IN1007 Programing in Java

Lecture 3: Methods and Recursion

Announcements

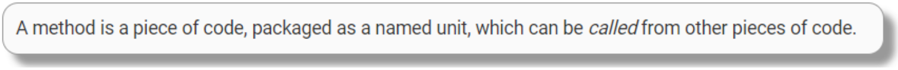
- **No lecture next week** (reading week)

- **First coursework will be available from 12th Nov →** deadline 17th Nov (see details on Moodle)

Today’s Lecture

- Methods:
  - **Defining** a method
  - **Calling** a method
  - Passing input to a method
  - Returning output from a method
  - Method overloading
- The call stack
- Recursion

Motivation: Why use methods?


A method is a piece of code, packaged as a named unit, which can be *called* from other pieces of code.

1. Write code which is easier to understand
2. Build complex programs in a ***modular*** way:
   build big things by first building smaller things and then composing them together
3. Write ***reusable*** code: write a piece of code once and use it many times.

More details:

[MDL_MACSE_PROGBOOT_2023-24: Lecture: Methods (city.ac.uk)]

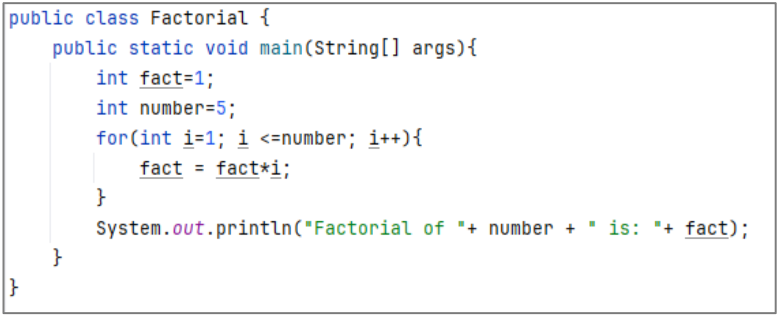
Example: Calculate the factorial of a number *n*

- Factorial of a number *n* is the product of all positive descending integers
- Denoted as *n!*
- Often used in mathematical calculations such as permutations and combinations.

For example:

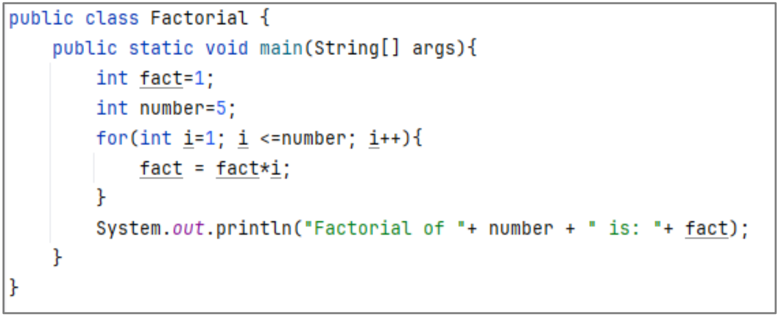
```
4! = 4*3*2*1 = 24
5! = 5*4*3*2*1 = 120
```

Simple program to calculate factorial



```java
public class Factorial {
    public static void main(String[] args){
        int fact=1;
        int number=5;
        for(int i=1; i <=number; i++){
            fact = fact*i;
        }
        System.out.println("Factorial of "+ number + " is: "+ fact);
    }
}
```

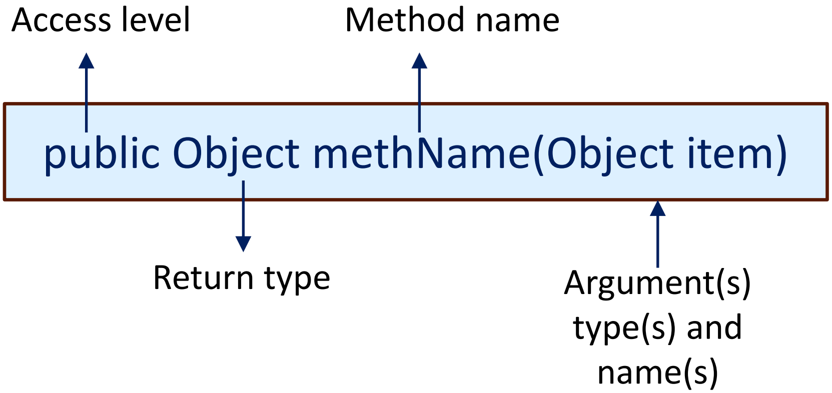
Simple program to calculate factorial



```
public class Factorial {
    public static void main(String[] args){
        int fact=1;
        int number=5;
        for(int i=1; i <=number; i++){
            fact = fact*i;
        }
        System.out.println("Factorial of "+ number + " is: "+ fact);
    }
}
```

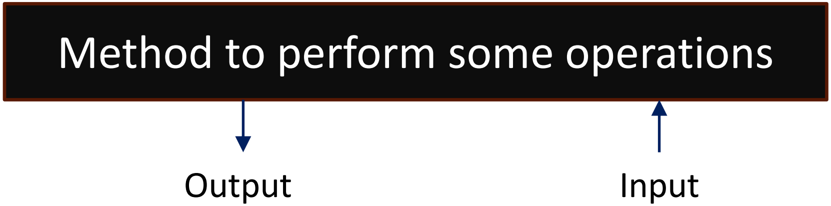
What if you like to calculate $_nC_k = \frac{n!}{k!\,(n-k)!}$. ?

