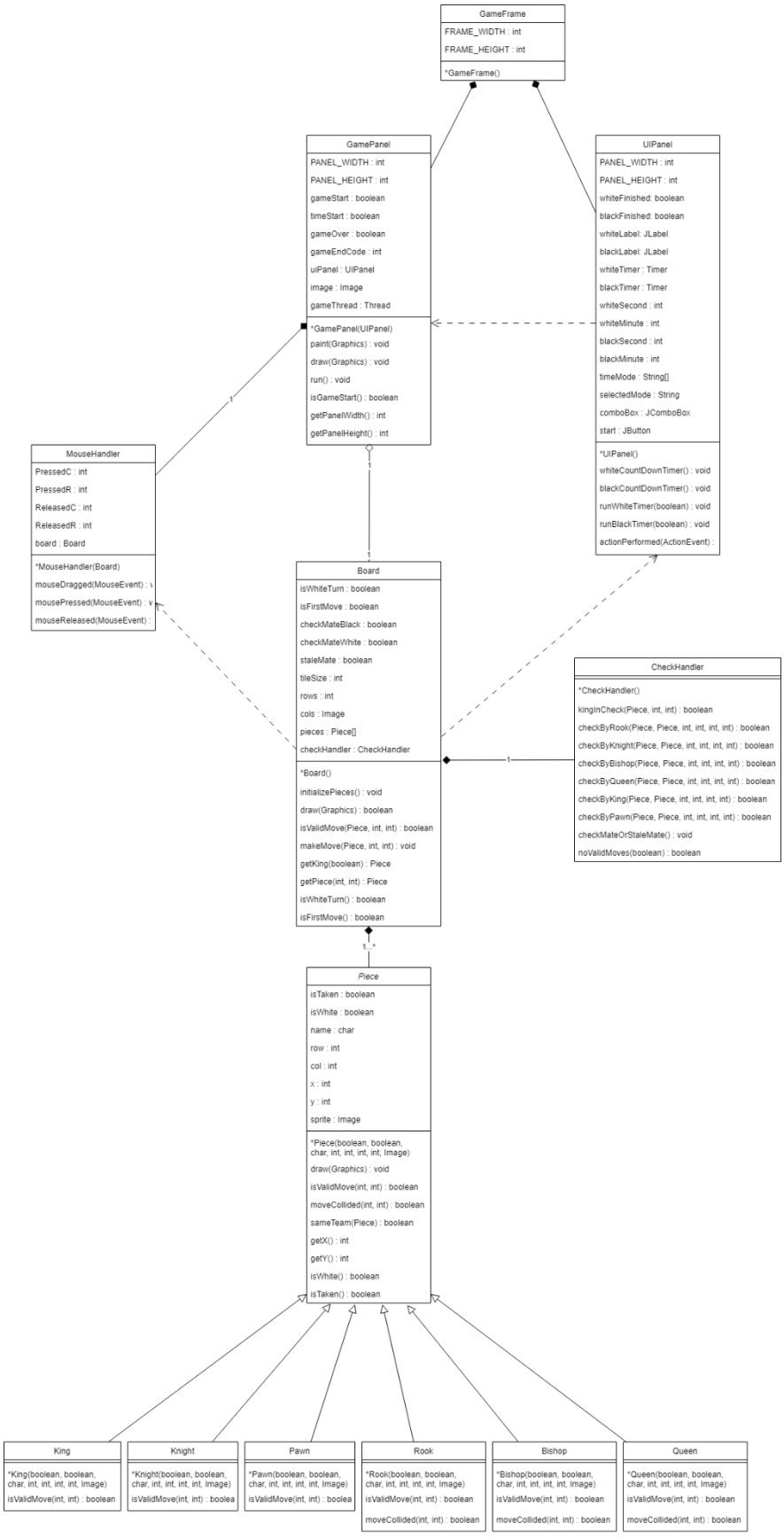
Exercise 1

1. Entities:
    - **Game Frame**: Responsible for handling the frame the game will run in
    - **Game Panel**: Responsible for handling the panel on which the board will be drawn. Serves almost like a canvas. It will also run a thread that will repaint this panel 60 times per second.
    - **UI Panel**: Responsible for handling a panel for User Interface with buttons, combo boxes, and the timers.
    - **Board**: Responsible for holding a 2d array of type Piece that will contain all the pieces of the board and their appropriate location.
    - **Mouse Handler**: Responsible for handling mouse interaction with the program.
    - **Check Handler:** Responsible for testing if a move will leave the king in check and to continuously test if any of the kings are under checkmate or stalemate.
    - **Piece:** An abstract entity that will hold the properties of every piece.
    - **Knight:** Responsible for handling knight movement.
    - **Bishop:** Responsible for handling bishop movement.
    - **Rook:** Responsible for handling rook movement.
    - **Queen:** Responsible for handling queen movement.
    - **King:** Responsible for handling king movement.
    - **Pawn:** Responsible for handling pawn movement.

