CS206

Midterm

Recursion

Midterm

- Q1 — reading code
  - 16 pts — mean 12.3
- Q2 — Weird Merge
  - 24 pts — mean 20.5
- Q3 — Big-O
  - 16 pts — mean 10
- Q4 —Code reading
  - 20 pts — mean 14.8
- Q5 — queue reversal
  - 24 pts — mean 17.2
- Overall mean 75

Q1

```
A1 firstA = new A1();
firstA.setVar(8);
A1 secondA = firstA;
secondA.setVar(42);
```

```
public D1(int num) {
    aa.add(0);
    for (int ii=1; ii<num; ii++) {
        aa.add(aa.get(ii-1)+ii);
    }
}
public String toString() {
    if (aa.size()> 10)
        return ""+aa.get(6);
    else
        return "-1";
}
```

```
public C1(int nNums) {
        nums = new int[nNums];
        nums[nNums-1]=1;
        for (int i = nNums-2; i >=0;
                i--)
        nums[i] = nums[i+1]*2;
    }
    @Override
    public String toString() {
        return "(" + nums[0] + ", "
+nums[nums.length/2] + ", " +
+nums[nums.length-1] + ")";
    }
```

Q2

```
import java.util.ArrayList;
public class P2<M> implements OrderedMerge<M> {
    @Override
    public ArrayList<Comparable<M>> merge(Comparable<M>[] list1,
Comparable<M>[] list2) {
        ArrayList<Comparable<M>> ret = new ArrayList<>();
        for (Comparable<M> mi1 : list1) {
            ret.add(mi1);
            for (Comparable<M> mi2:list2) {
                if (mi1.compareTo((M)mi2)>0)
                    ret.add(mi2);
            }
        }
        return ret;
    }}
```

Q3

```
public class Complexx {
    public long part1(int[] arra) {
        long res=1;
        for (int i=0; i<4000000; i++) {
            if (i<arra.length)
                res=res+arra[i];
            else
                res = res +
                    (long)Math.sqrt(i);
        }
        return res;
    }
    public long part2(int[] arra) {
        long res = 0;
        for (int i=0;i<arra.length;i++)
            res += arra[i];
        return res;
    }
```

```
public long part3(int[] arra) {
    long res=0;
    for (int i=1;
     i<arra.length; i=i*2)
        res += arra[i];
    return res;
}
public long part4(int[] arra) {
    long res=0;
    for (int i=0; i<arra.length;
        i++)
        res = res + part2(arra);
    return res;
}}
```

Q4

```
public void rotateList(int n) {
if (head==null || head.next==null) {
    return;
}
        for (int i=0; i<n; i++) {
            Node<T> nod = head;
            head = nod.next;
            head.prev=null;
            tail.next = nod;
            nod.prev=tail;
            nod.next=null;
            tail = nod;
        }
    }
```

Rotate the elements of a linked list, putting the first element at the end. For example, if the linked list is [1,2,3,4] and n=3 then the linked list will become [4,1,2,3]

Q5

```
public ArrayQueue<Q> reverse(ArrayQueue<Q> aq)
    {
        ArrayQueue<Q> revQ = new ArrayQueue<>();
        ArrayStack<Q> stk = new ArrayStack<>();
        ArrayStack<Q> stk2 = new ArrayStack<>();
        Q temp;
        while (null != (temp = aq.poll())) {
            stk.push(temp);
        }
        while (null != (temp=stk.pop())) {
            revQ.add(temp);
            stk2.push(temp);
        }
        while (null != (temp=stk2.pop())) {
            aq.add(temp);
        }
        return revQ;
    }
```

Jumping Jen and Jill

Suppose Jen gets a the following jumps from Jumper.
1, 2, 3, -1, -1, -1, -1, 4
Then the actual distance she would jump would be
1, 2, 3, -3 (pop the stack), -2 (pop again), -1 (pop again), 0 (stack empty), 4
So her distance from the start would be
0, 1, 3, 6, 3, 1, 0, 0, 4

In assignment I suggested developing the link list based stack and queue first. You could do this last, using an array-based stack and queue. Get everything working and only then change the internal of the stack and queue.

Recursion

Any method that calls itself, either directly or indirectly

Idea, take a problem,
break that problem down into a slightly simpler problem,
ask yourself to solve that slightly simpler problem,
repeat

The Factorial

- Recursive definition: $f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$

- Java method

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

Recursive Method

- Base case(s):
  - no recursive calls are performed
  - every chain of recursive calls must reach a base case eventually
- Recursive calls:
  - Calls to the same method in a way that progress is made towards a base case

