

LECTURE 9: RECURSION

From an online
ad for tee shirts

[www.redbubble.com/people/
lrfsa/works/35162846-to-
iterate-is-human-to-recurse-
divine?p=t-shirt](http://www.redbubble.com/people/lrfsa/works/35162846-to-iterate-is-human-to-recurse-divine?p=t-shirt)

Lecture 9

CS2110 – Fall 2021

LECTURE 9: RECURSION

PART 1: INTRODUCTION

Recursion

3



To iterate is human, to recurse divine.

Peter L. Deutsch

Look up Peter on Wikipedia.

Musician and computer scientist

Operating systems, SmallTalk implementation, more

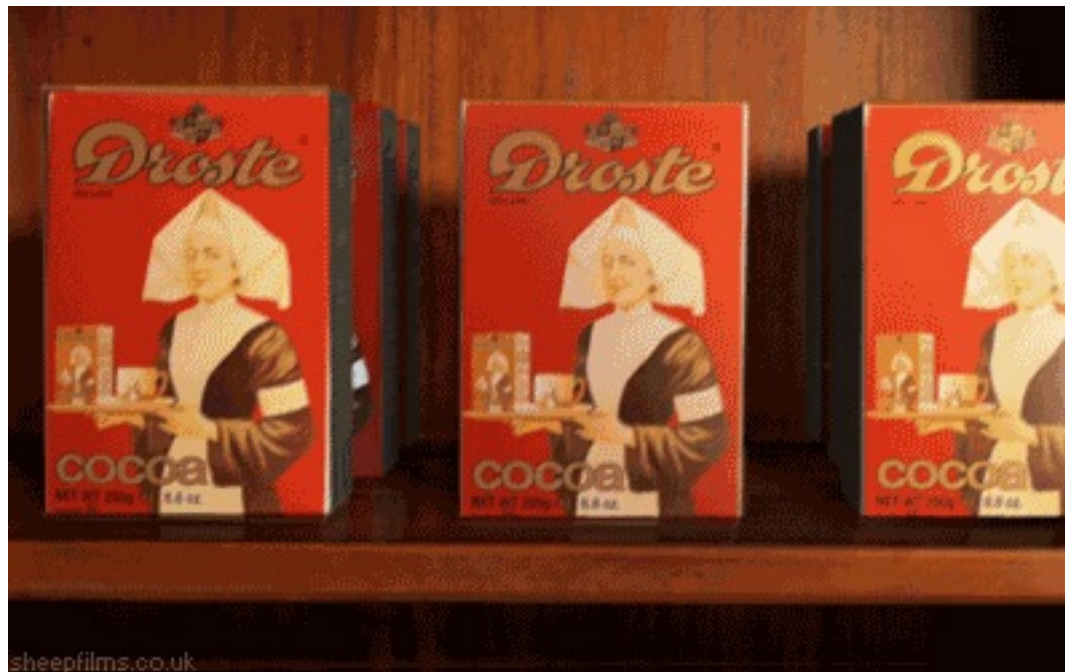
Recursion

4

To iterate is human, to recurse divine.

Peter L. Deutsch

Thou hast damnable iteration and art, indeed, able to corrupt a saint.
In Shakespeare's Henry IV, Pt I



To Understand Recursion...

