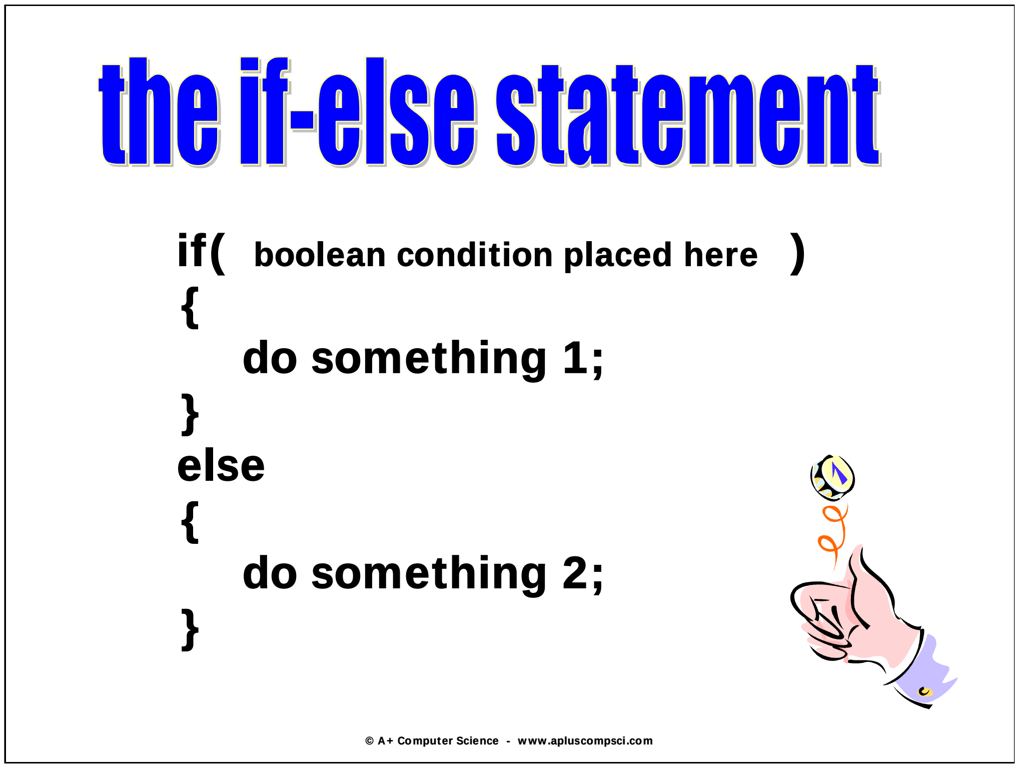
if-else Strings

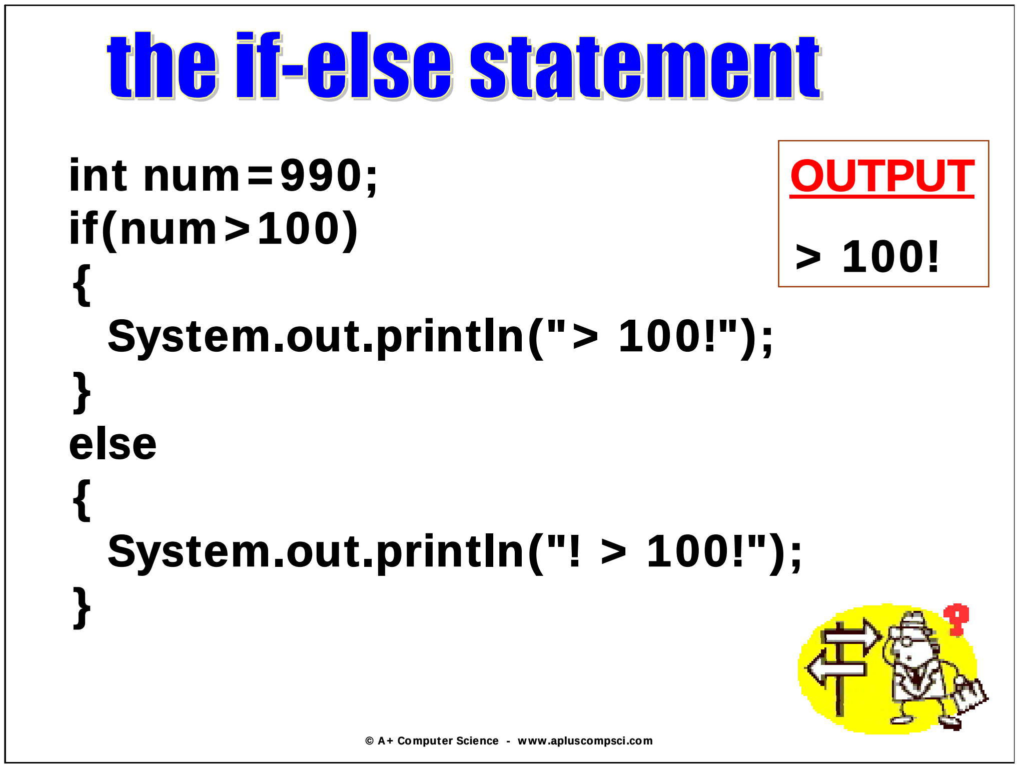
Lab 06



the if-else statement

```
if(  boolean condition placed here  )
{
    do something 1;
}
else
{
    do something 2;
}
```

If the condition is true, `do something 1` will occur.
If the condition is false, `do something 2` will occur.



the if-else statement

```
int num=990;
if(num>100)
{
  System.out.println("> 100!");
}
else
{
  System.out.println("! > 100!");
}
```

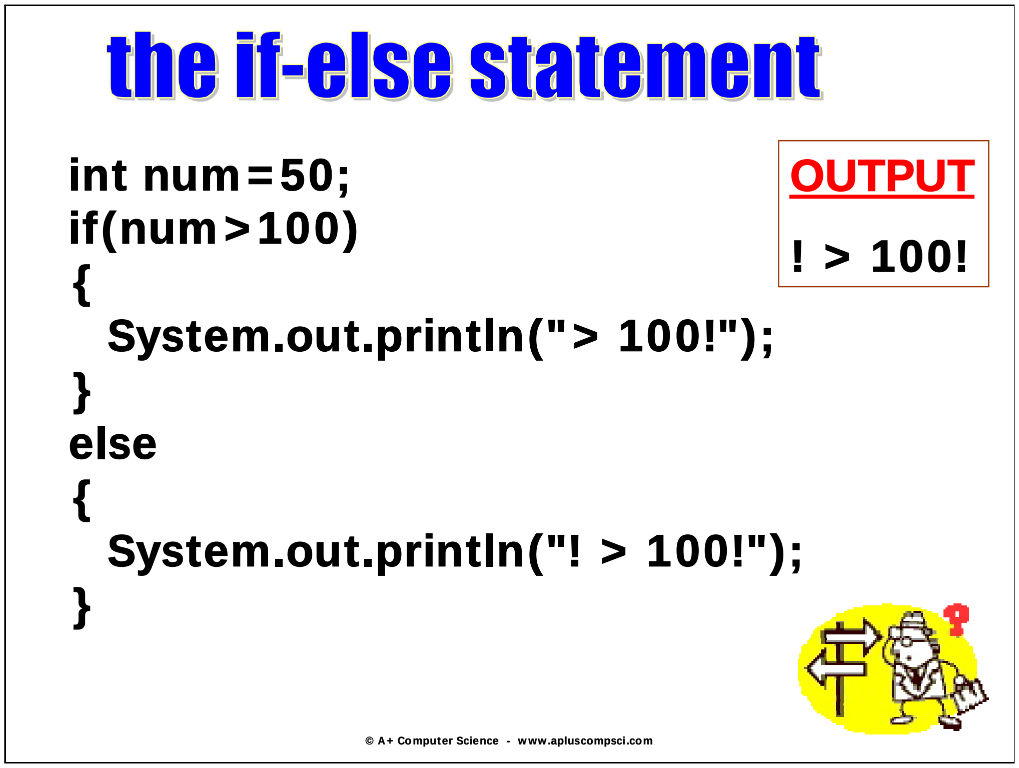
OUTPUT

```
> 100!
```

© A+ Computer Science - www.apluscompsci.com

If num is greater than 100, `>100!` is displayed.