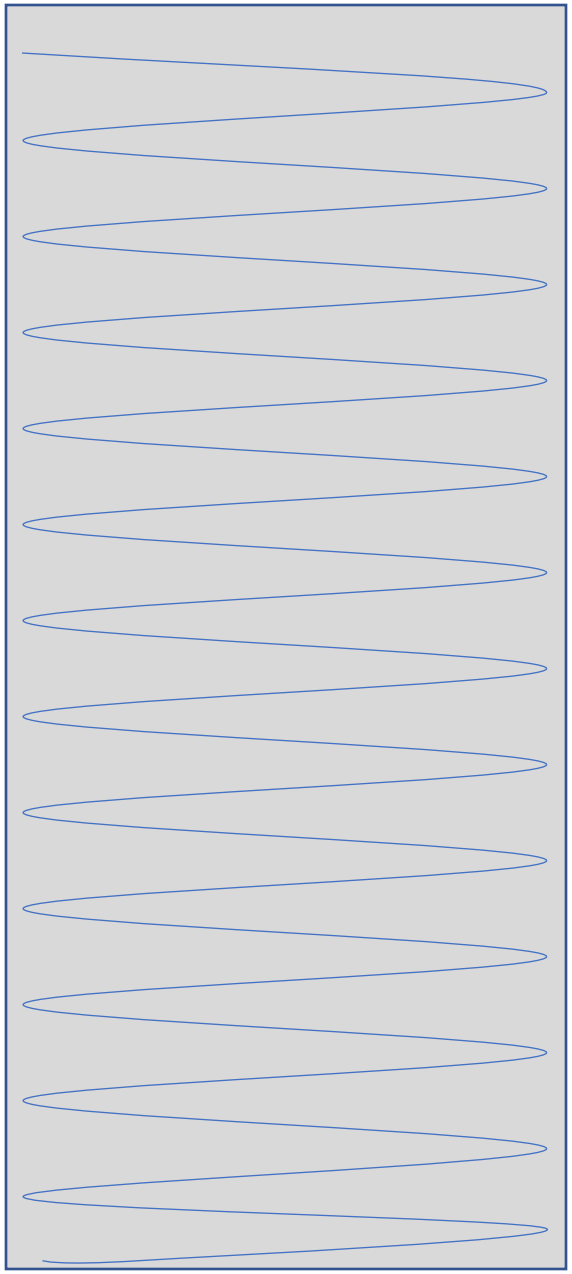
Compiled Code

```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

Call Stack



Compiled Code

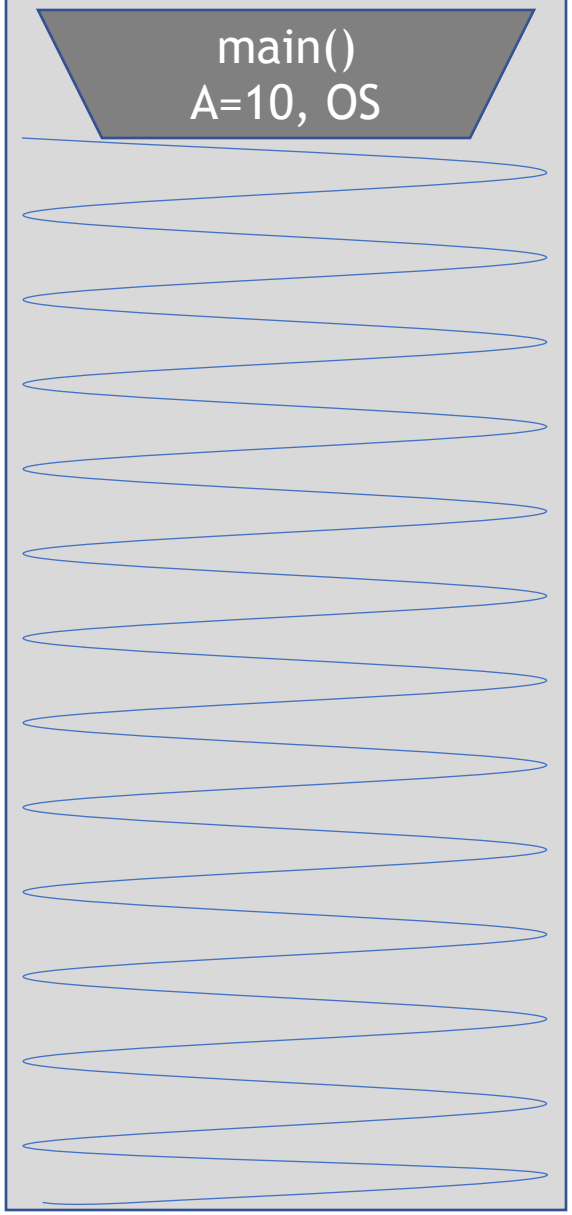
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

Call Stack



Compiled Code

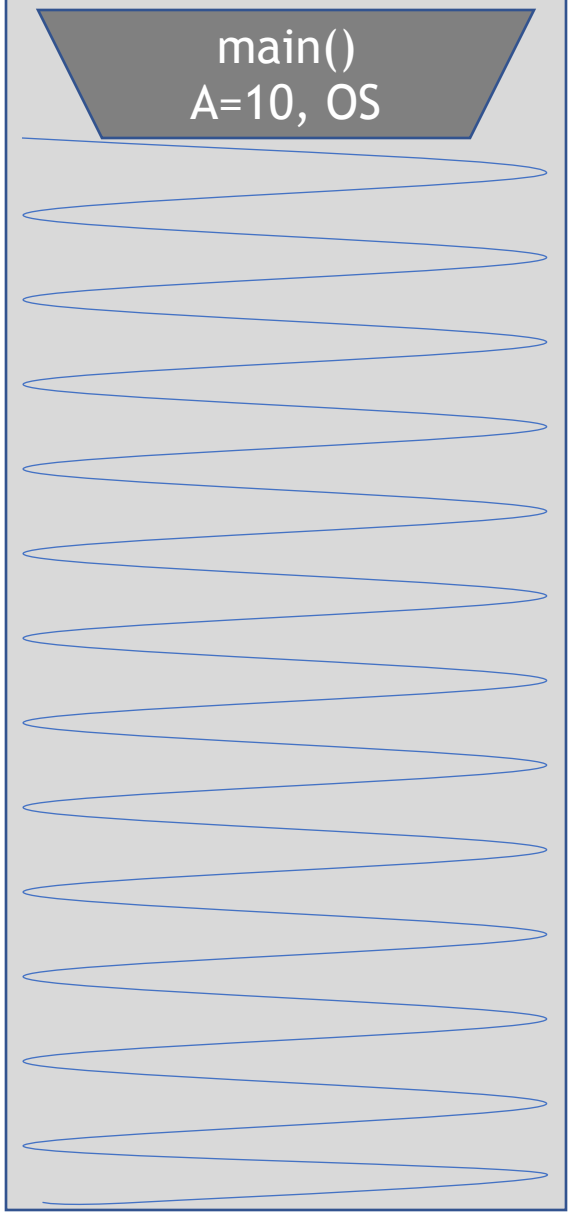
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

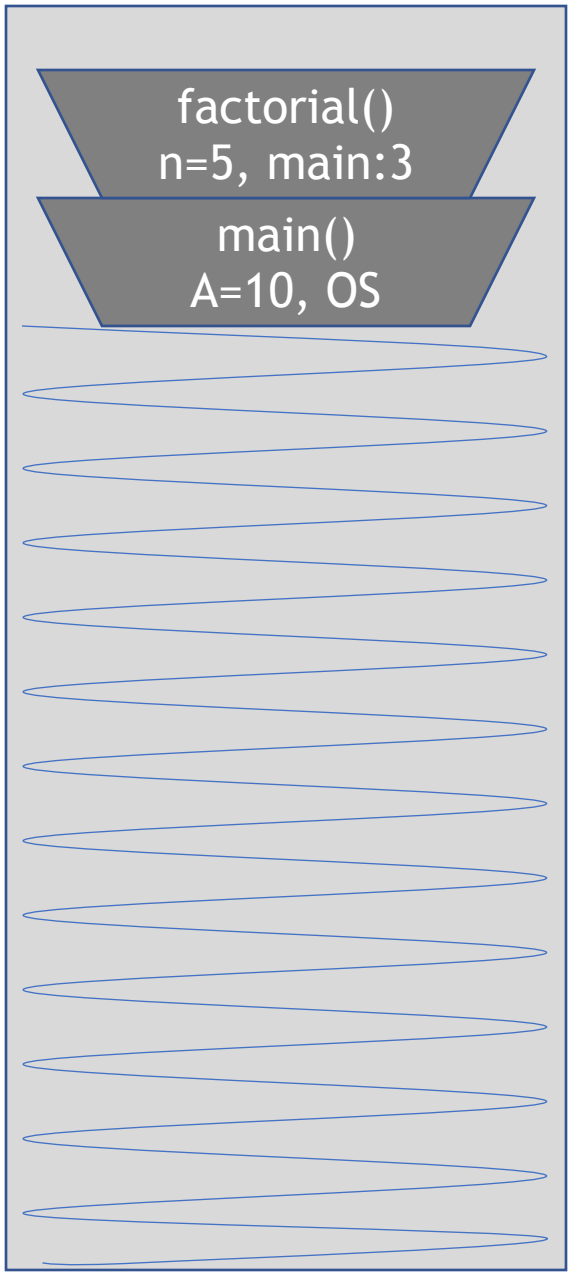
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

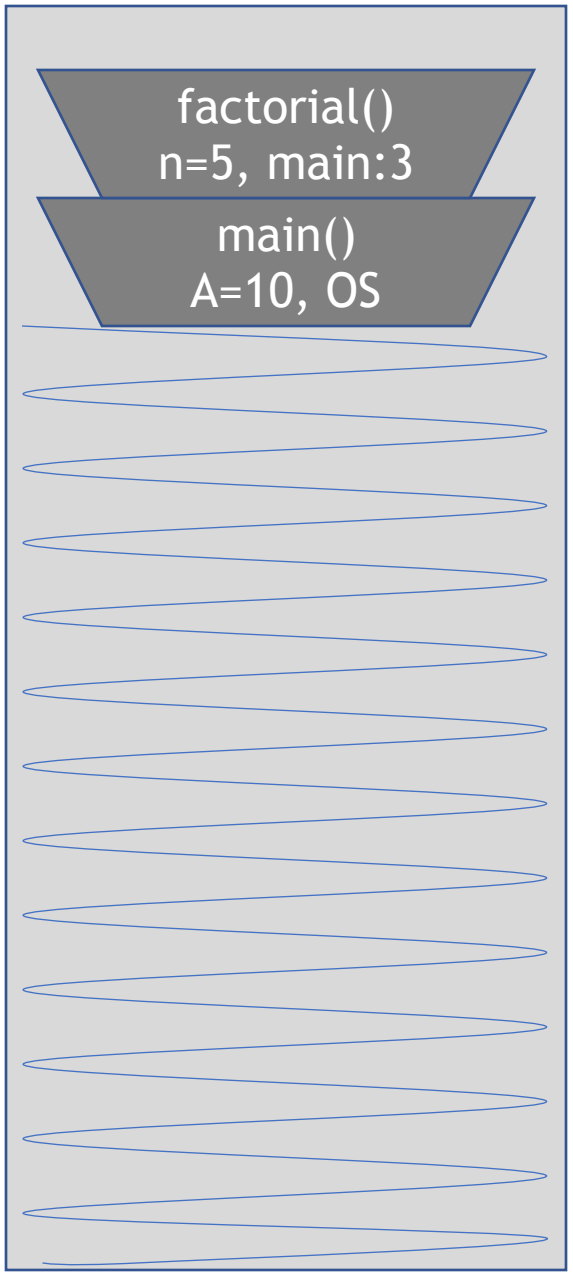
