Methods

Defining and Using Methods, Overloads

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University
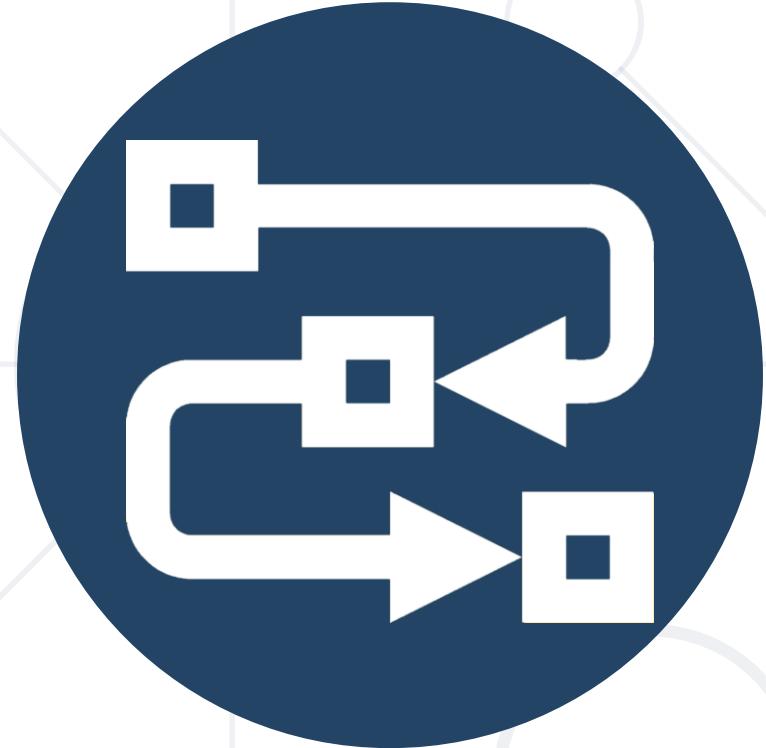
sli.do

#fund-csharp

Table of Contents

What is a Method

Void Method

Simple Methods

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

- Sample method **definition**

> **Method named PrintHelloWorld**

```
static void PrintHelloWorld()
{

    Console.WriteLine("Hello World");

}
```

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

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

```
PrintHelloWorld();
PrintHelloWorld();
```

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

```
static void PrintHello()
{
    Console.WriteLine("Hello");
}
```

Prints "Hello" on the console

```
static void Main()
{
    Console.WriteLine("Hello");
}
```

Main() is also a method

Declaring and Invoking Methods

Declaring Methods
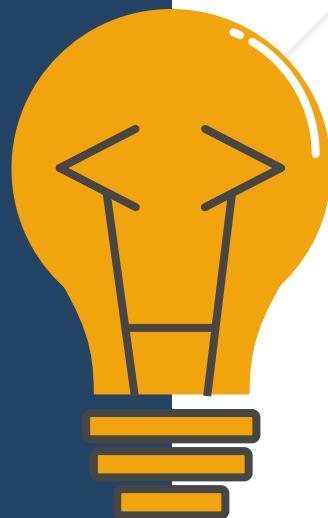
Type **Method Name** **Parameters**

```
static void PrintText(string text)
{
    Console.WriteLine(text);
}
```

Method Body

- Methods are declared **inside a class**
- Variables inside a method are **local**

Invoking a Method

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

```
static void PrintHeader()
{
    Console.WriteLine("----------");
}
```

Method Declaration

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

```
static void Main()
{
    PrintHeader();
}
```

Method Invocation

Invoking a Method

- A method can be invoked from

  - The main method – **Main()**

    ```
    static void Main()
    {
      PrintHeader();
    }
    ```

  - **Its own body** – recursion

    ```
    static void Crash()
    { Crash(); }
    ```
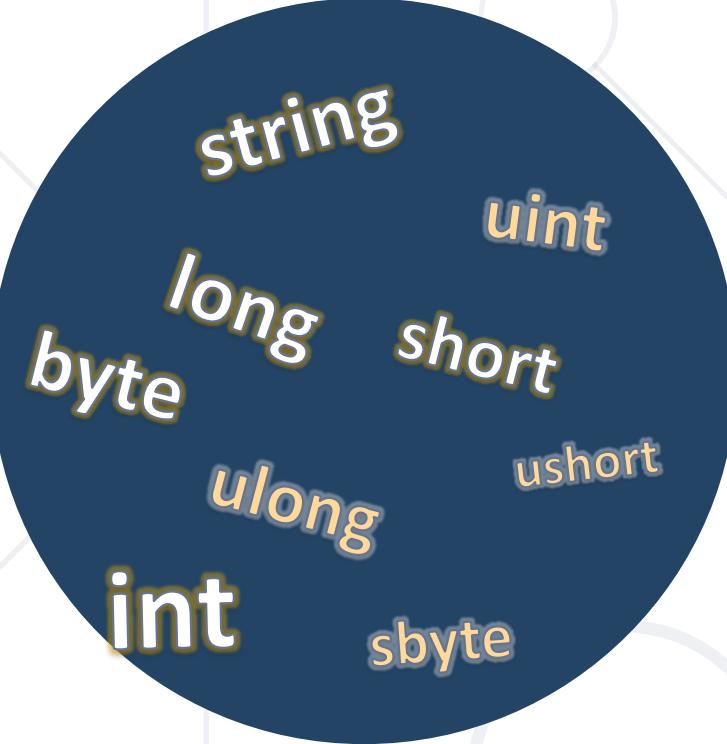
  - Some **other method**

