Seminar today…

Software Development 3 (F27SG)
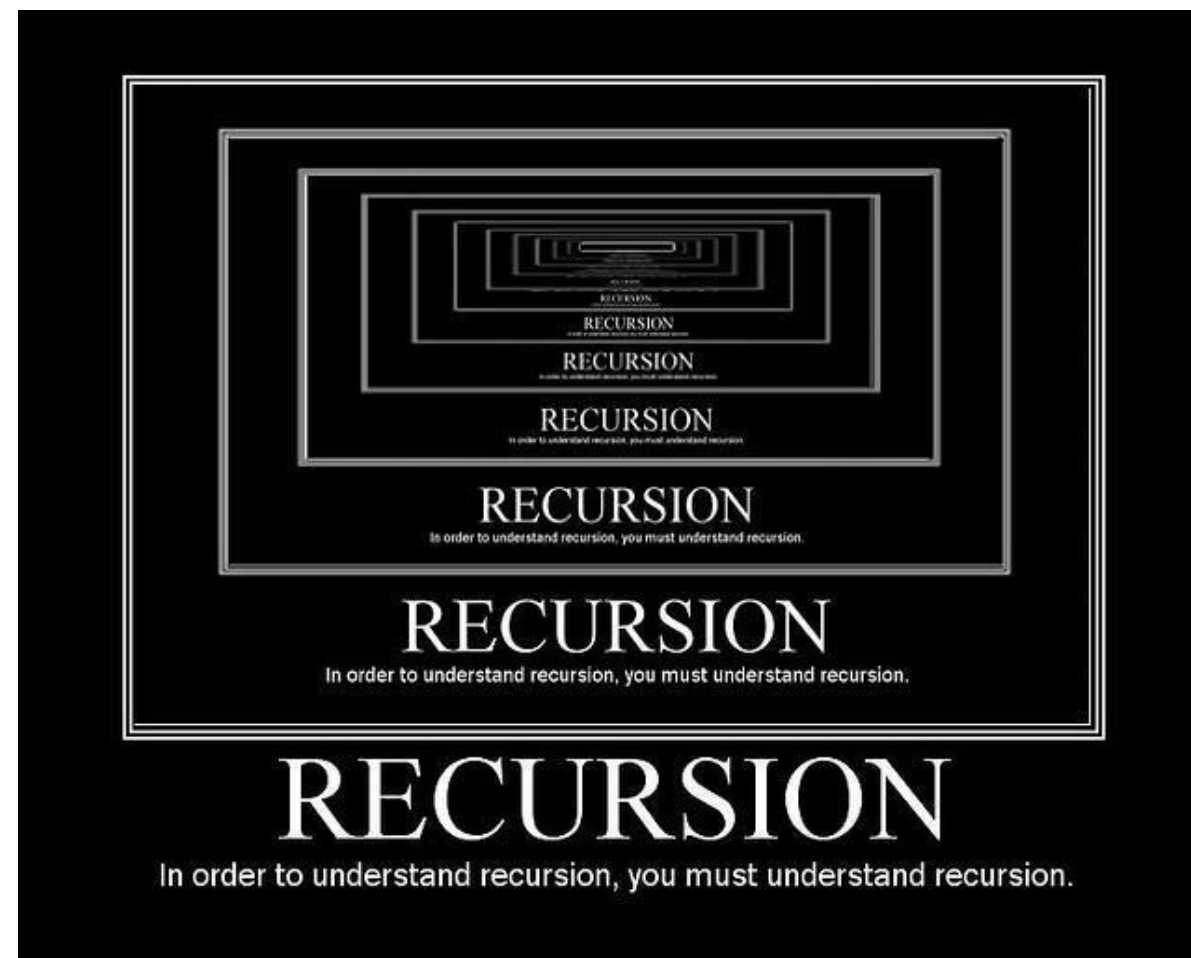
