Recursion: How It Works



- How should you think about recursion so you can use it to develop elegant recursive methods to solve certain problems?
- Answer: Pretend there is a FreeLunch class with a method that has the same contract as the code you're trying to write (but it works only for smaller problems)

- Why do those recursive methods work?
- Answer: Following the "confidencebuilding" approach, you can argue as follows:
 - Does it work on all "smallest" cases?

- Why do those recursive methods work?
- Answer: Following the "confidencebuilding" approach, you can argue as follows:
 - Does it work on all "smallest" cases?
 - Does it work on all "next smallest" cases?

- Why do those recursive methods work?
- Answer: Following the "confidencebuilding" approach, you can argue as follows:
 - Does it work on all "smallest" cases?
 - Does it work on all "next smallest" cases?
 - Does it work on all "next smallest" cases?

- Why do those recursive methods work?
- Answer: Following the "confidencebuilding" approach, you can argue as follows:
 - Does it work on all "smallest" cases?
 - Does it work on all "next smallest" cases?
 - Does it work on all "next smallest" cases?
 - -... (Formally, proof by mathematical induction)

Question Considered Now

- *How* do those recursive methods work?
 - As promised, we have come back to this, but we continue to advise...
 - If you insist on *thinking about recursion* this way (rather than simply sating your curiosity about how it works), you may never be fully capable of developing elegant recursive solutions to problems!

Example

```
private static String reversedString(String s) {
    if (s.length() == 0) {
        return s;
    } else {
        String sub = s.substring(1);
        String rSub = reversedString(sub);
        return rSub + s.charAt(0);
    }
```

Trace reversedString("OSU")

	s = "OSU"
<pre>if (s.length() == 0) { } else {</pre>	
	s = "OSU"
<pre>String sub = s.substring(1);</pre>	
	s = "OSU" sub = "SU"
<pre>String rSub = reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	

Trace reversedString("OSU")

	s = "OSU"	
<pre>if (s.length() == 0) { } else {</pre>		
	s = "OSU"	
<pre>String sub = s.substring(1);</pre>		
	s = "OSU" sub = "SU"	
<pre>String rSub = reversedString(sub);</pre>		
Question: This is a recursive call, so how does it work?		

Trace reversedString("OSU")



How Every Call Works

- First, the tracing table for the code making the call is *suspended* and that tracing table is *pushed onto the runtime stack*
 - The runtime stack, often called simply "the stack", is effectively just a stack of tracing tables (think Stack<TracingTable>), each partially filled in with the results of the code in that tracing table as executed so far

How Every Call Works

- A new tracing table is created, containing the code for the method body being called
- The argument values are copied from the suspended tracing table into the formal parameters to start the new tracing table
- Execution in the new tracing table continues until it calls a method...

The currently executing tracing table gets to here ... String ("OSU")

	s = "OSU"
if (s.length() == 0) {	
} else {	
<pre>String sub = s.substring(1);</pre>	
String rSub =	
reversedString(sub);	
<pre>return rSub + s.charAt(0);</pre>	

... and the (top of the) stack of suspended tables is here. String ("OSU")

	s = "OSU"
<pre>if (s.length() == 0) { } else {</pre>	
<pre>String sub = s.substring(1);</pre>	
<pre>String rSub = reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	

This call suspends the current tracing table ...

String("OSU")

	s = "OSU"
if (s.length() == 0) {	
} else {	
<pre>String sub = s.substring(1);</pre>	
String rSub =	
<pre>reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	





... and the tracing table for length begins. String ("OSU")

		this = "OSU"
	/*	
	* Body for length method	
	* from class String; we	
	* do not have this, so how	
	* do we know what it does?	
	* We look at its contract!	
	* When it finishes, we know	
	* it has not changed this	
	* (it could not even if it	
60 ⁹	* wanted to), and it returns	
	* the length of this .	
	*/	
		this = "OSU"
		length = 3

if (a.leogth() - 0) [...

