



ŠKODA AUTO University

# Computer Simulation of Logistics Processes

Programming in SimTalk

Jan Fábry

04/04/2023

# Programming in SimTalk

## Aim of the lecture

- To introduce basics of SimTalk language and work with methods.



# Programming in SimTalk

## Structure of the lecture

- **Method.**
  - Object structure of method.
  - Method syntax.
  - Keywords.
  - Data types.
  - Operations.
  - Comments.
  - Conventions.
- **Debug.**
- **Watch Window.**



# Programming in SimTalk

## What do we understand by the term METHOD

- Methods are **small parts of the program**, similar to procedure or function, written in programming languages (e.g., Basic, Pascal, C++).
- In the Plant Simulation, it is programmed by the programming language “**SimTalk**”:
  - SimTalk is derived from programming language “**Eiffel**” and it is similar to other programming languages.
- Method is a basic object of Plant Simulation, and it is fully **integrated into all its objects**. The source code is processed when the simulation is started (**Init**) or terminated (**EndSim**), or during the simulation run (if it is initialized).



# Programming in SimTalk

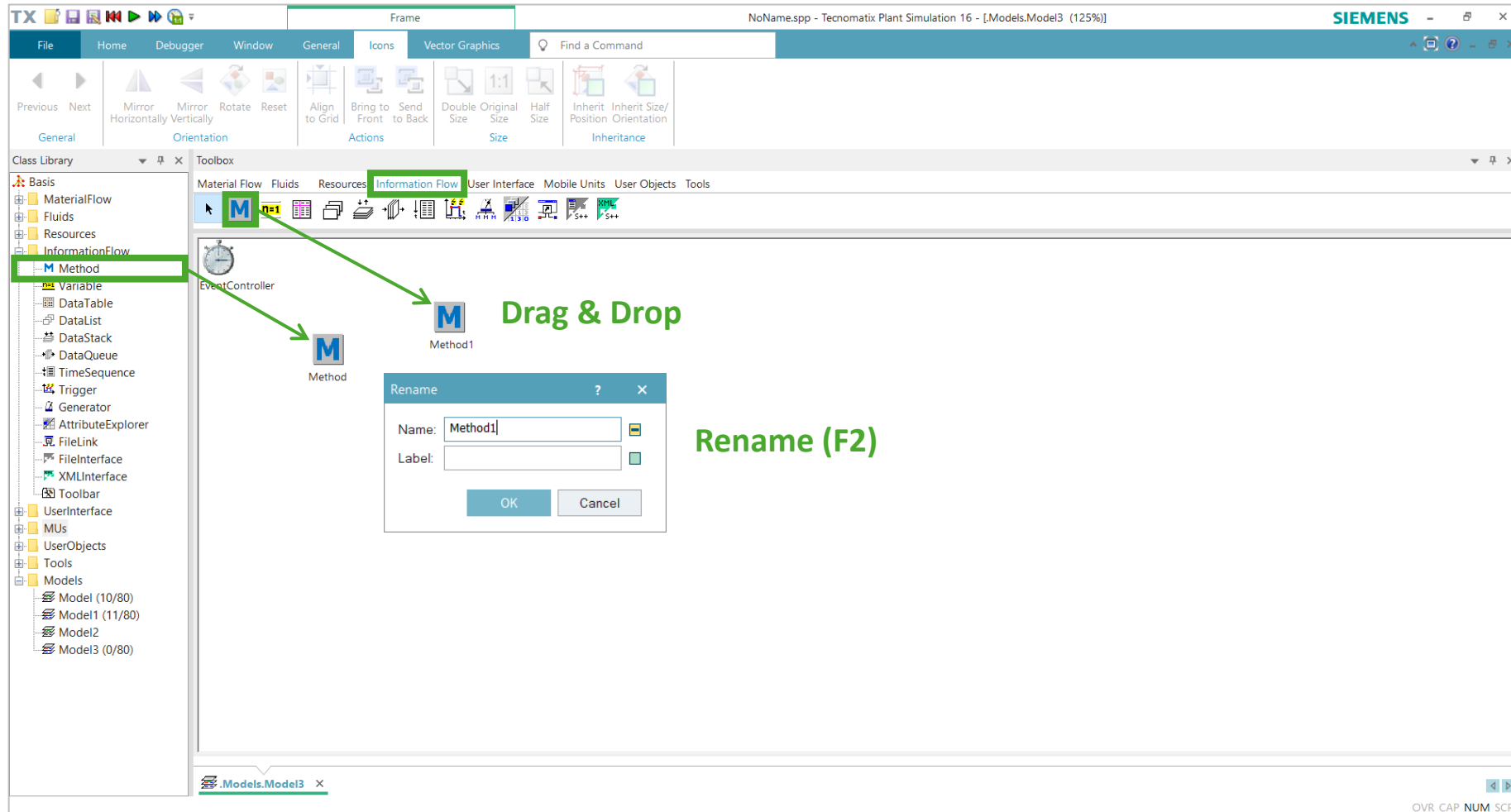
## Possibilities, usability and strengths of the methods