    ```
    static void PrintHeader()
    {
      PrintHeaderTop();
      PrintHeaderBottom();
    }
    ```

Methods with Parameters

Method Parameters

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

```csharp
static void PrintNumbers(int start, int end)
{
  for (int i = start; i <= end; i++)
  {
    Console.Write("{0} ", i);
  }
}
```

Multiple parameters separated by comma

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

```csharp
static void Main()
{
  PrintNumbers(5, 10);
}
```

Passing arguments at invocation

Method Parameters

- 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

```
static void PrintStudent(string name, int age, double grade)
{
    Console.WriteLine("Student: {0}; Age: {1}, Grade: {2}",
        name, age, grade);
}
```

Problem: Sign of Integer Number

- 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.org/Contests/Practice/Index/1208#0

Solution: Sign of Integer Number
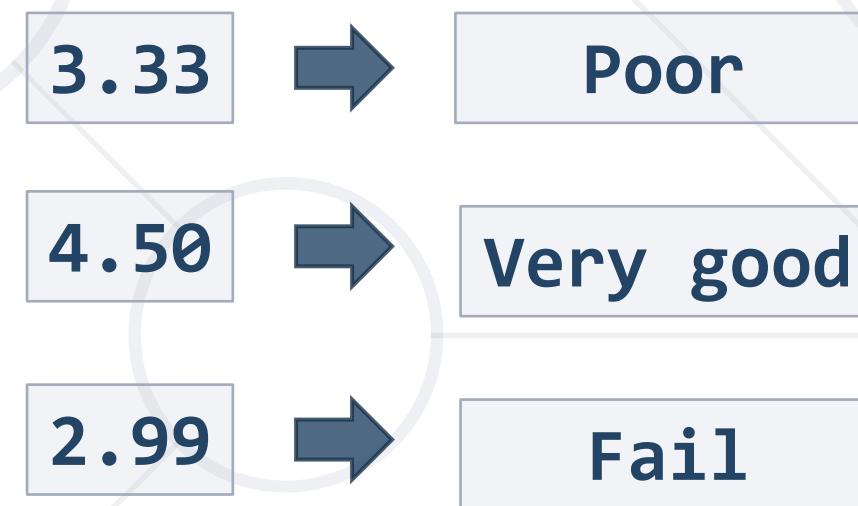
```
static void Main()
{ PrintSign(int.Parse(Console.ReadLine())); }

static void PrintSign(int number)
{
    if (number > 0)
        Console.WriteLine("The number {0} is positive", number);
    else if (number < 0)
        Console.WriteLine("The number {0} is negative.", number);
    else
        Console.WriteLine("The number {0} is zero.", number);
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#0

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.org/Contests/Practice/Index/1208#1

Solution: Grades

```csharp
static void Main()
{
  PrintInWords(double.Parse(Console.ReadLine()));
}
private static void PrintInWords(double grade)
{
  string gradeInWords = string.Empty;
  if (grade >= 2 && grade <= 2.99)
    gradeInWords = "Fail";
  // TODO: Write the rest
  Console.WriteLine(gradeInWords);
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#1

Optional Parameters

- Parameters can accept **default values**

```csharp
static void PrintNumbers(int start = 0, int end = 100)
{
    for (int i = start; i <= end; i++)
    {
        Console.Write("{0} ", i);
}}
```

