**Singleton Pattern**

* Software design pattern that restrict the instantiation of a class to one « single » instance. This is useful when exatcly one object is needed to coordinate actions across the system. Here in this game we want to implement only one game board. This pattern seems coherent in this situation.

Une image contenant texte, intérieur, toilette

Description générée automatiquement

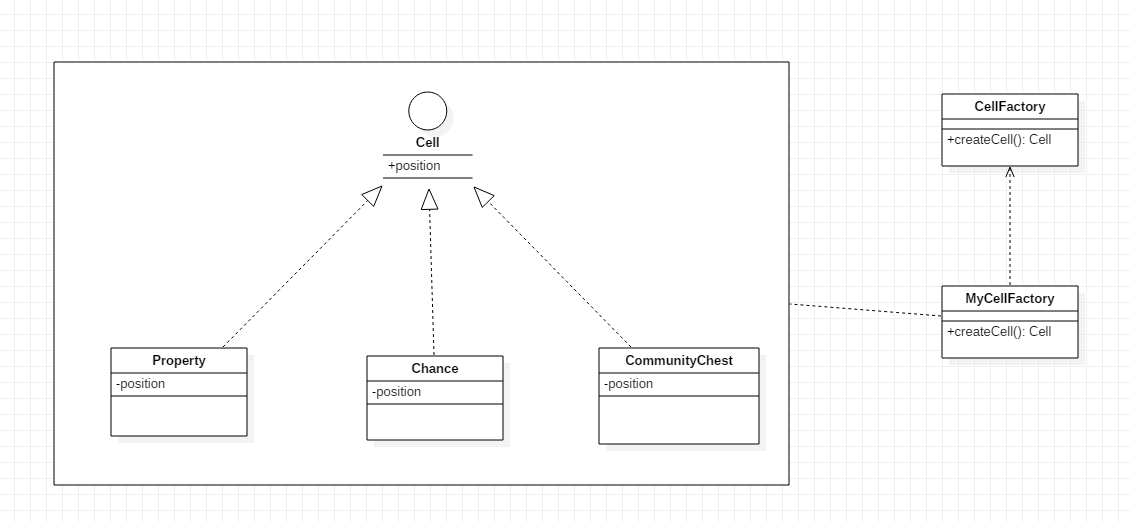
Une image contenant texte

Description générée automatiquementExemple de code (correction ex3 TD2) :

<https://www.youtube.com/watch?v=DcbxFugk5pM>

**Factory Pattern**

* Creational pattern that uses factory method to deal with the problem of creating objects without having to specify the exact class of the object that will be created. In our monopoly project, we want to create some objects without knowing exactly their type. Because they can have some differents behaviour during the game. We have several cases and depending on the position of these boxes, the case will be identified as a property or as a chance case or as a community case. It can be very interesting to implement this pattern to simplified the implementation of theses classes.

Example of codes :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquementUne image contenant texte

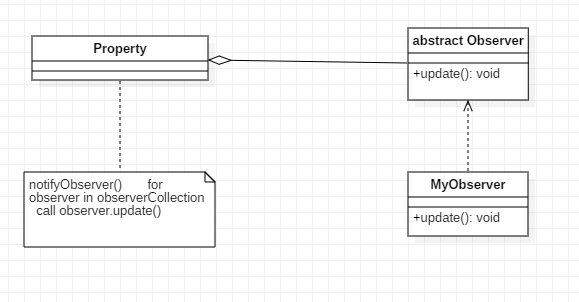
Description générée automatiquement

Une image contenant texte

Description générée automatiquement

**Observer Pattern**

* Software design pattern in which an object, named the subject, maintains a list of its dependants, called observers, and notifies them automatically of any state of changes, usually by calling one of their methods.



Example of code :

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement