Methods

Defining and Using Methods, Overloads
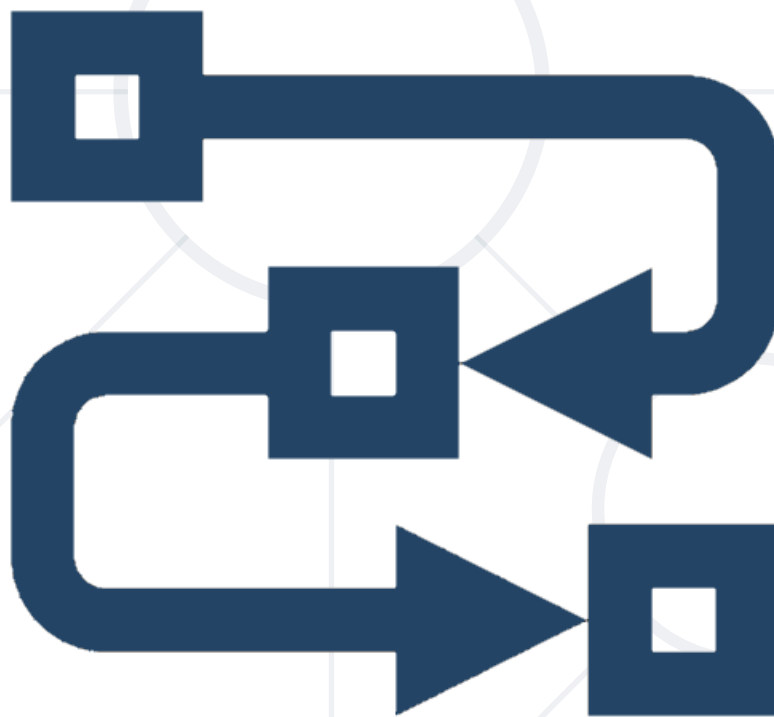
SoftUni Team

Technical Trainers

Table of Contents

2

SoftUni
Foundation

sli.do

#tech-java

What Is a Method
Void Method

Simple Methods

- **Named block of code**, that can be invoked later

- Sample method **definition**:

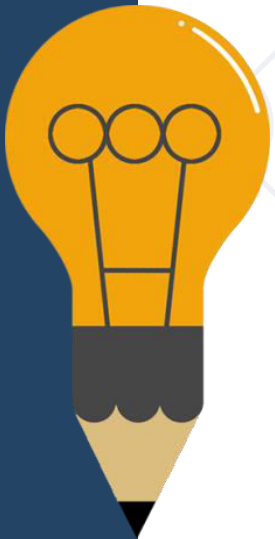> **Method named `printHelloWorld`**

```java
public static void printHello () {
    System.out.println("Hello World");
}
```

> **Method body always surrounded by { }**

- **Invoking** (calling) the method several times:

```java
printHello();
printHello();
```

Why Use Methods?

- More **manageable programming**
  - Splits large problems into small pieces
  - Better organization of the program
  - Improves code readability
  - Improves code understandability
- Avoiding **repeating code**
  - Improves code maintainability
- Code **reusability**
  - Using existing methods several times

Void Type Method

- Executes the code between the brackets

- Does not return result

```java
public static void printHello() {
    System.out.println("Hello");
}
```

Prints "Hello" on the console

```java
public static void main(String[] args) {
    System.out.println("Hello");
}
```

main() is also a method
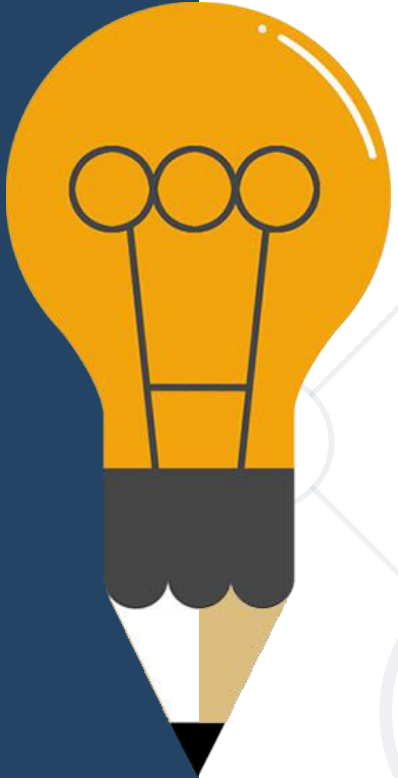
Naming and Best Practices

Naming Methods

- Methods naming guidelines

  - Use **meaningful** method names

  - Method names should answer the question:

    - **What does this method do**?