Lecture 5
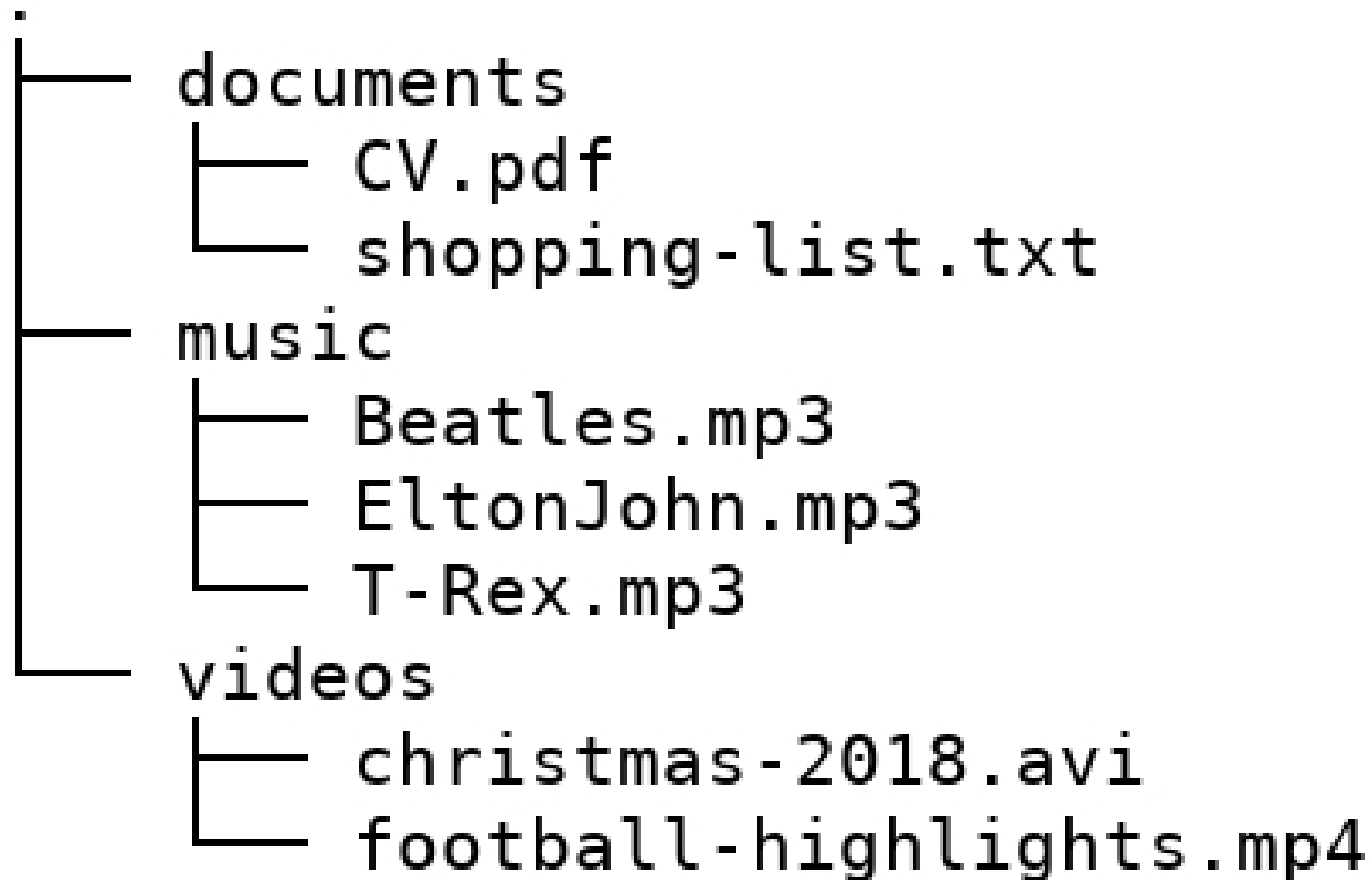
Recursion