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

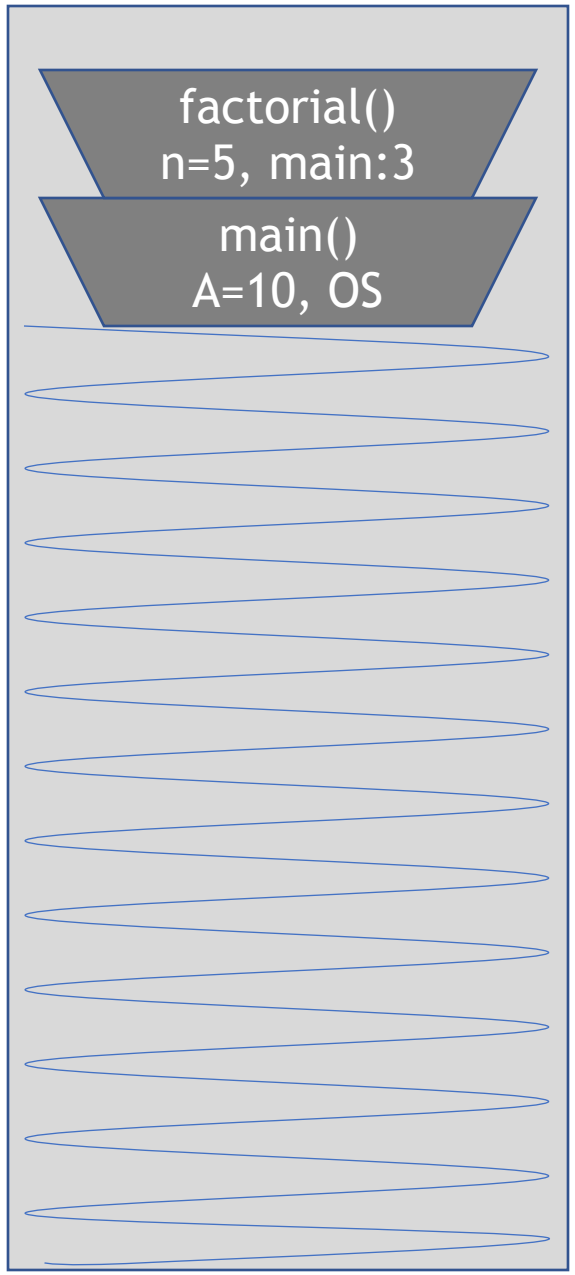
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

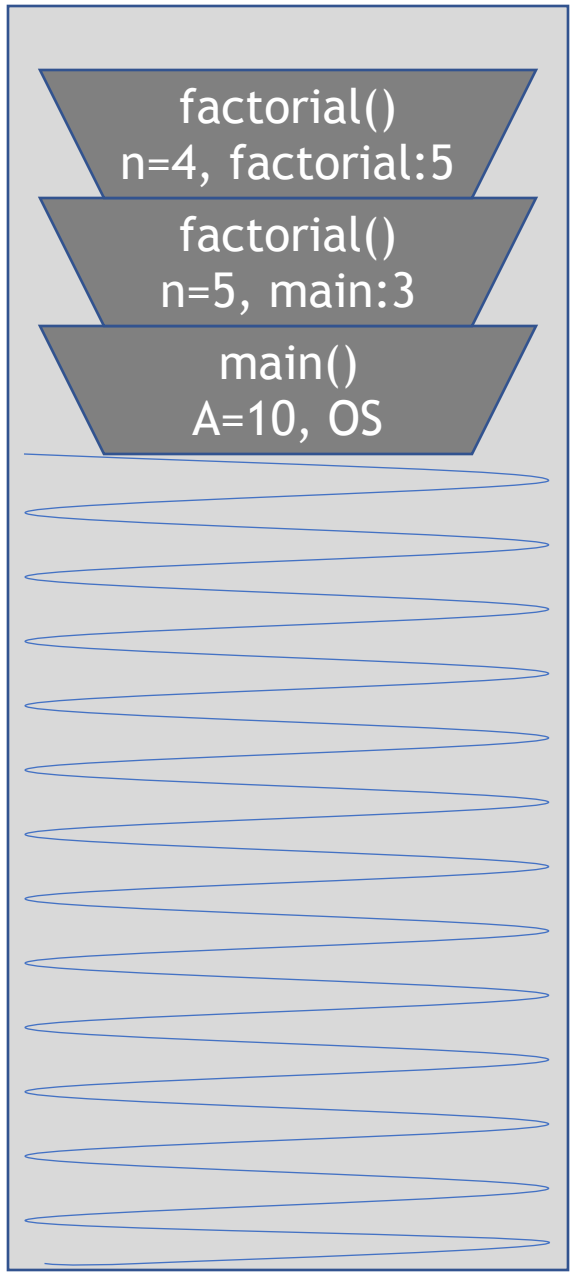
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

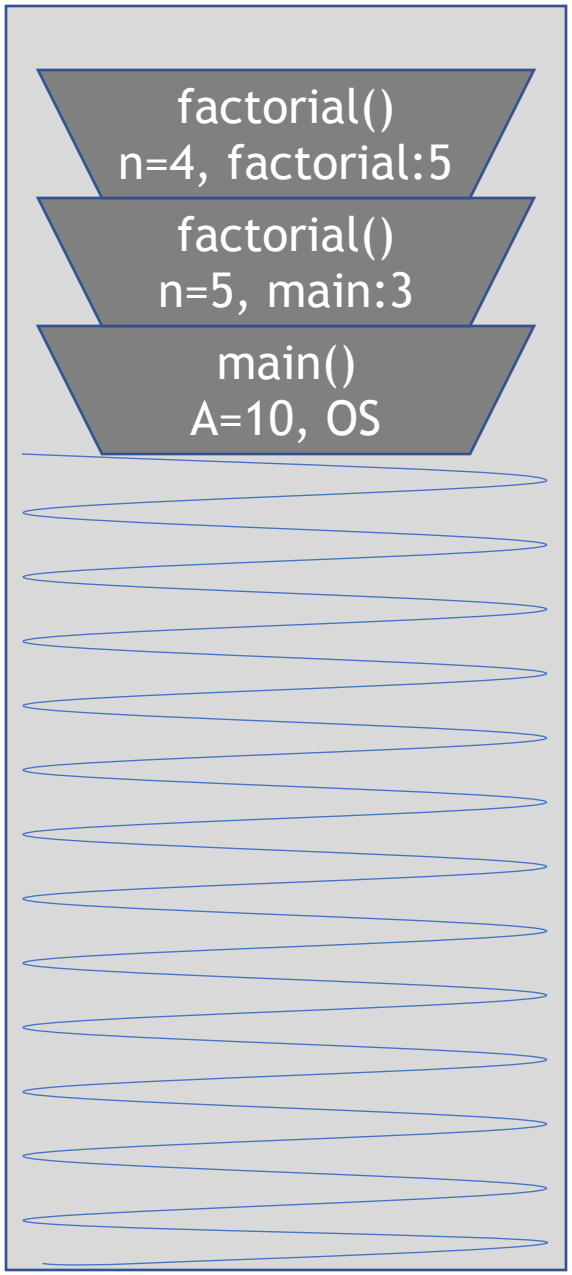
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
2.  int factorial(int n=4) {
3.    if (n == 1) {
4.      return 1;
5.    } else {
      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

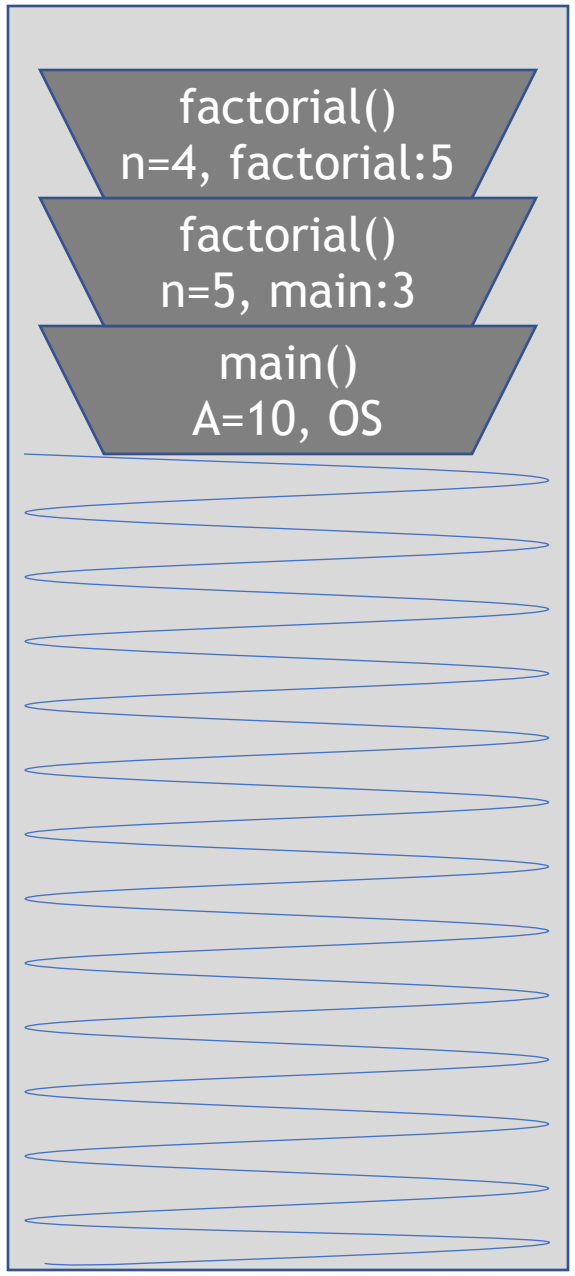