If num is not greater than 100, `!>100!` is displayed.



the if-else statement

```
int num=50;
if(num>100)
{
  System.out.println("> 100!");
}
else
{
  System.out.println("! > 100!");
}
```

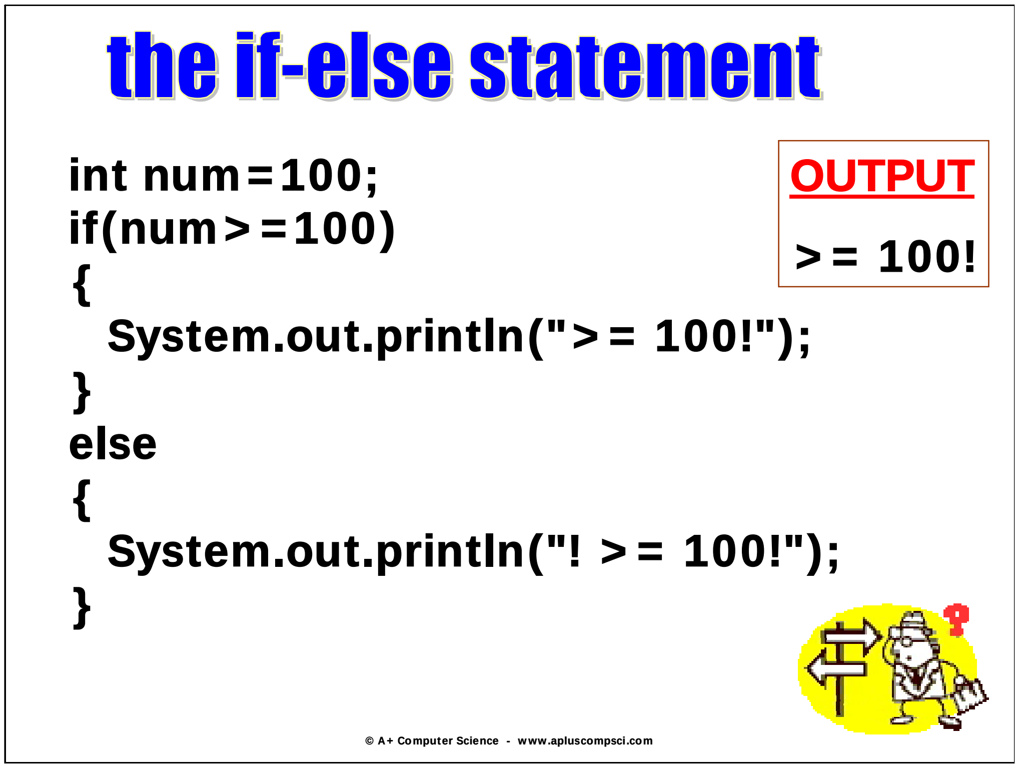
OUTPUT

```
! > 100!
```

© A+ Computer Science - www.apluscompsci.com

If num is greater than 100, `>100!` is displayed.

If num is not greater than 100, `!>100!` is displayed.



the if-else statement

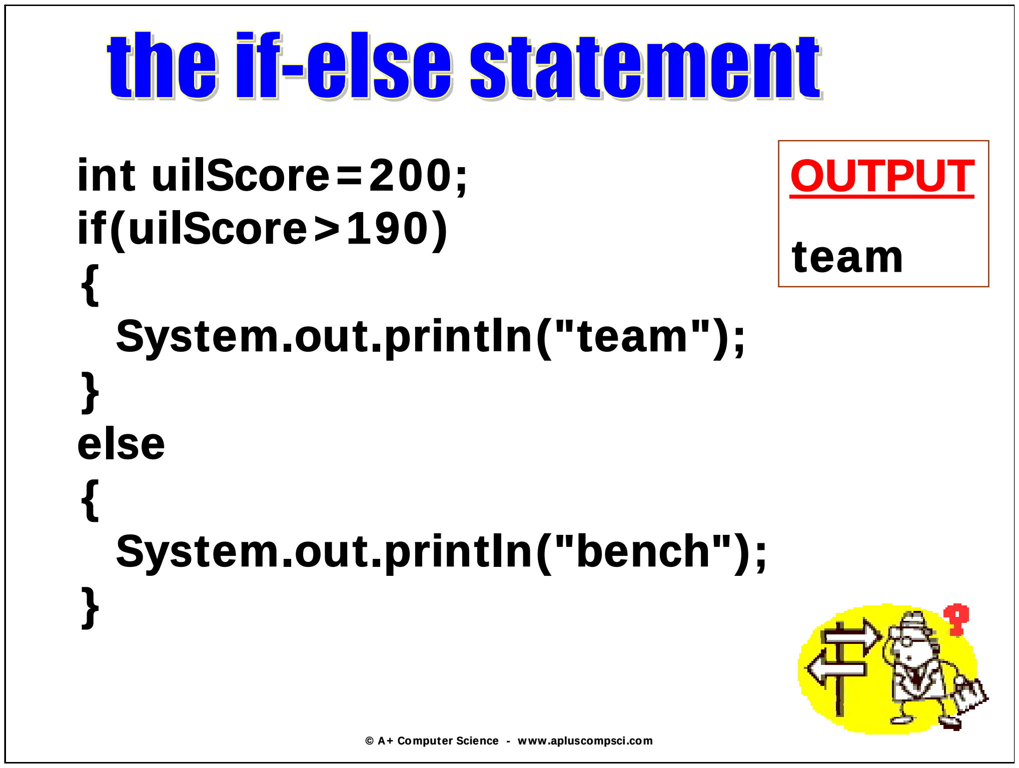
```
int num=100;
if(num>=100)
{
  System.out.println(">= 100!");
}
else
{
  System.out.println("! >= 100!");
}
```

OUTPUT

```
>= 100!
```

If num is greater than or equal to 100, `>=100!` is displayed.

If num is not greater than or equal to 100, `!>=100!` is displayed.



the if-else statement

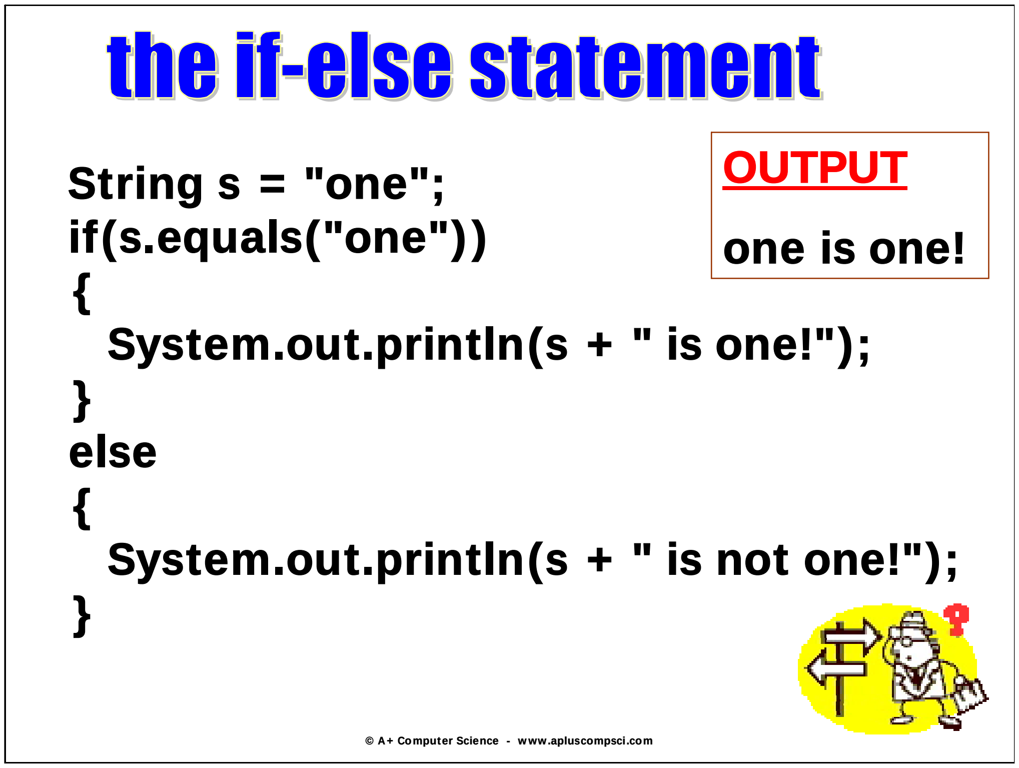
```
int uilScore=200;
if(uilScore>190)
{
  System.out.println("team");
}
else
{
  System.out.println("bench");
}
```

OUTPUT

```
team
```

If uilScore is greater than 190, `team` is displayed.

If uilScore is not greater than 190, `bench` is displayed.



the if-else statement

```
String s = "one";
if(s.equals("one"))
{
   System.out.println(s + " is one!");
}
else
{
   System.out.println(s + " is not one!");
}
```

OUTPUT

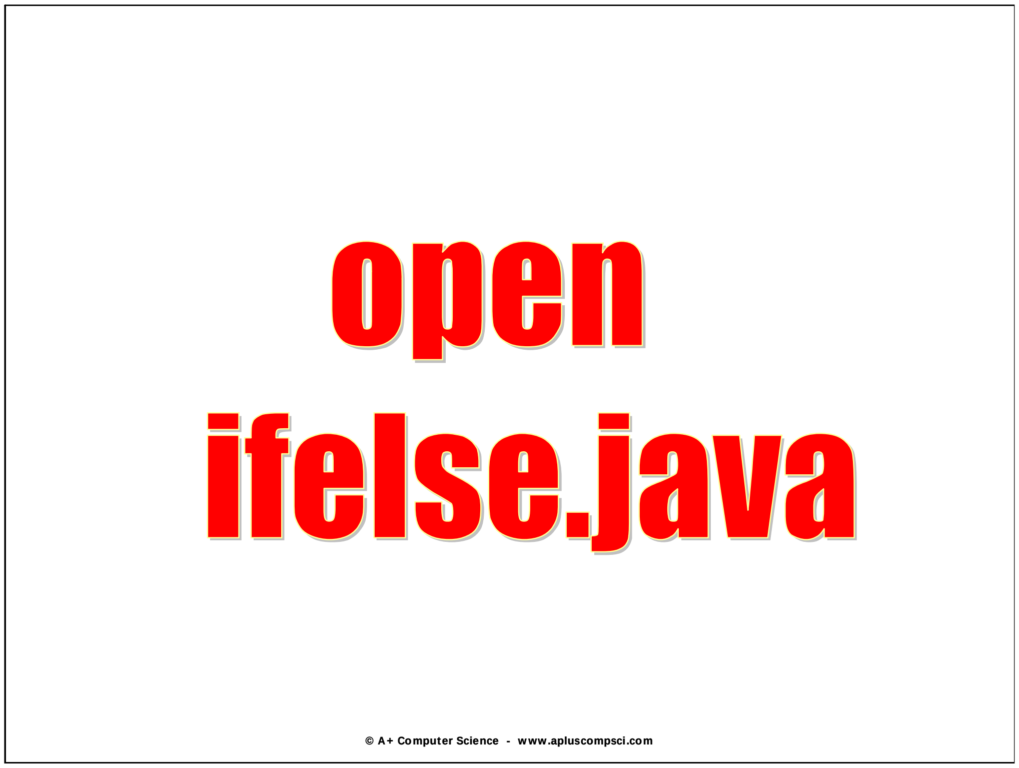
```
one is one!
```

`s` is a String reference. `s` stores the location/address of a String Object.

