Lesson 55: `String` Methods and Properties #1 (W19D1)

Balboa High School

Michael Ferraro

January 4, 2016

Do Now

1. Create a new project called `Lesson55` in the `workspace2`.
2. Download `L55DoNow.java` from here, import into the new project.
3. Execute the compiled class from the terminal shell as specified in the comments.
4. Does it behave as expected?

Aim

Students will learn the about the properties of `Strings` in Java and delve into some of the methods made available by the `String` class.

Strings are Special!

`String`s have special properties. . .

- You don't need to use a formal constructor — `new String("?")`

Strings are Special!

`String`s have special properties. . .

- You don't need to use a formal constructor — `new String("?")`
- Like numerical values, they can be operated upon by '+'

`String`s are Special!

`String`s have special properties. . .

- You don't need to use a formal constructor — `new String("?")`
- Like numerical values, they can be operated upon by '+'
- They're **immutable**

Immutability of `Strings`

- immutable:

Immutability of `Strings`

- immutable:
    - im- = not
    - mut = mutate, change
    - -able = able to

Immutability of `Strings`

- immutable:
  - im- = not
  - mut = mutate, change
  - -able = able to
- ∴ **immutable**: cannot be changed

Immutability of `String`s

- immutable:
  - im- = not
  - mut = mutate, change
  - -able = able to
- ∴ **immutable**: cannot be changed
- `String` objects, once created, **cannot be changed — ever!**

Immutability of Strings

- immutable:
  - im- = not
  - mut = mutate, change
  - -able = able to
- ∴ **immutable**: cannot be changed
- `String` objects, once created, **cannot be changed — ever!**
- Does the `"abc"` `String` object change in this case?

```
String myStr = "abc";
myStr += "123";
```

Immutability of Strings

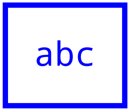
```
String myStr = "abc";

myStr += "123";
```

Immutability of Strings

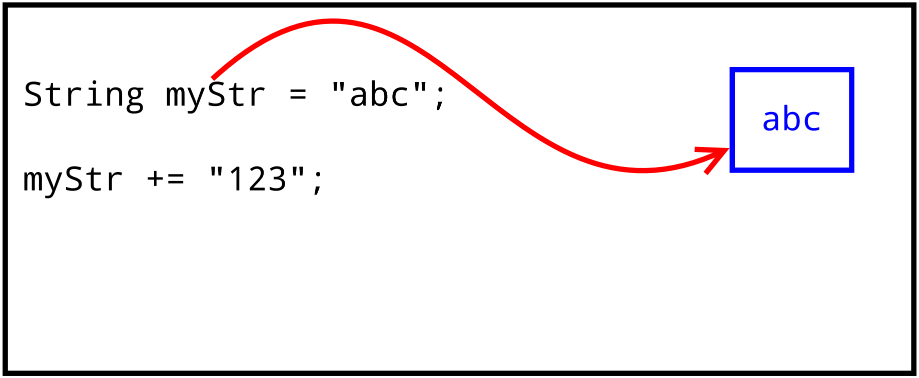
```
String myStr = "abc";

myStr += "123";
```



→ String object created in memory

Immutability of Strings

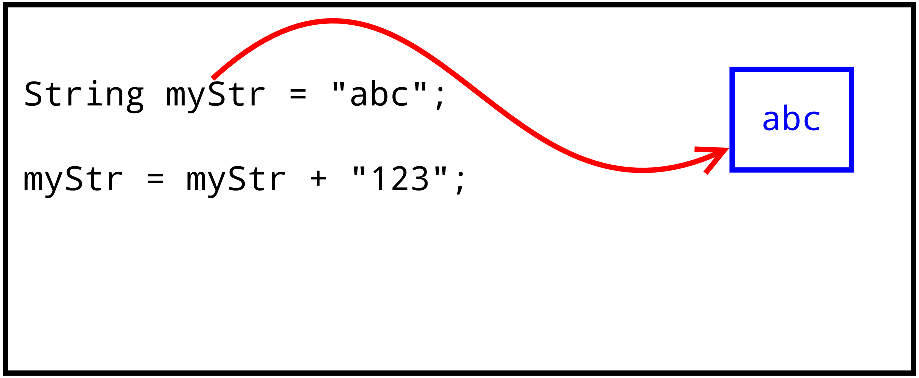


```
String myStr = "abc";

myStr += "123";
```

