**Exercise 1**

1.

Diagram, schematic

Description automatically generatedi.

ii. We drew the structure of the project with all its packages and classes, as well as connecting the packages with packages, and classes with classes to show their dependencies. Each package entails all its classes and interfaces, as well as for example the subclass “ghost” in “npc”. The runnable class is the Launcher, which displays many dependencies on the packages, as well.

iii. From the packages and classes we understand that there is the package “board” including the BoardFactory, which is constructing a board for the game with input from board layout, directions, and the abstract classes squares and units. Secondly, in the package “ game” there is the GameFactory, an abstract class “Game” and the SinglePlayerGame. Given that there is no class for MultiplayerGame, one may assume that the programme is a single player’s game only. Thirdly, the package “level” exists, including a LevelFactory and a PlayerFactory relating to the classes CollisionInteractionMap, DefaultPlayerInteractionMap, Level, MapParser, Pellet, Player, PlayerCollisions, as well as the interface CollisionMap. Fourthly, the package of the non-player character (npc) entailing the package “Ghost”, includes the abstract class “Ghost”, as well as the classes Blinky, Clyde, Inky, and Pinky (the ghosts), as well as the Ghost Factory and the final class Navigation, which means that this class can’t be inherited nor extended by other classes, implying that the navigation of the ghosts is unique and solely for the purpose of moving the ghosts. Furthermore, the Enum-class GhostColor contains unchangeable variables, so it can be assumed that the colors of each ghost does not change. The package also includes information about the package, stored in package-info.java. Fifthly, the package Sprites entails the classes AnimatedSprite, EmptySprite, ImageSprite, PacManSprites, and the SpriteStore, depicting the sprites in different shapes depending on the game’s context, as well as the interface Sprite, drawing the sprite. Lastly, the package ui (user interface) contains the BoardPanel, ButtonPanel, PacKeyListener, PacManUI, PacManUIBuilder, and ScorePanel, headed by the interface Action. These packages and classes build up to the Launcher of the game, and a predefined PacManConfigurationExemption in case the game cannot be loaded properly.

2.

i.

Diagram, schematic

Description automatically generated

ii. We started drawing the call graph with the main class of the Launcher as an entry point, as well as depicting every level of the game called within the Launcher, to demonstrate the complexity and the multilayeredness of the game.

iii. From the graph we can see that the launcher splits up in three paths: getGame, addSinglePlayerKeys, and makeGame. The addSinglePlayerKeys probably registers the user input via the keyboard and puts it into a direction (moveTowardsDirection), which in turn gets the player (getSinglePlayer). Additionally, to launch, the makeTowardsDirection is connected to getGame, implying that the move is part of the initialization of the game. To construct a game, the program uses makeGame, which in turn is depending on makeLevel and getGameFactory, and thus constructs the level and game over its level and its users from the GameFactory. The latter continues by getting the players from the getPlayerFactory, and the assigning of the Sprite in the getSpriteStore, where the path of the getGameFactory will be reunited with the makeGame path. Following the makeLevel path, it gets the map of the level (getLevelMap), as well as the map parser (getMapParser), which in turn creates a board (getBoardFactory) using the sprite store (getSpriteStore). The level factory (getLevelFactory) draws on the Sprite Store (getSpriteStore), as well as the ghost factory (getGhostFactory), which is within the path of the level factory.

**Exercise 2**

1.

**Step 1: Identify Classes.**

* Identifying the noon phrases and separating them into obvious classes, uncertain classes and nonsense classes.

|  |  |  |
| --- | --- | --- |
| **Noun Phrases** | | |
| **Obvious Classes** | **Uncertain candidates** | **Nonsense** |
| * Game * Board * Validity of the move * Simple move * Jump move * Multiple jump move * Game Won | * User/player * Move * Piece * Position * Checkers * Square * Character * Color * Red * White * Piece’s type * Pawn * King * Current piece position * Future piece position * King-row | * Computer * Corner * Terminal * Program * Row * Column * Version * Checkerboard * Round * Double-corner * Pair of dark squares * Player’s edge of the board * Opponent * Rules |

* From the noun phrases, eliminate the potential classes to the candidate ones

|  |
| --- |
| **Candidate Classes** |
| * Game * Board * Move * Validity of the move * Simple move * Jump move * Multiple jump move * Game Won * Player * Piece * Character * Piece’s color * Piece’s type * Pawn * King * King-row |

