Welcome.TU.code

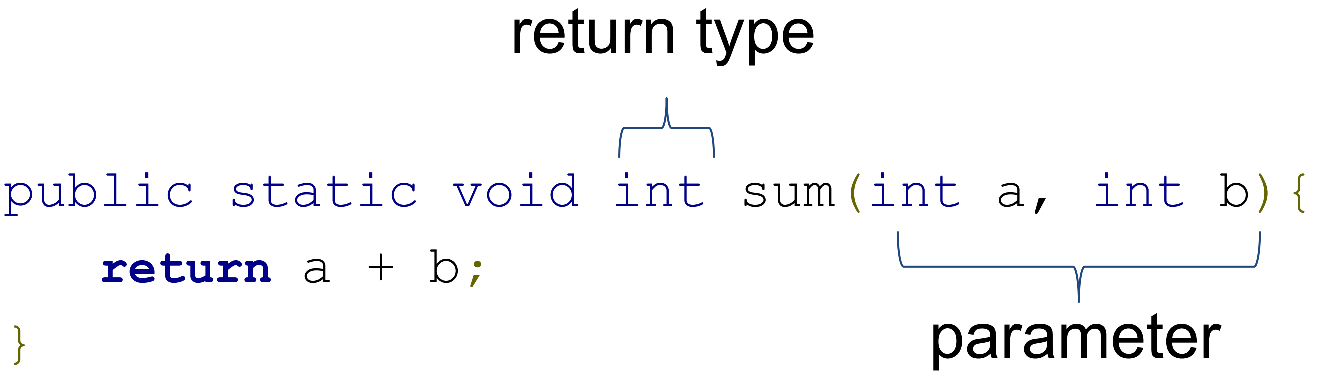
Recursion

Recap

- What types of errors do you know?
- Where are the differences?
- What are Exceptions?
- What kind of Exceptions do you know?

Recap

- What is a function?
- What is a return value?
- What is a parameter?



return type

```
public static void int sum(int a, int b){
    return a + b;
}
```

parameter

Recursion

- A Method which calls itself
- simplifies a problem each method call
- needs a stopping condition

Recursion Example

```
static void fairy()
{
   wish();
   wish();
   fairy();
}
```

Recursion Example 2

```
static void down( int n )
{
   System.out.print( n + ", " );
   down( n - 1 );
}
```

```
Output:
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, ...
```

Recursion Example 3

```
static void down( int n )
{
   if (n<=0){
      return; // End
   }
   System.out.print( n + ", " );
      down( n - 1 );
}
```

```
Output:
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

Exercise 1

```
static void up( int n )
{

...

}
```

Factorial

5! = 5 * 4 * 3 * 2 * 1 = 120

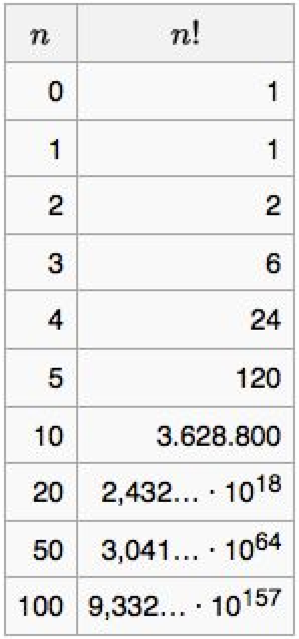
5! = 5 * 4!

5! = 5 * 4 * 3!

5! = 5 * 4 * 3 * 2!

5! = 5 * 4 * 3 * 2 * 1!

5! = 5 * 4 * 3 * 2 * 1



n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120
10	3.628.800
20	$2{,}432\ldots \cdot 10^{18}$
50	$3{,}041\ldots \cdot 10^{64}$
100	$9{,}332\ldots \cdot 10^{157}$

Factorial

```
fact(5) = 5 * fact(4)
              fact(4) = 4 * fact(3)
                            fact(3) = 3 * fact(2)
                            ...
```

fact(n) = n * fact(n-1);

Factorial

```
static int fact( int n )
{
   if (n==1){
      return 1; // End
   }
   System.out.println(n+"*fact("+(n-1)+")");
      return n * fact(n-1);
}

System.out.println(fact(5))
```

```
Output:
5*fact(4)
4*fact(3)
3*fact(2)
2*fact(1)
120
```

Recursion vs Loop

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

VS