→ `myStr` reference to `String` object made

Immutability of Strings



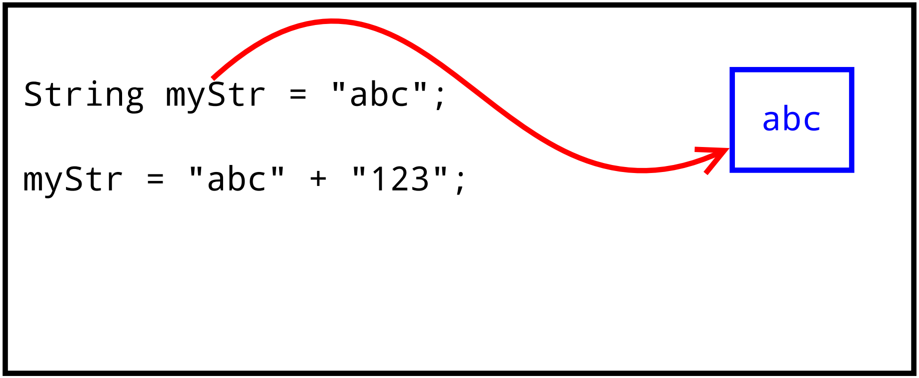
```
String myStr = "abc";

myStr = myStr + "123";
```

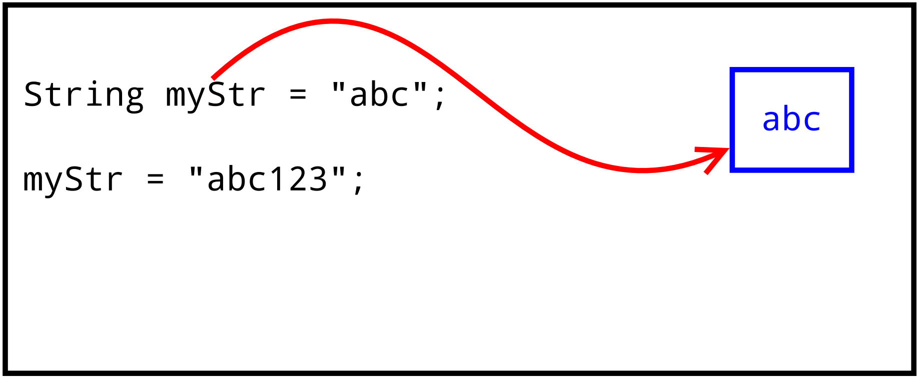
Immutability of Strings

```
String myStr = "abc";

myStr = "abc" + "123";
```



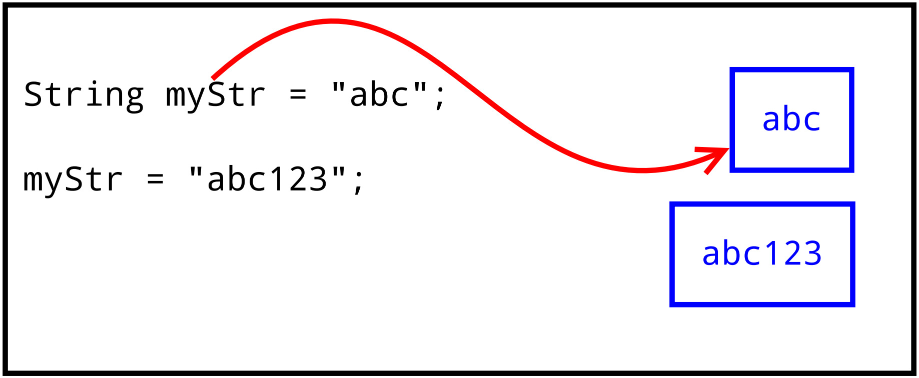
Immutability of Strings



```
String myStr = "abc";

myStr = "abc123";
```

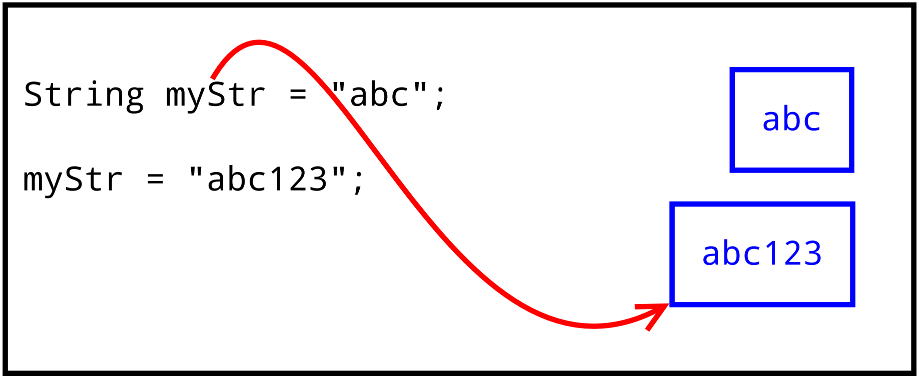
Immutability of Strings


```
String myStr = "abc";

myStr = "abc123";
```