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

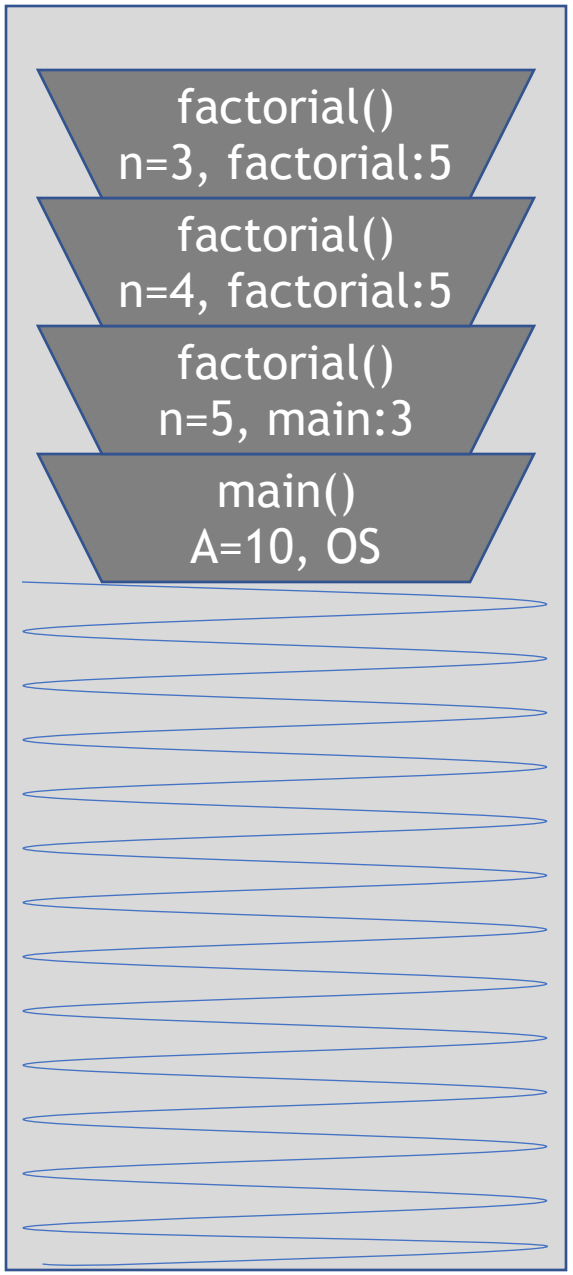
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

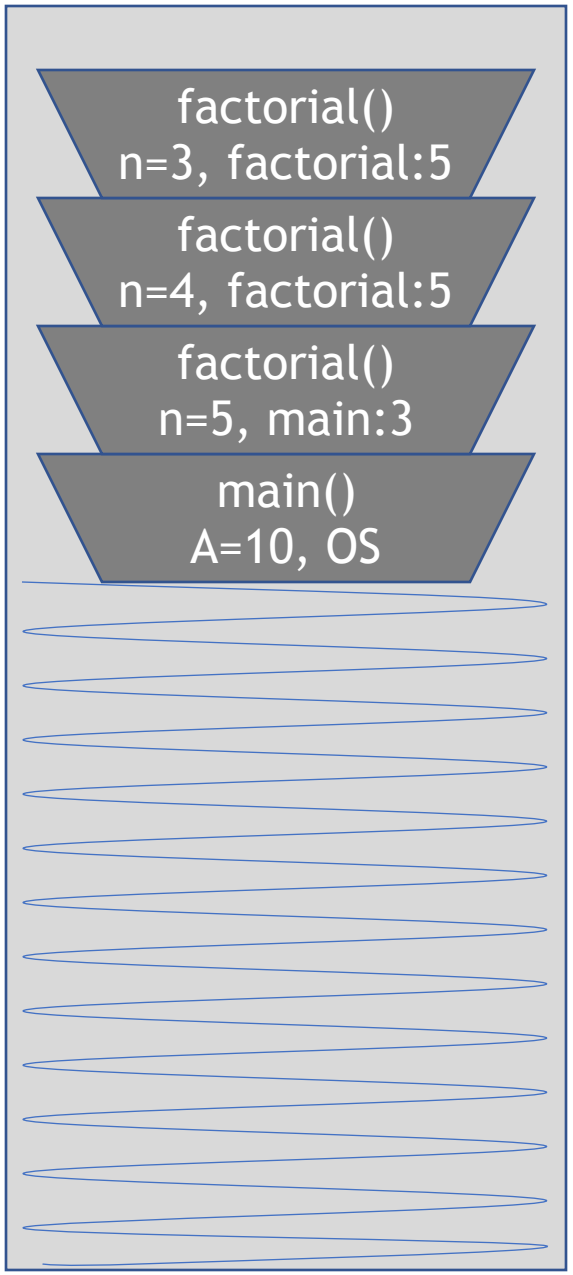
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=3) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Call Stack



Compiled Code

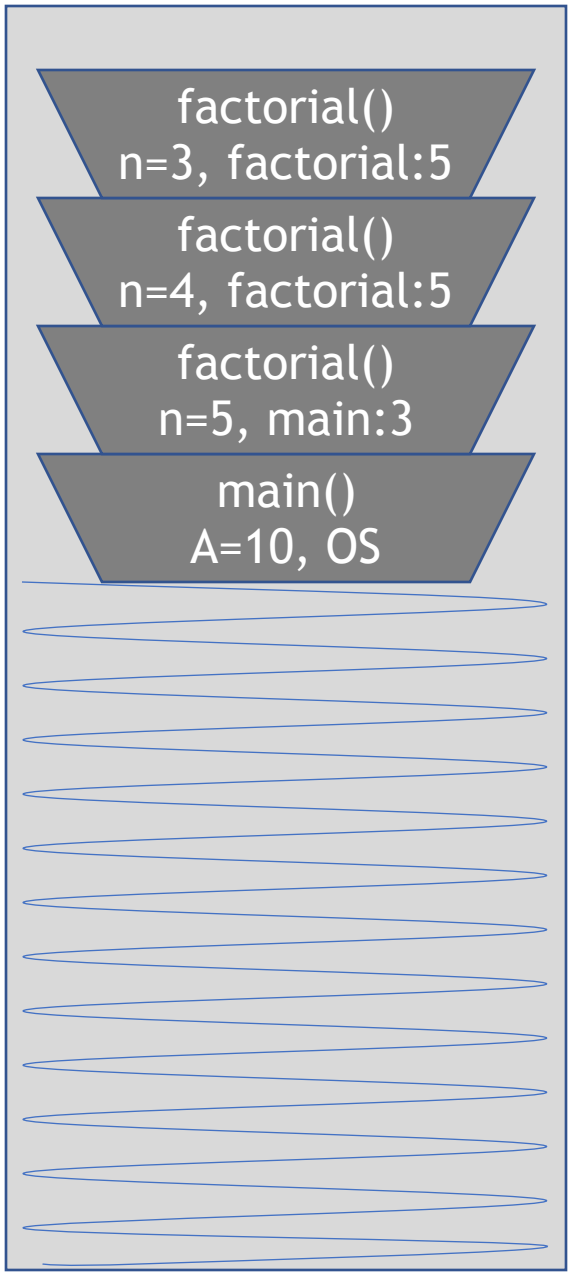
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

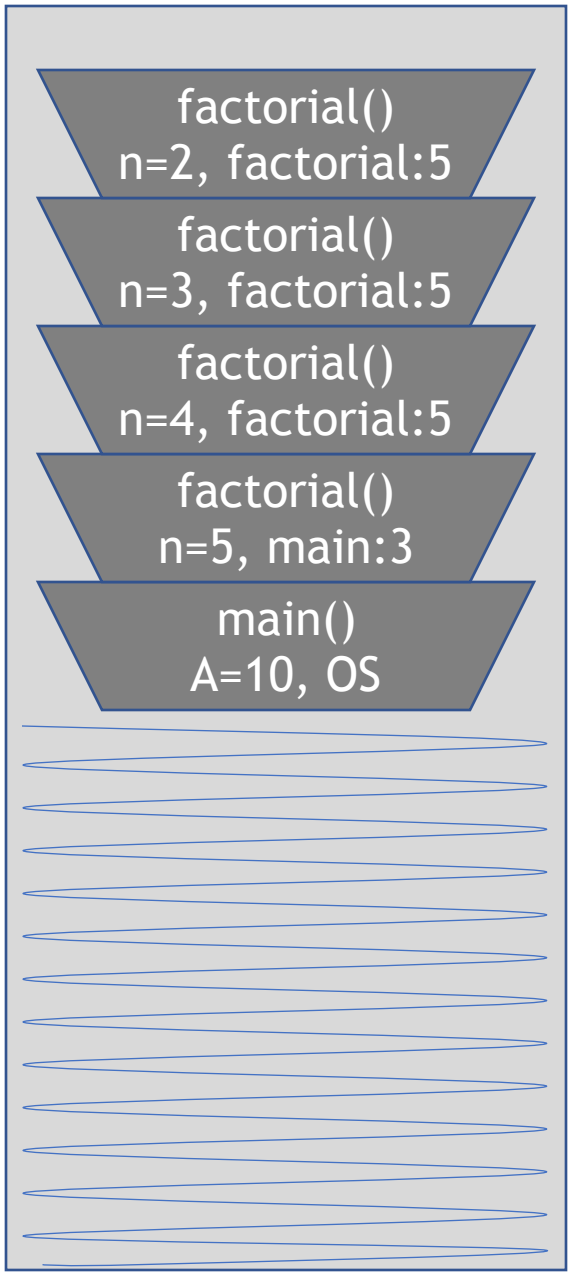
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

