11. Recursion

Harald Gall, Prof. Dr.
Michael Würsch

Institut für Informatik
Universität Zürich

http://seal.ifi.uzh.ch/info1

University of Zurich
Department of Informatics

s.e.a.l.
software evolution & architecture lab

Objectives

- become familiar with the idea of recursion
- learn to use recursion as a programming tool

Introduction to Recursion

- A recursive algorithm will have one subtask that is a small version of the entire algorithm's task
- A Java method definition is *recursive* if it contains an invocation of itself.
- The method *continues to call itself*, with ever simpler cases, until a base case is reached which can be resolved without any subsequent recursive calls.

Example: Exponent

```
private int power(int x, int y) {
   // y>=0 returns x^y

}
```

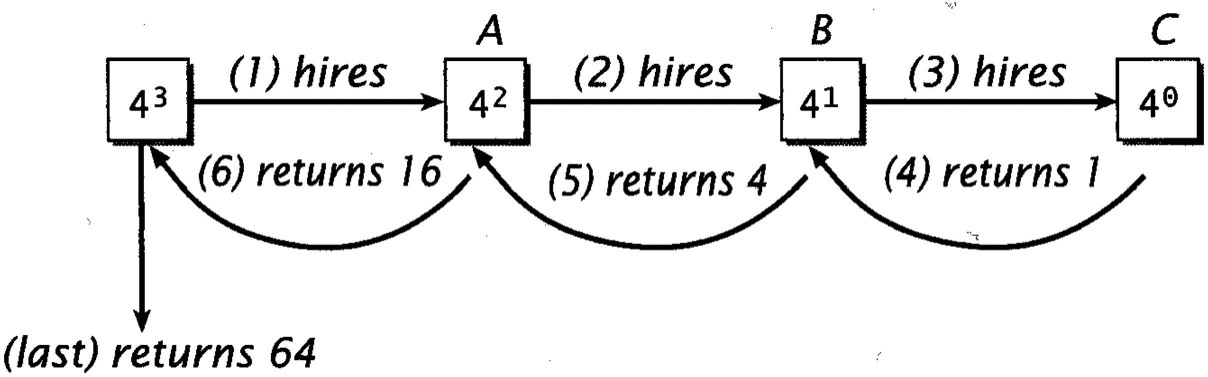
$x^y = 1 * x * x * ... * x$ (y times)

- if y == 0, then stop and return 1
- if y > 0,   then multiply x with the result of $x^{(y-1)}$

Exponent /2

```
private int power(int x, int y) {
   // y>=0 returns x**y

   if (y == 0)
      return 1;
   else {
        int assistantResult = power(x, y-1)
        return x * assistantResult;
   }
}
```

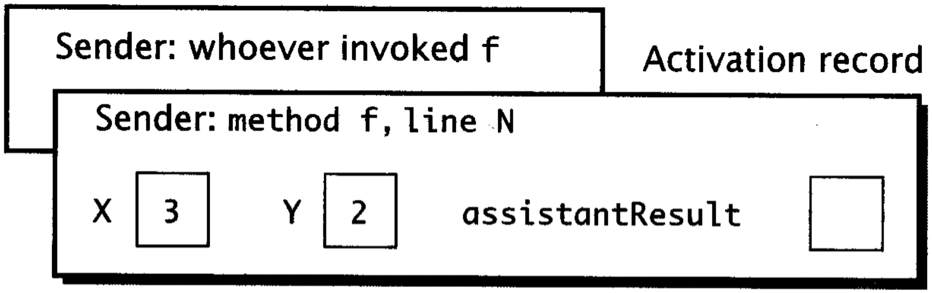


Activation records

- f() calls power(3, 2):

```
void f(..) {
   ...
   int q = power (3,2);
   ...
}
```

- activation record = memory block, with parameters, local variables, and return address:



Sender: whoever invoked f

Sender: method f, line N

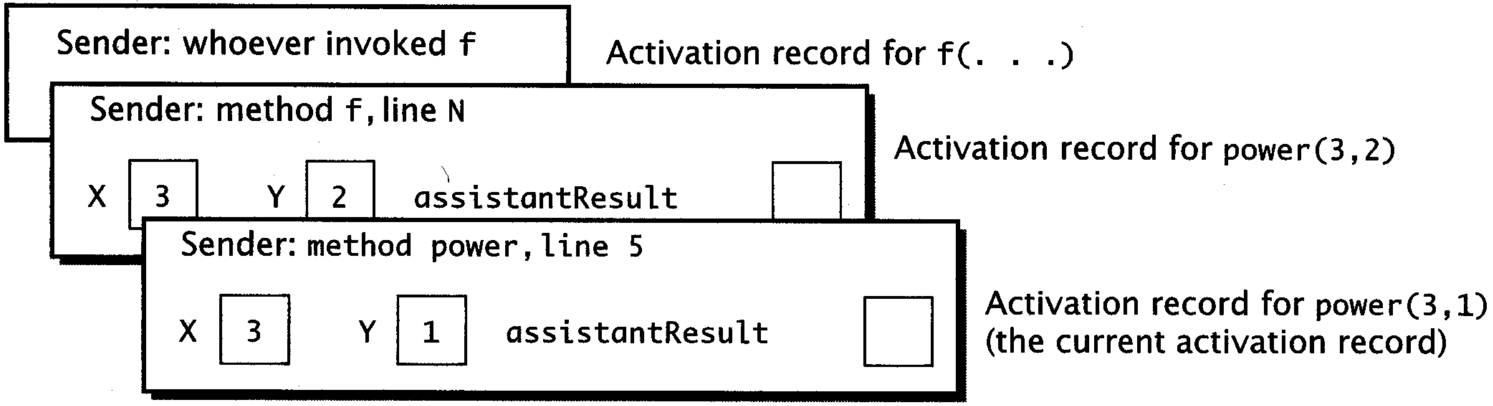
X 3 Y 2 assistantResult

Activation record for f(. . .)

Activation record for power(3,2)
(the current activation record)

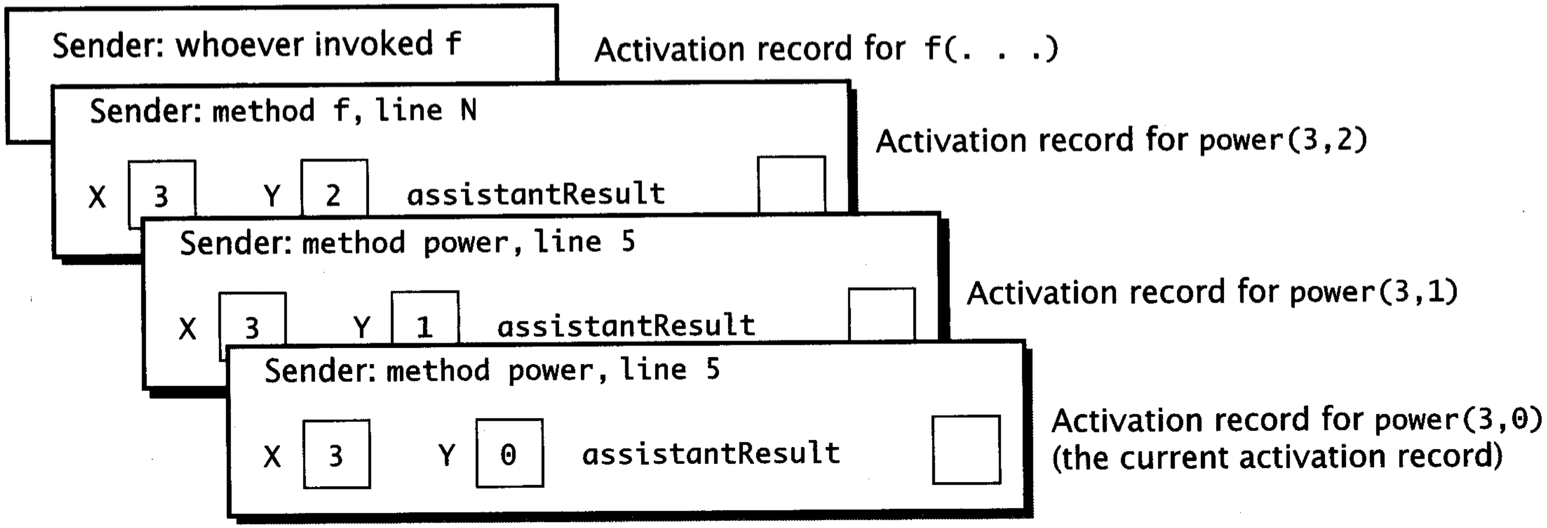
Stack of Activation records /2

- After `power (3,1) has been called:`


Sender: whoever invoked f
Activation record for f(. . .)
Sender: method f, line N
X 3 Y 2 assistantResult
Activation record for power(3,2)
Sender: method power, line 5
X 3 Y 1 assistantResult
Activation record for power(3,1)
(the current activation record)

