Introduction to Programming

Lab Session 1

(With material from the ETH Zurich course “Introduction to Programming”)

August 23, 2016

INNOPOLIS UNIVERSITY

News

Assignment 1:

- It is already published in Moodle.
- To be handed on September 20th, 2016 (via Moodle).

News

Assignment 1:

- It is already published in Moodle.
- To be handed on September 20th, 2016 (via Moodle).

Grading scheme:

- Assignments (20%)
- Mid Term (40%)
- Final Exam (40%)

In this Lab

- 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.

Classes and Objects

- 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.

Features

- 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

Features: Exercise

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?

Features: Exercise

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)

Features: Exercise

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)

Features: Exercise

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	X
b.close	
b.close (“Now”)	
b.open	
b.withdraw (100.50)	
b.withdraw (0)	

Features: Exercise

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	X
b.close	✓
b.close (“Now”)	
b.open	
b.withdraw (100.50)	
b.withdraw (0)	

Features: Exercise

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	X
b.close	✓
b.close (“Now”)	X
b.open	
b.withdraw (100.50)	
b.withdraw (0)	

Features: Exercise

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	*X*
b.close	✓
b.close (“Now”)	*X*
b.open	*X*
b.withdraw (100.50)	
b.withdraw (0)	

Features: Exercise

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	X
b.close	✓
b.close (“Now”)	X
b.open	X
b.withdraw (100.50)	X
b.withdraw (0)	

Features: Exercise

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	X
b.close	✓
b.close (“Now”)	X
b.open	X
b.withdraw (100.50)	X
b.withdraw (0)	✓

Class Text

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

Class Text

```
class PREVIEW   ←————————   Class name
feature
    explore
            -- Explore Zurich.
        do
            central_view.highlight
            zurich_map.animate
        end
end
```

Class Text

```
class PREVIEW ←———— Class name
feature
    explore ←———— Feature declaration
            -- Explore Zurich.
        do
            central_view.highlight
            zurich_map.animate
        end
end
```

Class Text

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

Class name

Feature declaration

Comment

Feature body

Class Text

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

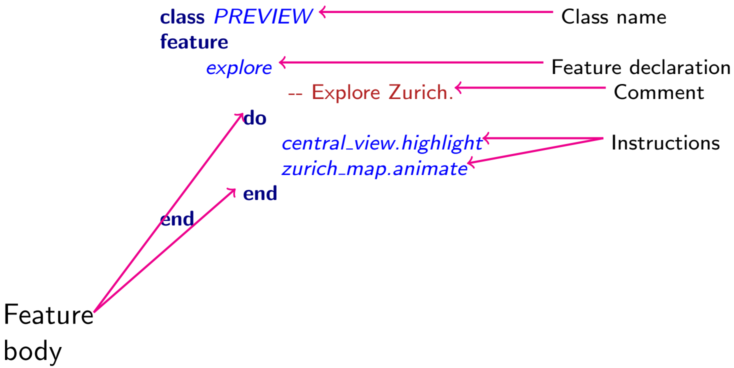
Class name

Feature declaration

Comment

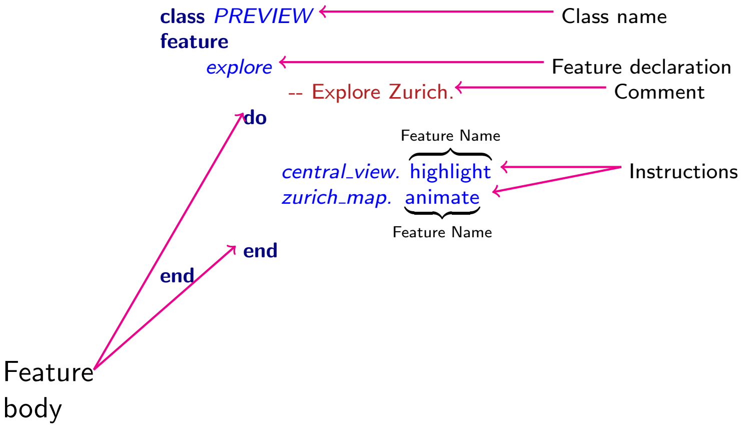
Feature body

Class Text



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

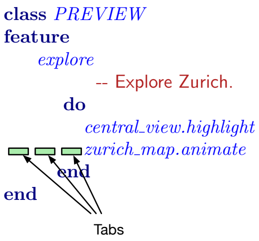
Class Text



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

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
```

Another example

```
class BANK_ACCOUNT
feature
    deposit (a_sum: INTEGER) ──────────→ Routine
            -- Add 'a_sum' to the account.
        do
            balance := balance + a_sum
        end

        balance: INTEGER
end
```

Another example

```
class BANK_ACCOUNT
feature
    deposit (a_sum: INTEGER) ——————→Routine
            -- Add 'a_sum' to the account.
        do
            balance := balance + a_sum
        end

        balance: INTEGER ——————→Attribute
end
```

Another example

```
class BANK_ACCOUNT
feature
    deposit (a_sum: INTEGER)                    Routine
            -- Add ‘a_sum’ to the account.
        do
            balance := balance + a_sum
        end

        balance: INTEGER                    Attribute
end
```

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

Another example