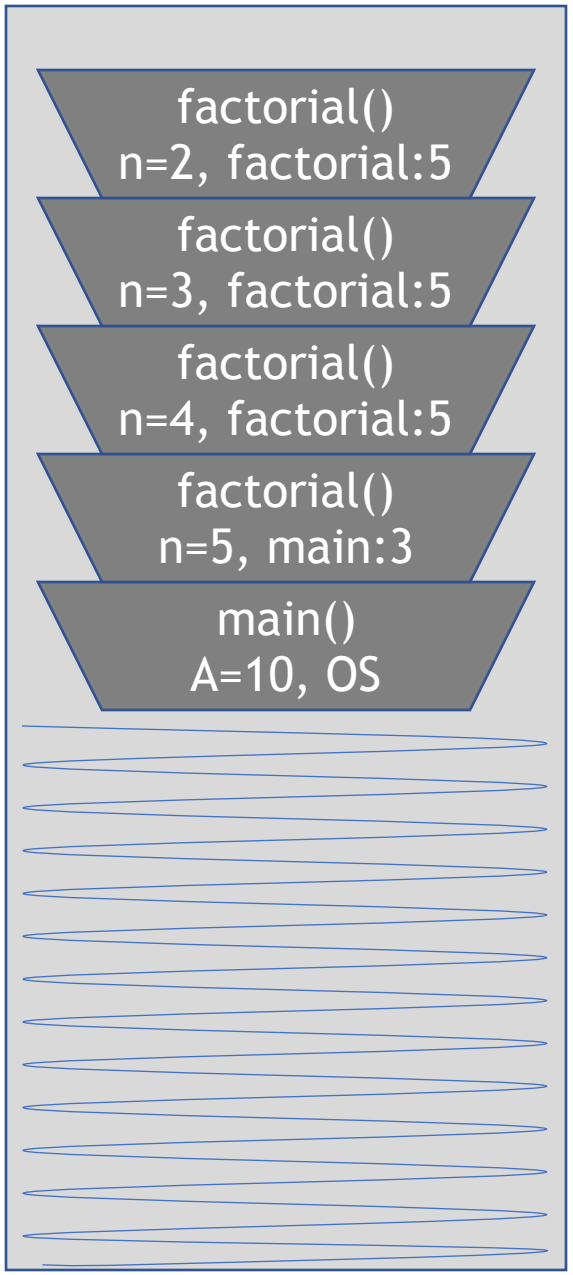
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

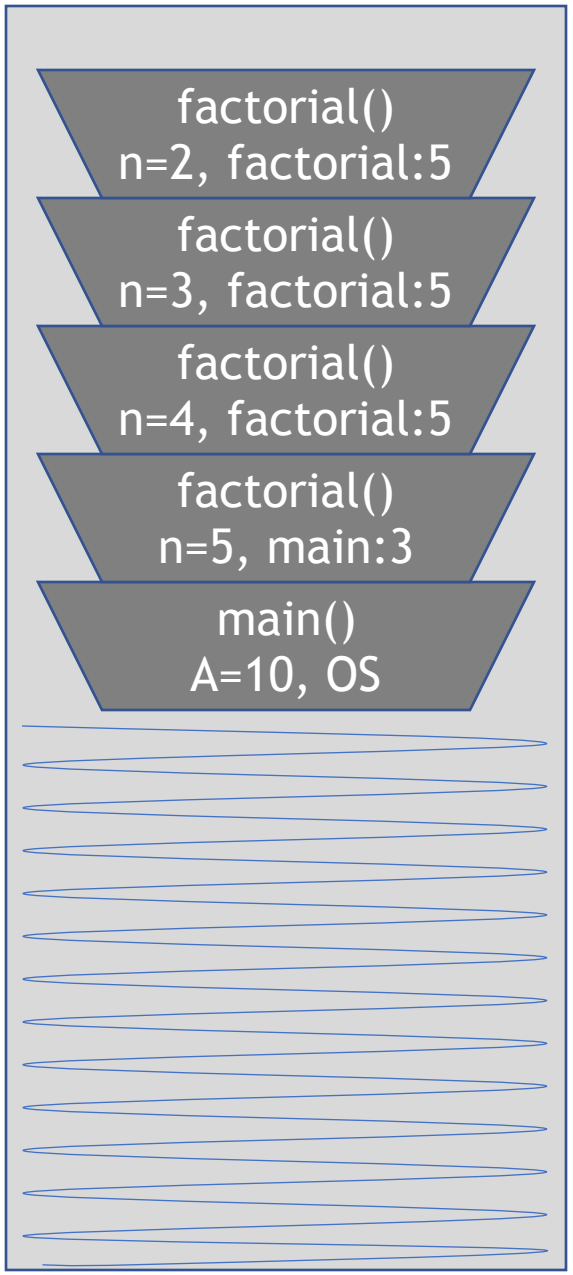
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



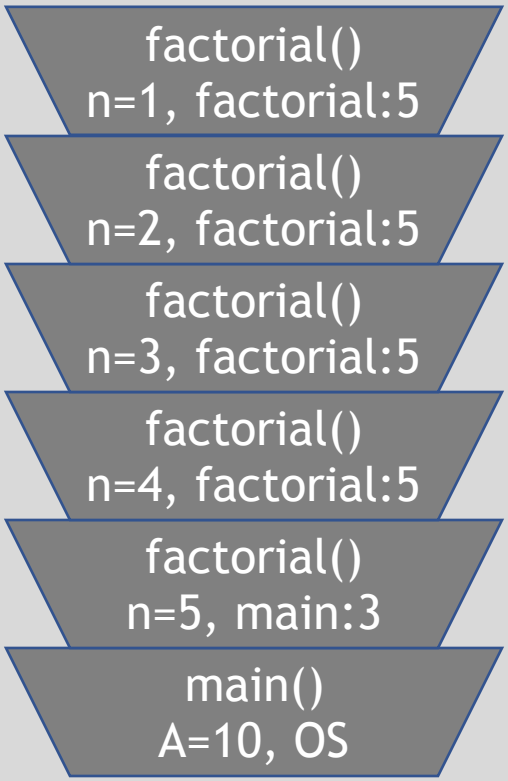
Compiled Code

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```



Compiled Code

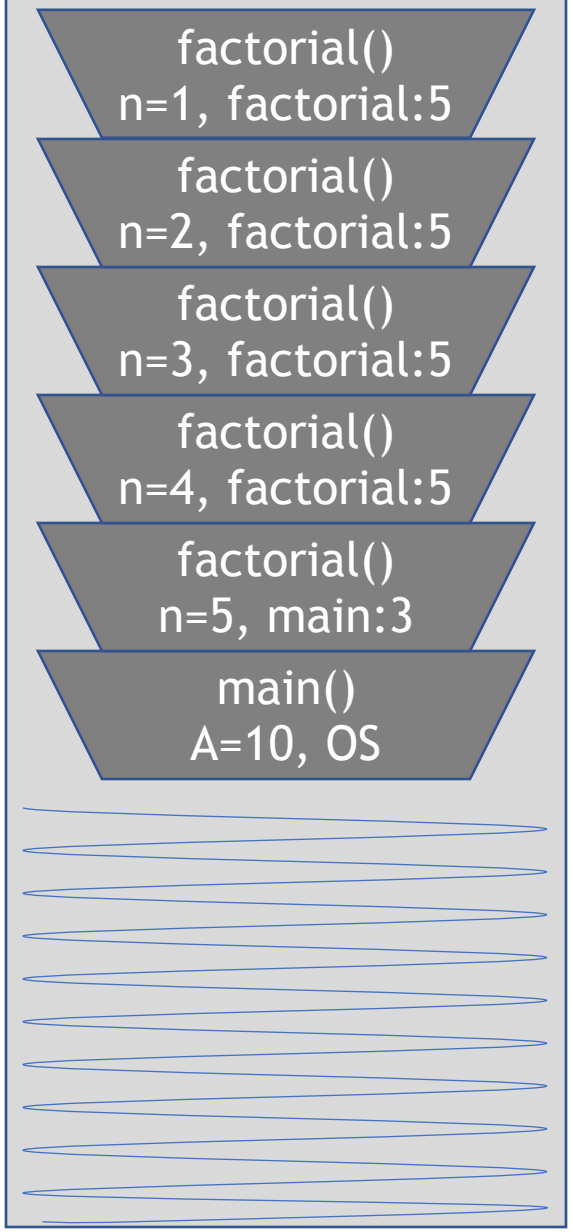
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int n=1) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Call Stack



Compiled Code

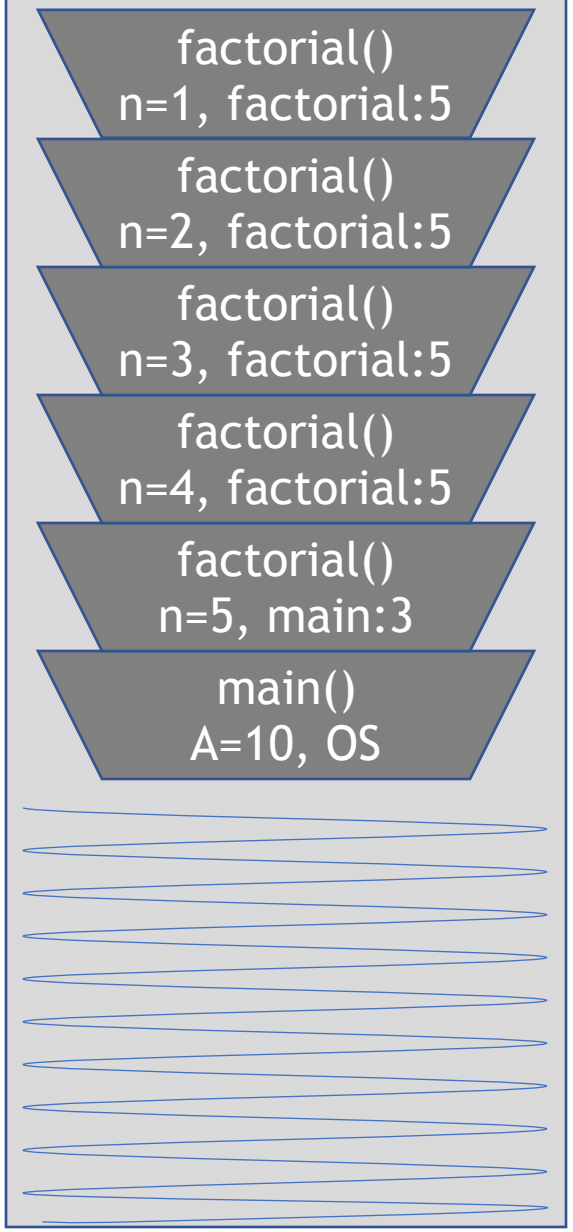