      ✓ `findStudent, loadReport, sine`

  - If you cannot find a good name for a method, think about whether it has a **clear intent**

🚫 `Method1, DoSomething, HandleStuff, SampleMethod`

Naming Method Parameters

- Method parameters names
  - Preferred form: [**Noun**] or [**Adjective**] + [**Noun**]
  - Should be in **camelCase**
  - Should be **meaningful**

  `firstName`, `report`, `speedKmH`, `usersList`, `fontSizeInPixels`, `font`

  - Unit of measure should be obvious

  `p`, `p1`, `p2`, `populate`, `LastName`, `last_name`, `convertImage`

Methods – Best Practices

- Each method should perform a **single**, well-defined task
  - A Method's name should **describe that task** in a clear and non-ambiguous way

- **Avoid** methods **longer than one screen**
  - **Split them** to several shorter methods

```
private static void printReceipt() {
    printHeader();
    printBody();
    printFooter();
}
```

Self documenting and easy to test

Code Structure and Code Formatting

- Make sure to use correct **indentation**

```
static void main(args) {
    // some code…
    // some more code…
}
```

```
static void main(args)
    {
        // some code…
// some more code…
}
```

- Leave a **blank line** between **methods**, after **loops** and after **if** statements

- Always use **curly brackets** for loops and `if` statements bodies

- **Avoid long lines** and **complex expressions**

Declaring and Invoking Methods

Declaring Methods

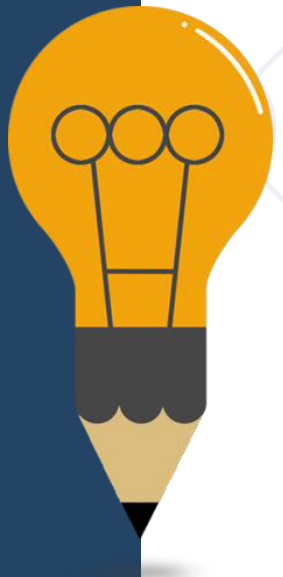SoftUni Foundation

Type   **Method Name**   **Parameters**

```
public static void printText(String text) {
    System.out.println(text);
}
```

Method Body

- Methods are declared **inside a class**
- **main()** is also a method
- Variables inside a method are **local**

Invoking a Method

- Methods are first **declared**, then **invoked** (many times)

```
public static void printHeader() {
    System.out.println("----------");
}
```

Method **Declaration**

- **Methods** can be **invoked (called)** by their name + **()**:

```
public static void main(String[] args) {
    printHeader();
}
```

Method **Invocation**

Invoking a Method (2)

- A method can be invoked from:

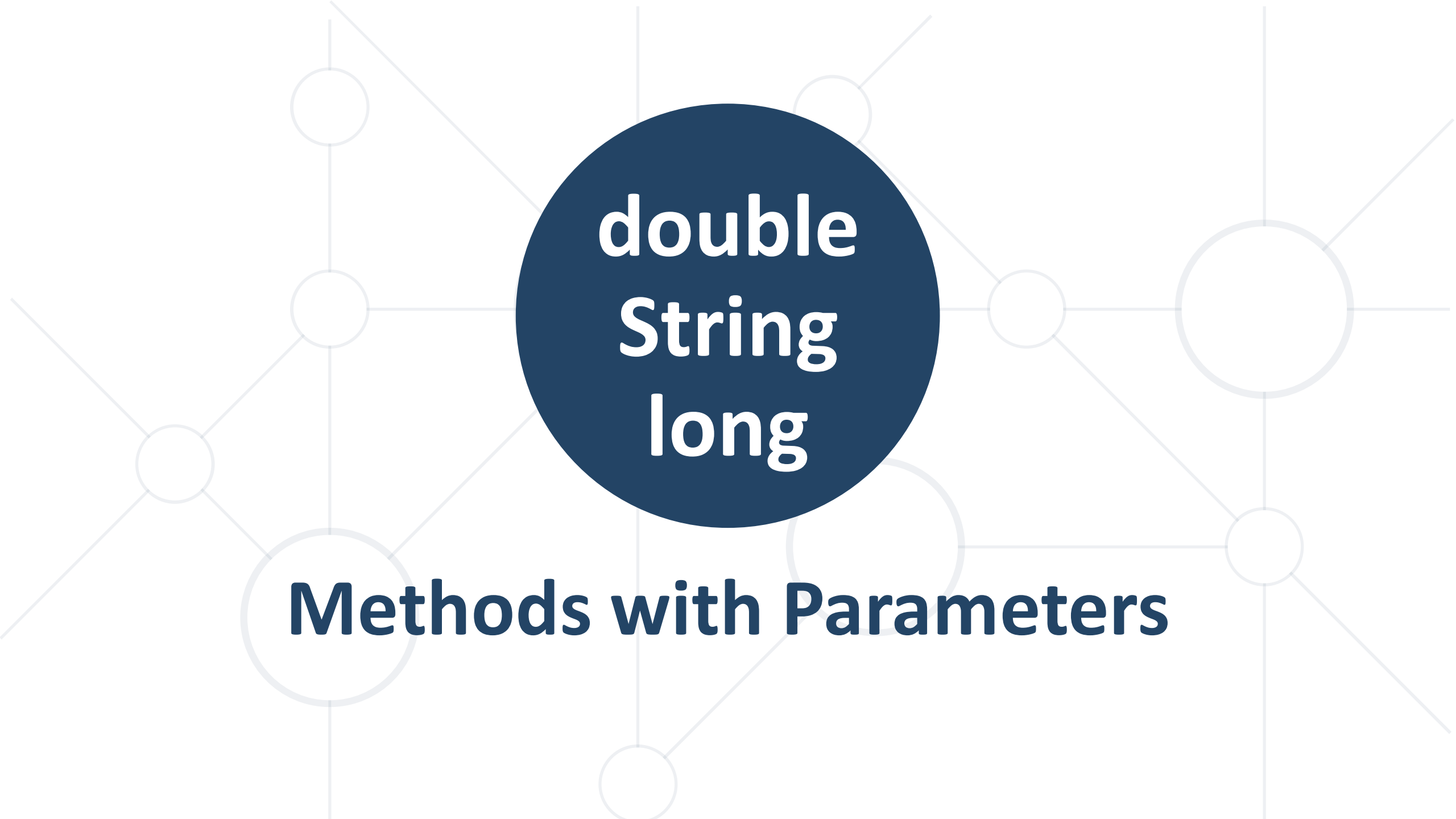   - The main method – **main()**

```java
public static void main(String[] args) {
    printHeader();
}
```

