



Einführung in die Programmierung Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 2



- Assignments
 - One assignment per week
 - Will be put online Monday (around 18:00)
 - Should be handed in within nine days (Wednesday, before 23:59)
- Testat
 - You have to hand in $n - 1$ out of n assignments
 - Must include the last one
 - Show **serious** effort
 - You have to hand in two mock exams
 - Military service or illness -> contact assistant
- Group mailing list
 - Is everybody subscribed (got an email)?



- Give you the intuition behind object-oriented (OO) programming
- Teach you about formatting your code
- Differentiate between
 - feature declaration and feature call
 - commands and queries
- Understand feature call chains
- Get to know the basics of EiffelStudio



- The main concept in Object-Oriented programming is the concept of **Class**.
- Classes are pieces of software code meant to model concepts, e.g. "student", "course", "university".
- Several classes make up a program in source code form.
- Objects are particular occurrences ("instances") of concepts (classes), e.g. "student Reto" or "student Lisa".
- A class **STUDENT** may have zero or more instances.

Classes and objects (continued)



- Classes are like templates (or molds) defining status and operations applicable to their instances.
- A sample class *STUDENT* can define:
 - A student's status: id, name and birthday
 - Operations applicable to all students: subscribe to a course, register for an exam.
- Each instance (object) of class *STUDENT* will store a student's name, id and birthday and will be able to execute operations such as subscribe to a course and register for an exam.
- Only operations defined in a class can be applied to its instances.



- A feature is an operation that may be applied to all the objects of a class.

- **Feature declaration vs. feature call**

- You declare a feature when you write it into a class.

```
set_name (a_name: STRING)  
    -- Set `name` to `a_name`.  
  
    do  
        name := a_name  
    end  
name: STRING
```

- You call a feature when you apply it to an object. The object is called the **target** of this feature call.
 - *a_person.set_name ("Peter")*
 - Arguments, if any, need to be provided in feature calls.
 - *computer.shut_down*
 - *computer.shut_down_after (3)*

Features: Exercise

Hands-On

- Class *BANK_ACCOUNT* defines the following operations:
 - *deposit (a_num: INTEGER)*
 - *withdraw (a_num: INTEGER)*
 - *close*
- If *b: BANK_ACCOUNT* (*b* is an instance of class *BANK_ACCOUNT*) which of the following feature calls are possible?
 - *b.deposit (10)* ✓
 - *b.deposit* ✗
 - *b.close* ✓
 - *b.close ("Now")* ✗
 - *b.open* ✗
 - *b.withdraw (100.50)* ✗
 - *b.withdraw (0)* ✓

Class text



```
class PREVIEW
```

Class name

```
feature
```

Feature declaration

```
explore
```

Comment

```
-- Explore Zurich.
```

Feature

```
do
```

body

```
central_view.highlight  
zurich_map.animate
```

Feature names

```
end
```

```
end
```

Instructions

Style rules



Class names are in upper-case

Use tabs, not spaces, to highlight the **structure** of the program: it is called **indentation**.

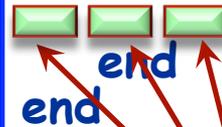
For feature names, use full words, not abbreviations.

Always choose identifiers that clearly identify the intended role

Use words from natural language (preferably English) for the names you define

For multi-word identifiers, use underscores

```
class
  PREVIEW
feature
  explore
    -- Explore Zurich.
  do
    central_view.highlight
    zurich_map.animate
  end
end
```



Another example



```
class  
  BANK_ACCOUNT
```

```
feature
```

```
  deposit (a_sum: INTEGER)  
    -- Add `a_sum` to the account.
```

```
  do
```

```
    balance := balance + a_sum
```

```
  end
```

```
  balance: INTEGER
```

```
end
```

Routine

Within comments, use ` and ' to quote names of arguments and features. This is because they will be taken into account by the automatic refactoring tools.