Default values

- The above method can be called in several ways
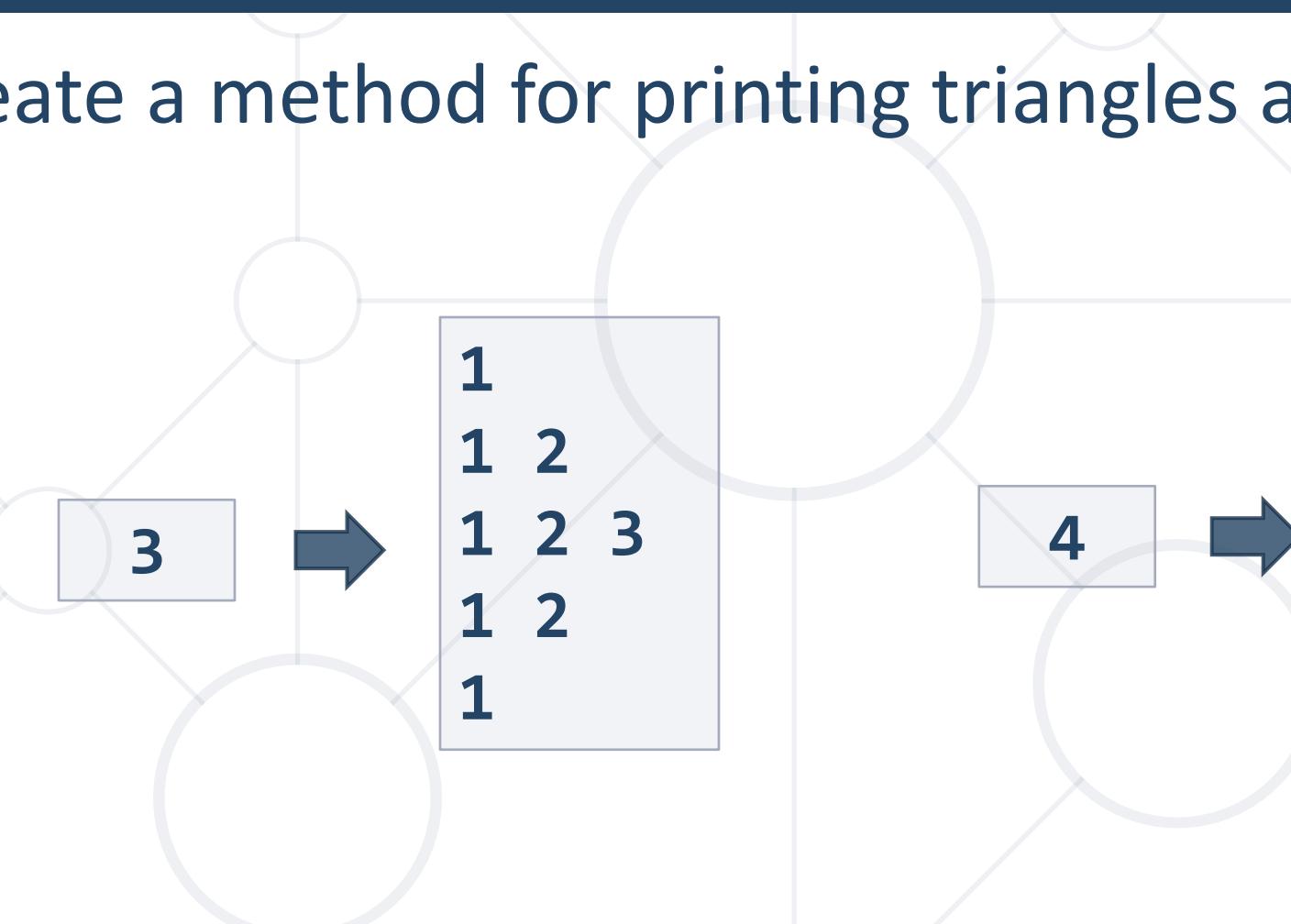
```csharp
PrintNumbers(5, 10);
```
```csharp
PrintNumbers(15);
```
```csharp
PrintNumbers(end: 40, start: 35);
```
```csharp
PrintNumbers();
```
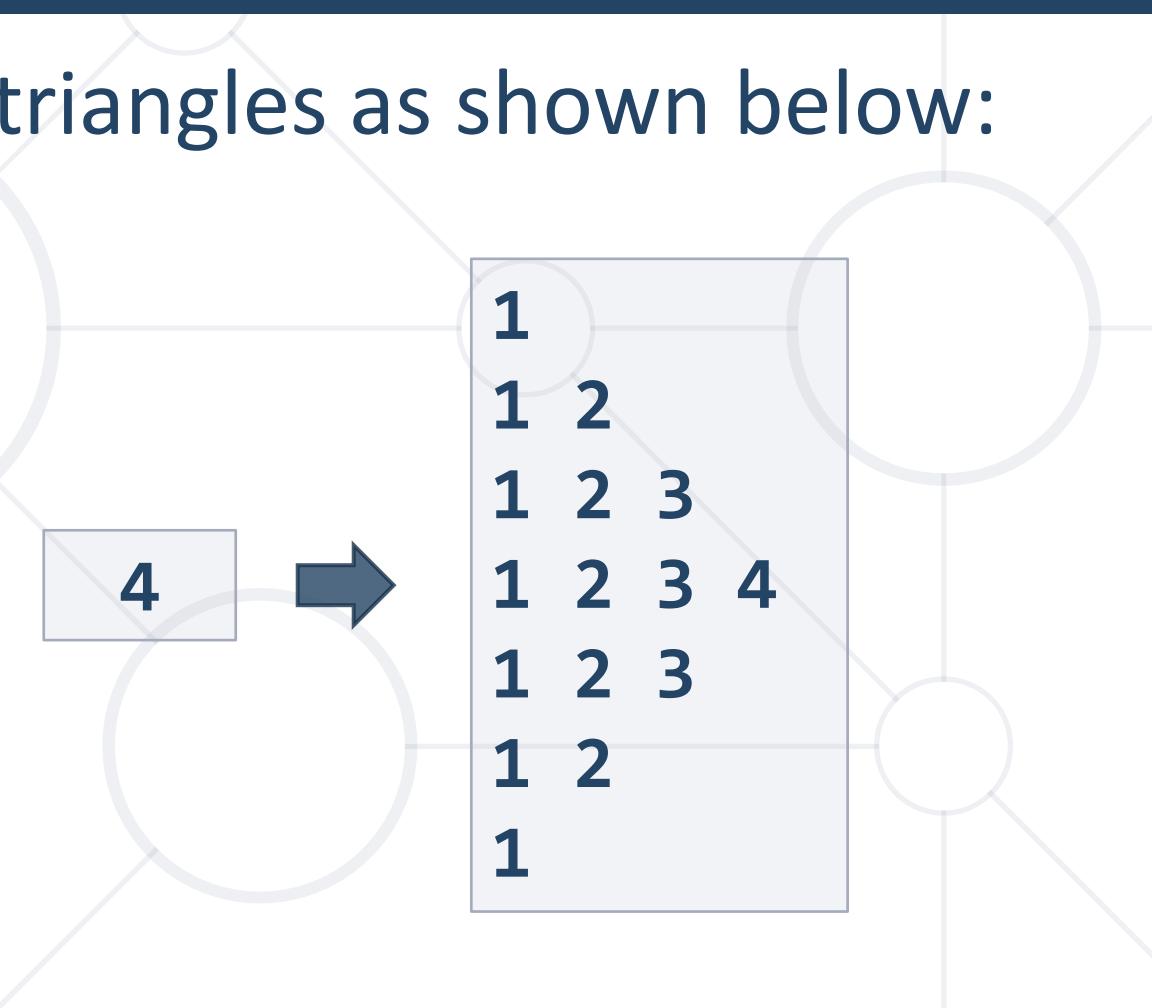
Can be skipped at method invocation

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.org/Contests/Practice/Index/1208#3

Solution: Printing Triangle

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

```csharp
static void PrintLine(int start, int end)
{
    for (int i = start; i <= end; i++)
    {
        Console.Write(i + " ");
    }
    Console.WriteLine();
}
```

Solution continues on next slide

Solution: Printing Triangle

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

```
static void PrintTriangle(int n)          Method with
{                                          parameter n
    for (int line = 1; line <= n; line++)
        PrintLine(1, line);        Lines 1...n


    for (int line = n - 1; line >= 1; line--)
        PrintLine(1, line);
}                                   Lines n-1...1
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#3

Value vs. Reference Types

Memory Stack and Heap

Value Types

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

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

Stack

i
42   (4 bytes)

ch
A   (2 bytes)

result
true   (1 byte)
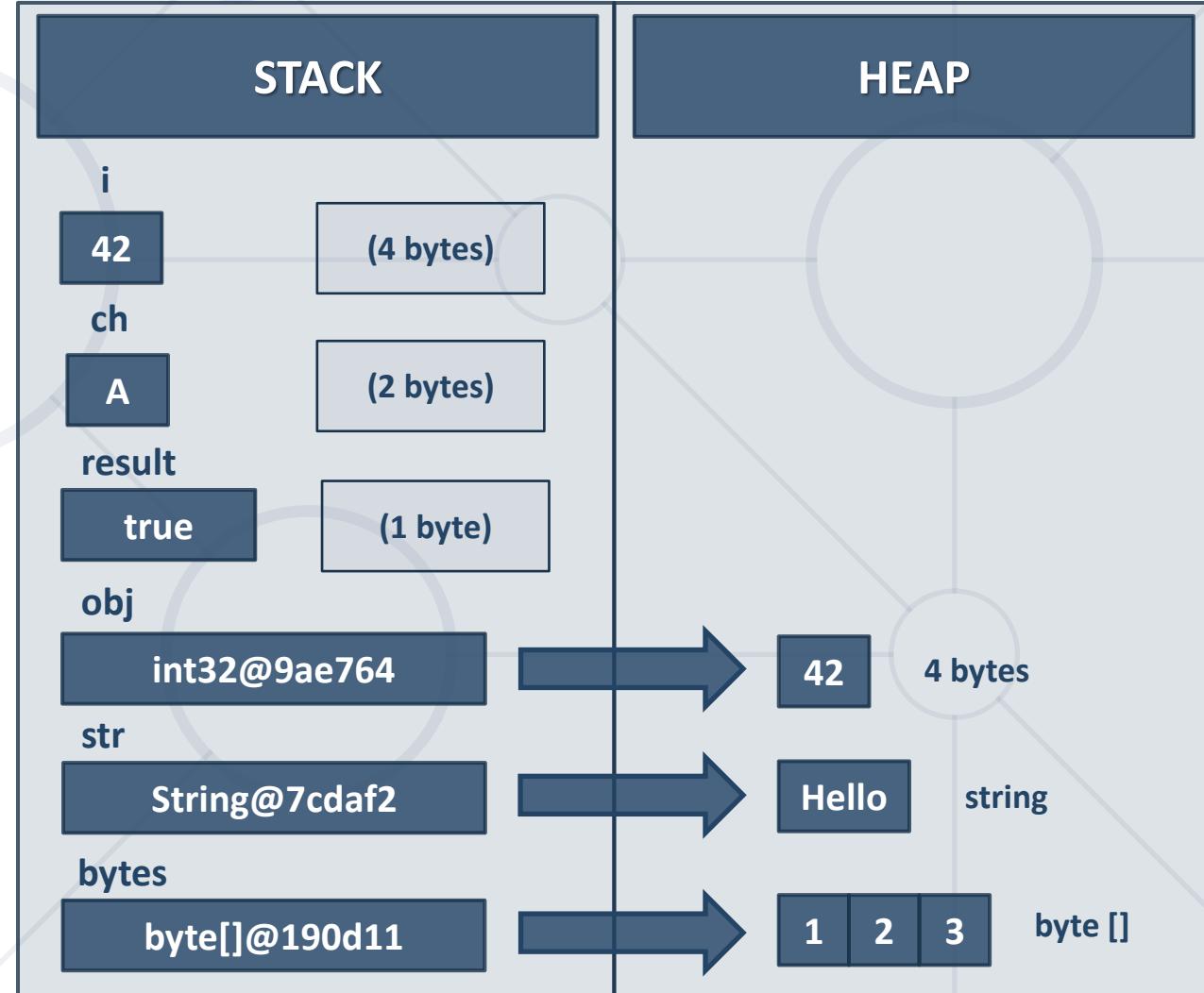
Reference Types

- **Reference type** variables hold a reference (pointer / memory address) of the value itself
  - **string**, **int[]**, **char[]**, **string[]**, **Random**
- 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';
bool result = true;
object obj = 42;
string str = "Hello";
byte[] bytes ={ 1, 2, 3 };
```

