

Lecture 11: Set Operations, Data Treatment (I/O), Object-Oriented Programming

B0B17MTB, BE0B17MTB – MATLAB

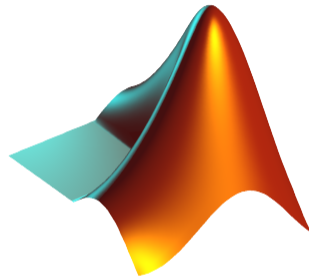
Miloslav Čapek, Viktor Adler, *et al.*

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@fel.cvut.cz

December 18, 2023
Winter semester 2023/24



1. Set Operations
2. Error Treatment
3. Data Import and Export
4. Data Types categorical and table
5. Object-Oriented Programming





Set Operations

- ▶ There exist following operations (operators) in MATLAB applicable to arrays or individual elements:
 - ▶ arithmetic (part #1),
 - ▶ relational (part #3),
 - ▶ logical (part #3),
 - ▶ `set` (part #11),
 - ▶ bit-wise (>> doc `bit-wise`).

- ▶ Set operations are applicable to vectors, matrices, arrays, cells, strings, tables,...

- ▶ Mutual sizes of these structures are usually not important.

Function	Description
<code>intersect</code>	intersection of two sets
<code>union</code>	union of two sets
<code>setdiff</code>	difference of two sets
<code>setxor</code>	exclusive OR of two sets
<code>unique</code>	unique values in a set
<code>sort</code>	sorting
<code>sortrows</code>	row sorting
<code>ismember</code>	is an element member of a set?
<code>issorted</code>	is a set sorted?



Set Operations: intersect and union

- ▶ Intersection of sets: `intersect`.

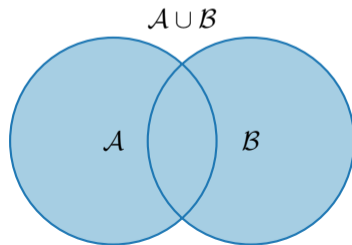
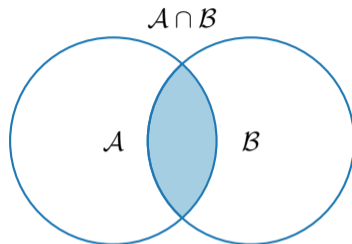
- ▶ **Example:** intersection of a matrix and a vector:

```
>> A = [1 -1; 3 4; 0 2];
>> b = [0 3 -1 5 7];
>> c = intersect(A, b)
% c = [-1; 0; 3]
```

- ▶ Union of sets: `union`.

- ▶ **Example:** All set operations can be carried out row-wise (in that case the number of columns has to be observed):

```
>> A = [1 2 3; 4 5 1; 1 7 1];
>> b = [4 5 1];
>> C = union(A, b, 'rows')
% C = [1 2 3; 1 7 1; 4 5 1]
```





Set Operations: setdiff and setxor

- ▶ Intersection of a set and complement of another set: `setdiff`.

- ▶ **Example:** All set operations return more than one output – we get the elements as well as the indexes:

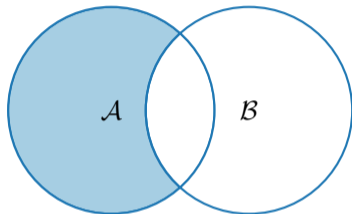
```
>> A = [1 1; 3 NaN];
>> B = [2 3; 0 1];
>> [C, ai] = setdiff(A,B)
% C = NaN, ai = 4, i.e.: C = A(ai)
```

- ▶ Exclusive intersection (XOR): `setxor`.

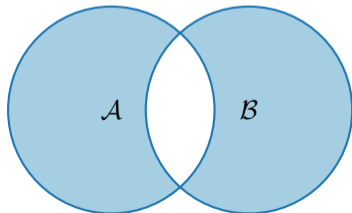
- ▶ **Example:** All set operations can be carried out either as `'stable'` (not changing the order of elements) or as `'sorted'`:

```
>> A = [5 1 0 4];
>> B = [1 3 5];
>> [C, ia, ib] = setxor(A, B, 'stable')
% C = [0 4 3], ia = [3; 4], ib = [2]
```

$$A \setminus B = A \cap B^c$$



$$A \oplus B = \overline{A \cap B}$$





Set Operations: unique

- ▶ Selection of unique elements of an array: `unique`.

