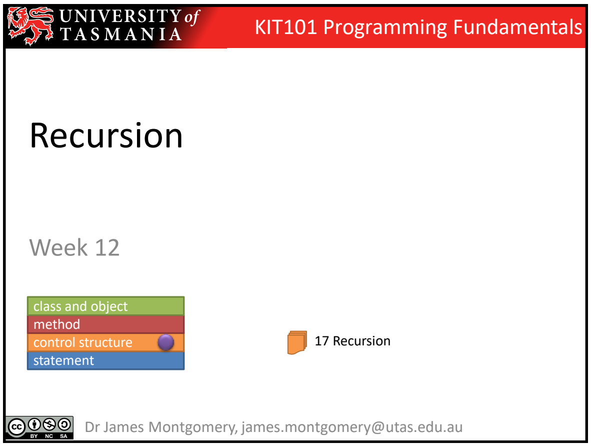


UNIVERSITY of TASMANIA

KIT101 Programming Fundamentals

Recursion

Week 12

- class and object
- method
- control structure ●
- statement

17 Recursion

Dr James Montgomery, james.montgomery@utas.edu.au

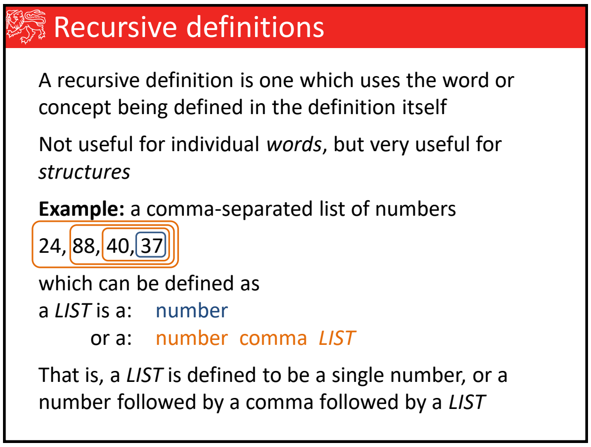
Матрёшка

Matryoshka

A **matryoshka doll** ("Russian doll") is

- a small solid wooden doll
- or a hollow wooden doll containing a **matryoska doll**

Russian Toys and Dissection of the toy by Fanghong is licensed under CC BY SA 3.0



Recursive definitions

A recursive definition is one which uses the word or concept being defined in the definition itself

Not useful for individual *words*, but very useful for *structures*

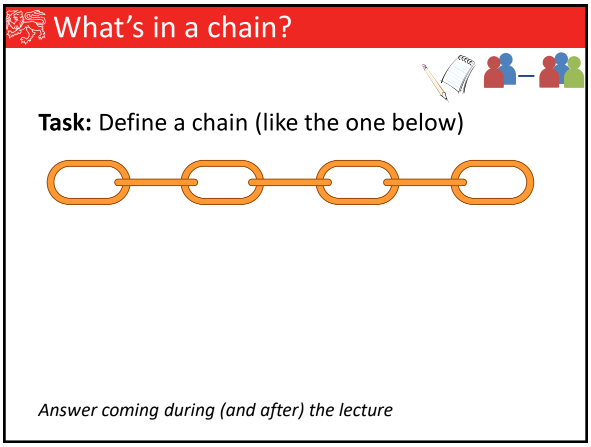
Example: a comma-separated list of numbers

24, 88, 40, 37

which can be defined as

a *LIST* is a: number
or a: number comma *LIST*

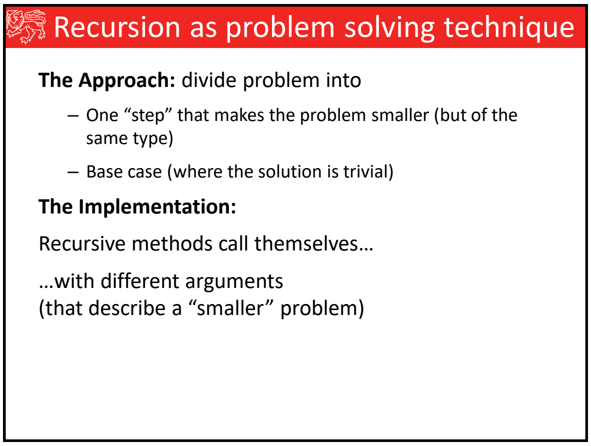
That is, a *LIST* is defined to be a single number, or a number followed by a comma followed by a *LIST*



What’s in a chain?

