
# Skeleton Code Breakdown

## *Class: Dastan*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | **R** : Int<br>**C** : Int<br>**NoOfPieces** : Int | Initialises the following protected attributes:<br>• **NoOfRows** from parameter **R**<br>• **NoOfColumns** from parameter **C**<br>• **MoveOptionOfferPosition** to 0<br>Instantiates two new **Player** objects – Player 1 with the **Direction** parameter of 1 and Player 2 with the **Direction** parameter of -1 – and appends them both to the protected list attribute **Players**.<br>Assigns the element at position 0 of the **Players** list (Player 1) to the protected attribute **CurrentPlayer**.<br>Invokes the following methods:<br>• **CreateMoveOptions()** – to add the five standard move options to each player.<br>• **CreateMoveOptionOffer()** – to add the five standard move options to the move offer option list.<br>• **CreateBoard()** – to create a standard size board.<br>• **CreatePieces()** – to add the standard player and Mirza pieces to the board using the parameter **NoOfPieces**. |
| **Return values** | n/a | |
| **CalculatePieceCapturePoints** (private) | | |
| **Parameters** | **FinishSquare Reference** : Int | Uses the **GetPieceInSquare** method to get the piece at the **Board** location from the **FinishSquareReference** parameter.<br>If there is a piece at that location, the **PointsIfCaptured** attribute for that piece is returned. If there is no piece at that location the method returns 0. |
| **Return values** | Integer | |
| **CheckIfGameOver** (private) | | |
| **Parameters** | n/a | Iterates through the **Board** list checking each square for a piece.<br>If the square contains a piece, the method returns true if the square contains a Kotla, and the piece in the square is a Mirza and belongs to the opponent of the player that owns that square. Under this scenario, the player who owns the Mirza has just captured their opponent's Kotla. If this isn't the case, the method checks to confirm if the piece contains either a Player 1 or Player 2 Mirza setting the **Player1HasMirza** and **Player2HasMirza** attributes appropriately.<br>A negated logical AND of these two attributes is returned. If both players have lost their Mirza, the method returns true, otherwise it returns false. |
| **Return values** | Boolean | |

 

<table>
<tr><td colspan="3">CheckSquareInBounds (private)</td></tr>
<tr><td>Parameters</td><td>SquareReference : Int</td><td rowspan="3">Used as an error handling method to check if the SquareReference parameter is within the bounds of the playing board.<br><br>The method initialises two local variables, Row and Col, using DIV to split off the row and MOD to split off the column from the SquareReference parameter. The method then checks to confirm if Row is outside of the range of 1 to the attribute NoOfRows and Col is outside of the range of 1 to the attribute NoOfColumns and returns false.<br><br>If both are in range, the method returns true.</td></tr>
<tr><td>Return values</td><td>Boolean</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">CheckSquareIsValid (private)</td></tr>
<tr><td>Parameters</td><td>SquareReference : Int<br>StartSquare : Boolean</td><td rowspan="3">Used to test if the SquareReference parameter is a valid Square choice.<br><br>The StartSquare parameter is passed as true if this method is being used to check when the player is selecting the location of a piece to move from (a ‘move from’ check), otherwise it is passed as false when the method is being used to check when the player is selecting the location to move a piece to (a ‘move to’ check).<br><br>The method firstly uses the CheckSquareInBounds() method to confirm that the square reference is within the bounds of the board and returns false if it is not.<br><br>The method then gets the piece at the Board location from the SquareReference parameter. If there is no piece at that location and this is a ‘move from’ check, the method returns false because the player has selected a blank square. If the StartSquare parameter is true – this is a ‘move to’ check, the method instead returns true because the player has selected a blank square.<br><br>If there is a piece already at the location chosen by the player, the method checks to confirm if that piece belongs to the current player. If it does and this is a ‘move from’ check, the method returns true. If this is a ‘move to’ check, the method returns false because the player is trying to move a piece onto one of their own pieces.<br><br>If the piece does not belong to the current player and this is a ‘move from’ check, the method returns false because the player is trying to select an opponent piece to move. If this is a ‘move to’ check, the method returns true as the player is attempting to take an opponent piece.</td></tr>
<tr><td>Return values</td><td>Boolean</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">CreateBoard (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="3">Uses nested iteration using the NoOfRows and NoOfColumns attributes to populate the Board list.<br><br>Player 1’s Kotla is placed to the left of the middle of row 1 and Player 2’s Kotla is placed in the middle (or right of middle if there is an even number of columns) of the value stored in the NoOfRows attribute.<br><br>The remaining locations are filled with an empty Square object.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

