



1

Object-Oriented Software Construction

Bertrand Meyer

Lecture 5:

Project and EiffelStudio Presentation



Agenda for today

3

- **Project presentation**
- EiffelStudio: The ISE Eiffel environment



2

Agenda for today

- Project presentation
- EiffelStudio: The ISE Eiffel environment



Organizational matters

4

- **Project deadline:** 30 June 2005 (last day of the semester)
- You will work on the project in **groups of 2**
- **Project specification available at:**
http://se.inf.ethz.ch/teaching/ss2005/0250/project/Project_Specification.pdf



Overview

5

- The project has 2 parts:
 - **Object Spyglass**
 - **Testing framework**
- Testing framework is a client of the Spyglass
- Spyglass must:
 - Expose its functionality through an interface
 - Remain independent of client implementation



Object Spyglass (2)

7

- Additional functionality:
 - Allow the user to navigate the object structure by expanding and collapsing object fields
 - Desirable features:
 - The ability to store user preferences about the layout of objects
 - Heuristics to apply these preferences to subsequent runs of the system



Object Spyglass (1)

6



- Main task: display a snapshot of the state of a running Eiffel system at a particular point of its execution
- This snapshot will contain:
 - The **Current** object
 - The arguments of the routine whose execution was interrupted
 - Any objects reachable from these (through attributes)



Contracts in Eiffel

8

- In the Eiffel method, **contracts** are:
 - Routine preconditions – properties that must hold whenever the routine is called
 - Routine postconditions – properties that the routine guarantees when it returns
 - Class invariants – properties that the instances of a class satisfy at all "stable" times
 - Loop variants and invariants – loop correctness constructs
 - **check** instructions – express the software writer's conviction that a certain property will be satisfied at certain stages of the computation
- Contracts are expressed with **assertions** (boolean expressions + **oid** notation)

Testing framework

9

- Relies on contract violations to signal bugs
- The Spyglass helps the user understand the cause of the bug by displaying the system state when the execution of the routine (where the contract violation occurred) started

Example – System under test (1)

11

```
class EMPLOYEE
...
feature -- BASIC_creation
  receive_salary (sum: INTEGER) is
    --- Deposit 'sum' in the employee's account
    --- After deducting taxes from it.
  require
    sum_positive: sum > 0
  do
    salary_account.deposit (sum - tax (sum))
  end

tax (sum: INTEGER): DOUBLE is
  --- Taxes that the employee pays for 'sum'.
  sum_positive: sum > 0
  local
    pension, health_insurance: DOUBLE
  do
    pension := 0.1 * sum
    health_insurance := 200 --- Health insurance premium is coded as a constant !
    result := pension + health_insurance
  ensure
    tax_positive: Result >= 0
    tax_less_than_sum: Result <= sum
  end
...
end
```

A user wants to test this procedure

Testing framework - Functionality

10

- Allows users to
 - Specify where the test code is located
 - Execute it
- When an assertion violation occurs:
 - Displays the type of the assertion
 - Displays the routine where the violation occurred
- Calls the Object Spyglass to display the state of the system when the execution of the routine started

Example – Test code

12

```
class ROOT_CLASS
  create
  make
  feature (NONE) -- Initialization
  make is
  local
  do
    --- Creation procedure
    an_account: ACCOUNT
    an_employee: EMPLOYEE
  do
    --- Test case 1
    create an_account.make (0)
    create an_employee.make (an_account)
    an_employee.receive_salary (1000) ✓ OK
  do
    --- Test case 2
    create an_account.make (0)
    create an_employee.make (an_account)
    an_employee.receive_salary (100) ✗ Postcondition of tax violated
  end
end
```



Example – What the system should do

13

- Allow the user to:
 - Specify that the test code is located in procedure *make* of class *ROOT_CLASS*
 - Run it
- When the postcondition violation occurs:
 - Stop execution
 - Inform the user about postcondition violation
- Display the state of the system when the execution of routine *tax* started

Testing
frame-
work

Object
Spyglass



What you must deliver

15

- Project code
- Requirements document
- Documentation:
 - User guide - how to use the tool
 - Developer guide - description of the architecture, main classes, limitations, how to extend the tool



Required tasks

14

- Develop:
 - Object Spyglass
 - Testing framework
- You decide on:
 - How you display objects in Spyglass
 - What user preferences regarding the display you store
 - How you use these preferences