Stack of Activation records /3

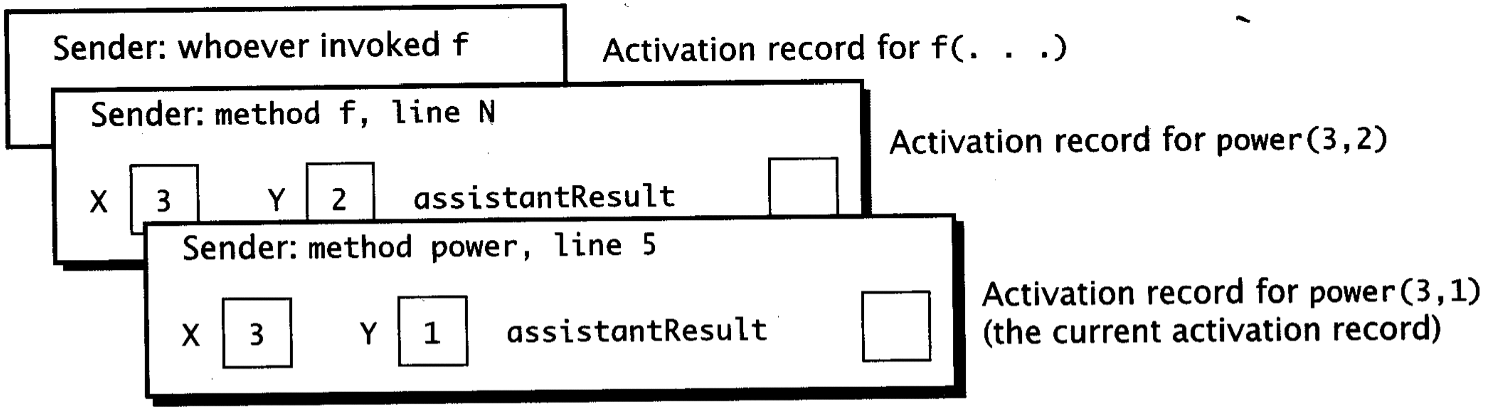
- After power (3,0) has been called


Sender: whoever invoked f
Activation record for f(. . .)
Sender: method f, line N
X 3
Y 2
assistantResult
Activation record for power(3,2)
Sender: method power, line 5
X 3
Y 1
assistantResult
Activation record for power(3,1)
Sender: method power, line 5
X 3
Y 0
assistantResult
Activation record for power(3,0)
(the current activation record)

Return

A return-Statement

- evaluates the return value (e.g., 1)
- deletes the current activation record
- replaces the expression that called the method with the return value
- continues execution of the caller



Sender: whoever invoked f

Activation record for f(. . .)

