CSCI 136:
Data Structures
and
Advanced Programming

Lecture 7

Recursion, part 1

Instructor: Dan Barowy

Williams

Announcements

- Lab 1 feedback coming today.
- If you had a Github snafu, see me after class.
- Lab 3: quasi-random partners
- "I know that the TA's have busy lives just as I do, but I would really love it if there were more TA hours on Saturdays. Sunday TA hours are busy and stressful. Would love if that was a possibility!"

Outline

Study tip

Pre/post conditions

Recursion

Recursion activity

Recursion tradeoffs

Life skill #7

Engineer the outcome you want

Life skill #7

Engineer the outcome you want

More specifically, ask yourself:
"What efforts **yield the greatest return on investment**?"

Do the things that **get you closest to your goal, fastest**.

If you do not know what your goal is, college is the time to **start thinking about it**!

Life skill #7

Engineer the outcome you want

If **your goal is an A grade**, then you might be tempted to think that copying will yield the greatest return.

First: **Getting an A is not really your goal.**
More likely: **getting a good job; personal satisfaction**.

Second: Suppose you got that job through cheating; **how long do you think you can keep it?**

Pre/post conditions

Pre-condition

A **pre-condition** is a **true/false statement** (a "predicate") that must always be true **prior to a code segment (e.g., a function) being called**. If a pre-condition is false, the result of executing the code is **undefined**.

Post-condition

A **post-condition** is a **true/false statement** (a "predicate") that must always be true **after a code segment (e.g., a function) is called**. Usually, if a pre-condition is false, there will be no guarantee that the post-condition is true.

Example

```
int z = x + 1;
```

What does this operation do?
(i.e., what is our desired post-condition?)

Example

```
int z = x + 1;
```

Are you sure?

(code)

Example

```
char x = 'a';
int z = x + 1;
```

`z` equals `98`

Are you sure?

Example

```
char x = 'a';
int z = x + 1;
```

What should our pre-conditions have been?

1. **`x`** is an **`int`**
2. **`x < Integer.MAX_VALUE`**

Pre/post conditions

- Recall **`charAt(int index)`** in Java **`String`** class
- What are the pre-conditions for **`charAt`**?
  - **`0 <= index < length()`**
- What are the post-conditions?
  - Method returns char at position index in string
- It's a good idea to put pre- and post-conditions in comments before your methods

```
/* pre: 0 ≤ index < length
 * post: returns char at position index
 */
public char charAt(int index) { … }
```

Pre/post conditions

- Pre and post conditions **form a contract**
- *Principle: Ensure Post-condition is satisfied if pre-condition is satisfied*
- Examples:
  - `s.charAt(s.length() – 1)`: index < length, so valid
  - `s.charAt(s.length() + 1)`: index > length, not valid
- These conditions document requirements that user of method should satisfy
- But, as comments, they are not enforced

`Assert` class

- Pre- and post-condition comments are useful as a programmer, but it would be really helpful to know as soon as a pre-condition is violated (and return an error)
- The **`Assert`** class (in **`structure5`** package) allows us to programmatically check for pre- and post-conditions

Remember: "Assume your code will fail."

`Assert` class

The **`Assert`** class contains the **`static`** methods

```
public static void pre(boolean test, String message);
public static void post(boolean test, String message);
public static void condition(boolean test, String message);
public static void fail(String message);
```

If the boolean test is NOT satisfied, an exception is raised, the message is printed and the program halts.

Example

```
// Pre: x is an int < MAX_VALUE
// Post: returns number one greater than number given
public static int addOne(int x) {
      Assert.pre(x < Integer.MAX_VALUE, "x must be an
integer less than MAX_VALUE.");
      int z = x + 1;
      Assert.post(z > x, "z must be greater than x.");
      return z;
}
```

General guidelines

1. State pre/post conditions in comments
2. Check conditions in code using **`Assert`**
3. Use **`Fail`** in unexpected cases (such as the default block of a switch statement)

- Any questions?
- You should use **`Assert`** in Lab 3

Recursion

Recursion

- General problem solving strategy
  - Split **big problem** into **smaller sub-problems**.
  - Sub-problems may look a lot like original; are often **smaller versions of same problem**!

Recursion

Recursion occurs when a thing is **defined in terms of itself**. The most common application of recursion in computer science, is **when a function is called within its own definition**.

Recursion

- **Many** algorithms are recursive
  - Often **easier to understand** (and prove correctness/state efficiency of) than non-recursive versions!

Recursion

- `n! = n × (n-1) × (n-2) × … × 1`
- How can we implement this?
  - We could use a `for` loop…

```
int product = 1;
for(int i = 1;i <= n; i++)
    product *= i;
```

- But we could also write it recursively….

Activity: Factorial

- `n! = n × (n-1) × (n-2) × … × 1`
- But we can also write it recursively.
- Work with a partner and see if you can come up with a recursive solution.

How did we know to look for that insight?

