



Software Architecture

Command pattern
Assignment 2, Task 1 solution

Undo/redo for square



⑩ Write a undo/redo mechanism for a square manipulation system

⑩ Commands available:

C i j -- Create square i with length j

M i j k -- Move square i by j resp. k units to
right/up

S i j -- Scale square i by factor j

U -- Undo last not yet undone C, M or S

R -- Redo last not yet redone U action

P -- Prints all squares in ascending order

Class SQUARE (1/4)



```
class SQUARE
create make
feature -- Initialization
    make (a_number, a_side: INTEGER) is
        -- Create a square.

        require
            a_number_positive: a_number > 0
            a_size_positive: a_side > 0

        do
            set_number (a_number)
            set_side_length (a_side)

        ensure
            number_set: number = a_number
            side_length_set: side_length = a_side

        end
end
```

Class SQUARE (2/4)



feature -- Access

number: INTEGER

-- Square number

x: INTEGER

--Ordinate

y: INTEGER

-- Abscissa

side_length: INTEGER

-- Side length

Class SQUARE (3/4)



feature -- Setting

```
set_number (a_number: INTEGER)
    -- Set number with a_number.
```

```
set_x (a_x: INTEGER)
    -- Set x with a_x.
```

```
set_y (a_y: INTEGER)
    -- Set y with a_y.
```

```
set_side_length (a_side: INTEGER)
    -- Set side_length with a_side.
```

Class SQUARE (4/4)



feature -- Basic operations

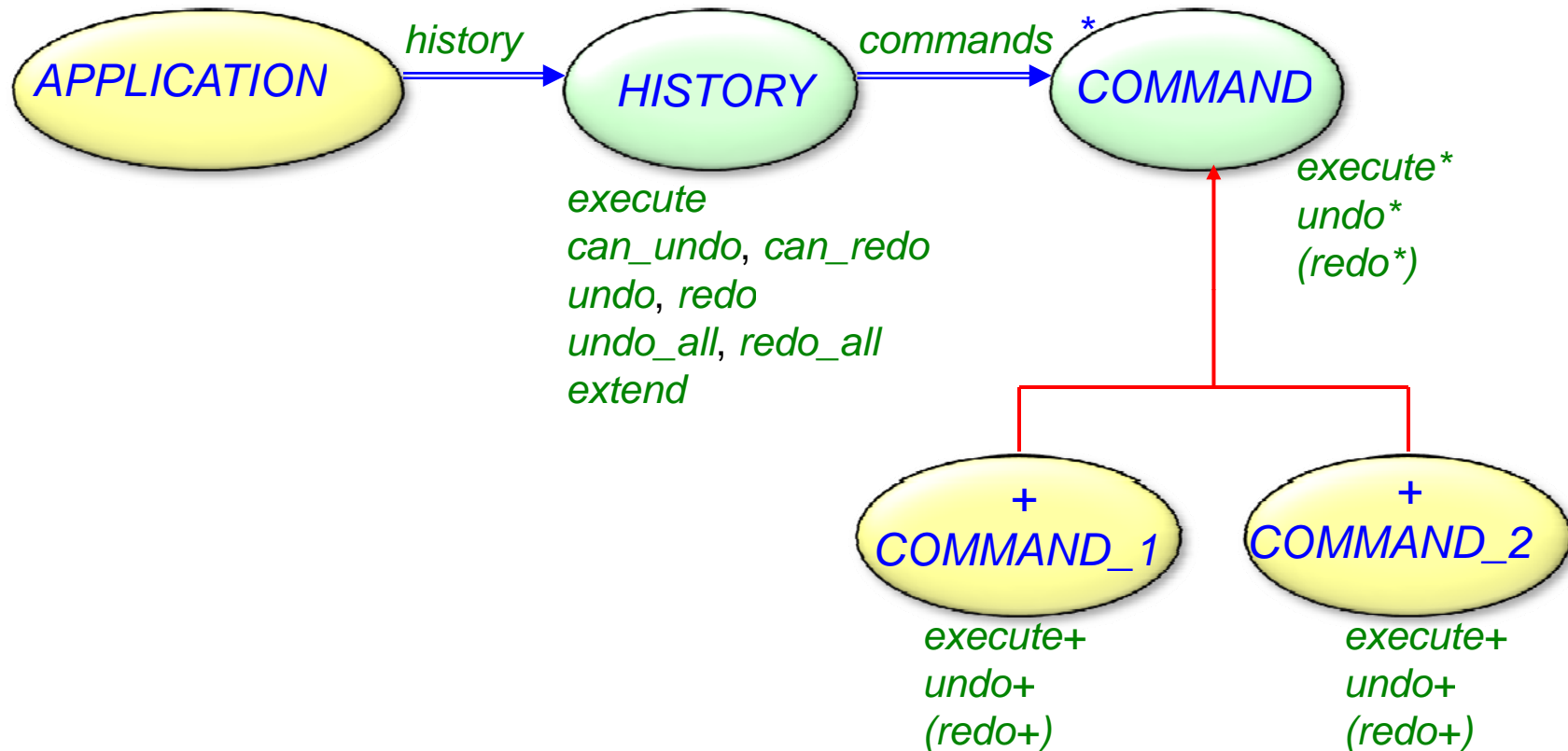
`move (a_x, a_y: INTEGER)`

-- Move square by `a_x` horizontally and `a_y` vertically.