2. Relationships between entities in this case consist of dependency, composition, aggregation and inheritance. Both panels have a composition relationship with the game frame, because the existence of the panel depends completely on the frame. The relationship between the board and the GamePanel, however, is of aggregation. As a board can exist on its own without a panel. Furthermore, the relationship between the Piece entity and the board is of composition, as the pieces will only exist if there is board available to hold them, and the number of pieces that can exist in a board goes from 0 to n. This same Piece class is abstract and will be the parent class of the following subclasses (meaning that they will inherit Piece's methods): Rook, Knight, Bishop, Queen, King, Pawn. The CheckHandler relationship with the Board will be of composition, as this can only exist if there is a board also present. And this same logic can be applied to the MouseHandler and GamePanel, as the nature of their relationship is also of composition. Finally, there are some dependencies between the entities too, like how the GamePanel depends on the UIPanel, the UIPanel on the board (to figure out which player has their turn), and how the MouseHandler depends on the Board's contents as well. These relationships and entities can be better understood on the following exercise.

3.

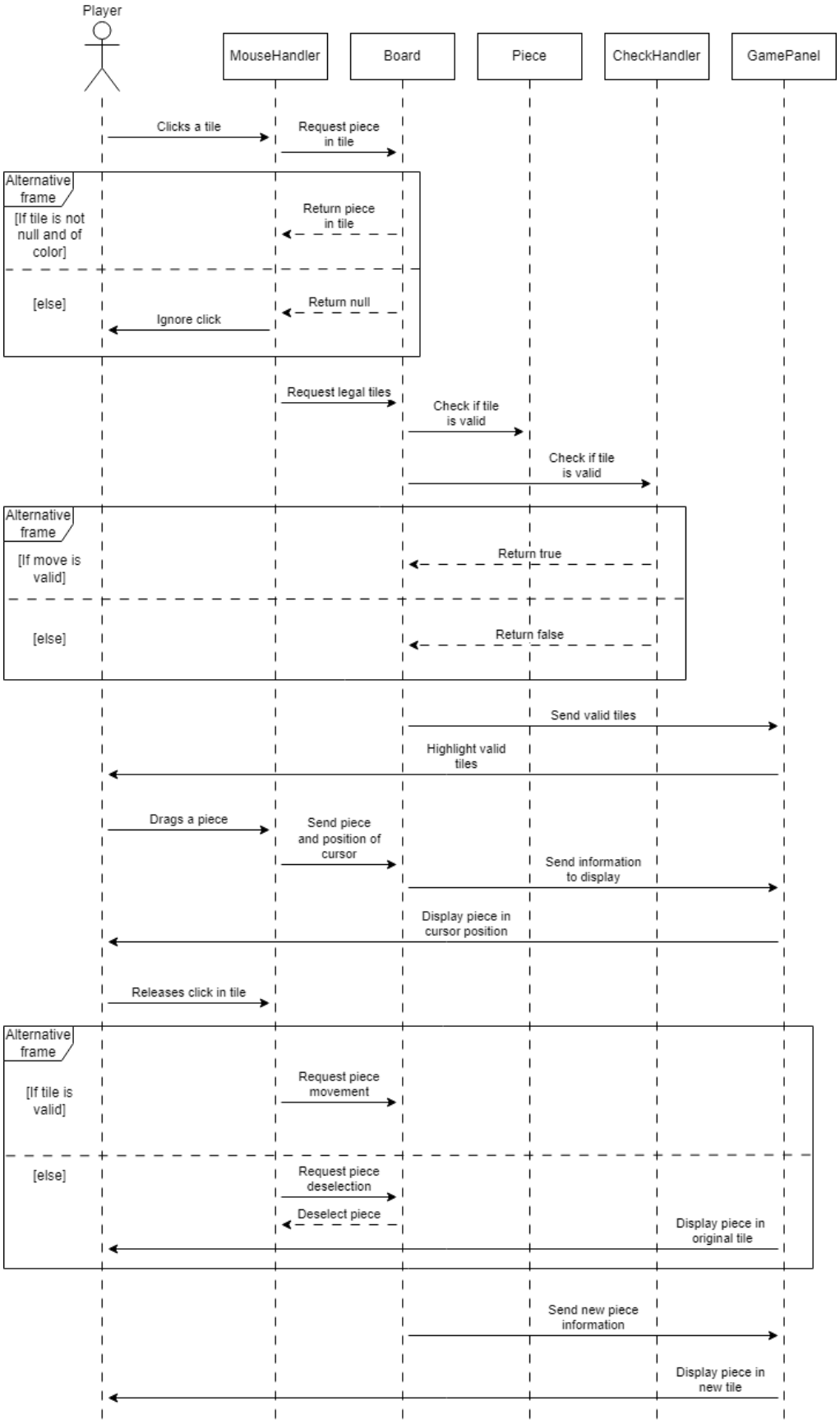

GameFrame
FRAME_WIDTH : int
FRAME_HEIGHT : int
*GameFrame()
GamePanel
PANEL_WIDTH : int
PANEL_HEIGHT : int
gameStart : boolean
timeStart : boolean
gameOver : boolean
gameEndCode : int
uiPanel : UIPanel
image : Image
gameThread : Thread
*GamePanel(UIPanel)
paint(Graphics) : void
draw(Graphics) : void
run() : void
isGameStart() : boolean
getPanelWidth() : int
getPanelHeight() : int
UIPanel
PANEL_WIDTH : int
PANEL_HEIGHT : int
whiteFinished: boolean
blackFinished: boolean
whiteLabel: JLabel
blackLabel: JLabel
whiteTimer : Timer
blackTimer : Timer
whiteSecond : int
whiteMinute : int
blackSecond : int
blackMinute : int
timeMode : String[]
selectedMode : String
comboBox : JComboBox
start : JButton
*UIPanel()
whiteCountDownTimer() : void
blackCountDownTimer() : void
runWhiteTimer(boolean) : void
runBlackTimer(boolean) : void
actionPerformed(ActionEvent) :
MouseHandler
PressedC : int
PressedR : int
ReleasedC : int
ReleasedR : int
board : Board
*MouseHandler(Board)
mouseDragged(MouseEvent) : v
mousePressed(MouseEvent) : v
mouseReleased(MouseEvent) :
Board
isWhiteTurn : boolean
isFirstMove : boolean
checkMateBlack : boolean
checkMateWhite : boolean
staleMate : boolean
tileSize : int
rows : int
cols : Image
pieces : Piece[]
