Back To Course Home

Grokking the Object Oriented Design Interview

0% completed

Search Course

Object-Oriented Design and UML
- Object-Oriented Basics
- OO Analysis and Design
- What is UML?
- Use Case Diagrams
- Class Diagram
- Sequence diagram
- Activity Diagrams

Object Oriented Design Case Studies
- Design a Library Management System
- Design a Parking Lot
- Design Amazon - Online Shopping System
- Design Stack Overflow
- Design a Movie Ticket Booking System
- Design an ATM
- Design an Airline Management System
- Design a Car Rental System
- Design LinkedIn
- Design Cricinfo
- Design Facebook - a social network
- Design Chess

Appendix
- Contact us
- Other courses

Course Certificate

Mark Course as Completed

# Design Chess

Let's design a system to play online chess.

**We'll cover the following**

- System Requirements
- Use case diagram
- Class diagram
- Activity diagrams
- Code

Chess is a two-player strategy board game played on a chessboard, which is a checkered gameboard with 64 squares arranged in an 8×8 grid. There are a few versions of game types that people play all over the world. In this design problem, we are going to focus on designing a two-player online chess game.

## System Requirements

We'll focus on the following set of requirements while designing the game of chess:

1. The system should support two online players to play a game of chess.
2. All rules of international chess will be followed.
3. Each player will be randomly assigned a side, black or white.
4. Both players will play their moves one after the other. The white side plays the first move.
5. Players can't cancel or roll back their moves.
6. The system should maintain a log of all moves by both players.
7. Each side will start with 8 pawns, 2 rooks, 2 bishops, 2 knights, 1 queen, and 1 king.
8. The game can finish either in a checkmate from one side, forfeit or stalemate (a draw), or resignation.

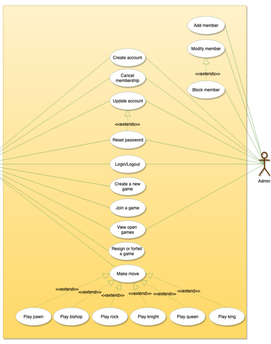
## Use case diagram

We have two actors in our system:

- **Player:** A registered account in the system, who will play the game. The player will play chess moves.
- **Admin:** To ban/modify players.

Here are the top use cases for chess:

- **Player moves a piece:** To make a valid move of any chess piece.
- **Resign or forfeit a game:** A player resigns from/forfeits the game.
- **Register new account/Cancel membership:** To add a new member or cancel an existing member.
- **Update game log:** To add a move to the game log.

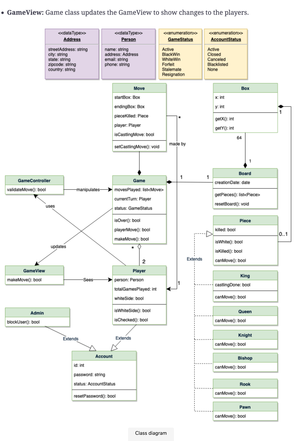



Use case diagram

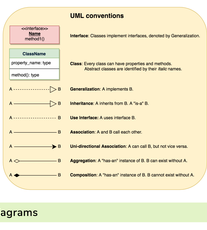
## Class diagram

Here are the main classes for chess:

- **Player:** Player class represents one of the participants playing the game. It keeps track of which side (black or white) the player is playing.
- **Account:** We'll have two types of accounts in the system: one will be a player, and the other will be an admin.
- **Game:** This class controls the flow of a game. It keeps track of all the game moves, which player has the current turn, and the final result of the game.
- **Box:** A box represents one block of the 8x8 grid and an optional piece.
- **Board:** Board is an 8x8 set of boxes containing all active chess pieces.
- **Piece:** The basic building block of the system, every piece will be placed on a box. This class contains the color the piece represents and the status of the piece (that is, if the piece is currently in play or not). This would be an abstract class and all game pieces will extend it.
- **Move:** Represents a game move, containing the starting and ending box. The Move class will also keep track of the player who made the move, if it is a castling move, or if the move resulted in the capture of a piece.
- **GameController:** Player class uses GameController to make moves.
- **GameView:** Game class updates the GameView to show changes to the players.

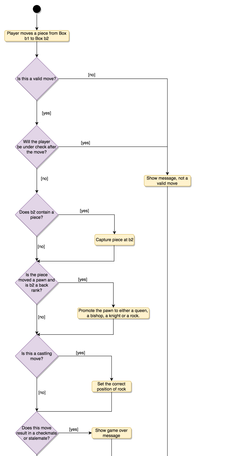



Class diagram




## Activity diagrams

**Make move:** Any Player can perform this activity. Here are the set of steps to make a move:



## Code

Here is the code for the top use cases.