Task: Define a chain (like the one below)

Answer coming during (and after) the lecture



Recursion as problem solving technique

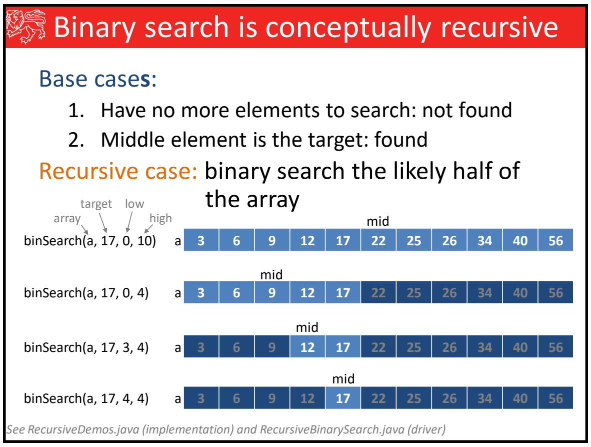
The Approach: divide problem into

- One “step” that makes the problem smaller (but of the same type)
- Base case (where the solution is trivial)

The Implementation:

Recursive methods call themselves…

…with different arguments
(that describe a “smaller” problem)



Binary search is conceptually recursive

Base cases:

1. Have no more elements to search: not found
2. Middle element is the target: found

Recursive case: binary search the likely half of the array

mid
binSearch(a, 17, 0, 10) a 3 6 9 12 17 22 25 26 34 40 56

mid
binSearch(a, 17, 0, 4) a 3 6 9 12 17 22 25 26 34 40 56

mid
binSearch(a, 17, 3, 4) a 3 6 9 12 17 22 25 26 34 40 56

mid
binSearch(a, 17, 4, 4) a 3 6 9 12 17 22 25 26 34 40 56

See RecursiveDemos.java (implementation) and RecursiveBinarySearch.java (driver)

Recursive solutions

General pattern for a recursive solution

1. test for stopping condition
2. If not at stopping condition, either
   - do one step towards solution
   - and call the method again to solve the rest

e.g. process a list

or

- call the method again to solve most of the problem
- and do the final step

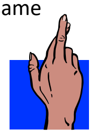
e.g. sum an array

Tip: the base case stops the recursion,
so care must be taken in defining the stopping condition

Devising recursive solutions

1. Find a case where the solution is trivial (the stopping condition)
2. Divide the problem up:

   One step & a smaller problem of exactly the same type (closer to the trivial solution)



3. Believe that the solution will work
4. Code and test the solution

Sum of the first *n* integers

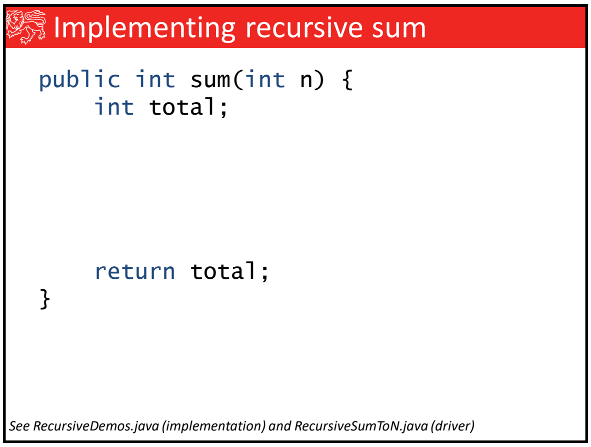
Task: Devise a recursive approach to calculate the sum of the positive integers up to *n*

i.e., sum(*n*) = 1 + 2 + … + *n*

Define the **base case** and **recursive case**

Tips:

- when is this problem simplest?
- when is the problem just one step more difficult than this simplest case?



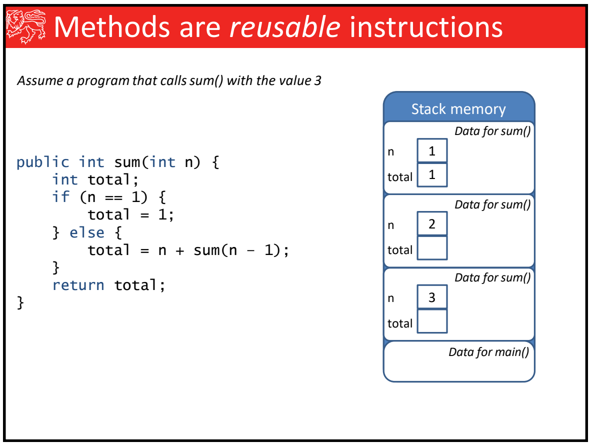
Implementing recursive sum

```
public int sum(int n) {
    int total;




    return total;
}
```

