Use of Design Patterns

Below are the design patterns we noticed were used in the classes of our project:

- **Information Expert**: is simply the general principle of assigning tasks or responsibilities to objects/classes which have the information necessary to fulfil the required responsibility.
  In the assigning of tasks for our classes, we referenced our models such as the class diagram, domain model, and sequence diagrams to decide which classes are responsible for certain tasks and to know which classes have the information required to handle certain tasks.
- **Controller**: this is the design pattern that helps to decide which class should take the responsibility of handling events generated from external actors. This pattern also helps in deciding if the class responsible for the input events should be one (façade) or many (session) controllers.
  For our classes, the GameBoard.java class bears the responsibility of managing events performed by external actors during the game play, and manages the logic of the game itself. We assume our controller to be one (façade) because it is the main coordinator of the game play. We acknowledge that if the game project were to be taken a step further, for example: a play between different computers on a network, then incorporating many (session) controllers could be considered.

And for the pattern that could be used is:

- **Indirection**: this pattern shows how we can avoid direct coupling between objects to keep coupling low and promote potential re-use of code. This is solved by assigning responsibility to intermediate classes/objects to create an indirection.
  In the case for our project, with more time and more understanding of the needed requirements we can avoid the direct access the main logic (GameBoard) has to other classes, we could build intermediate objects which will protect the inner designs of the classes against variation, and it will make each code more independent which will support the potential for re-use, for example: in a similar game idea, we could re-use some of the classes because they will be self-sufficient enough that they won't need another object for it to work.