Grading criteria

16

1. Correctness
 - Conformance to the specification provided by us
 - Conformance to the requirements document that you deliver
2. Design
 - Interfaces between modules
 - Extensibility
 - Use of design patterns (if applicable)
3. Quality of contracts
4. Quality of code
 - Easy to understand
 - Style guidelines
5. Testing (delivery of a test suite)
6. Documentation
 - Requirements document
 - User guide
 - Developer guide



Agenda for today

17

- Project presentation
- **EiffelStudio: The ISE Eiffel environment**



Material available online

19

- **Guided tour:**
http://se.inf.ethz.ch/teaching/ss2005/0250/readings/eiffel_studio_presentation.pdf



EiffelStudio

18

- Introduction to the IDE
- The Diagram Tool
- Debugging
- Demo



EiffelStudio

20

- **Introduction to the IDE**
- The Diagram Tool
- Debugging
- Demo

Introduction to the IDE

21

- **One development window divided into four panels:**
 - Editor
 - Context tool
 - Clusters pane
 - Features pane+ Search and Favorites
- **Toolbar customization**
- **Pick-and-drop mechanism**



The compiler

23

- Uses incremental compilation
- Supports .NET
- Project Settings Tool

The editor

22

- Syntax highlighting
- Syntax completion (CTRL+Space)
- Class name completion (SHIFT+CTRL+Space)
- Smart indenting
- Block indent or exdent
- Block commenting or uncommenting
- Infinite level of Undo/Redo (reset after a save)
- Quick search features (F3 and SHIFT+F3)



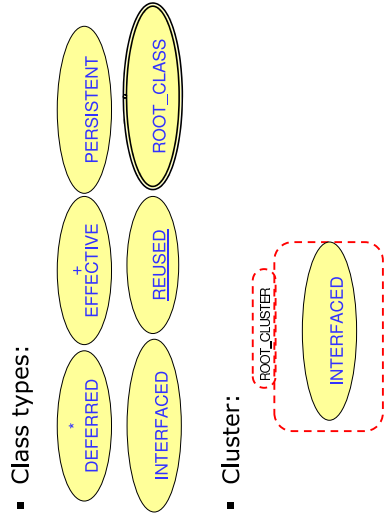
EiffelStudio

24

- Introduction to the IDE
- **The Diagram Tool**
- Debugging
- Demo

A quick run through BON

27



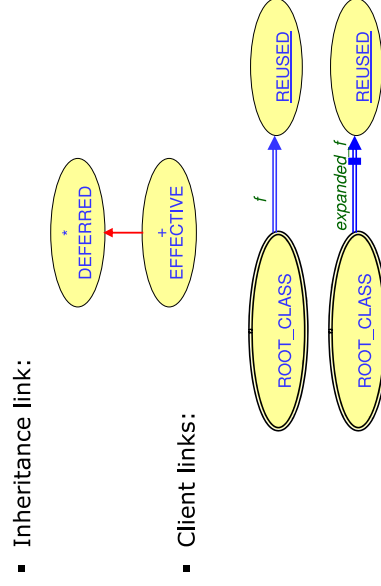
The Diagram tool

27

- Provides "Real time" roundtrip reverse engineering
- Synchronized at each compilation
- Allows for different views

A quick run through BON (cont'd)

28



EiffelStudio

28

- Introduction to the IDE
- The Diagram Tool
- Debugging**
- Demo



Getting started with the debugger

29

- The system must be melted/frozen (finalized systems cannot be debugged)
- Use the Project Settings Tool to specify command line arguments
- Click the *launch* button



Running the application

31

- New display of the Development Window to include debugging information about:
 - The current object (Object Tool)
 - The arguments to the function being debugged (local variables)
- Possibility to control the number of elements the debugger displays for special objects (Arrays, Strings)
- Once on a breakpoint: possibility to step over / into / out next statement
- Possibility to interrupt the application at anytime (*Pause Application* button or SHIFT+CTRL+F5)



Setting breakpoints

30

- Use the flat formats to add breakpoints
 - Tip: An efficient way of adding breakpoints consists in dropping a feature in the context tool
- Click in the margin to enable/disable single breakpoints
- Use the toolbar debug buttons to enable or disable all breakpoints globally



EiffelStudio

32

- Introduction to the IDE
- The Diagram Tool
- Debugging
- **Demo**



End of lecture 5