Defining Methods

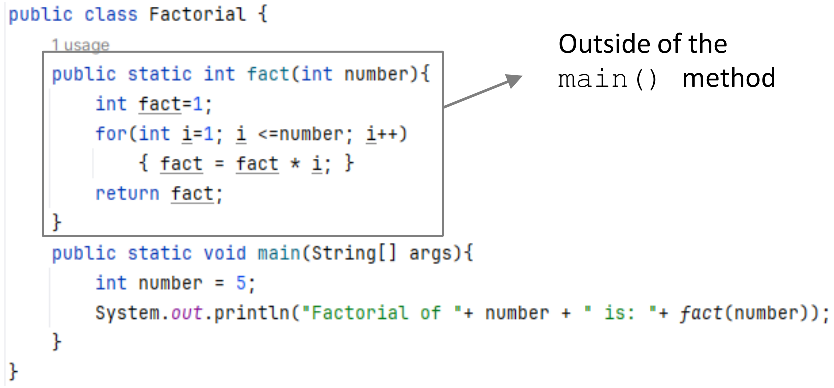


```
public Object methName(Object item)
```

Defining Methods



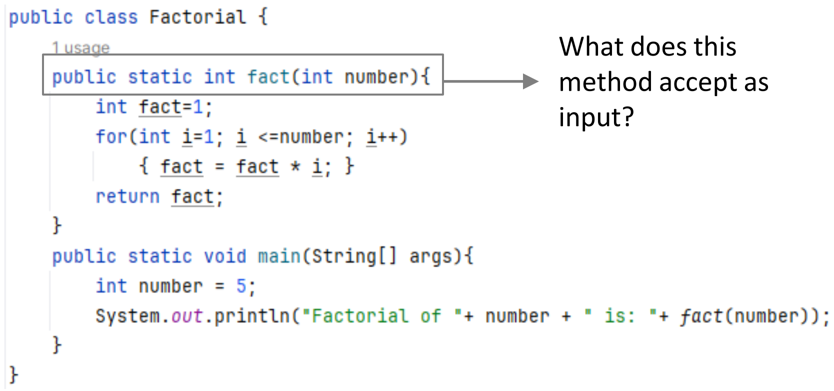
Defining a method: *fact()*



```
public class Factorial {
    1 usage
    public static int fact(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        System.out.println("Factorial of "+ number + " is: "+ fact(number));
    }
}
```

Outside of the `main()` method

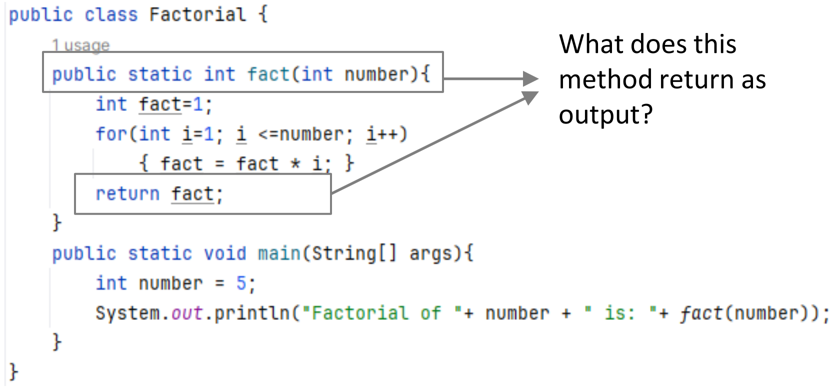
Defining a method: *fact()*



```
public class Factorial {
    1 usage
    public static int fact(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        System.out.println("Factorial of "+ number + " is: "+ fact(number));
    }
}
```

What does this method accept as input?

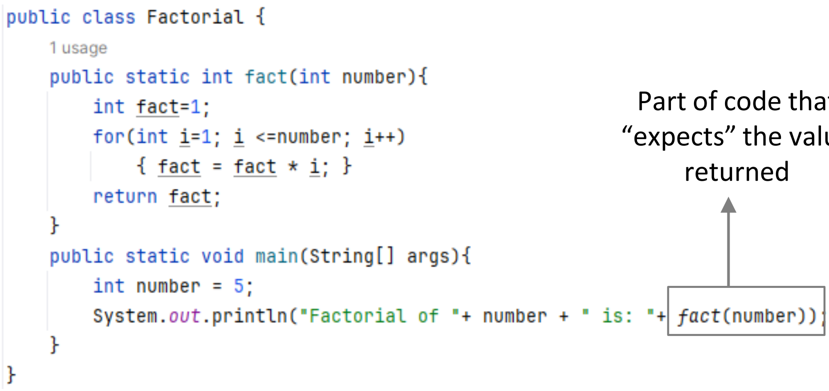
Defining a method: *fact()*



```
public class Factorial {
    1 usage
    public static int fact(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        System.out.println("Factorial of "+ number + " is: "+ fact(number));
    }
}
```

What does this method return as output?