- ▶ **Example:** Set operations are also applicable to arrays not (exclusively) containing numbers:

```
>> A = {'Joe', 'Tom', 'Sam'};
>> B = {'Tom', 'John', 'Karl', 'Joe'};
>> C = unique([A B])
% C = {'John', 'Karl', 'Joe', 'Sam', 'Tom'}
```

$$\begin{bmatrix} c & b & a & d \\ a & c & b & a \\ c & c & d & b \end{bmatrix} \subseteq \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

- ▶ It is possible to combine all above mentioned techniques.

- ▶ **Example:** Row-wise listing of unique elements of a matrix including indexes:

```
>> A = round(rand(10, 3)).*mod(10:-1:1, 3) '
>> [C, ai, ci] = unique(sum(A, 2), 'rows', 'stable')
```

- ▶ Interpret the meaning of the above code? Is the `rows` parameter necessary?



Set Operations I.

- ▶ Consider three vectors **a**, **b**, **c** containing natural numbers $x \in \mathbb{N}$ so that:
 - ▶ vector **a** contains all primes up to (and including) 1000,
 - ▶ vector **b** contains all sorted even numbers up to (and including) 1000,
 - ▶ vector **c** is complement of **b** in the same interval (also sorted).
- ▶ Find vector **v** so that $\mathbf{v} = \mathbf{a} \cap (\mathbf{b} + \mathbf{c})$.
 - ▶ What elements does **v** contain?
 - ▶ How many elements are there in **v**?

```

v =
Columns 1 through 18
   3   7  11  19  23  31  43  47  59  67  71  79  83 103 107 127 131 139
Columns 19 through 36
 151 163 167 179 191 199 211 223 227 239 251 263 271 283 307 311 331 347
Columns 37 through 54
 359 367 379 383 419 431 439 443 463 467 479 487 491 499 503 523 547 563
Columns 55 through 72
 571 587 599 607 619 631 643 647 659 683 691 719 727 739 743 751 787 811
Columns 73 through 87
 823 827 839 859 863 883 887 907 911 919 947 967 971 983 991
ans =
    87
  
```

600



Set Operations I.

- ▶ Consider three vectors **a**, **b**, **c** containing natural numbers $x \in \mathbb{N}$ so that:
 - ▶ vector **a** contains all primes up to (and including) 1000,
 - ▶ vector **b** contains all sorted even numbers up to (and including) 1000,
 - ▶ vector **c** is complement of **b** in the same interval (also sorted).
- ▶ Find vector **v** so that $\mathbf{v} = \mathbf{a} \cap (\mathbf{b} + \mathbf{c})$.
 - ▶ What elements does **v** contain?
 - ▶ How many elements are there in **v**?

```
v =
Columns 1 through 18
   3   7  11  19  23  31  43  47  59  67  71  79  83 103 107 127 131 139
Columns 19 through 36
 151 163 167 179 191 199 211 223 227 239 251 263 271 283 307 311 331 347
Columns 37 through 54
 359 367 379 383 419 431 439 443 463 467 479 487 491 499 503 523 547 563
Columns 55 through 72
 571 587 599 607 619 631 643 647 659 683 691 719 727 739 743 751 787 811
Columns 73 through 87
 823 827 839 859 863 883 887 907 911 919 947 967 971 983 991
ans =
    87
```




Set Operations II.b

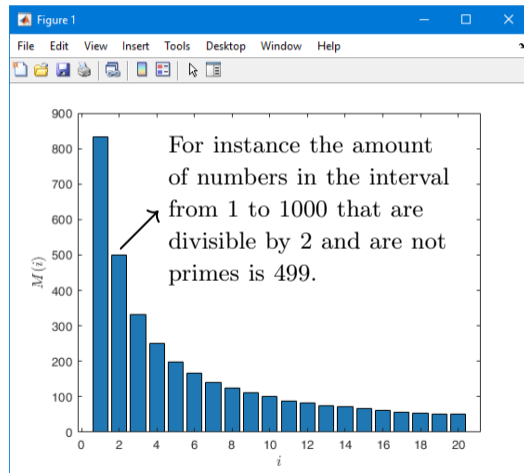
- ▶ Write previous exercise as a script:

```
%% script depicts number of integers from 1 to 1000 in  
% dependence on division remainders  
clear; clc;  
N = 1000;  
a = primes(N);  
b = 2:2:N;  
c = setdiff(1:N, b);  
w = setdiff(union(b, c), a);  
% ...  
    m = sum(not(mod(w, 3)));  
% ...
```