checkHandler : CheckHandler
*Board()
initializePieces() : void
draw(Graphics) : boolean
isValidMove(Piece, int, int) : boolean
makeMove(Piece, int, int) : void
getKing(boolean) : Piece
getPiece(int, int) : Piece
isWhiteTurn() : boolean
isFirstMove() : boolean
CheckHandler
*CheckHandler()
kingInCheck(Piece, int, int) : boolean
checkByRook(Piece, Piece, int, int, int, int) : boolean
checkByKnight(Piece, Piece, int, int, int, int) : boolean
checkByBishop(Piece, Piece, int, int, int, int) : boolean
checkByQueen(Piece, Piece, int, int, int, int) : boolean
checkByKing(Piece, Piece, int, int, int, int) : boolean
checkByPawn(Piece, Piece, int, int, int, int) : boolean
checkMateOrStaleMate() : void
noValidMoves(boolean) : boolean
Piece
isTaken : boolean
isWhite : boolean
name : char
row : int
col : int
x : int
y : int
sprite : Image
*Piece(boolean, boolean, char, int, int, int, int, Image)
draw(Graphics) : void
isValidMove(int, int) : boolean
moveCollided(int, int) : boolean
sameTeam(Piece) : boolean
getX() : int
getY() : int
isWhite() : boolean
isTaken() : boolean
King
*King(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolean
Knight
*Knight(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolea
Pawn
*Pawn(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolea
Rook
*Rook(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolean
moveCollided(int, int) : boolean
Bishop
*Bishop(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolean
moveCollided(int, int) : boolean
Queen
*Queen(boolean, boolean, char, int, int, int, int, Image)
isValidMove(int, int) : boolean
moveCollided(int, int) : boolean
1
1
1
1
1
1..*

Exercise 2

Use case: Move piece.

1. Objects involved:
    - **MouseHandler:** To detect mouse interaction.
    - **Board:** To update piece information.
    - **Piece:** To handle tile validity.
    - **CheckHandler:** To handle further tile validity.
    - **GamePanel:** To display moved pieces to the user.

2&3.


Player
MouseHandler
Board
Piece
CheckHandler
GamePanel
Clicks a tile
Request piece in tile
Alternative frame
[If tile is not null and of color]
Return piece in tile
[else]
Return null
Ignore click
Request legal tiles
Check if tile is valid
Check if tile is valid
Alternative frame
[If move is valid]
Return true
[else]
Return false
Send valid tiles
Highlight valid tiles
Drags a piece
Send piece and position of cursor
Send information to display
Display piece in cursor position
Releases click in tile
Alternative frame
[If tile is valid]
Request piece movement
[else]
Request piece deselection
Deselect piece
Display piece in original tile
Send new piece information
Display piece in new tile