STACK		HEAP

i

| 42 | (4 bytes) |

ch

| A | (2 bytes) |

result

| true | (1 byte) |

obj

| int32@9ae764 | → | 42 | 4 bytes |

str

| String@7cdaf2 | → | Hello | string |

bytes

| byte[]@190d11 | → | 1 | 2 | 3 | byte [] |

```csharp
public static void Main() {

    int num = 5;

    Increment(number, 15);          number == 5

    Console.WriteLine(number);

}


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

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

Example: Reference Types

```
public static void Main() {

    int[] nums = { 5 };

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

    Console.WriteLine(nums[0]);

}


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

    nums[0] += value;

}                                  nums[0] == 20
```

Value vs. Reference Types

Returning Values from Methods

The Return Statement

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

- Returns the specified value

```
static string ReadFullName()
{
    string firstName = Console.ReadLine();
    string lastName = Console.ReadLine();
    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

    ```
    decimal total = GetPrice() * quantity * 1.20m;
    ```
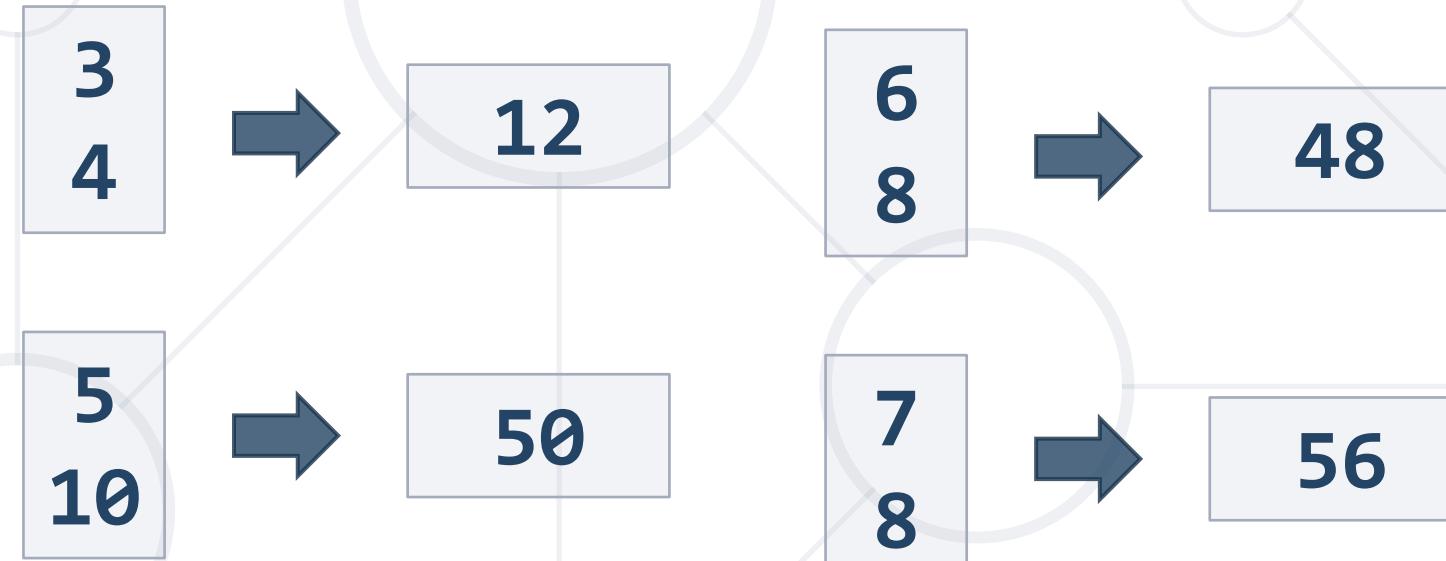
  - **Passed** to another method

    ```
    int age = int.Parse(Console.ReadLine());
    ```

Problem: Calculate Rectangle Area

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

3 4	➡	12	
5 10	➡	50	

6 8	➡	48	
7 8	➡	56	

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#5

Solution: Calculate Rectangle Area
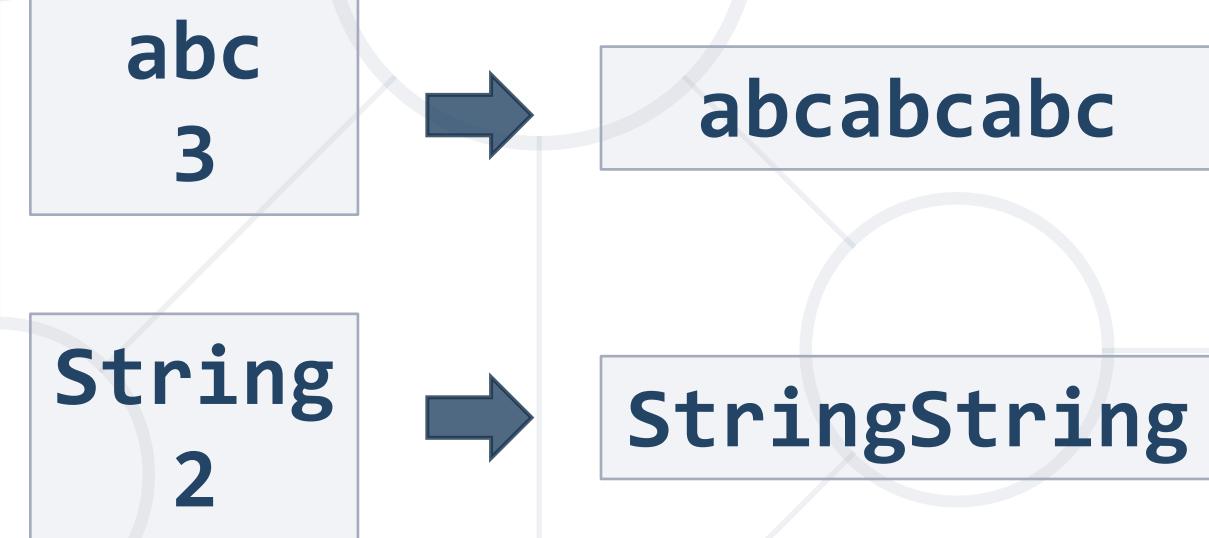
```
static void Main()
{
  double width = double.Parse(Console.ReadLine());
  double height = double.Parse(Console.ReadLine());
  double area = CalcRectangleArea(width, height);
  Console.WriteLine(area);
}
```

```
static double CalcRectangleArea(double width,double height)
{
  return width * height;
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#5

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.org/Contests/Practice/Index/1208#6

Solution: Repeat String

```csharp
static void Main()

{

    string inputStr = Console.ReadLine();

    int count = int.Parse(Console.ReadLine());


    string result = RepeatString(inputStr, count);

    Console.WriteLine(result);

}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#6

Solution: Repeat String

```csharp
private static string RepeatString(string str, int count)
{
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < count; i++)
        result.Append(str);
    return result.ToString();
}
```

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#6

Problem: Math Power

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

$$2^8 \Rightarrow 256 \qquad 3^4 \Rightarrow 81$$

```
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.org/Contests/Practice/Index/1208#7

Live Exercises

Overloading Methods

Method Signature

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

```
static void Print(string text)
{
    Console.WriteLine(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

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

```
static void Print(string text)
{
    Console.WriteLine(text);
}
```

```
static void Print(int number)
{
        Console.WriteLine(number);
}
```

```
static void Print(string text, int number)
{
    Console.WriteLine(text + ' ' + number);
}
```

Different method signatures

Signature and Return Type

- Method's return type **is not part** of its signature
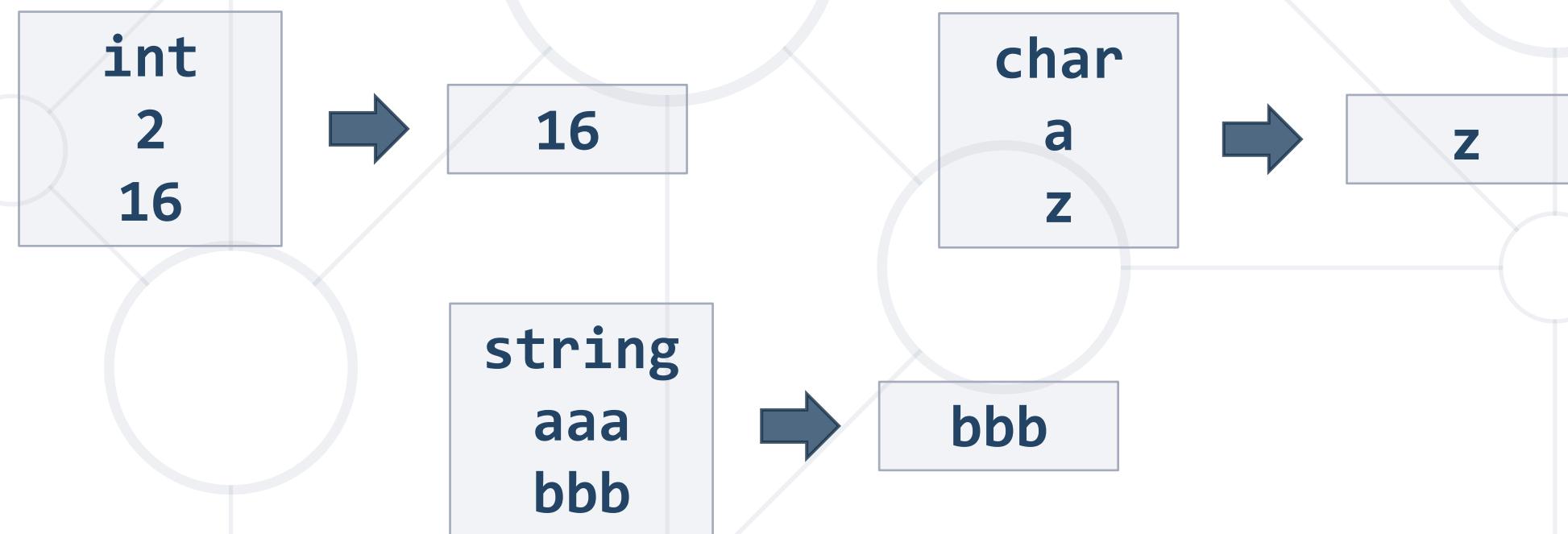
```
static void Print(string text)
{
  Console.WriteLine(text);
}
static string Print(string text)
{
  return text;
}
```