- ▶ Modify the script in the way to calculate how many elements of **w** are divisible by numbers 1 to 20.
 - ▶ Use for instance `for` loop to get the result.
 - ▶ Plot the results using `bar` function.



Set Operations II.c





Set Operations III.a

- ▶ Radio relay link operates at frequency of 80 GHz at 20 km distance with 64-QAM modulation.
 - ▶ Phase stability of $\pm 0.5^\circ$ is required for sufficiently low bit error rate without using synchronization and coding.
 - ▶ That corresponds to the change of distance between antennas equal to $\pm 5 \mu\text{m}$.
 - ▶ The statistics of link distance with normal distribution containing 10^6 elements can be generated as:

```
L = 20e3; % length of path
deviation = 5e-6; % standard deviation
N = 1e6; % number of trials
distances = L + randn(1, N)*deviation; % random distances
```

- ▶ How many times is the distance L contained in the vector distances?
- ▶ How many unique elements are there in distances?
- ▶ Can the distribution be considered continuous?





Catching Errors I.

- ▶ Used particularly in the cases where unexpected event can occur:
 - ▶ in general operations with files (reading, saving),
 - ▶ evaluation of encapsulated code (function `eval`, `assignin`),
 - ▶ working with variables, properties of which (*e.g.*, `size`) is not yet known,
 - ▶ evaluation of code related to an object that may not exist anymore (GUI).

```
try
    % regular piece of code
catch
    % code that is evaluated if the regular code failed
end
```

- ▶ It is possible (and is recommended) to use an identifier of the error.



Catching Errors II.

- ▶ Error identifier can be used to decide what to do with the error.
 - ▶ **Example:** In the case of multiplication error caused by different size of vectors, it is possible to display a warning.
 - ▶ Also, the error can be later raised again either by evoking the last error occurred or as a new error with its own identifier.

```
try
    A = [1 1 1];
    B = [1 1];
    c = A.*B;
catch exc
    if strcmp(exc.identifier, 'MATLAB:dimagree')
        disp('Mind the vector size!');
    end
    % throw(exc); % local stack shown
    % rethrow(exc); % complete stack shown
end
```



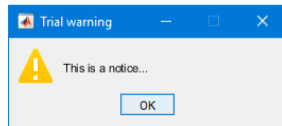
Warning Message in MATLAB

- ▶ Warning message in MATLAB is displayed using function warning.

```
a = 1e3;  
if a > 1e2  
    warning('Input coefficient has to be smaller than 10!');  
end
```

- ▶ The function is used by MATLAB, therefore, it is possible to temporarily deactivate selected internal warnings.
- ▶ Function lastwarn returns last warning activated.
- ▶ It is advantageous to use function warndlg with GUI (it just show a window, not throws the warning).

```
f = warndlg('This is a notice..', ...  
           'Trial warning', 'modal');
```





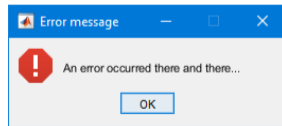
Error Message in MATLAB

- ▶ Error message (in red color) is displayed using function `error`.

```
a = 100;  
if a > 10  
    error('Input has to be equal of smaller than 10!');  
end
```

- ▶ Terminates program execution.
 - ▶ Identifier can be attached.
- ▶ It is advantageous to use function `errordlg` with GUI (it just show a window, not throws the error).

```
f = errordlg('An error occurred there and there..',  
...  
    'Error message', 'modal');
```





Launching External Programs

- ▶ Rarely used.
- ▶ External programs are launched using the exclamation mark (!).
 - ▶ The whole line after the “!” is processed as operation system command:

```
>> !calc
```

- ▶ If you don't want to interrupt execution of Matlab by the launch, add “&”:

```
>> !calc &  
>> !notepad notes.txt &
```

- ▶ It is possible to run MATLAB with several ways:

```
>> doc matlab Windows  
>> doc matlab UNIX
```



Data Import and Export I.