 

| **CreateChowkidarMoveOption** (private) | | |
|---|---|---|
| **Parameters** | **Direction** : Int | Instantiates a new **MoveOption** for the chowkidar move. This method uses the **Direction** parameter to instantiate new **Move** objects – one for each valid move location for this move option.<br><br>The first new **Move** parameter is the number of rows to move from starting location to finishing location. The second new **Move** parameter is the number of columns to move from the starting location to finishing location. Both values are relative to the starting location.<br><br>A **Direction** of 1 moves down the board – for Player 1, and a **Direction** of -1 moves up the board – for Player 2. Each new **Move** object is added to the chowkidar **NewMoveOption** object which is then returned.<br><br>See pre-release document for a graphical representation of valid move positions (shown from the viewpoint of Player 2). |
| **Return values** | **NewMoveOption** : MoveOption | |

| **CreateCuirassierMoveOption** (private) | | |
|---|---|---|
| **Parameters** | **Direction** : Int | Instantiates a new **MoveOption** for the cuirassier move. This method uses the **Direction** parameter to instantiate new **Move** objects – one for each valid move location for this move option.<br><br>The first new **Move** parameter is the number of rows to move from starting location to finishing location. The second new **Move** parameter is the number of columns to move from the starting location to finishing location. Both values are relative to the starting location.<br><br>A **Direction** of 1 moves down the board – for Player 1, and a **Direction** of -1 moves up the board – for Player 2. Each new **Move** object is added to the cuirassier **NewMoveOption** object which is then returned.<br><br>See pre-release document for a graphical representation of valid move positions (shown from the viewpoint of Player 2). |
| **Return values** | **NewMoveOption** : MoveOption | |

| **CreateFaujdarMoveOption** (private) | | |
|---|---|---|
| **Parameters** | **Direction** : Int | Instantiates a new **MoveOption** for the faujdar move. This method uses the **Direction** parameter to instantiate new **Move** objects – one for each valid move location for this move option.<br><br>The first new **Move** parameter is the number of rows to move from starting location to finishing location. The second new **Move** parameter is the number of columns to move from the starting location to finishing location. Both values are relative to the starting location.<br><br>A **Direction** of 1 moves down the board – for Player 1, and a **Direction** of -1 moves up the board – for Player 2. Each new **Move** object is added to the faujdar **NewMoveOption** object which is then returned.<br><br>See pre-release document for a graphical representation of valid move positions (shown from the viewpoint of Player 2). |
| **Return values** | **NewMoveOption** : MoveOption | |

 

<table>
<tr><td colspan="3">CreateJazairMoveOption (private)</td></tr>
<tr><td>Parameters</td><td>Direction : Int</td><td rowspan="3">Instantiates a new MoveOption for the jazair move. This method uses the Direction parameter to instantiate new Move objects – one for each valid move location for this move option.<br><br>The first new Move parameter is the number of rows to move from starting location to finishing location. The second new Move parameter is the number of columns to move from the starting location to finishing location. Both values are relative to the starting location.<br><br>A Direction of 1 moves down the board – for Player 1, and a Direction of -1 moves up the board – for Player 2. Each new Move object is added to the jazier NewMoveOption object which is then returned.<br><br>See pre-release document for a graphical representation of valid move positions (shown from the viewpoint of Player 2).</td></tr>
<tr><td>Return values</td><td>NewMoveOption : MoveOption</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">CreateRyottMoveOption (private)</td></tr>
<tr><td>Parameters</td><td>Direction : Int</td><td rowspan="3">Instantiates a new MoveOption for the ryott move. This method uses the Direction parameter to instantiate new Move objects – one for each valid move location for this move option.<br><br>The first new Move parameter is the number of rows to move from starting location to finishing location. The second new Move parameter is the number of columns to move from the starting location to finishing location. Both values are relative to the starting location.<br><br>A Direction of 1 moves down the board – for Player 1, and a Direction of -1 moves up the board – for Player 2. Each new Move object is added to the ryott NewMoveOption object which is then returned.<br><br>See pre-release document for a graphical representation of valid move positions (shown from the viewpoint of Player 2).</td></tr>
<tr><td>Return values</td><td>NewMoveOption : MoveOption</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">CreateMoveOption (private)</td></tr>
<tr><td>Parameters</td><td>Name : String<br>Direction : Int</td><td rowspan="3">Uses selection on the Name parameter to select the associated Create*****MoveOption method and return the MoveOption from that method.</td></tr>
<tr><td>Return values</td><td>MoveOption</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">CreateMoveOptionOffer (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="3">Adds the five default MoveOptions to the MoveOptionOffer string list attribute.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

