Trusted Components
Reuse, Contracts and Patterns

Prof. Dr. Bertrand Meyer
Dr. Karine Arnout
Lecture 17: Composite, Flyweight
Agenda for today

- Composite pattern / Library
- Flyweight pattern / Library
Composite pattern: Intent

“Way to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.” [GoF, p 163]

Architecture of a typical application
Composite pattern

Transparency version

Safety version
Flaws of the approach

- To make `MY_COMPOSITE` a composite of parts of type `MY_LEAF`, one must:
  - Create a descendant `MY_COMPONENT` of class `COMPONENT`
  - Make `MY_COMPOSITE` inherit from `COMPOSITE`
  - Make `MY_LEAF` inherit from `LEAF`
  - Redefine `i_th` to return an instance of `MY_COMPONENT`

**Genericity**

“Natural to represent a composite as a generic class with a parent and a set of children” [Jézéquel 1999]
Composite Library

Transparency version

Safety version
deferred class

COMPONENT \[ G \]

feature -- Basic operation

\textit{do\_something is}

-- Do something.

defered
end

feature -- Status report

\textit{is\_composite: BOOLEAN is}

-- Is component a composite?

do

\textbf{Result} := \textit{False}
end
end

Mechanism enabling componentization: Unconstrained genericity
class  
    \textit{COMPOSITE} \[ G \]
inherit  
    \textit{COMPONENT} \[ G \]
redefine  
    \textit{is\_composite}
end
create  
    make,  
    make\_from\_components
feature \{NONE\}  -- Initialization
    make is  
        -- Initialize component \textit{parts}.
        do  
            create \textit{parts.make}
        end
end
make_from_components (some_components: like parts) is
   -- Set parts to some_components.
   require
      some_components_not_void: some_components /= Void
      no_void_component: not some_components.has (Void)
   do
      parts := some_components
   ensure
      parts_set: parts = some_components
   end

feature -- Status report
   is_composite: BOOLEAN is
      -- Is component a composite?
      do
         Result := True
      end
trust\_int\_library, safety version (4/5)

\begin{verbatim}
feature -- Basic operation
    do\_something \textbf{is}
        -- Do something.
        do
            from \textit{parts}.\textit{start} until \textit{parts}.\textit{after} loop
                \textit{parts}.\textit{item}.\textbf{do\_something}
                \textit{parts}.\textit{forth}
        end
end

feature -- Access
    \textit{item}: COMPONENT \{G\} \textbf{is}
        -- Current part of composite
        do
            Result := \textit{parts}.\textit{item}
        end

ensure
    definition: Result = \textit{parts}.\textit{item}
    component\_not\_void: Result /= Void
end
\end{verbatim}
(feature -- Others
   -- Access: i_th, first, last
   -- Status report: has, is_empty, off, after, before
   -- Measurement: count
   -- Element change: add
   -- Removal: remove
   -- Cursor movement: start, forth, finish, back

feature {NONE}  -- Implementation
   parts: LINKED_LIST [like item]
      -- Component parts
      -- (which are themselves components)

invariant
   is_composite: is_composite
   parts_not_void: parts /= Void
   no_void_part: not parts.has (Void)
end
Composite pattern vs. Composite Library

- BORROWABLE
- COMPOSITE
- BOOK
- COMPOSITE_BORROWABLE
- ENCYCLOPEDIA
Composite: Componentization outcome

- Completeness
  - All cases of the Composite pattern

- Usefulness
  - Reusable
  - Easy-to-use

- Faithfulness
  - Similar to a traditional implementation of Composite (with genericity)

- Type-safety
  - Type-safe (unconstrained genericity, assertions)

- Performance
  - Same order as the Composite pattern

- Extended applicability
  - No more cases
Agenda for today

- Composite pattern / Library
- Flyweight pattern / Library
Flyweight pattern: Description

- **Intent**: “Use sharing to support large numbers of fine-grained objects efficiently.” [GoF, p 195]