- Methods react in certain events **during the simulation run**.
- They ask for the conditions **defined by user**.
- Methods define **the conditions for the simulation run**.
- Methods perform **the commands**.
- They change and expand **the behavior of individual objects** of the simulation model.
- Via **dialogs**, they help to create a model offered even to inexperienced users.
- Method **usability** at each of simulation models.
- Via methods, the model becomes:
  - **Adaptable**.
  - Fully **controllable**.
  - Easily **changeable**.

# Programming in SimTalk



## Object Method



# Programming in SimTalk

## Basic method window, inheritance, saving

The screenshot shows the SimTalk software interface. The 'Edit' menu is open, and the 'Inheritance' and 'Apply Changes' buttons are highlighted with a green box. A green box contains text explaining the importance of saving changes with the 'Apply Changes' button. Another green box contains a warning about source code inheritance.

The possibility of source code inheritance out of the parent method. For own method modifications, it is necessary to disable it.

**CAUTION!**

In that case that after the writing of own source code it is the inheritance function activated, everything is rewritten. Step back is not possible!

The changes made must always be confirmed and saved with the "Apply Changes" button.



# Programming in SimTalk

## Basic method syntax

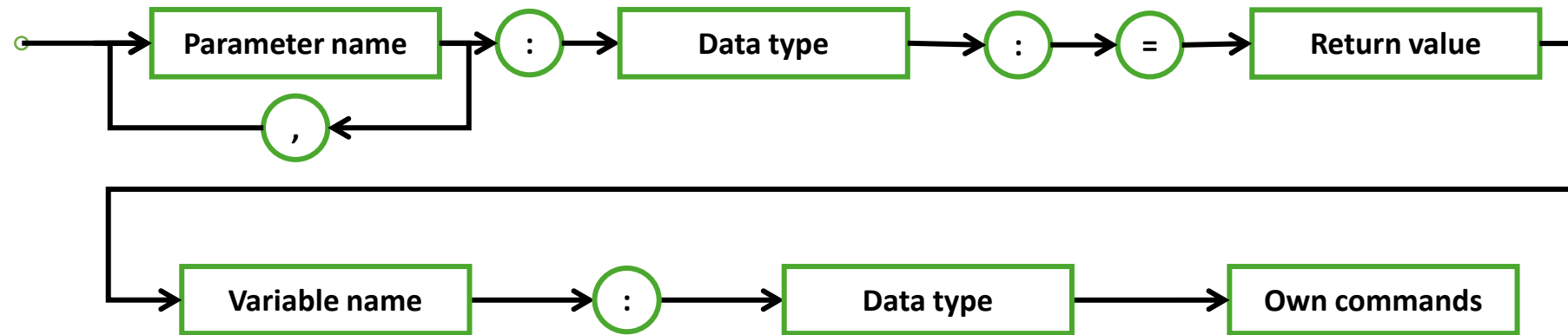
[PARAMETER]	→	<b>Additional object information</b> , which called the method, the method becomes a <b>function</b> , the number of referenced and declared parameters and data types must match.
[RESULT OF FUNCTION]	→	<b>Function result</b> (must be assigned to the triggering object by the keyword "RESULT" when returning the value).
[LOCAL VARIABLES]	→	They are defined and are available <b>only in the specified method</b> . After performing, they lose its value. It is redefined on each new call and must be declared before its use (defined by the keyword "VAR").
[OWN COMMAND]	→	Method <b>performance</b> .