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.org/Contests/Practice/Index/1208#8
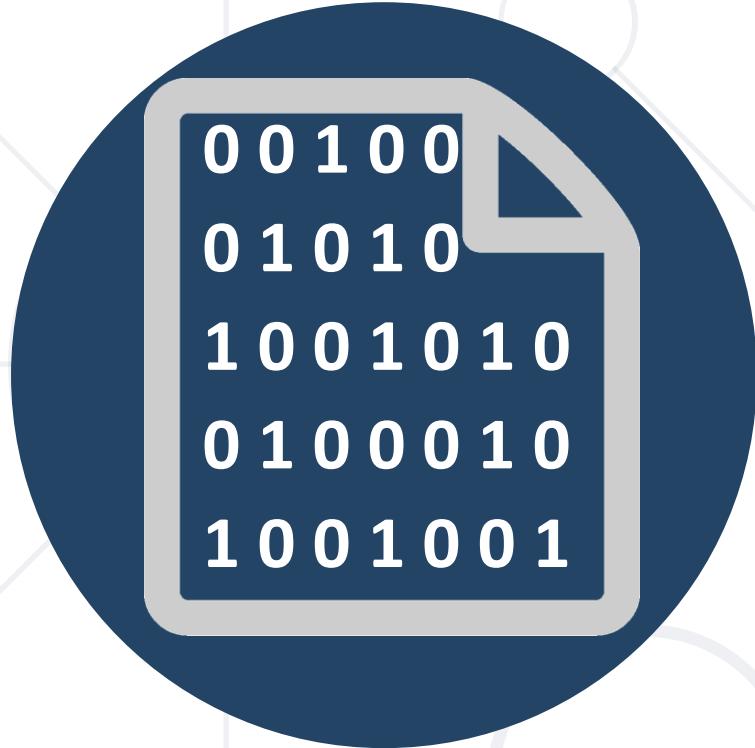
# Live Exercises

Program Execution Flow

Program Execution
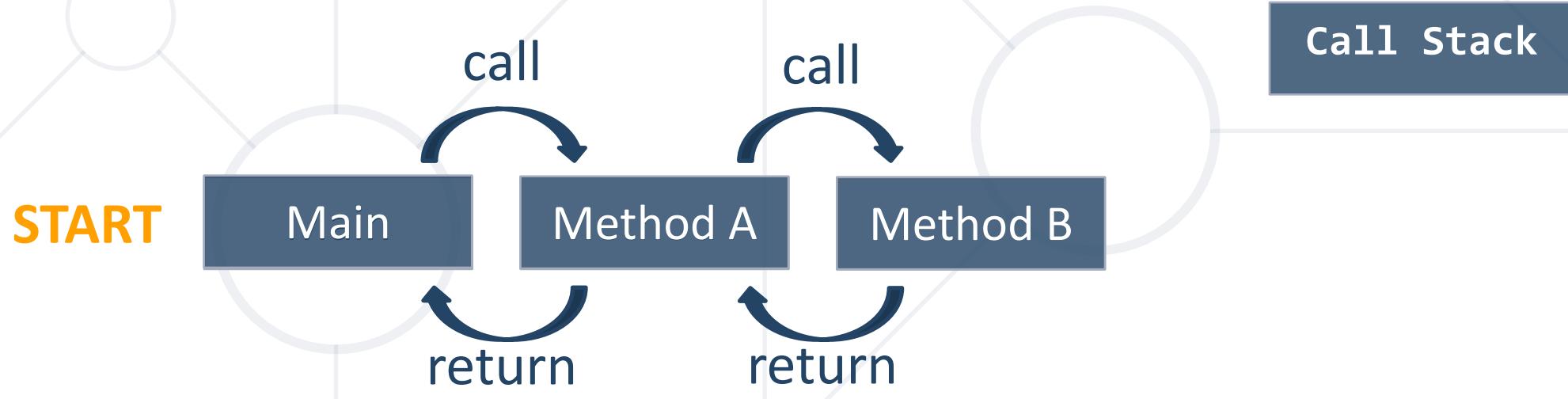
- The program continues, after a method execution completes

```csharp
static void Main()
{

    Console.WriteLine("before method executes");
    PrintLogo();
    Console.WriteLine("after method executes");
}
```

```csharp
static void PrintLogo()
{

    Console.WriteLine("Company Logo");
    Console.WriteLine("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**

Call Stack

call   call

START   Main   →   Method A   →   Method B

return   return

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:

  - You may need to use `Math.Abs()` for negative numbers

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

Check your solution here: https://judge.softuni.org/Contests/Practice/Index/1208#9

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, DirtyHack`

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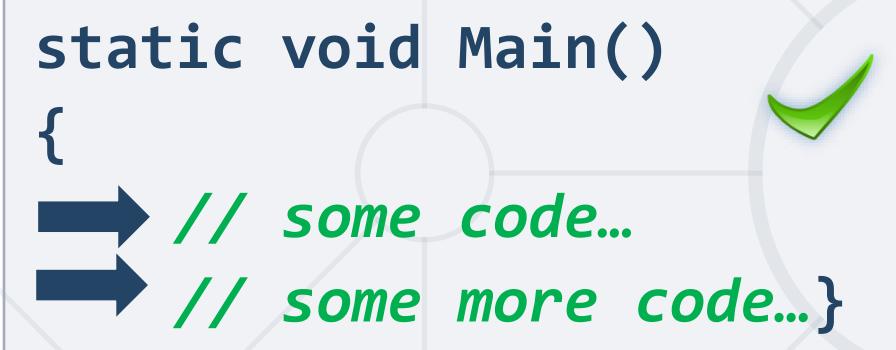
- Each method should perform a **single**, well-defined task
  - A method's name should **describe that task** in a clear and non-ambiguous way

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