    - Its own body – **recursion**

```java
static void crash() {
    crash();
}
```

    - Some **other method**

```java
public static void printHeader() {
    printHeaderTop();
    printHeaderBottom();
}
```

Method Parameters

- Method **parameters** can be of **any data type**

```java
static void printNumbers(int start, int end) {
    for (int i = start; i <= end; i++) {
        System.out.printf("%d ", i);
    }
}
```

Multiple parameters separated by comma

- Call the method with certain values (**arguments**)

```java
public static void main(String[] args) {
    printNumbers(5, 10);
}
```

Passing arguments at invocation

Method Parameters (2)

- You can pass **zero** or **several** parameters

- You can pass parameters of **different types**

- Each parameter has **name** and **type**

Multiple parameters of different types

Parameter type

Parameter name

```
public static void printStudent(String name, int age, double grade) {
    System.out.printf("Student: %s; Age: %d, Grade: %.2f\n",
        name, age, grade);
}
```

- Create a method that prints the **sign** of an integer number **n**:

| 2 | ➡ | The number 2 is positive. |

| -5 | ➡ | The number -5 is negative. |

| 0 | ➡ | The number 0 is zero. |

Check your solution here: https://judge.softuni.bg/Contests/1260

Solution: Sign of Integer Number

```java
public static void main(String[] args) {
    printSign(Integer.parseInt(sc.nextLine()));
}

public static void printSign(int number) {
    if (number > 0)
        System.out.printf("The number %d is positive.", number);
    else if (number < 0)
        System.out.printf("The number %d is negative.", number);
    else
        System.out.printf("The number %d is zero.", number);
}
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Problem Grades

■ Write a method that receives a grade between 2.00 and 6.00 and prints the corresponding grade in words

■ 2.00 - 2.99 - "Fail"

■ 3.00 - 3.49 - "Poor"

■ 3.50 - 4.49 - "Good"

■ 4.50 - 5.49 - "Very good"

■ 5.50 - 6.00 - "Excellent"

| 3.33 | ➡ | Poor |

| 4.50 | ➡ | Very good |

| 2.99 | ➡ | Fail |

Check your solution here: https://judge.softuni.bg/Contests/1260

Solution Grades

```java
public static void main(String[] args) {

    printInWords(Double.parseDouble(sc.nextLine()));

}

public static void printInWords(double grade) {

    String gradeInWords = "";

    if (grade >= 2 && grade <= 2.99)

        gradeInWords = "Fail";

    //TODO: make the rest

    System.out.println(gradeInWords);

}
```

Problem: Printing Triangle

- Create a method for printing triangles as shown below:

```
3  →   1
       1  2
       1  2  3
       1  2
       1
```

```
4  →   1
       1  2
       1  2  3
       1  2  3  4
       1  2  3
       1  2
       1
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Solution: Printing Triangle (1)

- Create a method that **prints a single line**, consisting of numbers from a **given start** to a **given end**:

```java
public static void printLine(int start, int end) {
    for (int i = start; i <= end; i++) {
        System.out.print(i + " ");
    }
    System.out.println();
}
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Solution: Printing Triangle (2)

- Create a method that prints the **first half (1..n)** and then the **second half (n-1...1)** of the triangle:

Method with parameter n