```
class BANK_ACCOUNT
feature
    deposit (a_sum: INTEGER)  ──────→ Routine
            -- Add 'a_sum' to the account.
        do
            balance := balance + a_sum
        end

        balance: INTEGER  ──────→ Attribute
end
```

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

The `state` of the object is defined by the values of its attributes.

Kinds of features: commands and queries

Commands

- Might 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

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?

Exercise: query or command?

Exercise: query or command?

- Tell the balance of a bank account

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?
- List the clients of a bank whose total deposits are over 100,000 RUB.

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?
- List the clients of a bank whose total deposits are over 100,000 RUB.
- Change the account type of a client

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?
- List the clients of a bank whose total deposits are over 100,000 RUB.
- Change the account type of a client
- How much money can a client withdraw at a time?

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?
- List the clients of a bank whose total deposits are over 100,000 RUB.
- 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

Exercise: query or command?

- Tell the balance of a bank account
- Withdraw 400 RUB from a bank account
- Who is the owner of a bank account?
- List the clients of a bank whose total deposits are over 100,000 RUB.
- 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 RUB into a bank account

Command-query separation principle

“Asking a question shouldn’t change the answer”

i.e. a query

Query or command?

```
class DEMO
feature
 procedure_name (a1: T1; a2, a3: T2)
     -- Comment
   do
    ...
   end
 function_name (a1: T1; a2, a3: T2): T3
     -- Comment
   do
    Result := ...
   end
 attribute_name: T3
    -- Comment
```

Query or command?

```
class DEMO
feature
 procedure_name (a1: T1; a2, a3: T2)
     -- Comment
   do
    ...
   end
 function_name (a1: T1; a2, a3: T2): T3
     -- Comment
   do
    Result := ...
   end
 attribute_name: T3
    -- Comment
```

Command (no result)
body

Query or command?

```
class DEMO
feature
 procedure_name (a1: T1; a2, a3: T2)
     -- Comment
   do
    ...
   end
 function_name (a1: T1; a2, a3: T2): T3
     -- Comment
   do
    Result := ...
   end
 attribute_name: T3
   -- Comment
```

Command (no result)
body

Query (result)
body

Query or command?

```
class DEMO
feature
 procedure_name (a1: T1; a2, a3: T2)
     -- Comment
   do
     ...
   end
 function_name (a1: T1; a2, a3: T2): T3
     -- Comment
   do
     Result := ...
   end
 attribute_name: T3
   -- Comment
```

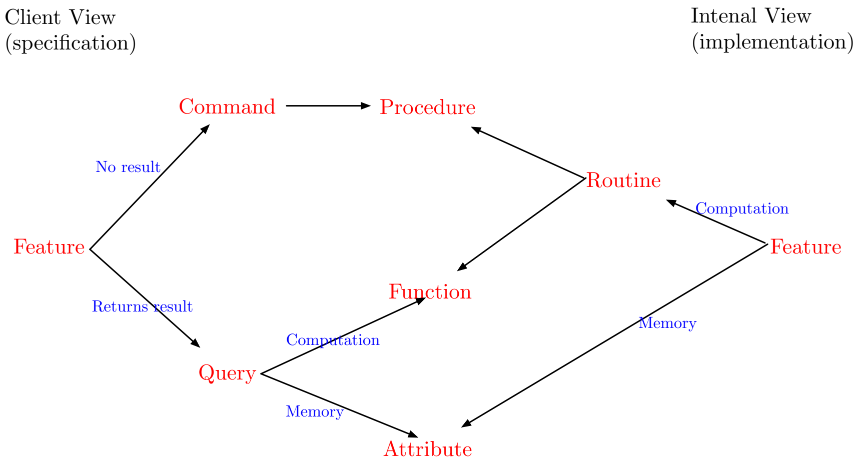
Command (no result)
body

Query (result)
body

Query (result)
no body

Features: the full story

Features: the full story



General form of feature call instructions

$$\underbrace{\underbrace{Object1}\, .query1}_{targets}.command\ (\underbrace{object2.query2, object3}_{arguments})$$

General form of feature call instructions

$$\underbrace{\underbrace{Object1}\,.query1}_{targets}.command\ (\underbrace{object2.query2, object3}_{arguments})$$

- Targets and arguments can be query calls themselves.

General form of feature call instructions

$$\underbrace{\underbrace{Object1}\,.query1}_{targets}.command\,(\underbrace{object2.query2, object3}_{arguments})$$

- Targets and arguments can be query calls themselves.
- Where are *query1*, *query2* defined?
- Where is *command* defined?

Qualified vs. unqualified feature calls

- All features have to be called on some target (object.)
- The **current object** is the name of the target object from the perspective of the feature that was called. I.e., when *x.f* is called, **Current** is *x* during the execution of *f*.
    - A `qualified` feature call has an explicit target.
    - An `unqualified` feature call has **Current** as an implicit target.

Qualified vs. unqualified feature calls

- All features have to be called on some target (object.)
- The **current object** is the name of the target object from the perspective of the feature that was called. I.e., when *x.f* is called, **Current** is *x* during the execution of *f*.
  - A `qualified` feature call has an explicit target.
  - An `unqualified` feature call has **Current** as an implicit target.