5

recursion



All

Images

Videos

Books

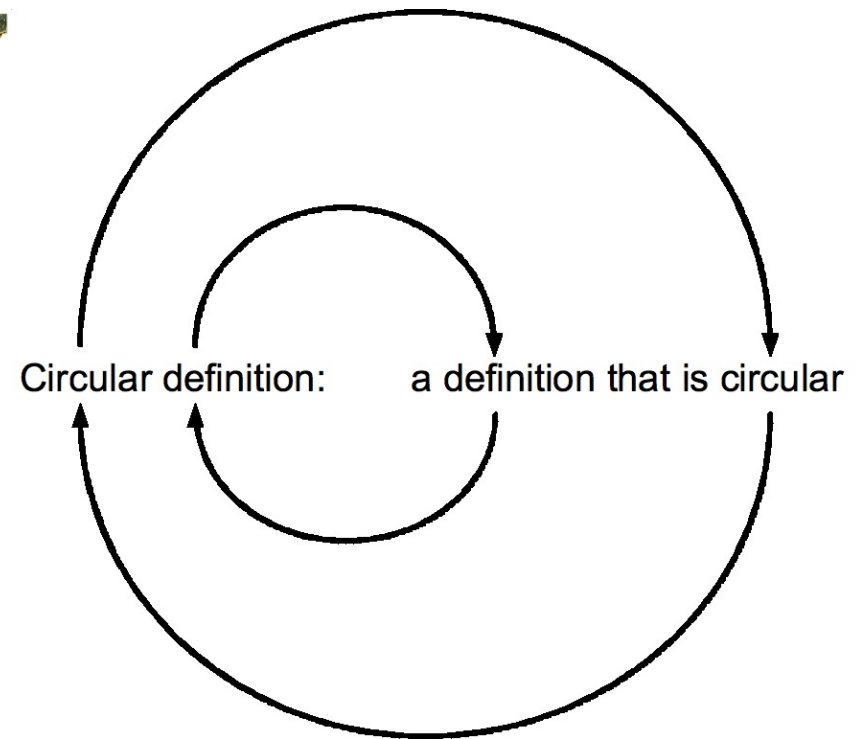
More

Settings

Tools

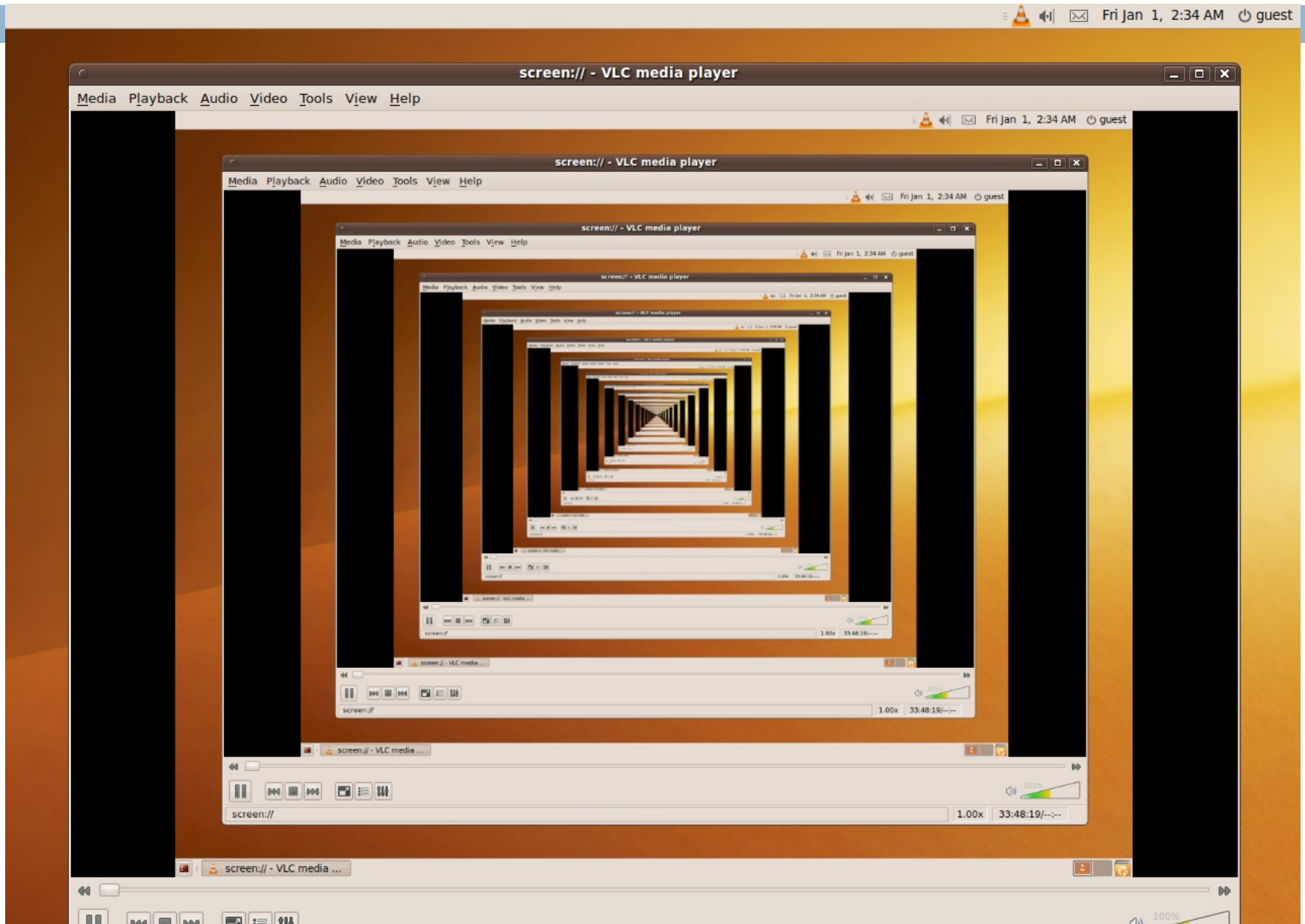
About 10,400,000 results (0.60 seconds)