String sub = <u>s.substring(1)</u>;

return rSub + s.charAt(0)/

else

String rSub =
 reversedString(sub);

a -

How Every Return Works

- When the currently executing tracing table reaches a return statement, or for a void method falls off the end of the body, the results of the call are reflected in the tracing table on the *top of the stack*
- That tracing table is *popped off the stack* and it becomes the currently executing tracing table, resuming execution from the point where it was suspended

When this call returns ... String("OSU")



else

String cSub =

reversedString(sub);

... its results are reflected in the calling table ... String ("OSU")



... and that table is **popped** to resume execution. String ("OSU")

	s = "OSU"
<pre>if (s.length() == 0) { } else {</pre>	
<pre>String sub = s.substring(1);</pre>	
<pre>String rSub = reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	





... which is a *recursive* call! But it is nothing special. String ("OSU")

		s = "OSU"
if (s	h() == 0) {	
} else		
		s = "OSU"
String .	= s.substring(1);	
		s = "OSU"
		sub = "SU"
String rS	ub =	
reverse	dString(sub);	
return rS	ub + s.charAt(0);	



... and a new table for the body of the called method ... String ("OSU")

		s = "SU"
	<pre>if (s.length() == 0) { } else {</pre>	
	<pre>String sub = s.substring(1);</pre>	
•	<pre>String rSub = reversedString(sub);</pre>	
* 7*		
	<pre>return rSub + s.charAt(0);</pre>	

	s = "osu"
<pre>if (s.length() == 0) () else (</pre>	
	s = "080"
String sub = <u>s.substring(1);</u>	
	s = "050" sub = "30"
String <u>rSub</u> - reversedString (aub) ;	
<pre>return rSub + s.charAt(0);</pre>	

... begins *with its own* variables and values.

	s = "SU"
<pre>if (s.length() == 0) { } else {</pre>	
<pre>String sub = s.substring(1);</pre>	
<pre>String rSub = reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	

String("OSU")

	s = "csu"
<pre>if (s.lacgth() == 0) (} else {</pre>	
	s = "050"
String sub = <u>s.substring(1)</u> ;	
	s = "OSJ" sub = "SU"
String <u>rSub</u> - reverseditring (sub) ;	
<pre>seturn rSub + s.charAt(0);</pre>	



The current tracing table is suspended and pushed ... String ("OSU")

V	V	
	$\rho = "SD"$	
<pre>if (c.longth() == 0) {) else (</pre>		
	$s = {}^{\mu}SU^{\mu}$	
Similar sub = $s_{i,i}$ substrains(1);		
	s = "sv" sub = "v"	
String rSub = reversedString(sub);		
<pre>return rSub + s.sharAt(0);</pre>		
	s = "cso"	
<pre>if (s.length() == 0) () else (</pre>		
	s = "053"	

erse i	
	s = "oso"
String sub = <u>s.substring(1)</u> ;	
	s = "OSJ" sub = "SU"
<pre>String <u>rSub</u> = reversedString(sub);</pre>	
<pre>seturn gSub + s.charAt(0);</pre>	

... and a new table for the body of the called method ... String ("OSU")

		s = "U"
	<pre>if (s.length() == 0) { } else {</pre>	
<pre>s = "SD" if (s.length() == 0) { } =len (</pre>	<pre>String sub = s.substring(1);</pre>	
$s = \frac{v_{SU}}{s}$ String sub = scatheting(1); $s = \frac{v_{SU}}{s}$ String slob =		
return 1986 + sistarāt(0);	<pre>String rSub = reversedString(sub);</pre>	
s = "OSU" if (s.length() == 0) (i clos (s = "OSU" String sub = s.substring(1);		
s = "000" sub = "35" severaed(tring (aub);		
return räuh - sisharåt(0);	<pre>return rSub + s.charAt(0);</pre>	

... begins with its own variables and values.

		s = "U"
	<pre>if (s.length() == 0) { } else {</pre>	
φ = "SD" if (c.length i) == 0({ }	<pre>String sub = s.substring(1);</pre>	
$s = {}^{\mu}SD^{\mu}$ String sub = graphicking (1); $s = {}^{\mu}SD^{\mu}$ sub = {}^{\mu}D^{\mu}		
String IEEE - KeraraedString (ach) ;	String rSub =	
return nömh + sistharäh(0): s = "OSU"	<pre>reversedString(sub);</pre>	
<pre>if (s.length() -= 0) (else (</pre>		
sub = "CEO" sub = "SUb = severand(tring(ash);		
return gRug - gigharāt(0);	<pre>return rSub + s.charAt(0);</pre>	

String("OSU")



The current tracing table is String("OSU") suspended and pushed ...

	s =	*8*
<pre>if (s.length() 0) { } else {</pre>		
	8 =	•U*
String sub = <u>alabatcing(1)</u> ;		
	9 = auto	- 50 - 50
Realize of the	000	
string rain = sexexaedStaine(min);		
return rSub + sucharAt(0);		
		Recei
	2 -	-30-
if (<u>s.length</u> () == 0) {) else (
	s =	*SC*
String and $-$ standarding (1):		
	2 = 	"SU"
Real and a finds a	800	- 0
string fain - (duc) ;		
<pre>return rSub + s.charAt(0);</pre>		
	s =	"09U"
<pre>if (s.lecgth() -= 0) () else (</pre>		
	5 =	"053"
String sub = <u>s.substring(1)</u> ;		
	s = sub	"053" - "30"

) else {	
	s = "080"
String sub = <u>s.substring(1);</u>	
	$s = POSO^{\pi}$ sub = "S0"
String <u>rSub</u> - reversedString(sub);	
<pre>return rSub + s.charAt(0);</pre>	

... and a new table for the body of the called method ... String ("OSU")

<pre>s = *g* if (s_length() -= 0) (i plan (</pre>		S = ""
$s = vy^{*}$ String sub = <u>a_aubal_ting</u> (1); $\mu = vy^{*}$	if (s.length() == 0) {	
String rSub		
return rSub + sectorAt(0);	return s;	

	a = "SC"
<pre>if (s.length() == 0) {) else (</pre>	
	$s = 2SC^{n}$
String $adb = a_{s}a_{s}b_{s}b_{s}(a)c_{s}$	
	s = "so" sob = "o"
String rBub = reversedString(sub);	
return <u>rSub</u> + <u>s.charAt</u> (0);	

	s = "oso"
<pre>if (s.length() == 0) () else (</pre>	
	s = "050"
String sub = <u>s.substring(1)</u> ;	
	s = "050" sub = "30"
String <u>zSub</u> - reverseditring (aub) ;	
<pre>return rSub + s.charAt(0);</pre>	

... begins *with its own* variables and values.

S+ring("OSU")

<pre>s = *5* if (s.length() 0) (}</pre>		S = ""
$String sub = s_s substring (1);$ $S = \frac{\pi \sigma^2}{2}$	if (s.length() == 0) {	
String gRub		
return röch + sisterätil0);	return s;	

	a = "SC"
<pre>if (c.length() == 0) {) else (</pre>	
	$s = 2SC^{n}$
String $adh = a_{stability}(1)$;	
	s = "S0" sub = "0"
String <u>rBub</u> = reversedString(sub);	
<pre>return gSub + g.charAt(0);</pre>	

	s = "oso"
<pre>if (s.langth() == 0) () else (</pre>	
	s = "080"
String sub = <u>s.substring(1)</u> ;	
	s = "OSJ" sub = "SU"
String <u>zSub</u> - reverseditring (aub) ;	
<pre>return rSub + sucharAt(0);</pre>	

Soon, this retu	tracing table urns	g("OSU")
<pre>s = *s* if (a.kenth() 0) [</pre>		S = ""
$s = s_{x}$ String sub = s_x bistring (1); $a = s_{x}$	if (, ngth() == 0) {	
sub = ** Stoing rEak = cereisedSiciog(muh);		S = ""
return rSub + g.scherAt(0);	return s;	
<pre>p = "SU" if (s.length() == 0) () else (</pre>		
<pre>String ada = stathatting(1); s = "SC" sub = "SC" sub = "SC"</pre>		
Storing (Sink = reversedString(sub);		
return g5ub + g.ghagAt10);		
<pre>s = "050" if (s.lensth() == 0) (l else (</pre>		
String sub = s.substring(1); s = "050" sub = "55"		
String <u>23ub</u> asverasditiing (aub);		

... its results are reflected in the calling location ... String ("OSU")

	s = -vgs
<pre>if (s.length() -= 0) { } else {</pre>	
	$S = {}^{\mu}U^{\mu}$
String sub = a.aubstring(1);	
	a = "S" sub = ""
String <u>rSub</u> = reversedString(aub);	
<pre>return rSub + s.charAt(0);</pre>	
	p = "SU"
<pre>if (c.length() == 0) { } else {</pre>	
	$s = {}^{p}sc^{p}$
Similar and $-$ substanting (1):	
	s = "sv" sub = "v"
String rSub = reversedString(sub);	
<pre>return gSub + g.sharAt(0);</pre>	
	8 = "050"
<pre>if (s.lecgth() == 0) () else {</pre>	
	$\rho = "080"$
String sub = <u>s.substring(1);</u>	
	s = "050" sub = "30"
String <u>rSub</u> - reversedString (sub);	

return rSub + s.char5t(0);

... and that table is **popped** to resume execution.

String("OSU")

		S = "U"
	<pre>if (s.length() == 0) { } else {</pre>	
		S = "U"
<pre>s = "SD" f (c.length() == 0) { else /</pre>	<pre>String sub = s.substring(1);</pre>	
$s = \frac{v_{SU}v}{sub} = \frac{s_{SU}v_{SUb}}{sub} = \frac{s_{SU}v_{SUb}}{sub} = \frac{v_{SU}v_{SUb}}{sub} = v_{S$		s = "U" sub = ""
return röch + sisteräti0); s = "050"	<pre>String rSub = reversedString(sub);</pre>	
<pre>f (a.legith() == 0) (else (</pre>		
return räus - s.charAt10);	<pre>return rSub + s.charAt(0);</pre>	

if (c.length() -- 0) else

if (s.length() -- 0)

| else |

String gSub



... its results are reflected in the calling location ... String ("OSU")

V		
	<u>a</u> =	"80"
<pre>if (s.length() == 0) {) else (</pre>		
	8 =	"SC"
Similar sub = $s_{i,i}$ substrains(1);		
	s = sub	"SU" - "O"
String rSub = reversedString(sub);		
return gSub + g.scharat(0);		

	s = "090"
<pre>if (s.langth() == 0) () else (</pre>	
	s = "093"
String sub = <u>s.substring(1)</u> ;	
	s = "OSJ" sub = "SU"
String <u>zSub</u> - reverseditring (aub) ;	
<pre>return rSub + s.charAt(0);</pre>	

... and that table is popped to resume execution. String ("OSU")

		s = "SU"
	<pre>if (s.length() == 0) { } else {</pre>	
		s = "SU"
	<pre>String sub = s.substring(1);</pre>	
		s = "SU" sub = "U"
•	String rSub = reversedString(sub);	
*		
	<pre>return rSub + s.charAt(0);</pre>	

	s = "oso"
<pre>if (s.lacgth() == 0) () else (</pre>	
	s = "080"
String sub = <u>s.substring(1)</u> ;	
	s = "050" sub = "30"
String <u>rSub</u> - reversedString (sub) /	
<pre>return rSub + s.charAt(0);</pre>	





... and that table is popped to resume execution. String ("OSU")

	s = "OSU"
<pre>if (s.length() == 0) {</pre>	
} erse {	
	s = "OSU"
<pre>String sub = s.substring(1);</pre>	
	s = "OSU"
	sub = "SU"
String rSub =	
<pre>reversedString(sub);</pre>	
<pre>return rSub + s.charAt(0);</pre>	

Soon, this tracing ta returns	able String	("OSU")
		s = "OSU"
if } e .	gth() == 0) {	
		s = "OSU"
Str	<pre>sub = s.substring(1);</pre>	
		s = "OSU" sub = "SU"
Stri	rSub =	
rev	sedString(sub);	
		s = "OSU" sub = "SU" rSub = "US"
return	<pre>rSub + s.charAt(0);</pre>	

Finally!

- The value returned to the original calling program is the string "USO"
 - Phew!
 - And it is even correct: the result of reversing the string "OSU" is the string "USO"

Conclusion

- Each call to a method—whether recursive or not—effectively results in the creation of a new tracing table containing the body of the called method
- Each tracing table has its own variables:
 - Its own formal parameters
 - Its own local variables

Conclusion

 If you really think you can reason about recursive code by mentally executing this kind of a series of events to check your thinking, then ... you're deluding yourself

Conclusion

 If you really think you can reason about recursive code by mentally executing this kind of a series of events to check your thinking, then ... you're deluding yourself

And if you don't believe it yet, try mentally executing this way for code that makes multiple recursive calls from each tracing table.