Pattern Designs

Observer/Observable pattern

This is the first pattern we implemented. We used this on the Score. The class GameLogic displays the score in the GUI, so it is beneficial to make GameLogic an observer of Score. Now every time Score is updated (when the user combines some Gems), GameLogic is notified and will update the Score in the GUI. This is done by the methods setChanged() and notifyObservers() in Score and the method update() in GameLogic.

Factory Pattern

Why factories?
Our Bejeweled game will make use of ‘special gems’. We therefore have to create several different kinds of Gems. This way we can centralize gem creation by means of the factory pattern instead of intantiating all kinds of Gems by using “new <GemName>”.

How?
We will make a GemFactory class which will take on the responsibilities of making Gems with the method createGem(int row, int col, GemType type, SpecialType special). Additionally we will make a new enum type: SpecialType. Depending on the SpecialType given to the createGem method, it will create a normal gem or one of the kinds of special Gem.