棋牌类面向对象设计

文泰来 老师

课程大纲

- 棋牌类OOD题型
- 棋牌类OOD解题思路
- Tic Tac Toe
- Chinese chess
- Black jack
- Design pattern总结

棋牌类

棋牌类题型

- 棋类

- 象棋，国际象棋，围棋，军旗，跳棋，五子棋…

棋牌类题型

- 棋类

- 象棋，国际象棋，围棋，军旗，跳棋，五子棋 ...

- 类棋类

- Tic Tac Toe, 扫雷

棋牌类题型

- 棋类

- 象棋，国际象棋，围棋，军旗，跳棋，五子棋 …

- 类棋类

- Tic Tac Toe, 扫雷

- 牌类

- Black jack, 德州扑克, 斗地主, 狼人杀

棋牌类题型

- 频率：中高

棋牌类题型

- 频率：中高
- 难度：高

棋牌类题型

- 频率：中高
- 难度：高
- 题目比较多变，不同的棋牌，玩法不同

棋牌类解题思路

棋牌类解题思路

- 棋牌类的特点：跟Hotel reservation / Elevator / Vending Machine 有什么区别？

棋牌类解题思路

- 棋牌类的特点：

- 玩家

棋牌类解题思路

- 棋牌类的特点：

- 玩家
- 规则

棋牌类解题思路

- 棋牌类的特点：

- 玩家
- 规则
- 胜负

棋牌类解题思路

- 棋牌类的特点：

- 玩家
- 规则
- 胜负
- 积分

棋牌类解题思路

- 棋牌类的特点：

- 玩家
- 规则
- 胜负
- 积分

针对棋牌类的特点来做Clarification

棋牌类解题思路

- 棋牌类术语

棋牌类解题思路

- 棋牌类术语

Board

Suit

Hand

…

棋牌类解题思路

- 棋牌类术语

Board
Suit
Hand
…

针对棋牌类的术语，可以在Core Object的时候进行考虑

棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些状态（State）？

棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些状态（State）？

- Initialization (摆盘，洗牌…)

棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些状态（State）？

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)

棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些阶段?

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算)

棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些状态（State）？

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算) + Tie (流局)

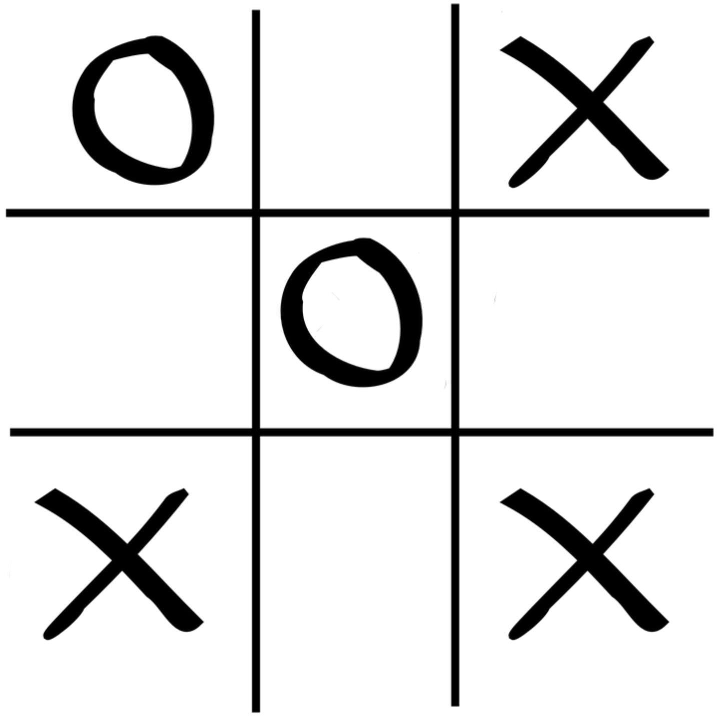
棋牌类解题思路

- 棋牌类的状态：一局棋牌，分为哪些状态（State）？

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算) + Tie (流局)

针对棋牌类的状态，来做Use cases

Tic Tac Toe



Tic Tac Toe

Can you design a Tic-Tac-Toe game, so that it can support two player play against each other?

Clarify

Clarify

- 玩家
- 规则
- 胜负
- 积分

Clarify

- 玩家

Clarify

- 玩家： 是否需要专门的Player类？

Clarify

- 玩家： Player之间有什么区别

Clarify

- 玩家： Player之间有什么区别

玩家A： X

玩家B： O

Clarify

- 玩家 ： Player之间有什么区别

玩家A： X
玩家B： O

```
currentPlayer = “X”;

changePlayer()
{
    if(currentPlayer.equals(“X”)) currentPlayer = “O”;
    else currentPlayer = “X”;
}
```

Clarify

- 扩展性不好？

玩家A： X
玩家B： O

```
currentPlayer = “X”;

changePlayer()
{
    if(currentPlayer.equals(“X”)) currentPlayer = “O”;
    else currentPlayer = “X”;
}
```

