Object-Oriented Software Construction

Bertrand Meyer

Lecture 14: Presentation of EiffelStudio and Ace Files

Ilinca Ciupa

Agenda for today

- EiffelStudio: The ISE Eiffel environment
- Ace files: Control files for Eiffel projects

EiffelStudio

- Introduction to the IDE
- The Diagram Tool
- Debugging
Material available online

- Guided tour:

Introduction to the IDE

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

EiffelStudio

- Introduction to the IDE
- The Diagram Tool
- Debugging

The editor

- 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)
The compiler

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

A quick run through BON

- Class types:
  - DEFERRED
  - EFFECTIVE
  - PERSISTENT
  - INTERFACED
  - REUSED
  - ROOT_CLASS

- Cluster:
  - ROOT_CLUSTER
  - INTERFACED

EiffelStudio

- Introduction to the IDE
- The Diagram Tool
- Debugging

A quick run through BON (cont’d)

- Inheritance link:
  - DEFERRED
  - EFFECTIVE

- Client links:
  - ROOT_CLASS
  - REUSED
  - ROOT_CLASS
  - REUSED
**The Diagram tool**
- Provides “Real time” roundtrip reverse engineering
- Synchronized at each compilation
- Allows for different views

**Getting started with the debugger**
- 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

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

**Setting breakpoints**
- 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
Running the application

- 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)

Why do we need ace files?

- Ace – Assembly of Classes in Eiffel
- Lace – Language for Assembling Classes in Eiffel
- Lace is the language for writing ace files
- Ace files necessary for specifying:
  - The root class of the system
  - The files that contain the classes of the system (grouped in clusters)

Agenda for today

- EiffelStudio: The ISE Eiffel environment
- Ace files: Control files for Eiffel projects

Generation and editing

- Generation:
  - Automatically done by EiffelStudio when creating a new project
    - Or
      - By hand
- Editing
  - Through the “Project Settings” dialog
    - Or
    - By hand
Example ace file

```

system

root

ROOT_CLASS: "make"

default

assertion (require)

debug ("DEBUG_TAG")

cluster

root_cluster: "."

option

assertion (all): ROOT_CLASS

end

a_subcluster (root_cluster): "&a_subcluster"

all base: "$ISE_EIFFEL/library/base"

exclude

"desc","table_eiffel"

end

all vision2: "$ISE_EIFFEL/library/vision2"

eexternal

include_path: "$ISE_EIFFEL/library\windows\include"

object: "$ISE_EIFFEL/library\windows\include\ISE\EIFFEL\lib\eiffel.lib"

end
```

“root” clause

```

root

ROOT_CLASS: "make"
```

- Specifies the root class and its creation procedure that will be called to start execution of the system

“system” clause

```

system

sample
```

- Gives the name of the system
- Executable file produced will have same name

“default” clause

```

default

assertion (require)

debug ("DEBUG_TAG")
```

- Contains the compilation options of the project (for more options than illustrated here see EiffelStudio Help)
- “assertion” option
  - Which types of assertions are checked
  - Possible values: no, require (default), ensure, invariant, loop, check, all
- “debug” option
  - Activate code written inside debug blocks
    ```
    debug ("DEBUG_TAG")
    ```
    -- Debug code is here.
    ```
    end
    ```
“cluster” clause

```plaintext
clusteroot_cluster: "."
option
  assertion (all): ROOT_CLASS
end
a_subcluster (root_cluster): "/a_subcluster"
all base: "$/ISE_EIFFEL/library/base"
end
"desc"; "table_effe3"
end
all vision2: "$/ISE_EIFFEL/library/vision2"
```

- Locates the files that contain the classes of the system (files with the .e extension)
- Possibility to override the assertion checking level of the whole system for individual classes
- Use keyword all before cluster name to recursively explore subdirectories of specified directory

End of lecture 14

Additional info on ace files

- EiffelStudio Help (“Lace syntax”)
- “Object-Oriented Software Construction”, 2nd edition, Bertrand Meyer
  - Chapter 7: “The static structure: classes”, subsection “Assembling a system” (pp. 198 - 200)
  - Chapter 11: “Design by Contract: building reliable software”, subsection “Monitoring assertions at run time” (pp. 392 - 394)