Introduction to Programming

Bertrand Meyer

Exercise Session 5
Ilinca Ciupa, Stephanie Balzer, Michela Pedroni
7 November 2006

Learning goals

- Part 1: You know how to use EiffelStudio
- Part 2: Reading class interfaces and calling features
- Part 3: Hints for assignment 3

Submitting logs

1. In an explorer navigate to the directory where traffic is located
2. Change to example/chapter2_preview/EIFGENs/preview
3. If you have logging enabled you find a directory called BACKUP. Rename it to <username_es> (e.g. pedronim_es)
4. Zip it (e.g. use WinZip or 7zip)
5. Upload the created zip to http://www.dcs.bbk.ac.uk/~gngch01/fileupload.html

EiffelStudio

- Introduction to the IDE
- Debugging
- Demo

Documentation

- EiffelStudio Help
  - Guided Tour very good as introduction, to get you started
  - www.eiffel.com
Introduction to the IDE

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

The editor

- Syntax highlighting
- Syntax completion (CTRL+Space)
- Class name completion (SHIFT+CTRL+Space)
- Smart indenting
- Block indent or exdent (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)

The compiler

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

Getting started with the debugger

- The system must be melted/frozen (finalized systems cannot be debugged)
- Click the start button

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)
- 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)
Part 2: Class interfaces and feature calls

Objects or classes? (example)

```plaintext
class game
...
feature -- Access
  map_name: string
    -- Name of the map to be loaded for the game
  last_player: player
    -- Last player that moved
...
(continued on next slide)
```

Objects or classes? (example)

```plaintext
... feature -- Status report
  is_occupied (a_location: traffic_place): boolean is
    -- Check if 'a_location' is occupied by some flat hunter.
  require
    a_location_exists: a_location /= Void
  local
    old_cursor: cursor
  do
    Result := False
    -- Remember old cursor position.
    old_cursor := players.cursor
...```

Objects or classes?

- Terms "class" and "type" used interchangeably for now
- Queries (attributes and functions) have a return type. However, when executing the query, you get an object.
- You call features on objects.
- Arguments given to a routine are objects, they all have certain types.
- Local variables declared in a routine are objects, but they all have certain types.
- Result and Current are objects.
- Inheritance is a relation defined between classes.

Skeleton of a class

```plaintext
class CLASS_NAME
  inherit CLASS_NAME_ANCESTOR
  feature
    feature_name_1
    feature_name_2
    ...
    feature_name_n
  end
```

Hands-On
Feature categorization

Feature

Command

Procedure

Query

Function

Attribute

Expression

Instruction

Query or command?

Commands:

commandname (a1: T1; a2, a3: T2) is

-- Comment

require ...

do ...

ensure ...

end

Attributes:

attributename : TYPE

-- Comment

optional

mandatory

Functions:

functionname (a1: T1; a2, a3: T2):TYPE is

-- Comment

require ...

do ...

ensure ...

end


Query or command? (example)

class

GAME

...

feature -- Access

map_name: STRING

-- Name of the map to be loaded for the game

last_player: PLAYER

-- Last player that moved

...

(continued on next slide)


Objects or classes? (example)

-- Loop over all players to check if one occupies
-- a_location.

from

players.start

-- do not consider estate agent, hence skip the first
-- entry in ‘players’.

players.forth

until

players.after or Result

loop

if players.item.location = a_location then

Result := True

end

players.forth

end

-- Restore old cursor position.

players.go_to(old_cursor)

end


Objects or classes? (example)

declaration versus call

Feature Declaration

feature_name (args_decl) is

-- Comment

do ...

end

Feature call

object.feature_name (args)
**An example**

**Feature Declaration**

```plaintext
set_name (a_name: STRING) is
  "-- Set 'name' to 'a_name'.
  do
    name := a_name
  end
```

**Feature call**

```plaintext
a_person.set_name ("Peter")
```

---

**Instructions**

General form of feature call instruction:

```plaintext
query1.query2.command (query3.query4, query5)
```

Call chains for targets and arguments:
- Where are query1, query3 and query5 defined?
- Where is query2 defined?
- Where is query4 defined?
- Where is command defined?

---

**Assignment 3 – Task 2**

Given:
- Where to start looking for the feature (Class, feature)

To do:
- Find return type of the expression
- For which of the features can you use the query as an argument
  - put_string (s: STRING)
  - increase (i: INTEGER)
  - set_state (b: BOOLEAN)

---

**Assignment 3 – Task 3**

3.1 – 3.4:
- Read through feature initialize_scene of class GAME_SCENE
- Change a couple of lines of code

3.5:
- Extend feature statistics in class FLAT_HUNTER.DISPLAYER
- For this you need to know about basic types!
A few words about basic types (1)

- **INTEGER**
  - Represents integer numbers, such as 0, 1, -1, 10, -100, 1234, ...
  - Declaration: `i: INTEGER`
  - Assignment: `i := 1`
  - Operators: `i := 10 + j`

- **REAL**
  - Represents real numbers, such as 0.0, 1.2, -1.5, 10.234, -100.001, 1234.1, ...
  - Declaration: `r: REAL`
  - Assignment: `r := 1.5`

A few words about basic types (2)

- **DOUBLE**
  - Similar to reals, but more precision

- **CHARACTER**
  - Represents single characters, such as 'a', 'x', '0', 'Z', ...
  - Declaration: `c: CHARACTER`
  - Assignment: `c := 'I'`

A few words about basic types (3)

- **BOOLEAN**
  - Represents the 2 boolean values **True** and **False**
  - Declaration: `b: BOOLEAN`
  - Assignment: `b := True`

A few words about basic types (4)

- (Most) other types are non-basic
- Normal feature calling applies just the same as for all types (basic and non-basic)

End exercise session 5