



Object-Oriented Software Construction

Bertrand Meyer

Lecture 20: Some design principles

Iilina Ciupa

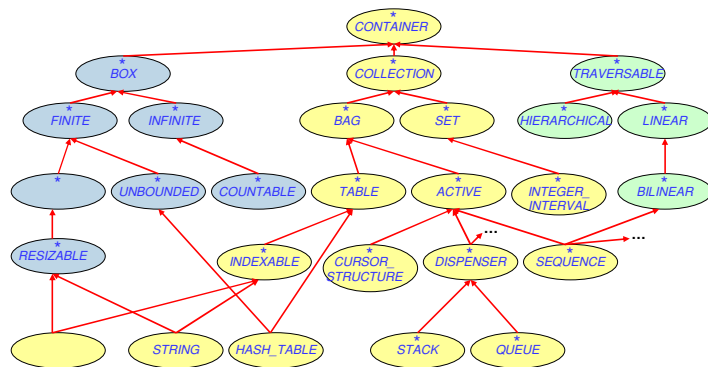


The notion of dispenser

- **Examples:**
 - Stacks
 - Queues
 - Priority queues
 - ...



Container structures

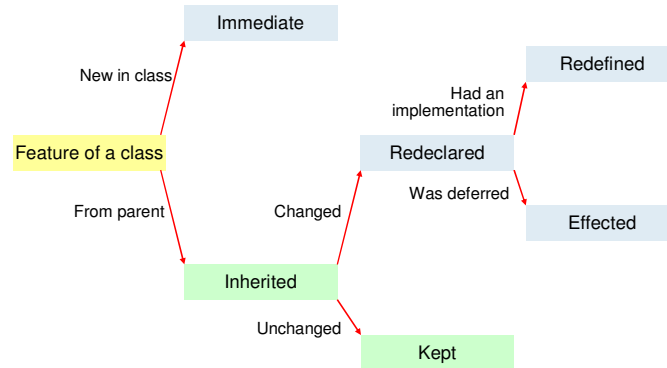


How big should a class be?

- **The first question is how to measure class size. Candidate metrics:**
 - Source lines.
 - Number of features.
- **For the number of features the choices are:**
 - With respect to information hiding:
 - Internal size: includes non-exported features.
 - External size: includes exported features only.
 - With respect to inheritance:
 - Immediate size: includes new (immediate) features only.
 - Flat size: includes immediate and inherited features.
 - Incremental size: includes immediate and redeclared features.

The features of a class

5



Most useful measure is incremental size. Easy to measure.

Some statistics from EiffelBase

7

- Percentages, rounded.
- 149 classes, 1823 exported features. (Includes Lex and Parse.)

| | |
|--------------------|----|
| 0 to 5 features | 45 |
| 6 to 10 features | 17 |
| 11 to 15 features | 11 |
| 16 to 20 features | 9 |
| 21 to 40 features | 13 |
| 41 to 80 features | 4 |
| 81 to 142 features | 1 |

The shopping list approach

6

- If a feature may be useful, it probably is.
- An extra feature cannot hurt if it is designed according to the spirit of the class (i.e. properly belongs in the underlying abstract data type), is consistent with its other features, and follows the principles of this presentation.
- No need to limit classes to "atomic" features.

Some statistics from EiffelVision

8

- Percentages, rounded. 546 classes, 3666 exported features.

| | |
|-------------------|----|
| 0 to 5 features | 68 |
| 6 to 10 features | 12 |
| 11 to 15 features | 7 |
| 16 to 20 features | 4 |
| 21 to 40 features | 6 |
| 41 to 78 features | 2 |

Including non-exported features

9

- Percentage rounded. All features (about 7600).

| | Base | Vision |
|---------------------|------|--------|
| 0 to 5 features | 37 | 55 |
| 6 to 10 features | 23 | 18 |
| 11 to 15 features | 7 | 7 |
| 16 to 20 features | 6 | 5 |
| 21 to 40 features | 16 | 10 |
| 41 to 80 features | 9 | 4 |
| 81 or more features | 2 | 0.4 |

- Ratio of total features to exported features: 1.27 (EiffelBase), 1.44 (EiffelVision)

The size of feature interfaces

11

- More relevant than class size for assessing complexity.
- Statistics from Base, Lex and Parse (exported features only):

| | |
|--|------|
| Number of features | 1823 |
| Percentage of queries | 59% |
| Percentage of commands | 41% |
| Average number of arguments to a feature | 0.4 |
| Maximum number | 3 |
| No argument | 60% |
| One argument | 37% |
| Two arguments | 3% |
| Three arguments | 0.3% |

Minimalism revisited

10

- The language should be small (ETL: "*The language design should provide a good way to express every operation of interest; it should avoid providing two.*")
- The library, in contrast, should provide as many useful facilities as possible.
- Key to a non-minimalist library:
 - Consistent design.
 - Naming.
 - Contracts.
- Usefulness and power.