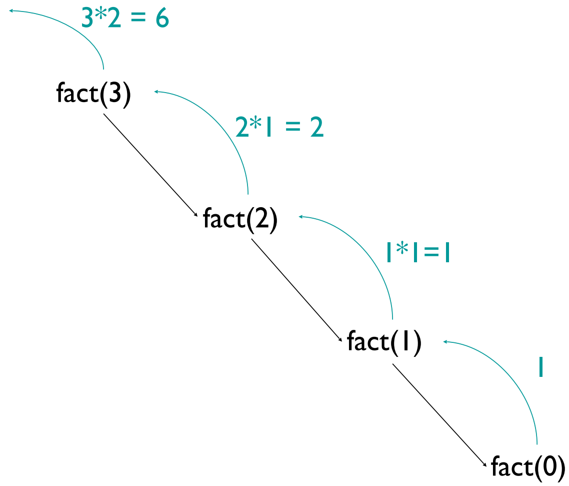
- `n! = n × (n-1) × (n-2) × … × 1`
- `n! = n × (n-1)`
- `0! = 1`

Recursion: formal structure

- Recursion is a good solution when a problem fits a **basic pattern**:
- It has at least one "terminating" rule that **does not** use recursion, called the **base case**.
- It has at least one rule that **does** use recursion, called the **recursive case**. The recursive case should **reduce the problem toward the base case**.

We will talk about formal (i.e., "inductive") proofs for recursion next class.

Graphically…



Building a wall (recursively)

What are our base/recursive cases?

(suppose we have infinite bricks)

Recursion tradeoffs

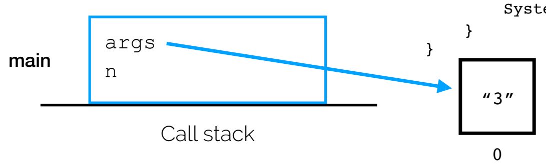
- Advantages
  - Often easier to construct recursive solution
  - Code is usually cleaner
  - Some problems do not have obvious non-recursive solutions
- Disadvantages
  - Time cost of recursive calls
  - Memory cost (need to store state for each recursive call until base case is reached)

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

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

Call stack

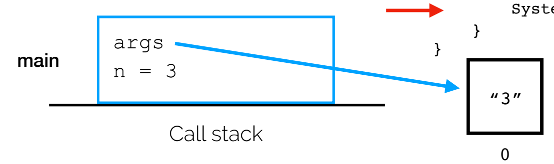
Call program with input "3".

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

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



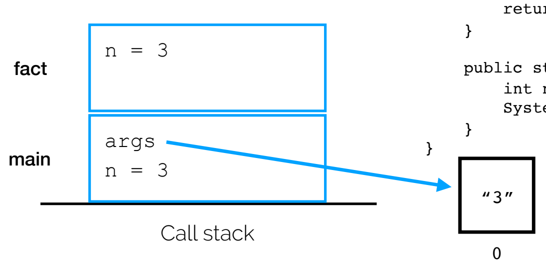
Call program with input **"3"**.

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

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

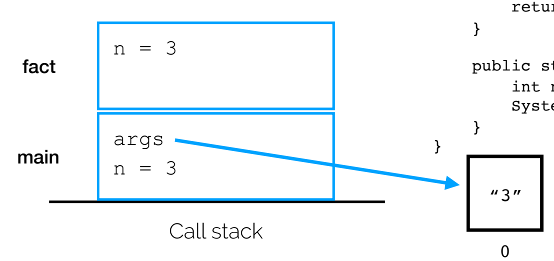


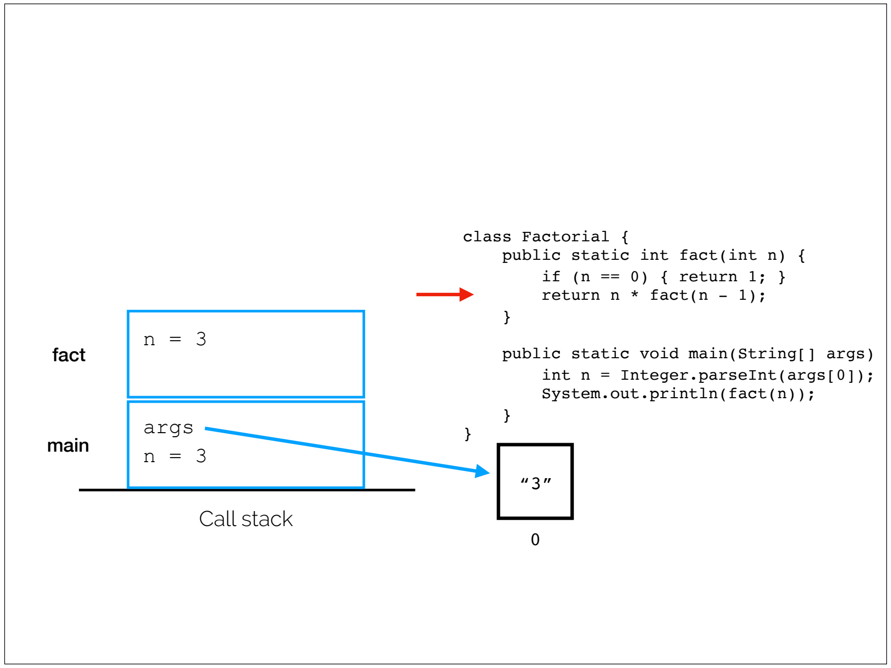
I skipped a subtlety here; did you spot it?