```java
public static void printTriangle(int n) {
    for (int line = 1; line <= n; line++)
        printLine(1, line);

    for (int line = n - 1; line >= 1; line--)
        printLine(1, line);
}
```

Lines 1...n

Lines n-1...1

Check your solution here: https://judge.softuni.bg/Contests/1260

Live Exercises
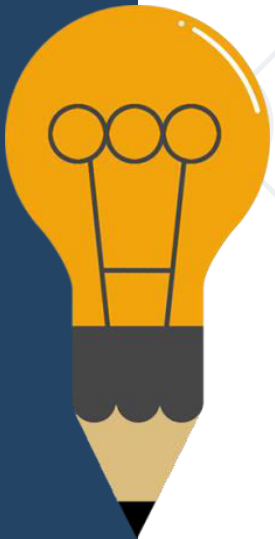
Returning Values From Methods

The Return Statement

- The **return** keyword immediately stops the method's execution

- Returns the specified value

```java
public static String readFullName(Scanner sc) {
    String firstName = sc.nextLine();
    String lastName = sc.nextLine();
    return firstName + " " + lastName;
}
```

Returns a String

- Void methods can be **terminated** by just using **return**

Using the Return Values

- Return value can be:

  - **Assigned** to a variable

  ```
  int max = getMax(5, 10);
  ```

  - **Used** in expression

  ```
  double total = getPrice() * quantity * 1.20;
  ```

  - **Passed** to another method

  ```
  int age = Integer.parseInt(sc.nextLine());
  ```

Problem: Calculate Rectangle Area

- Create a method which returns rectangle area with given width and height

3 4	➡ 12
6 8	➡ 48
5 10	➡ 50
7 8	➡ 56

Check your solution here: https://judge.softuni.bg/Contests/1260

```java
public static void main(String[] args) {
    double width = Double.parseDouble(sc.nextLine());
    double height = Double.parseDouble(sc.nextLine());
    double area = calcRectangleArea(width, height);
    System.out.printf("%.0f%n",area);
}
```

```java
public static double calcRectangleArea(double width,
                                       double height) {
  return width * height;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Problem: Repeat String

- Write a method that receives a string and a repeat count n

- The method should return a new string

abc 3	➡	abcabcabc

String 2	➡	StringString

Check your solution here: https://judge.softuni.bg/Contests/1260

33
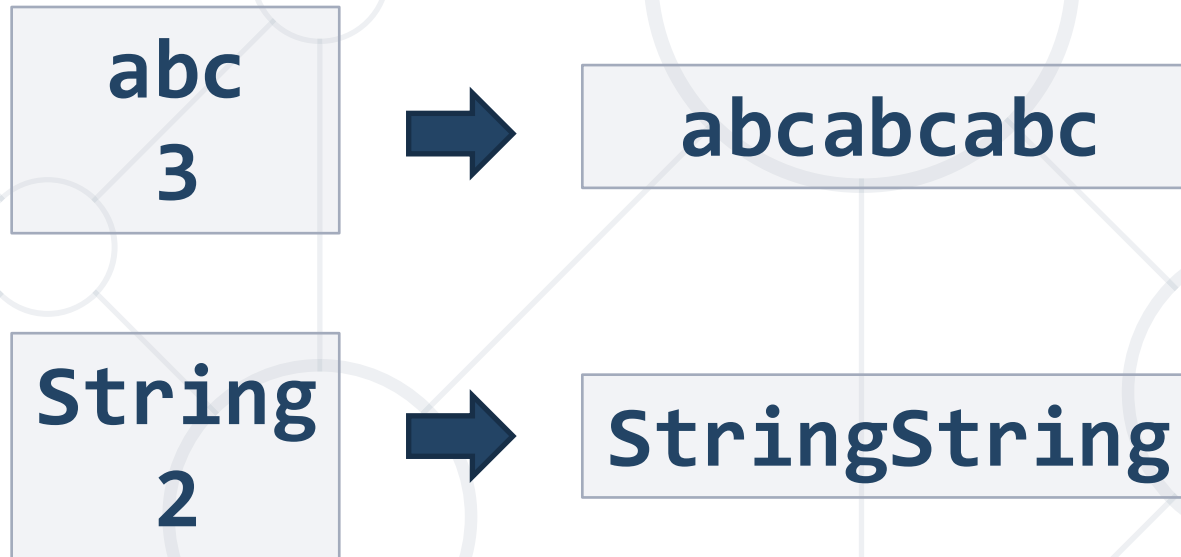
Solution: Repeat String

```java
public static void main(String[] args) {

    String inputStr = sc.nextLine();

    int count = Integer.parseInt(sc.nextLine());

    System.out.println(repeatString(inputStr, count));

}

private static String repeatString(String str, int count) {

    String result = "";

    for (int i = 0; i < count; i++) result += str;

    return result;

}
```

Problem: Math Power

- Create a method that calculates and returns the value of a **number raised to a given power**