The size of feature interfaces (cont'd)

12

- Including non-exported features:

| | |
|--|------|
| Average number of arguments to a feature | 0.5 |
| Maximum number | 6 |
| No argument | 57% |
| One argument | 36% |
| Two arguments | 5% |
| Three arguments | 1% |
| Four arguments | 0.6% |
| Five or six arguments | 0.2% |

The size of feature interfaces (cont'd) 13

- Statistics from EiffelVision (546 classes, exported features only):

| | |
|--|------|
| Number of features | 3666 |
| Percentage of queries | 39% |
| Percentage of commands | 61% |
| Average number of arguments to a feature | 0.7 |
| Maximum number | 7 |
| No argument | 49% |
| One arguments | 32% |
| Two arguments | 15% |
| Three arguments | 3% |
| Four arguments | 0.4% |
| Five to seven arguments | 0.4% |

Recognizing options from operands 15

- Two criteria to recognize an option:
 - There is a reasonable default value.
 - During the evolution of a class, operands will normally remain the same, but options may be added.

Operands and options 14

- Two possible kinds of argument to a feature:
 - Operands: values on which feature will operate.
 - Options: modes that govern how feature will operate.
- Example: printing a real. The number is an operand; format properties (e.g. number of significant digits, width) are options.

```
print (real_value, number_of_significant_digits,  
      zone_length, number_of_exponent_digits, ...)
```

```
my_window.display (x_position, y_position,  
                  height, width, text, title_bar_text, color, ...)
```

Operands and options 16

- The Option Principle:
 - The arguments of a feature should only be operands.
 - Options should have default values, with procedures to set different values if requested.
 - For example:

```
my_window.set_background_color ("dd-blue")  
...  
my_window.display
```

Operands and options

17

- Useful checklist for options:

| Option | Default | Set | Accessed |
|--------------|---------|---|-------------------------|
| | | | |
| Window color | White | <i>set_background_color</i> | <i>background_color</i> |
| | | | |
| Hidden? | No | <i>set_visible</i> <i>set_hidden</i> | <i>hidden</i> |

Naming (2)

19

| Class | Features | | |
|------------|------------|-------------|---------------|
| ARRAY | <i>put</i> | <i>item</i> | |
| STACK | <i>put</i> | <i>item</i> | <i>remove</i> |
| QUEUE | <i>put</i> | <i>item</i> | <i>remove</i> |
| HASH_TABLE | <i>put</i> | <i>item</i> | <i>remove</i> |

Naming (1)

18

| Class | Features | | |
|------------|---------------|---------------|----------------------|
| ARRAY | <i>enter</i> | <i>entry</i> | |
| STACK | <i>push</i> | <i>top</i> | <i>pop</i> |
| QUEUE | <i>add</i> | <i>oldest</i> | <i>remove_oldest</i> |
| HASH_TABLE | <i>insert</i> | <i>value</i> | <i>delete</i> |

Naming rules

20

- Achieve consistency by systematically using a set of standardized names.
- Emphasize commonality over differences.
- Differences will be captured by:
 - Signatures (number and types of arguments and result).
 - Assertions.
 - Comments.

Grammatical rules

21

- Procedures (commands): verbs, infinitive form. Examples: *make, put, display*.
- Boolean queries: adjectives, e.g. *full*. Also (especially in case of potential ambiguity) names of the form *is_some_property*. Example: *is_first*.
 - In all cases, you should usually choose the form of the property that is false by default at initialization (making it true is an event worth talking about). Example: *is_erroneous*.
- Other queries: nouns or adjectives. Examples: *count, error_window*.
- Do not use verbs for queries, in particular functions; this goes with the command-query separation principle (prohibition of side-effects in functions).

Feature categories (cont'd)

23

- Standard categories (the only ones in EiffelBase):
 - Initialization
 - Access
 - Measurement
 - Comparison
 - Status report
 - Status setting
 - Cursor movement
 - Element change
 - Removal
 - Resizing
 - Transformation
 - Conversion
 - Duplication
 - Basic operations
 - Obsolete
 - Inapplicable
 - Implementation
 - Miscellaneous

Feature categories

22

class C inherit

...

feature -- Category 1

... Feature declarations

feature {A, B} -- Category 2

... Feature declarations

feature {NONE} -- Category n

... Feature declarations

invariant

...

end

Obsolete features and classes

24

- A central problem in the computer field: how to reconcile progress with the protection of the installed base?
- Obsolete features and classes support smooth evolution.
- In class *ARRAY*:

```
enter (i: V; v: T) is
  obsolete "Use `put (value, index)'"
do
  put (v, i)
end
```



Obsolete classes

25

```
class ARRAY_LIST [G] obsolete
```

```
"["
```

```
  Use MULTI_ARRAY_LIST instead  
  (same semantics, but new name  
  ensures more consistent terminology).  
  Caution: do not confuse with ARRAYED_LIST  
  (lists implemented by one array each).
```

```
]"
```

```
inherit
```

```
  MULTI_ARRAY_LIST [G]
```

```
end
```



26

End of lecture 20