 

<table>
<tr><td colspan="3">CreateMoveOptions (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">Adds the five default MoveOptions to the MoveOptionQueue for each player.<br><br>This method calls the CreateMoveOption() method passing the move Name and Direction parameters for each of the five default move options, adding the return MoveOption to the MoveOptionQueue for Player 1 and Player 2 in the Players list.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">CreatePieces (private)</td></tr>
<tr><td>Parameters</td><td>NoOfPieces : Int</td><td rowspan="2">Places the default playing pieces and Mirza for each player onto the board.<br><br>The method uses the NoOfPieces parameter to place that many standard playing pieces onto the board – placing Player 1 pieces on row 2 and Player 2 pieces on the penultimate row. Pieces are given the parameters of a name, which player they belong to, their points value if captured and their symbol on the board. Player 1 pieces are given the symbol ‘!’. Player 2 pieces are given a single quote symbol using an escape character to correctly interpret it as a string.<br><br>The method also places the Player 1 and 2 Mirzas into their associated Kotlas by halving the NoOfColumns attribute to work out the middle position in a row. Both Mirzas are given a points value if captured of 5. Player 1 Mirza is given the symbol of ‘1’ and Player 2 Mirza is given the symbol of ‘2’.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">DisplayBoard (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">Iterates through the Board list to print it out onto the screen. The method works by using the following steps:<br><br>• Iterate through to the number of columns printing the column number and a space.<br>• Iterate through to the number of columns printing a short sequence of hyphens.<br>• Use nested iteration to print out the associated symbol for each square on the board preceded by a ‘|’. If there is a piece in the square the symbol for that piece is also printed, otherwise a blank space is printed.<br>• Print a final ‘|’ symbol at the end of each row.<br>• Iterate through to the number of columns printing a short sequence of hyphens followed by two blank lines.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">DisplayFinalResult (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">The winner of the game is the player with the highest score when this method is called. The method compares the score of both players using the GetScore() method.<br><br>If Player 1 has a higher score than Player 2 then the method uses the GetName() method to print the Player 1 name concatenated with ‘is the winner!’; otherwise it prints the Player 2 name concatenated with ‘is the winner!’ If the player scores match, ‘Draw!’ is printed to the screen.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
</table>

 

| **DisplayState** (private) | | |
|---|---|---|
| **Parameters** | n/a | Used as part of the main menu system in the **PlayGame()** method to display information about the **CurrentPlayer**.<br><br>The method first calls the **DisplayBoard()** method to print the board to the screen followed by the current move option offer for a player to choose if they want.<br><br>It then uses the **GetPlayerStateAsString()** method to display the score and move option queue for the current player followed by the current player name. |
| **Return values** | n/a | |
| **GetIndexOfSquare** (private) | | |
| **Parameters** | **SquareReference** : Int | Used to convert a **SquareReference** input to a list position in the **Board** list for the associated **Square**.<br><br>The method initialises two local variables, **Row** and **Col**, using DIV to split off the row and MOD to split off the column from the **SquareReference** parameter.<br><br>1 is subtracted from both variables to make them zero bound and then the **Row** is multiplied by the **NoOfColumns** attribute and added to the **Col** attribute and returned. |
| **Return values** | Integer | |
| **GetPointsForOccupancyByPlayer** (private) | | |
| **Parameters** | **CurrentPlayer** : Player | Used to calculate the total points for any squares occupied by the **CurrentPlayer**.<br><br>The method initialises an integer variable **ScoreAdjustment** to 0 then iterates through the **Board** list looking for squares which are occupied by the current player.<br><br>The **GetPointsForOccupancy** method is called on each square in the **Board** list which by default will return 0. The method is overridden by the **Kotla** class to return 5 if the **Kotla** belongs to current player and is occupied by the current player **Mirza** or a current player piece. It will return 1 if the **Kotla** occupied by a current player piece or **Mirza** belong to the opponent player.<br><br>Points are totaled up in the **ScoreAdjustment** variable as the iteration progresses.<br><br>This total is then returned. |
| **Return values** | **ScoreAdjustment** : Int | |
| **GetSquareReference** (private) | | |
| **Parameters** | **Description** : String | Used to get a square reference on the board from the user.<br><br>The method uses the **Description** parameter to concatenate an appropriate output to the user so that this method can be used for either a start or finish square reference.<br><br>Input from the user is casted without any error handling and stored in a local integer variable **SelectedSquare** and then returned. |
| **Return values** | **SelectedSquare** : Int | |

