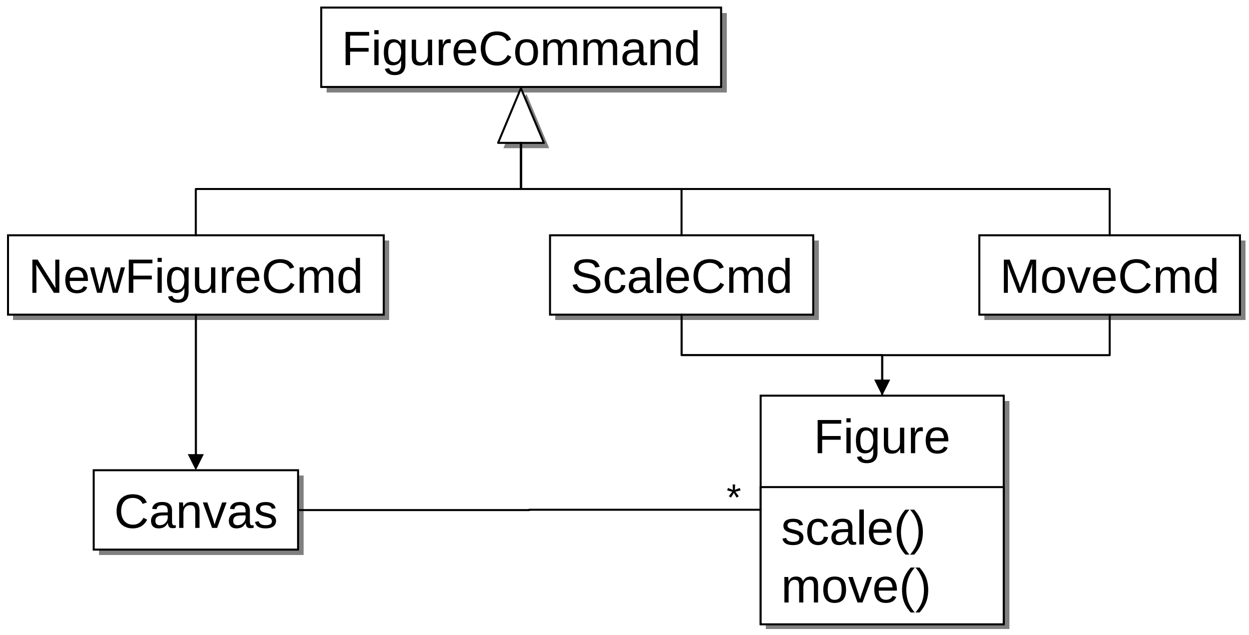
Design Patterns and Frameworks	**Exercise Sheet No. 7**
Dipl.-Inf. Steffen Zschaler	Software Engineering Group
INF 2097	Institute for Software and Multimedia Technology
`http://st.inf.tu-dresden.de/teaching/dpf`	Department of Computer Science
	Technische Universität Dresden
	01062 Dresden

Architecture Mismatch: Support Patterns

Task 1: Memorable Graphics

Many interactive applications require an undo mechanism so that tentative commands can be reverted and a redo mechanism so that such reversions can be undone again. This requires that some part of the application's state be stored and kept available for undoing modifications.

In this task we are going to design a graphical editing application. Thus, the state of the application consists of the various graphical elements, and relevant operations include creation, moving, and scaling of such elements. The following diagram shows an excerpt of the basic structure of the application.



1a)

Define an interface for `FigureCommand` that allows to enable support for undoing and redoing individual steps in editing a graphic.

1b)

To implement undo, we need to store the state of the currently selected figure before performing a change. Then, we can use this information to perform an undo. However, explicitly accessing a figure's state breaks encapsulation. What would be needed is something that allows us to hand out state information without breaking the class's state. What design pattern can we use to solve this problem and how would we do this?

Task 2: Cellular Automaton Vision

When we designed the cellular-automaton application, we had a similar problem of encapsulation as in the previous task: A panel displaying the current state of a cell grid, needs to access that state and needs to be aware of the general structure of the grid (number of dimensions, etc.) Exposing this information through the grid's interface, however, breaks encapsulation. How can we adapt the approach from the previous task to solve this issue?

Task 3: Notification on Time

Construct a `Timer` class that implements an endlessly ticking timer delivering events whenever the time changes. The timer ticks each 10 milliseconds. Use the EVENT NOTIFIER pattern [1] to allow clients of the timer to register for notifications every 10, 100, 1,000, or 60,000 milliseconds.

3a)

What is the structure of the EVENT NOTIFIER pattern?

3b)

Use the pattern to implement the `Timer`.

Bibliography

1. Dirk Riehle. *The Event Notification Pattern – Integrating Implicit Invocation with Object-Orientation.* In: Theory and Practice of Object Systems. 2, 1 (1996). http://www.riehle.org/computer-science/research/1996/tapos-1996-event.html