Rob Stewart

Recursion

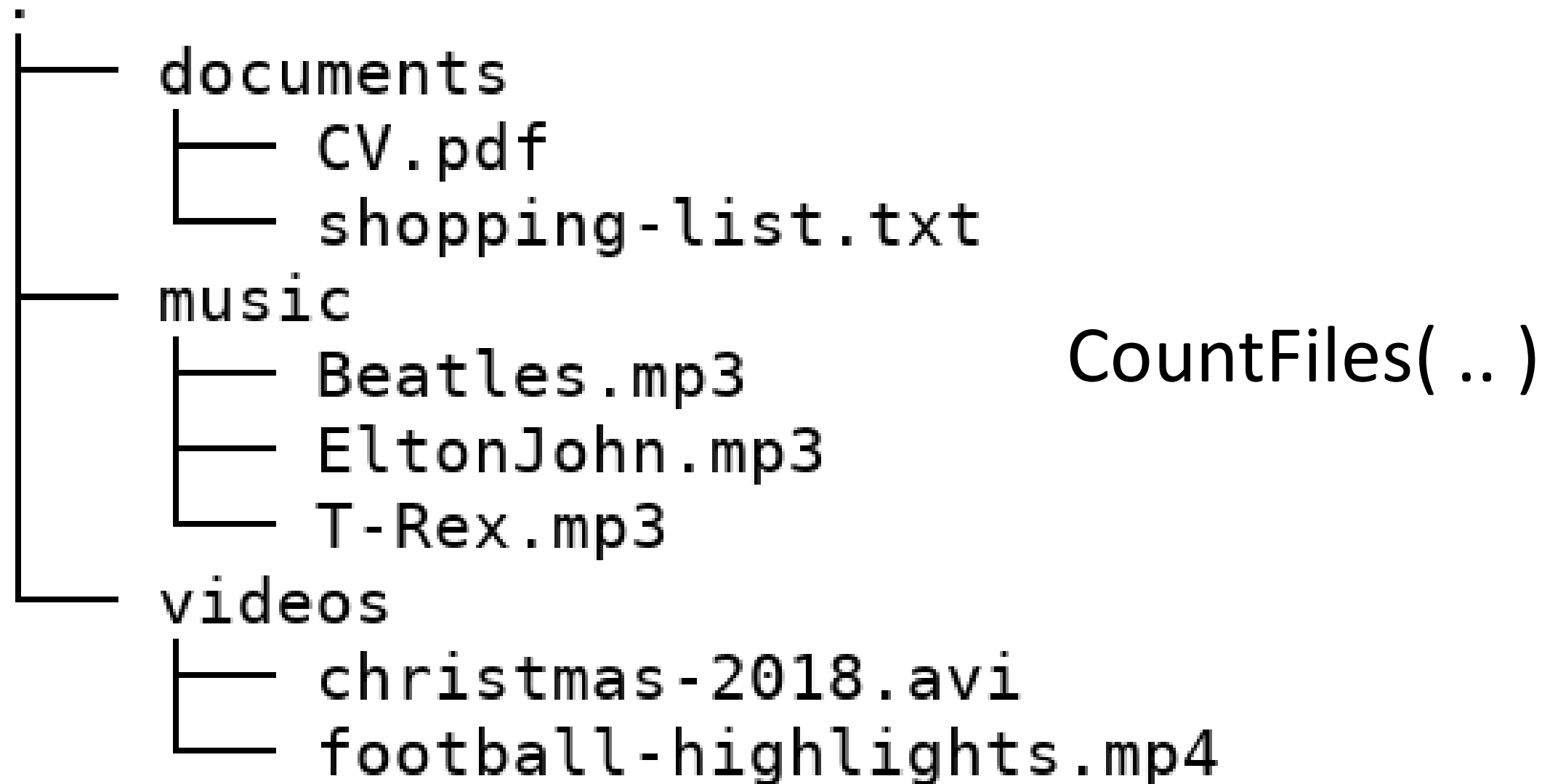- Dynamic data structures often uses a technique called **recursion**

Recursion

- A **recursive method** is a method that calls itself
  - typically with changed arguments,
  - or the with global class variables changed

All Files in a Path

```
.
├── documents
│   ├── CV.pdf
│   └── shopping-list.txt
├── music
│   ├── Beatles.mp3
│   ├── EltonJohn.mp3
│   └── T-Rex.mp3
└── videos
    ├── christmas-2018.avi
    └── football-highlights.mp4

3 directories, 7 files
```

All Files in a Path

```
.
├── documents
│   ├── CV.pdf
│   └── shopping-list.txt
├── music
│   ├── Beatles.mp3
│   ├── EltonJohn.mp3
│   └── T-Rex.mp3
└── videos
    ├── christmas-2018.avi
    └── football-highlights.mp4

3 directories, 7 files
```

CountFiles(..)