 

| **PlayGame** (public) | | |
|---|---|---|
| **Parameters** | n/a | This method is the main game playing loop which is held in the loop using the local Boolean variable **GameOver**.<br><br>The method firstly displays the current game state which includes the board and the current player queue. It then asks the current player to choose a move option (1–3) from their move option queue or select 9 if they would like to choose a move offer.<br><br>If the user selects option 9, the method calls **UseMoveOptionOffer()** to display the move offer submenu and then displays the current game state again. The method loops until the user selects a valid move option.<br><br>The method then asks the user to enter in a **StartSquareReference** containing the piece that they would like to move. Using the **GetSquareReference()** and **CheckSquareIsValid()** methods, it will repeatedly ask until the user gives a valid location.<br><br>The method then repeats this process but asking for a **FinishSquareReference** containing the location of where the player wants to move the piece to. The method then uses the **CheckPlayerMove()** method to confirm that the **StartStartReference** and **FinishSquareReference** are valid for the selected move choice.<br><br>If the move is legal, the method performs the following steps:<br><br>• Calculates any points if the move captures an opponent piece using the **CalculatePieceCapturePoints()** method and storing it in **PointsForPieceCapture**.<br>• Updates the player score based on the position of the move option used from the player queue using the **ChangeScore()** method.<br>• Updates the player queue to move the select **MoveOption** choice to the back of the queue using the **UpdateQueueAfterMove()** method.<br>• Calls the **UpdateBoard()** method to update the position of pieces based on the **StartSquareReference** and **FinishSquareReference**.<br>• Calls the **UpdatePlayerScore()** method to update the current player score with **PointsForPieceCapture.**<br>• Prints the updated score for the current player onto the screen.<br><br>This method does not deal with the case where the move is not legal, it simply just ignores the move and completes the player turn without informing the player.<br><br>The method then checks which player is currently playing and swaps to the opposing player. It then uses the **CheckIfGameOver()** to check if the current player has got their Mirza into the opponent Kotla or the opponent Mirza has been captured which stops the main game playing loop.<br><br>After the main game playing loop the method calls the **DisplayState()** method to print the final position of the playing board and then calls the **DisplayFinalResult()** method to confirm which player has won. |
| **Return values** | n/a | |

 