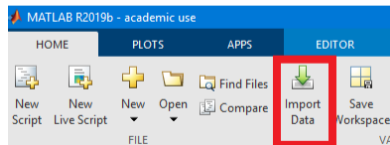
- ▶ MATLAB supports wide range of file formats:
 - ▶ *e.g.*, mat, txt, xls, jpeg, bmp, png, wav, avi, and others,
 - ▶ for details see MATLAB → Data Import and Analysis → Data Import and Export → Supported File Formats for Import and Export.
 - ▶ Packages exist for work with, for instance, dwg and similar formats.
 - ▶ It is possible to read a general file containing ASCII characters as well.

- ▶ In this course we shall see how to:
 - ▶ read data from file, read image, read files line by line (see **Lecture 6**),
 - ▶ store in file, write in file,
 - ▶ import from Excel,
 - ▶ export to Excel.



Data Import and Export II.

- ▶ Following can be applied to whole group of formats:
 - ▶ Home → Import Data,
 - ▶ command `uiimport` and proceed with a following interface,
 - ▶ file drag and drop to MATLAB Workspace window.
- ▶ For storing in various formats see following functions.
 - ▶ `save`, `writematrix`, `writetable`, `imwrite`, `audiowrite`, ...





Functions `cd`, `pwd`, `dir`

- ▶ Function `cd` changes current folder:

```
cd FD % jumps into FD folder
cd    % lists current folder
cd .. % jumps up one directory
cd \  % jumps up to root
```

- ▶ Function `pwd` identifies current folder.
- ▶ Function `dir` lists current folder content.
- ▶ For other functions (`mkdir`, `rmdir`, ...) see MATLAB Documentation.



Completion/Parsing of File Paths: `fullfile`, `fileparts`

- ▶ Build full file name from parts with function `fullfile`,
 - ▶ *i.e.*, insert automatically correct separator (Windows: `\\`, Unix: `/`).
- ▶ Use whenever you work with paths.
- ▶ To get the correct separator for current platform use function `filesep`.

- ▶ Parse full path into file path, file name, and extension with function `fileparts`.

```
myPath = {'Data', 'Corrected'};  
myFile = 'measuredData';  
myExt = '.mat';  
  
f = fullfile(myPath{:}, [myFile myExt])  
  
[myPath2, myFile2, myExt2] = fileparts(f)
```

```
f =  
    'Data\Corrected\measuredData.mat'  
myPath2 =  
    'Data\Corrected'  
myFile2 =  
    'measuredData'  
myExt2 =  
    '.mat'  
>>
```



Import from Excel

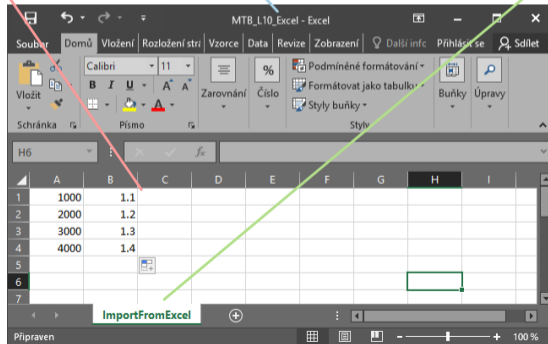
- ▶ Use function `readmatrix` to import into Excel.
 - ▶ Alternatively, use aforementioned function `uiimport`.

```
Data = readmatrix('MTB_L10_Excel.xlsx', 'Sheet', 'ImportFromExcel',
                 'Range', 'A1:B4')
```

cells

file name

sheet

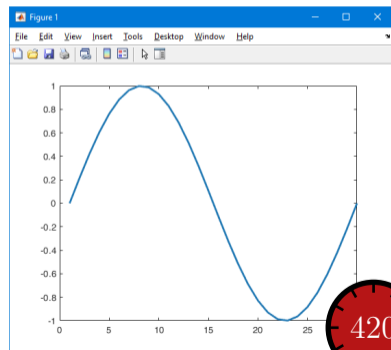


```
Data =
    1.0e+03 *
    1.0000    0.0011
    2.0000    0.0012
    3.0000    0.0013
    4.0000    0.0014
>>
```



Import from Excel

- ▶ Read all numerical data from Excel file `measurement1.xlsx` on course's webpage.
 - ▶ Thereafter, plot dependence of values in column values on values in column experiment.
 - ▶ Verify the size of data read.





