Introduction to Data Structures and algorithms (F28SG)

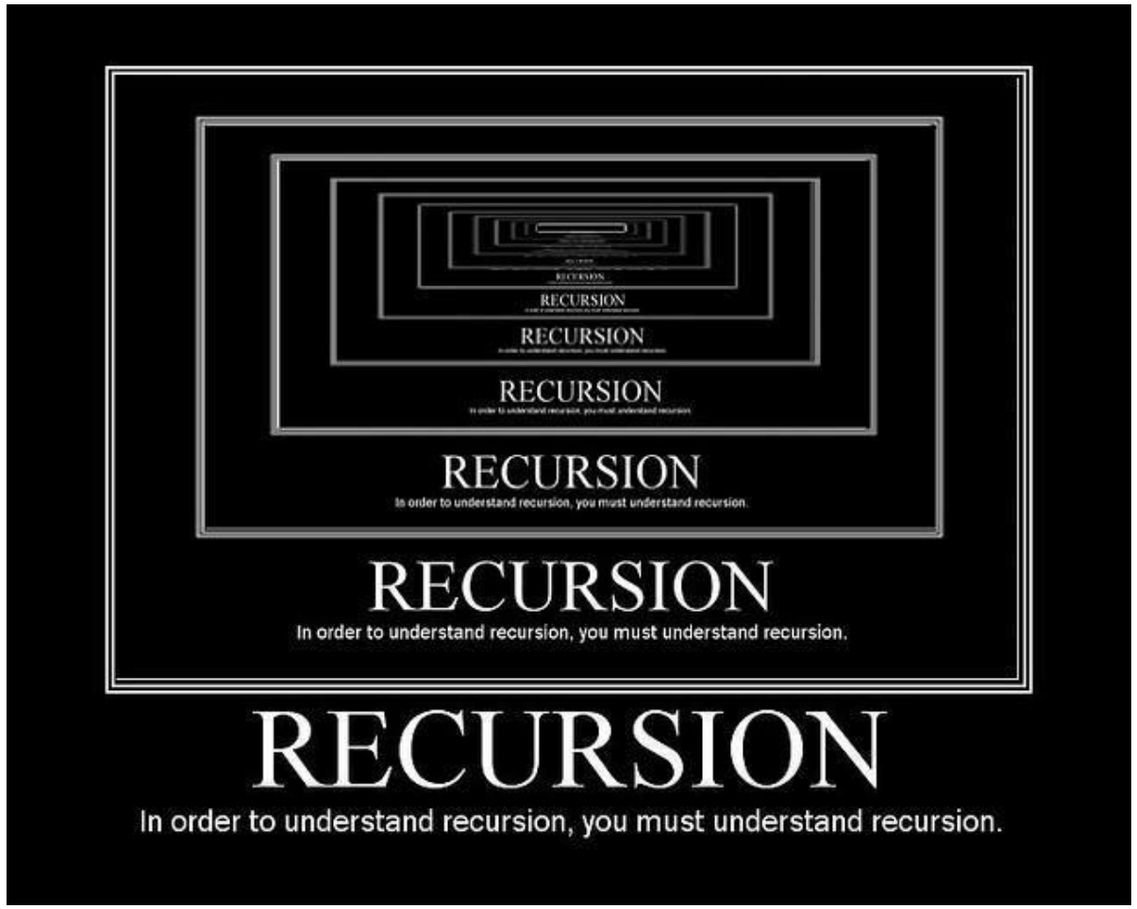
Lecture 4

Recursion

Rob Stewart

Recursion

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



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

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

- *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*

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:

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

- In Java we can write this as

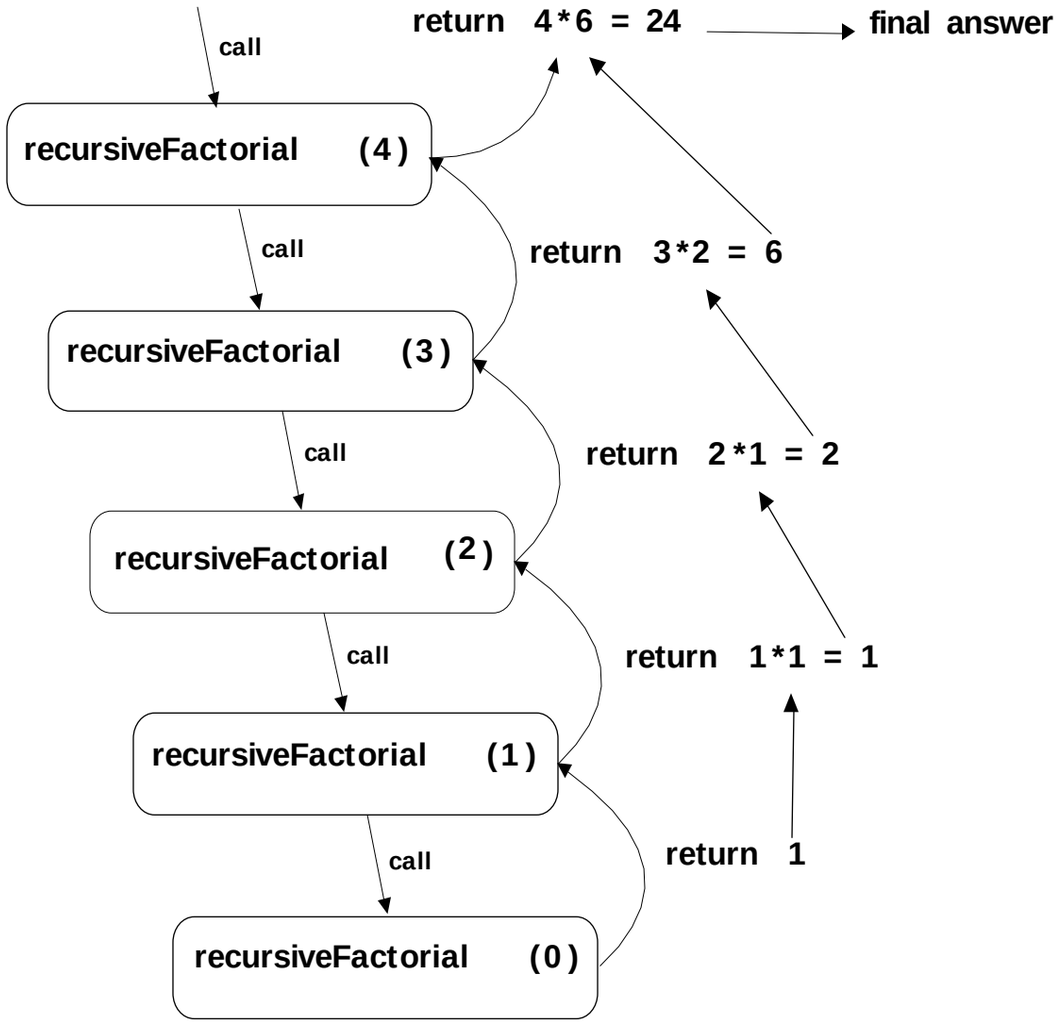
```
public static int recursiveFactorial(int n){
  if (n == 0) return 1;
  else return n * recursiveFactorial(n-1);
}
```

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);
}
```

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

3 directories, 7 files
```

CountFiles(..)

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
```

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;
}
```

Base case

Recursive case

```
.
├── 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
```

Writing Recursive Methods

- We separate between two cases
  - **base** cases
  - **step** cases (recursive cases)
- Base cases
  - values for the input variables
  - ***method does not call itself***
- Step/recursive cases
  - ***method calls itself*** with different parameters
  - must make progress towards a base case

Converging to the base case

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

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

Count even values in array of integers

Evens.java

```
public static int countEvens(int[] numbers) {
  // implement this method...
}


// .. using this method
private static int countEvens(int[] numbers, int index)
{
  // implement this recursively
}
```

Hint:

base case is when you reach end of the array

Recursion Example

Using only:

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

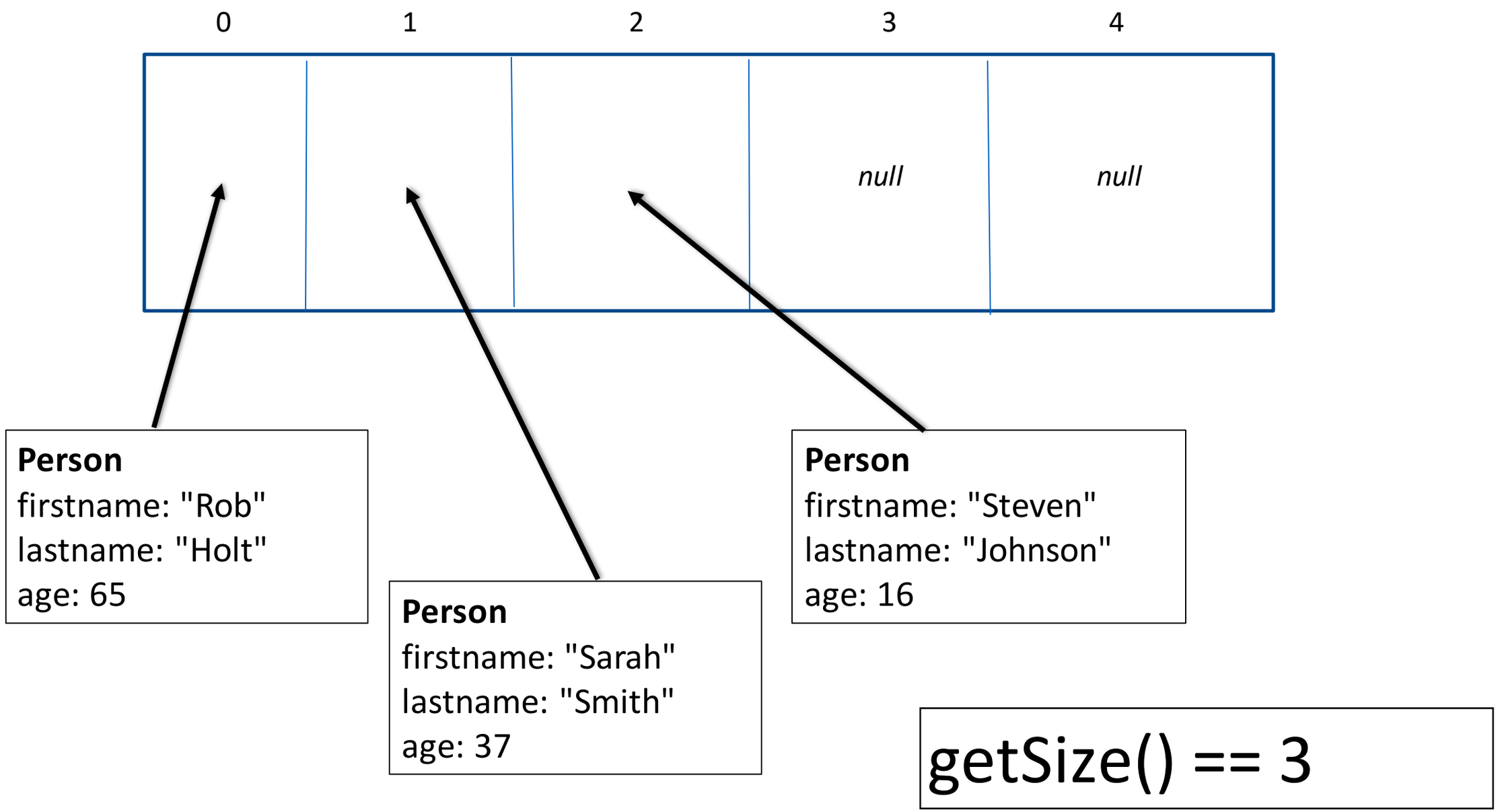
Hint: consider Using equals(“”)

```
public int length(String str){
    // implement this method
}
```

length(“foo”) = 3
length(“”) = 0
length(“software”) = 8

Hint: this is the base case

Recursive maxAge()



Recursive maxAge()

```
Public class Collection {
  ...

  public int maxAge() {
      // recursive code here
  }

}
```

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

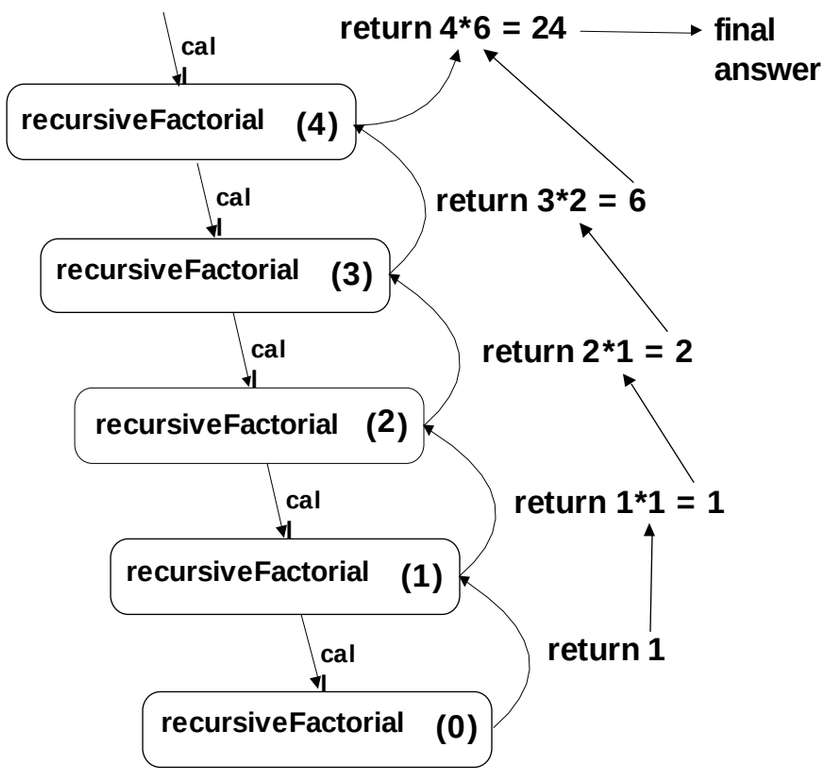
mymethod(A,0,4);

For each step in the recursion trace you should include

- Arguments of mymethod
- Value of A

recursion trace

example



Summary

- Introduced programming technique called **recursion**

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