<table>
<tr><td colspan="3">UseMoveOptionOffer (private)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="3">Used to place the move from the MoveOptionOffer list into the current player move option queue.<br><br>The method asks the player for a position to place the current offer move from the MoveOptionOffer list and, without using any error handling, stores the result in the local integer variable ReplaceChoice. The method then uses the UpdateMoveOptionQueueWithOffer() method on the CurrentPlayer object to replace the user selected position move with the current offer move from the MoveOptionOffer list. The method then reduces the player score using the ChangeScore() method based on the position of the move option selected by the player to replace.<br><br>The method then updates MoveOptionOfferPosition variable with a random number from 0 to 4 to select a new move from the MoveOptionOffer list.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">UpdateBoard (private)</td></tr>
<tr><td>Parameters</td><td>StartSquareReference : Int<br>FinishSquareReference : Int</td><td rowspan="3">Performs the actual move of a piece from one location on the board to another.<br><br>The method uses the RemovePiece() method on the Board list index calculated from the StartSquareReference parameter. This piece is subsequently passed as a parameter to the SetPiece() method to be placed at the Board list index calculated from the FinishSquareReference parameter.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">UpdatePlayerScore (private)</td></tr>
<tr><td>Parameters</td><td>PointsForPieceCapture : Int</td><td rowspan="3">Calculates the change in player score for the move which the player has just made.<br><br>The method calls the GetPointsForOccupancyByPlayer() method on the current player to create a total points score for any Kotlas which are occupied by the player. This is then added to the PointsForPieceCapture parameter which contains the points for any opponent pieces captured in that move.<br><br>The combined total is then used to update the current player score using the ChangeScore() method.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="2"></td></tr>
</table>

 

## *Class: Piece*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | **T** : String<br>**B** : Player<br>**P** : Int<br>**S** : String | Initialises the following protected attributes:<br>• **TypeOfPiece** from parameter **T**<br>• **BelongsTo** from parameter **B**<br>• **PointsIfCaptured** from parameter **P**<br>• **Symbol** from parameter **S** |
| **Return values** | n/a | |
| **GetBelongsTo** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **BelongsTo**. |
| **Return values** | **BelongsTo** : Player | |
| **GetPointsIfCaptured** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **PointsIfCaptured**. |
| **Return values** | **PointsIfCaptured** : Int | |
| **GetSymbol** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **Symbol**. |
| **Return values** | **Symbol** : String | |
| **GetTypeOfPiece** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **TypeOfPiece**. |
| **Return values** | **TypeOfPiece** : String | |

## *Class: Square*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | n/a | Initialises the following protected attributes:<br>• **PieceInSquare** to null<br>• **BelongsTo** to null<br>• **Symbol** to ‘ ‘ |
| **Return values** | n/a | |
| | | |
| **ContainsKotla** (public) <<overrideable>> | | |
| **Parameters** | n/a | If the **Symbol** attribute is a ‘K’ or a ‘k’, this method returns true to confirm that there is a Kotla piece in this square, otherwise it returns false. |
| **Return values** | Boolean | |
| **GetBelongsTo** (public) << overrideable >> | | |
| **Parameters** | n/a | Returns the value of the protected attribute **BelongsTo**. |
| **Return values** | **BelongsTo** : Player | |
| **GetPieceInSquare** (public) << overrideable >> | | |
| **Parameters** | n/a | Returns the value of the protected attribute **PieceInSquare**. |
| **Return values** | **PieceInSquare** : Piece | |

 

| **GetPointsForOccupancy** (public) << overrideable >> | | |
|---|---|---|
| **Parameters** | **CurrentPlayer** : Player | Base class method for the **GetPointsForOccupancy()** method in the Kotla class to override.<br>If the method was not overridden, it would just return 0. |
| **Return values** | Integer | |
| **GetSymbol** (public) << overrideable >> | | |
| **Parameters** | n/a | Returns the value of the protected attribute **Symbol**. |
| **Return values** | **Symbol** : String | |
| **RemovePiece** (public) << overrideable >> | | |
| **Parameters** | n/a | Used for removing a piece from a square.<br>The method makes a temporary copy of the **Piece** in the attribute **PieceInSquare** in a local variable **PieceToReturn**, then sets the attribute to null to delete the piece in the square. It then returns the variable **PieceToReturn**. |
| **Return values** | **PieceToReturn** : Piece | |
| | | |
| **SetPiece** (public) << overrideable >> | | |
| **Parameters** | **P** : Piece | Assigns the **P** parameter to the protected attribute **PieceInSquare**. |
| **Return values** | n/a | |