All Files in a Path

```
7 .
2 ├── documents
1 │   ├── CV.pdf
1 │   └── shopping-list.txt
3 ├── music
1 │   ├── Beatles.mp3
1 │   ├── EltonJohn.mp3
1 │   └── T-Rex.mp3
2 └── videos
1     ├── christmas-2018.avi
1     └── football-highlights.mp4
```
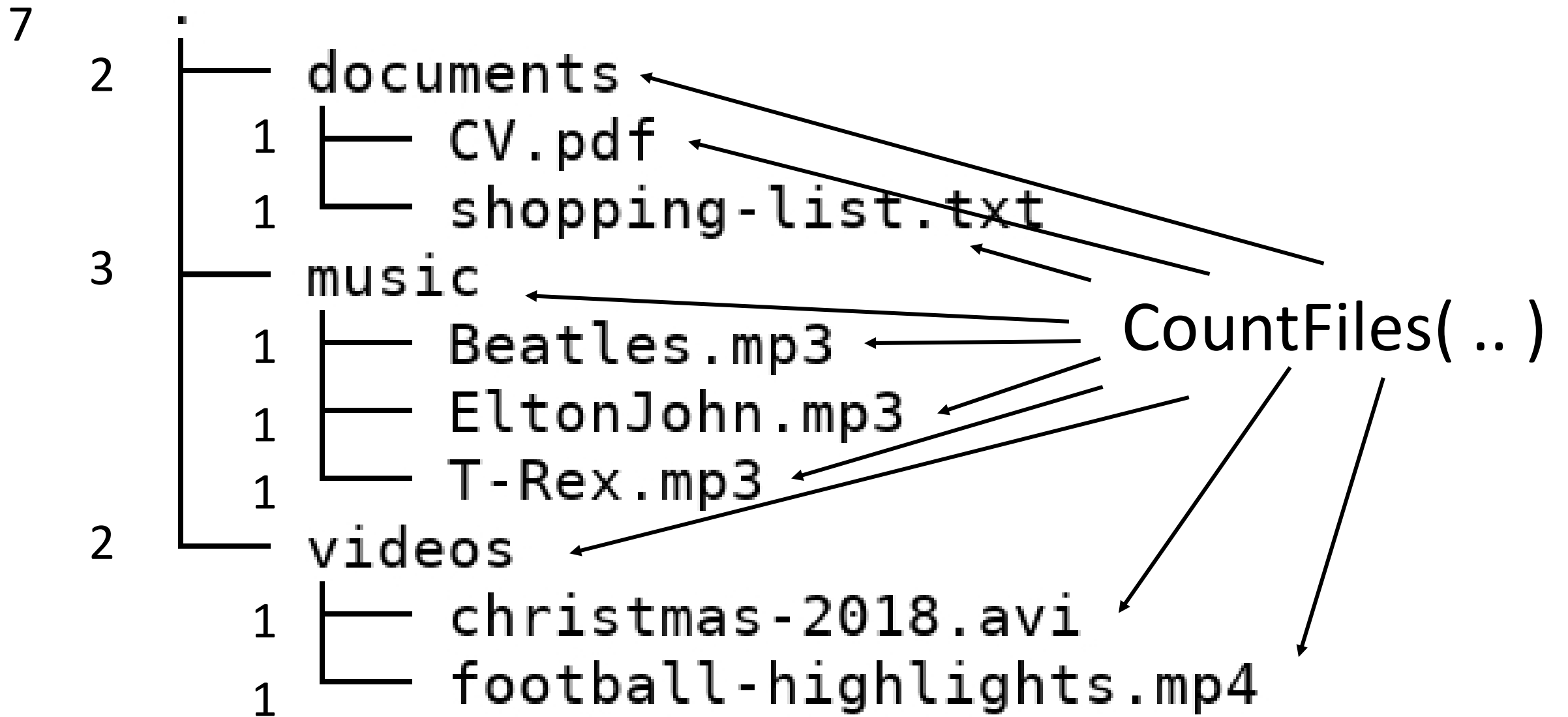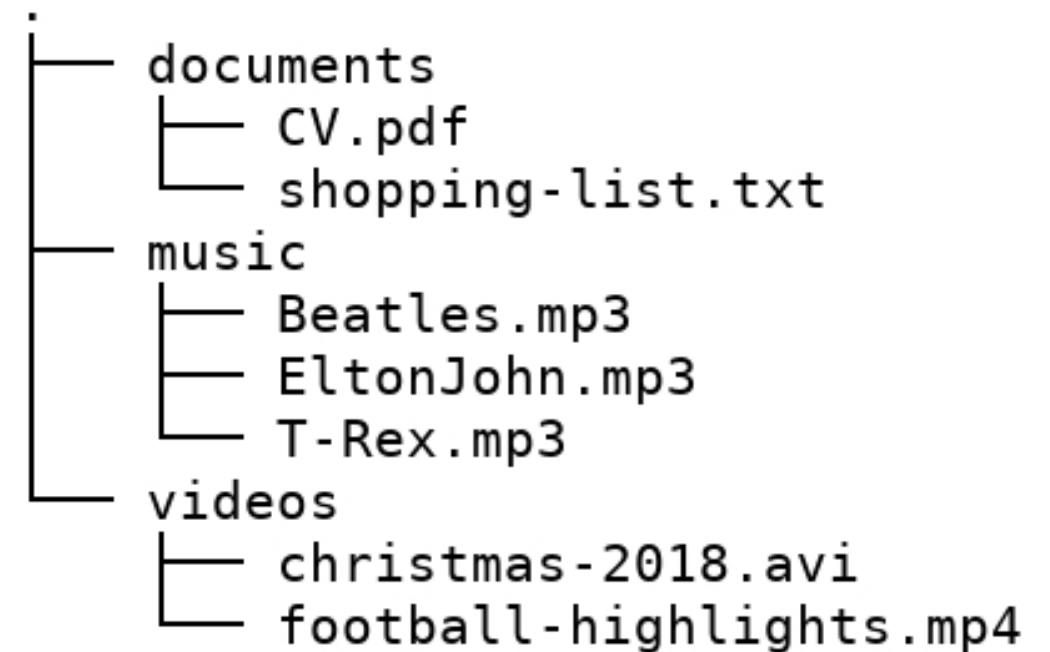
CountFiles(..)