See RecursiveDemos.java (implementation) and RecursiveSumToN.java (driver)



Methods are *reusable* instructions

Assume a program that calls sum() with the value 3

```
public int sum(int n) {
    int total;
    if (n == 1) {
        total = 1;
    } else {
        total = n + sum(n - 1);
    }
    return total;
}
```

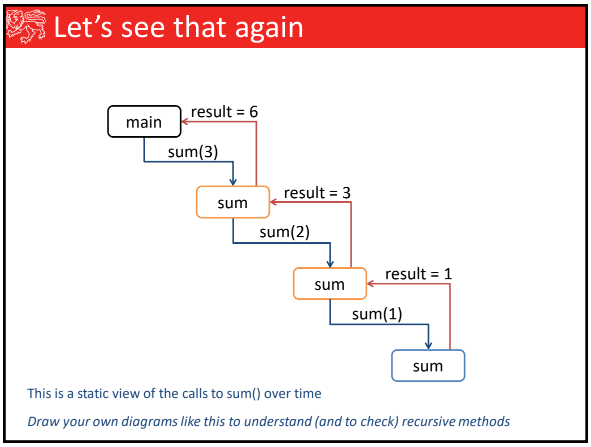
Stack memory

Data for sum()
n 1
total 1

Data for sum()
n 2
total

Data for sum()
n 3
total

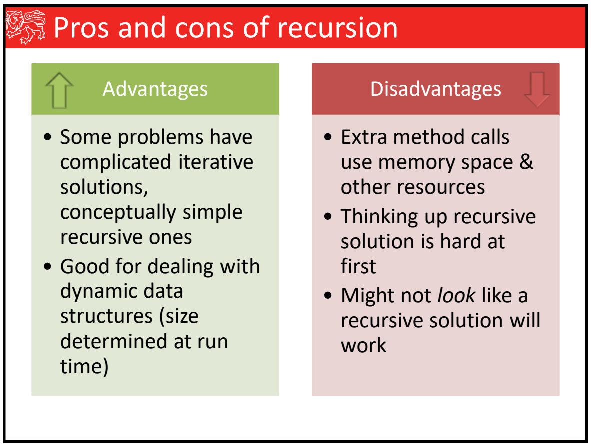
Data for main()



Let’s see that again

This is a static view of the calls to sum() over time

Draw your own diagrams like this to understand (and to check) recursive methods



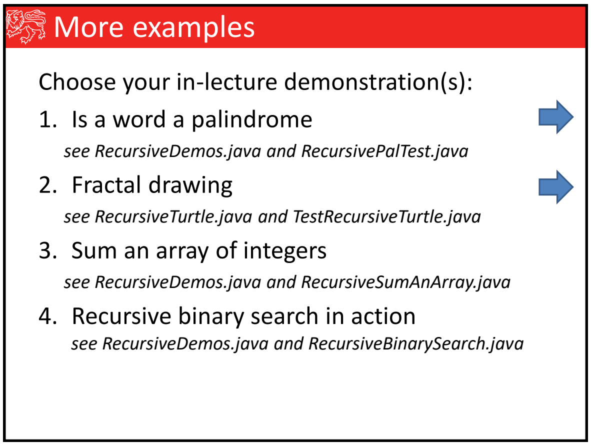
Pros and cons of recursion

Advantages

- Some problems have complicated iterative solutions, conceptually simple recursive ones
- Good for dealing with dynamic data structures (size determined at run time)

Disadvantages

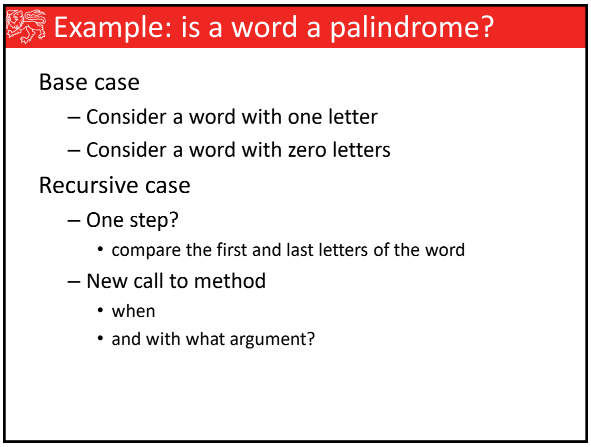
- Extra method calls use memory space & other resources
- Thinking up recursive solution is hard at first
- Might not *look* like a recursive solution will work