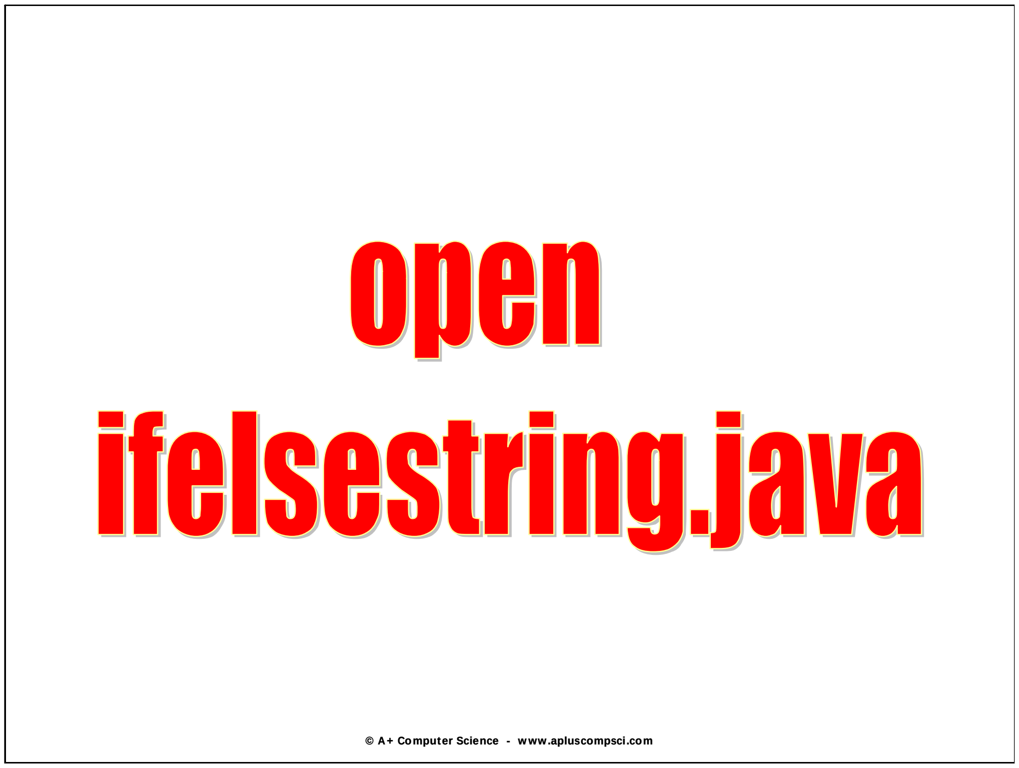
The `equals()` method compares the contents of two String Objects to see if they contain the same letters in the same order in the same case.

If `s` contains the letters `one`, `one is one!` is displayed.

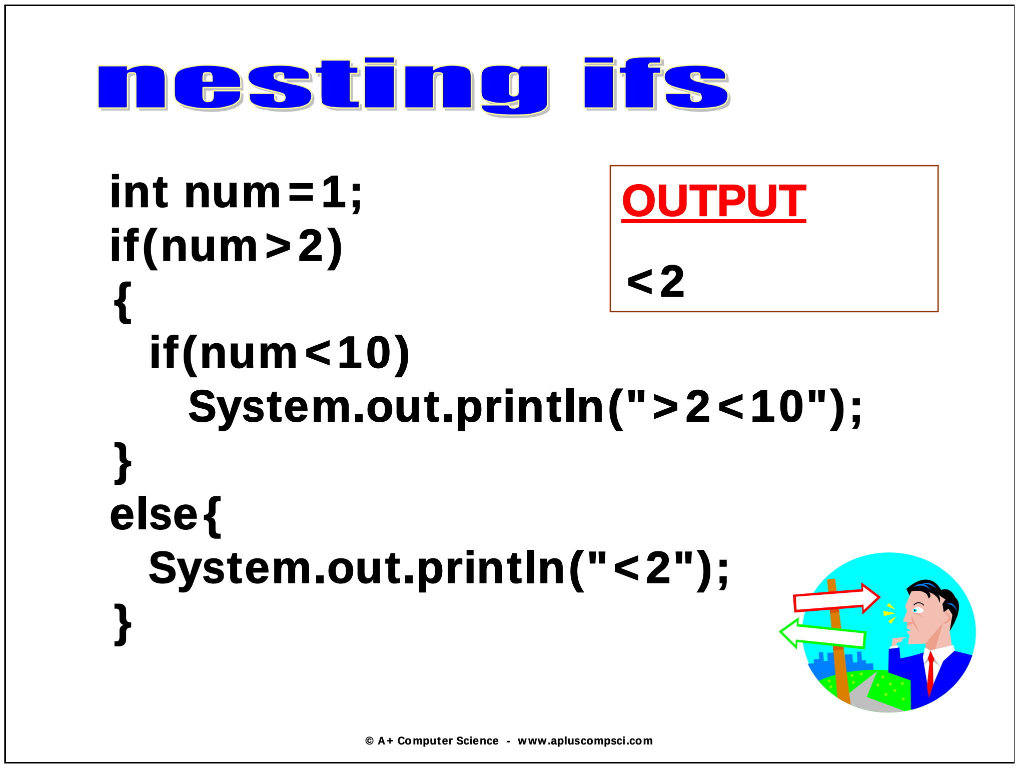
If `s` does not contain the letters `one`, `letters is not one!` is displayed.



open
ifelse.java



open ifelsestring.java



nesting ifs

```
int num=1;
if(num>2)
{
  if(num<10)
    System.out.println(">2<10");
}
else{
  System.out.println("<2");
}
```

OUTPUT

```
<2
```

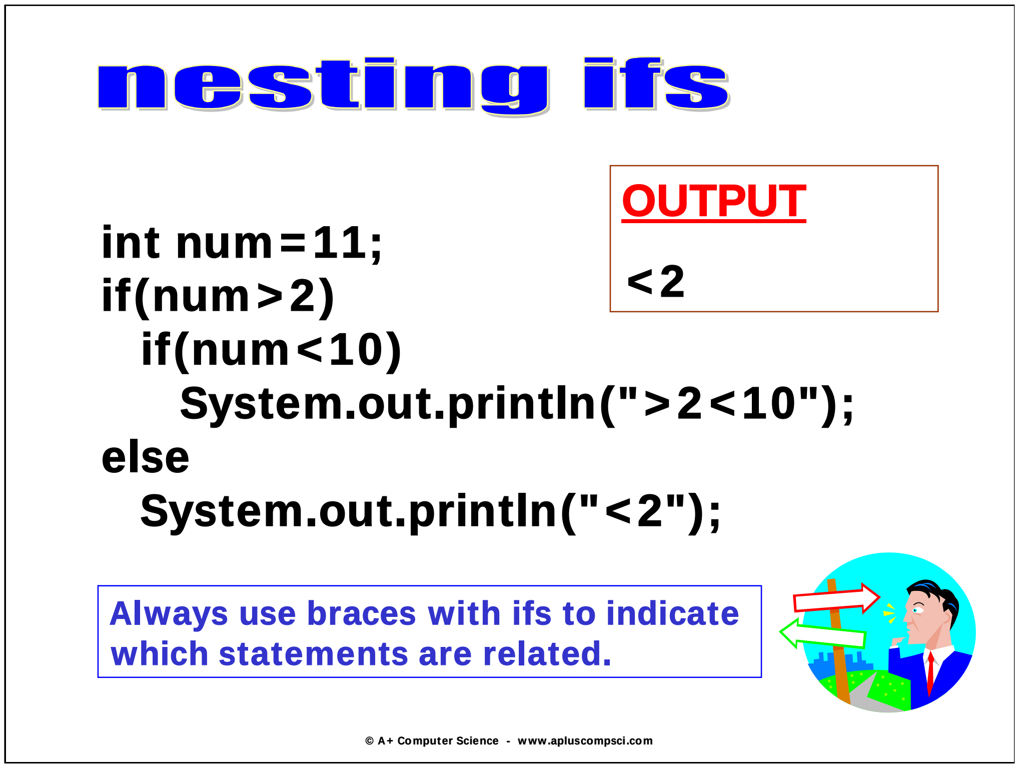
© A+ Computer Science - www.apluscompsci.com

Nesting occurs when one thing is placed inside of another thing.

`if(num<10)` has been nested inside of `if(num>2)`

`if(num<10)` will only be tested if `if(num>2)` is true.

