TDT4240
Exercise 2

Stein-Otto Svorstøl
and Andreas Drivenes

Spring 2015

Programming exercises

Handed in seperately. In task 2 we implemented the Pong-game with the singleton pattern, and in task 3 we implemented it with MVC. It's all one project, and one set of files as MVC and sngleton can easily be combined.

Task 3: Theory

a) 1) Observer: Design pattern
   2) State: Design pattern
   3) Template method: Design pattern
   4) Model View Controller: Architectural design patter
   5) Abstract factory: Design pattern
   6) Pipe-and-filter (pipeline): Can both be used as a architectural pattern, and a design pattern.

b) 1) The Singleton pattern restricts the creation of an object to a single instance across all computations. In our implementation of the Singleton pattern in the game Pong, the objects ScoreView, GameStateView, BallView, ScoreController, GameStateController and GameLayerController are singletons. We instantiate a private static variable in each of these classes and use a private constructor to restrict object creation. To get the running instance of an object, we can simply call BallView.getInstance(), or ScoreController.getInstance(). The advantage of this approach, is that we have control over the object creation, and we are absolutely sure that we only have one running instance.
   2) The Model-View-Controler architectural pattern defines how files should be places

c) The MVC-pattern has the major advantage of helping us to give different tasks to different people, as all code has a logical place to be, without us really having to define any real structure. This also helps making the code base more maintainable as it's easier ot find bugs or improve features for othher developers (they know where things 'should' be), more extendable (it's easy to give a controller a extra view for instance) and modular (logical places to put code that communicates with other systems for instance). We implemented it in the following way:

   (a) Models: We only made one model, as there is only one distinct object that has properites that may be modified by different controllers, and that is the player. The PlayerModel has properties for name and score. If the system was to work over the Intenret, then one may also add the coordinates in the server models.

   (b) Views: We moved all the sprites out to be views. They now do not have their own data, but gets data/are being controlled by the controllers.

   (c) Controllers: We made the State- and Layer-classes to be controllers, and gave them more power over the views. We also added Player-Controller which controls one player, and has the movement logic etc. (Model only has data, while view only shows the player on the screen.) The ScoreController is just a controller for just that, the score. IT does about the same thing as the "Scoreboard"-class we used before.