Clarify

- 什么时候需要Player类？（Player之间还会有什么区别？）

Clarify

- 什么时候需要Player类？（Player之间还会有什么区别？）

积分

Player
- Int score

Clarify

- 规则

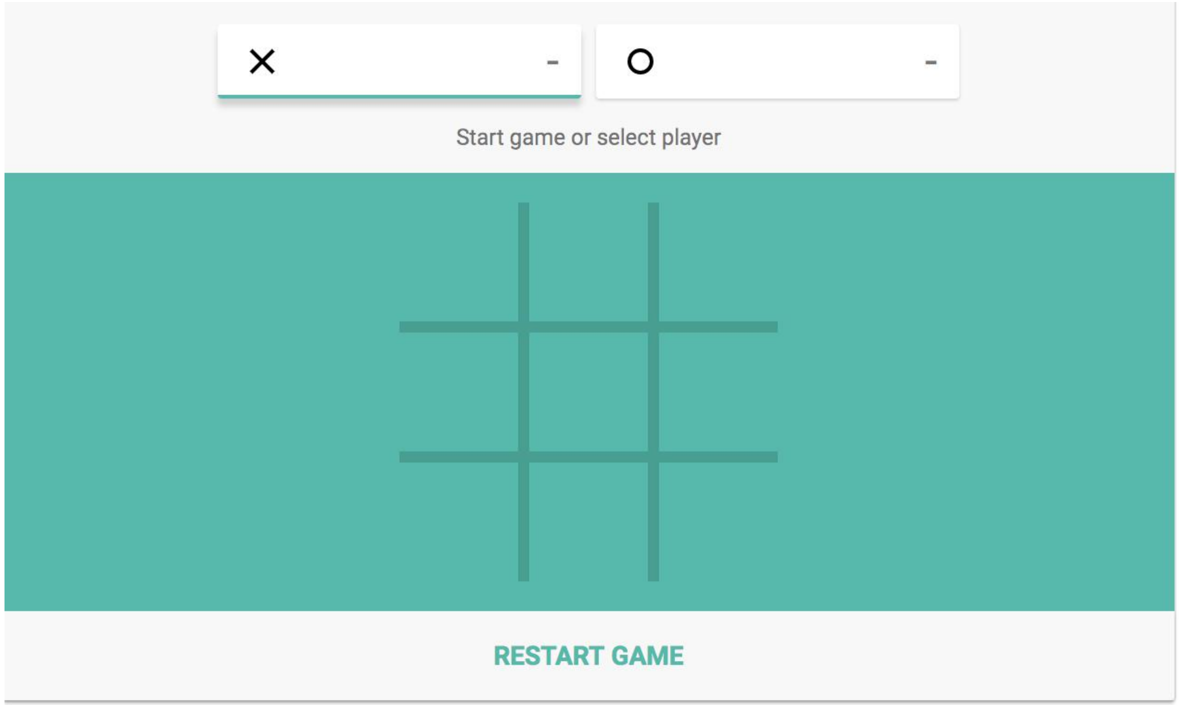
Clarify

- 规则

If you don’t understand how to play this game, this is the time to ask.

Clarify

- 规则

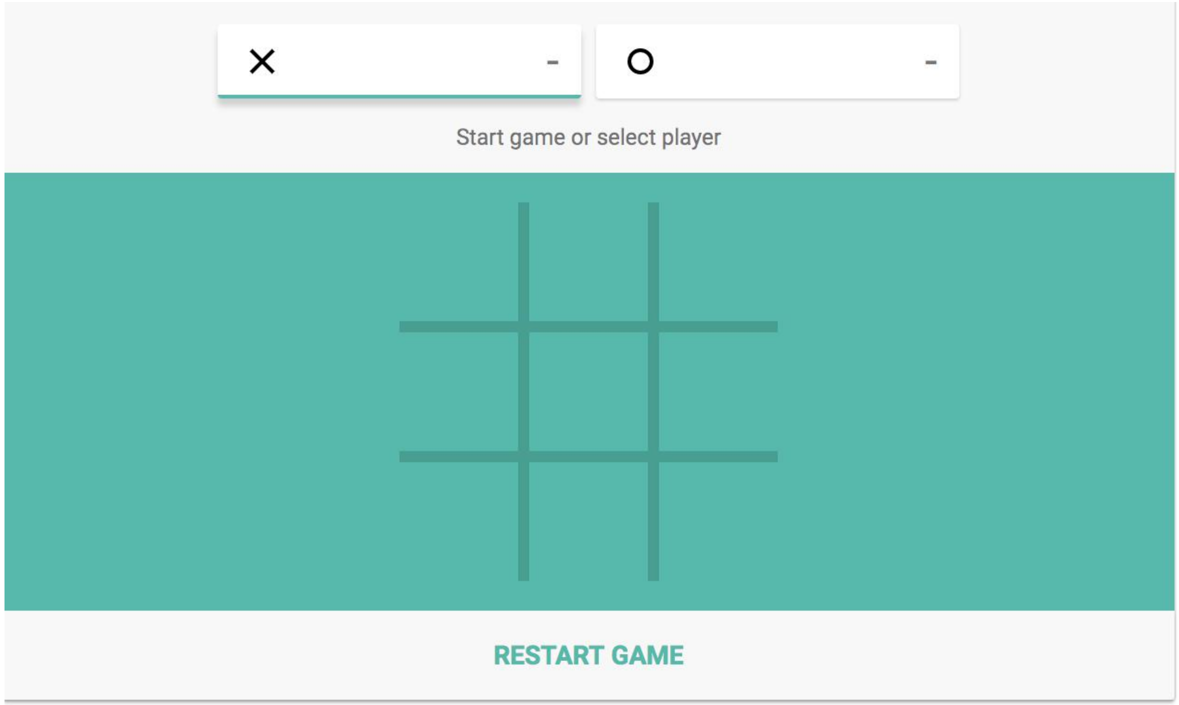


Who takes the first move?

- X?
- O?
- Take turns?
- Random?

Clarify

- 规则



What's the size of the board?

- 3 X 3?
- Larger?

Clarify

- 规则

对于本题：X always takes the first move

对于本题：3 X 3

Clarify

- 胜负

确认胜负规则

Clarify

- 积分

对于本题，不需要考虑积分

Core Object

Core Object

- 参考棋牌类的专业名词来考虑

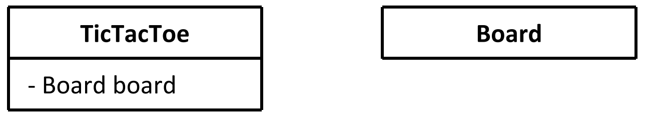
- Board
- Suit
- Hand
- Move
- …

Core Object

Core Object

TicTacToe
Board

Core Object



Use case

棋牌类游戏的三种状态

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算) + Tie (流局)

Use case

- Initialization (摆盘，入座，洗牌..)

Use case

- Initialization (摆盘，入座，洗牌..)

- Initialize the board

Use case

- Play (下棋，出牌…)

Use case

- Play (下棋，出牌…)

- Make move

Use case

- Play (下棋，出牌…)

- Make move
- Change player

Use case

- Win/Lose check (胜负结算) + Tie (流局)

Use case

- Win/Lose check (胜负结算) + Tie (流局)

- Check if X win / Check if O win / Check if board full

Classes

TicTacToe
- Board board

Board

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Initialize the board

- Clear the board an set everything to be empty

Classes

TicTacToe
- Board board

Board

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Classes

TicTacToe
- Board board

Board
- char[][] board

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Classes

TicTacToe
- Board board

Board
- char[][] board
+ void initializeBoard()

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Make move

- Check current move is for ‘X’ or ‘O’
- Place move at a pointed location

Classes

TicTacToe
- Board board - Char currentMove

Board
- char[][] board
+ void initializeBoard()

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Classes

TicTacToe
- Board board - Char currentMove

Board
- char[][] board
+ void initializeBoard() + void makeMove(int row, int col, char currentMove)

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Classes

TicTacToe
- Board board - Char currentMove
+ void makeMove(int row, int col)

Board
- char[][] board
+ void initializeBoard() + void makeMove(int row, int col, char currentMove)

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Change player

- Change current move from X to O or O to X

Classes

TicTacToe
- Board board - Char currentMove
+ void makeMove(int row, int col) - void changePlayer()

Board
- char[][] board
+ void initializeBoard() + void makeMove(int row, int col, char currentMove)

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Check Win / Lose / Tie

- Check if there is a winner
- Check if the board is full if there is no winner

Classes

TicTacToe
- Board board - Char currentMove
+ void makeMove(int row, int col) - void changePlayer()

Board
- char[][] board
+ void initializeBoard() + void makeMove(int row, int col, char currentMove) + boolean checkWin()

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

Classes

TicTacToe
- Board board - Char currentMove
+ void makeMove(int row, int col) - void changePlayer()

Board
- char[][] board
+ void initializeBoard() + void makeMove(int row, int col, char currentMove) + boolean checkWin() + boolean isBoardFull()

Use cases
Initialize board
Make move
Change player
Check for win / lose / tie

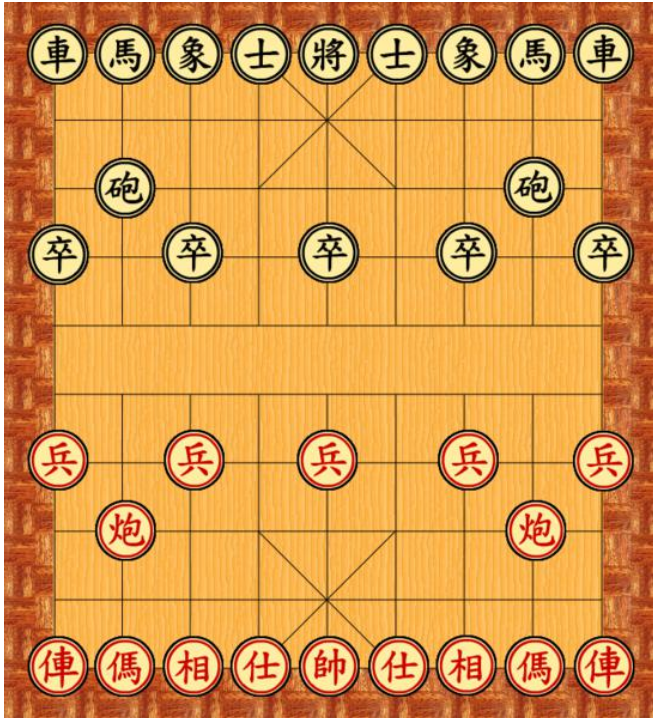
Classes

```
Simulator.java

makeMove(1,1);
```

```
TicTacToe.java

public void makeMove(int row, int col)
{
    board.makeMove(row, col, currentMove);
    if(board.checkWin())
    {
        print(currentMove + " win !");
    }
    else if(board.isBoardFull())
    {
        print("It's a tie");
    }
    changePlayer();
}
```

Chinese Chess

Chinese Chess



Chinese Chess

- 对于本题：腾讯象棋大厅

Clarify

- 玩家
- 规则
- 胜负
- 积分

Clarify

- 玩家

Clarify

- 玩家： 每位玩家有什么区别?

Clarify

- 玩家 ： 每位玩家有什么区别?

- 积分

Clarify

- 玩家 ： 每位玩家有什么区别?

- 积分
- 执红或执黑

Clarify

- 玩家 ： 每位玩家有什么区别？

- 积分
- 执红或执黑

对于本题：

- 每位玩家有自己的积分
- 每局游戏随机分配红黑

Clarify

- 规则

Clarify

- 规则

- 象棋走法的规则

Clarify

- 规则

- 象棋走法的规则
- 时间规则

Clarify

- 规则

- 象棋走法的规则
- 时间规则

对于本题：

常规象棋规则
无时间限制

Clarify

- 胜负

Clarify

- 胜负

- 如何判定平局？

Clarify

- 胜负

- 如何判定平局？

Solution 1: 如果下的步数超过一定数量，判定平局

Clarify

- 胜负

- 如何判定平局？

Solution 1: 如果下的步数超过一定数量，判定平局
Solution 2: 电脑判定，如果双方一直在走重复的步子，判定平局

Clarify

- 胜负

- 如何判定平局？

Solution 1: 如果下的步数超过一定数量，判定平局
Solution 2: 电脑判定，如果双方一直在走重复的步子，判定平局
Solution 3: 如果双方选手都要求平局，判断平局

Clarify

- 胜负

- 如何判定平局？

Solution 1: 如果下的步数超过一定数量，判定平局
Solution 2: 电脑判定，如果双方一直在走重复的步子，判定平局
Solution 3: 如果双方选手都要求平局，判断平局