Calling a method: *fact()*

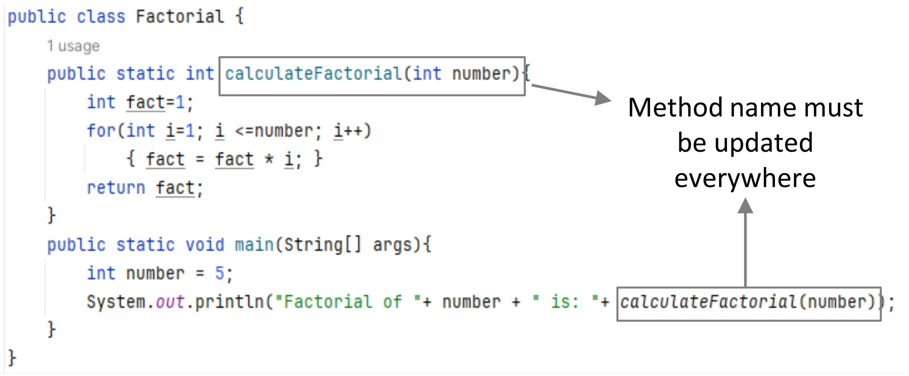


```
public class Factorial {
    1 usage
    public static int fact(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        System.out.println("Factorial of "+ number + " is: "+ fact(number));
    }
}
```

Part of code that "expects" the value returned

Better Practice in naming a method

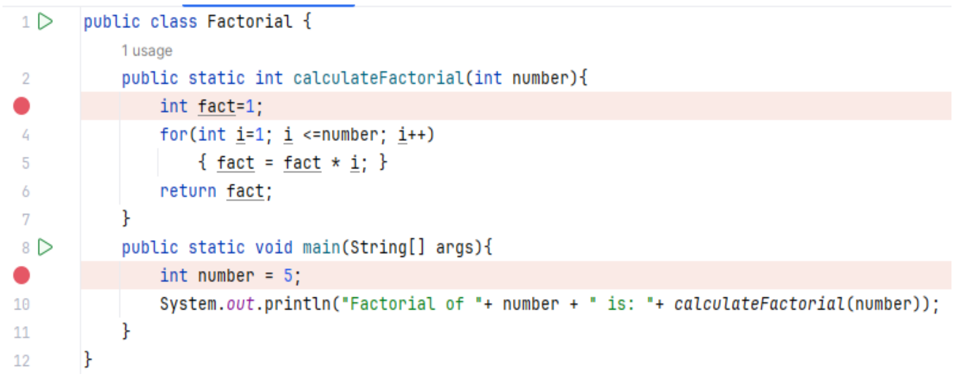
Lowercase, ideally a verb



```
public class Factorial {
    1 usage
    public static int calculateFactorial(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        System.out.println("Factorial of "+ number + " is: "+ calculateFactorial(number));
    }
}
```

Program Flow with Methods

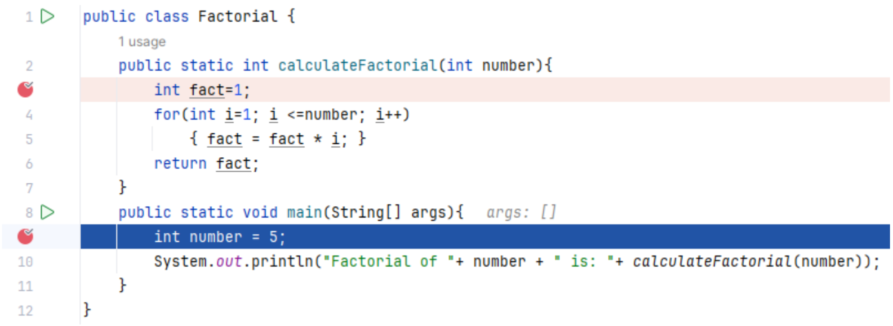
Create two breakpoints at lines 3 and 9, then click “debug”



```
 1  public class Factorial {
        1 usage
 2      public static int calculateFactorial(int number){
 ●          int fact=1;
 4          for(int i=1; i <=number; i++)
 5              { fact = fact * i; }
 6          return fact;
 7      }
 8      public static void main(String[] args){
 ●          int number = 5;
10          System.out.println("Factorial of "+ number + " is: "+ calculateFactorial(number));
11      }
12  }
```