Export to Excel

- ▶ Function `xlswrite` is used to export data from MATLAB to Excel.
 - ▶ **Example:** Write data `fx` in file `file.xlsx` in sheet `Sheet1` in line 1 starting with column A.

```
fx = 1:10;  
writematrix(fx, 'file.xlsx');
```

- ▶ **Example:** Write data `fx` in file `file2.xlsx` in sheet `NewSheet` in column B starting with line 1.

```
fx = 1:10;  
writematrix(fx, 'file2.xlsx', 'Sheet', 2, 'Range', 'B1');
```



Export to Excel

- ▶ Evaluate function

$$f(x) = \cos(x) + \frac{\cosh(x)}{10}$$

on the interval $x \in [-\pi, \pi]$ with step 0.01.

- ▶ Resulting variables x and $f(x)$ write to file `Excel_file.xlsx` in the 1st sheet, variable x is in column A, variable $f(x)$ is in column B.
- ▶ Verify whether data written in the sheet are correct.



Saving and Loading Binary Data (Reminder)

- ▶ Numerical data can be saved *en block*:
 - ▶ Notice the vector transposition.
 - ▶ tsv extension here because of TikZ.

- ▶ Load binary data from file line by line:

- ▶ Save binary data into file line by line:
- ▶ See also: [Lecture 6](#).

```
x = 0:0.01:2*pi;
fx = sin(x) .* cos(x).^2 + x.^(1/3);
Data = [x.' fx.'];
save('myData.tsv', 'Data', '-ascii');
```



```
fid = fopen('myData.tsv');
while ~feof(fid)
    thisLine = fgetl(fid) % your data...
end
fclose(fid);
```

```
fid = fopen('myData2.txt', 'w+');
Data = {'this is the first line', ...
        'this is the second line'};
for iLine = 1:length(Data)
    fprintf(fid, '%s\n', Data{iLine});
end
fclose(fid);
```



Data Type categorical

- ▶ Array of qualitative data with values from finite set of discrete non-numerical data.
 - ▶ Array of non-numerical values corresponding to a category (*e.g.*, to the category “mean of transport” correspond following values: scooter, wheelbarrow, ...).
 - ▶ Values can be specified by name (*e.g.*, values 'r', 'g', 'b', they can be an attribute for name 'red', 'green', 'blue').
 - ▶ categorical arrays has its own icon in MATLAB Workspace.

Workspace	
Name ▲	Value
 A	3x3 cell
 B	3x3 categorical



Creation of categorical Arrays

- ▶ Creation of categorical array from an arbitrary array of values (e.g., cell array of strings):

```
A = {'r' 'b' 'g'; ...
     'g' 'r' 'b'; ...
     'b' 'r' 'g'} % cell array of strings
B = categorical(A) % categorical arrays
categories(B)     % listing of individual categories
```

- ▶ Wide range of tools for combining, adding, removing, renaming, arranging, ...
- ▶ For more see >> doc `categorical arrays`

```
A =
3x3 cell array

    {'r'}    {'b'}    {'g'}
    {'g'}    {'r'}    {'b'}
    {'b'}    {'r'}    {'g'}

B =
3x3 categorical array

    r    b    g
    g    r    b
    b    r    g

ans =
3x1 cell array

    {'b'}
    {'g'}
    {'r'}
```



Advantages of categorical Arrays

- ▶ More natural arranging of data by names.
 - ▶ Note: as in numerical arrays, logical operator `eq (==)` is used to compare strings in categorical arrays instead of function `strcmp ()` used with strings.
- ▶ Mathematical arranging of strings.
 - ▶ Setting “size” in other than alphabetical manner (*e.g.*, `small < medium < large`):


```
allSizes = {'medium', 'large', 'small', ...  
           'small', 'medium', 'large', ...  
           'medium', 'small'};  
valueset = {'small', 'medium', 'large'};  
sizeOrd = categorical(allSizes, valueset, 'Ordinal', true);  
comparison = sizeOrd > fliplr(sizeOrd)
```

- ▶ Memory is used efficiently to store data.
 - ▶ Data in memory is not stored as string.
 - ▶ Only categories are stored as string in memory.