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=1) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

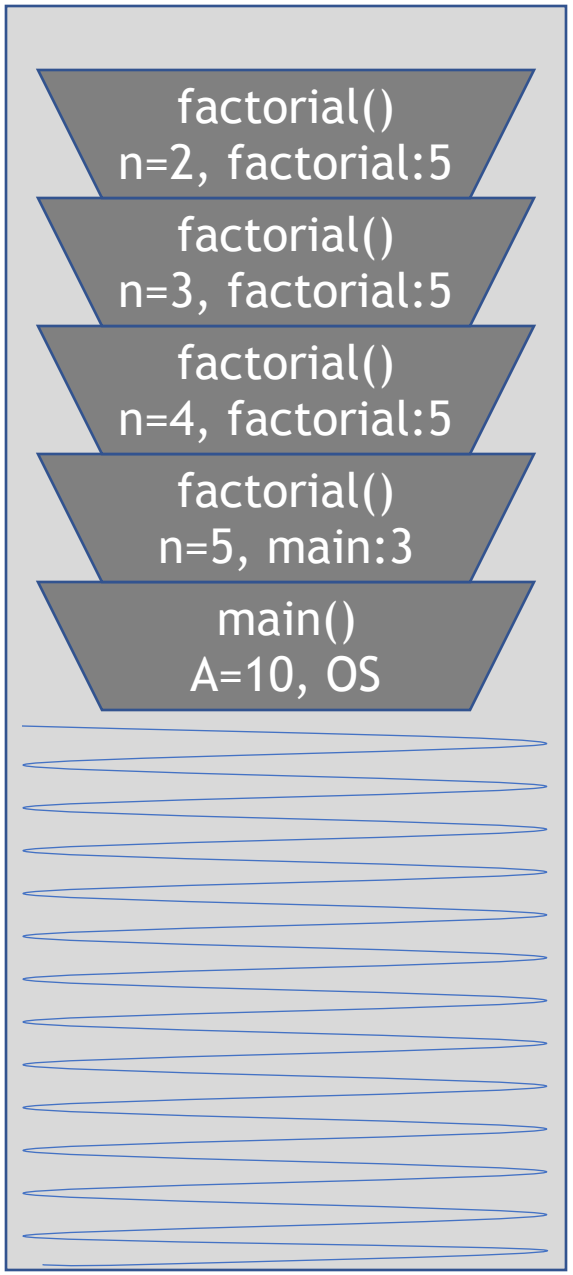
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 1;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

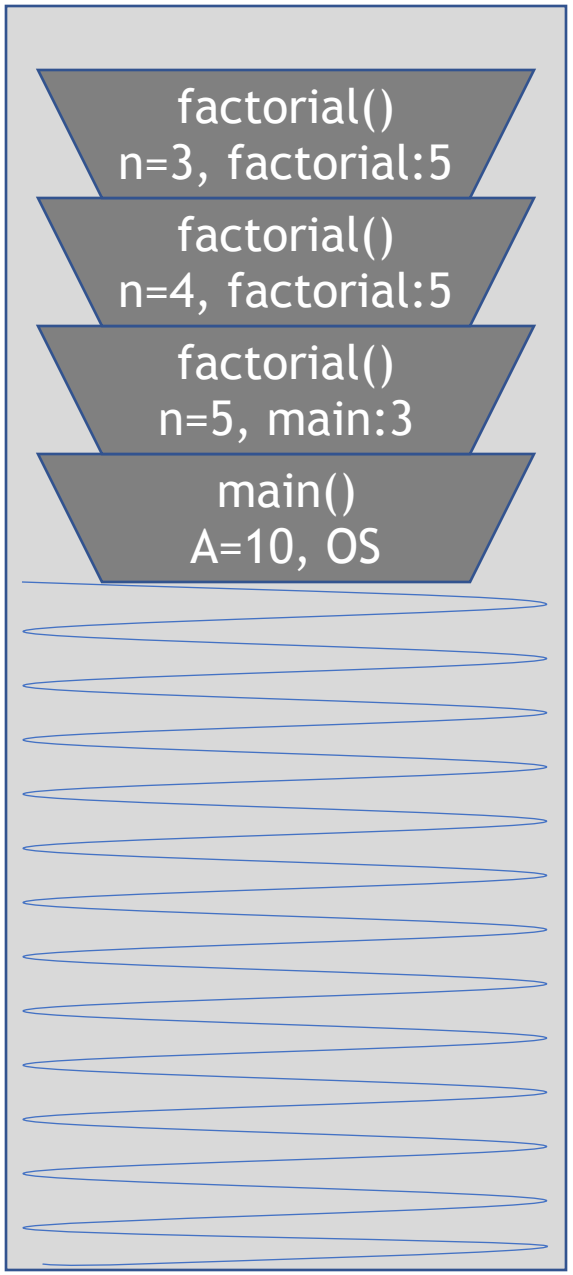
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 2;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

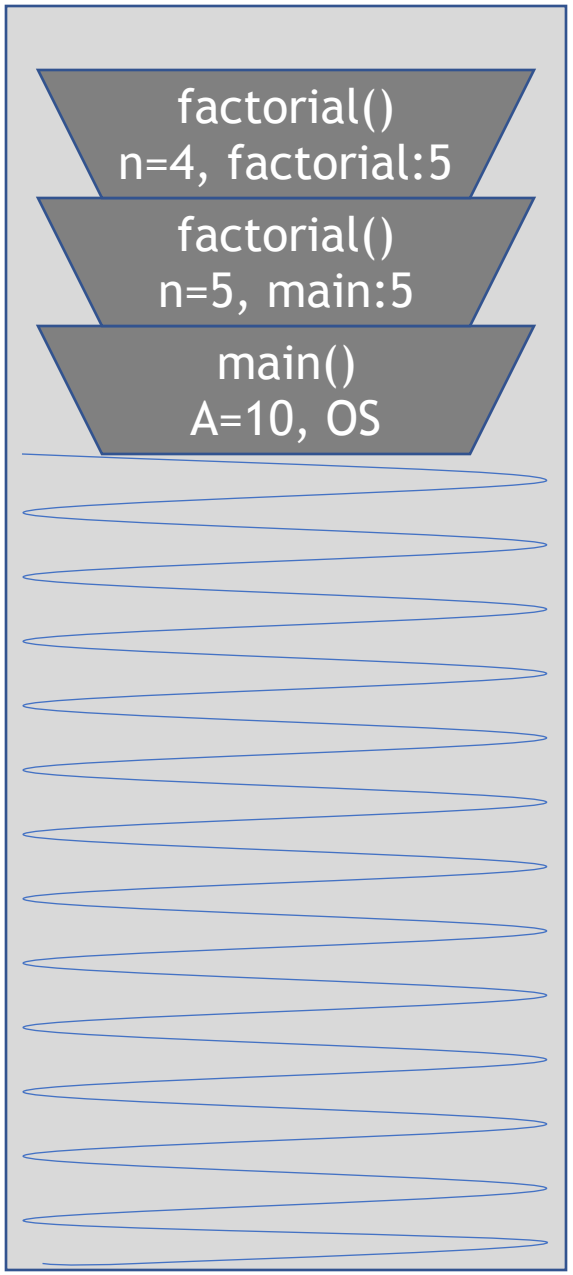
