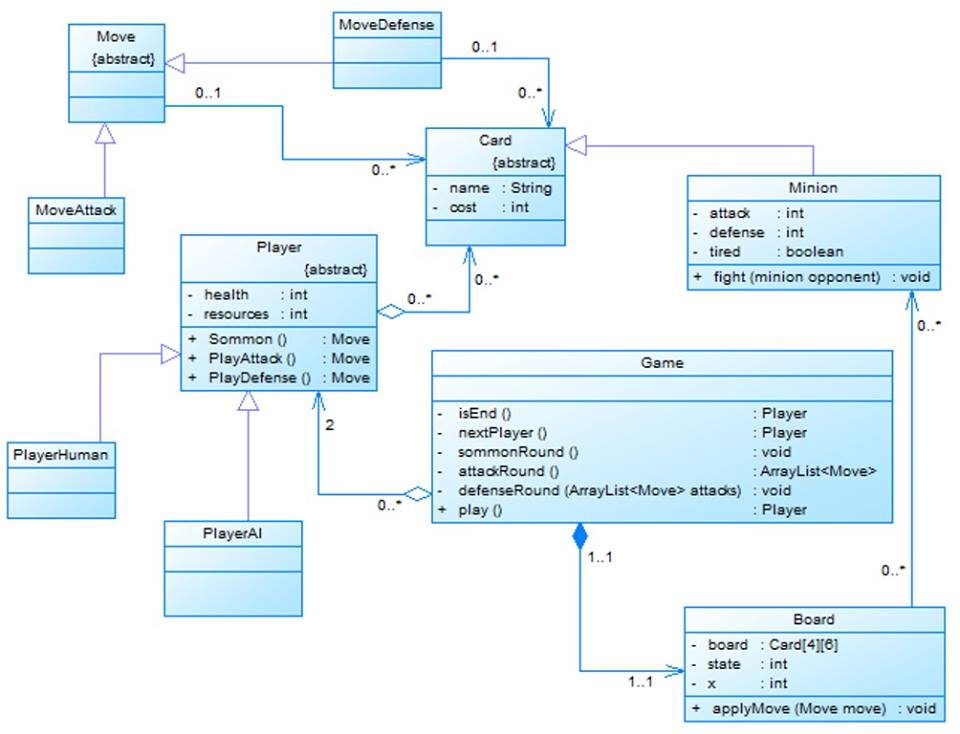
Projet – développement java

Application : jeu de cartes de type « combat de créatures »

# Modélisation Objet du problème et de l’implémentation :

Ci-dessous, le diagramme de classe et les détails des classes pour résoudre le problème…

## Diagramme de classe :



## Les classes :

* **Player** : (joueur) classe abstraite
  + Int : health = nombre de points de vie du joueur, il commence avec un total de 10 points de vie. Lorsque le joueur atteint 0 points de vie, il meurt.
  + ArrayList<Card> : deck = liste de cartes en main, le joueur possède seulement 4 cartes en main (il n’y a pas de pioche dans ce jeu).
  + Int : resources = nombre de points de ressources que possède le joueur, les points de ressources sont utilisés pour invoquer des cartes.
  + Move sommon (Card c) = fonction permettant au joueur d’invoquer la carte c sur le plateau de jeu.
  + Move playAttack() = Renvoie le coup correspondant à l’attaque du joueur. (fonction abstraite)
  + Move playDefense() = Renvoie le coup correspondant à la défense du joueur. (fonction abstraite)
* **PlayerHuman** : (joueur de type humain) héritant de la classe Player.
* **PlayerAI** : (joueur de type ordinateur) héritant de la classe Player.
* **Card** : (carte) classe abstraite
  + String : name = nom de la carte.
  + Int : cost = coût de la carte en ressources.
* **Minion** : (carte de type créature) héritant de la classe Card.
  + Int : attack = nombre de points d’attaque.
  + Int : defense = nombre de points de défense.
  + Boolean : tired = boolean indiquant si la carte est en état fatiguer (true) ou pas (false).
  + void fight () = fonction permettant à la carte de type créature d’attaquer.
* **Board** : (plateau de jeu)
  + Int : state = initialiser à 6. Nombre d’états différents dans lesquels peuvent être les cartes : main adverse, terrain adverse, ligne de combat adverse, ligne de combat actif, terrain actif, main actif.
  + Int : x = initialiser à 4. Défini le nombre de cartes pouvant être dans le même état.
  + Card[][] : board = tableau deux dimension de cartes représentant le plateau de jeu. De taille x\*state.
  + Move applyMove(Move m) = fonction permettant d’appliquer un coup m.
* **Game** : permet le bon fonctionnement d’une partie.
  + Player isEnd() = fonction permettant de voir si la partie est terminée, renvoie également le joueur qui gagne cette partie.
  + Player nextPlayer() = fonction renvoyant le joueur suivant pour gérer à qui est le tour de jouer.
  + void sommonRound() = fonction permettant de gérer le tour d’invocation du joueur courant.
  + void attackRound() = fonction permettant de gérer le tour d’attaque du joueur courant.
  + void defenseRound(ArrayList<Move> attacks) = fonction permettant de gérer le tour de défense du joueur courant en fonction des attaques reçues.
  + Player play() = fonction permettant de gérer le bon déroulement d’une partie, renvoie également le joueur gagnant cette partie.
* **Move** : (coup) classe abstraite
* **MoveAttack** : (coup de type attaque) héritant de Move.
* **MoveDefense** : (coup de type attaque) héritant de Move.