2^8 ➡ **256** 5.5^3 ➡ **166.375**
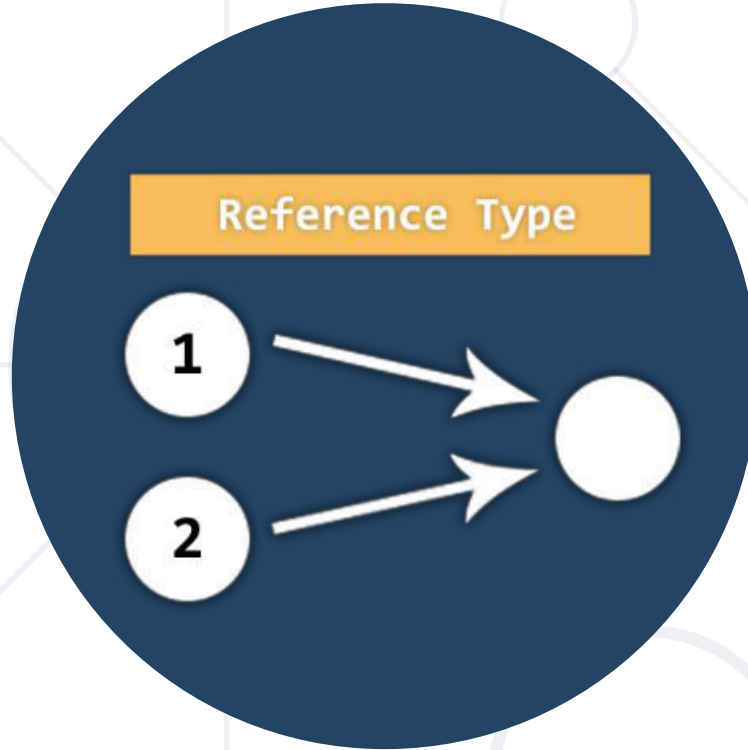
```
public static double mathPower(double number, int power) {
    double result = 1;
    for (int i = 0; i < power; i++)
        result *= number;
    return result;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Live Exercises

Value vs. Reference Types
Memory Stack and Heap

Value vs. Reference Types

pass by reference

pass by value

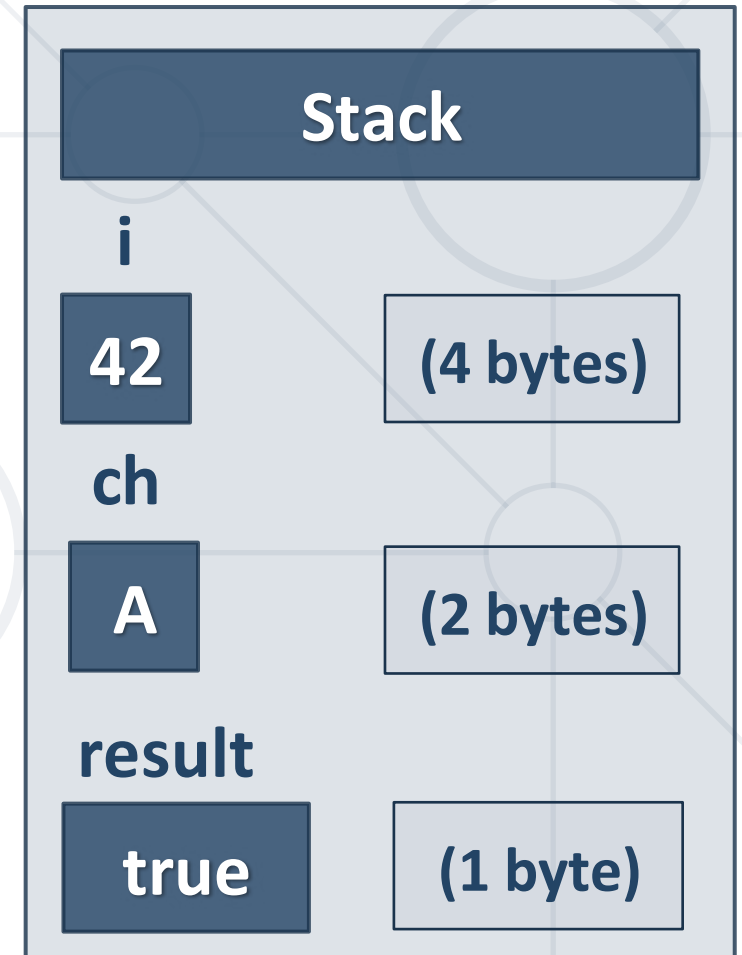cup =

cup =

fillCup(   )

fillCup(   )

Value Types

- **Value type** variables hold directly their value
    - **int**, **float**, **double**, **boolean**, **char**, …
- Each variable has its own **copy** of the **value**