3 directories, 7 files

All Files in a Path

```
public int countFIles(File filePath) {
  if ( filePath.isFile() ) {
    return 1;
  }


  int count = 0;
  File[] files = filePath.listFiles();
    for (File fileOrDir : files) {


    }

  return count;
}
```

```
.
├── documents
│   ├── CV.pdf
│   └── shopping-list.txt
├── music
│   ├── Beatles.mp3
│   ├── EltonJohn.mp3
│   └── T-Rex.mp3
└── videos
    ├── christmas-2018.avi
    └── football-highlights.mp4

3 directories, 7 files
```

All Files in a Path

```
public int countFIles(File filePath) {
  if ( filePath.isFile() ) {
    return 1;
  }


  int count = 0;
  File[] files = filePath.listFiles();
    for (File fileOrDir : files) {
        count += countFiles(fileOrDir);
    }
  return count;
}
```
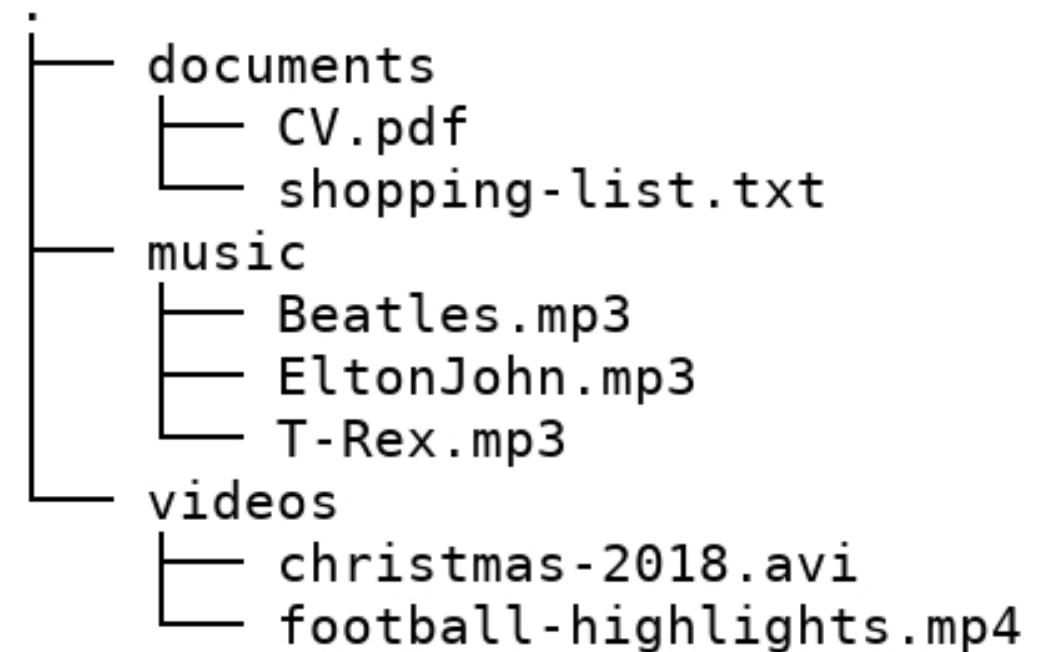
```
.
├── documents
│   ├── CV.pdf
│   └── shopping-list.txt
├── music
│   ├── Beatles.mp3
│   ├── EltonJohn.mp3
│   └── T-Rex.mp3
└── videos
    ├── christmas-2018.avi
    └── football-highlights.mp4

3 directories, 7 files
```