对于本题：采用solution 1

Clarify

- 积分

Clarify

- 积分

对于本题：胜+1， 负-1，平局+0

Core Object

ChineseChess

Core Object

Player
ChineseChess

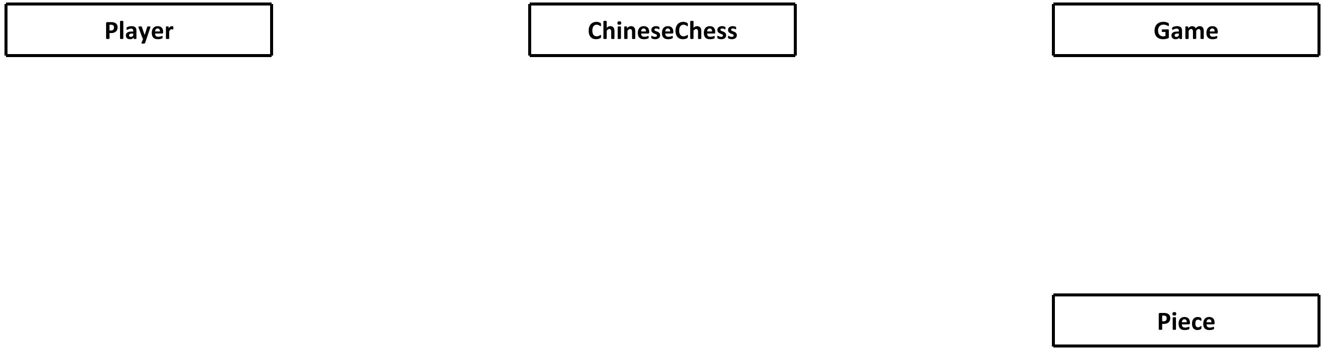
Core Object

Player

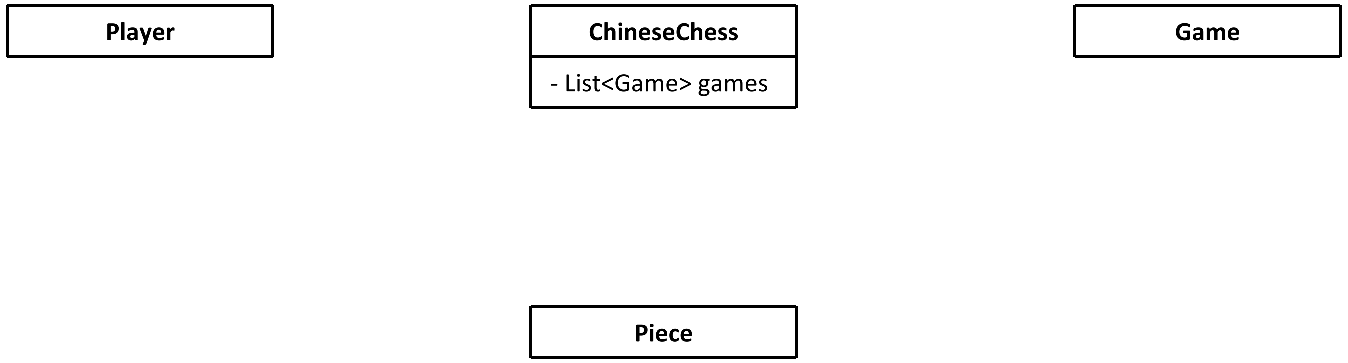
ChineseChess

Game

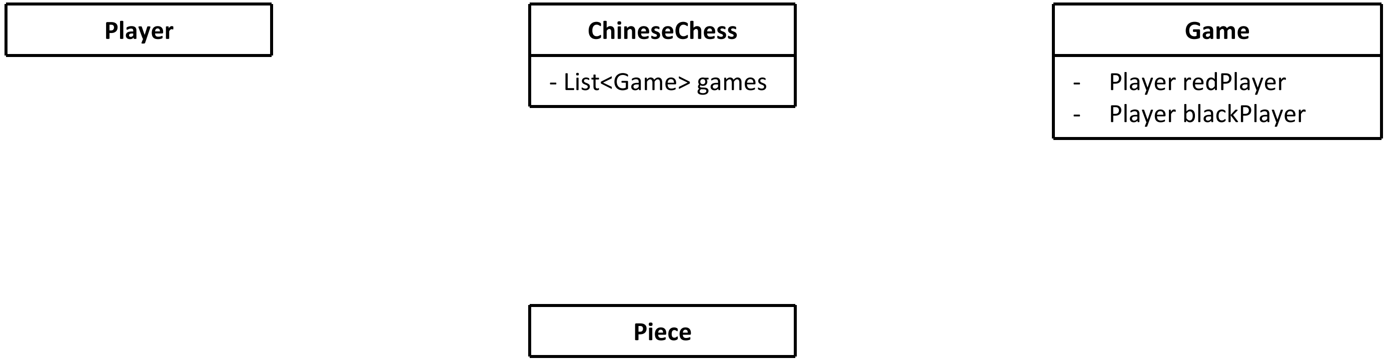
Core Object



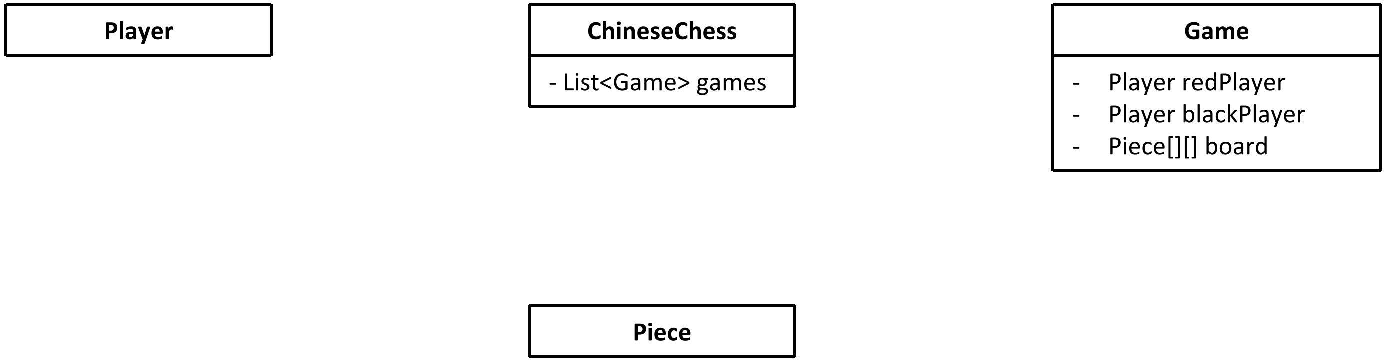
Core Object



Core Object


Player
ChineseChess
- List<Game> games
Game
- Player redPlayer
- Player blackPlayer
Piece

Core Object


Player
ChineseChess
- List<Game> games
Game
- Player redPlayer
- Player blackPlayer
- Piece[][] board
Piece

Use case

棋牌类游戏的三种状态

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算) + Tie / Draw (平局)

Use case

- Initialization (摆盘，洗牌…)

Use case

- Initialization (摆盘，洗牌…)

- Join game

Use case

- Initialization (摆盘，洗牌…)

- Join game
- Set up game

Use case

- Play (下棋，出牌…)

Use case

- Play (下棋，出牌…)

- Make move

Use case

- Play (下棋，出牌…)

- Make move
- Change player

Use case

- Win/Lose check (胜负结算) + Tie / Draw (平局)

Use case

- Win/Lose check (胜负结算) + Tie / Draw (平局)

- Check for win

Use case

- Win/Lose check (胜负结算) + Tie / Draw (平局)

- Check for win
- Increase steps

Use case

- Win/Lose check (胜负结算) + Tie / Draw (平局)

- Check for win
- Increase steps
- Calculate points

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Piece[][] board

Piece

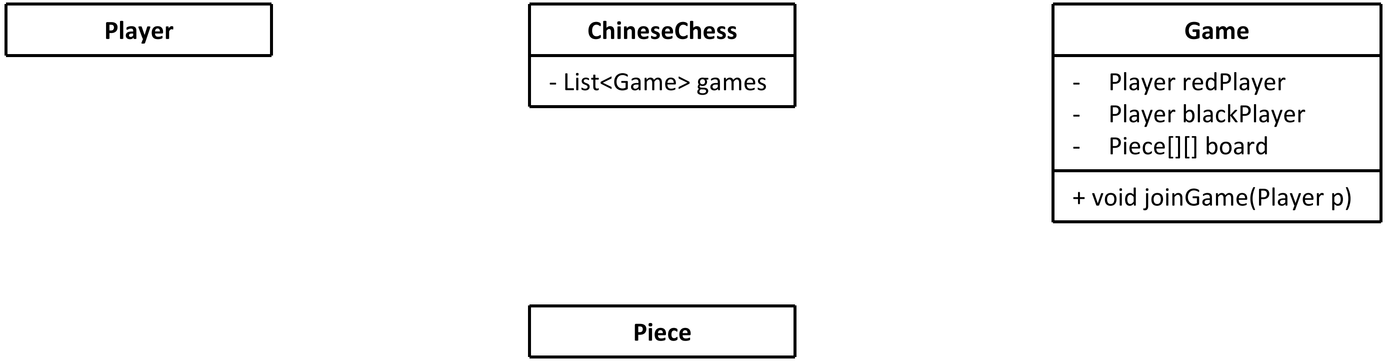
Use cases
Join game
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Join game

A player joins a game to play

Classes



Use cases

- Join table
- Set up game
- Make move
- Change player
- Check for win
- Increase steps
- Calculate points

Classes

- Set up game

Initialize the board with all pieces placed at the right place.