- **Motivation**: see [next](#)
Without the Flyweight pattern (1/2)

class
  CLIENT
...
feature -- Basic operation
draw_lines is
  -- Draw some lines in color.
  local
    line1, line2: LINE
    red: INTEGER
  do
    ...
      create line1.make (red, 100, 200)
      line1.draw
      create line2.make (red, 100, 400)
      line2.draw
    ...
end
...
end

Creates one LINE object for each line to draw
class interface
   LINE
create
   make
feature -- Initialization
   make (a_color, x, y: INTEGER)
      -- Set color to a_color, x as x_position, and y as y_position.
   ensure
      color_set: color = a_color
      x_set: x_position = x
      y_set: y_position = y
feature -- Access
   color: INTEGER
      -- Line color
   x_position, y_position: INTEGER
      -- Line position
feature -- Basic operation
   draw
      -- Draw line at position (x_position, y_position) with color.
end
With the Flyweight pattern (1/3)

class CLIENT

feature -- Basic operation
draw_lines is

-- Draw some lines in color.

local

line_factory: LINE_FACTORY
red_line: LINE
red: INTEGER

do

... 

red_line := line_factory.new_line (red)
red_line.draw (100, 200)
red_line.draw (100, 400)

...

end

end

Creates only one LINE object per color
With the Flyweight pattern (2/3)

class interface
  LINE_FACTORY

feature -- Initialization
  new_line (a_color: INTEGER): LINE
    -- New line with color a_color
    ensure
      new_line_not_void: Result /= Void

  ...
end
class interface
  LINE
create
  make
feature -- Initialization
  make (a_color: INTEGER) is
    -- Set color to a_color.
    ensure
      color_set: color = a_color
feature -- Access
  color: INTEGER
    -- Line color
feature -- Basic operation
  draw (x, y: INTEGER)
    -- Draw line at position (x, y) with color.
end
Two kinds of properties:

- Intrinsic characteristics stored in the flyweight
- Extrinsic characteristics moved to the client (typically a “flyweight context”)
Flyweight context

- External characteristics are not stored in the flyweight object → client must handle them

- A possibility is to create a `FLYWEIGHT_CONTEXT` describing a list of flyweights grouped by `FLYWEIGHT_ZONE`s with the same external characteristic (e.g. characters with the same color in a row of a book page)
Shared/unshared and (non-)composite objects

- Two kinds of flyweights:
  - Composites (shared or unshared)
  - Non-composites (shared)
Flyweight pattern: Description

- **Intent:** “Use sharing to support large numbers of fine-grained objects efficiently.”

- **Participants:**
  - **FLYWEIGHT:** Offers a service `do_something` to which the extrinsic characteristic will be passed
  - **SHARED_FLYWEIGHT:** Adds storage for intrinsic characteristic
  - **COMPOSITE_FLYWEIGHT:** Composite of shared flyweight; May be shared or unshared
  - **FACTORY:** Creates and manages the flyweight objects
  - **CLIENT:** Maintains a reference to flyweight, and computes or stores the extrinsic characteristics of flyweight
Flyweight pattern: Typical architecture
Flaws of the approach

- Not a reusable component
- Code repetition
- Low maintainability

**No Code Repetition principle**

If you find yourself having to copy and paste code, just stop: there is something wrong with your design.
Mechanisms enabling componentization: constrained genericity, agents

- Factory Library and Composite Library
Flyweight Library: use of the Composite Library

- Two kinds of flyweights:
  - Non-composites (shared)
  - Composites (shared or unshared)

→ Use the Composite Library
  - \textit{FLYWEIGHT} [G] inherits from \textit{COMPONENT} [\textit{FLYWEIGHT} [G]]
  - \textit{COMPOSITE\_FLYWEIGHT} [G, H] inherits from \textit{FLYWEIGHT} [G] and \textit{COMPPOSITE} [\textit{FLYWEIGHT} [G]]