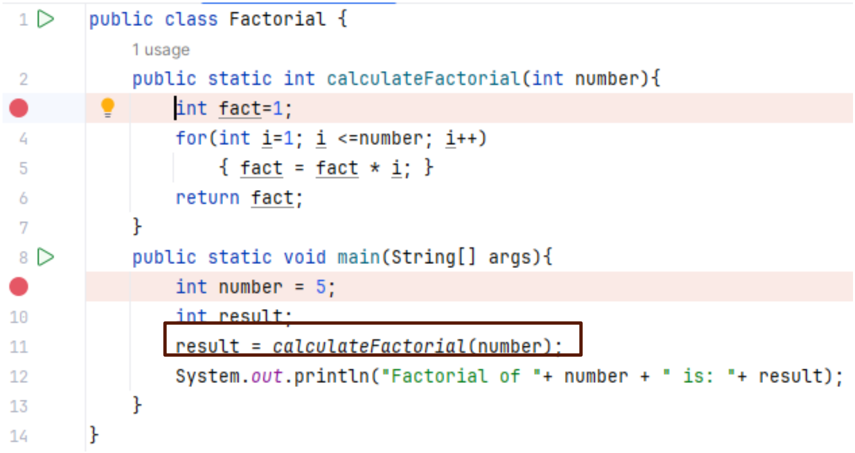
Program Flow with Methods

Line 9 gets highlighted first. ***Why?***



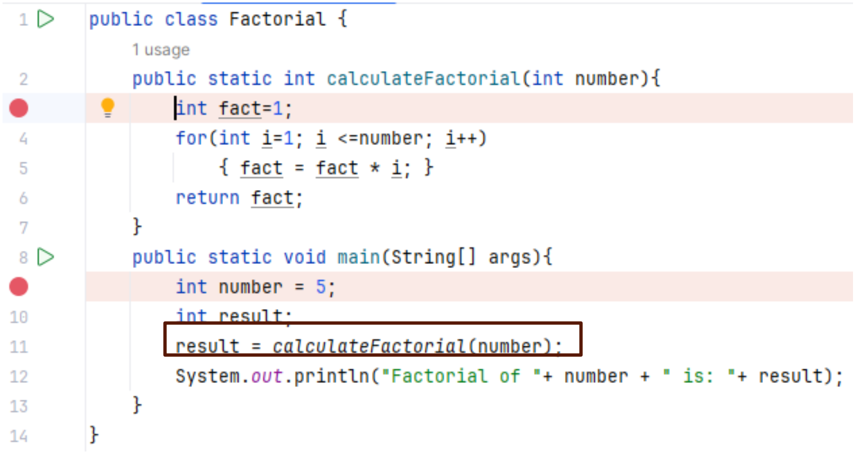
```
1   public class Factorial {
        1 usage
2       public static int calculateFactorial(int number){
            int fact=1;
4           for(int i=1; i <=number; i++)
5               { fact = fact * i; }
6           return fact;
7       }
8       public static void main(String[] args){   args: []
            int number = 5;
10          System.out.println("Factorial of "+ number + " is: "+ calculateFactorial(number));
11      }
12  }
```

Try moving the method call to a separate line:



```
public class Factorial {
    1 usage
    public static int calculateFactorial(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        int result;
        result = calculateFactorial(number);
        System.out.println("Factorial of "+ number + " is: "+ result);
    }
}
```

Now comment out line 11 and see what happens



```
1     public class Factorial {
          1 usage
2         public static int calculateFactorial(int number){
●             int fact=1;
4             for(int i=1; i <=number; i++)
5                 { fact = fact * i; }
6             return fact;
7         }
8         public static void main(String[] args){
●             int number = 5;
10            int result;
11            result = calculateFactorial(number);
12            System.out.println("Factorial of "+ number + " is: "+ result);
13        }
14    }
```

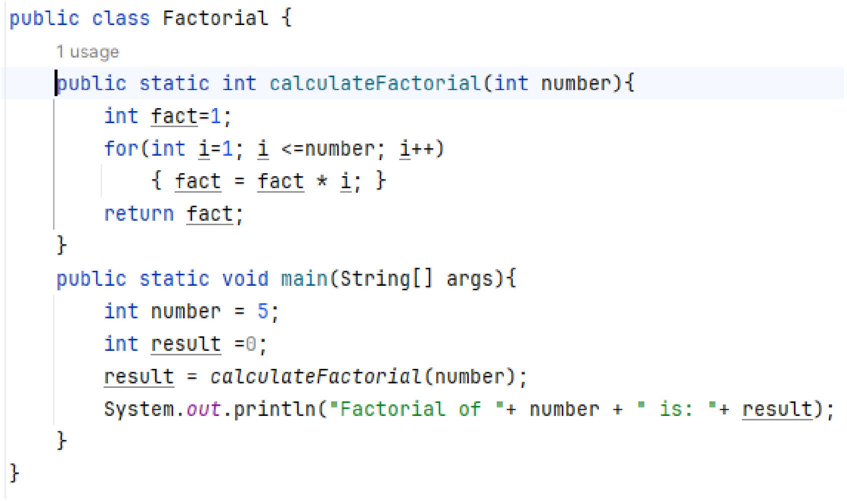
Program Flow with Methods

Code inside a method will only be executed after the method has been called

An exception to this rule is the main() method, which gets executed automatically once a program has started execution

