CSE 21
Intro to Computing II

Lecture 10

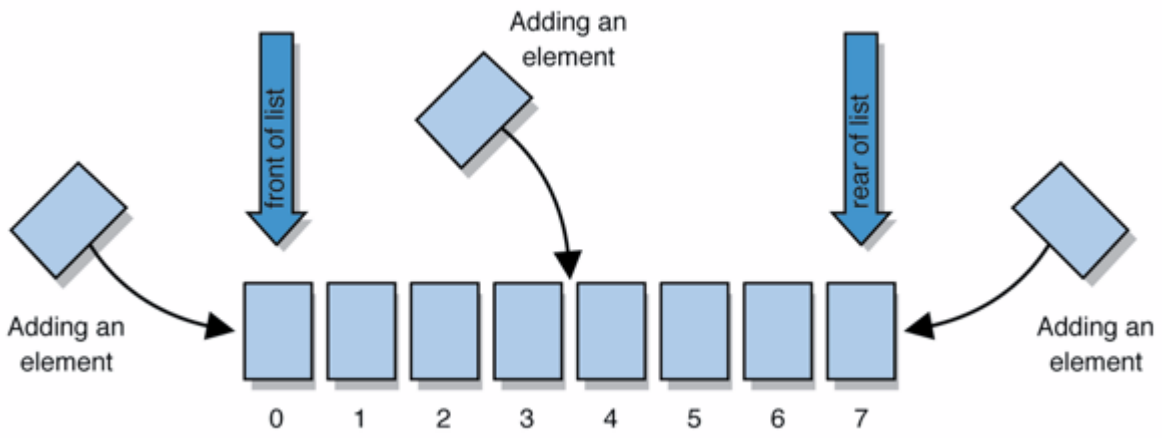
Objects: String and Scanner (1)

Today

- Objects: String and Scanner
- Lab
  - Lab 11 due this week (4/15 – 4/21)
  - Lab 12 assigned this week
    - Recursion
    - Due in one week
    - **Required** to show work to a TA (or me) for full credit
  - Project 2 due this week on Friday, 4/20
    - **Required** to show work to a TA (or me) for full credit
- Reading Assignment
  - Sections 9.1 – 9.5 and 12.1 – 12.6 (including participation activities)
    - Work on the **Participation Activities** in each section to receive participation grade at the end of semester (based on at least 80% completion)
    - Work on **Challenge Activities** to receive extra credit
  - Participation and Challenge activities evaluated at the end of semester

List of Objects (review)

- An ordered sequence of elements:
  - each element is accessible by a 0-based **index**
  - a list has a **size** (number of elements that have been added)
  - elements can be added to the front, back, or elsewhere
  - in Java, a list can be represented as an **ArrayList** object



Contents of a List (review)

- Rather than creating an array of boxes, create an object that represents a "list" of items. (initially an empty list.)

  { }

- You can add items to the list.
  - The default behavior is to add to the end of the list.

    {first, second, third, fourth}

- The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - Think of an "array list" as an automatically resizing array object.
  - Internally, the list is implemented using an array and a size field.
- Many Methods: add, remove, get, set, contains, etc.

Creating ArrayLists (review)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an **ArrayList**, you must specify the type of elements it will contain between **<** and **>.**
  - This is called a *type parameter* or a *generic class*.
  - Allows the same **ArrayList** class to store lists of different types.

```
ArrayList<String> names = new ArrayList<String>();
names.add("John Smith");
names.add("Jerry West");
```

ArrayList as a parameter (review)

```
public static void methodName(ArrayList<Type> param) { ... }
```

- Example:

```
// Removes all plural words from the given list.
public static void removePlural(ArrayList<String> list) {
    String str;
    for (int i = 0; i < list.size(); i++) {
        str = list.get(i);
        if (str.endsWith("s")) { // or if (list.get(i).endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- You can also return a list:

```
public static ArrayList<Type> methodName(params) { ... }
```

ArrayList of primitives? (review)

- The type you specify when creating an **ArrayList** must be an object type; it cannot be a primitive type.

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use **ArrayList** with primitive types by using special classes called wrapper classes in their place.

```
// creates a list of Integers
ArrayList<Integer> list = new ArrayList<Integer>();
```

- Other wrapper classes: Double, Character, Boolean

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
String s4 = new String();
```

Strings are …

- Very much like arrays
- Examples:

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

- Everything in Java is an Object and we always access objects with pointers – Just like arrays
  - 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());
```

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

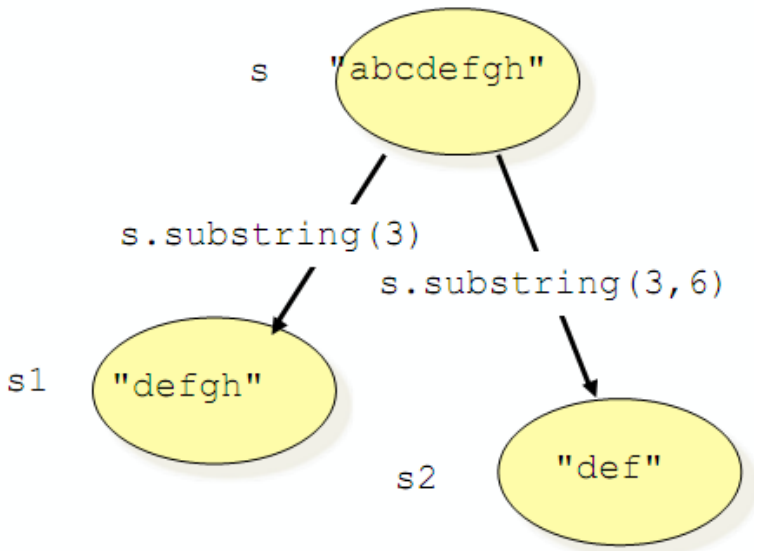
Substrings (2)

- 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.**



Concatenating 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**.

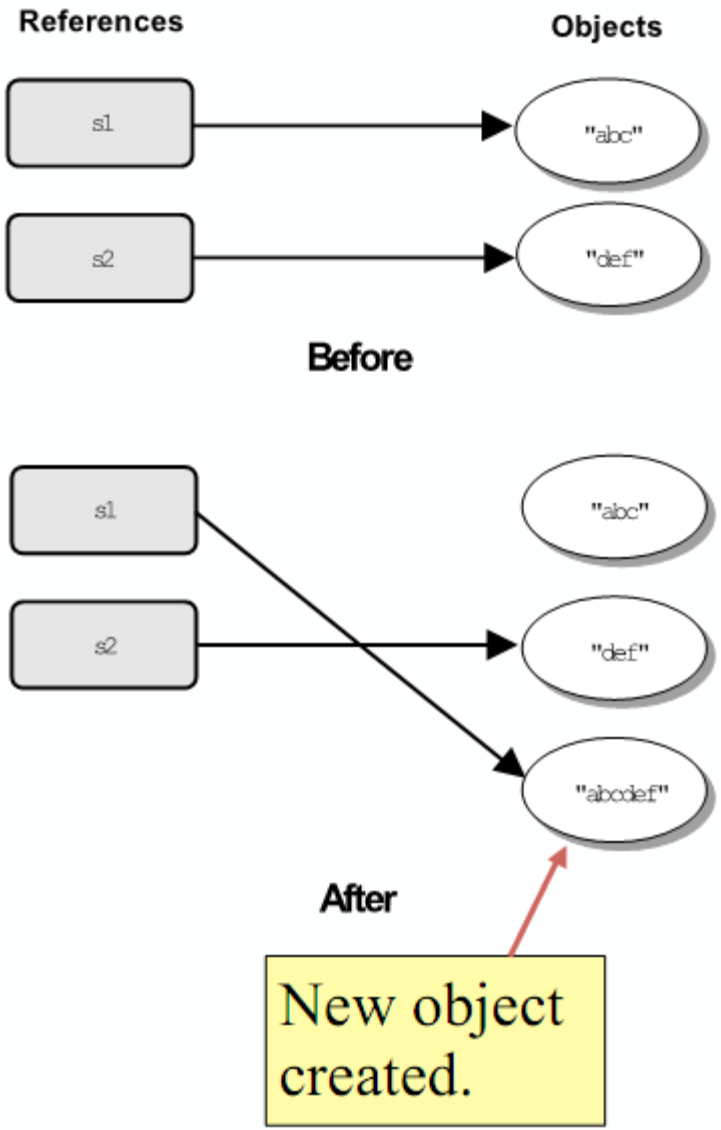
Concatenating Strings (2)

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

System.out.println("Before 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



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 to Scanner constructor as well
- kdb.next(); // get next input word
- line.next(); // also gets next input word
- 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 a Delimiter

```
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
```

Parsing Strings with Multiple Delimiters

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

- Delimiting characters are comma and plus: ',' and '+'
- OUTPUT:

```
This
is
an
example
```

Reading Files

```
import java.io.*;
String filename = "nums.txt";
Scanner input = new Scanner (new FileReader(filename)); // or FileInputStream
                                                        // instead of FileReader
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
		line.close();
	}
	input.close();
} catch (FileNotFoundException e){ // Catch Error
	System.out.println(e);
} catch (NoSuchElementException e) { // Catch Error
	System.out.println(e);
}
```

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
		line.close();
	}
	input.close();
} catch (FileNotFoundException e){ // Catch Error
	System.out.println(e);
} catch (NoSuchElementException e) { // Catch Error
	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

     try {
        ...
        ...
    }
}
```

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();
}
```

Example File Out

```
String filename = "Result.txt";

try {
        FileWriter output = new FileWriter(filename);
        String outstr = "";
        for (int i = 0; i < arr.length; i++) {
                outstr = (arr[i] + "\t");
                output.write(outputstr);
        }
        output.close();
} catch (Exception e) {
        System.out.println(e);
}
```

Filenames and Paths

```
String s;
s = "myfile.txt";          // in current folder (Project folder in case of Eclipse)
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));
```