Uses the safety version of the Flyweight Library where the \textit{COMPONENT} does not know about its parent to allow a same flyweight object to be part of different composites
Class *FLYWEIGHT* (1/3)

defered class
  *FLYWEIGHT* [\(G \rightarrow SHARABLE\) create make end]
inherit
  *COMPONENT* [FLYWEIGHT [\(G\)]]
rename
do_something as do_something_component
end
feature -- Initialization
make (a_procedure: like procedure) is
  -- Set a_procedure to a_procedure.
require
  a_procedure_not_void: a_procedure /= Void
do
  procedure := a_procedure
ensure
  procedure_set: procedure = a_procedure
end
Class **FLYWEIGHT (2/3)**

**feature** -- Access

```plaintext
external_characteristic \( (a\_context: \text{FLYWEIGHT\_CONTEXT} [G]): G \text{ is} \)
-- External characteristic of flyweight in \( a\_context \)

require

\( a\_context\_not\_void: a\_context \neq \text{Void} \)

\textbf{do}

\( \textbf{Result} := a\_context.\text{external}\_characteristic \)

\textbf{ensure}

\( \text{external}\_characteristic\_not\_void: \textbf{Result} \neq \text{Void} \)

\textbf{end}
```

**procedure**: `PROCEDURE [ANY, TUPLE [FLYWEIGHT [G], FLYWEIGHT\_CONTEXT [G]]]
-- Procedure called by \texttt{do\_something} for shared flyweights`
Class **FLYWEIGHT (3/3)**

feature -- Element change

```
set_external_characteristic (a_characteristic: like external_characteristic;
   a_context: FLYWEIGHT_CONTEXT [G]) is
   -- Set external_characteristic of a_context to a_characteristic.
   require
   a_characteristic_not_void: a_characteristic /= Void
   a_context_not_void: a_context /= Void
   do
   a_context.start
   ensure
   external_characteristic_set:
   a_context.external_characteristic /= Void and then
   a_context.external_characteristic = a_characteristic
   end
```

feature -- Output

```
do_something (a_context: FLYWEIGHT_CONTEXT [G]) is deferred end
   -- Do something with flyweight according to a_context.
   require
   a_context_not_void: a_context /= Void
   end
```
Flyweight constraint: \textit{SHARABLE}

\texttt{deferred class COMPOSITE [G -> SHARABLE create make end, H -> HASHABLE]}

With:

\texttt{deferred class SHARABLE inherit FLYWEIGHT_CONSTANTS feature make (a_code: \texttt{like code}) is ... code: INTEGER ... end}

Actual generic parameter must conform to \textit{SHARABLE} because the library requires \textit{COMPOSITE}s to contain only \textit{SHARABLE} objects

External characteristic must be represented by an \texttt{INTEGER} code
Example: A *BOOK* is made of *SENTENCE*es, which are made of *CHARACTER*es (i.e. *BOOK* is a *COMPOSITE_FLYWEIGHT* of *CHARACTER*es)

With the Flyweight pattern:

```
new_sentence: SENTENCE is
  -- New sentence

local
  context: FLYWEIGHT_CONTEXT

do
  create Result.make
  Result.set_text ("Successful componentization")
  create context.make ('e')
  Result.draw (context)
end
```
With the Flyweight Library:

```java
new_sentence: SENTENCE is
    -- New sentence
    local
    context: FLYWEIGHT_CONTEXT [CHARACTER]
    do
        create Result.make (agent draw)
        Result.set_text ("Successful componentization")
        create context.make ('e')
        Result.do_something (context)
    end
```

Use of an agent to initialize the flyweight
Flyweight: Componentization outcome

- Completeness
  - All cases of the Flyweight pattern

- Usefulness
  - Reusable
  - Easy-to-use

- Faithfulness
  - Similar to a traditional implementation of Flyweight (with genericity, Composite Library and Factory Library)

- Type-safety
  - Type-safe (constrained genericity, agents, assertions, and the Composite Library and Factory Library)

- Performance
  - Same order as the Flyweight pattern

- Extended applicability
  - No more cases
Complementary material (1/2)

- From Patterns to Components:
  - Chapter 10: Composite
  - Chapter 11: Flyweight

- Further reading:
Further reading:


End of lecture 17