**Step 2: Determine the Responsibilities of each class**

* Identifying the verbs that determines-responsibilities for all the candidate classes

|  |
| --- |
| **Class:** Players |
| **Responsibilities** |
| * Input a move until they enter a valid one on the terminal * Loses the game |

|  |
| --- |
| **Class:** RunGame |
| **Responsibilities** |
| * Starts the game * Display on the terminal the board with the pieces in their initial positions * Asks the player to enter their move until they enter a valid one * Checks player’s turn |

|  |
| --- |
| **Class:** ValidateMove |
| **Responsibilities** |
| * Checking the validity of the player’s move |

|  |
| --- |
| **Class:** Move |
| **Responsibilities** |
| * Performs a simple move * Performs a jump * Performs a multiple jump * Performs player’s valid move |

|  |
| --- |
| **Class:** Simple\_Move |
| **Responsibilities** |
| * Does not capture opponent’s piece * Moves piece to a vacant dark square * Moves pawn diagonally forward away from the player * Moves king diagonally backward toward or diagonally forward away the player |

|  |
| --- |
| **Class:** Jump |
| **Responsibilities** |
| * Checks if any piece can make a jump move * Captures opponent’s piece * Moves pawn 2 squares diagonally forward away from the player * Moves king 2 squares diagonally backward toward or forward away or from the player |

|  |
| --- |
| **Class:** Multiple\_Jump |
| **Responsibilities** |
| * Makes more than one jumps with the same piece * Captures one of the opponents’ piece through each jump * After one jump checks if there is option for another jump |

|  |
| --- |
| **Class:** First Character |
| **Responsibilities** |
| * Represent the piece’s color |

|  |
| --- |
| **Class:** Color |
| **Responsibilities** |
| * Represents a player * Define the direction of a piece’s move |

|  |
| --- |
| **Class:** Second Character |
| **Responsibilities** |
| * Represent the piece’s type |

|  |
| --- |
| **Class:** Piece’s type |
| **Responsibilities** |
| * Represent if a piece is pawn or king |

|  |
| --- |
| **Class:** Piece |
| **Responsibilities** |
| * Piece’s color. * Piece’s Type. * Represent a player. * Moves on the board from their current to their future position. * Moves to vacant dark squares. |

|  |
| --- |
| **Class:** Board |
| **Responsibilities** |
| * Stores the board with the default positions of the pieces. * Prints the updated version of the board (with the new positions of the pieces based on a valid move). * Displays if a piece is captured (removes it from the updated version). |

|  |
| --- |
| **Class:** Pawn |
| **Responsibilities** |
| * Moves to a vacant dark square. * Makes simple move (Moves diagonally forward away from the player). * Makes jump move (Jumps diagonally forward away from the player). * Makes multiple jump move . * Captures the opponents’ piece. |

|  |
| --- |
| **Class:** King (Move) |
| **Responsibilities** |
| * Moves to a vacant dark square. * Moves diagonally forward away from the player. * Moves diagonally backward toward the player. * Jumps diagonally forward away from the player. * Jumps diagonally backwards toward the player. * Make multiple jump move of any combination forward or backward jumps. * Capture the opponents’ piece. |

|  |
| --- |
| **Class:** King-Row (Move) |
| **Responsibilities** |
| * Is the furthest row of each player. * When the pawn goes there , becomes a king. |

|  |
| --- |
| **Class:** GameWon |
| **Responsibilities** |
| * Check if the game has been finished.   + Check if a player has 0 pieces.   + Check if the player whose turn is, cannot move. * Prints the winner. |

* Exploring scenarios and eliminate the cards by removing the unnecessary ones.

|  |
| --- |
| **Class:** RunGame |
| **Responsibilities** |
| * Starts the game. * Display on the terminal the board with the pieces in their initial positions. * Asks the player to enter their move until they enter a valid one * Checks player’s turn. |

|  |
| --- |
| **Class:** ValidateMove |
| **Responsibilities** |
| * Checking the validity of the player’s move. |

|  |
| --- |
| **Class:** Move |
| **Responsibilities** |
| * Performs a simple move. * Performs a jump. * Performs a multiple jump. * Performs player’s valid move. |

|  |
| --- |
| **Class:** Simple\_Move |
| **Responsibilities** |
| * Does not capture opponent’s piece. * Moves piece to a vacant dark square. * Moves pawn diagonally forward away from the player. * Moves king diagonally backward toward or diagonally forward away the player. |

|  |
| --- |
| **Class:** Jump |
| **Responsibilities** |
| * Checks if any piece can make a jump move. * Captures opponent’s piece. * Moves pawn 2 squares diagonally forward away from the player. * Moves king 2 squares diagonally backward toward or forward away or from the player. |