```
static int fact( int n ){
    int fact=0;
    for (int i=1; i<=n; i++){
        fact = fact * i;
    }
        return fact;
}
```

Recursion (vs. Loop)

every recursion could be implemented as loop
every loop could be implemented as recursion

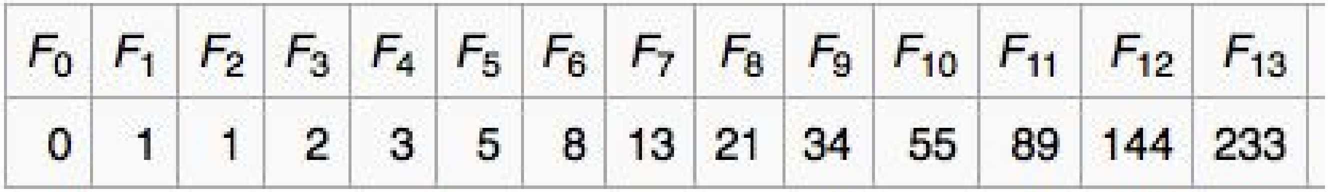
+ easier solutions
- more complex solutions

$\}$ depending on situation

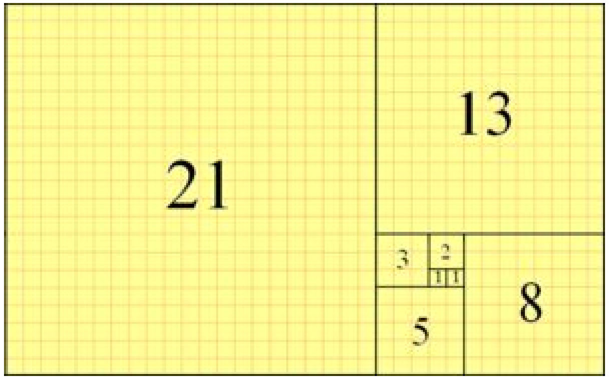
- runtime overhead

Task Fibonacci Numbers

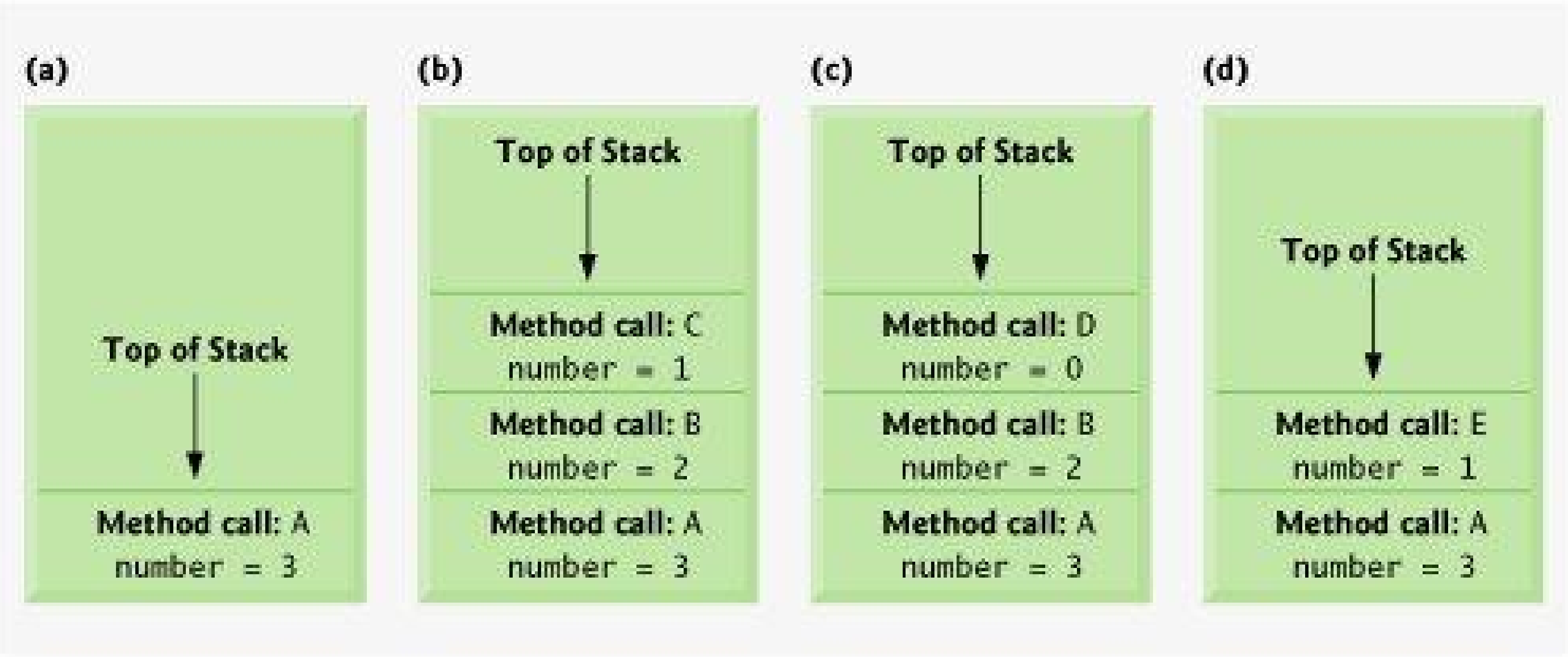
$$F_n = F_{n-1} + F_{n-2},$$



F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
0	1	1	2	3	5	8	13	21	34	55	89	144	233



Method call handling on stack


(a)
Top of Stack
Method call: A
number = 3
(b)
Top of Stack
Method call: C
number = 1
Method call: B
number = 2
Method call: A
number = 3
(c)
Top of Stack
Method call: D
number = 0
Method call: B
number = 2
Method call: A
number = 3
(d)
Top of Stack
Method call: E
number = 1
Method call: A
number = 3