Did you mean: **recursion**



Recursion

6



Recursion – Real Life Examples

7

<noun phrase> is <noun>, or

<adjective> <noun phrase>, or

<adverb> <noun phrase>

Example:

terrible horrible no-good very bad day

Recursion – Real Life Examples

8

<noun phrase> **is** <noun>, or

<adjective> <noun phrase>, or

<adverb> <noun phrase>

ancestor(p) **is** parent(p), or

parent(ancestor(p))

great great great great great great great great great great
great great grandmother.

$0! = 1$

$n! = n * (n-1)!$

1, 1, 2, 6, 24, 120, 720, 5050, 40320, 362880, 3628800, 39916800,
479001600...

Sum the digits in a non-negative integer

9

/** = sum of digits in n.

* Precondition: $n \geq 0$ */

public static int sum(int n) {

if ($n < 10$) **return** n;

sum calls itself!

// { n has at least two digits }

// return first digit of n + sum of digits in rest of n

return $n \% 10$ + **sum**($n / 10$);

}

$\text{sum}(7) = 7$

$\text{sum}(8703) = 3 + \text{sum}(870)$

$= 3 + 0 + \text{sum}(87)$

$= 3 + 0 + 7 + \text{sum}(8)$

Two different questions, two different answers

10

1. How is it **executed**?

(or, why does recursion even work?)

2. How do we **understand** recursive methods?

(or, how do we **write/develop** recursive methods?)

The end

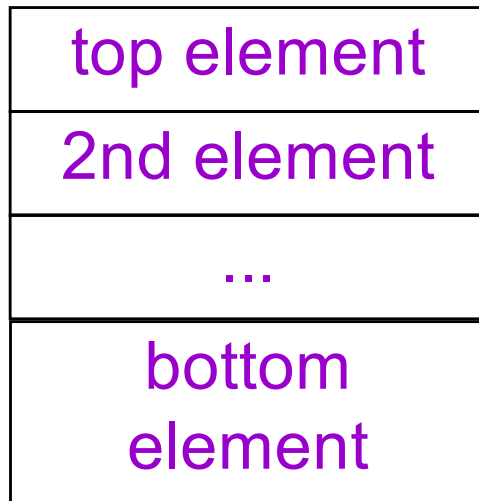
LECTURE 9: RECURSION

PART 2: THE CALL STACK

Stacks and Queues

12

↑
stack grows



first	second	...	last
-------	--------	-----	------

Americans wait in a line. The Brits wait in a queue !

Stack: list with (at least) two basic ops:

- * Push an element onto its top
- * Pop (remove) top element

Last-In-First-Out (LIFO)

Like a stack of trays in a cafeteria

Queue: list with (at least) two basic ops:

- * Append an element
- * Remove first element

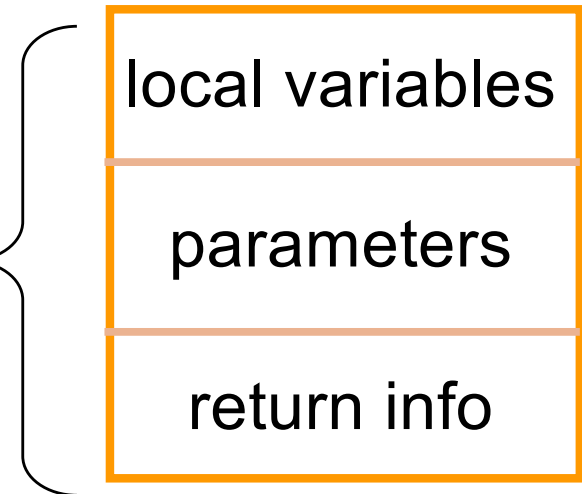
First-In-First-Out (FIFO)

Stack Frame

13

A “frame” contains information about a method call:

At runtime Java maintains a **a frame** **stack** that contains frames for all method calls that are being executed but have not completed.



Method call: push a frame for call on **stack**. Assign argument values to parameters. Execute method body. Use the frame for the call to reference local variables and parameters.

End of method call: pop its frame from the **stack**; if it is a function leave the return value on top of **stack**.