Attribute

The **state** of the object is defined by the values of its attributes



➤ Commands

- Modify the state of objects
- Do not have a return value
- May or may not have arguments
- Examples: register a student to a course, assign an id to a student, record the grade a student got in an exam
- ... other examples?

➤ Queries

- Do not modify the state of objects
- Do have a return value
- May or may not have arguments
- Examples: what is the age of a student? What is the id of a student? Is a student registered for a particular course?
- ... other examples?

Exercise: query or command?

Hands-On

- What is the balance of a bank account?
- Withdraw 400 CHF from a bank account
- Who is the owner of a bank account?
- Who are the clients of a bank whose total deposits are over 100,000 CHF?
- Change the account type of a client
- How much money can a client withdraw at a time?
- Set a minimum limit for the balance of accounts
- Deposit 300 CHF into a bank account



"**Asking** a question **shouldn't change** the answer"

i.e. a query

Query or command?



class *DEMO*

feature

command

procedure_name (*a1*: *T1*; *a2*, *a3*: *T2*)

-- *Comment*

do

...

end

- no result
- body

query

function_name (*a1*: *T1*; *a2*, *a3*: *T2*): *T3*

-- *Comment*

do

Result := ...

Predefined variable
denoting the result

end

- result
- body

query

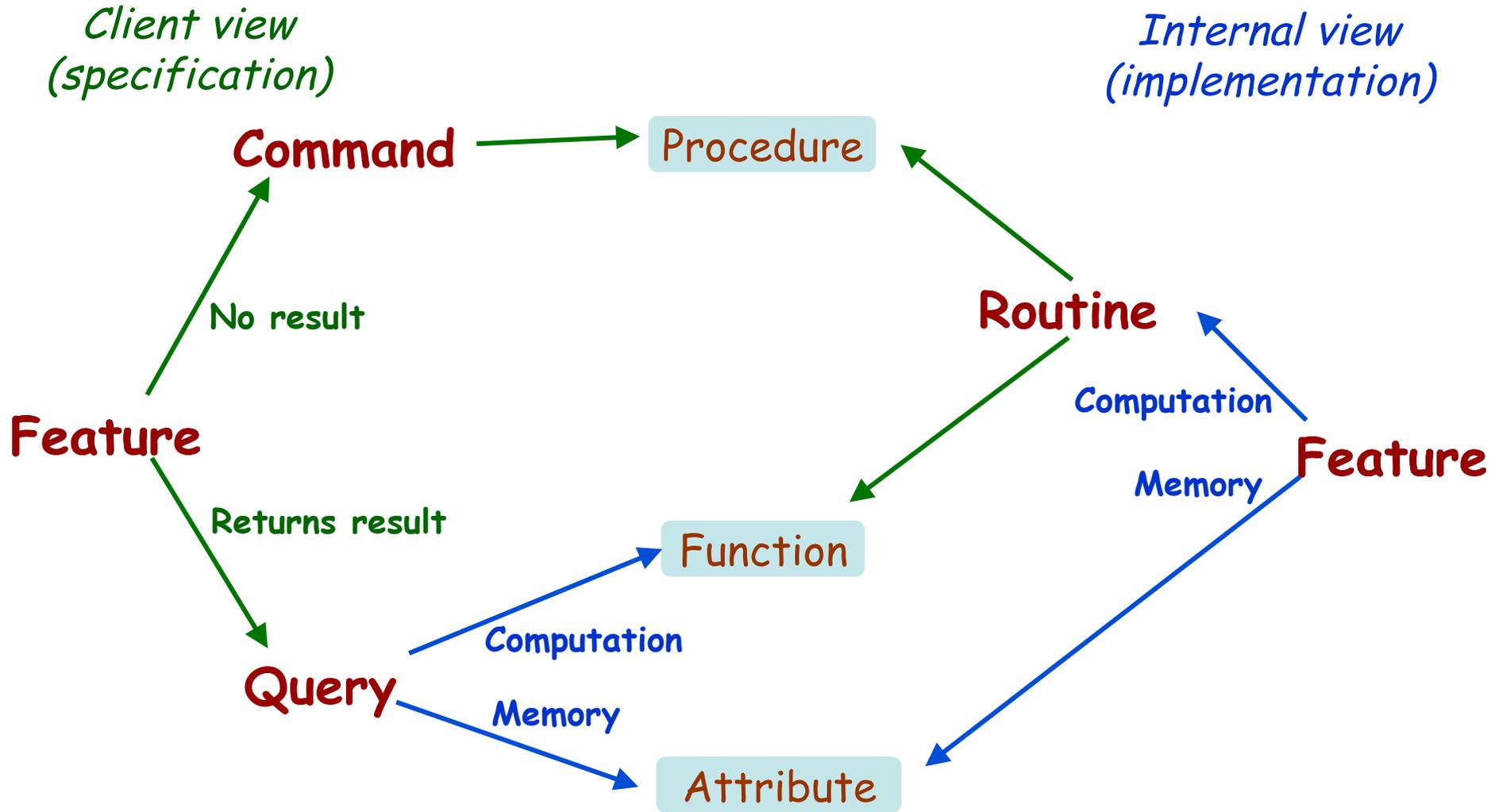
attribute_name: *T3*

-- *Comment*

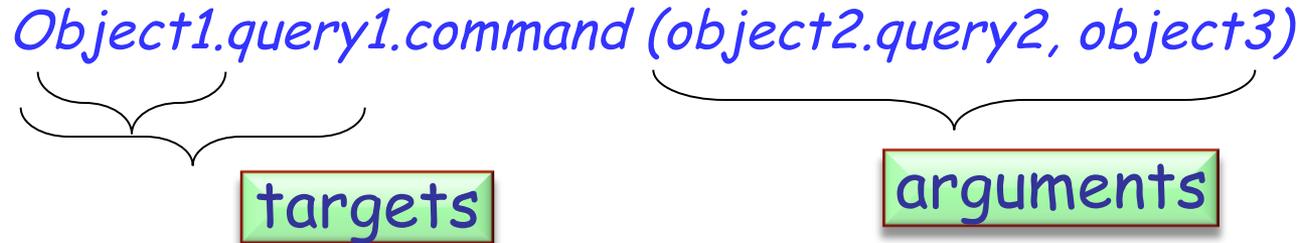
- result
- no body

end

Features: the full story



General form of feature call instructions



- Targets and arguments can be query calls themselves.

Hands-On

- Where are *query1*, *query2* defined?
- Where is *command* defined?

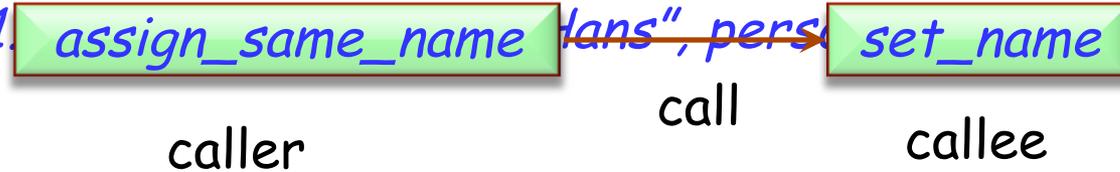
Qualified vs. unqualified feature calls

- All features have to be called on some **target**.
- The **current object** of a feature is the object on which the feature is called. (what's the other name for this object?)
 - A **qualified** feature call has an explicit target.
 - An **unqualified** feature call is one whose target is the current object. The target is left out for convenience.

```
assign_same_name (a_name: STRING; a_other_person: PERSON)
  -- Set `a_name`
  `a_other_person`.
  do
    a_other_person
    set_name
  end
```



```
person1. assign_same_name "ans", person2. set_name
```



caller call callee

- EiffelStudio is a software tool (IDE) to develop Eiffel programs.

