Using Classes

String Class (part 2)

Lecture Contents

- Declaring a `String`
- **`String`** Class Methods

```
String toUpperCase()
String toLowerCase()
int length()
char charAt(int index)
int indexOf(String str)
int indexOf(String str, int start)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
boolean equalsIgnoreCase(String other)
int compareTo(String other)
```

These methods in part 1

`String` Class – Calling Methods

- Now we discuss the `equals` and `equalsIgnoreCase` methods:

```
String toUpperCase()
String toLowerCase()
int length()
char charAt(int index)
int indexOf(String str)
int indexOf(String str, int start)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
boolean equalsIgnoreCase(String other)
int compareTo(String other)
```

Declaring a `String`

- We've been declaring strings using:

```
String s = "Hello World!";
```

Declaring a `String`

- These are (almost) equivalent ways to declare and initialize a string:

```
String s = "Hello World!";

String s;
s = "Hello World!";
```

Declaring a `String`

- These are (almost) equivalent ways to declare and initialize a string:

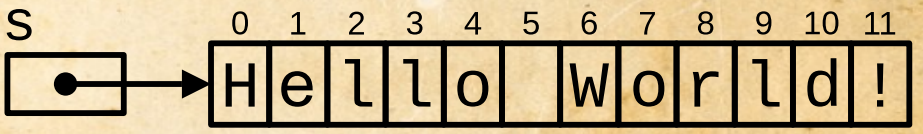
```
String s = "Hello World!";

String s;
s = "Hello World!";

String s = new String("Hello World!");

String s;
s = new String("Hello World!");
```

- The latter two are the “normal” way to declare and initialize objects.
  - The first ways are “sort cut” ways just for the `String` class objects.

Storage of Strings and the `equals` Method

- A `String` is an ***object*** in Java.
- Objects in Java are stored by ***reference***
  - This means the object data is stored in one place, and the ***variable*** contains a number that represents the ***location*** of the object.
  - The following is the code to declare and initialize a string, as well as a diagrammatic representation of the `String` object.

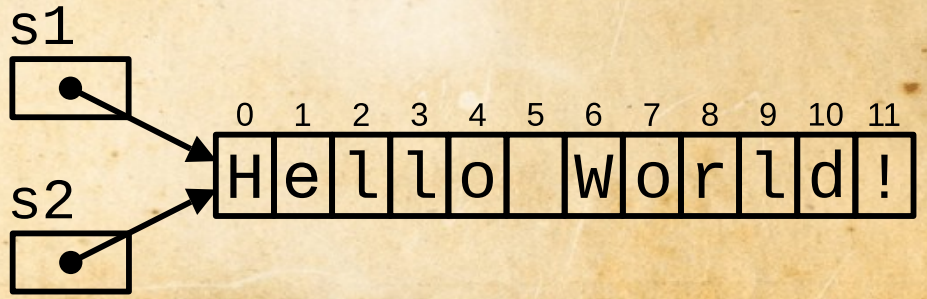
```
String s = "Hello World!";
```



Storage of Strings and the `equals` Method

- A `String` is an ***object*** in Java.
- Objects in Java are stored by ***reference***
- The Java compiler will optimize storage if the program uses the same string in different locations:

```
String s1 = "Hello World!";
String s2 = "Hello World!";
```

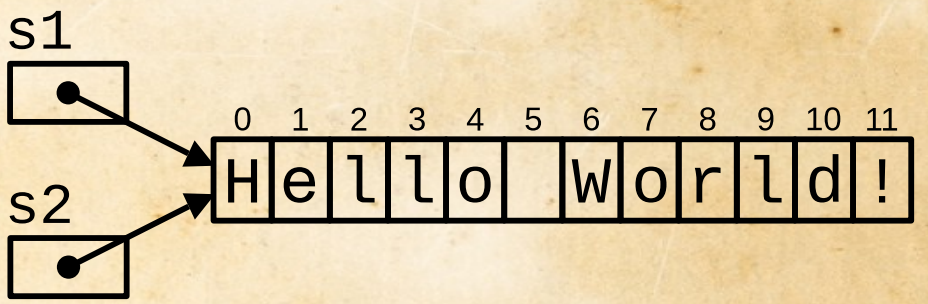


There is only one copy of the `string` *"*`Hello World!`*" stored in memory.*

Storage of Strings and the `equals` Method

- Since both S1 and S2 refer to the same location...