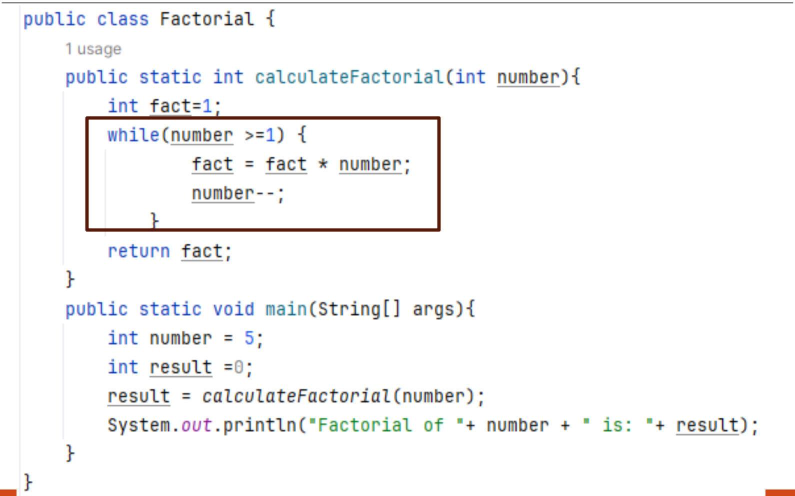
Another special type of method is class constructors. We will visit these in Week 7.

Modify the `calculateFactorial()` method to use **while** instead of a **for** loop



```
public class Factorial {
    1 usage
    public static int calculateFactorial(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        int result =0;
        result = calculateFactorial(number);
        System.out.println("Factorial of "+ number + " is: "+ result);
    }
}
```

We have modified the value of the variable number inside the `calculateFactorial()` method. What happens if you print the number at the end of the `main()` method?



```
public class Factorial {
    1 usage
    public static int calculateFactorial(int number){
        int fact=1;
        while(number >=1) {
                fact = fact * number;
                number--;
            }
        return fact;
    }
    public static void main(String[] args){
        int number = 5;
        int result =0;
        result = calculateFactorial(number);
        System.out.println("Factorial of "+ number + " is: "+ result);
    }
}
```

Passing Input to a Method

- Methods:
  - **Defining** a method
  - **Calling** a method
  - **Passing input to a method**
  - Returning output from a method
  - Method overloading
- The call stack
- Recursion

Primitive data types are passed *by value*

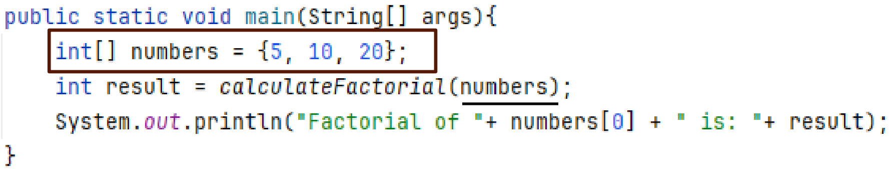
This means that any changes to the value of the parameter only exist within the scope of the method.

When the method returns, the local copy of the parameters is gone and any changes are lost.

What about arrays?

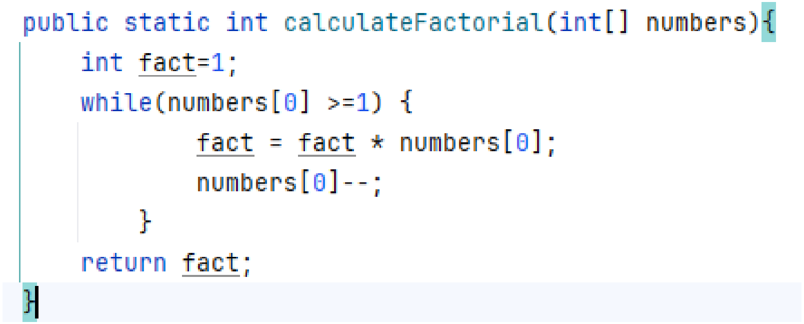
Try this:

- Change the variable number to an array of numbers
- Pass the entire array as a parameter to the method



```java
public static void main(String[] args){
    int[] numbers = {5, 10, 20};
    int result = calculateFactorial(numbers);
    System.out.println("Factorial of "+ numbers[0] + " is: "+ result);
}
```

Notice that you need to modify the method definition to accept an array as parameter

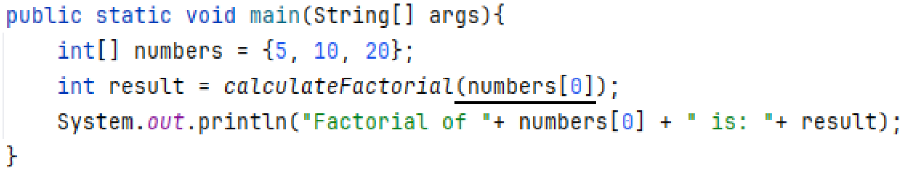


```
public static int calculateFactorial(int[] numbers){
    int fact=1;
    while(numbers[0] >=1) {
            fact = fact * numbers[0];
            numbers[0]--;
        }
    return fact;
}
```

What difference do you observe in program output?