More examples

Choose your in-lecture demonstration(s):

1. Is a word a palindrome
   see RecursiveDemos.java and RecursivePalTest.java
2. Fractal drawing
   see RecursiveTurtle.java and TestRecursiveTurtle.java
3. Sum an array of integers
   see RecursiveDemos.java and RecursiveSumAnArray.java
4. Recursive binary search in action
   see RecursiveDemos.java and RecursiveBinarySearch.java



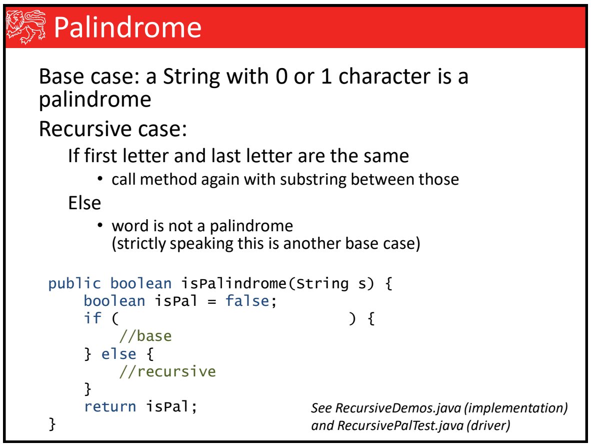
Example: is a word a palindrome?

Base case

- Consider a word with one letter
- Consider a word with zero letters

Recursive case

- One step?
  - compare the first and last letters of the word
- New call to method
  - when
  - and with what argument?



Palindrome

Base case: a String with 0 or 1 character is a palindrome

Recursive case:

If first letter and last letter are the same

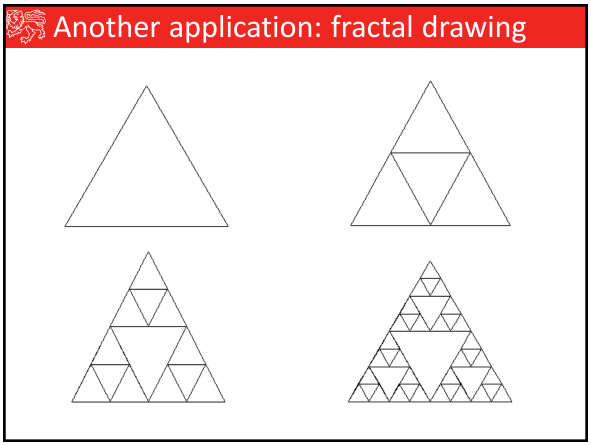
- call method again with substring between those

Else

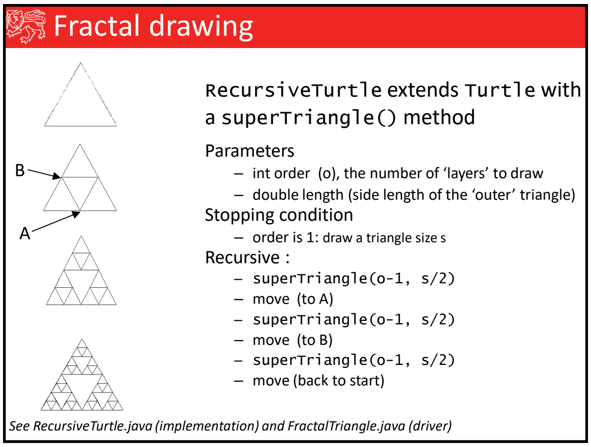
- word is not a palindrome
  (strictly speaking this is another base case)

```
public boolean isPalindrome(String s) {
    boolean isPal = false;
    if (                    ) {
        //base
    } else {
        //recursive
    }
    return isPal;
}
```

See RecursiveDemos.java (implementation) and RecursivePalTest.java (driver)



Another application: fractal drawing



Fractal drawing

RecursiveTurtle extends Turtle with a superTriangle() method

Parameters

- int order (o), the number of 'layers' to draw
- double length (side length of the 'outer' triangle)

Stopping condition

- order is 1: draw a triangle size s

Recursive :

- superTriangle(o-1, s/2)
- move (to A)
- superTriangle(o-1, s/2)
- move (to B)
- superTriangle(o-1, s/2)
- move (back to start)

See RecursiveTurtle.java (implementation) and FractalTriangle.java (driver)