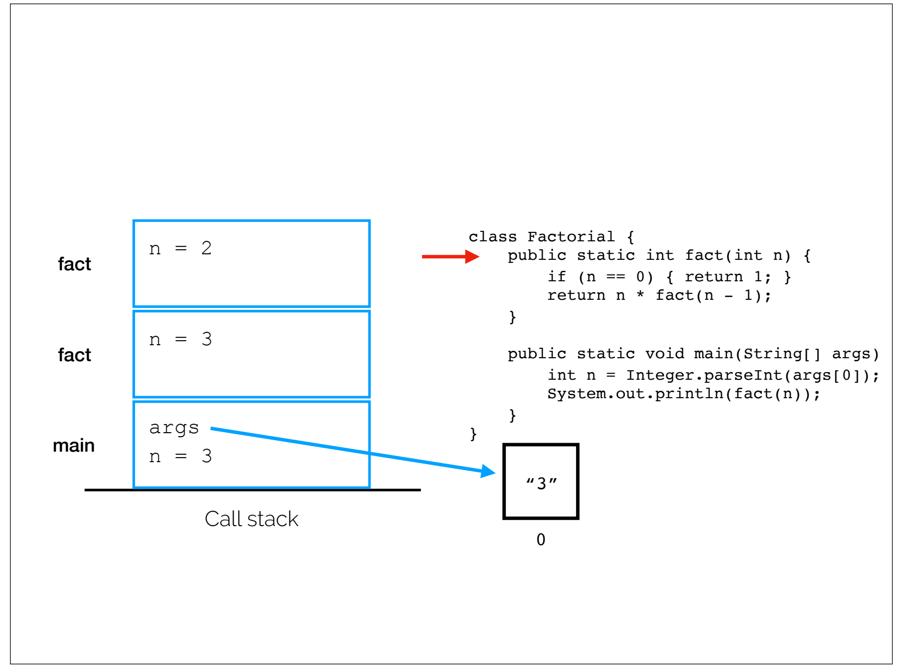
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