Integrated Development Environment

- Help & Resources

- Online guided tour: in EiffelStudio help menu
- <http://eiffel.com/developers/presentations/>
- <http://www.eiffel.com/>
- <http://dev.eiffel.com/>
- <http://docs.eiffel.com/>
- <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-367.pdf>

Components



- editor
- context tool
- clusters pane
- features pane
- compiler
- project settings
- ...

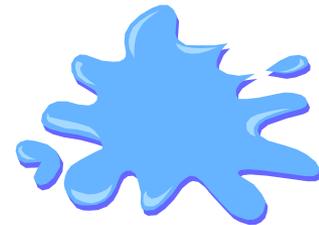


- Syntax highlighting
- Syntax completion
- Auto-completion (CTRL+Space)
- Class name completion (CTRL+SHIFT+Space)
- Smart indenting
- Block indenting or unindenting (TAB and SHIFT+TAB)
- Block commenting or uncommenting (CTRL+K and SHIFT+CTRL+K)
- Infinite level of Undo/Redo (reset after a save)
- Quick search features (first CTRL+F to enter words then F3 and SHIFT+F3)
- Pretty printing (CTRL+SHIFT+P)

Compiler highlights



- Melting: uses quick incremental recompilation to generate bytecode for the changed parts of the system. Used during development (corresponds to the button "Compile").
- Freezing: uses incremental recompilation to generate more efficient C code for the changed parts of the system. Initially the system is frozen (corresponds to "Freeze...").
- Finalizing: recompiles the entire system generating highly optimized code. Finalization performs extensive time and space optimizations (corresponds to "Finalize...")





- The system must be melted/frozen (finalized systems cannot be debugged).
- Setting and unsetting breakpoints
 - An efficient way consists in dropping a feature in the context tool.
 - Alternatively, you can select the flat view
 - Then click on one of the little circles in the left margin to enable/disable single breakpoints.
- Use the toolbar debug buttons to enable or disable all breakpoints globally.



- Run the program by clicking on the Run button.
- Pause by clicking on the Pause button or wait for a triggered breakpoint.
- Analyze the program:
 - Use the call stack pane to browse through the call stack.
 - Use the object tool to inspect the current object, the locals and arguments.
- Run the program or step over (or into) the next statement, or out of the current one.
- Stop the running program by clicking on the Stop button.



If EiffelStudio happens to crash:

- You should submit an official bug by pressing the button appearing together with the crash
- Login: [ethinfo1](#), Password: [ethinfo1](#)

How to submit a bug 1: submit bug



- + base
- + base_pr
- project

ion
n.
here
fel World

EiffelStudio Error

Internal EiffelStudio Exception

An internal failure occurred. If this happens even after relaunching EiffelStudio, perform a clean recompilation.