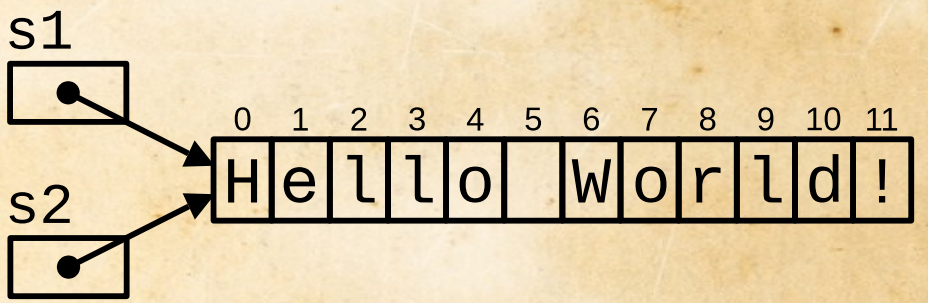
```
String s1 = "Hello World!";
String s2 = "Hello World!";
System.out.println( s1 == s2 );
```



Storage of Strings and the `equals` Method

- Since both S1 and S2 refer to the same location...

```
String s1 = "Hello World!";
String s2 = "Hello World!";
System.out.println( s1 == s2 );
```

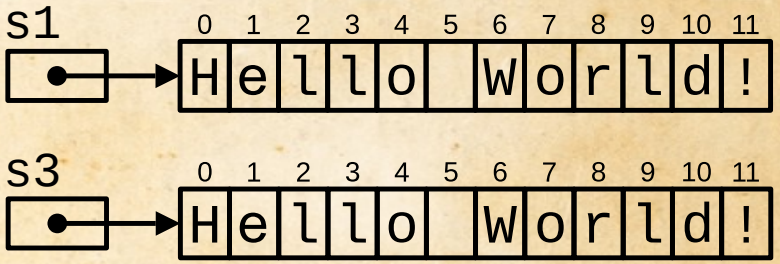



```
true
```

Storage of Strings and the `equals` Method

- When we use the `new` keyword to initialize a string, the compiler will not do the same optimization.

```
String s1 = "Hello World!";
String s3 = new String ("Hello World!");
System.out.println( s1 == s3 );
```



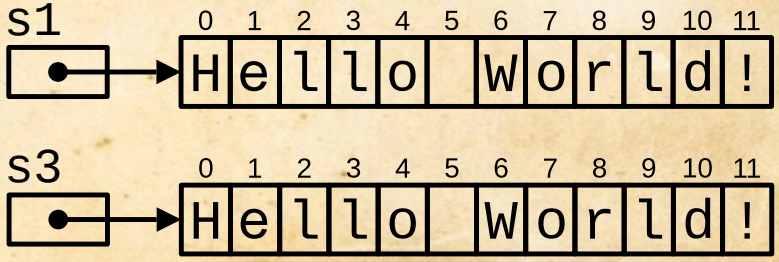
There are two copies of the `string` *“*`Hello World!`*” stored in memory.*

```
false
```

Storage of Strings and the `equals` Method

- If you wish to compare the ***value*** of the data stored in two different objects, use the **`equals`** method.
  - For `String` objects, **`equals`** compares character-by-character to decide if two strings are the same.

```
String s1 = "Hello World!";
String s3 = new String ("Hello World!");
System.out.println( s1 == s3 );
System.out.println( s1.equals(s3) );
```




```
false
true
```

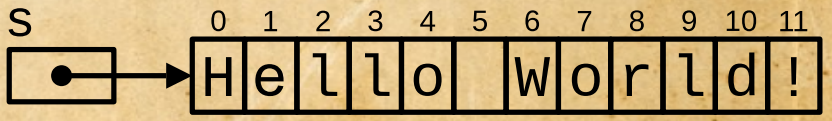
Storage of Strings and the `equals` Method

- Recall using the **`Math`** class methods…

```
int x = Math.sqrt(5);
int y = Math.abs(x);
```

- How does the usage of these **`String`** class methods differ?

```
String s = "Hello World!";
int x = s.length();
int y = s.indexOf("l");
```



Storage of Strings and the `equals` Method

- Recall using the **`Math`** class methods…

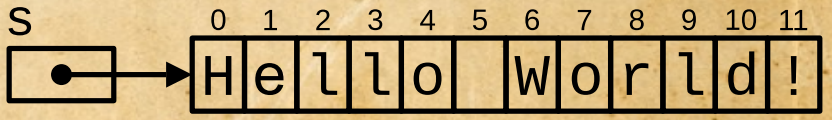
```
int x = Math.sqrt(5);
int y = Math.abs(x);
```

`Math` *is the* ***class*** *identifier*.