```
int i = 42;
char ch = 'A';
boolean result = true;
```

Stack

i
42   (4 bytes)

ch
A   (2 bytes)
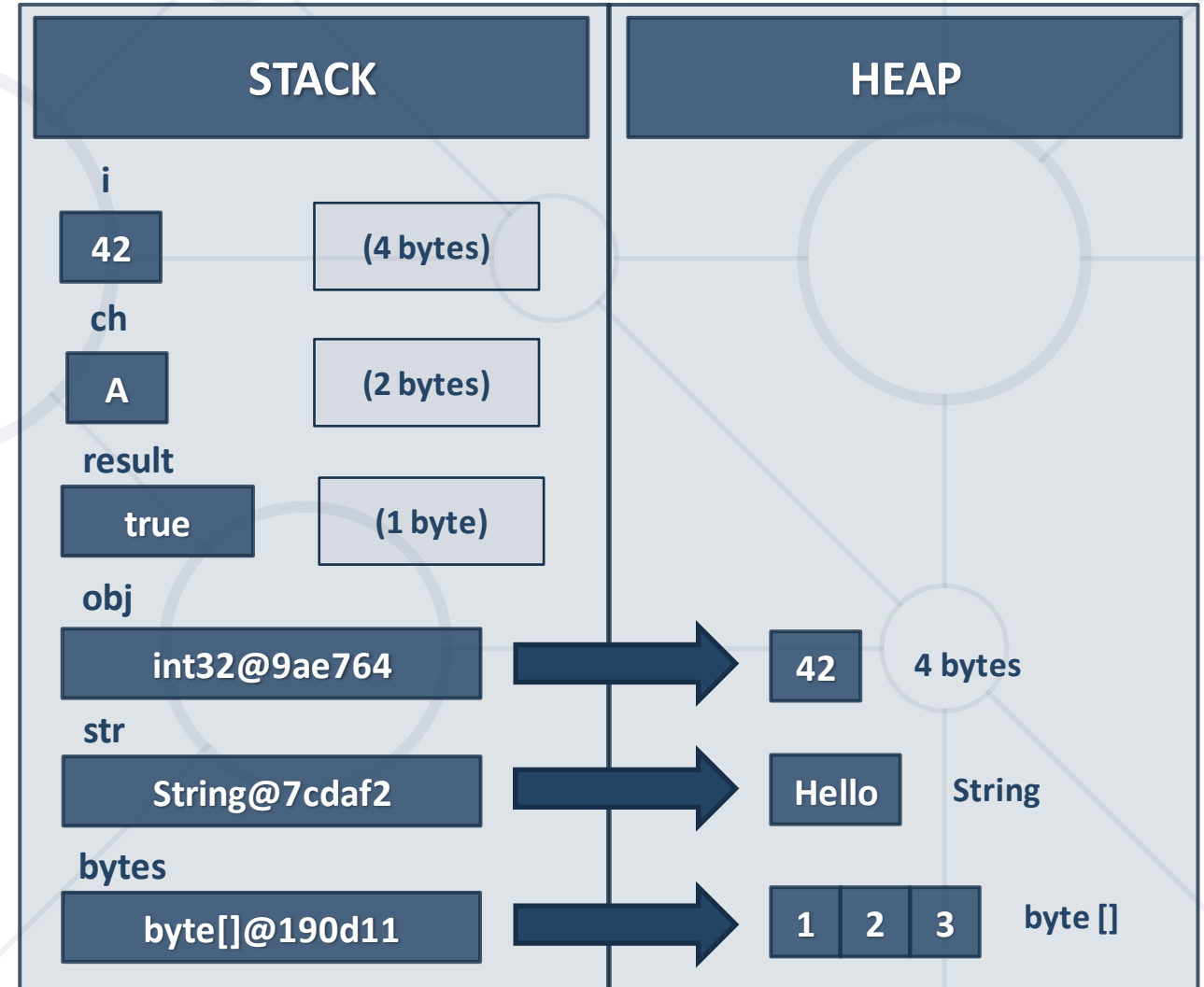
result
true   (1 byte)

Reference Types

- **Reference type** variables hold a reference (pointer / memory address) of the value itself

  - **String**, **int[]**, **char[]**, **String[]**

- Two reference type variables can **reference** the **same object**

  - Operations on both variables access / modify **the same data**

Value Types vs. Reference Types


```
int i = 42;
char ch = 'A';
boolean result = true;
Object obj = 42;
String str = "Hello";
byte[] bytes ={ 1, 2, 3 };
```

STACK

i
42   (4 bytes)

ch
A   (2 bytes)

result
true   (1 byte)

obj
int32@9ae764

str
String@7cdaf2

bytes
byte[]@190d11

HEAP

42   4 bytes

Hello   String

1 2 3   byte []

Example: Value Types

```java
public static void main(String[] args) {

    int num = 5;

    increment(num, 15);          num == 5

    System.out.println(num);

}


public static void increment(int num, int value) {

    num += value;
                          num == 20
}
```

```java
public static void main(String[] args) {

    int[] nums = { 5 };

    increment(nums, 15);          nums[0] == 20

    System.out.println(nums[0]);

}


public static void increment(int[] nums, int value) {

    nums[0] += value;             nums[0] == 20
}
```

Live Exercises

Overloading Methods

Method Signature

- The combination of method's **name** and **parameters** is called **signature**