Data Type table

- ▶ Array in form of a table that enables to have columns of various data types and sizes (similar to `cell` array).
 - ▶ Each column has to have the same number of lines (same as matrix).
 - ▶ Tables have its own icon in MATLAB Workspace.
- ▶ For more see `doc >> table`.

Workspace	
Name ▲	Value
 T	4x2 table



Creation of table

- Created by inserting individual vectors as columns of the table:

```
name = {'Miloslav'; 'Viktor'; 'Michal'; 'Vit'};
matlabSemester = [3; 3; 2; 1];
favoriteDrink = categorical({'b'; 'm'; 'w'; 'w'}, ...
    {'w'; 'm'; 'b'}, ...
    {'wine'; 'milk'; 'beer'});

T = table(matlabSemester, favoriteDrink, 'RowNames', name)
```

```
T =
    4x2 table

    matlabSemester    favoriteDrink
    _____    _____
    Miloslav         3         beer
    Viktor           3         milk
    Michal           2         wine
    Vit              1         wine

>>
```




Advantages of table

- ▶ Well-structured data,
- ▶ access to data via numerical and name indexing,
 - ▶ *e.g.*, listing all “Smiths” in the table and display their “age”,
- ▶ possibility to store metadata in table’s properties,
 - ▶ *e.g.*, for column “age” it is possible to set unit to “year”.



Classes in MATLAB

- ▶ MATLAB supports OOP since version 2008a.
- ▶ Classes are defined in separate m-files.
- ▶ Real-time update of objects in the workspace when classes are changed.
- ▶ No requirement for memory allocation or deallocation.
- ▶ This course does not provide a general OOP theory.

```
classdef student
    % Class properties
    properties
        name % Full name of student
        number % Identification number
        information % Details about student's results
    end
    % Class methods
    methods
        function obj = student ()
            % Constructor for student object
        end

        function out = getStudentInfo(obj)
            % Operation above the structure properties
            out = obj.information;
        end
    end
end
```



Properties Validation

- ▶ Properties can be validated through the functionality similar to arguments block.

```
property (dim1, dim2) class {fcn1, fcn2} = value
property name      dimensions  data type validation functions default value
```

- ▶ Example:

```
properties
    name (1,1) string {mustBeText} = string.empty
    number (1,1) double {mustBeReal, mustBePositive}
    information (1,1) studentInfo {mustBeNonempty}
end
```

- ▶ Validation functions can check correct data type (`mustBeNumeric`), compare inputs to some predefined values (`mustBeGreaterThan`), or if it is valid value (`mustBeMember`).
- ▶ See `>> doc property validation`



Constructor and Destructor Methods

- ▶ Special methods that are called during the creation of an object and its termination.
- ▶ Constructor
 - ▶ initialize object's properties,
 - ▶ same name as of the class,
 - ▶ initialized object returned as output argument.
- ▶ Destructor
 - ▶ called when object is deleted,
 - ▶ implemented in method `delete`,
 - ▶ memory is cleaned automatically.

```
function obj = student (name,  
number)  
    obj.name = name;  
    obj.number = number;  
end
```

```
function delete(obj)  
    obj.terminateStudent();  
    obj.cleanRecords();  
end
```



Properties Attributes

- ▶ Properties have attributes specifying their behavior against the user.

```
properties (attrName1 = attrValue1, attrName2 = attrValue2)
    name % Full name of student
    number % Identification number
    information % Details about student's results
end
```

- ▶ Access attributes `Access`, `SetAccess`, and `GetAccess` specify how the user can access properties to modify them.
 - ▶ `public` – access from any code (default).
 - ▶ `private` – access from the defining class or its subclasses.
 - ▶ `protected` – access only by members of the defining class.
 - ▶ `immutable` – set only by the constructor (only for `SetAccess`).
- ▶ `Hidden` sets the property's visibility to the user.
- ▶ `Constant` ensures immutability across all class instances.
- ▶ `>> doc property attributes`



Methods Attributes

- ▶ Similar functionality as in the case of properties attributes.

```
methods (attrName1 = attrValue1)
    function out = getStudentInfo(obj)
        ...
    end
end

methods (attrName2 = attrValue2)
    function out = getStudentCreditCard(obj)
        ...
    end
end
```

- ▶ Attributes `Access`, `Hidden`, and `Abstract` are similar as in case of properties.
- ▶ Attribute `Static` specifies methods without dependence on class properties. `Sealed` specifies the method's behavior for inheritance.



Inheritance

- ▶ Classes can be defined in a hierarchical sense. Useful when multiple classes share the same properties and methods.
- ▶ Properties and methods can be redefined or called from the superclass. Attributes set additional behavior.

```
classdef vehicle
    properties
        numberOfWheels
    end

    methods
        function obj = vehicle(n)
            obj.numberOfWheels = n;
        end
    end
end
```

```
classdef car < vehicle
    properties
        brand
    end

    methods
        function obj = car(n, brand)
            obj@vehicle(n);
            obj.brand = brand;
        end
    end
end
```



handle Class

- ▶ Specific class, which sets the behavior of an object to be a reference variable, not a value variable.
- ▶ handle class allows definition of events.
 - ▶ Triggering actions based on the status of objects.
- ▶ All graphical objects inherit from handle.

- ▶ handle subclass

```
classdef myClass < handle
    properties
        value
    end

    methods
        function setVal(obj, val)
            obj.value = val;
        end
    end
end
```

- ▶ Modified objects do not have to be assigned to variables.

```
v = valueClas
v.changeValue(10)
```




Method Overriding

- ▶ Class can define methods with the same name as basic MATLAB functions.
- ▶ In the case of the subclass, the superclass methods implementation can also be overridden if it is not protected by an attribute `sealed`.
- ▶ Implementation of method overriding standard function does not cause problems to other occurrences.
- ▶ **Example:** Depending on the object type, the same function `plot` can visualize data, graphs, and fitted curves or can be implemented specially for your object.

```
classdef student
    properties
        picture = imread("peppers.png");
    end

    methods
        function obj = plot()
            imshow(obj.picture)
        end
    end
end
```



Advanced Class Example

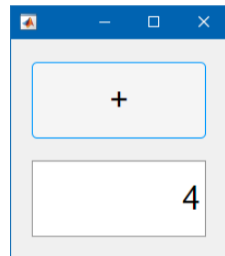
► Object-oriented programming.

► Run it by app = myApp

```
classdef myApp < handle
    properties (Access = private)
        hFig
        hEdit
        hButt
        result
    end

    methods
        function obj = myApp()
            obj.result = 0;
            obj.hFig = uifigure('Position', 200*ones(1, 4));
            obj.hEdit = uieditfield(obj.hFig, 'numeric', ...
                'Position', [20, 20 160 70], 'Value', obj.result);
            obj.hButt = uibutton(obj.hFig, 'Text', '+', ...
                'Position', [20, 110, 160, 70]);
            set([obj.hEdit, obj.hButt], 'FontSize', 30);
            obj.hButt.ButtonPushedFcn = @(src, event)obj.increment();
        end
    end

    methods (Access = private)
        function increment(obj)
            obj.result = obj.result + 1;
            obj.hEdit.Value = obj.result;
        end
    end
end
```



Questions?

B0B17MTB, BE0B17MTB – MATLAB
matlab@fel.cvut.cz

December 18, 2023
Winter semester 2023/24

This document has been created as a part of B(E)0B17MTB course.
Apart from educational purposes at CTU in Prague, this document may be reproduced, stored, or transmitted only with the prior permission of the authors.

Acknowledgement: Filip Kozák, Pavel Valtr, Michal Mašek, and Vít Losenický.