Recursion

- Recursion is a very powerful technique
  - heavily used in **functional programming languages**
  - it may feel unnatural at first
  - but provides clear and elegant solution to many problems
- *Anything you can do with recursion in Java, you can also do with iteration (for/while loops)*
  - *… but the code may be harder to understand*

Writing Recursive Methods

- Base cases
  - values for the input variables where the method does not call itself

- Step/recursive cases
  - values for input variables where the method calls itself
  - each chain of recursive calls must make progress towards a base case

Writing Recursive Methods

- A recursive method is
  - a method that calls itself
- The parameters are changed for each call
- We separate between two cases
  - **base** cases
  - **step** cases (recursive cases)
- The type of case you are in
  - depends on the values of the input variables

Recursion example

- The classic example of recursion is the factorial function:

$$n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n$$

- Recursively, this is written:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times f(n-1) & \text{else} \end{cases}$$

- In Java we can write this as

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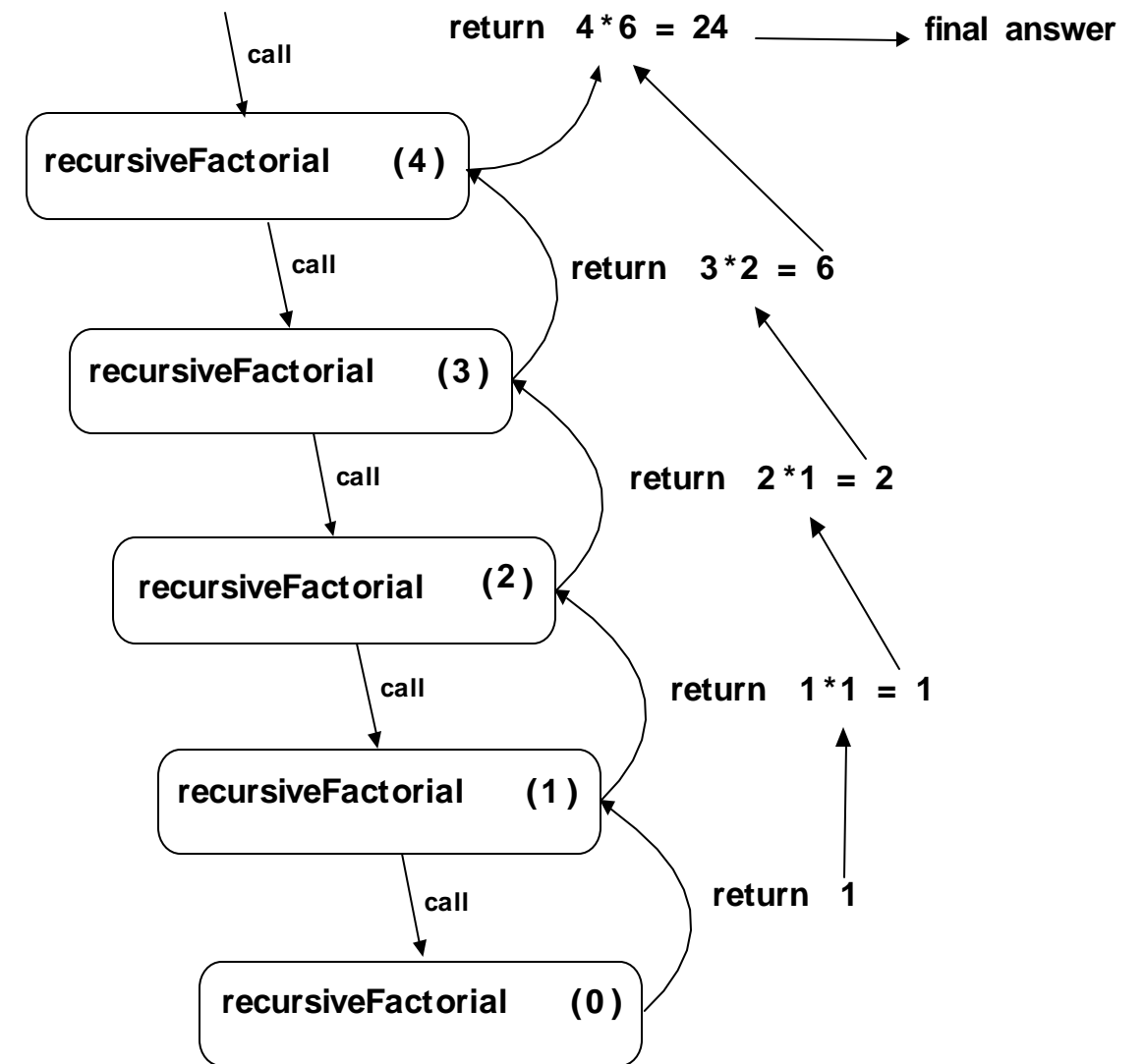
base

step

Recursion visualised

- The example shows how 4! is computed
  - each box is a method call
  - a (down) arrow from a caller to a callee
  - an (up) arrow from a callee to a caller with the return value