```
assign_same_name (a_name: STRING; a_other_person: PERSON)
        -- Assigns 'a_name' to current person and to
'a_other_person' name.
    do
        a_other_person.set_name (a_name)
        set_name (a_name)
    end
```

Qualified vs. unqualified feature calls

- All features have to be called on some target (object.)
- The **current object** is the name of the target object from the perspective of the feature that was called. I.e., when *x.f* is called, **Current** is *x* during the execution of *f*.
  - A `qualified` feature call has an explicit target.
  - An `unqualified` feature call has **Current** as an implicit target.

```
assign_same_name (a_name: STRING; a_other_person: PERSON)
        -- Assigns 'a_name' to current person and to
'a_other_person' name.
    do
        a_other_person.set_name (a_name)        Qualified call
        set_name (a_name)
    end
```

Qualified vs. unqualified feature calls

- All features have to be called on some target (object.)
- The **current object** is the name of the target object from the perspective of the feature that was called. I.e., when *x.f* is called, **Current** is *x* during the execution of *f*.
  - A `qualified` feature call has an explicit target.
  - An `unqualified` feature call has **Current** as an implicit target.

```
assign_same_name (a_name: STRING; a_other_person: PERSON)
        -- Assigns 'a_name' to current person and to
'a_other_person' name.
    do
        a_other_person.set_name (a_name)        Qualified call
        set_name (a_name)        Unqualified call, same as

                                        Current.set_name (a_name)