Classes

Player

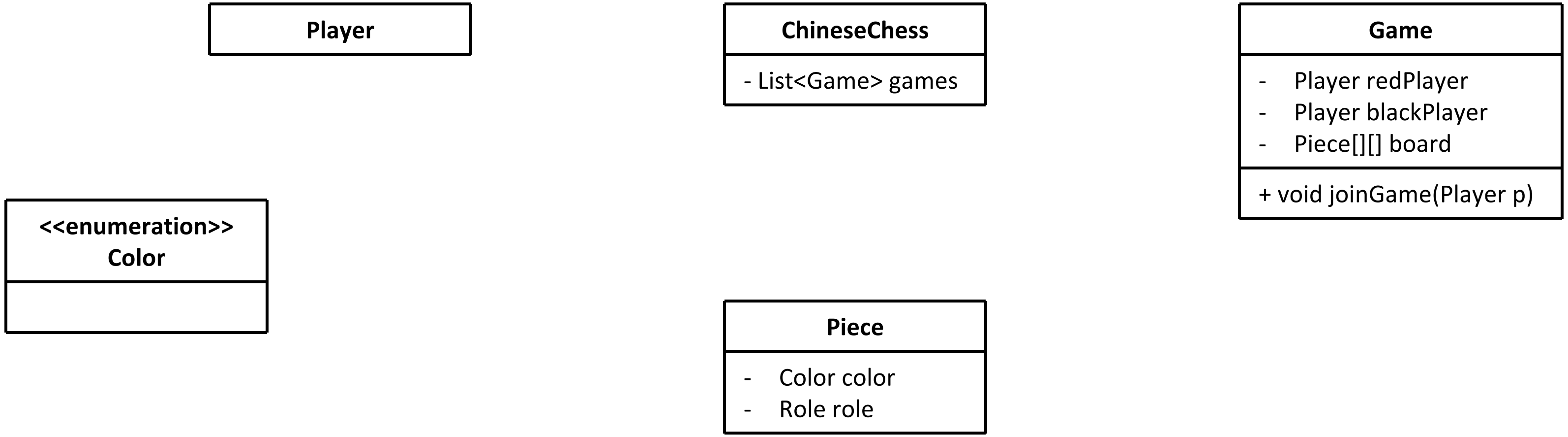
ChineseChess
- List<Game> games

Game
- Player redPlayer - Player blackPlayer - Piece[][] board
+ void joinGame(Player p)

Piece
- Color color - Role role

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes



Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

ChineseChess
- List<Game> games

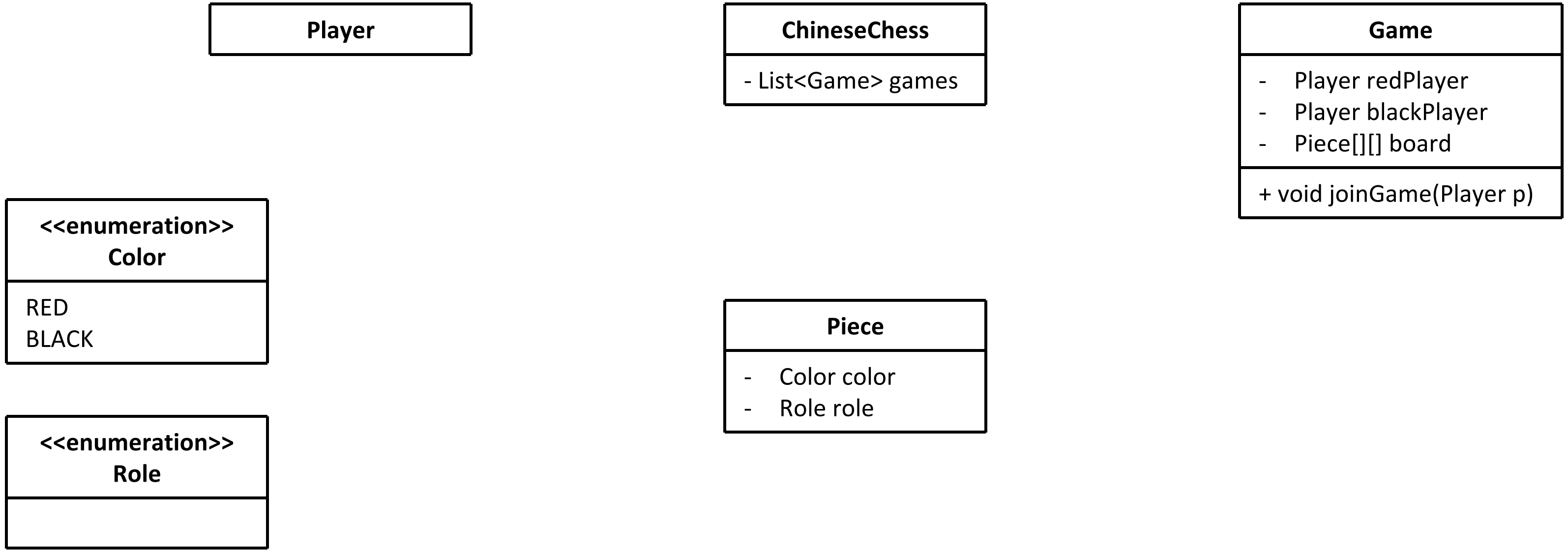
Game
- Player redPlayer - Player blackPlayer - Piece[][] board
+ void joinGame(Player p)

<<enumeration>> Color
RED BLACK

Piece
- Color color - Role role

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes



Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Piece[][] board

+ void joinGame(Player p)