The else is associated with `if(num>2).` Without the braces, the else would be associated with `if(num<10)` as if and else are paired based on proximity.



nesting ifs

```
int num=11;
if(num>2)
  if(num<10)
    System.out.println(">2<10");
else
  System.out.println("<2");
```

OUTPUT

```
<2
```

Always use braces with ifs to indicate which statements are related.

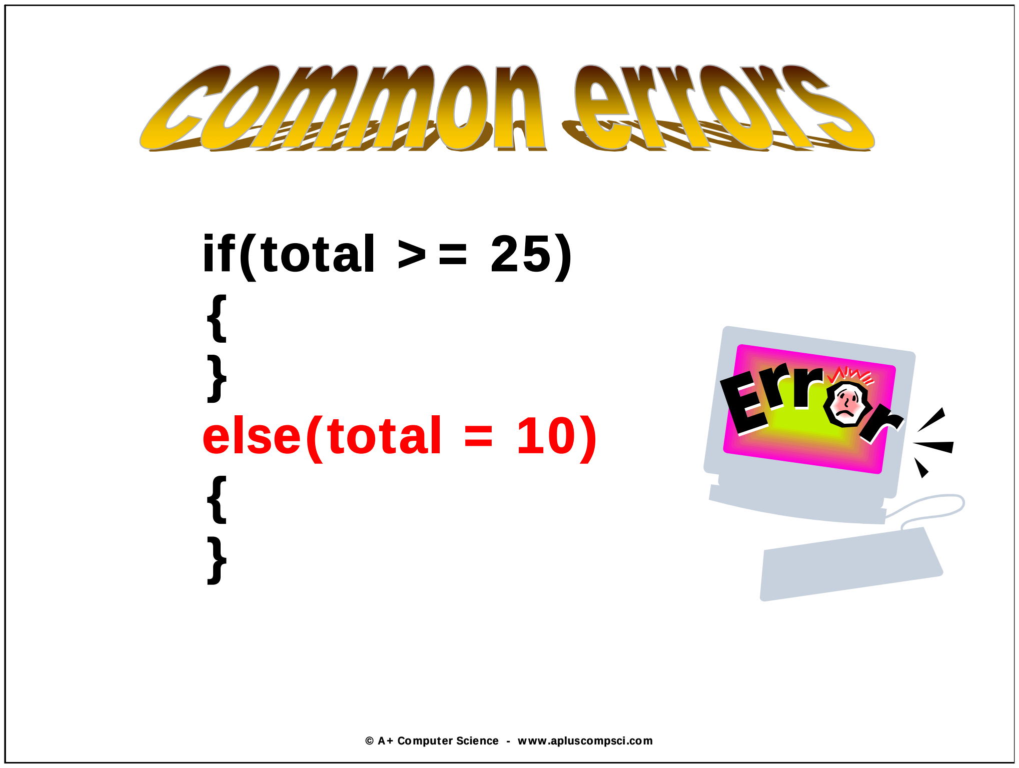
Nesting occurs when one thing is placed inside of another thing.

`if(num<10)` has been nested inside of `if(num>2)`

`if(num<10)` will only be tested if `if(num>2)` is true.

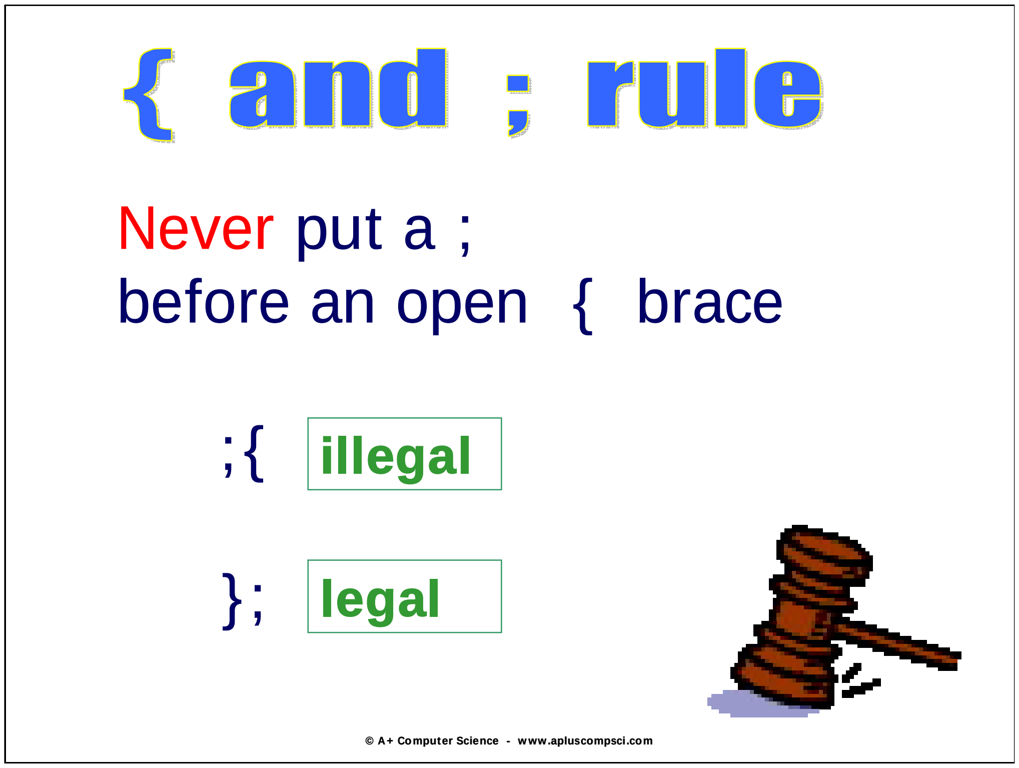
The else is associated with `if(num<10).` If braces were present around `if(num<10)`, the else would be associated with `if(num>2)`as if and else are paired based on proximity.

open
ifnesting.java
danglingelse.java



common errors

```
if(total >= 25)
{
}
else(total = 10)
{
}
```

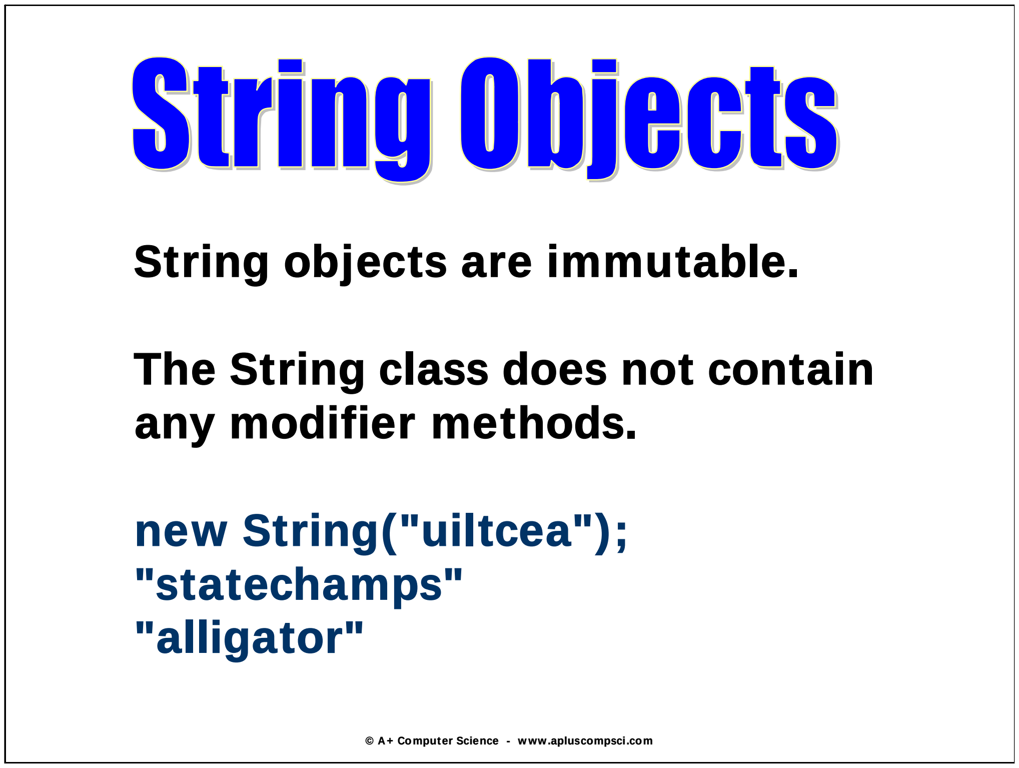


{ and ; rule

Never put a ;
before an open { brace

;{ **illegal**

}; **legal**



String Objects

String objects are immutable.