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 6;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

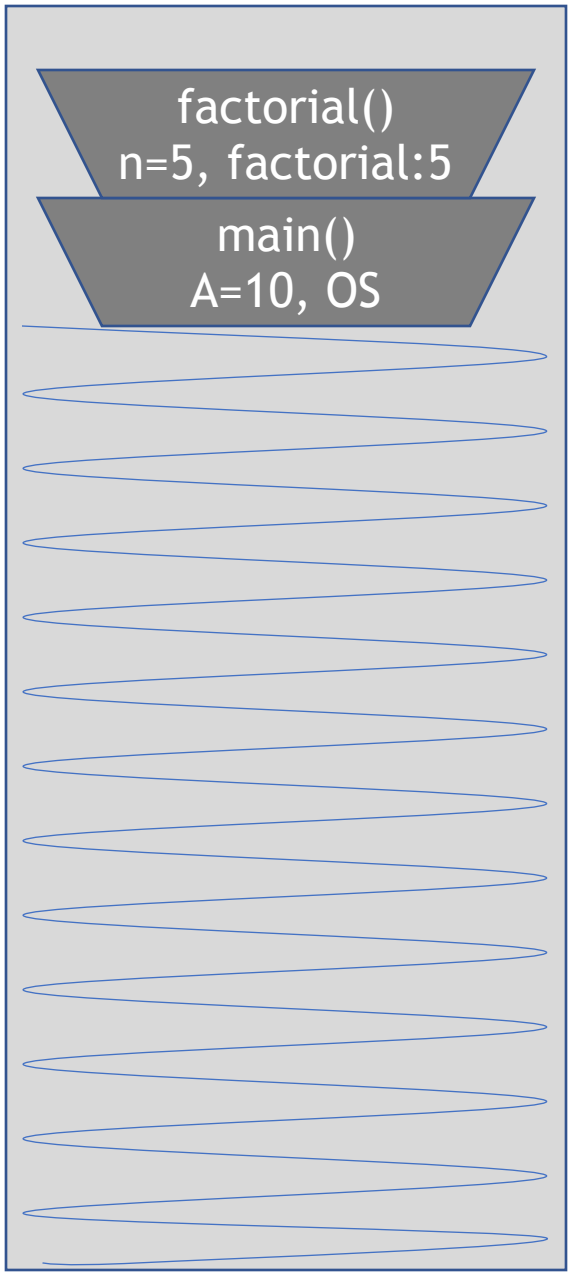
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 24;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

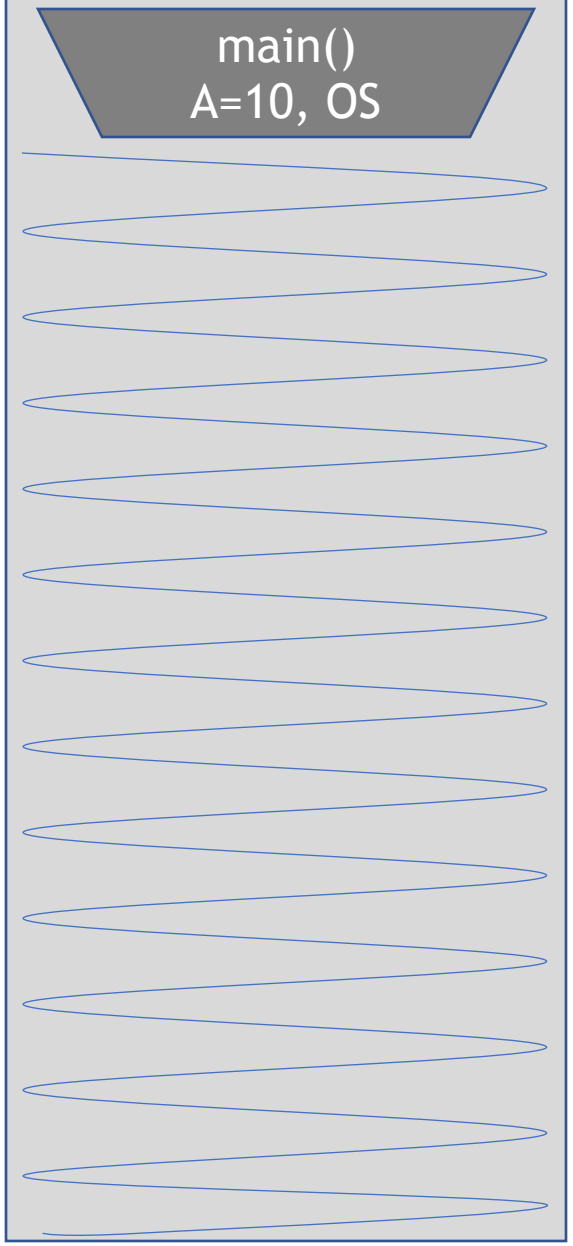
```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   void main() {
2.     int A = 10;
3.     int B = 120;
4.     System.out.println(B);
5.   }
```

Call Stack



Recursion

A method that calls itself, either directly or indirectly
Importantly, need a way to stop