```
public static void print(String text) {

    System.out.println(text);

}
```

Method's **signature**

- Signature **differentiates** between methods with same names

- When methods with the **same name** have **different signature**, this is called method "**overloading**"

Overloading Methods

SoftUni Foundation

- Using same name for multiple methods with different **signatures** (method **name** and **parameters**)

```java
static void print(int number) {
    System.out.println(number);
}
```

```java
static void print(String text) {
    System.out.println(text);
}
```

```java
static void print(String text, int number) {
    System.out.println(text + ' ' + number);
}
```

Different method signatures

Signature and Return Type

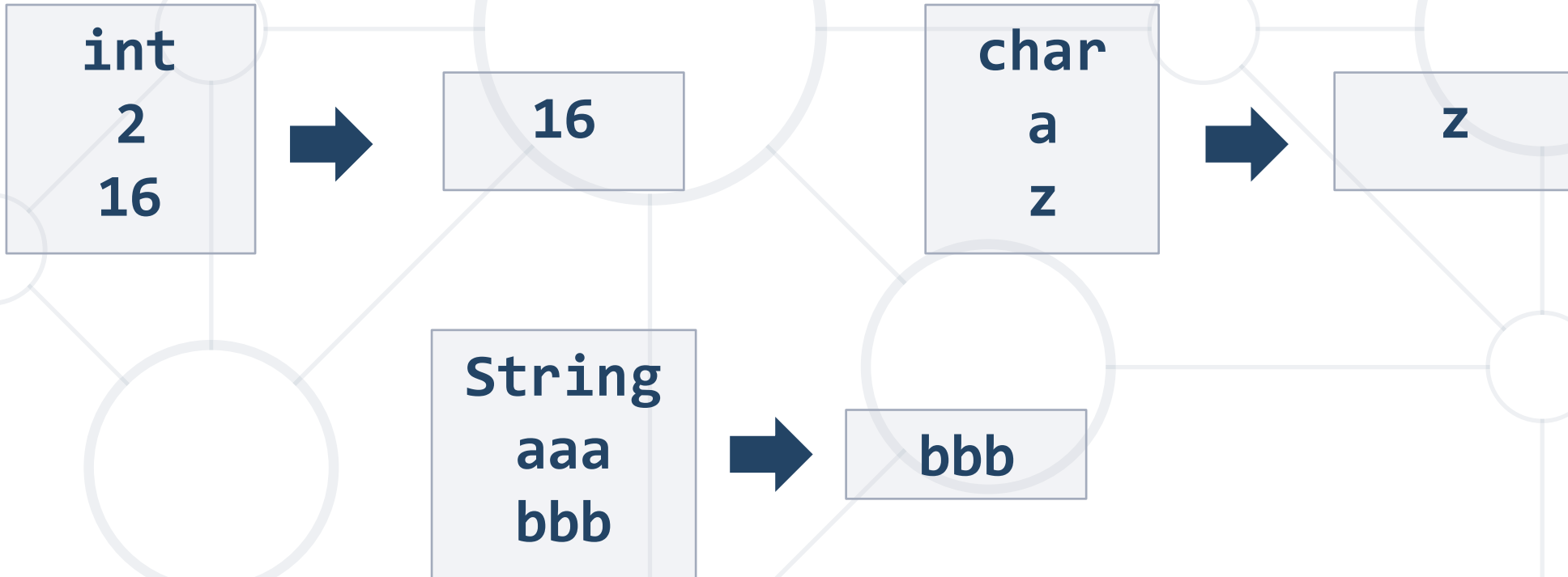- Method's return type **is not part** of its signature

```java
public static void print(String text) {
    System.out.println(text);
}

public static String print(String text) {
    return text;
}
```

Compile-time error!

- How would the compiler know **which method to call**?
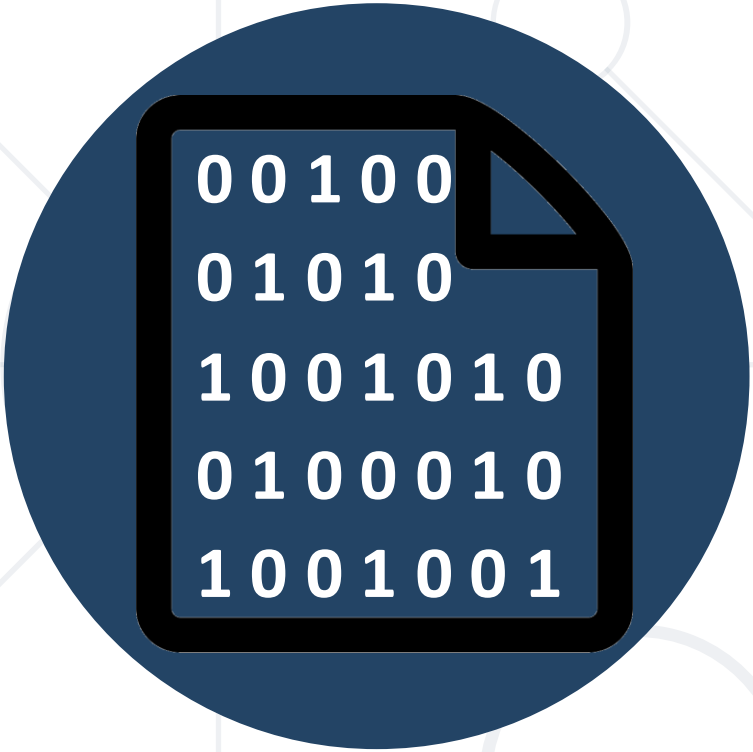
Problem: Greater of Two Values

- Create a method **getMax()** that **returns the greater** of two values (the values can be of type **int**, **char** or **String**)

