# GoF Roundup

Task 1: Memorable Graphics

In this task we are going to design a graphical editing application. The application drawing area of our application consists of a *Canvas*, which contains *Figures*. Currently, it is possible to add and remove figures, as well as to change their position and size. The picture shows an small part of the class diagram of our application.
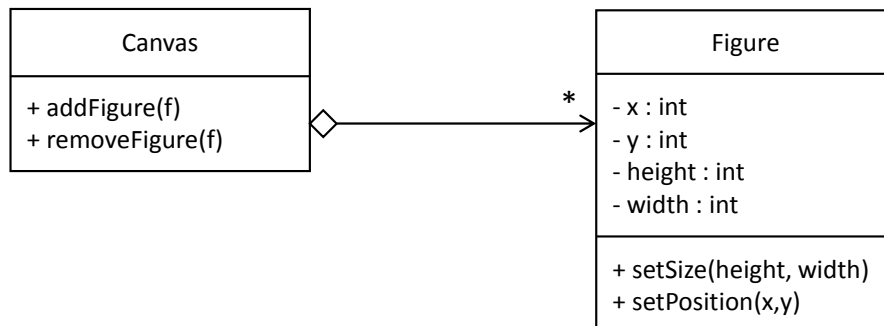
1a) We want to be able to support a macro functionality. This means, that we want to record all possible manipulation operations (i.e., create figure, remove figure, move figure, scale figure) and want to re-execute them in a script. Therefore, we need to be able handle manipulation operations as first-class-citizen of the application.

What design pattern can be used to objectify the manipulation operations? Draw a class diagram for the operations *create, remove, scale and move*, that support the macro functionality.

1b) Let's consider we want to integrate a third-party figure library which offers *FancyFigures*. Unfortunately, the class FancyFigure has a different interface, then our figures.
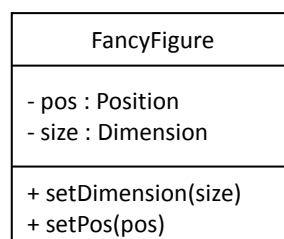

What design pattern can we apply, to integrate the FancyFigure class in our application, without changing the Figures interface? Draw a class diagram!

1c) Our interactive application requires 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.

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? Draw a class diagram.

Task 2: Cleaning Robots

Let's consider a modern production company that uses cleaning robots to keep the production halls nice and clean. Currently, two different kinds of robots are supported: Vacuum-Cleaning and Wet-Cleaning robots. As depicted in the figure, both robots share a same common functionality: When a new route is provided, they will drive to every point in this route (when there is no other robot currently) and clean that position. The driving and cleaning functionality however, depends on the concrete robot.
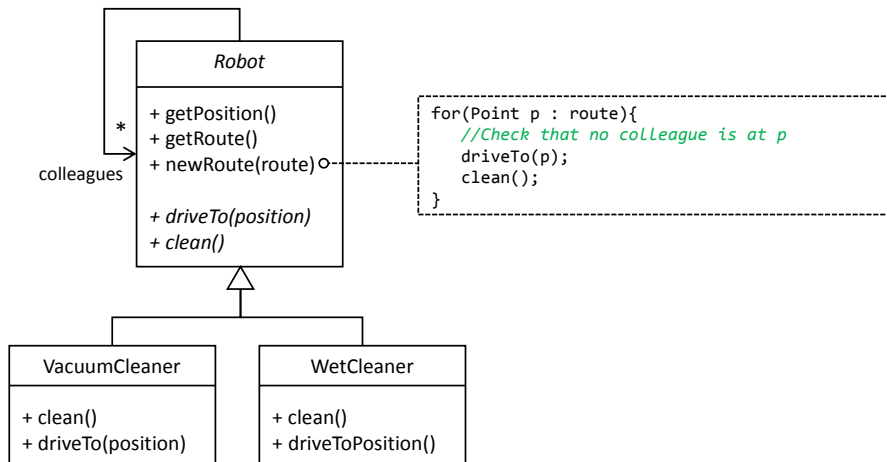
2a) What design pattern was used, to model both shared and robot-specific behavior in one class hierarchy? What are the pros and what are the cons?

2b) One of the main problems of mobile robot is the battery that runs empty very quickly. The robot manufacturer provides you the source code of a battery sensor component. This component receives the remaining power with a frequency of 10Hz via a serial interface. The robot class should be notified, when the battery level changes. In future, other software components (e.g., a graphical monitoring interface) should be notified as well.

What design pattern can be applied to decouple the battery sensor from the robot class, potentially supporting more listeners of the battery level? Change the class diagram accordingly.

2c) Now, the robot is notified, whenever the battery level changes. When the battery level is below a certain threshold, it should stop its normal cleaning behavior and should drive back to it's charging station.

What design pattern can be used to model the different states (normal and critical) of the battery? How can the normal behaviour of the `newRoute(route)` method be changed, according to the state? Change the class diagram accordingly.

2d) The company owns more than 100 robots, which manage a peer-to-peer connection to every other robot, to check their position and prevent collisions. Thus, more than 100000 network connections are active constantly. The local communication infrastructure cannot handle this amount of network traffic. The company asks you to change the current design of the application in a way, that not every robot has to manage a direct connection to all other robots.

What design pattern can be applied? Draw a class diagram.

2e) What drawback does this design introduce? Sketch possible solutions.