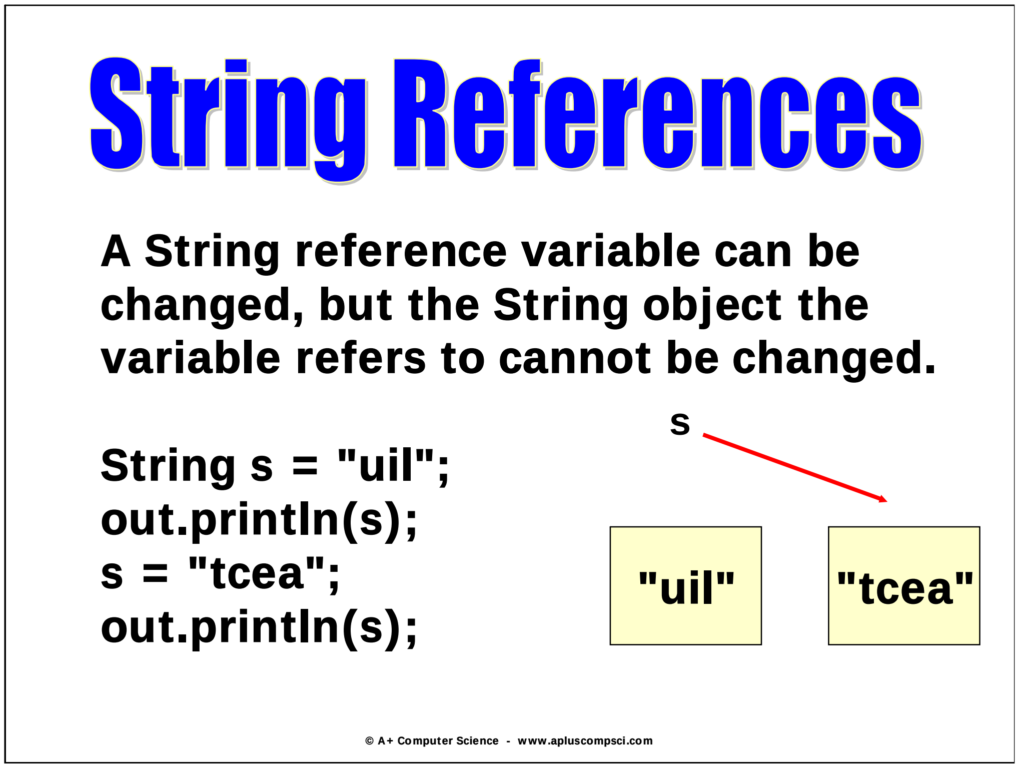
The String class does not contain any modifier methods.

```
new String("uiltcea");
"statechamps"
"alligator"
```

© A+ Computer Science - www.apluscompsci.com

Once a String Object has been instantiated, that String Object can never be modified.

The String class does not contain any modifier methods.



String References

A String reference variable can be changed, but the String object the variable refers to cannot be changed.

```
String s = "uil";
out.println(s);
s = "tcea";
out.println(s);
```

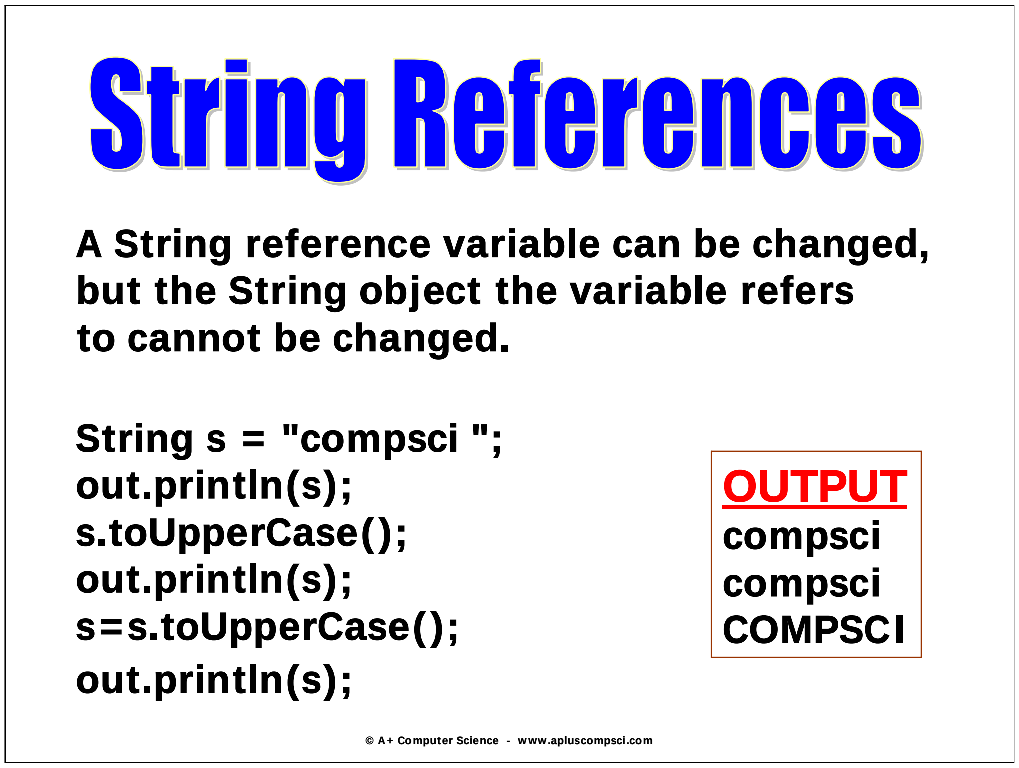
`s` is a reference that refers to a String Object.

`s` starts out referring to String Object `uil`.

`s` stores the location/address of String Object `uil`.

`s` is then referred to the String Object `tcea`.

`s` stores the location/address of String Object `tcea`.



String References

A String reference variable can be changed, but the String object the variable refers to cannot be changed.

```
String s = "compsci ";
out.println(s);
s.toUpperCase();
out.println(s);
s=s.toUpperCase();
out.println(s);
```

OUTPUT

```
compsci
compsci
COMPSCI
```

© A+ Computer Science - www.apluscompsci.com

`s` is a reference that refers to a String Object.

`s` starts out referring to String Object `compsci`.

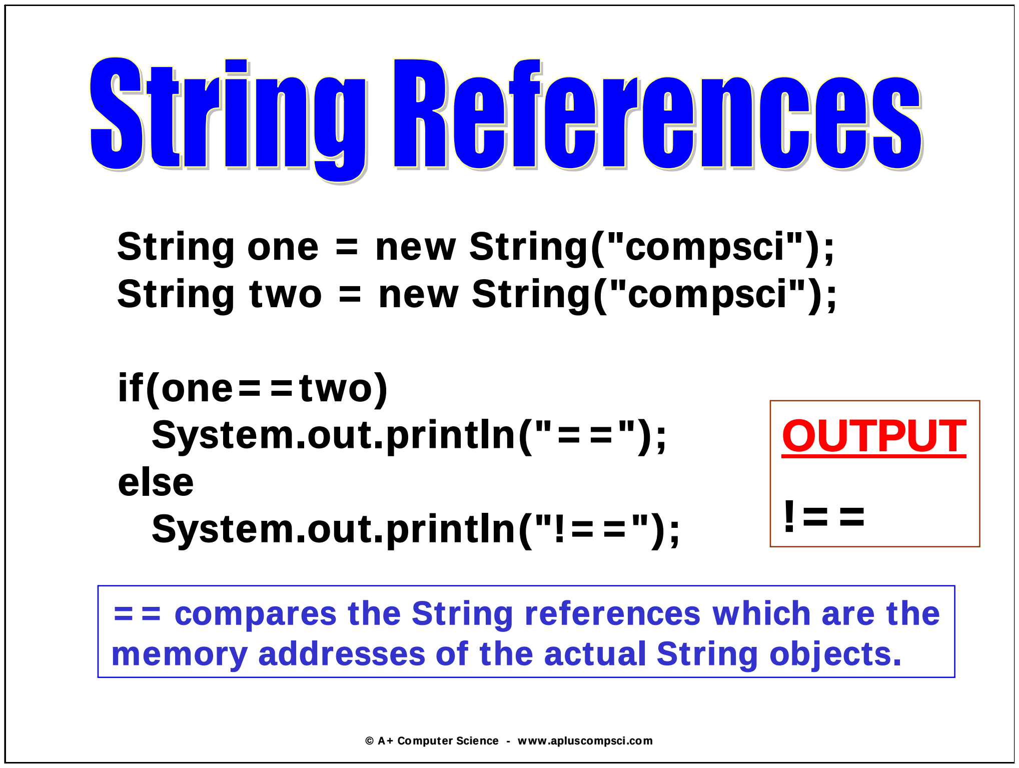
`s.toUpperCase()` returns a new String `COMPSCI`.

`s` is still referring to String Object `compsci`.

`s` is referred to `s.toUpperCase()`.

s.toUpperCase() returns a new String `COMPSCI`.

`s` is now referring to String Object `COMPSCI`.



String References

```
String one = new String("compsci");
String two = new String("compsci");

if(one==two)
  System.out.println("==");
else
  System.out.println("!==");
```

OUTPUT

```
!==
```

== compares the String references which are the memory addresses of the actual String objects.