Sender: method f, line N

X 3 Y 2 assistantResult

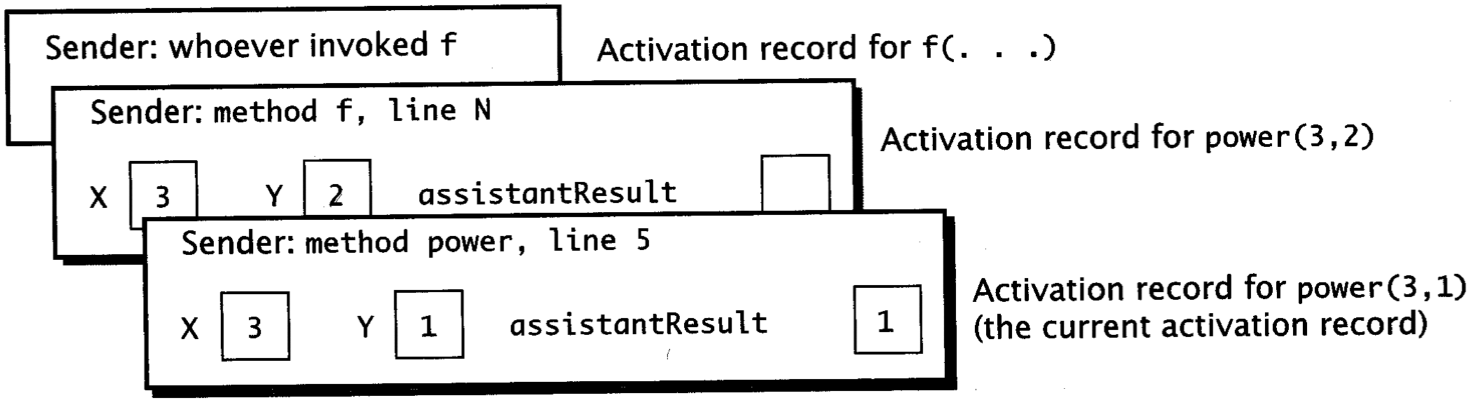
Activation record for power(3,2)

Sender: method power, line 5

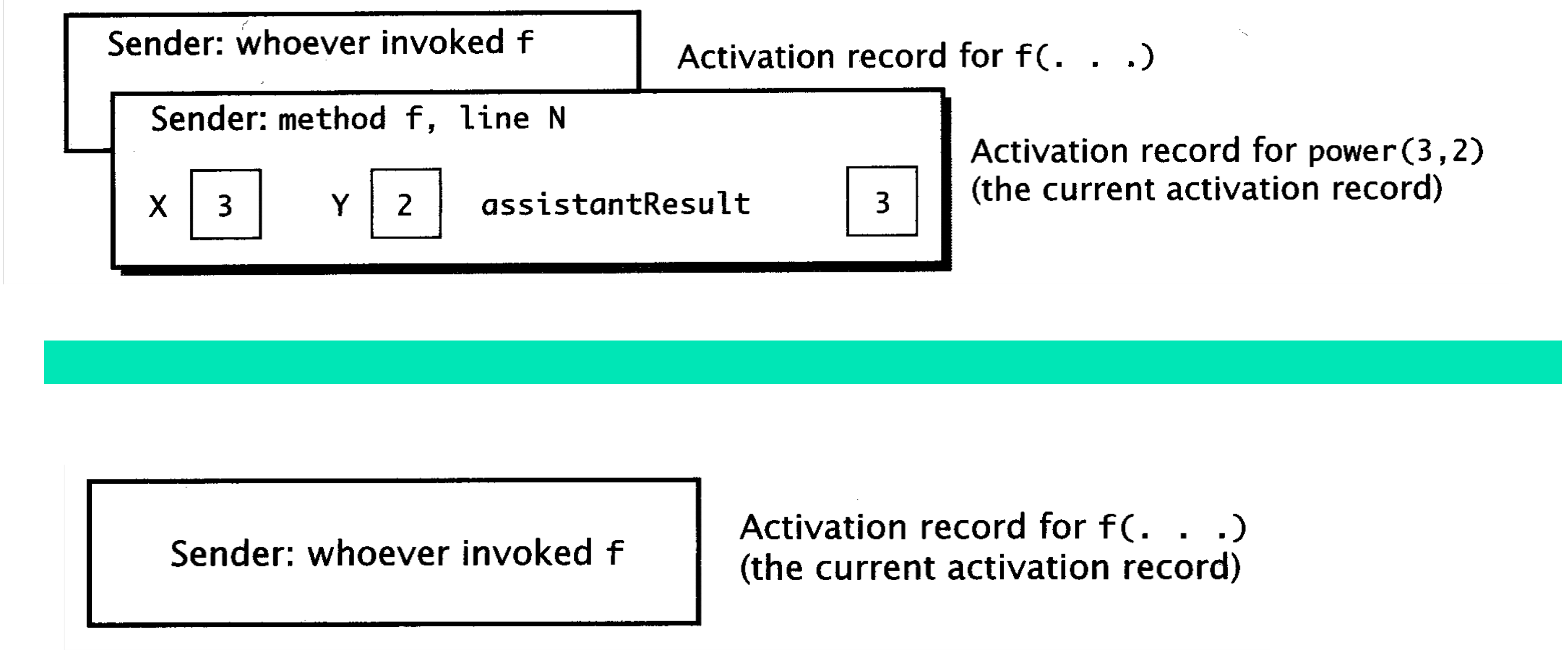
X 3 Y 1 assistantResult

Activation record for power(3,1)
(the current activation record)