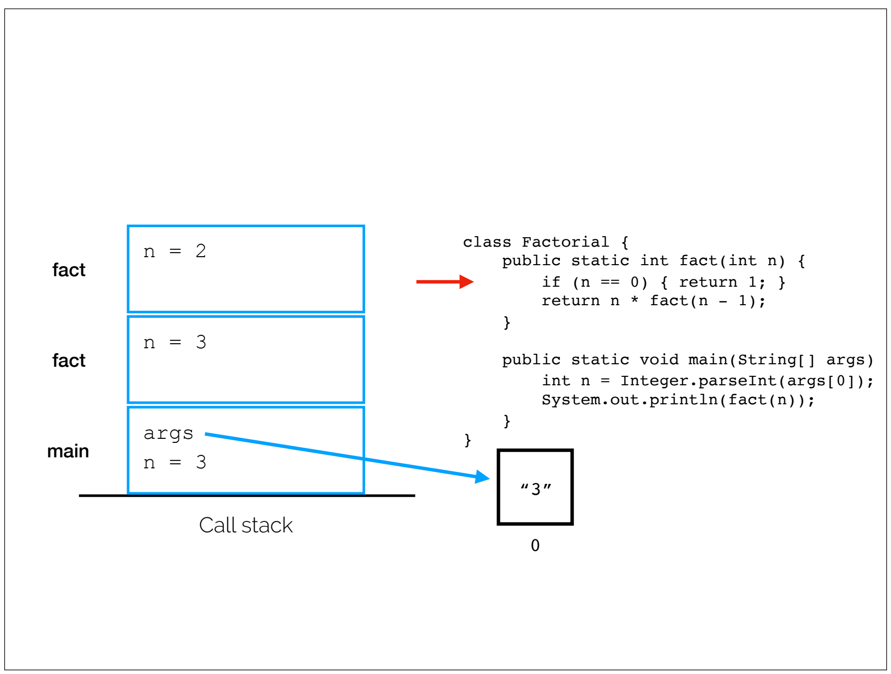


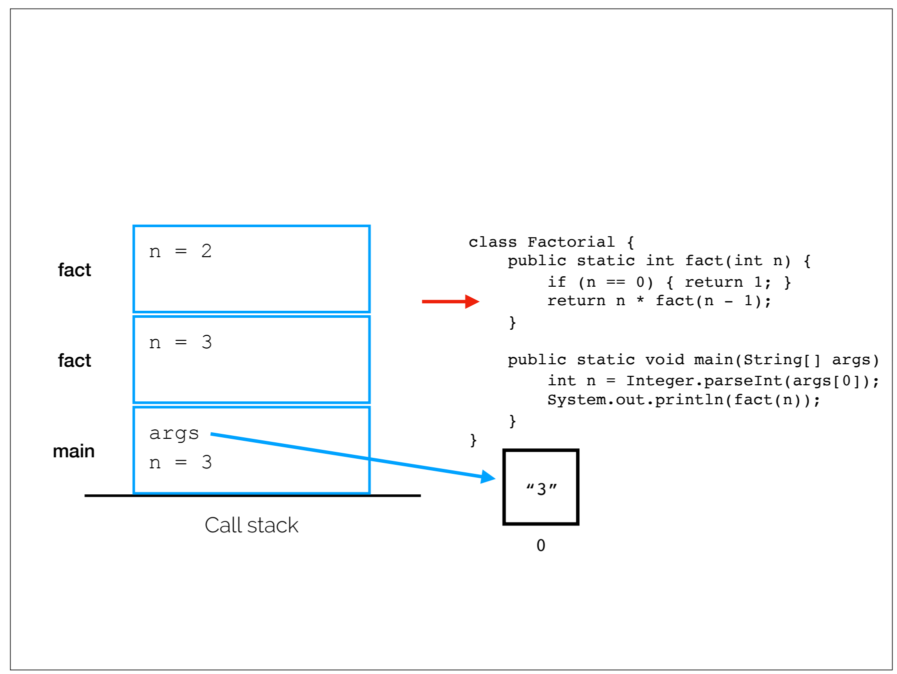

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

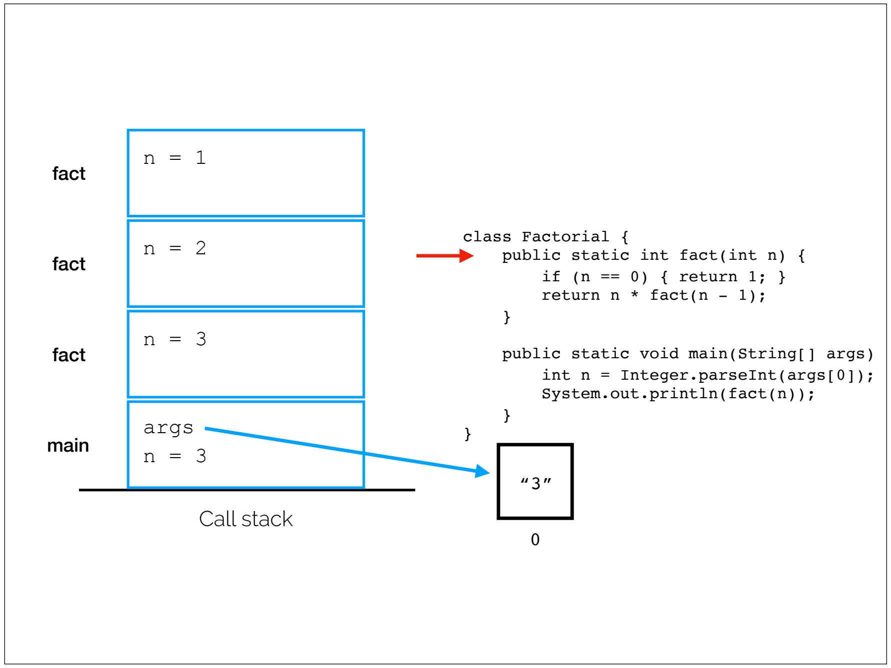


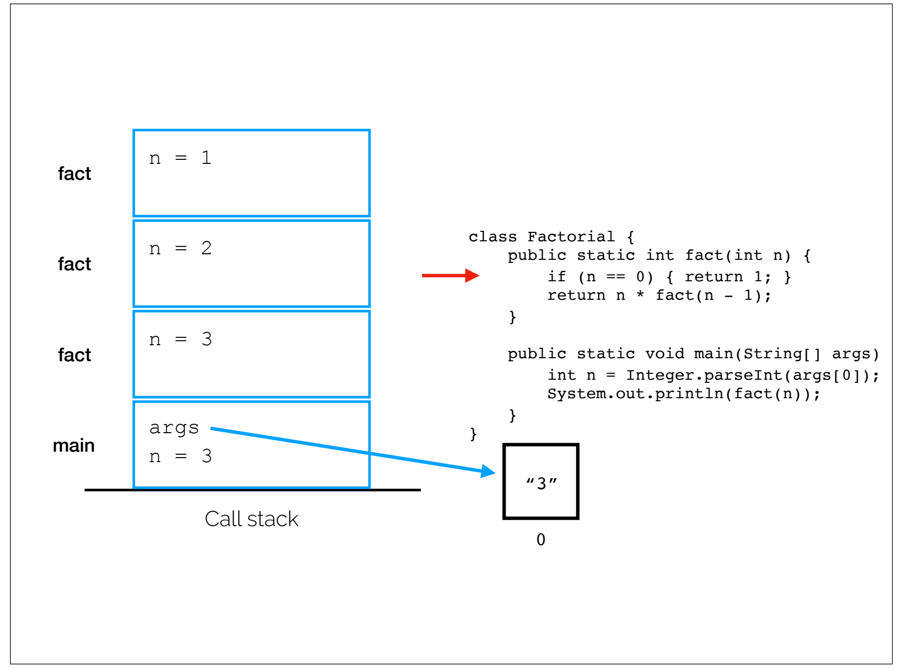

fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

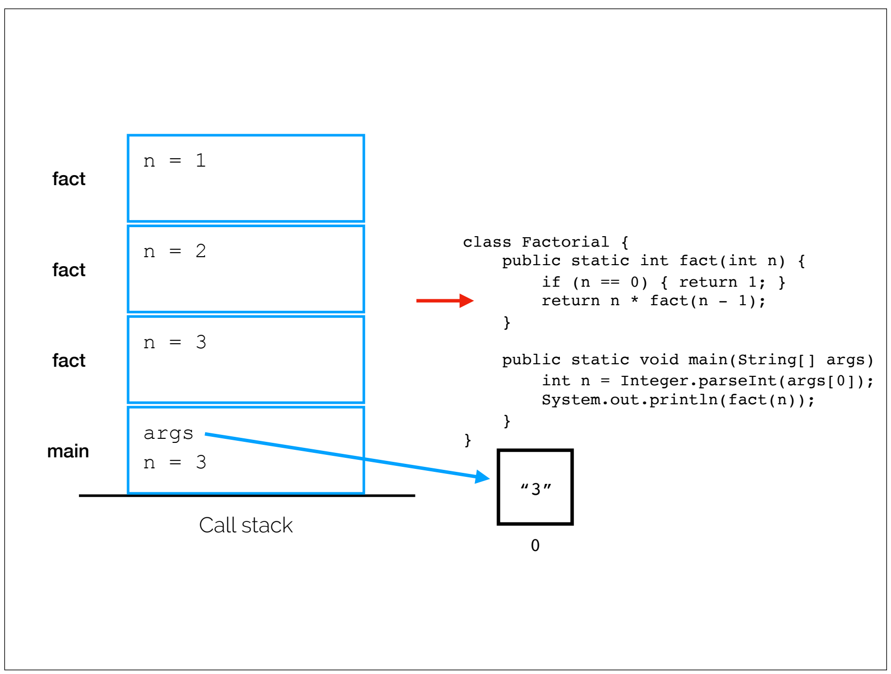

fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

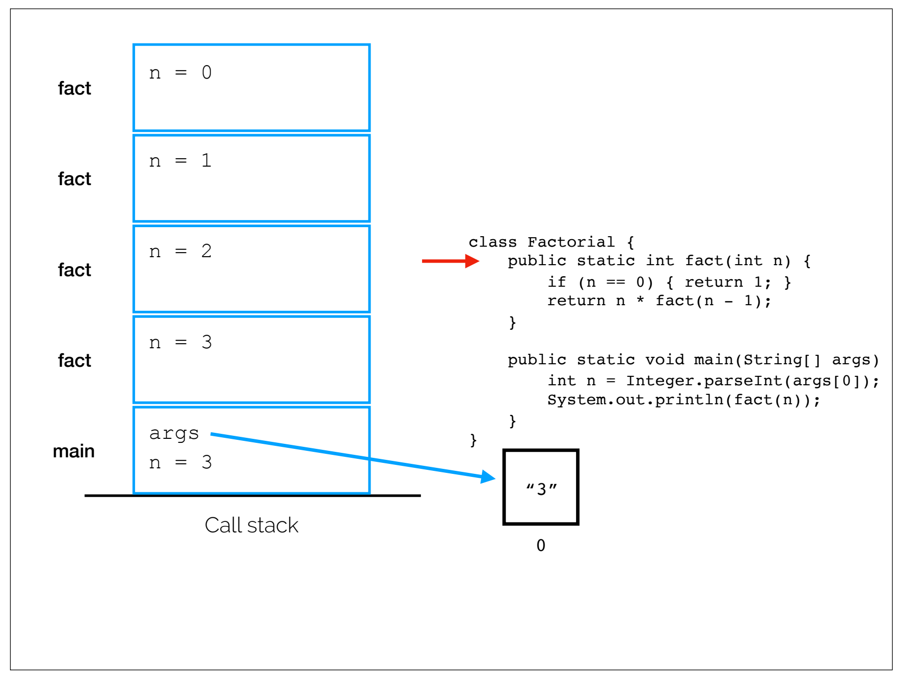

fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

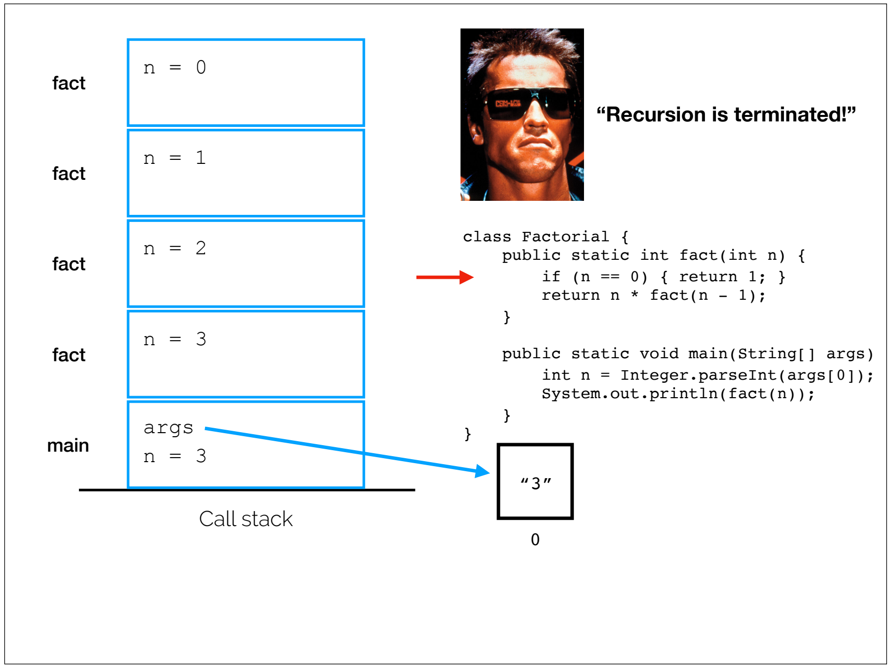

fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

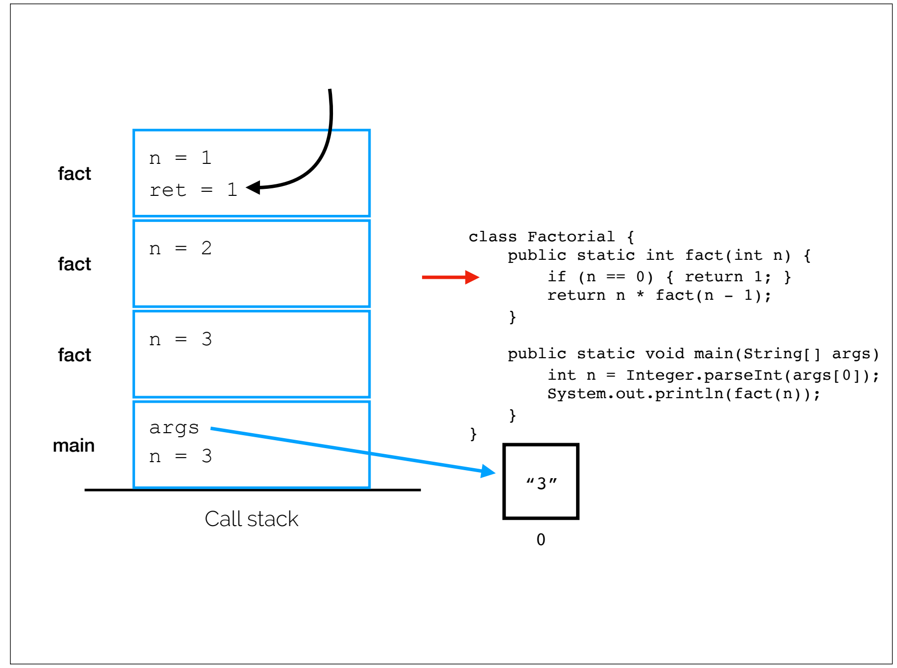