    end
```

EiffelStudio

EiffelStudio

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

Editor

- 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...”), this may take longer.

Debugger: setup

- The system must be melted/frozen (finalized systems cannot be debugged).
- Setting and unsetting breakpoints
  - An efficient way consists of dropping the feature you want the breakpoint in, into 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.

Debugger: run

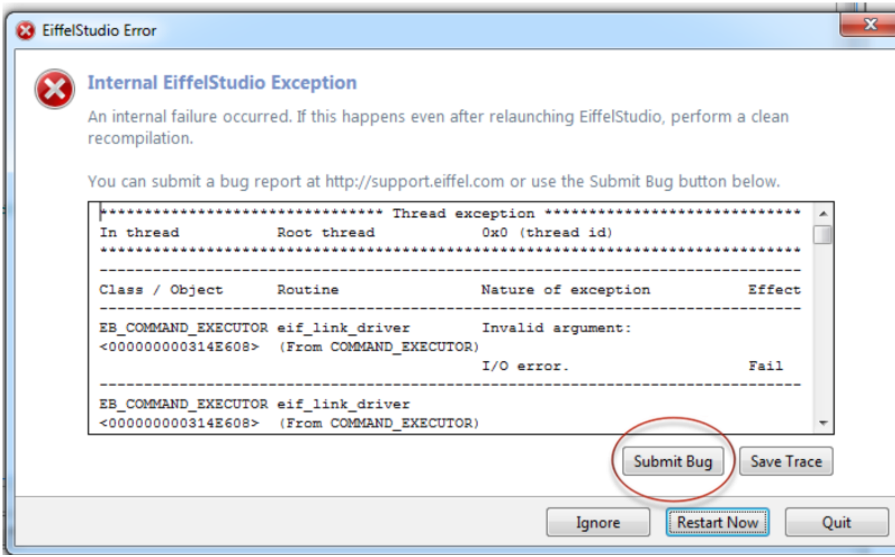
- Run the program by clicking on the Run button.
- Pause by clicking on the Pause button or wait for a triggered breakpoint.
- Analyze the program:
    - Use the call stack pane to browse through the call stack.
    - Use the object tool to inspect the current object, the locals and arguments.
- Run 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.

Found a bug in EiffelStudio?

If EiffelStudio happens to crash:

- You should submit an official bug report by pressing the button that appears when EiffelStudio crashes
- Login: intro_prog_innopolis, Password: introprog1

How to submit a bug 1: submit bug


EiffelStudio Error

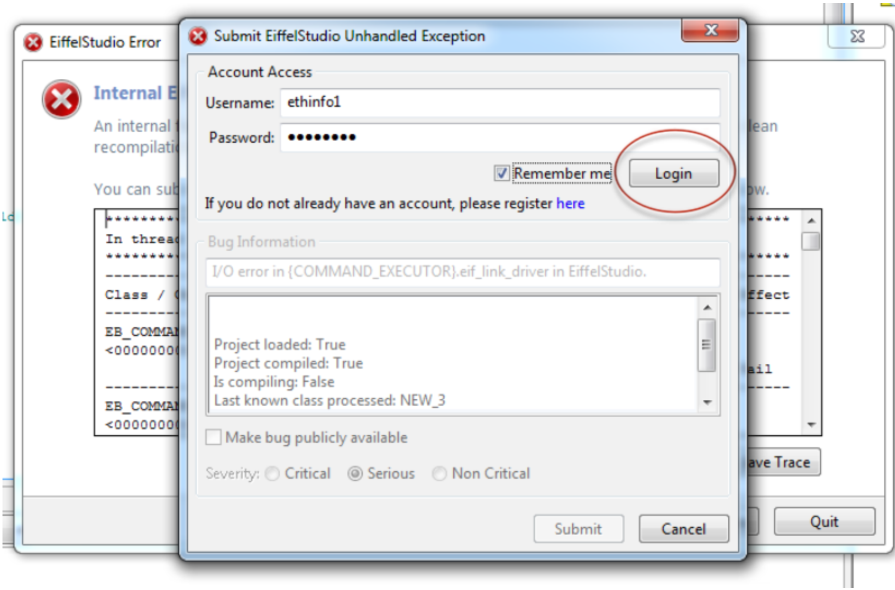
Internal EiffelStudio Exception

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)
```

How to submit a bug 2: login



EiffelStudio Error

Internal E

An internal
recompilati

You can sub

```
*********
In thread
*********
---------
Class / O
---------
EB_COMMAN
<00000000
---------
EB_COMMAN
<00000000
```

Submit EiffelStudio Unhandled Exception

Account Access

Username: ethinfo1

Password: ●●●●●●●●●

[✓] Remember me

Login

If you do not already have an account, please register here

Bug Information

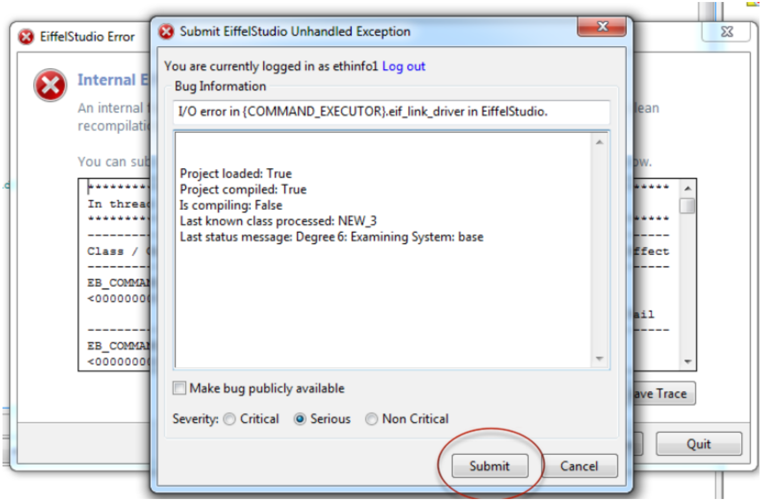
I/O error in {COMMAND_EXECUTOR}.eif_link_driver in EiffelStudio.

```
Project loaded: True
Project compiled: True
Is compiling: False
Last known class processed: NEW_3
```

[ ] Make bug publicly available

Severity: Critical [✓ Serious] Non Critical

How to submit a bug 3: submit



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

Thank you!