|  |
| --- |
| **Class:** Multiple\_Jump |
| **Responsibilities** |
| * Makes more than one jumps with the same piece. * Captures one of the opponent’s piece through each jump. * After one jump checks if there is option for another jump. |

|  |
| --- |
| **Class:** Piece |
| **Responsibilities** |
| * Piece’s color * Piece’s Type * Represent a player * Moves on the board from their current to their future position. * Moves to vacant dark squares. |

|  |
| --- |
| **Class:** Board |
| **Responsibilities** |
| * Stores the board with the default positions of the pieces. * Prints the updated version of the board (with the new positions of the pieces based on a valid move). * Displays if a piece is captured (removes it from the updated version). |

|  |
| --- |
| **Class:** Pawn |
| **Responsibilities** |
| * Moves to a vacant dark square. * Makes simple move (Moves diagonally forward away from the player). * Makes jump move (Jumps diagonally forward away from the player). * Makes multiple jump move ... * Captures the opponents’ piece |

|  |
| --- |
| **Class:** King (Move) |
| **Responsibilities** |
| * Moves to a vacant dark square. * Moves diagonally forward away from the player. * Moves diagonally backward toward the player. * Jumps diagonally forward away from the player. * Jumps diagonally backwards toward the player. * Makes multiple jumps move of any combination forward or backward jumps. * Captures the opponents’ piece. |

|  |
| --- |
| **Class:** GameWon |
| **Responsibilities** |
| * Check if the game has been finished.   + Check if a player has 0 pieces.   + Check if the player whose turn is, cannot move. * Prints the winner |

**Step 3: Identifying Collaborations.**

* Checking which classes cannot fulfill their responsibilities by themselves and connecting them with other classes that can help the first ones to do it.

|  |  |
| --- | --- |
| **Class:** RunGame | **Superclass:** |
| **Responsibilities** | **Collaborators: Board** |
| * Starts the game. * Displays on the terminal the board with the pieces in their initial positions. * Asks the player to enter their move until they enter a valid one. * Checks player’s turn. | * Board * GameWon * Move * ValidateMove |

|  |  |
| --- | --- |
| **Class:** ValidateMove | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Checking the validity of the player’s move. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Move | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Performs a simple move. * Performs a jump. * Performs a multiple jump. * Performs player’s valid move. | * RunGame * ValidateMove * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** Simple\_Move | **Superclass:** Move |
| **Responsibilities** | **Collaborators:** |
| * Moves piece to a vacant dark square. * Moves pawn 1 square diagonally forward away from the player. * Moves king 1 square diagonally backward toward or diagonally forward away the player. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Jump | **Superclass: Move** |
| **Responsibilities** | **Collaborators:** |
| * Checks if any piece can make a jump move. * Captures opponent’s piece. * Moves pawn 2 squares diagonally forward away from the player. * Moves king 2 squares diagonally backward toward or forward away or from the player. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Multiple\_Jump | **Superclass: Move** |
| **Responsibilities** | **Collaborators:** |
| * Performs multiple jumps. * After one jump checks if there is option for another jump. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Piece | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Piece’s color * Piece’s Type * Moves on the board from their current to their future position. * Moves to vacant dark squares. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Board | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Stores the board with the default positions of the pieces. * Prints the updated version of the board (with the new positions of the pieces based on a valid move). * Displays if a piece is captured (removes it from the updated version). | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Pawn | **Superclass:** Piece |
| **Responsibilities** | **Collaborators:** |
| * Moves to a vacant dark square. * Makes simple move (Moves diagonally forward away from the player). * Makes jump move (Jumps diagonally forward away from the player) * Makes multiple jump move . * Captures the opponents’ piece. | * RunGame * Move * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** King | **Superclass: Piece** |
| **Responsibilities** | **Collaborators:** |
| * Moves to a vacant dark square * Moves diagonally forward away from the player * Moves diagonally backward toward the player * Jumps diagonally forward away from the player * Jumps diagonally backwards toward the player * Make multiple jumps move of any combination forward or backward jumps * Captures the opponents’ piece | * RunGame * Move * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** GameWon | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Check if the game has been finished   + Check if a player has 0 pieces   + Check if the player whose turn is, cannot move * Prints the winner | * RunGame |

Step 4: Final Cards

* Removing/ merging cards and deriving to the final cards.