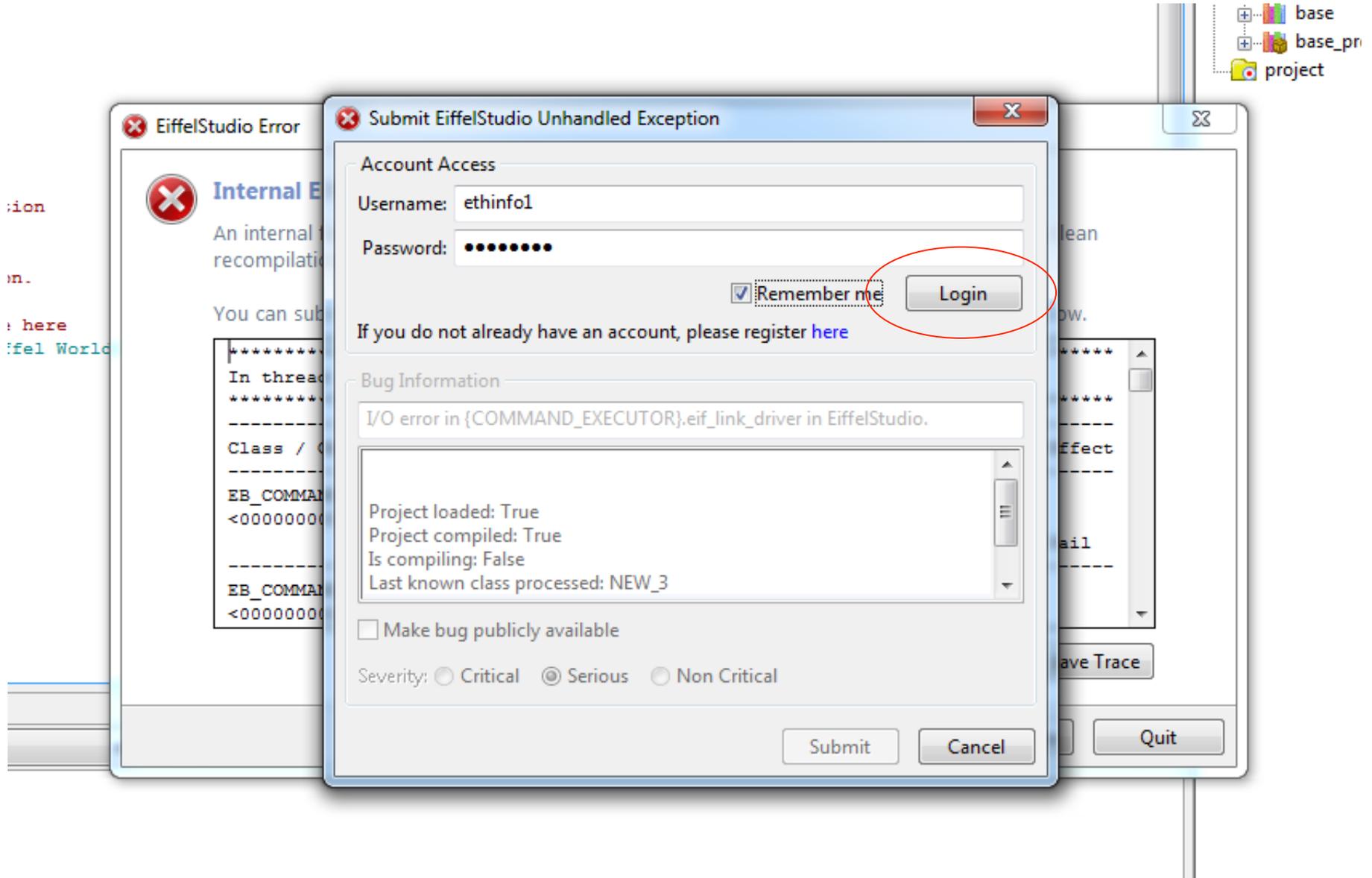
You can submit a bug report at <http://support.eiffel.com> or use the Submit Bug button below.

```
***** Thread exception *****
In thread      Root thread      0x0 (thread id)
*****
-----
Class / Object  Routine      Nature of exception  Effect
-----
EB_COMMAND_EXECUTOR eif_link_driver  Invalid argument:
<000000000314E608> (From COMMAND_EXECUTOR)
                                I/O error.          Fail
-----
EB_COMMAND_EXECUTOR eif_link_driver
<000000000314E608> (From COMMAND_EXECUTOR)
```

Submit Bug **Save Trace**

Ignore **Restart Now** **Quit**

How to submit a bug 2: login



How to submit a bug 3: submit



base
base_pre
project

EiffelStudio Error

Internal Error

An internal error occurred during recompilation.

You can submit this bug to the EiffelStudio team.

In thread *****

Class / Object

EB_COMMAND
<00000000

EB_COMMAND
<00000000

Submit EiffelStudio Unhandled Exception

You are currently logged in as ethinfo1 [Log out](#)

Bug Information

I/O error in {COMMAND_EXECUTOR}.eif_link_driver in EiffelStudio.

Project loaded: True
Project compiled: True
Is compiling: False
Last known class processed: NEW_3
Last status message: Degree 6: Examining System: base

Make bug publicly available

Severity: Critical Serious Non Critical

Submit Cancel