~/Documents/HEIG/24-25/POO/Lab08/moves.java

```java
package chess.moves;

/**
 * @author Lestiboudois Maxime & Parisod Nathan
 * @date 09/01/2025
 */

import chess.Board;
import chess.Square;

/**
 * Interface pour valider si un chemin entre deux cases sur un échiquier est dégagé.
 */
public interface PathValidator {
    /**
     * Vérifie si le chemin entre deux cases sur l'échiquier est libre de toute obstruction.
     *
     * @param board l'échiquier représentant l'état actuel du jeu.
     * @param start la case de départ.
     * @param end la case d'arrivée.
     * @return {@code true} si le chemin est dégagé, {@code false} sinon.
     */
    boolean isPathClear(Board board, Square start, Square end);
}
```

```java
package chess.moves;

/**
 * @author Lestiboudois Maxime & Parisod Nathan
 * @date 09/01/2025
 */

import chess.Board;
import chess.Square;

/**
 * Vérifie si le chemin entre deux cases sur l'échiquier est dégagé.
 * Implémente l'interface {@link PathValidator}.
 *
 * Lève une exception {@link IllegalArgumentException} si la case de départ
 * est identique à la case d'arrivée.</p>
 */
public class DefaultPathValidator implements PathValidator {
    /**
     * Vérifie si le chemin entre deux cases est dégagé.
     *
     * @param board l'état actuel de l'échiquier.
     * @param start la case de départ.
     * @param end la case d'arrivée.
     * @return {@code true} si le chemin est dégagé, {@code false} sinon.
     * @throws IllegalArgumentException si la case de départ est identique à la case d'arrivée.
     */
    @Override
    public boolean isPathClear(Board board, Square start, Square end) {
        int stepX = end.getX() - start.getX();
        int stepY = end.getY() - start.getY();

        if(stepX == 0 && stepY == 0) {
            throw new IllegalArgumentException("Aucun déplacement");
        }

        int x = (stepX > 0) ? start.getX() + 1 : (stepX == 0) ? end.getX() : start.getX() - 1;
```

```
        int y = (stepY > 0) ? start.getY() + 1 : (stepY == 0) ? end.getY() : start.getY() - 1;

        while (x != end.getX() || y != end.getY()) {
            if (board.getSquare(x, y).isOccupied()) {
                return false; // Une pièce bloque le chemin
            }

            x = (stepX > 0) ? x + 1 : (stepX == 0) ? end.getX() : x - 1;
            y = (stepY > 0) ? y + 1 : (stepY == 0) ? end.getY() : y - 1;
        }

        return true;
    }
}


package chess.moves;

/**
 * @author Lestiboudois Maxime & Parisod Nathan
 * @date 09/01/2025
 */

/**
 * Représente les différents types de déplacements possibles pour les pièces d'échecs.
 * Chaque type de déplacement vérifie si les coordonnées fournies respectent ses règles spécifiques.
 */
public enum MoveType {
    /**
     * Déplacement en diagonale.
     * Valide si les deux décalages sont égaux et strictement positifs.
     */
    DIAGONAL {
        @Override
        public boolean isValid(int deltaX, int deltaY) {
            return deltaX == deltaY && deltaX > 0;
        }
    },
    /**
     * Déplacement horizontal.
     * Valide si le décalage vertical est nul et le décalage horizontal est strictement positif.
     */
    HORIZONTAL {
        @Override
        public boolean isValid(int deltaX, int deltaY) {
            return deltaY == 0 && deltaX > 0;
        }
    },
    /**
     * Déplacement vertical.
     * Valide si le décalage horizontal est nul et le décalage vertical est strictement positif.
     */
    VERTICAL {
        @Override
        public boolean isValid(int deltaX, int deltaY) {
            return deltaX == 0 && deltaY > 0;
        }
    },
    /**
     * Déplacement en forme de "L".
     * Valide si les décalages correspondent aux mouvements possibles d'un cavalier (2x1 ou 1x2).
     */
    L_SHAPE {
        @Override
```

```
        public boolean isValid(int deltaX, int deltaY) {
            return (deltaX == 2 && deltaY == 1) || (deltaX == 1 && deltaY == 2);
        }
    };
    /**
     * Vérifie si un déplacement est valide pour ce type de mouvement.
     *
     * @param deltaX le décalage horizontal entre la case de départ et la case d'arrivée.
     * @param deltaY le décalage vertical entre la case de départ et la case d'arrivée.
     * @return {@code true} si le déplacement respecte les règles du type de mouvement, {@code false} sinon.
     */
    public abstract boolean isValid(int deltaX, int deltaY);
}
```

```
package chess.moves;

/**
 * @author Lestiboudois Maxime & Parisod Nathan
 * @date 09/01/2025
 */

import chess.Board;
import chess.Square;

/**
 * Interface représentant un coup spécial dans un jeu d'échecs
 * (par exemple, roque ou prise en passant).
 */
public interface SpecialMove {
    /**
     * Vérifie si le coup spécial est valide pour l'état actuel de l'échiquier.
     *
     * @param board l'échiquier représentant l'état actuel du jeu.
     * @param start la case de départ du mouvement.
     * @param end la case d'arrivée du mouvement.
     * @return {@code true} si le coup spécial est valide, {@code false} sinon.
     */
    boolean isValid(Board board, Square start, Square end);

    /**
     * Exécute le coup spécial sur l'échiquier.
     *
     * @param board l'échiquier représentant l'état actuel du jeu.
     * @param start la case de départ du mouvement.
     * @param end la case d'arrivée du mouvement.
     */
    void execute(Board board, Square start, Square end);
}
```