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

return 4*6 = 24 → final answer

call

recursiveFactorial (4)

call   return 3*2 = 6

recursiveFactorial (3)

call   return 2*1 = 2

recursiveFactorial (2)

call   return 1*1 = 1

recursiveFactorial (1)

call   return 1

recursiveFactorial (0)

Converging to the base case

- **recursiveFactorial(n)** only works when n >= 0
  - if n < 0 then it will not move towards a base cases
  - and will eventually result in a *stack overflow*

Recursion Example

Using only:
- str.isEmpty()
- str.substring(int beginIndex)
- Recursive call to *length(..)*
- The + operator

Eclipse demo

```
public int length(String str){
    // implement this method
}
```
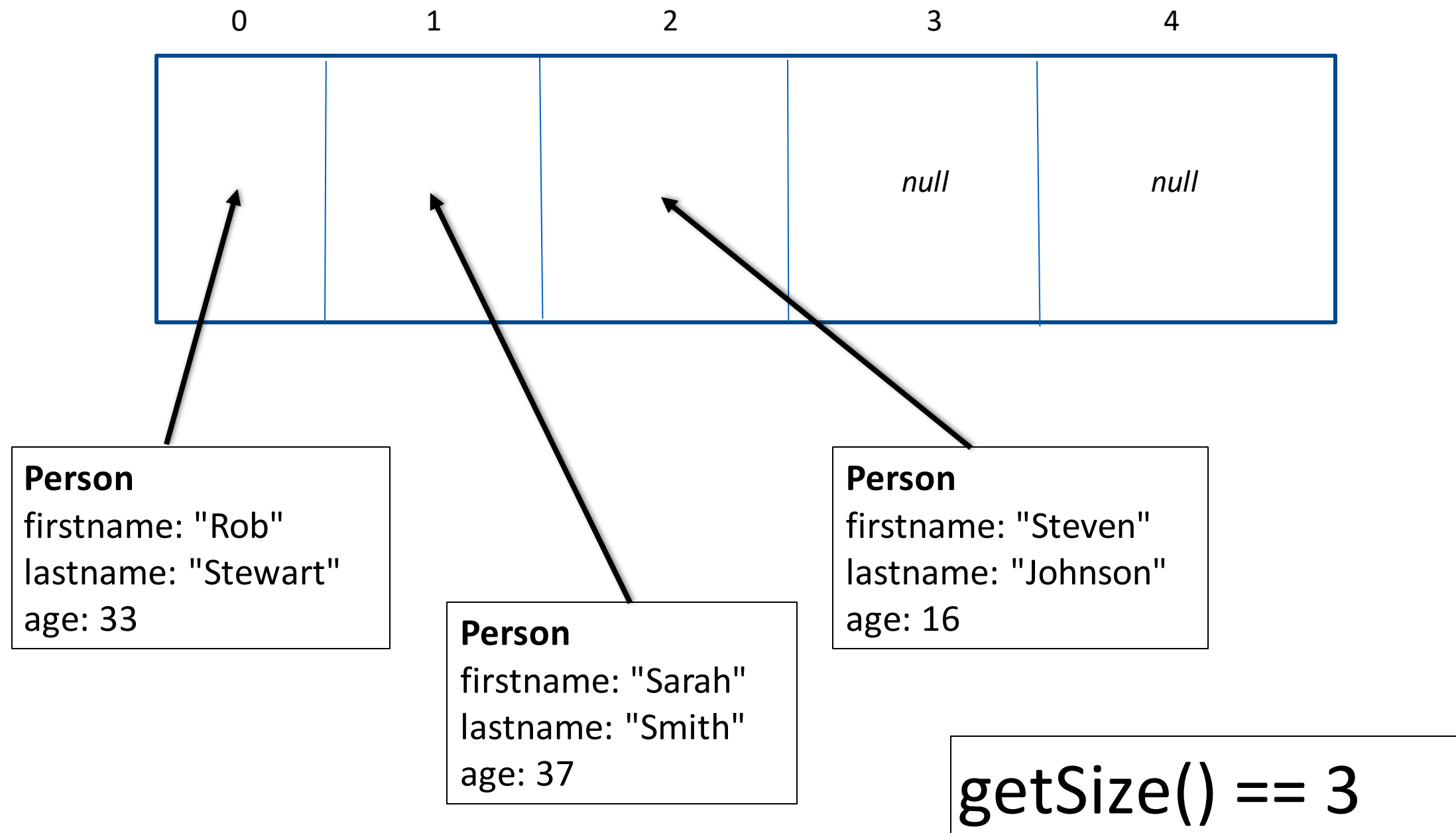
length("foo") = 3
length("") = 0
length("software") = 8