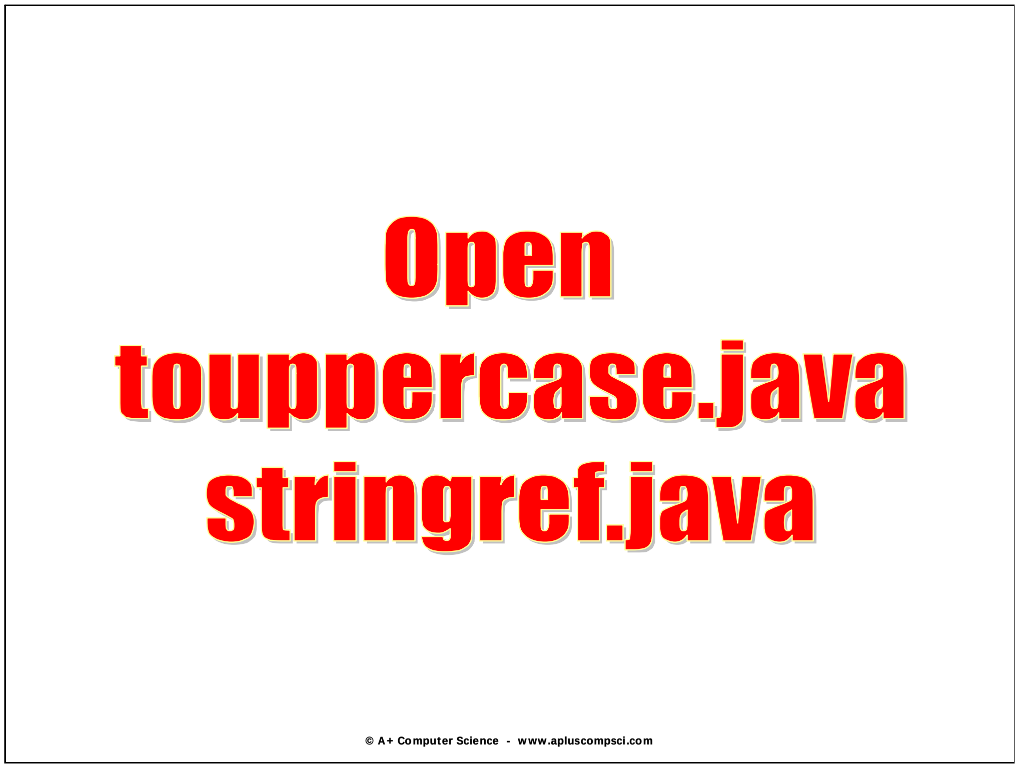
© A+ Computer Science - www.apluscompsci.com

`one` is a reference that refers to a String Object.

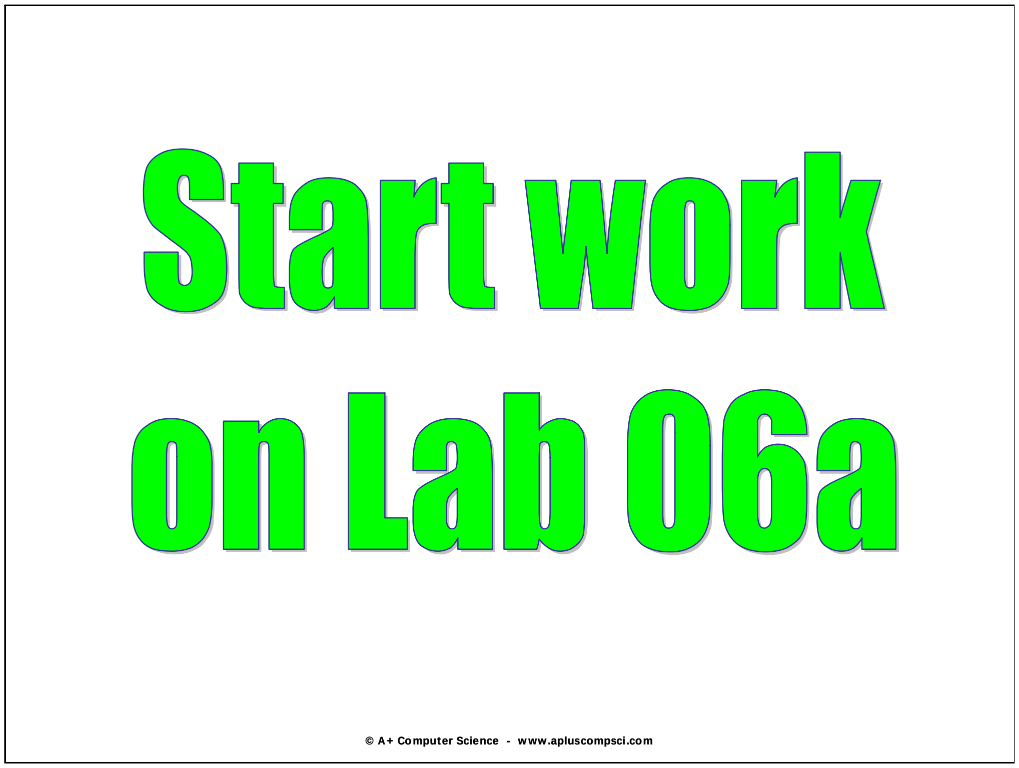
`two` is a reference that refers to a String Object.

`one` starts out referring to a String Object `compsci`.
`two` starts out referring to a different String Object `compsci`.

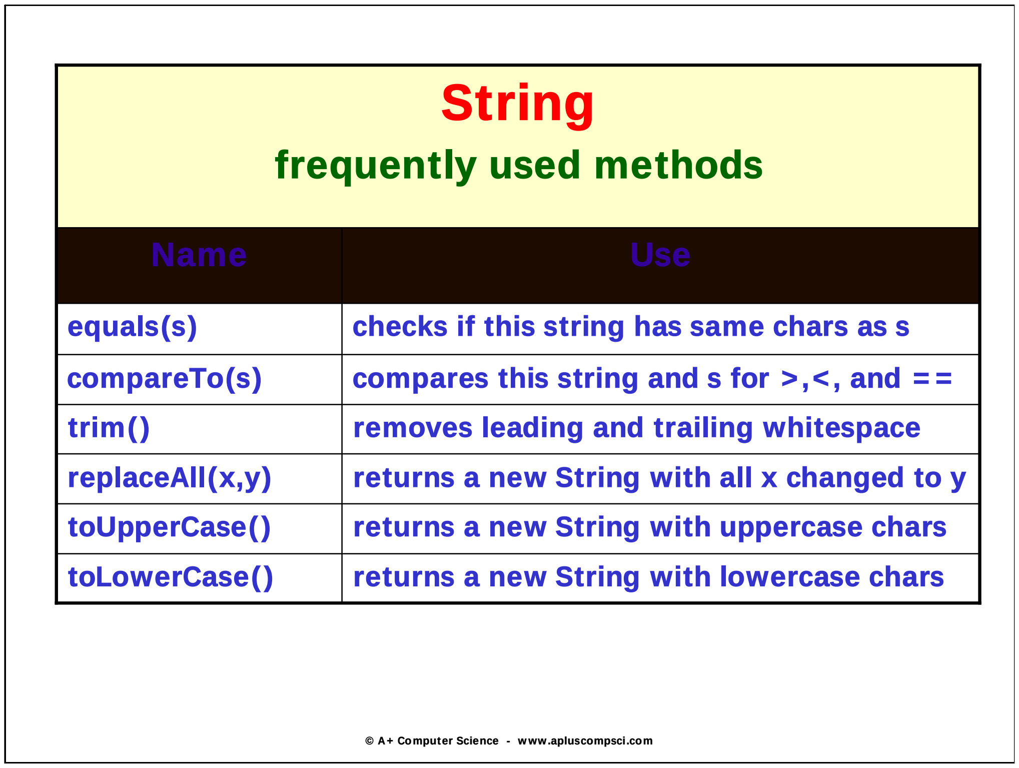
`one==two` compares the locations/addresses stored in one and two.
`one` and `two` do not store the same location/address.



Open touppercase.java stringref.java



Start work on Lab 06a

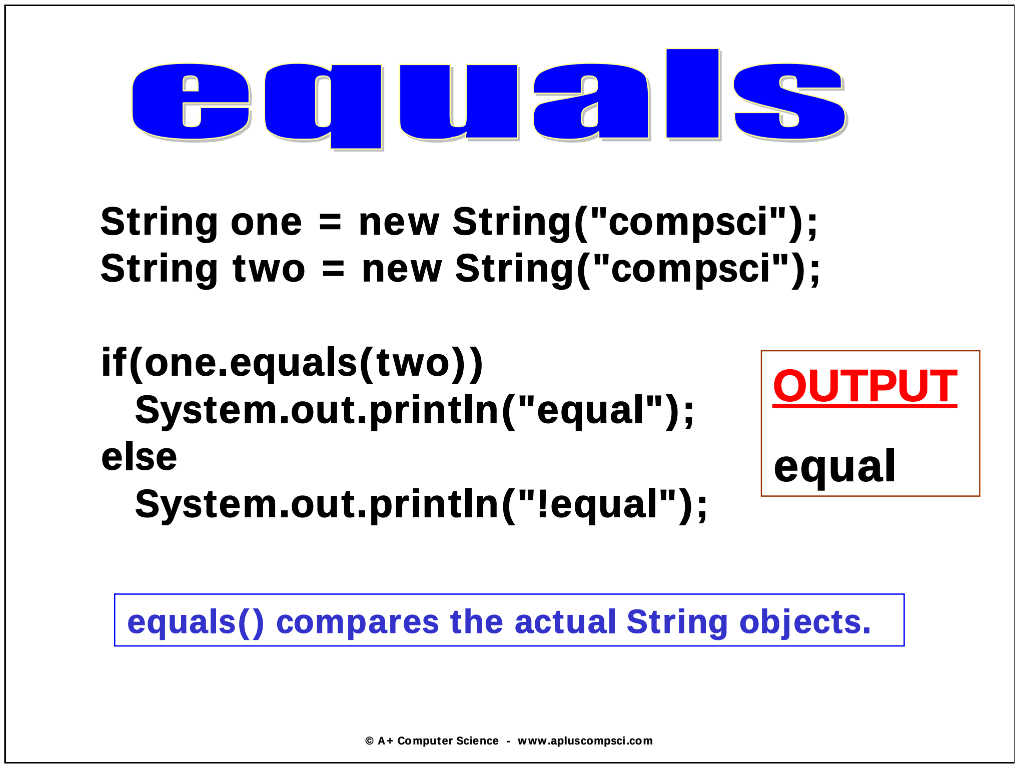


String
frequently used methods

Name	Use
equals(s)	checks if this string has same chars as s
compareTo(s)	compares this string and s for >,<, and ==
trim()	removes leading and trailing whitespace
replaceAll(x,y)	returns a new String with all x changed to y
toUpperCase()	returns a new String with uppercase chars
toLowerCase()	returns a new String with lowercase chars

The chart above lists some very common and very useful String class methods.