Try this:

- Pass the first element of the array only as a parameter to the method
- You will need to modify the method definition accordingly



```
public static void main(String[] args){
    int[] numbers = {5, 10, 20};
    int result = calculateFactorial(numbers[0]);
    System.out.println("Factorial of "+ numbers[0] + " is: "+ result);
}
```

What difference do you observe in program output?

Array names as reference

Array names are passed in Java by value

Try adding a line in your `calculateFactorial()` method to change the block of memory that your array points to:

```
numbers = new int[] {1,2,3,4};
```

This change will not be visible outside the scope of the method

However, array names are considered a reference to all the members of the array. Changes to the members will outlast the lifetime of the method.

This is called passing parameters *by reference.*

Passing Information in Method Parameters

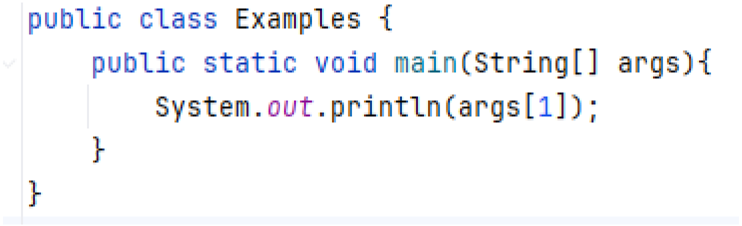
- Parameter names are unique within their scope.
- You cannot have the same name for more than one parameter.
- Naming a local variable with the same name as a parameter will override the parameter value.
    - **Example:** try declaring a variable called number in your calculateFactorial() method.

Returning output from a method

- Methods:
  - **Defining** a method
  - **Calling** a method
  - **Passing input to a method**
  - **Returning output from a method**
  - Method overloading

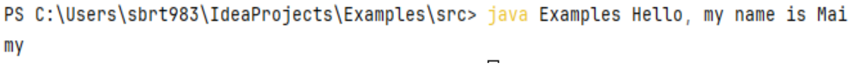
- The call stack
- Recursion

Methods that do not return a value: `void`



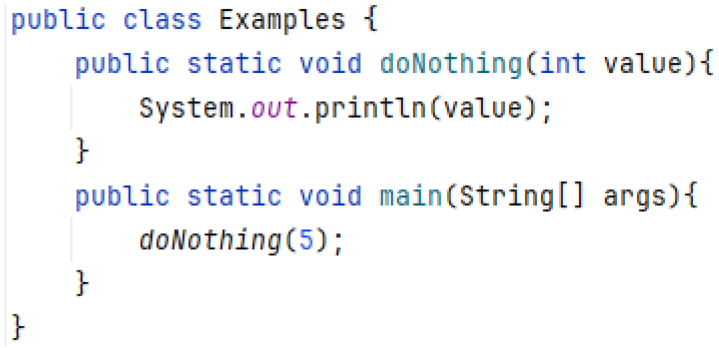
```java
public class Examples {
    public static void main(String[] args){
        System.out.println(args[1]);
    }
}
```

Passing parameters to `main()` must be done in a terminal

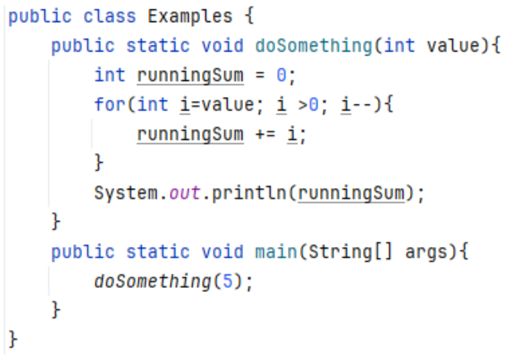


```
PS C:\Users\sbrt983\IdeaProjects\Examples\src> java Examples Hello, my name is Mai
my
```

A method that does nothing



```
public class Examples {
    public static void doNothing(int value){
        System.out.println(value);
    }
    public static void main(String[] args){
        doNothing(5);
    }
}
```



```
public class Examples {
    public static void doSomething(int value){
        int runningSum = 0;
        for(int i=value; i >0; i--){
            runningSum += i;
        }
        System.out.println(runningSum);
    }
    public static void main(String[] args){
        doSomething(5);
    }
}
```

Method that calculates something and prints it without returning any values to the calling code.