## *Class: Kotla (inherits from Square)*

| **Identifier / Data** | | **Description** |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | **P** : Player<br>**S** : String | Initialises the following parent attributes:<br>• **BelongsTo** from parameter **P**<br>• **Symbol** from parameter **S** |
| **Return values** | n/a | |
| **GetPointsForOccupancy** (public) <<overrides>> | | |
| **Parameters** | **CurrentPlayer** : Player | Overrides the **GetPointsForOccupancy()** method from the base class to return the score points when a square is occupied.<br>The method checks first to see if there is a piece in the Kotla square. If there is not, the method returns zero points.<br>If there is a piece in the Kotla square, the method then checks to see if the Kotla square belongs to the same player as the **CurrentPlayer** passed in as a parameter. If they match and the piece in the Kotla is either a Mirza or a standard piece also owned by the **CurrentPlayer**, the method returns five points. If Kotla square belongs to the **CurrentPlayer**, but there is no Mirza or standard piece in it, the method returns zero points.<br>If the Kotla square belongs to the opponent player and the piece in it is either a Mirza or a standard piece owned by the **CurrentPlayer**, the method returns one point, otherwise it returns zero points. |
| **Return values** | Integer | |

 

## *Class: MoveOption*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | **N** : String | Initialises the following protected attributes:<br>• **Name** from parameter **N**<br>• **PossibleMoves** to an empty **Move** list |
| **Return values** | n/a | |
| **AddToPossibleMoves** (public) | | |
| **Parameters** | **M** : Move | Adds the **M** parameter to the protected **PossibleMoves** list. |
| **Return values** | n/a | |
| **CheckIfThereIsAMoveToSquare** (public) | | |
| **Parameters** | **StartSquareReference** : Int<br>**FinishSquareReference** : Int | Used to check if the start and finish locations supplied by the player are valid start and finish locations for this **MoveOption**.<br>The method initialises four local variables, **StartRow** and **StartColumn** together with **FinishRow** and **FinishColumn**. The method uses DIV to split off the **StartRow** and MOD to split off the **StartColumn** from the **StartSquareReference** parameter. It then uses the same techniques to split off the **FinishRow** and **FinishColumn** from the **FinishSquareReference** parameter.<br>The method then iterates through each **Move** in the **PossibleMoves** list checking if the **StartRow**, **StartColumn** and **FinishRow**, **FinishColumn** combination represent a valid move for one of the possible positions a piece could move to. |
| **Return values** | Boolean | |
| **GetName** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **Name**. |
| **Return values** | **Name** : String | |

## *Class: Move*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | **R** : Int<br>**C** : Int | Initialises the following protected attributes:<br>• **RowChange** from parameter **R**<br>• **ColumnChange** from parameter **C** |
| **Return values** | n/a | |
| **GetRowChange** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **RowChange**. |
| **Return values** | **RowChange** : Int | |
| **GetColumnChange** (public) | | |
| **Parameters** | n/a | Returns the value of the protected attribute **ColumnChange**. |
| **Return values** | **ColumnChange** : Int | |

 

## *Class: MoveOptionQueue*

*This class does not have a specific constructor and therefore uses the default constructor*

| Identifier / Data | | Description |
|---|---|---|
| <<constructor>> | | |
| **Parameters** | n/a | Initialises the **Queue** private attribute to an empty **MoveOption** list. |
| **Return values** | n/a | |
| **Add** (public) | | |
| **Parameters** | **NewMoveOption** : MoveOption | Adds the **NewMoveOption** parameter to the protected **Queue** list. |
| **Return values** | n/a | |
| **GetMoveOptionInPosition** (public) | | |
| **Parameters** | **Pos** : Int | Returns the **MoveOption** at the index **Pos** in the **Queue** list. |
| **Return values** | MoveOption | |
| **GetQueueAsString** (public) | | |
| **Parameters** | n/a | Initialises a local empty string variable **QueueAsString** and a local integer variable **Count** which it assigns 1.<br><br>The method then iterates through the **Queue** list concatenating the **Count** variable together with the name of each **Move** in the **Queue** (using the **GetName()** method), incrementing the **Count** variable on each loop.<br><br>The method then returns the **QueueAsString** variable. |
| **Return values** | **QueueAsString** : String | |
| **MoveItemToBack** (public) | | |
| **Parameters** | **Position** : Int | Used for moving a **MoveOption** to the back of the **Queue** list.<br><br>The method makes a temporary copy of the **MoveOption** at the index **Position** in the **Queue** list.<br><br>The method then uses the static method **RemoveAt()** on the **Queue** list to remove the **MoveOption** at the index **Position**.<br><br>It then appends the temporary copy of the **MoveOption** back into the **Queue** list which has the effect of placing it at the end of the queue. |
| **Return values** | n/a | |
| **Replace** (public) | | |
| **Parameters** | **Position** : Int<br>**NewMoveOption** : MoveOption | Assigns the **NewMoveOption** parameter into the **Queue** list at the index given in the **Position** parameter. |
| **Return values** | n/a | |

