Paths

```
String s;
s = “myfile.txt”;                 // in current folder
s = “../myfile.txt”;              // previous folder
                                  // (relative path)
s = “data/myfile.txt”;            // (relative path)
s = “C:/tmp/myfile.txt”;          // full path specified
s = “C:\temp-file.txt”;           // Error! (\t is a tab!)
s = “C:\\tmp\\myfile.txt”;        // Ok in windows

Scanner input = new Scanner (new FileReader(s));
```