A)
I have created 2 string array in order to keep row and column values. I used a 2-dimensional array in order to create a table with several columns and rows. After that I have created a nested loop in order to create an 8x8 board. After the creation of the board I have put every piece at its place.
The difficult thing for me in the creation of the board was to put the numbers and the letters all around the board , and I looked on the internet and I use string functions(concatenation).

B)
I have initialized 2-D array and I use nested loops in order to put the square object in it's. While I put the square object it wants from me third parameter I used this keyword to point the chessboard place.
C)
1)Board

public void initialize()

-It places all the pieces in their own place.
-There is no parameter.
-It doesn't return anything (void).
-I created 2 different for loops , one for each player , and added all the pieces in the correct squares by using the 2-dimensional array that I have created while making the ChessBoard.

`public boolean isWhiteTurn()`

-After the move of the black player this value will return True and the move pass to the White player.
-There is no parameter.
-It returns a boolean value that will decide which player is gone to move.
-I created a simple method that would return either False or True each time a player make a move.

public boolean isFinished()

-It checks if either the game is end or not.
-There is no parameter.
-It returns a boolean value that will turn True if there is no piece left for one of the players.
-It counts each piece remaining for both of the players by checking every square with a nested for loop. If there is no white piece or if there is no black piece, that means that the game is over and the method will return True.

public void nextPlayer()

-This method alterns the turns. If White has played, the method will give the turn to the Black player.
-There is no parameter.
-It doesn't return anything (voide).
-If the value of whitePlaying is True, it will become false so it will be the Black player's turn.

public String toString()

-It fills the borders of the ChessBoard with the numbers and letters.
-There is no parameter.
-It returns a String value called representation that will fill the borders of the ChessBoard.
-I can’t explain how I implemented the method because copied this part of the code from stackoverflow and asked help from a friend.

2)Square
public boolean isAtLastRow(int color)

-It checks if a piece it at the last (opposite) Row.
- color: The color of the piece that is checked.
-It returns a boolean value that becomes True if the piece is at the last Row.
-I created 2 different if statements where at the beginning it controls the piece’s color. If its color is white and it is located in the 7. Row it will return true. If its color is black and it is located in the 0. Row it will return true. If both of them are not True it will return False.

public boolean isEmpty()

-It checks if the coordinate that we choose is empty or not.
-There is no parameter.
-It returns a boolean value that will return true if there is no piece in the square that we selected.
-If the value of piece is null in that square (if there is nothing) it will return True, otherwise it will return False.

public boolean isAtSameColumn(Square square)

-*It checks if the square object is on the same column as the given square object.*
-square: the object of the square to compare.
-It returns *true* if square objects is on the same column else it returns false.
-If the given square and our square objects are at the same column is checked. After it return a boolean value according to condition.

public boolean isAtSameRow(Square square)

-It c*hecks if the square object is on the same row as the given square object.*
-square: the object of the square to compare
-It returns *true* if square objects is on the same row else it returns false.
-If the given square and our square objects are at the same row is checked. After it return a boolean value according to condition.

public boolean isDiagonal(Square square)

It c*hecks if the square object is on the same diagonal as the given square object.*

-square: the object of the square to compare.

-It returns *true* if square objects is at the same diagonal else it returns false.

-If the given square and our square objects are at the same diagonal is checked. After it return a boolean value according to condition.

public boolean isNeighborColumn(Square square)

It checks if the square object is in the neighbour column as the given square object
-square: the object of the square to compare

-It returns *true* if square objects is in the neighbour column else it returns false.

-If the given square and our square objects are at the same column is checked. After it return a boolean value according to condition.

public void putQueen(int color)

-It creates a new queen.
-color: The color of the piece that has reached the last row.
-It doesn't return anything (void).
-It takes the piece that has reached the last row and it creates a new Queen object and it places the queen in the same coordinate and color.

public void eliminate()

-It eliminates the pawn that has reached the last row.
-There is no parameter.
-It doesn't return anything (voide).
-It changes the value of piece into null so it basically destroys the piece.

D)
When we call a method, the method is dynamically defined at run time, not at compile time.For example if we choose a pawn the piece will transform in a pawn , if we choose the king our piece will transform in the king. That's how the main class benefits from polymorphism.
I have defined the part class as abstract and also defined the canMove method inside the part as abstract. Since canMove that I have defined within the piece class is not the same for all pieces, this method is overrided in each part class.

E)
All pieces inherit the same move() method (except the pawn object)
public void move(String to)

-It makes the movement of the pieces possible.
-to: is the coordinate of the target movement.
-It doesn't return anything (void).

-It takes the parameter as an input in order to know the new coordinate. After that it deletes the piece in the previous coordinate and creates the same piece in the new coordinate. Finally it set the coordinate of the piece in the new coordinate.
(For the pawn the only difference for this method is that if the pawn is in the last row, the pawn will be replaced with a queen).

All the different pieces inherit the method canMove() but this time the method is different for each piece because there are different movement rules for different pieces. But the only difference is the implementation of the code.
public boolean canMove(String to)

-It checks if the new coordinate of the piece is a valid coordinate.
-to: is the coordinate of the target movement.
-It returns a boolean value that will become true if the move is a valid move.
-The implementation changes for every piece. But the common thing about them is that the isValid variable's value is False at the beginning.
For the Bishop :

Only diagonal moves can make the value of isValid True.
It checks if there is any other piece between the bishop and the target coordinate, if there is any other piece the value of isValid will remain False.
If there is no other piece between the target coordinate and the current coordinate, or there is a piece of the opposite color in the target coordinate the value of isValid will become True.
If there is another piece between the bishop and the target coordinate or there is a piece of the same color in the target coordinate the variable isValid will remain False.

For the King:
There are a few if conditions but in summary ;
The value of isValid will become True if we set the new coordinate at maximum 1 square distant and if the new coordinate is empty or there is a piece of the opposite color. Otherwise the value of isValid will remain False.

For the Knight:
It checks if one of the row/column has changed by 1 value and the other one by 2 values. If it is not like this the values of isValid will remain False. If the move is like described, it will check if there is a piece of the same color in the target coordinate. If there is another piece of the same color the value of isValid will remain False, otherwise will become True.

For the pawn:
Here we should check 2 different situations;
1)If the pawn is in its initial position:
It checks if the target coordinate is at maximum 2 value above/below (it depends by the color) of the initial value. If this condition is True and there is no other piece between the initial coordinate and the target coordinate the value of isValid will become True .

2)If the pawn is not in its initial position;
The same thing happens like the first situation but the only difference is that the maximum values difference between the target coordinate and the current coordinate must be 1.

For both of the situations there is another move that can be valid. If there is a piece of the opposite color in the diagonal front (the front will change depending on the color) at 1 square distance, and if we set that coordinate as the target coordinate the value of isValid will return True.

For the Queen:
Every direction can make the value of isValid True. It checks if there is any other piece between the queen and the target coordinate, if there is any other piece the value of isValid will remain False.
If there is no other piece between the target coordinate and the current coordinate, or there is a piece of the opposite color in the target coordinate, the value of isValid will become True.
If there is another piece between the queen and the target coordinate, or there is a piece of the same color in the target coordinate the variable isValid will remain False.

For the Rook:
Only moves in the same row or moves in the column are valid and can change the value of isValid into True.
It checks if there is any other piece between the rook and the target coordinate, if there is any other piece the value of isValid will remain False.
If there is no other piece between the target coordinate and the current coordinate, or there is a piece of the opposite color in the target coordinate, the value of isValid will become True.
If there is another piece between the rook and the target coordinate, or there is a piece of the same color in the target coordinate the variable isValid will remain False.