fact
n = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}
"3"
0

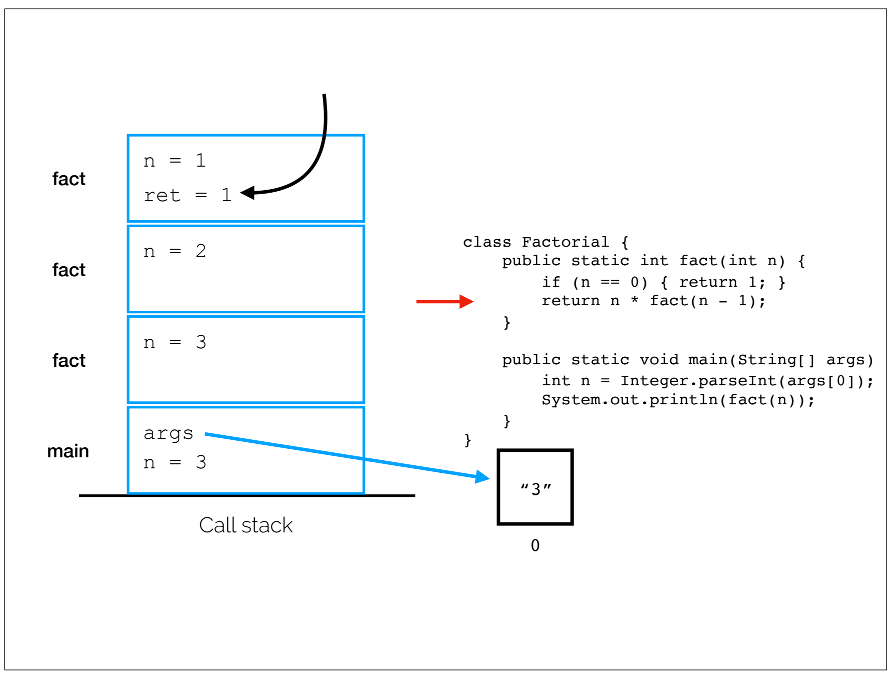

fact
n = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}
"3"
0

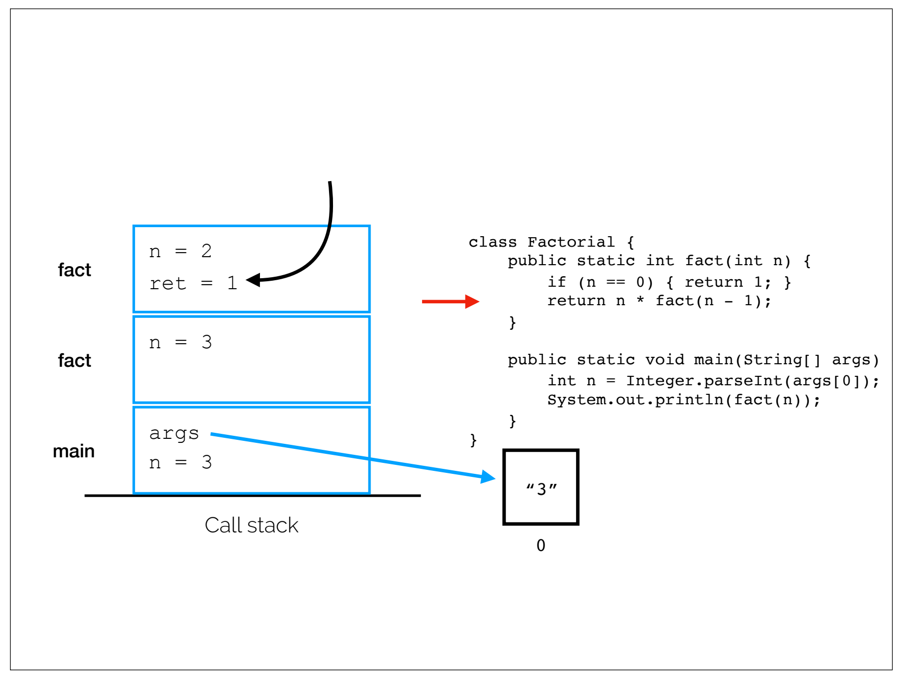

fact
n = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}
"3"
0