Recursive maxAge()

	0	1	2	3	4
				null	*null*

Person
firstname: "Rob"
lastname: "Stewart"
age: 33

Person
firstname: "Sarah"
lastname: "Smith"
age: 37

Person
firstname: "Steven"
lastname: "Johnson"
age: 16

getSize() == 3

Recursive maxAge()

```
Public class Collection {

    ...

    public int maxAge() {
        // recursive code here

    }

}
```

Eclipse demo

Exercise

```
public void swap(int arr[], int i,int j){
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
}
```

The **swap** method swaps two locations in an array. Explain what the following method does:

```
1.    public void mymethod(int[] arr, int i,int j){
2.        if (i < j){
3.            swap(arr,i,j);
4.            mymethod(arr,i+1,j-1);
5.        }
6.    }
```

a) Will it terminate?

b) What is the base case? What is the step case?
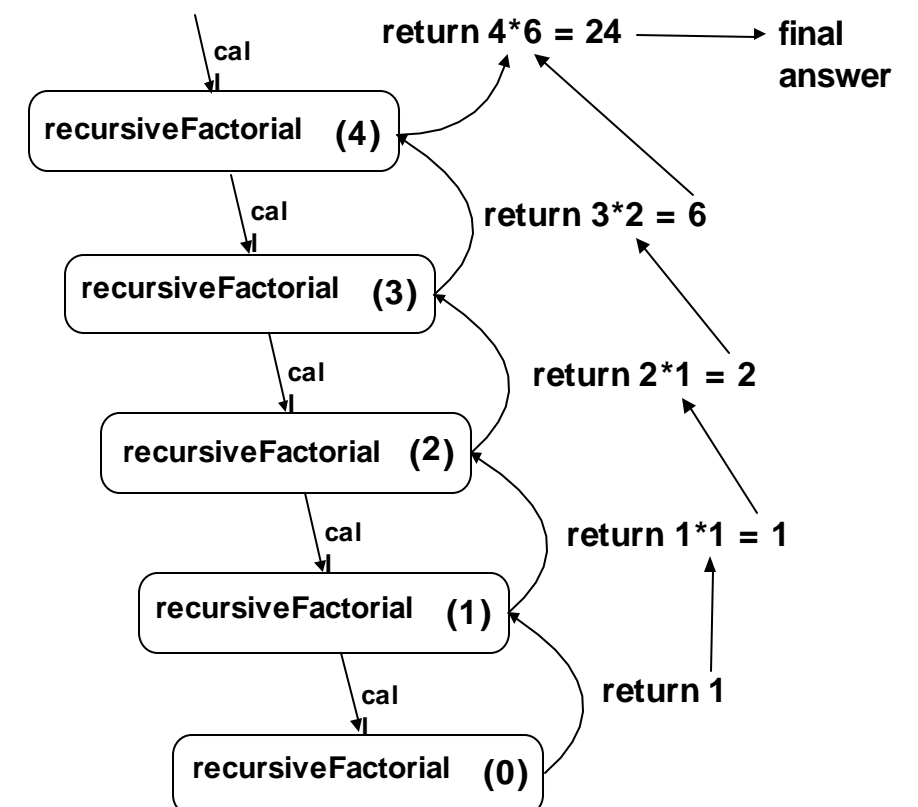
c) Given that **int[] array A = {4,3,6,2,5}**

 Give the *recursion trace* for when calling

   mymethod(A,0,4);

For each step in the recursion trace you should include

* Arguments of mymethod
* Value of A

recursion trace example

Recursion Space Race

www.socrative.com
student login
Room: *SD32019*

Summary

- Introduced programming technique called **recursion**

- Attendance sheet

- **Next lecture**:
  - **Dynamic data structures**: linked lists
  - **Recursive operations:** on linked lists