- How does the usage of these **`String`** class methods differ?

```
String s = "Hello World!";
int x = s.length();
int y = s.indexOf("l");
```

`s` *is the* ***object*** *identifier*.



- `String` methods operate on a `String` object, above this is object "`s`".
- We don't have a `Math` object, only a `Math` class.

`String` Class – Calling Methods

- Now we discuss the `compareTo` method:

```
String toUpperCase()
String toLowerCase()
int length()
char charAt(int index)
int indexOf(String str)
int indexOf(String str, int start)
String substring(int start)
String substring(int start, int end)
boolean equals(String other)
boolean equalsIgnoreCase(String other)
int compareTo(String other)
```

`String` Class – `compareTo()`

- If we wish to put two strings in alphabetical order, we can use the `compareTo` method.
- The return value for `s1.compareTo(s2)` is an integer that is:
  - Equal to zero (`0`) if the strings are exactly equal
  - A positive value if `s1` is earlier than `s2`, alphabetically
  - A negative value if `s1` is later than `s2`, alphabetically.

    (upper case comes before lower case)

`String` Class – `compareTo()`

- If we wish to put two strings in alphabetical order, we can use the `compareTo` method.
- The return value for `s1.compareTo(s2)` is an integer that is:
  - Equal to zero (`0`) if the strings are exactly equal
  - A positive value if `s1` is earlier than `s2`, alphabetically
  - A negative value if `s1` is later than `s2`, alphabetically.

In actuality, it compares the ***unicode*** values of each character of the string, one-by-one, and returns the value of the first character that differs:

```
s1.charAt(k) – s2.charAt(k)
```

Or, if they're the same until the end of a shorter string:

```
s1.length() – s2.length()
```

`String` Class – `compareTo()`

ASCII Character Set (0x20-0x7F)

hex	char	hex	char	hex	char	hex	char	hex	char	hex
20	*space*	30	0	40	@	50	P	60	`	70
21	!	31	1	41	A	51	Q	61	a	71
22	“	32	2	42	B	52	R	62	b	72
23	#	33	3	43	C	53	S	63	c	73
24	$	34	4	44	D	54	T	64	d	74
25	%	35	5	45	E	55	U	65	e	75
26	&	36	6	46	F	56	V	66	f	76
27	'	37	7	47	G	57	W	67	g	77
28	(	38	8	48	H	58	X	68	h	78
29	)	39	9	49	I	59	Y	69	i	79
2A	*	3A	:	4A	J	5A	Z	6A	j	7A
2B	+	3B	;	4B	K	5B	[	6B	k	7B
2C	,	3C	<	4C	L	5C	\	6C	l	7C
2D	-	3D	=	4D	M	5D	]	6D	m	7D
2E	.	3E	>	4E	N	5E	^	6E	n	7E
2F	/	3F	?	4F	O	5F	_	6F	o	7F

`String` Class – `compareTo()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 72 (0x48) and unicode 104 (0x68)

`String` Class – `compareTo()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 72 (0x48) and unicode 104 (0x68)

```
-32
```

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The difference between unicode 104 (0x68) and unicode 72 (0x48)

```
-32
32
```

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

```
-32
32
-7
```

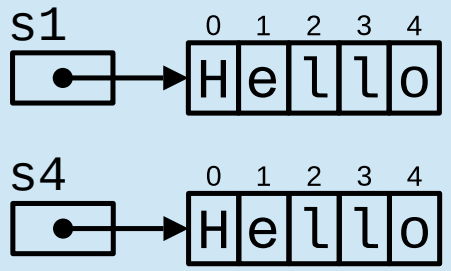
The difference in length between s1 and s2 (5 – 12)

`String` Class – `equals()`

- Determine the output of each:

```
String s1 = "Hello";
String s2 = "hello";
String s3 = "Hello World!";
String s4 = new String("Hello");
System.out.println(s1.compareTo(s2));
System.out.println(s2.compareTo(s1));
System.out.println(s1.compareTo(s3));
System.out.println(s1 == s4);
System.out.println(s1.equals(s4));
System.out.println(s1.compareTo(s4));
```

The *references* are different, but the data stored in each is the same.




```
-32
32
-7
false
true
0
```

Using Classes

String Class (part 2)