# Programming in SimTalk

## Syntax diagram of method



# Programming in SimTalk

## Syntax diagram of method

- Example – function returns larger of two entered values:

```
param number1: integer, number2: integer --argument data type
-> integer --result data type

if number1 >= number2 then --finding which of the numbers is larger
    result:= number1 --if value 1 is greater than value 2 return value 1
else
    result:= number2 --otherwise return value 2
end
```

*Function call from another method*

```
-> integer --result data type
result:= mGetMax(85,25) --return the maximum of the entered values using the created method
|
```



# Programming in SimTalk

## Keywords of SimTalk

---

acceleration	downto	method	result	to
and	else	mod	return	true
any	elseif	next	root	until
array	end	not	rootfolder	var
basis	exitloop	object	self	void
boolean	false	or	speed	wait
byref	for	param	stack	waitexpired
continue	forget	pi	stopuntil	waituntil
create	if	print	string	weight
current	integer	prio	switch	when
date	length	queue	table	while
datetime	list	real	then	
div	loop	repeat	time	

---



# Programming in SimTalk

## Data type

- **Integer** integer digits (-2 147 483 648; +2 147 483 647)
- **Real** real digits
- **Time** hh:mm:ss.ss
- **Date** 1.1.1970 – 31.12.9999
- **DateTime** date including time (dd.MM.yyyy HH:mm:ss)
- **Boolean** true, false
- **String** string, it is written in upper quotation marks “ ”
- **Object** object in simulation model
- **Table** data type for embedded table



# Programming in SimTalk

## Data types

- **Length** assigned unit [m]\*
  - **Speed** assigned unit [m·sec<sup>-1</sup>]\*
  - **Acceleration** assigned unit [m·sec<sup>-2</sup>]\*  
used for “Line” and “Transporter” objects
  - **Weight** assigned unit [kg]\*
  - **Money** general variable, it uses data type REAL
  - **List** shares the built-in attributes of the object CARDFILE
  - **Stack** similar function as STACKFILE, uses LIFO
  - **Queue** built-in attributes of the object QueueFile, uses FIFO
  - **Any** universal variable, it can take on all values
- Value range:  
 $-8,9 \cdot 10^{307} \leq \text{real} \leq 8,9 \cdot 10^{307}$

!Data type TIME, LENGTH, WEIGHT, SPEED, ACCELERATION are not compatible.

\* On exit, the units are converted on the units set by the user in the **Tools > Model Settings/Preferences > Units >**.



# Programming in SimTalk

## Operations

- **Assignment of value:**
  - `<Object>.<attribute> := <new value>`
- **Logical operations:**
  - Symbols ( = , /= , > , >= , < , <= , ~= )
  - Values ( true , false )
  - Operators for connection ( and , or , not )
- **Arithmetic operators:**
  - Addition, subtraction, multiplication, division ( + , - , \* , / )
  - Function (goniometric, logarithmic, exponential)
- **Input and output operations:**
  - It is used to transfer data from input and output parameters



# Programming in SimTalk

## Logical operators

- Operators for data type **Integer**:

Symbol	Description	Result type	Example
<b>+</b>	Addition	integer	
<b>-</b>	Subtraction	integer	
<b>*</b>	Multiplication	integer	
<b>//</b>	Quotient	integer	17//5=3
<b>\%</b>	Remainder (modulo operation)	integer	17\%5=2
<b>/</b>	Division	integer, real	
<b>=</b>	Equality	boolean	
<b>/=</b>	Inequality	boolean	
<b>&gt;</b>	Greater than	boolean	
<b>&lt;</b>	Less than	boolean	
<b>&gt;=</b>	Greater than or equal to	boolean	
<b>&lt;=</b>	Less than or equal to	boolean	