Memorize method call execution!

14

A frame for a call contains parameters, local variables, and other information needed to properly execute a method call.

To execute a method call:

1. push a frame for the call on the stack,
2. assign argument values to parameters,
3. execute method body,
4. pop frame for call from stack, and (for a function) push returned value on stack

When executing method body look in frame for call for parameters and local variables.

Frames for methods sum main method in the system

16

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

Frame for method in the system
that calls method main

frame:

n ____
return info

frame:

r ____ args ____
return info

frame:

?
return info

Example: Sum the digits in a non-negative integer

17

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}  
  
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

Frame for method in the system
that calls method main: main is
then called

main

r ____ args null
return info

system

?
return info

Example: Sum the digits in a non-negative integer

18

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

Method main calls sum:

main

system

n 824
return info

r ____ args null
return info

?
return info

Example: Sum the digits in a non-negative integer

19

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

$n \geq 10$ sum calls sum:

main

system

n <u>82</u> return info
n <u>824</u> return info
r ____ args <u>null</u> return info
? return info

Example: Sum the digits in a non-negative integer

20

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

$n \geq 10$. sum calls sum:

main

system

n 8
return info

n 82
return info

n 824
return info

r ____ args null
return info

?
return info

Example: Sum the digits in a non-negative integer

21

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

$n < 10$ sum stops: frame is popped
and n is put on stack:

main

system

n <u>8</u> return info
n <u>82</u> return info
n <u>824</u> return info
r ____ args <u>null</u> return info
? return info

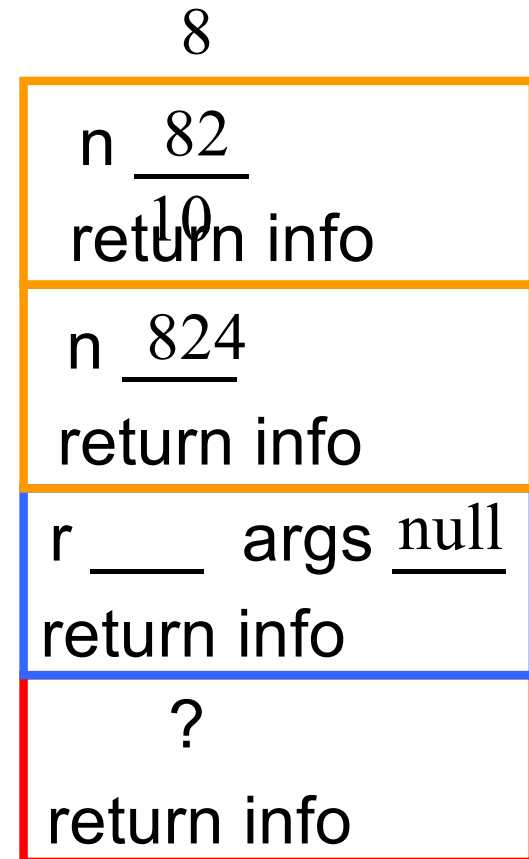
Example: Sum the digits in a non-negative integer

22

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

main



Using return value 8 stack computes
 $2 + 8 = 10$ pops frame from stack puts
return value 10 on stack

Example: Sum the digits in a non-negative integer

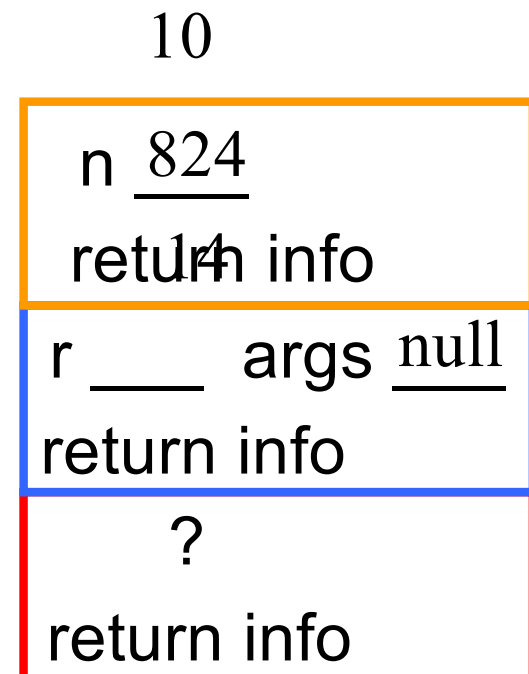
23

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

main

Using return value 10 stack computes
 $4 + 10 = 14$ pops frame from stack
puts return value 14 on stack