**Enums, DataTypes, Constants:** Here are the required enums, data types, and constants:

Java | Python

```java
public enum GameStatus {
  ACTIVE, BLACK_WIN, WHITE_WIN, FORFEIT, STALEMATE, RESIGNATION
}

public enum AccountStatus {
  ACTIVE, CLOSED, CANCELED, BLACKLISTED, NONE
}

public class Address {
  private String streetAddress;
  private String city;
  private String state;
  private String zipCode;
  private String country;
}

public class Person {
  private String name;
  private Address address;
  private String email;
  private String phone;
}

```

**Box:** To encapsulate a cell on the chess board:

Java | Python

```java
public class Box {
  private Piece piece;
  private int x;
  private int y;

  public Box(int x, int y, Piece piece) {
    this.setPiece(piece);
    this.setX(x);
    this.setY(y);
  }

  public Piece getPiece() {
    return this.piece;
  }

  public void setPiece(Piece p) {
    this.piece = p;
  }

  public int getX() {
    return this.x;
  }

  public void setX(int x) {
    this.x = x;
  }

  public int getY() {
```

**Piece:** An abstract class to encapsulate common functionality of all chess pieces:

Java | Python

```java
public abstract class Piece {

  private boolean killed = false;
  private boolean white = false;

  public Piece(boolean white) {
    this.setWhite(white);
  }

  public boolean isWhite() {
    return this.white == true;
  }

  public void setWhite(boolean white) {
    this.white = white;
  }

  public boolean isKilled() {
    return this.killed == true;
  }

  public void setKilled(boolean killed) {
    this.killed = killed;
  }

  public abstract boolean canMove(Board board, Box start, Box end);
}
```

**King:** To encapsulate King as a chess piece:

Java | Python

```java
public class King extends Piece {
  private boolean castlingDone = false;

  public King(boolean white) {
    super(white);
  }

  public boolean isCastlingDone() {
    return this.castlingDone == true;
  }

  public void setCastlingDone(boolean castlingDone) {
    this.castlingDone = castlingDone;
  }

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // we can't move the piece to a box that has a piece of the same color
    if(end.getPiece().isWhite() == this.isWhite()) {
      return false;
    }

    int x = Math.abs(start.getX() - end.getX());
    int y = Math.abs(start.getY() - end.getY());
    if(x + y == 1) {
      // check if this move will not result in king being attacked, if so return true
      return true;
    }
```

**Knight:** To encapsulate Knight as a chess piece:

Java | Python

```java
public class Knight extends Piece {
  public Knight(boolean white) {
    super(white);
  }

  @Override
  public boolean canMove(Board board, Box start, Box end) {

    // we can't move the piece to a box that has a piece of the same color
    if(end.getPiece().isWhite() == this.isWhite()) {
      return false;
    }

    int x = Math.abs(start.getX() - end.getX());
    int y = Math.abs(start.getY() - end.getY());
    return x + y == 2;
  }
}


```

**Board:** To encapsulate a chess board:

Java | Python

```java
public class Board {
  Box[][] boxes;

  public Board() {
    this.resetBoard();
  }

  public Box getBox(int x, int y) {

    if (x < 0 || x > 7 || y < 0 || y > 7) {
      throw new Exception("Index out of bound");
    }

    return boxes[x][y];
  }

  public void resetBoard() {
    // initialize white pieces
    boxes[0][0] = new Box(0, 0, new Rook(true));
    boxes[0][1] = new Box(0, 1, new Knight(true));
    boxes[0][2] = new Box(0, 2, new Bishop(true));
    //...
    boxes[1][0] = new Box(1, 0, new Pawn(true));
    boxes[1][1] = new Box(1, 1, new Pawn(true));
    //...

    // initialize black pieces
    boxes[7][0] = new Box(7, 0, new Rook(false));
```

**Player:** To encapsulate a chess player:

Java | Python

```java
public class Player extends Account{
  private Person person;
  private boolean whiteSide = false;

  public Player(Person person, boolean whiteSide){
    this.person = person;
    this.whiteSide = whiteSide;
  }

  public boolean isWhiteSide() {
    return this.whiteSide == true;
  }
}

```

**Move:** To encapsulate a chess move:

Java | Python

```java
public class Move {
  private Player player;
  private Box start;
  private Box end;
  private Piece pieceMoved;
  private Piece pieceKilled;
  private boolean castlingMove = false;

  public Move(Player player, Box start, Box end){
    this.player = player;
    this.start = start;
    this.end = end;
    this.pieceMoved = start.getPiece();
  }

  public boolean isCastlingMove() {
    return this.castlingMove == true;
  }

  public void setCastlingMove(boolean castlingMove) {
    this.castlingMove = castlingMove;
  }
}

```

**Game:** To encapsulate a chess game:

Java | Python

```java
public class Game {
  private Player[] players;
  private Board board;
  private Player currentTurn;
  private GameStatus status;
  private List<Move> movesPlayed;

  private void initialize(Player p1, Player p2) {
    players[0] = p1;
    players[1] = p2;

    board.resetBoard();

    if(p1.isWhiteSide()) {
      this.currentTurn = p1;
    } else {
      this.currentTurn = p2;
    }

    movesPlayed.clear();
  }

  public boolean isEnd() {
    return this.getStatus() != GameStatus.ACTIVE;
  }

  public boolean getStatus() {
    return this.status;
```

Back
Design a Restaurant Management sys...
Next
Design an Online Stock Brokerage Sys...
Mark as Completed
Report an Issue
Ask a Question