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

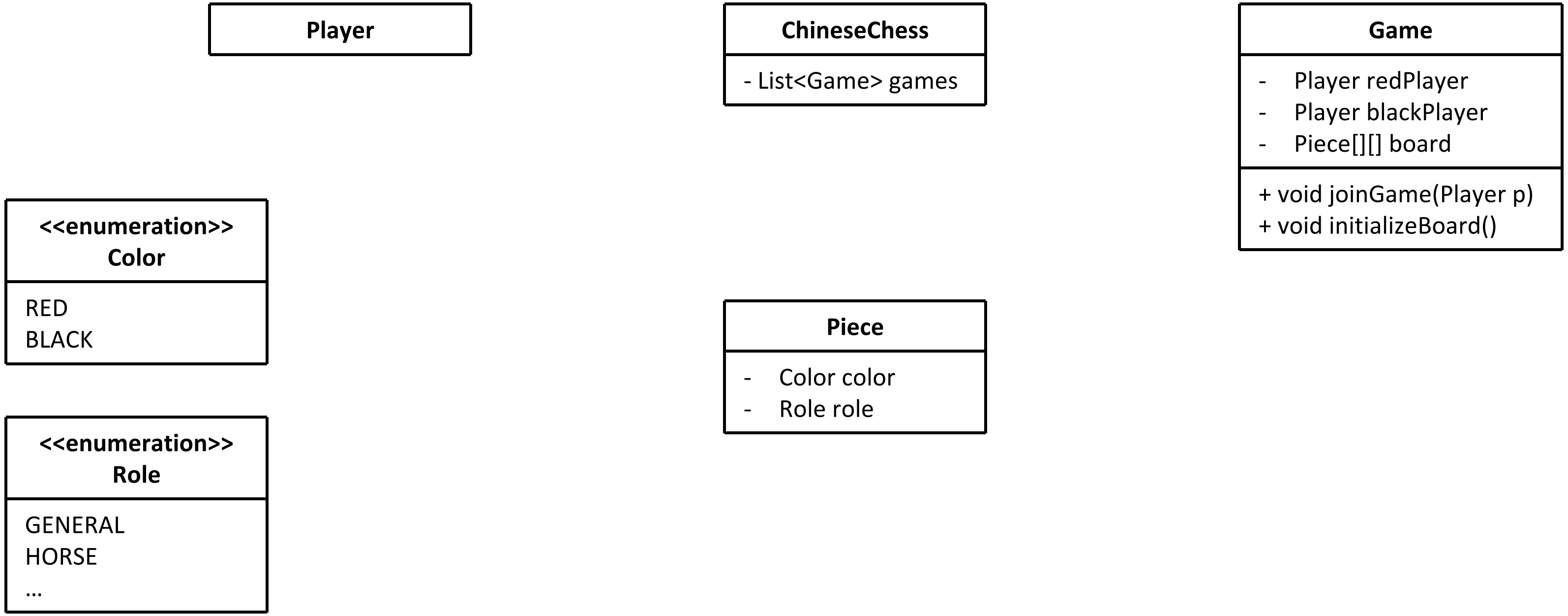
GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Enum: https://crunchify.com/why-and-for-what-should-i-use-enum-java-enum-examples/

Classes



Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Make move

- Determine which player should take the move
- Check if the move if valid, if yes, return true and make the move, if not return false

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board

+ void joinGame(Player p)
+ void initializeBoard()

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Change player

- Switch player

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Check for win

- Check if the current player wins

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()
- boolean ifCurrentPlayerWin()

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Increase steps

- Increase steps
- If reach a MAX step, call it a draw

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board
- Int steps

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()
- boolean ifCurrentPlayerWin()

<<enumeration>>
Color

RED
BLACK

<<enumeration>>
Role

GENERAL
HORSE
…

Piece

- Color color
- Role role

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board
- Int steps

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()
- boolean ifCurrentPlayerWin()
- Boolean gameDraw()

<<enumeration>>
Color

RED
BLACK

Piece

- Color color
- Role role

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

- Calculate points

If current player wins, reward current player and take one point off from other one.

Classes

Player

ChineseChess
- List<Game> games

Game
- Player redPlayer - Player blackPlayer - Player currentPlayer - Piece[][] board - Int steps
+ void joinGame(Player p) + void initializeBoard() + boolean move(Piece piece, int row, int col) - void changePlayer() - boolean ifCurrentPlayerWin() - boolean gameDraw() - Void rewardCurrentPlayer ()

<<enumeration>> Color
RED BLACK

<<enumeration>> Role
GENERAL HORSE …

Piece
- Color color - Role role

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

- Int points

ChineseChess

- List<Game> games

Game

```
-    Player redPlayer
-    Player blackPlayer
-    Player currentPlayer
-    Piece[][] board
-    Int steps
```

```
+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()
-    boolean ifCurrentPlayerWin()
-    boolean gameDraw()
-    Void rewardCurrentPlayer ()
```

<<enumeration>>
Color

RED
BLACK

Piece

```
-    Color color
-    Role role
```

<<enumeration>>
Role

GENERAL
HORSE
…

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Classes

Player

- Int points

+ void updatePointsBy(int diff)

ChineseChess

- List<Game> games

Game

- Player redPlayer
- Player blackPlayer
- Player currentPlayer
- Piece[][] board
- Int steps

+ void joinGame(Player p)
+ void initializeBoard()
+ boolean move(Piece piece, int row, int col)
- void changePlayer()
- boolean ifCurrentPlayerWin()
- boolean gameDraw()
- Void rewardCurrentPlayer ()

<<enumeration>>
Color

RED
BLACK

<<enumeration>>
Role

GENERAL
HORSE
…

Piece

- Color color
- Role role

Use cases
Join table
Set up game
Make move
Change player
Check for win
Increase steps
Calculate points

Blackjack

- Can you design blackjack?

All you need to know about Blackjack

5 Player
1 Dealer

Initialize 2 cards

Initialize bets

All you need to know about Blackjack

2 – 10 worth 2– 10

Jack/Queen/King = 10

A = 1 or 11

All you need to know about Blackjack

All you need to know about Blackjack

Player 1 call deal -> stop

Now he got 11 + 2 + 6 = 19

Or 1 + 2 + 6 = 9

All you need to know about Blackjack

Player 2 call deal

Now he got 10 + 5 + 8 = 23

Exceeds 21, he lost

Dealer took his chips

All you need to know about Blackjack

Dealer shows his cards

He has to keeping dealing until Reaches 17 or more

All you need to know about Blackjack

Dealer can stop or continue.

If dealer == player, dealer wins

Clarify

- 玩家
- 规则
- 胜负
- 积分

Clarify

- 玩家： How many player can we support in a table?

Clarify

- 玩家： Is there a fixed dealer or players take turn to become dealer?

Clarify

- 规则

Clarify

- 规则： What if we run out of cards?

Clarify

- 规则： Can dealer run out of bets?

Clarify

- 胜负

Clarify

- 积分

Clarify

- 积分：How many initial bets does a player have?