Example: Sum the digits in a non-negative integer

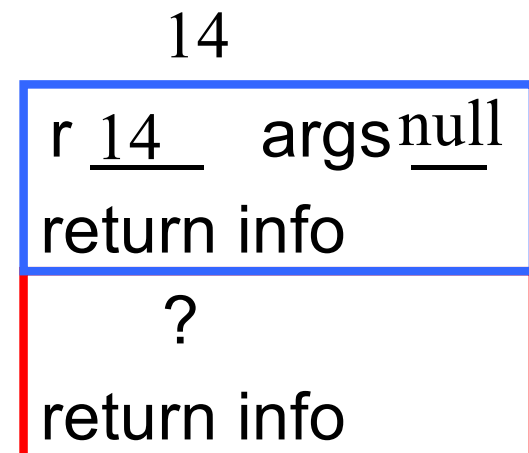
24

```
public static int sum(int n) {  
    if (n < 10) return n;  
    return n%10 + sum(n/10);  
}
```

```
public static void main(  
    String[] args) {  
    int r= sum(824);  
    System.out.println(r);  
}
```

Using return value 14 main stores
14 in r and removes 14 from stack

main



2 different questions, 2 different answers

25

1. How is it **executed**?

(or, why does this even work?)

It's **not** magic! Trace the code's execution using the method call algorithm, drawing the stack frames as you go.

Use only to gain understanding / assurance that recursion works.

2. How do we **understand** recursive methods?

(or, how do we **write/develop** recursive methods?)

This requires a totally different approach.

QUIZ

End

LECTURE 9: RECURSION

PART 3: UNDERSTANDING RECURSIVE METHODS

Math def to Java function

27

Factorial function:

$$0! = 1$$

$$n! = n * (n-1)! \text{ for } n > 0$$

$$\text{(e.g.: } 4! = 4*3*2*1=24\text{)}$$

Easy to make math definition into a Java function!

```
public static int fact(int n) {  
    if (n == 0) return 1;  
  
    return n * fact(n-1);  
}
```

Exponentiation:

$$b^0 = 1$$

$$b^c = b * b^{c-1} \text{ for } c > 0$$

```
public static int exp(int b, int c) {  
    if (c == 0) return 1;  
  
    return b * exp(b, c-1);  
}
```

How to understand what a call does

28

Make a copy of the method spec,
replacing the parameters of the
method by the arguments

spec says that the
value of a call
equals the sum of
the digits of n

sumDigs(654)

sum of digits of **n**

sum of digits of **654**

```
/** = sum of the digits of n.  
 * Precondition: n >= 0 */  
public static int sumDigs(int n) {  
    if (n < 10) return n;  
    // n has at least two digits  
    return n%10 + sumDigs(n/10);  
}
```

Understanding a recursive method

29

Step 1. Have a **precise spec!**

Step 2. Check that the method works in **the base case(s)**: That is, cases where the parameter is **small** enough that the result can be computed simply and without recursive calls.

If $n < 10$ then n consists of a single digit.

Looking at the spec, we see that that digit is the required sum.

```
/** = sum of the digits of n.  
 * Precondition:  $n \geq 0$  */  
public static int sumDigs(int n) {  
    if ( $n < 10$ ) return n;  
    // n has at least two digits  
    return  $n \% 10$  + sumDigs( $n / 10$ );  
}
```

Understanding a recursive method

30

Step 1. Have a precise spec!

Step 2. Check that the method works in **the base case(s)**.

Step 3. Look at the **recursive case(s)**. In your mind replace each recursive call by what it

does according to the method spec and verify that the correct result is then obtained.

```
/** = sum of the digits of n.
```

```
 * Precondition:  $n \geq 0$  */
```

```
public static int sumDigs(int n) {
```

```
    if (n < 10) return n;
```

```
    // n has at least two digits
```

```
    return n%10 + sumDigs(n/10);
```

```
}
```

```
return n%10 + sumDigs(n/10);
```

```
return n%10 + (sum of digits of n/10);    // e.g. n = 843
```


Understanding a recursive method

31

Step 1. Have a precise spec!

Step 2. Check that the method works in **the base case(s)**.

Step 3. Look at the **recursive case(s)**. In your mind replace each recursive call by what it does acc. to the spec and verify correctness.

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the pars of the method.