```
public void badRecurse(int c)
{
  System.out.println("A" + c);
  badRecurse(c-1);
}
```

Class Recurser

```
public void goodRecurse(int c)
{
  System.out.println("B" + c);
  if (c<=0) return;
  goodRecurse(c-1);
}
```

Recursion — return values

```
/**
 * A recursive function to add two positive numbers
 * @param num1 one of the numbers
 * @param num2 another number
 * @return the sum of the two numbers
 */
public int rAdder(int num1, int num2) {
    if (num2<=0)
        return num1;
    return rAdder(num1+1, num2-1);
}
```

Recursion — returning values & private recursive functions

```
public BigInteger fibonacci(int n) {
        if (n<=0) return BigInteger.valueOf(0);
         if (n<3) return BigInteger.valueOf(1);
         return iFibonacci(BigInteger.valueOf(1), BigInteger.valueOf(1)
 n-2);
     }
 private BigInteger iFibonacci(BigInteger fibNumA, BigInteger
 fibNumB, int counter)
     {
         if (counter==1)
             return fibNumA.add(fibNumB);
         return iFibonacci(fibNumB, fibNumA.add(fibNumB),
 counter-1);
     }
```

recursion practice

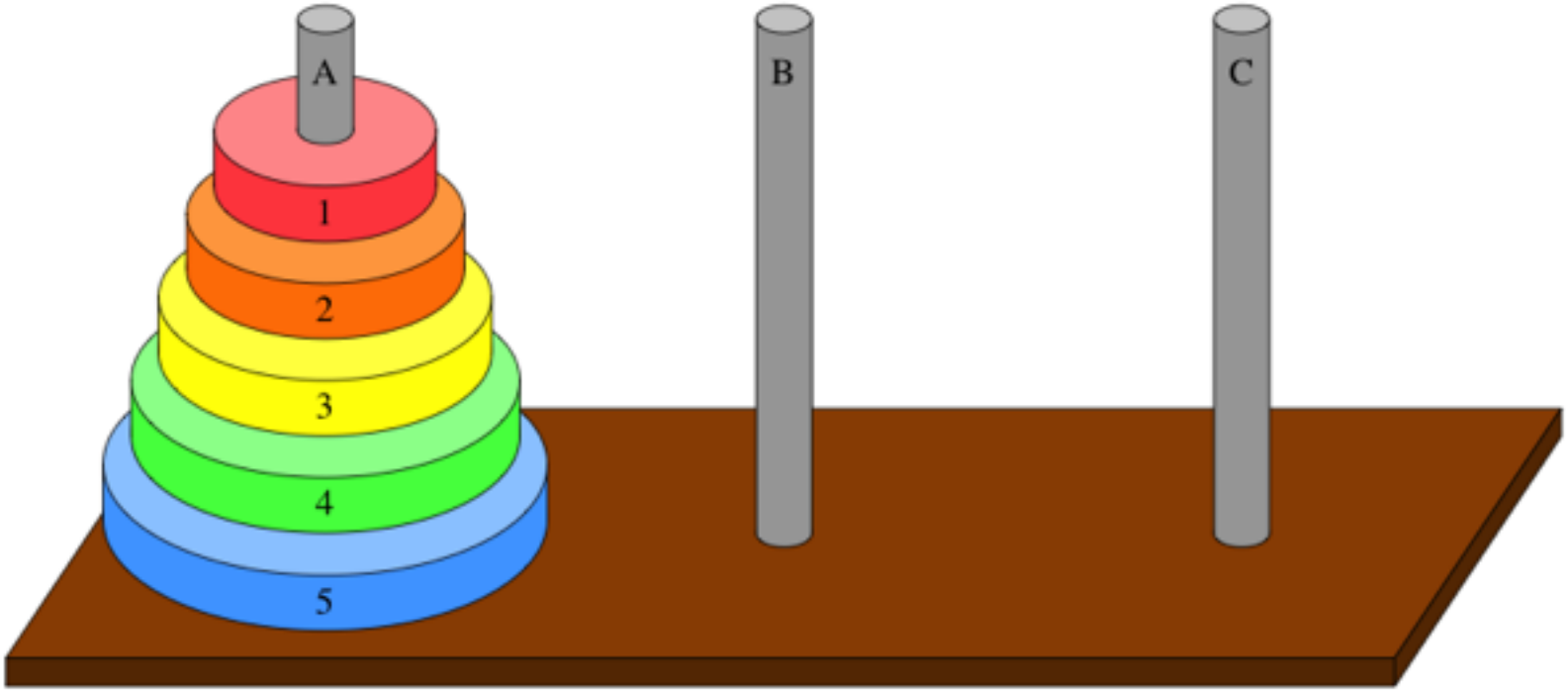
```
    /**
     * Implement multiplication recursively using addition
     * For example, given the args 7 and 4 write a recursive function
     * that computes 7+7+7+7
     * @param i1 a number
     * @param i2 another number
     * @return i1*i2
     */
    public int multiply(int i1, int i2);

    /**
     * Write a recursive function to add all the values in the array
     * Hint, this method should not be recursive.  Rather make a
     * private recursive function and call that from here
     * @param array
     * @return the sum of the numbers in the array
     */
    public int addArray(int[] array);
```

more returning values

```
public ArrayList<Integer> rAccmulate(int count)
{
    if (count <= 0)
        return new ArrayList<Integer>();
    ArrayList<Integer> alAcc = rAccmulate(count-1);
    alAcc.add(count);
    return alAcc;
}
```

Towers of Hanoi



Writing to Files

- In the simplest case as easy as println
  - outputter.java
- Lots for for complex scenarios
  - java.io
  - java.nio.channel
  - java.nio.files