Clarify

- 对于本题：

- 无人数上限
- 每桌有Fixed dealer
- 牌永远够用
- Dealer的筹码永远够用
- 每个人有同样的初始筹码

Core object

- 牌类游戏比较固定的Core object framework

Core object

- 牌类游戏比较固定的Core object framework

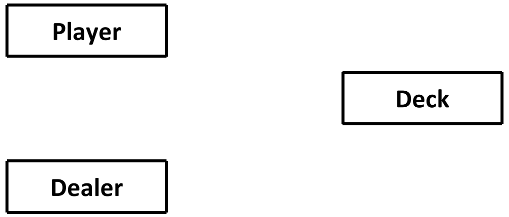
Core object

- 牌类游戏比较固定的Core object framework

Player
Deck

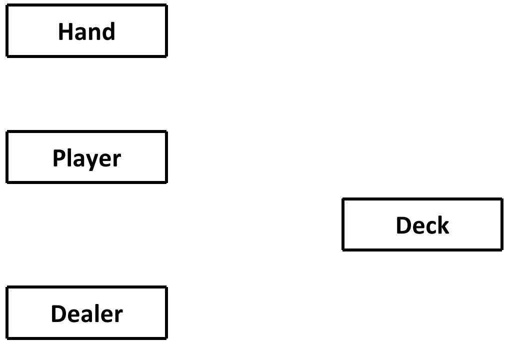
Core object

- 牌类游戏比较固定的Core object framework



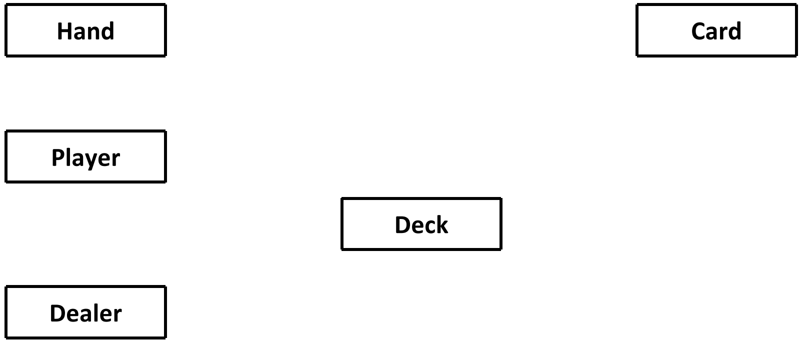
Core object

- 牌类游戏比较固定的Core object framework



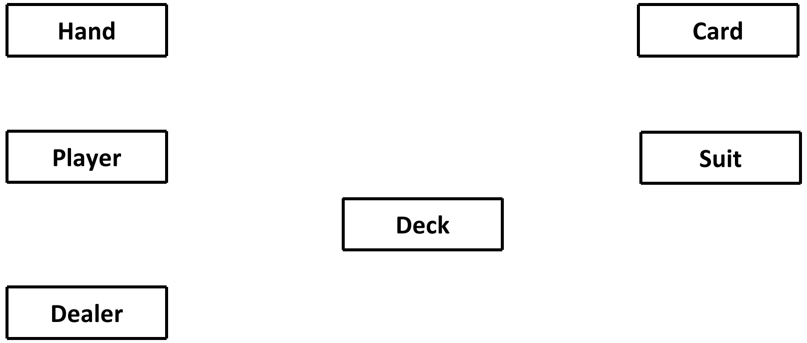
Core object

- 牌类游戏比较固定的Core object framework



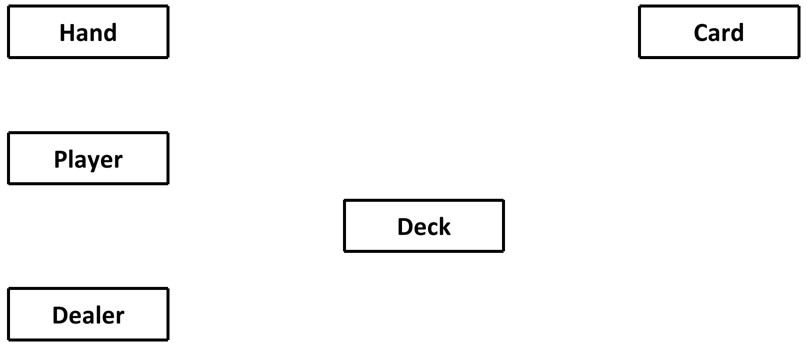
Core object

- 牌类游戏比较固定的Core object framework



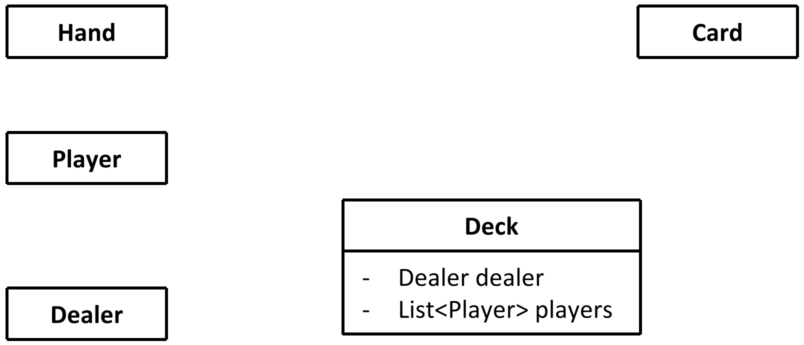
Core object

- 牌类游戏比较固定的Core object framework



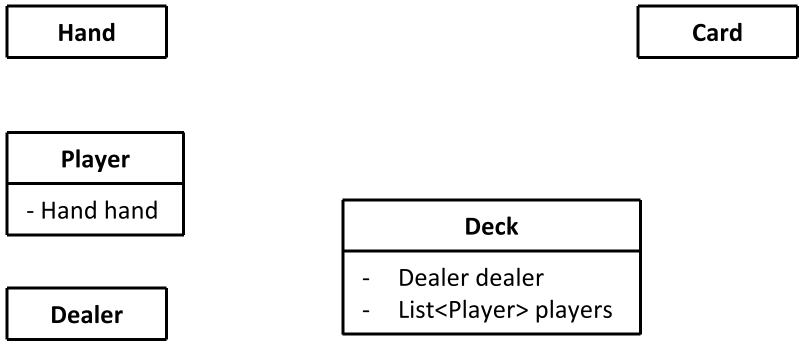
Core object

- 牌类游戏比较固定的Core object framework



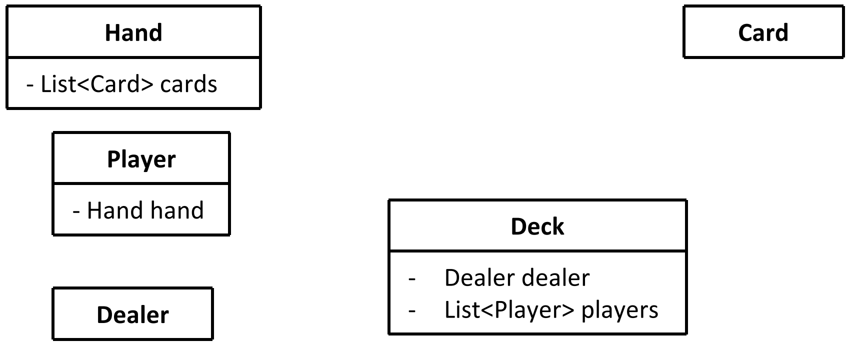
Core object

- 牌类游戏比较固定的Core object framework



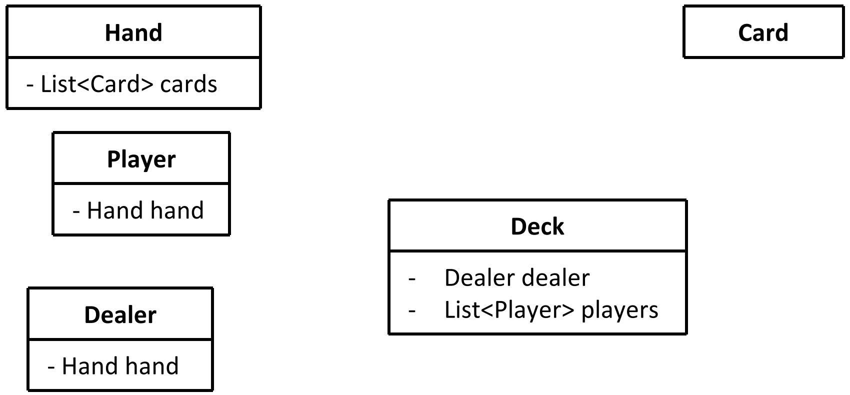
Core object

- 牌类游戏比较固定的Core object framework



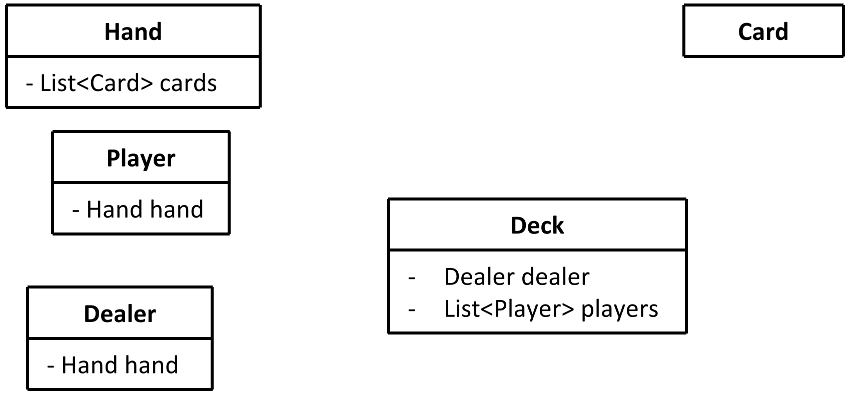
Core object

- 牌类游戏比较固定的Core object framework



Core object

- 牌类游戏比较固定的Core object framework



Use case

棋牌类游戏的三种状态

- Initialization (摆盘，洗牌…)
- Play (下棋，出牌…)
- Win/Lose check (胜负结算) + Tie / Draw (平局)

Use case

- Initialization (摆盘，洗牌…)

- Join table

Use case

- Initialization (摆盘，洗牌…)

- Join table
- Place bet

Use case

- Initialization (摆盘，洗牌…)

- Join table
- Place bet
- Get initial cards

Use case

- Play (下棋，出牌…)

- Deal

Use case

- Play (下棋，出牌…)

- Deal
- Increase bet

Use case

- Play (下棋，出牌…)

- Deal
- Increase bet
- Stop dealing

Use case

- Play (下棋，出牌…)

- Deal
- Increase bet
- Stop dealing

Use case

- Win/Lose check (胜负结算) + Tie / Draw (平局)

- Compare score
- Take/Lose bets

Classes

Deck
- Dealer dealer
- List<Player> players

Hand
- List<Card> cards

Player
- Hand hand

Dealer
- Hand hand

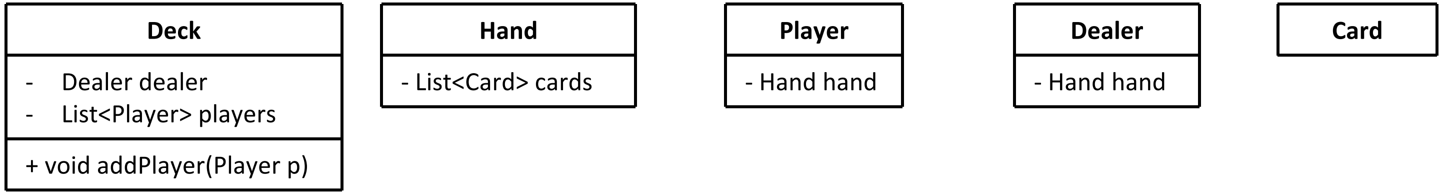
Card

Use cases
Join table
Place bet
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Join table

- Player join the deck

Classes



Deck

- Dealer dealer
- List<Player> players

+ void addPlayer(Player p)

Hand

- List<Card> cards

Player

- Hand hand

Dealer

- Hand hand

Card