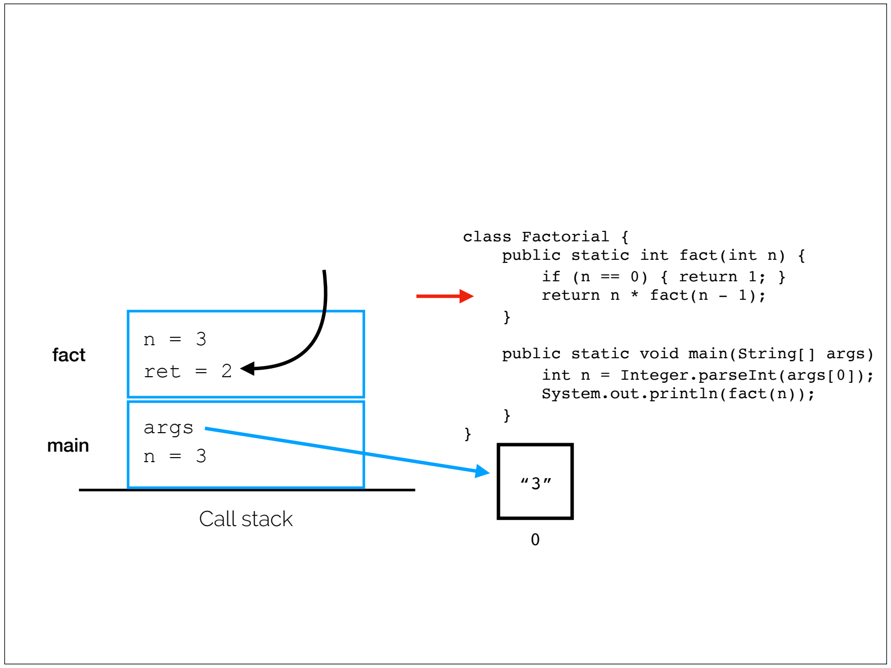

fact
n = 0
fact
n = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}
"3"
0

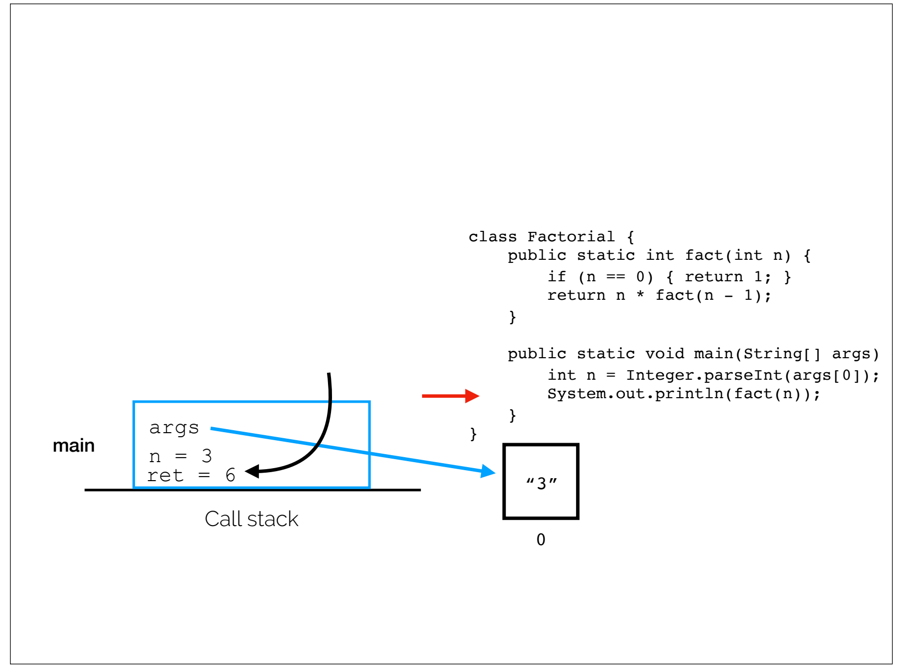

fact
n = 0
fact
n = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
"Recursion is terminated!"
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