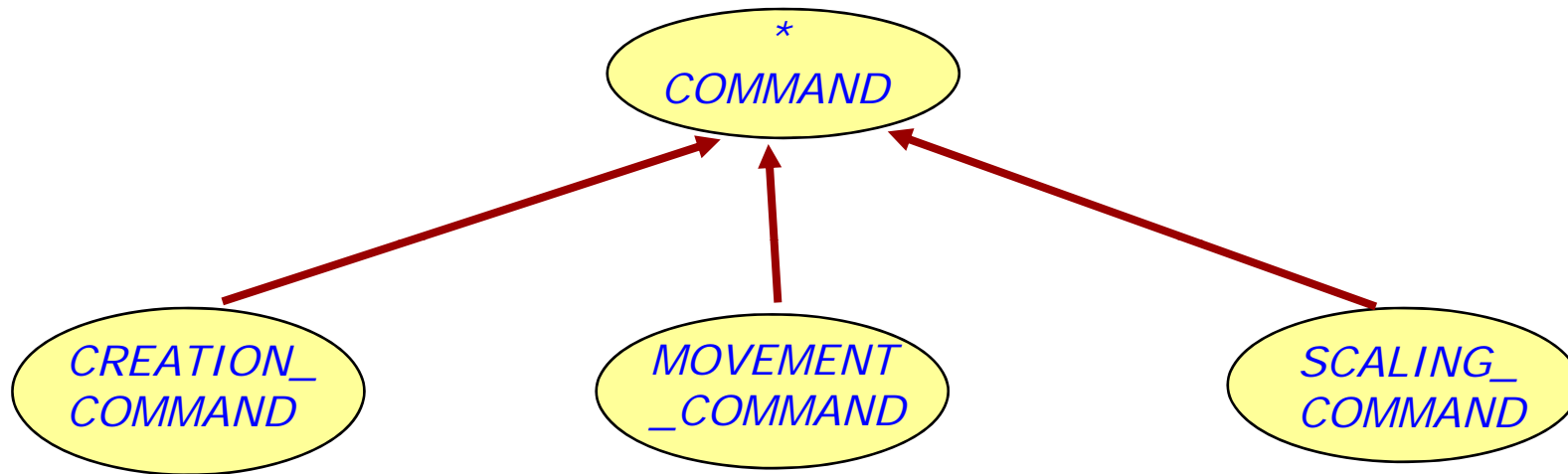
`scale (a_factor: INTEGER)`

-- Scale current square by `a_factor`.

Command pattern: overall architecture



Class hierarchy for SMS



COMMAND (1/2)



```
class COMMAND
```

```
feature -- Status report
```

```
    is_done: BOOLEAN
```

```
        -- Has current command been executed?
```

```
feature -- Basic operations
```

```
    execute is
```

```
        -- Execute current command.
```

```
        require
```

```
            not_done: not is_done
```

```
        deferred
```

```
        ensure
```

```
            command_done: is_done
```

```
        end
```

COMMAND (2/2)



undo is

-- Undo current command on a_squares.

require

is_done: is_done

deferred

ensure

not_done: not is_done

end

end

CREATION_COMMAND (1/3)



```
class CREATION_COMMAND inherit COMMAND
```

```
create make
```

```
feature -- Initialization
```

```
make (a_squares: ARRAY [SQUARE]; i, j: INTEGER) is
```

```
-- Initialize command for creation of square with number `i` and  
-- side length `j`.
```

```
require
```

```
a_squares_attached: a_squares /= Void
```

```
j_positive: j > 0
```

```
do
```

```
squares := a_squares
```

```
create square.make (i, j)
```

```
ensure
```

```
squares_set: squares = a_squares
```

```
square_created: square /= Void and then (square.number = i  
square.side_length = j)
```

```
and
```

```
end
```

CREATION_COMMAND (2/3)



feature -- Basic operations

execute is

-- Execute current command on.

do

squares.force (square, square.number)

is_done := **True**

ensure

square_inserted: squares.item (square.number) = square

end

undo is

-- Undo current command.

do

squares.put (**Void**, square.number)

is_done := **False**

ensure

square_removed: squares.item (square.number) = **Void**

end

CREATION_COMMAND (3/3)



feature{NONE} -- Implementation

squares: ARRAY [SQUARE]
-- List of squares

square: SQUARE
-- Square to put into lists

end

Remaining



Classes such as

`MOVEMENT_COMMAND,`
`SCALING_COMMAND`

are implemented alike.

A history is needed to store commands:

```
cmd_history: LINKED_LIST [COMMAND]
```

...

```
cmd_history.extend (cmd1)
```

```
cmd_history.extend (cmd2)
```