ETHZ D-INFK Prof. Dr. B. Meyer	Software Architecture – Exercise

Exercise 2: Design Patterns

Assignment 1

The task is to write an undo-redo mechanism for a simple system. The example is a Square Manipulation System (SMS).

Input

The program reads its data as successive lines from standard input. A line has one of the following forms, where i, j, and k are non-negative integers:

C i j
M i j k
S i j
U
R
P

Your program may assume that the input follows this syntax; error handling is not required.

Semantics

Note that no graphical display is needed. All information about squares will be output textually (see the P command).

The C command (Create) creates square number i with side length j, centered at the origin. It replaces any previous square with number i.

The M command (Move) moves square number i by j pixels to the right and k pixels upwards. If there is no square numbered i, the command has no effect.

The S command (Scale) scales square numbered i by a factor of j. If there is no square numbered i, the command has no effect.

The U command (Undo) cancels the last not-yet-undone C, M or S command. If none remains to be undone, U has no effect.

The R command (Redo) is applicable only if the last executed command was U or R, otherwise it does nothing. It re-executes the most recent C, M or S command undone and not yet redone.

The P command (Print) prints on standard output details of all squares in the system. Squares are printed in ascending order with respect to number, one per line, where each line has the format

i j k l

i is the square number, j its horizontal coordinate, k its vertical coordinate and l its side length. The values should be separated by single spaces.

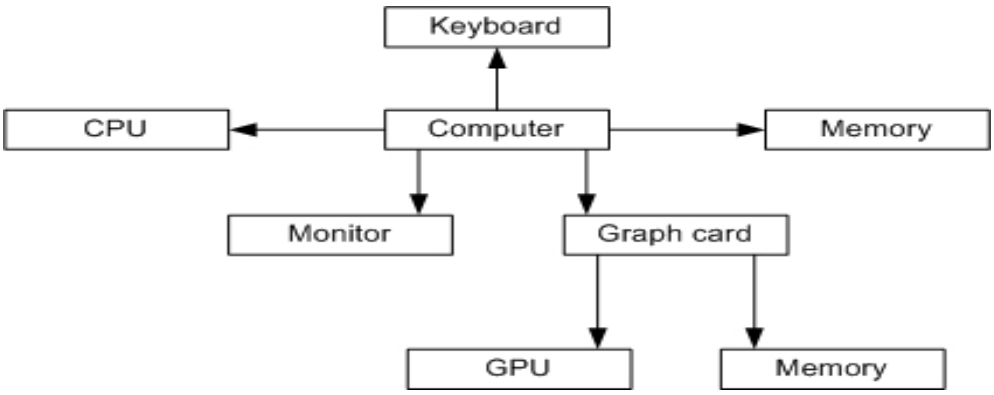
Hint

The standard solution for undo-redo, which you are encouraged to use, is documented in both of the course textbooks: Meyer ("Undo-Redo") and Gamma et al. ("Command pattern"). The central abstraction in this solution is the undoable command. There is a class COMMAND whose instances represent undoable commands (here one other than U, R or P); for each command kind, a descendant class of COMMAND is created, for example, MOVE, with attributes containing just enough information to do and undo one execution of the command. For example, in a text editor, a class for "delete a line" would have attributes storing the contents of a line and its position in the file. The program contains a history list – a list of COMMAND objects that can be traversed back and forth to process a sequence of U and R commands. Note that there is no fixed limit on the number of undo-redo levels.

Assignment 2

Background

Consider the following computer architecture:



In the graph, *CPU*, *GPU*, *memory*, *monitor* and *keyboard* are atomic components, while *graph card* and *computer* are composites. Every atomic component has a price associated with it. The price of a composite part is the sum of all its components. For example, the price of the *computer* is the sum of

the price of the *monitor*, *CPU, keyboard, memory* and *the graph card*, and the price of *the graph card* is the sum of the price of the *GPU* and the *memory*.

Task

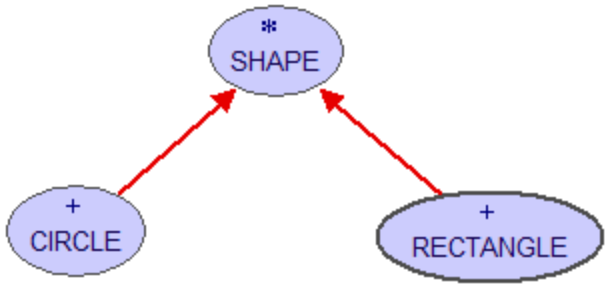
Use composite pattern to model the computer architecture. Then write a visitor using the visitor pattern to calculate the price of all the memory components (the price of other kinds of components are ignored) in computer architecture.

Assignment 3

Background

Consider a shape display system which can display shapes in several output formats.

All shape classes in the system have a deferred class SHAPE as their ancestor. For example, the following figure shows part of the shape hierarchy:



Classes SCREEN and PRINTER are used to display shapes on a screen or printer respectively:

class SCREEN
feature

draw_line (*x1*, *y1*, *x2*, *y2:* INTEGER) **is …**
-- Display a line from (x1, y1) to (x2, y2).

draw_pixel (*x*, *y:* INTEGER) **is …**
-- Display a pixel at (x, y).

draw_circle (*x*, *y*, *r:* INTEGER) **is …**
-- Display a circle at (x,y) with radius r.

end

class PRINTER
feature

print_line (*x1*, *y1*, *x2*, *y2:* INTEGER) **is …**
-- Print a line from (x1, y1) to (x2, y2).

print_pixel (*x*, *y:* INTEGER) **is …**
-- Print a pixel at (x, y).

print_circle (*x*, *y*, *r:* INTEGER) **is …**
-- Print a circle at (x,y) with radius r.

end

Task

Design a system which allows new shapes and new output formats to be added easily. You are required to add a new shape POLYGON and a new output format XML_WRITER.

Hints

Consider the Bridge pattern to allow the shape hierarchy and the output format hierarchy to evolve independently.

You don't need to write code to support fancy output mechanisms; you can simply print a message to indicate that a shape has been output in some format. For example a message "A pixel is printed at (x, y)." will show that a pixel has been printed by a printer at (x, y).