# Programming in SimTalk

## Logical operators

- Operators for data type **Real and Boolean**:

Symbol	Description	Result type
+	Addition	real
-	Subtraction	real
*	Multiplication	real
/	Division	real
=	Equality	boolean
/=	Inequality	boolean
>	Greater than	boolean
<	Less than	boolean
>=	Greater than or equal to	boolean
<=	Less than or equal to	boolean
~=	About equal	boolean
<~=	Less then or about equal to	boolean
>~=	Greater than or about equal to	boolean





# Programming in SimTalk

## Logical operators

- Operators for data type **String**:

Symbol	Expression	Description	Result type	Syntax
<b>+</b>		Addition	string	
<b>=</b>		Equality	boolean	
<b>/=</b>		Inequality	boolean	
<b>==</b>		About equal (no distinguish between lower/upper case)	boolean	
	<b>strToLower</b>	all upper-case letters change to lower-case	string	<i>strToLower(&lt;string&gt;)</i>
	<b>strToUpper</b>	all lower-case letters change to upper-case	string	<i>strToUpper(&lt;string&gt;)</i>
	<b>strCopy</b>	copying of a part of string	string	<i>strcopy(&lt;string&gt;,&lt;integer1&gt;,&lt;integer2&gt;)</i>
	<b>strIncl</b>	inserting of a text into string	string	<i>strincl(&lt;string1&gt;,&lt;string2&gt;,&lt;integer&gt;)</i>
	<b>strOmit</b>	deleting of part of string	string	<i>stromit(&lt;string&gt;,&lt;integer1&gt;,&lt;integer2&gt;)</i>
	<b>strlen</b>	defines the string length-size	integer	<i>strlen(&lt;string&gt;)</i>
	<b>strlpos</b>	defines the position of text in the string	integer	<i>strpos(&lt;string1&gt;,&lt;string2&gt;)</i>



# Programming in SimTalk

## Logical operators

- Expressions of data type **Boolean** (TRUE, FALSE):
  - The result of a Boolean value when using **and**, **or**, **not**.

Variable	Connective	Variable		Result
TRUE	<b>and</b>	TRUE	→	TRUE
TRUE	<b>and</b>	FALSE	→	FALSE
FALSE	<b>and</b>	TRUE	→	FALSE
FALSE	<b>and</b>	FALSE	→	FALSE
TRUE	<b>or</b>	TRUE	→	TRUE
TRUE	<b>or</b>	FALSE	→	TRUE
FALSE	<b>or</b>	TRUE	→	TRUE
FALSE	<b>or</b>	FALSE	→	FALSE
	<b>not</b>	TRUE	→	FALSE
	<b>not</b>	FALSE	→	TRUE



# Programming in SimTalk

## Functions

- Functions for data type **String**:
  - **strToLower**
    - `print strToLower("ABCDEF")`      -- returns "abcdef"
  - **strToUpper**
    - `print strToUpper("abcdef")`      -- returns "ABCDEF"
  - **strCopy**
    - `print strCopy("abcdef",2,3)`      -- returns "bcd"
    - `print strCopy("abcdef",4,10)`      -- returns "def"
    - `print strCopy("abcdef",-1,4)`      -- returns "ab" (-1 and 0 are included)



# Programming in SimTalk

## Functions

- Functions for data type **String**:
  - **strIncl**
    - `print strIncl("XYZ","abcdef",3)` -- returns "abXYZcdef"
    - `print strIncl("XYZ","abcdef",-1)` -- returns "XYZabcdef"
    - `print strIncl("XYZ","abcdef",1)` -- returns "XYZabcdef"
    - `print strIncl("XYZ","abcdef",20)` -- returns "abcdefXYZ"
  - **strOmit**
    - `print strOmit("abcdef",3,2)` -- returns "abef"
    - `print strOmit("abcdef",0,3)` -- returns "cdef" (0 is included)
  - **strlen**
    - `print strlen("abcd")` -- returns 4
  - **strpos**
    - `print strlpos("b","abcdfb")` -- returns 2