`equals()` and `compareTo()` are used quite often.
`trim()` and `replaceAll()` are very useful, but that widely used.
`toUpperCase()` and `toLowerCase()` can be very useful in certain situations.



equals

```
String one = new String("compsci");
String two = new String("compsci");

if(one.equals(two))
  System.out.println("equal");
else
  System.out.println("!equal");
```

OUTPUT

```
equal
```

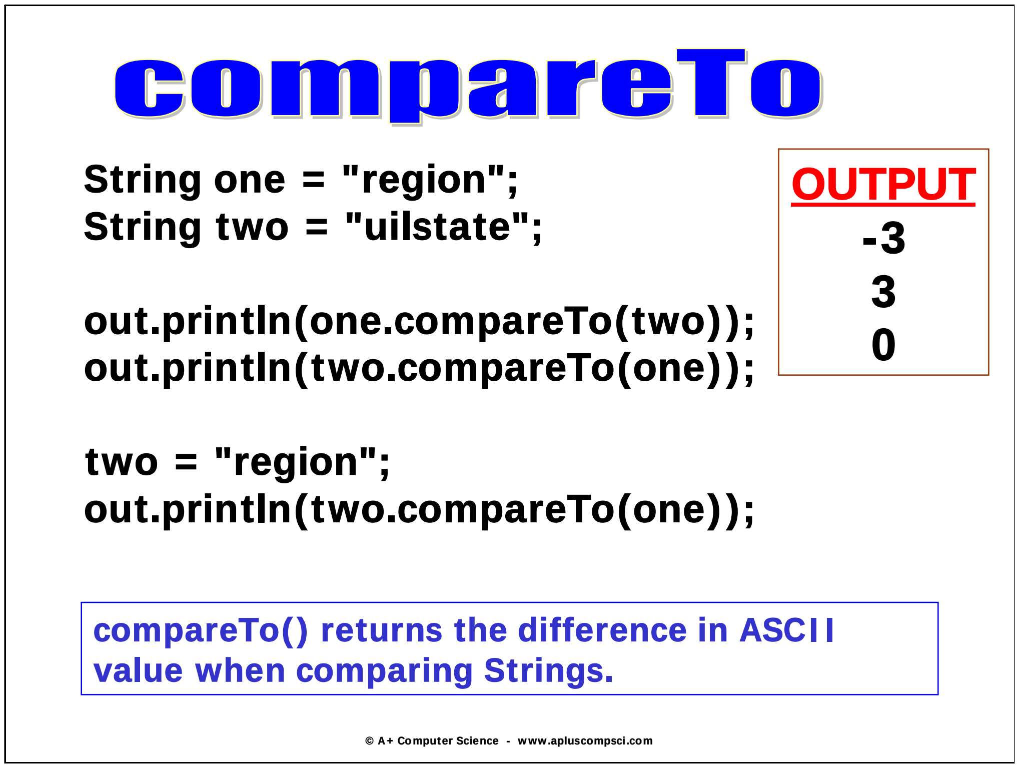
equals() compares the actual String objects.

`one` is a reference that refers to a String Object.

`two` is a reference that refers to a String Object.

`one` starts out referring to a String Object `compsci`.
`two` starts out referring to a different String Object `compsci`.

`one.equals(two)` compares the contents of the String Objects referred to by `one` and `two`.
`one` and `two` both refer to String Objects `compsci`.
`one.equals(two)` is true.



compareTo

```
String one = "region";
String two = "uilstate";

out.println(one.compareTo(two));
out.println(two.compareTo(one));

two = "region";
out.println(two.compareTo(one));
```

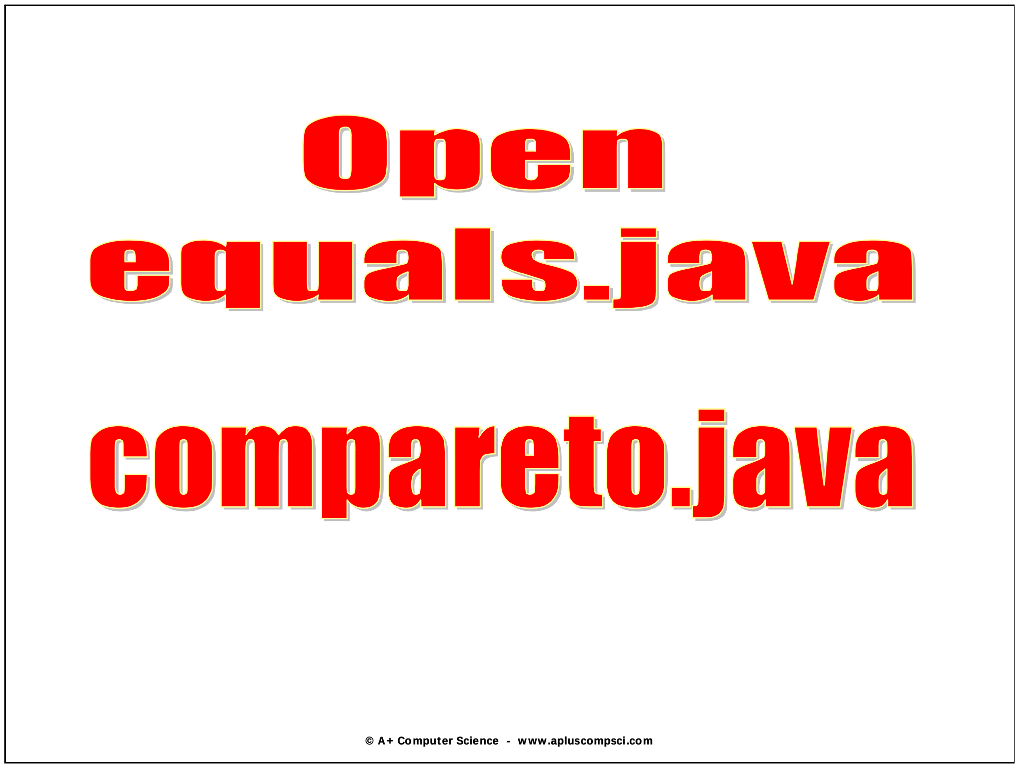
OUTPUT

```
-3
3
0
```

compareTo() returns the difference in ASCII value when comparing Strings.

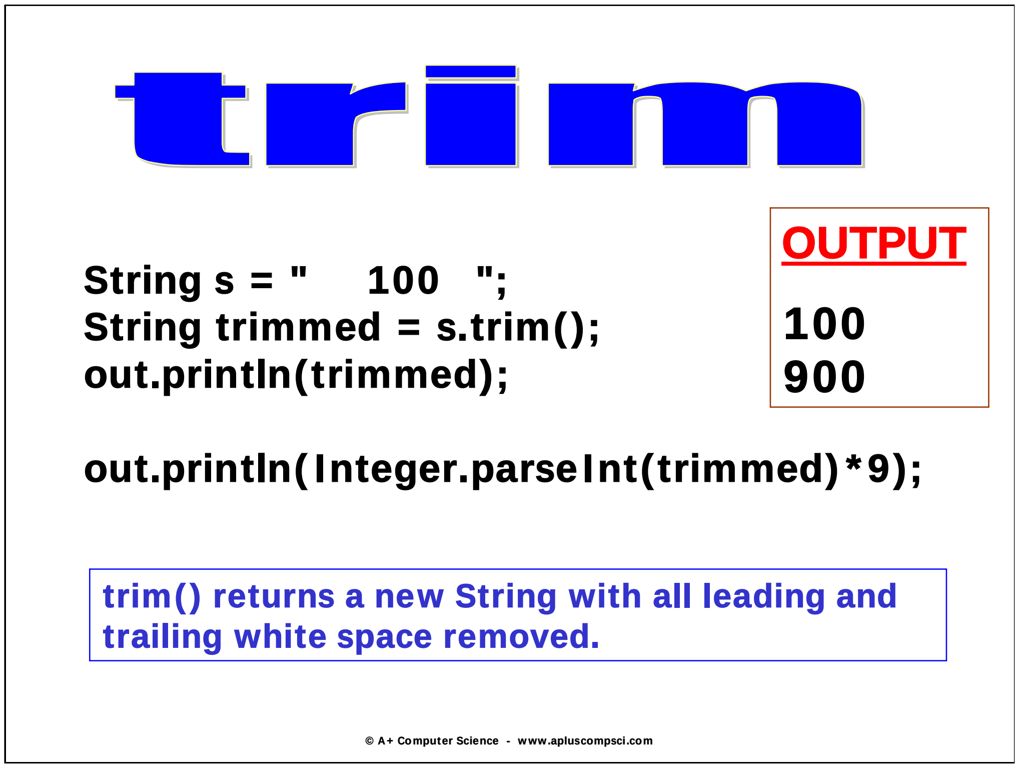
The String `compareTo()` method compares the letters stored in two String Objects.