$n/10 < n$, so it will get smaller until it has one digit

```
/** = sum of the digits of n.  
 * Precondition: n >= 0 */  
public static int sumDigs(int n) {  
    if (n < 10) return n;  
    // n has at least two digits  
    return n%10 + sumDigs(n/10);  
}
```

Understanding a recursive method

32

Step 1. Have a precise spec!

Important! Can't do anything further without a precise spec.

Step 2. Check that the method works in **the base case(s)**.

Step 3. Look at the **recursive case(s)**. In your mind replace each recursive call by what it does according to the spec and verify correctness.

Once you get the hang of it this is what makes recursion easy! This way of thinking is based on math induction which we don't cover in this course.

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the parameters of the method

Writing a recursive method

33

Step 1. Write a precise spec!

Step 2. Write and verify code for **the base case(s)**.

Step 3. Write and verify code for the **recursive case(s)**. In doing so, attempt to write the answer in terms of the same problem (it looks like the specification) but on a smaller scale. That allows you to replace that “same problem on a smaller scale” by a recursive call.

Step 4. (No infinite recursion) Make sure that the args of recursive calls are in some sense smaller than the parameters of the method

QUIZZES

End

LECTURE 9: RECURSION

PART 4: WRITING RECURSIVE METHODS

Examples of writing recursive functions

35

In this part, we demo writing recursive functions using the approach outlined below. The java file we develop will be placed on the course webpage some time after the lecture.

Step 1. Have a precise **spec!**

Step 2. Write the **base case(s)**.

Step 3. Look at all other cases. See how to define these cases in terms **of smaller problems of the same kind**. Then implement those definitions using recursive calls for those **smaller problems of the same kind**.

Step 4. Make sure recursive calls are “smaller” (no infinite recursion).

Check palindrome-hood

36


A palindrome is a String that reads the same backward and forward:

`isPal("racecar") → true`

`isPal("pumpkin") → false`

A String with at least two characters is a palindrome if

- (0) its first and last characters are equal and
- (1) chars between first & last form a palindrome:

e.g. 
AMANAPLANACANALPANAMA
have to be a palindrome

A recursive definition!

□ A man a plan a caret a ban a myriad a sum a lac a liar a hoop a pint a catalpa a gas
an oil a bird a yell a vat a caw a pax a wag a tax a nay a ram a cap a yam a gay a tsar
a wall a car a luger a ward a bin a woman a vassal a wolf a tuna a nit a pall a fret a
watt a bay a daub a tan a cab a datum a gall a hat a fag a zap a say a jaw a lay a wet a
gallop a tug a trot a trap a tram a torr a caper a top a tonk a toll a ball a fair a sax a
minim a tenor a bass a passer a capital a rut an amen a ted a cabal a tang a sun an ass
a maw a sag a jam a dam a sub a salt an axon a sail an ad a wadi a radian a room a
rood a rip a tad a pariah a revel a reel a reed a pool a plug a pin a peek a parabola a
dog a pat a cud a nu a fan a pal a rum a nod an eta a lag an eel a batik a mug a mot a
nap a maxim a mood a leek a grub a gob a gel a drab a citadel a total a cedar a tap a
gag a rat a manor a bar a gal a cola a pap a yaw a tab a raj a gab a nag a pagan a bag
a jar a bat a way a papa a local a gar a baron a mat a rag a gap a tar a decal a tot a led
a tic a bard a leg a bog a burg a keel a doom a mix a map an atom a gum a kit a
baleen a gala a ten a don a mural a pan a faun a ducat a pagoda a lob a rap a keep a
nip a gulp a loop a deer a leer a lever a hair a pad a tapir a door a moor an aid a raid
a wad an alias an ox an atlas a bus a madam a jag a saw a mass an anus a gnat a lab a
cadet an em a natural a tip a caress a pass a baronet a minimax a sari a fall a ballot a
knot a pot a rep a carrot a mart a part a tort a gut a poll a gateway a law a jay a sap a
zag a fat a hall a gamut a dab a can a tabu a day a batt a waterfall a patina a nut a
flow a lass a van a mow a nib a draw a regular a call a war a stay a gam a yap a cam
a ray an ax a tag a wax a paw a cat a valley a drib a lion a saga a plat a catnip a pooh
a rail a calamus a dairyman a bater a canal Panama

Example: Is a string a palindrome?

38

```
/** = "s is a palindrome" */  
public static boolean isPal(String s) {  
    if (s.length() <= 1)  
        return true;  
  
    int n= s.length()-1;  
    // s = s[0] + s[1..n-1] + s[n]  
    return s.charAt(0) == s.charAt(n) && isPal(s.substring(1,n));  
}
```