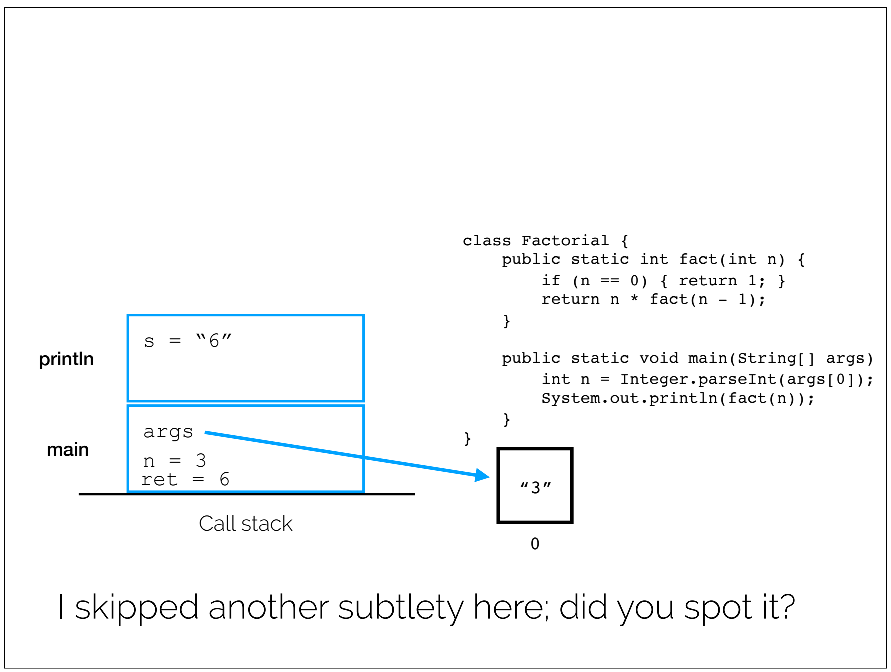

fact
n = 1
ret = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

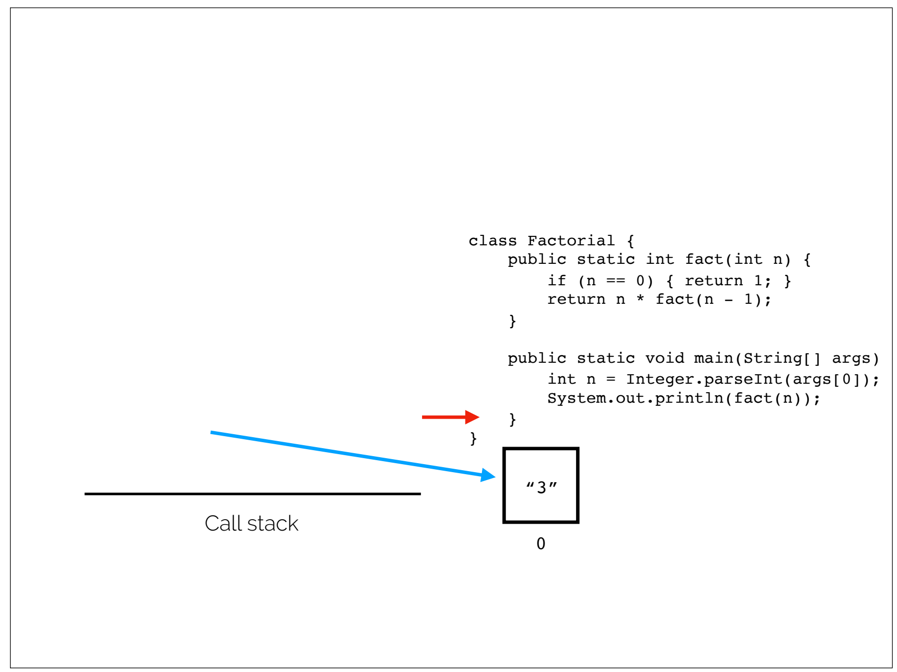

fact
n = 1
ret = 1
fact
n = 2
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}


fact
n = 2
ret = 1
fact
n = 3
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}


fact
n = 3
ret = 2
main
args
n = 3
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}


main
args
n = 3
ret = 6
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}


println
s = "6"
main
args
n = 3
ret = 6
Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}
I skipped another subtlety here; did you spot it?


Call stack
"3"
0
class Factorial {
public static int fact(int n) {
if (n == 0) { return 1; }
return n * fact(n - 1);
}
public static void main(String[] args)
int n = Integer.parseInt(args[0]);
System.out.println(fact(n));
}
}

Recap & Next Class

Today we learned:

Pre/post conditions

Recursion

Recursion activity

Recursion tradeoffs

Next class:

Mathematical induction