# Programming in SimTalk

## Functions

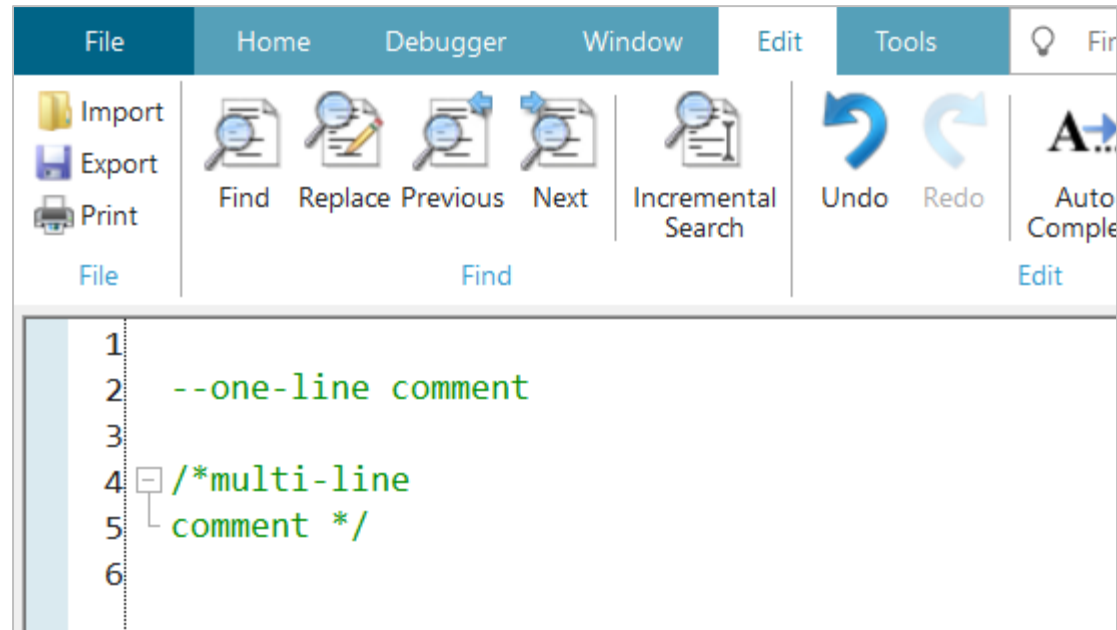
- Change of **data type**:

Function	Argument	Result
to_str	(<...>)	string
num_to_bool	(<integer>)	boolean
bool_to_num	(<boolean>)	real
str_to_num	(<string>)	real
str_to_bool	(<string>)	boolean
str_to_time	(<string>)	time
str_to_date	(<string>)	date
str_to_datetime	(<string>)	datetime
str_to_length	(<string>)	length
str_to_weight	(<string>)	weight
str_to_speed	(<string>)	speed
str_to_obj	(<string>)	object

# Programming in SimTalk

## Comment in method

- It is used as **remark** on the formulated instructions in the method. Instructions are clear and understandable even for another person. In the method itself, comments **improve the orientation**.
- The text is highlighted in **the green color** in the method.
- Creating a comment :
  - One-line** comment " -- text ".
  - Multi-line** comment "/\* text \*/".