|  |  |
| --- | --- |
| **Class:** RunGame | **Superclass:** |
| **Responsibilities** | **Collaborators: Board** |
| * Starts the game. * Displays on the terminal the board with the pieces in their initial positions. * Asks the player to enter their move until they enter a valid one. * Checks player’s turn. | * Board * GameWon * Move * ValidateMove |

|  |  |
| --- | --- |
| **Class:** ValidateMove | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Checking the validity of the player’s move. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Move | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Performs a simple move. * Performs a jump. * Performs a multiple jump. * Performs player’s valid move. | * RunGame * ValidateMove * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** Simple\_Move | **Superclass:** Move |
| **Responsibilities** | **Collaborators:** |
| * Moves piece to a vacant dark square. * Moves pawn 1 square diagonally forward away from the player. * Moves king 1 square diagonally backward toward or diagonally forward away the player. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Jump | **Superclass: Move** |
| **Responsibilities** | **Collaborators:** |
| * Checks if any piece can make a jump move. * Captures opponent’s piece. * Moves pawn 2 squares diagonally forward away from the player. * Moves king 2 squares diagonally backward toward or forward away or from the player. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Multiple\_Jump | **Superclass: Move** |
| **Responsibilities** | **Collaborators:** |
| * Performs multiple jumps. * After one jump checks if there is option for another jump. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Piece | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Piece’s color * Piece’s Type * Moves on the board from their current to their future position. * Moves to vacant dark squares. | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Board | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Stores the board with the default positions of the pieces. * Prints the updated version of the board (with the new positions of the pieces based on a valid move). * Displays if a piece is captured (removes it from the updated version). | * RunGame * Move |

|  |  |
| --- | --- |
| **Class:** Pawn | **Superclass:** Piece |
| **Responsibilities** | **Collaborators:** |
| * Moves to a vacant dark square. * Makes simple move (Moves diagonally forward away from the player). * Makes jump move (Jumps diagonally forward away from the player) * Makes multiple jump move . * Captures the opponents’ piece. | * RunGame * Move * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** King | **Superclass: Piece** |
| **Responsibilities** | **Collaborators:** |
| * Moves to a vacant dark square * Moves diagonally forward away from the player * Moves diagonally backward toward the player * Jumps diagonally forward away from the player * Jumps diagonally backwards toward the player * Make multiple jump move of any combination forward or backward jumps * Captures the opponents’ piece | * RunGame * Move * SimpleMove * Jump * MultipleJump |

|  |  |
| --- | --- |
| **Class:** GameWon | **Superclass:** |
| **Responsibilities** | **Collaborators:** |
| * Check if the game has been finished   + Check if a player has 0 pieces   + Check if the player whose turn is, cannot move * Prints the winner | * RunGame |

2.

Main classes:

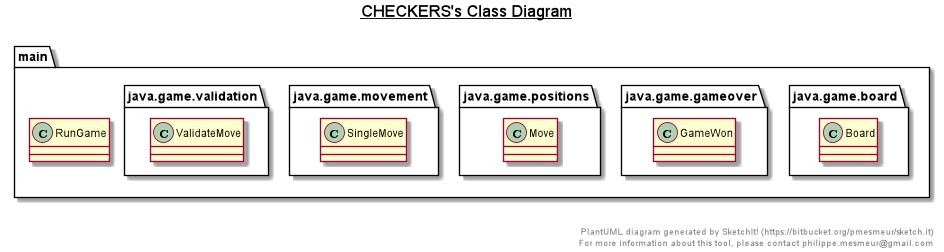
* fff
* ffff
* ffff
* fff
* fff

3.

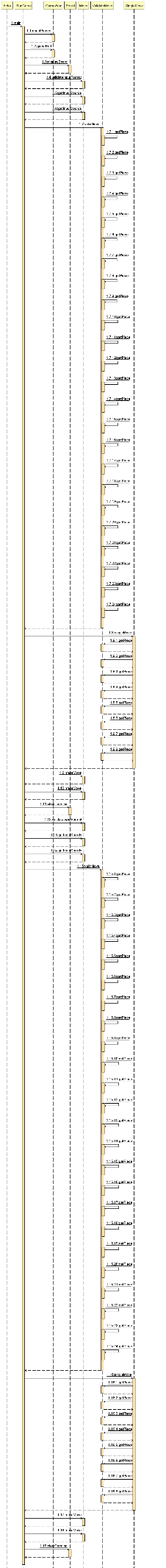
Less important classes:

* uuiu
* uuu
* uuu
* uuu
* uuu

4.



5.



5.