Return /2


Sender: whoever invoked f
Activation record for f(. . .)
Sender: method f, line N
X 3 Y 2 assistantResult
Activation record for power(3,2)
Sender: method power, line 5
X 3 Y 1 assistantResult 1
Activation record for power(3,1)
(the current activation record)

Return /3



Sender: whoever invoked f

Activation record for f(. . .)

Sender: method f, line N

X 3 Y 2 assistantResult 3

Activation record for power(3,2)
(the current activation record)

Sender: whoever invoked f

Activation record for f(. . .)
(the current activation record)

Example: Digits to Words

- Write a definition that accepts a single integer and produces words representing its digits.
- Example
  - input: `223`
  - output: `two two three`

Digit to Words: Specification

If number has multiple digits, decompose algorithm into two subtasks

- Display all digits but the last as words
- Display last digit as a word

First subtask is smaller version of original problem

- Same as original task, one less digit

Recursion Guidelines

- The definition of a recursive method typically includes an `if-else` statement.
  - One branch represents a base case which can be solved directly (without recursion).
  - Another branch includes a recursive call to the method, but with a “simpler” or “smaller” set of arguments.
- Ultimately, a base case must be reached (termination).

Termination

- You need to have a return-statement that does not make a recursive call
- The return statement needs to be before the recursive call

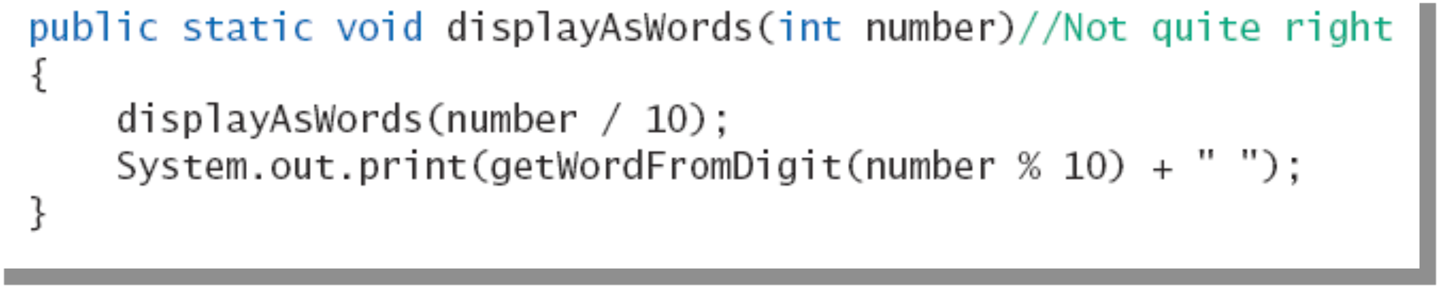
```
if (y == 0)
   return 1;
else { ...
   // recursive call
}
```

Infinite Recursion

- If the recursive invocation inside the method does not use a “simpler” or “smaller” parameter, a base case may never be reached.
- Such a method continues to call itself forever (or at least until the resources of the computer are exhausted as a consequence of *stack overflow*)
- This is called *infinite recursion*

Infinite Recursion

- Suppose we leave out the stopping case



```
public static void displayAsWords(int number)//Not quite right
{
    displayAsWords(number / 10);
    System.out.print(getWordFromDigit(number % 10) + " ");
}
```

- Nothing stops the method from repeatedly invoking itself
  - Program will eventually crash when computer exhausts its resources (stack overflow)