```
private static void PrintReceipt()
{
    PrintHeader();
    PrintBody();
    PrintFooter();
}
```
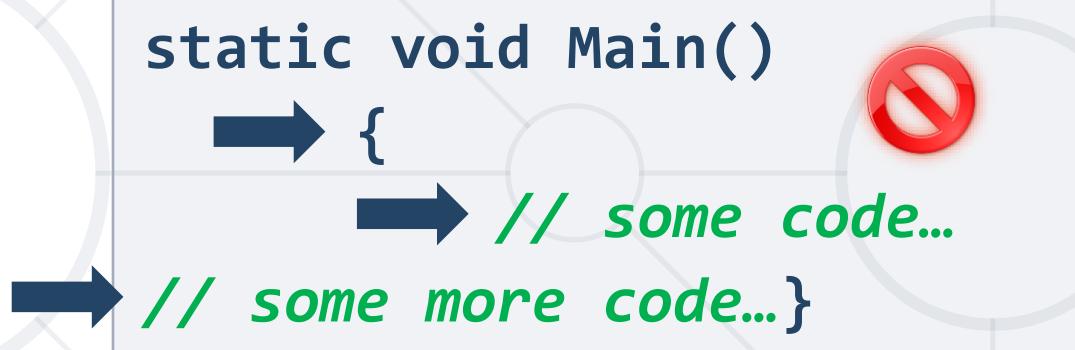
Self documenting **and easy to test**

Code Structure and Code Formatting

- Make sure to use correct **indentation**

```
static void Main()
{
    // some code…
    // some more code…}
```
✓

```
static void Main()
  {
        // some code…
// some more code…}
```
🚫
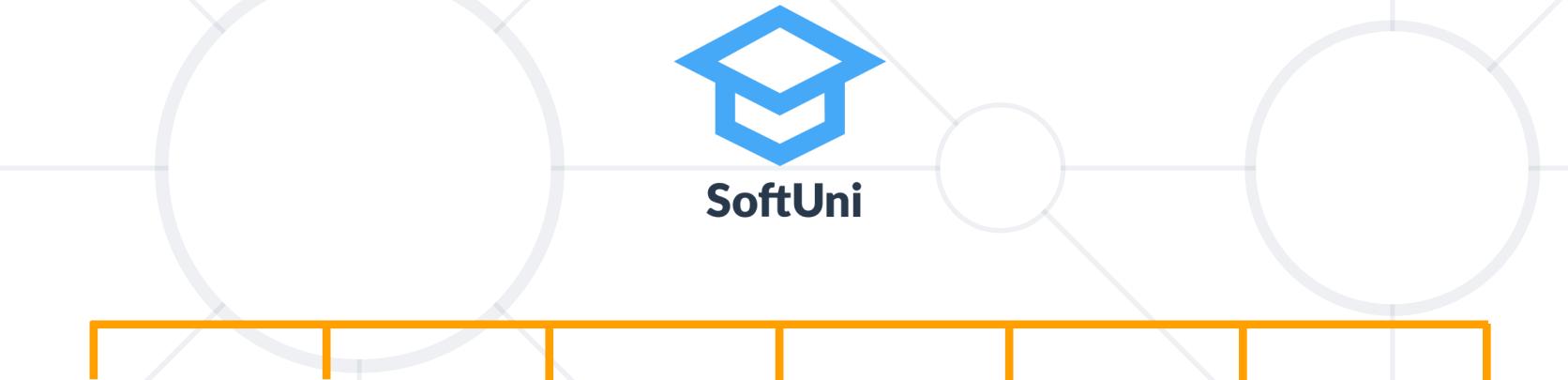
- Leave a **blank line** between **methods**, after **loops** and after **if** statements

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

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

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 Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers

  - softuni.bg, about.softuni.bg

- Software University Foundation

  - softuni.foundation

- Software University @ Facebook

  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**

- Unauthorized copy, reproduction or use is illegal

- © SoftUni – https://about.softuni.bg/

- © Software University – https://softuni.bg