```
int
2
16
```
→
```
16
```

```
char
a
z
```
→
```
z
```

```
String
aaa
bbb
```
→
```
bbb
```

Check your solution here: https://judge.softuni.bg/Contests/1260

Live Exercises

Program Execution Flow

Program Execution

- The program continues, after a method execution completes:

```java
public static void main(String[] args) {
    System.out.println("before method executes");
    printLogo();
    System.out.println("after method executes");
}
```

```java
public static void printLogo() {
    System.out.println("Company Logo");
    System.out.println("http://www.companywebsite.com");
}
```

Program Execution – Call Stack

- "The stack" **stores information** about the **active subroutines** (methods) of a computer program

- Keeps track of **the point** to which each active subroutine should **return control** when it **finishes executing**
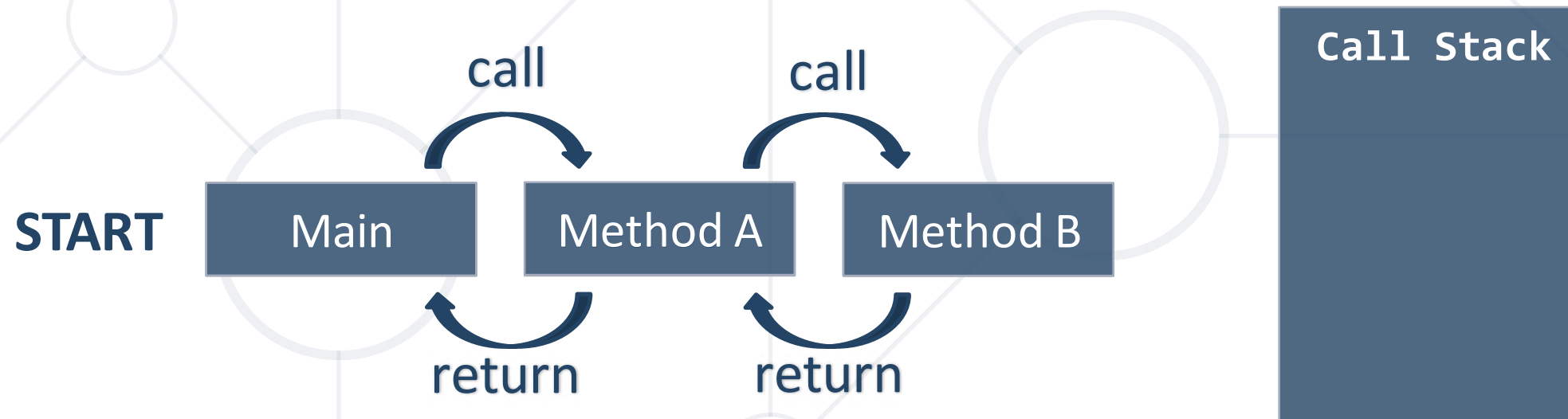
Problem: Multiply Evens by Odds

- Create a program that **multiplies the sum** of **all even digits** of a number **by the sum of all odd digits** of the same number:

  - Create a method called **getMultipleOfEvensAndOdds()**

  - Create a method **getSumOfEvenDigits()**

  - Create **getSumOfOddDigits()**

  - You may need to use **Math.abs()** for negative numbers

-12345	➡	Evens: 2 4 Odds: 1 3 5	➡	Even sum: 6 Odd sum: 9	➡	54

Check your solution here: https://judge.softuni.bg/Contests/1260
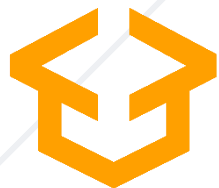
Live Exercises

Summary

- Break large programs into simple **methods** that solve small sub-problems

- Methods consist of **declaration** and **body**

- Methods are invoked by their **name** + **()**

- Methods can accept **parameters**

- Methods can **return** a value or nothing (**void**)

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SoftUni Organizational Partners

ИС ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities

  - softuni.bg

- Software University Foundation

  - http://softuni.foundation/

- Software University @ Facebook

  - facebook.com/SoftwareUniversity

- Software University Forums

  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license