## Executing Method Calls

This lecture tells you precisely how method calls are executed (a few details will have to wait until we get to classes and objects). The rules we give work automatically for recursive methods.

YOU MUST MEMORIZE THE RULES FOR EXECUTING A METHOD CALL AND BE ABLE TO EXECUTE A METHOD CALL YOURSELF.

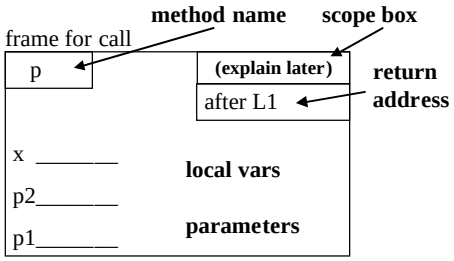
**Readings:**
Weiss, Section 7.3.3, page 241-242, discusses this topic briefly but doesn't say enough.

1

## Frame for a call (also called "activation record")

```
public static void p(int p1, int p2) {
    int x;
    x= p1;
    ...
}
```

**procedure call:** L1: p (2+5, 9-1);




2

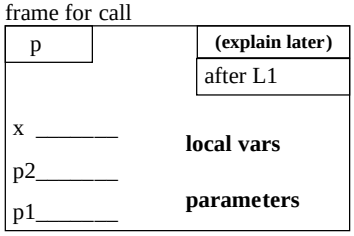
**Frame for a call**

```
public static void p(int p1, int p2) {
    int  x;
    x=  p1;
    ...
}
```

**procedure call:** L1: p (2+5, 9-1);

When method body is being executed, look in frame for local variables and parameters.

frame for call

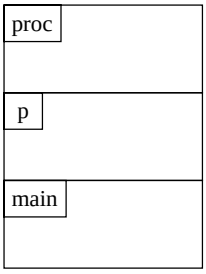



3

**Frames are placed on the call stack and removed when call is finished**

```
public static void p(int p1, int p2) {
    int x= 5;
    L2: proc(x, p1+p2);
    ...
}
```

```
public static void main (String[] pars) {
    L1: p (2+5, 9-1);
}
```




**stack:** a list with two operations:
(1) push an element onto it;
(2) pop an element off it.

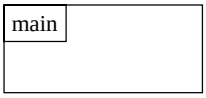
**call stack**

4

**Execution of procedure call**

L1: p (2+5, 9-1);

**1.** Evaluate the arguments and push them onto the call stack.
**2.** Push locations for the rest of the frame for the call onto the stack.
**3.** Put in frame: name of method, local variables, return address, and scope box (note that the arguments have already been assigned to the parameters).
**4.** Execute method body --look in frame at top of stack for all variables.
**5.** Pop the frame from the stack; continue executing at the return address in popped frame.




**call stack**

5

**Execution of procedure call**

**1.** Evaluate the arguments and push them onto the call stack.
**2.** Push locations for the rest of the frame for the call onto the call stack.
**3.** Put in frame: method name, local variables, return address, and scope box (arguments have already been assigned to parameters).
**4.** Execute method body --look in frame at top of stack for all variables.
**5.** Pop the frame from the stack; continue executing at the return address in popped frame.

**Five-minute quiz on Tuesday, 11 Sept.**
**You will have to write this sequence of instructions and follow it in executing a method call.**
**Everyone should get 100 on the quiz!**
**Memorize this sequence of instructions!!**
**Practice executing the sequence yourself!!!**

6

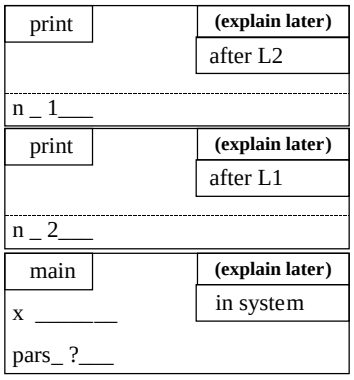
**Executing some procedure calls**

```
public class Example {
    public static void main (String[] pars) {
        L1: print(2);
    }

   // Print integers 0..n in reverse order
   // Precondition: 0 <= n
   public static int print(int n) {
       if (n == 0) {
           System.out.println(n);
           return;
       }
       // {n > 0}
      System.out.println(n);
      L2: print(n-1);
   }
}
```

We'll execute this program on the blackboard

7

**Snapshot of call stack just before executing method body of print for the second time**




8

**Evaluation of a function call** max(5,3)

**Consider executing**

**int** b= max(3,5) + max(4,6);

**Two points:**

**(0)** A call like max(3,5) yields a value, which is used in place of the call. We have to change our execution rules to take this into account.

**(1)** This statement has TWO calls in it, so we have to revise our notion of a "return address". It's not always the next statement. We won't deal with this in detail but will just assume we understand how to do it.

9

**Evaluation of a function call** max(5,3)

**1.** Evaluate the arguments and push them onto the call stack.
**2.** Push locations for the rest of the frame for the call onto the stack.
**3.** Put in frame: name of method, local variables, return address, and scope box (note that the arguments have already been assigned to the parameters).
**4.** Execute method body --look in frame at top of stack for all variables, **until a statement return e; is to be executed**.
**5. Evaluate e; replace the frame on the top of the stack by the value of e**; continue executing at the return address in popped frame.

**(the value at the top of the stack will be used as the value of the function call and will be popped from stack when used)**

10

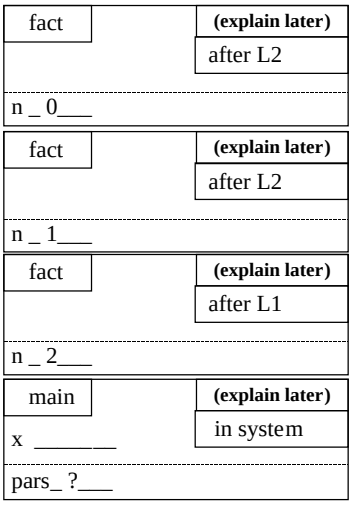
**Execute Some Calls**

```
public class Example {
    // Test method fact
    public static void main (String[ ] pars) {
        L1: int x=  fact(2);
    }

    // = !n  (for n >= 0)
    public static int fact(int n) {
        if (n == 0) {
          return 1;
        }
        // {n > 0}
        return n * /* L2: */ fact(n-1);
    }

}
```

**We'll execute this program on the blackboard.**

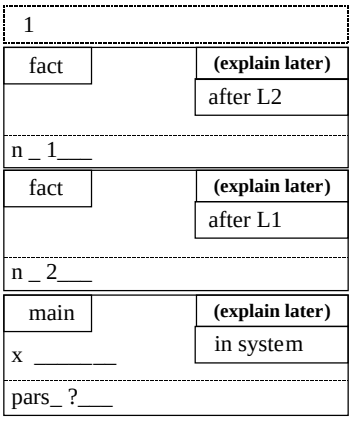
11

**Snapshot of call stack just before executing the body of fact for the third time**



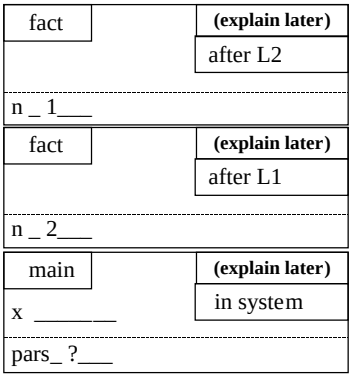

12

**Snapshot of call stack just after the call fact(n-1) with n = 1 is finished**




13

**Snapshot of call stack just after evaluation of n * fact(n-1) with n = 1 is finished**




14