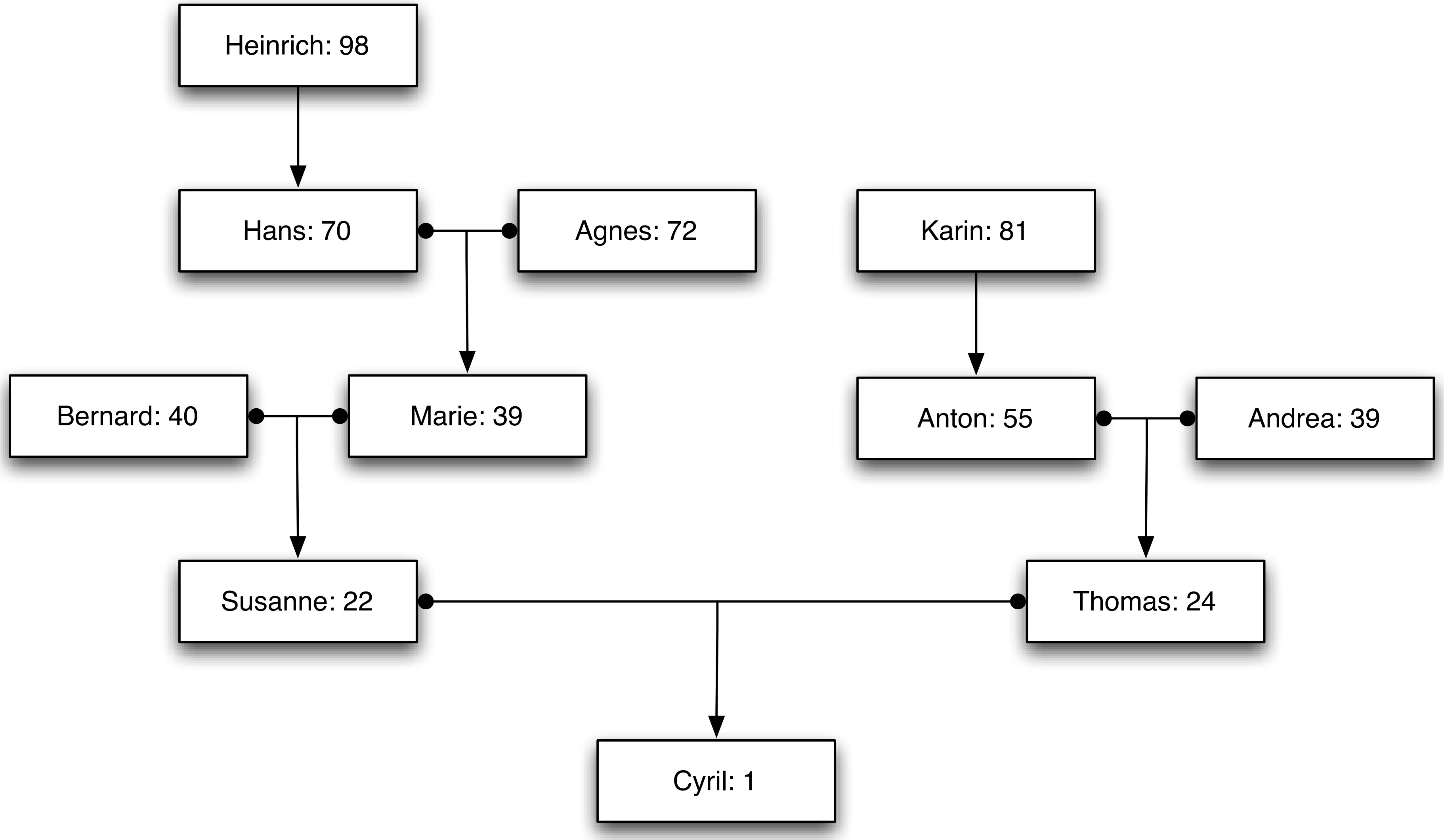
Recursive Versus Iterative

- Any method including a recursive call can be rewritten to do the same task *without* recursion
- Recursive method
  - Uses more storage space than iterative version
  - Also runs slower
- However in *some* programming tasks, recursion is a better choice, a more elegant solution

Overloading is Not Recursion

- If a method name is overloaded and one method calls another method with the same name but with a different parameter list, this is **not** recursion
- Of course, if a method name is overloaded and the method calls itself, this **is** recursion
- Overloading and recursion are neither synonymous nor mutually exclusive

Example: Family Tree



Summary

- To avoid infinite recursion recursive method should contain two kinds of cases
  - A recursive call
  - A base (stopping) case with no recursive call
- Good examples of recursive algorithms
  - Binary search algorithm
  - Merge sort algorithm
  - Operations in tree structures