Task: Modify this to return a value

Method Overloading

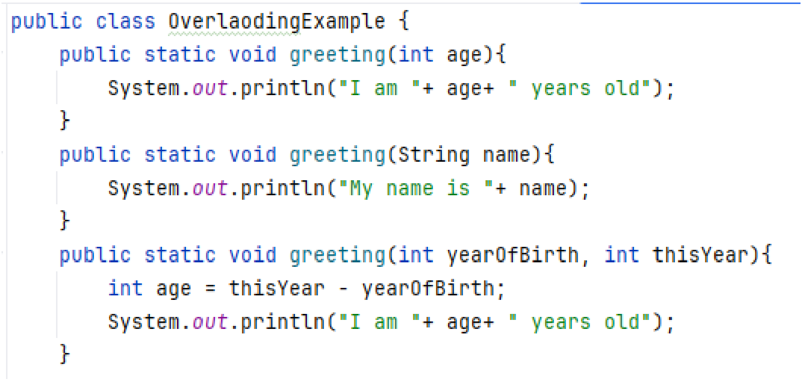
- Methods:
  - **Defining** a method
  - **Calling** a method
  - Passing input to a method
  - Returning output from a method
  - **Method overloading**
- The call stack
- Recursion

Method Signature

The name and parameters (input variables) types of a method constitute its signature

A method can have more than one signature. This is called method overloading

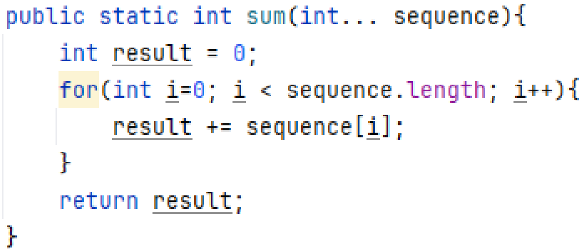
Method Overloading Example



```
public class OverlaodingExample {
    public static void greeting(int age){
        System.out.println("I am "+ age+ " years old");
    }
    public static void greeting(String name){
        System.out.println("My name is "+ name);
    }
    public static void greeting(int yearOfBirth, int thisYear){
        int age = thisYear - yearOfBirth;
        System.out.println("I am "+ age+ " years old");
    }
```

Unknown number of arguments

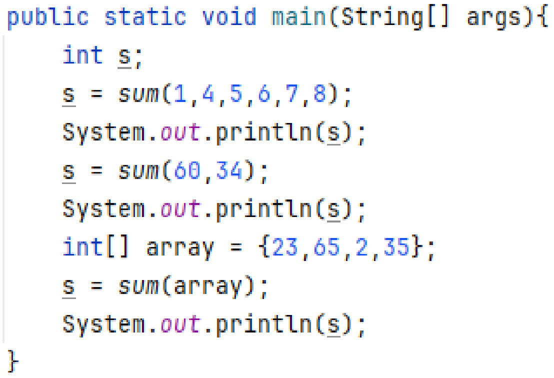
Methods can be called on an unknown number of arguments, using a construct called `varargs`



```java
public static int sum(int... sequence){
    int result = 0;
    for(int i=0; i < sequence.length; i++){
        result += sequence[i];
    }
    return result;
}
```

Try passing different number lists as parameters when calling sum()

Different ways to call sum()

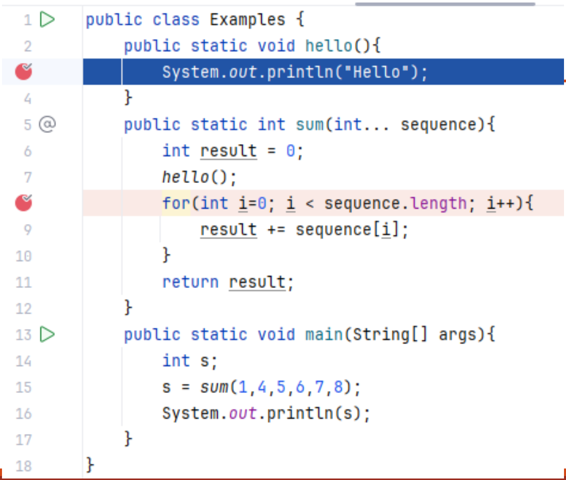


```
public static void main(String[] args){
    int s;
    s = sum(1,4,5,6,7,8);
    System.out.println(s);
    s = sum(60,34);
    System.out.println(s);
    int[] array = {23,65,2,35};
    s = sum(array);
    System.out.println(s);
}
```

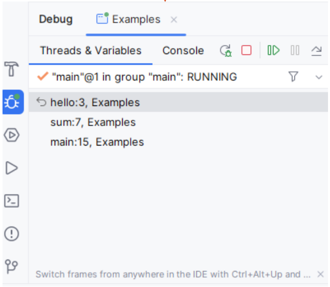
The Call Stack

- Methods:
  - **Defining** a method
  - **Calling** a method
  - Passing input to a method
  - Returning output from a method
  - Method overloading
- **The call stack**
- Recursion

The Java Call Stack: An Example



```
public class Examples {
    public static void hello(){
        System.out.println("Hello");
    }
    public static int sum(int... sequence){
        int result = 0;
        hello();
        for(int i=0; i < sequence.length; i++){
            result += sequence[i];
        }
        return result;
    }
    public static void main(String[] args){
        int s;
        s = sum(1,4,5,6,7,8);
        System.out.println(s);
    }
}
```



```
"main"@1 in group "main": RUNNING
hello:3, Examples
sum:7, Examples
main:15, Examples
```

Switch frames from anywhere in the IDE with Ctrl+Alt+Up and ...

