Exercice Architecture :

Nous avons utilisé une architecture MVC :

- Le Modèle correspond au BoardGame
- La Vue correspond au Printer
- Le Contrôleur correspond à l'InputReader

Le tétris est en effet un jeu qui fait interagir un joueur avec des pièces qu'il doit placer sur un plateau.

L'idée est pour leur joueur de regarder l'état du plateau et de réagir en fonction de la pièce qu'il a à placer. Ces deux derniers éléments sont contenus dans le BoardGame : le Modèle et retransmis à l'écran via le Printer : la Vue qui va donc faire la passerelle entre l'état du Modèle et le joueur.

Le joueur va alors, pour placer sa pièce, donner des instructions qui seront reçues par l'InputReader : le Contrôleur. Ce dernier va traduire ces entrées en mouvements grâce à l'énumération Movement et les transmettre au BoardGame qui modifiera son état en fonctions des règles du jeu contenues dans son comportement (ses méthodes).

Exercice Design Pattern / Solid :

Lors de ce projet, nous nous sommes appuyés sur nos connaissances en java pour respecter au mieux les principes de la programmation orientée objet.

Comme on peut le voir sur l'architecture, notre jeu s'appuie sur trois éléments principaux :

- Le Boardgame qui représente le plateau de jeu : le support du tétris.
- L'inputReader qui va récupérer les entrées du joueur.
- Le Printer qui s'occupe de l'affichage du BoardGame.

Ces trois éléments sont des singletons ce qui nous assure qu'un seul et unique exemplaire de chacun pourra être instancié. Ce principe assure une certaine simplicité à notre architecture et évite tout conflit en termes d'appels.

Nous avons aussi appliqué plusieurs principes SOLID, notamment le « Single responsibility principle » qui stipule qu'une classe ne doit posséder qu'une et une seule responsabilité.

Par exemple, notre inputReader a la seule responsabilité de récupérer les entrées de l'utilisateur et y faire correspondre un mouvement. Ceci nous apporte aussi de la simplicité en évitant les redondances et assure que chaque classe aura une fonctionnalité précise.

Nous nous sommes aussi appuyés sur le « Interface segregation principle », ceci s'illustre dans notre projet avec l'objet MovingPiece qui est le seul à implémenter l'interface « Movable ». C'est en effet l'unique élément de notre jeu qui doit avoir la fonctionnalité de bouger. Cette interface nous permet de bien séparer les éléments pouvant bouger de ceux qui sont statiques.