→ new `String` object made in memory — original is not changed, it's **immutable!**

Immutability of Strings

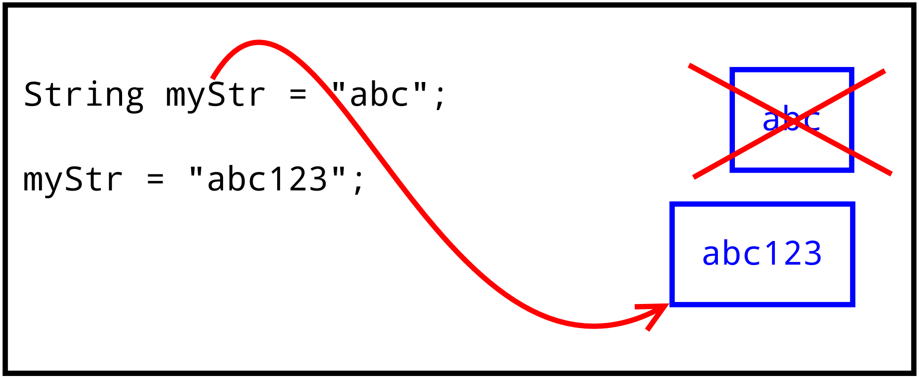


```
String myStr = "abc";

myStr = "abc123";
```

→ `myStr` reference updated, original object **dereferenced**

Immutability of Strings



```
String myStr = "abc";

myStr = "abc123";
```

→ original object is **garbage collected**

Immutability of Strings

- What happens now? (Try it!)

```
String myStr1 = "abc";
String myStr2 = myStr1; //two refs to
                        //same obj
myStr1 += "123";

System.out.println(myStr1);
System.out.println(myStr2);
```

Immutability of Strings

- What happens now? (Try it!)

```
String myStr1 = "abc";
String myStr2 = myStr1; //two refs to
                        //same obj
myStr1 += "123";

System.out.println(myStr1);
System.out.println(myStr2);
```

- When `myStr1` is pointed to a new object (with value `"abc123"`), `myStr2` is still pointing to the original `String` object (with value `"abc"`).

Immutability of Strings

- What happens now? (Try it!)

```
String myStr1 = "abc";
String myStr2 = myStr1; //two refs to
                        //same obj
myStr1 += "123";

System.out.println(myStr1);
System.out.println(myStr2);
```

- When `myStr1` is pointed to a new object (with value `"abc123"`), `myStr2` is still pointing to the original `String` object (with value `"abc"`).
- The `"abc"` `String` will not be garbage-collected. Why?

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

-
  ```
  String a = "Twinkie!";
  System.out.println( a.length() );
  ```

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

- `String a = "Twinkie!";`
  `System.out.println( a.length() );` $\leftarrow$ **8**

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

-
  ```
  String a = "Twinkie!";
  System.out.println( a.length() );
  ```
  $\leftarrow$ **8**

-
  ```
  String b = "";
  System.out.println( b.length() );
  ```

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

-
  ```
  String a = "Twinkie!";
  System.out.println( a.length() );
  ```
  $\leftarrow$ **8**

-
  ```
  String b = "";
  System.out.println( b.length() );
  ```
  $\leftarrow$ **0**

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

-
  ```
  String a = "Twinkie!";
  System.out.println( a.length() );
  ```
  $\leftarrow$ **8**

-
  ```
  String b = "";
  System.out.println( b.length() );
  ```
  $\leftarrow$ **0**

-
  ```
  String c = null;
  System.out.println( c.length() );
  ```

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

- `String a = "Twinkie!";`
  `System.out.println( a.length() );` ← **8**
- `String b = "";`
  `System.out.println( b.length() );` ← **0**
- `String c = null;`
  `System.out.println( c.length() );` ← **NullPointerException**

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

- `String a = "Twinkie!";`
  `System.out.println( a.length() );` ← **8**
- `String b = "";`
  `System.out.println( b.length() );` ← **0**
- `String c = null;`
  `System.out.println( c.length() );` ← **NullPointerException**
- `System.out.println( "igloos are cold".length() );`

String Methods: `length()`

One you already know: `length()`

Predict the printed output:

- `String a = "Twinkie!";`
  `System.out.println( a.length() );` ← **8**
- `String b = "";`
  `System.out.println( b.length() );` ← **0**
- `String c = null;`
  `System.out.println( c.length() );` ← **NullPointerException**
- `System.out.println( "igloos are cold".length() );` ← **15**

String Methods: `trim()`

A new one: `trim()`

- Returns a **copy** with leading and trailing whitespace — i.e., spaces (`" "`), tabs (`\t`), & newlines (`\n`) — removed

String Methods: `trim()`

A new one: `trim()`

- Returns a **copy** with leading and trailing whitespace — i.e., spaces (`" "`), tabs (`\t`), & newlines (`\n`) — removed
- Example: (Try it!)

```
String s1 = "\t hello there ";
System.out.println("*" + s1 + "*");
System.out.println("*" + s1.trim() + "*");
// was s1 modified?
```

String Methods: `trim()`

A new one: `trim()`

- Returns a **copy** with leading and trailing whitespace — i.e., spaces (`" "`), tabs (`\t`), & newlines (`\n`) — removed
- Example: (Try it!)

```
String s1 = "\t hello there ";
System.out.println("*" + s1 + "*");
System.out.println("*" + s1.trim() + "*");
// was s1 modified?
```

- Very useful when importing lines of text from a text file (file I/O is later in this unit)

String Methods: `equals()`

- Use `equals()` when you want to see if the *value* of two `Strings` is the same; returns a `boolean`

String Methods: `equals()`

- Use `equals()` when you want to see if the *value* of two `Strings` is the same; returns a `boolean`
- Ex: `s1.equals(s2)`

String Methods: `equals()`

- Use `equals()` when you want to see if the *value* of two `Strings` is the same; returns a `boolean`
- Ex: `s1.equals(s2)`
- Fix the `L55DoNow` code so that you get the intended result!

String Methods: `equals()`

- So what was going on with `"=="` in the original form of the Do Now?

String Methods: `equals()`

- So what was going on with `"=="` in the original form of the Do Now?
- == was comparing the memory addresses of the two objects, seeing if they were literally pointing to the same object.

String Methods: equals()

- So what was going on with "==" in the original form of the Do Now?
- == was comparing the memory addresses of the two objects, seeing if they were literally pointing to the same object.
- What happens now?

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

if ( s1 == s2 ) {
   System.out.println("s1 & s2 refer to same obj");
} else {
   System.out.println("s1 & s2 refer to unique objs");
}
```

String Methods: `equals()`

In practice, "==" works *most* of the time like `equals()` due to a Java feature called *interning*

String Methods: `equals()`

In practice, “==” works *most* of the time like `equals()` due to a Java feature called *interning*

- If a `String` is created with the same value as another already in existence, then the reference for the new one is pointed at the preexisting one (saves memory)

String Methods: `equals()`

In practice, "==" works *most* of the time like `equals()` due to a Java feature called *interning*

- If a `String` is created with the same value as another already in existence, then the reference for the new one is pointed at the preexisting one (saves memory)
- Don't *count* on "==" happening to work! Just use `equals()` instead to always be sure.

String Methods: `charAt()`

`charAt(pos)` returns the character at the specified `int` position.

Ex: `String str1 = "Scooby Doo!";`

- `char c = str1.charAt(0); //1st letter`

String Methods: `charAt()`

`charAt(pos)` returns the character at the specified `int` position.

Ex: `String str1 = "Scooby Doo!";`

- `char c = str1.charAt(0); //1st letter`
- `char d = str1.charAt(4); //'b'`

String Methods: `charAt()`

`charAt(pos)` returns the character at the specified `int` position.

Ex: `String str1 = "Scooby Doo!";`

- `char c = str1.charAt(0); //1st letter`
- `char d = str1.charAt(4); //'b'`
- `char e = str1.charAt( str1.length() );`
  `// TRY THIS ONE!`

`charAt()` Tasks

1. Write a method that takes a `String` and prints out every character, in order, but only one character per line. For example, sending "`abcdef`" will print

```
a
b
c
d
e
f
```

2. Write another method that takes a `String` and returns another `String` that is a reversed version of the original. For example, sending "`12345`" will return "`54321`".

HW

- Finish as much of PS #10, §§1-3, inclusive, as you can. If you run short on time, you can fill out the table in §3.3 later.
- Bring headphones/earbuds to next class if you're able. (I will have some to lend.)