 

## *Class: Player*

<table>
<tr><th colspan="2">Identifier / Data</th><th>Description</th></tr>
<tr><td colspan="3"><<constructor>></td></tr>
<tr><td>Parameters</td><td>N : String<br>D : Int</td><td rowspan="2">Initialises the following protected attributes:<ul><li>Score to 100</li><li>Name from parameter N</li><li>Direction from parameter D</li></ul></td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">AddToMoveOptionQueue (public)</td></tr>
<tr><td>Parameters</td><td>NewMoveOption : MoveOption</td><td rowspan="2">Adds the NewMoveOption parameter to the private Queue attribute.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">ChangeScore (public)</td></tr>
<tr><td>Parameters</td><td>Amount : Int</td><td rowspan="2">Increments the protected attribute Score by the Amount parameter.</td></tr>
<tr><td>Return values</td><td>n/a</td></tr>
<tr><td colspan="3">CheckPlayerMove (public)</td></tr>
<tr><td>Parameters</td><td>Pos : Int<br>StartSquareReference : Int<br>FinishSquareReference : Int</td><td rowspan="3">Used to check if a move selected by a player is valid by using the CheckIfThereIsAMoveToSquare() method.<br>The method creates a temporary move object for the move selected from the player queue using the Pos parameter.<br>The method then passes the StartSquareReference and FinishSquareReference parameters to the CheckIfThereIsAMoveToSquare() method to confirm if the references represent a valid move within the selected move option.</td></tr>
<tr><td>Return values</td><td>Boolean</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">GetDirection (public)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">Returns the value of the protected attribute Direction.</td></tr>
<tr><td>Return values</td><td>Direction : Int</td></tr>
<tr><td colspan="3">GetName (public)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">Returns the value of the protected attribute Name.</td></tr>
<tr><td>Return values</td><td>Name : String</td></tr>
<tr><td colspan="3">GetPlayerStateAsString (public)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="3">Used to expose the GetQueueAsString() method in the MoveOptionQueue class to the Dastan class through the player.<br>The method returns a concatenation of the player score attribute and the player queue represented as a single string using the GetQueueAsString() method.</td></tr>
<tr><td>Return values</td><td>String</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="3">GetScore (public)</td></tr>
<tr><td>Parameters</td><td>n/a</td><td rowspan="2">Returns the value of the protected attribute Score.</td></tr>
<tr><td>Return values</td><td>Score : Int</td></tr>
</table>

 

| **SameAs** (public) | | |
|---|---|---|
| **Parameters** | **APlayer** : Player | Used to check if the **APlayer** parameter is the same as this player object.<br><br>The method first checks to confirm if an actual player object has been passed as a parameter and returns false if it is null.<br><br>If not, the method compares the name of the **APlayer** parameter with the name of this player object. If they match, the method returns true as this represents that they are the same player, otherwise it returns false. |
| **Return values** | Boolean | |
| **UpdateMoveOptionQueueWithOffer** (public) | | |
| **Parameters** | **Position** : Int<br>**NewMoveOption** : MoveOption | Used to expose the **Replace()** method in the **MoveOptionQueue** class to the **Dastan** class through the player.<br><br>The method calls the **Replace()** method on the player queue, passing the **Position** and **NewMoveOption** parameters. This will replace the move option as the index of **Position** with the **NewMoveOption** parameter. |
| **Return values** | n/a | |
| **UpdateQueueAfterMove** (public) | | |
| **Parameters** | **Position** : Int | Used to expose the **MoveItemToBack()** method in the **MoveOptionQueue** class to the **Dastan** class through the player.<br><br>The method calls the **MoveItemToBack()** method on the player queue passing the **Position** parameter minus one to make it zero bound. This will cause the move option at that index in the player queue to be moved to the back of the queue. |
| **Return values** | n/a | |

 