The difference in ASCII of the first two letters that do not match is returned.



Open equals.java

compareto.java



trim

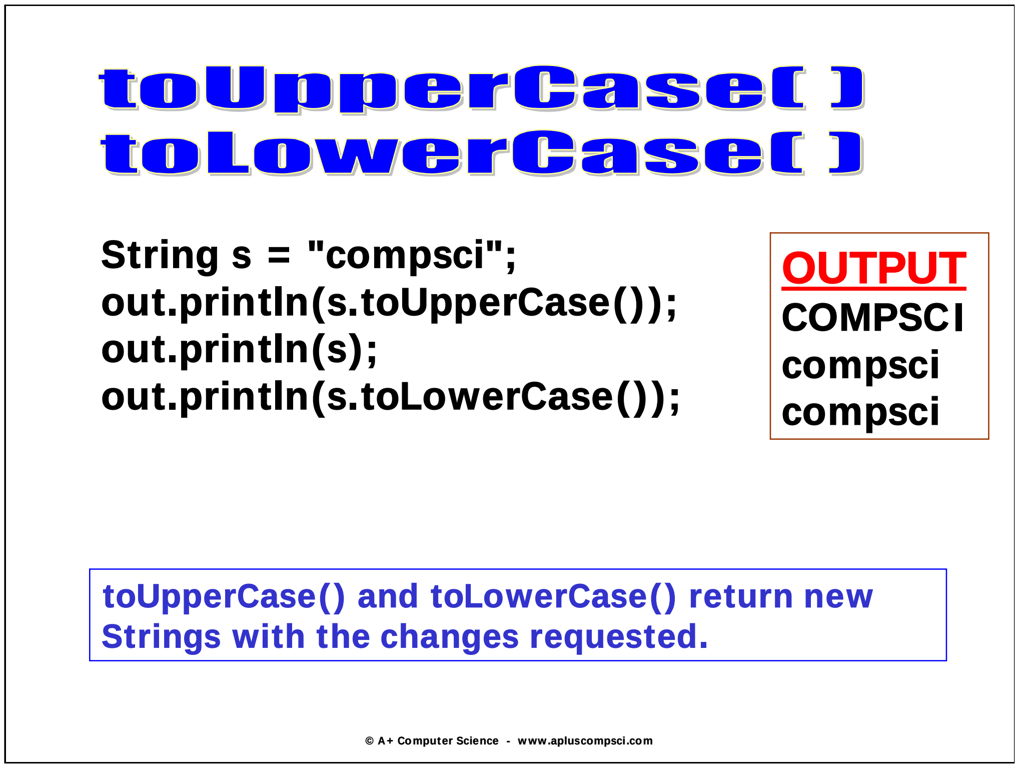
```
String s = "    100    ";
String trimmed = s.trim();
out.println(trimmed);

out.println(Integer.parseInt(trimmed)*9);
```

OUTPUT

```
100
900
```

trim() returns a new String with all leading and trailing white space removed.

The `trim()` method is useful to remove leading and trailing spaces.



toUpperCase( )
toLowerCase( )

```
String s = "compsci";
out.println(s.toUpperCase());
out.println(s);
out.println(s.toLowerCase());
```

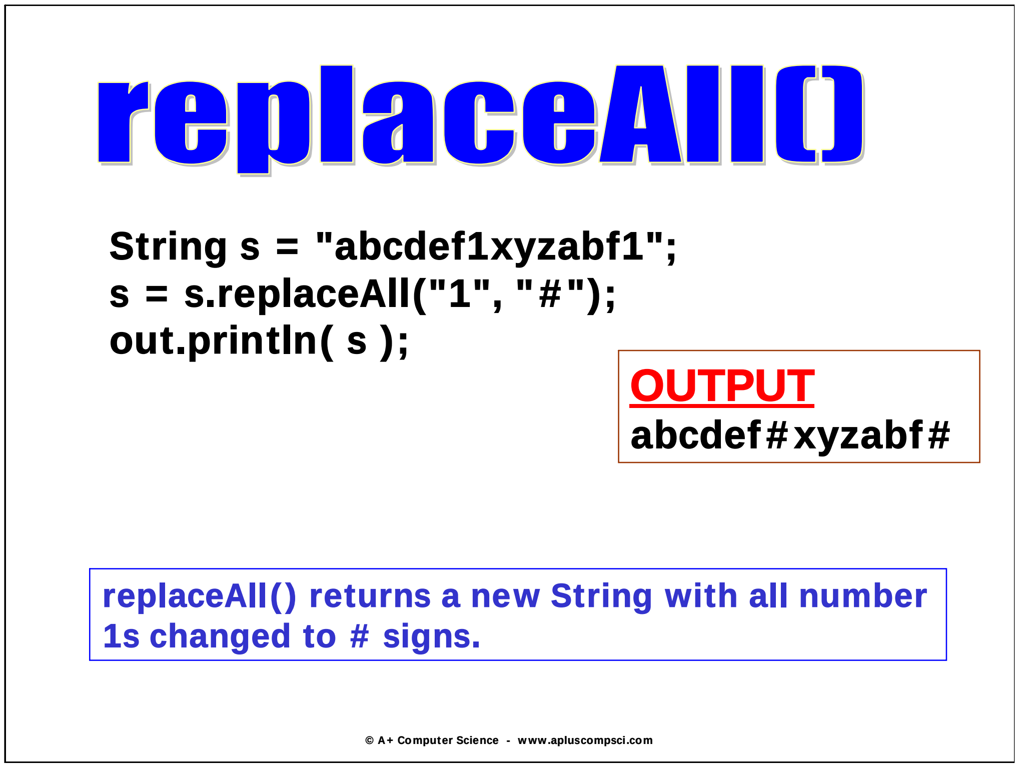
OUTPUT

```
COMPSCI
compsci
compsci
```

toUpperCase() and toLowerCase() return new Strings with the changes requested.

`toLowerCase()` and `toUpperCase()` both return new String Objects. The new String Object contains the same letters as the original String in all uppercase or all lowercase.

`toLowerCase()` and `toUpperCase()` do not change the original String Object. Both return a new String with the changes requested.



replaceAll()

```
String s = "abcdef1xyzabf1";
s = s.replaceAll("1", "#");
out.println( s );
```

OUTPUT

```
abcdef#xyzabf#
```

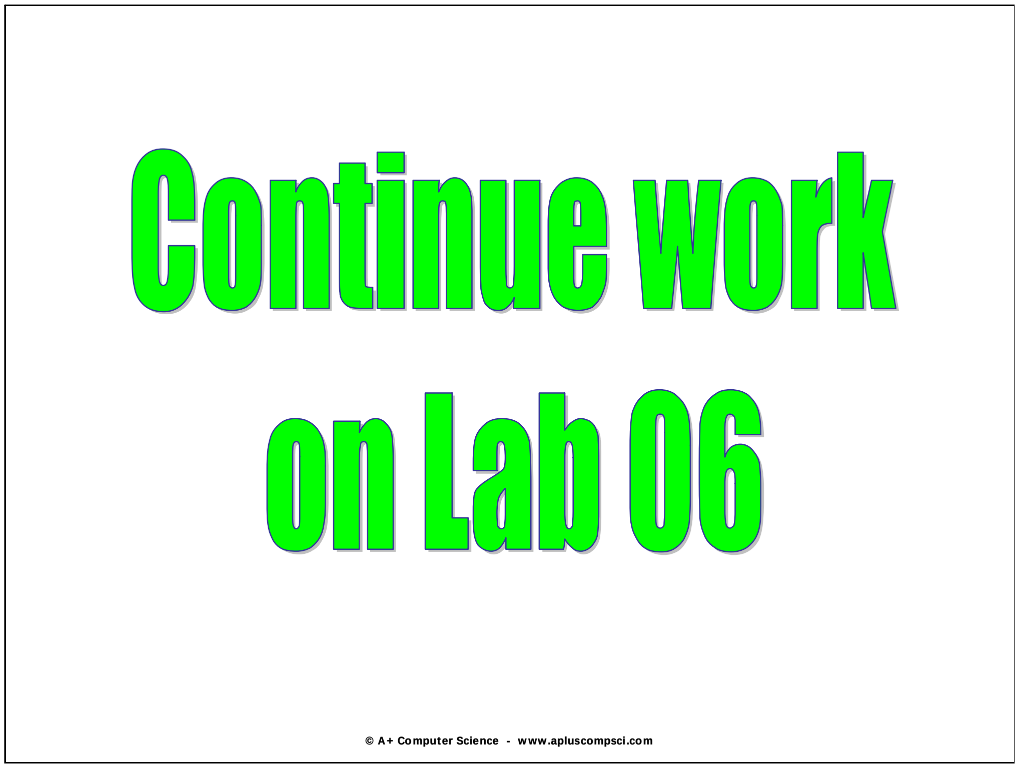
replaceAll() returns a new String with all number 1s changed to # signs.

`replaceAll()` returns a new String with the changes requested.
`replaceAll()` does not change the original String.

`replaceAll()` returns a new String with the specified letters replaced with the provided letters.

Open
replaceall.java
touppercase.java

Open
trim.java
stringtonums.java



Continue work on Lab 06