The Call Stack

Nice explanation:
https://www.youtube.com/watch?v=dH2LG3wxXbQ&t=346s

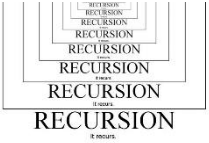
Recursion

Recursion is the process of finding a solution to a problem using a simpler version of itself

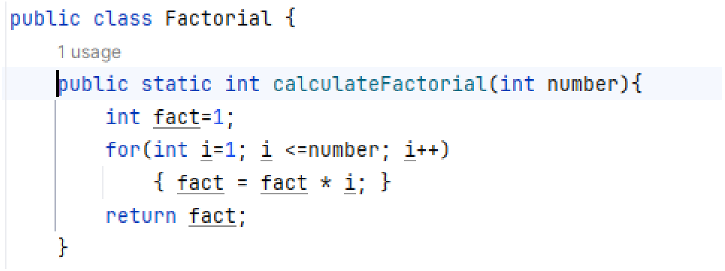
A (dumb) example:

Steps to find your way home!

1. If you are home, stop moving
2. Take one step toward home
3. Find your way home

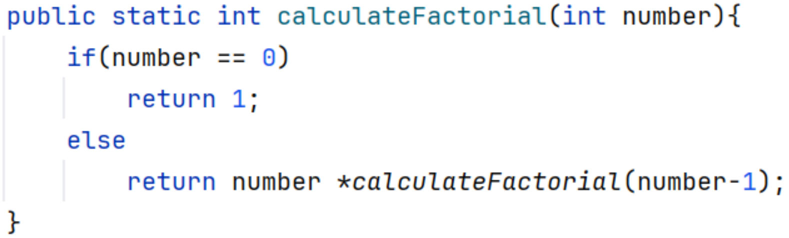


Take a moment with the Factorial example... What would it look like using recursion?



```
public class Factorial {
    1 usage
    public static int calculateFactorial(int number){
        int fact=1;
        for(int i=1; i <=number; i++)
            { fact = fact * i; }
        return fact;
    }
```

Recursive Factorial



```java
public static int calculateFactorial(int number){
    if(number == 0)
        return 1;
    else
        return number *calculateFactorial(number-1);
}
```

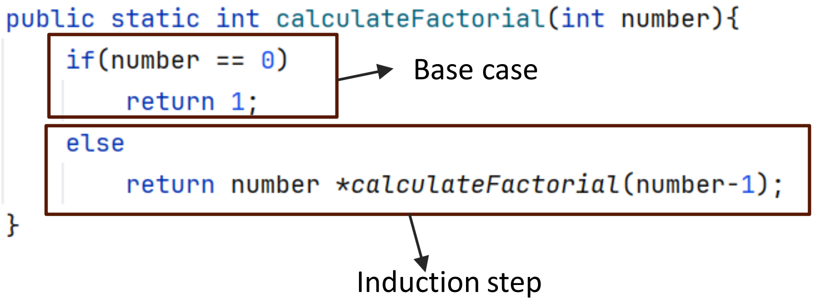
https://cscircles.cemc.uwaterloo.ca/java_visualize/

Structure of a recursive method

A base case → when to stop the recursion

An induction step → calling the method on a "smaller" argument (stepping toward the base case)

Recursive Factorial



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

Recursion Example: Fibonacci Sequence

$$f(1) = f(2) = 1$$

$$\text{For all } n > 0 \quad f(n+2) = f(n+1) + f(n)$$

$$f(1) = 1, f(2) = 1, f(3) = 2, f(4) = 3, f(5) = 5, \ldots$$

```java
public class Fibonacci {
    public static void main(String[] args) {
        System.out.println(fibo(15));
    }

    public int fibo(int n) {
       if ((n==1) || (n==2)) {
          return 1;
       } else {
          return (fibo(n-2) + fibo(n-1));
       }
    }
}
```

A Palindrome

A **palindrome** is a word, number, phrase, or other sequence of symbols that reads the same backwards as forwards, such as *madam* or *racecar.*

Recursion Example: Test Palindrome

Write a recursive method which tests whether a String is palindromic (like "abbccbba" for example).

Hint: you can use

- `s.substring(i,j)` to get the substring of s from index i to index j
- `s.charAt(i)` to get the char at index i in s
- `s.length()` to get the length of s

```
public class TestPalindrome{
   public static boolean isPalindrome(String s){
      if(s.length() == 0 || s.length() == 1){
         return true;
      }
      if(s.charAt(0) == s.charAt(s.length()-1)){
         return
            isPalindrome(s.substring(1, s.length()-1));
      }
      return false;
   }

   public static void main(String[] args){
      System.out.println(isPalindrome("aabbcbbaa"));
      System.out.println(isPalindrome("aabbcbaa"));
   }
}
```

Today’s Lecture

- Methods:
  - **Defining** a method
  - **Calling** a method
  - Passing input to a method
  - Returning output from a method
  - Method overloading

- The call stack
- Recursion