The screenshot shows the SimTalk IDE interface. The top menu bar includes File, Home, Debugger, Window, Edit, Tools, and a search icon. Below the menu bar, there are several toolbars: File (Import, Export, Print), Find (Find, Replace, Previous, Next, Incremental Search), and Edit (Undo, Redo, Auto Complete). The main code editor area shows a list of lines (1-6) on the left. The code content is as follows:

```
1
2  --one-line comment
3
4  /*multi-line
5  comment */
6
```



# Programming in SimTalk

## Method conventions

- In Plant Simulation's help, the following spelling **conventions** are applied for easier reading:
  - **The formulations** in the methods are written in **proportional font** (`courier`).
  - **Attribute names** start with **upper case** letter (e.g. `Buffer.Capacity`).
  - **Methods names** start with **lower case** letter, in case that the method is compounded from two words, the second one starts with upper case letter (e.g., `bodyshop.Removeobserver`).
  - **Keywords** are written in the **blue** font.
  - **Comments** are written in the **green** font.



# Programming in SimTalk

## Debug in the method

- **Immediate stop** in the current step of the performed method because of the message reasons.
- **Reasons for Debug message:**
  - Automatically on an error that disables the method to process.
  - Manually for tracking the method and searching the error.
  - Purposefully for messages in case of not performed events (e.g., if `@.Move()` then else debug; end;).
- **Activation of Debug:**
  - Automatically for an error occurrence during the simulation run.
  - Set of breakpoint (class/instance breakpoint)
  - Manually during method performance (in menu, the function key F8).
  - By writing word “Debug” straight into the method.
- While activation of Debug, **the window** displaying the method, which caused this situation, **is opened**.



# Programming in SimTalk



## Window Debug

Defining local variables ←

Class Breakpoint ←

Own Commands ←

Actual position of the performed method ←

Watch window ←

```
var a: real := 5
var b: real := 10
var S: real := a * b
debug
print S
```

Name	Value
a	5.0
b	10.0
S	50.0

# Programming in SimTalk

## Watch window

- It **informs** about actual initializing and calling object, variable status etc.
- Subwindow "**Variables**" and "**Anonymous Identifiers**":
  - Displaying of variable parameters.
- Column "**Value**":
  - Displaying of the actual status, i.e. value of parameters.
  - "VOID" – status/value is not available (empty parameter).

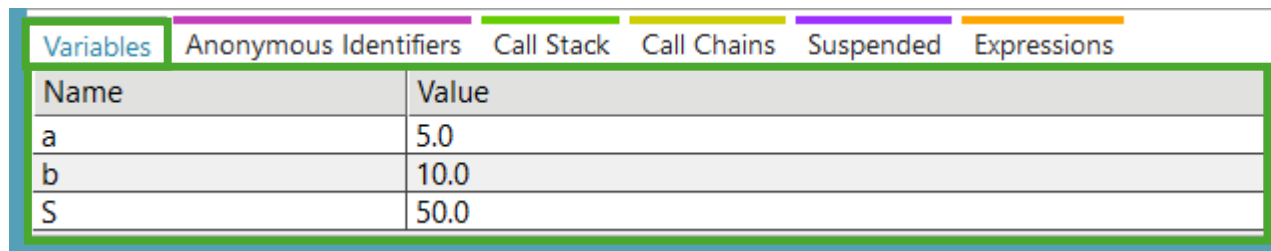
Name	Value
@	VOID
?	VOID
current	.Models.Model
self	.Models.Model.Method
root	.Models.Model
rootfolder	VOID

Name	Value
a	5.0
b	10.0
S	50.0

# Programming in SimTalk

## Watch window

- “ @ ”
  - Anonymous identifier for mobile units (object of MU library).
  - It is used as the initiator, so the one that runs the method.
- “ ? ”
  - Anonymous identifier for static objects (objects of Material Flow library).
  - Usually, it is the object, calling the method.
- “ S, a, b ”
  - Defined local variables, which are processed by the method.



Name	Value
a	5.0
b	10.0
S	50.0



ŠKODA AUTO University

# Thank you for attention

**Jan Fábry**

Department of Production, Logistics and Quality Management

✉ [fabry@savs.cz](mailto:fabry@savs.cz)

🌐 [www.janfabry.cz](http://www.janfabry.cz)

[www.savs.cz](http://www.savs.cz)