Use cases
Join table
Place bet
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer
- List<Player> players
+ void addPlayer(Player p)

Hand
- List<Card> cards

Player
- Hand hand
+ void joinGame(Deck d)

Dealer
- Hand hand

Card

Use cases
Join table
Place bet
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Place bets

- Player place bets

Classes

Deck
- Dealer dealer - List<Player> players
+ void addPlayer(Player p)

Hand
- List<Card> cards

Player
- Hand hand - int totalBets
+ void joinGame(Deck d)

Dealer
- Hand hand

Card

Use cases
Join table
Place bet
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players
+ void addPlayer(Player p)

Hand
- List<Card> cards

Player
- Hand hand - int totalBets - Int currentBets
+ void joinGame(Deck d) + void placeBets(int amount)

Dealer
- Hand hand

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Get initial hands

- Each player and dealer get 2 initial cards

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle()

Hand
- List<Card> cards

Player
- Hand hand - int totalBets - Int bets
+ void joinGame(Deck d) + void placeBets(int amount)

Dealer
- Hand hand

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Shuffle cards

- http://massivealgorithms.blogspot.com/2015/07/shuffle-cards-cracking-coding-interview.html

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards(Player p)

Hand
- List<Card> cards

Player
- Hand hand - int totalBets - Int bets
+ void joinGame(Deck d) + void placeBets(int amount)

Dealer
- Hand hand

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck

- Dealer dealer
- List<Player> players
- List<Card> cards

+ void addPlayer(Player p)
+ void shuffle()
+ void dealInitialCards()

Hand

- List<Card> cards

Player

- Hand hand
- int totalBets
- Int bets

+ void joinGame(Deck d)
+ void placeBets(int amount)

Dealer

- Hand hand

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards()

Hand
- List<Card> cards

Player
- Hand hand - int totalBets - Int bets
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard (Card c)

Dealer
- Hand hand
+ void insertCard (Card c)

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck

- Dealer dealer
- List<Player> players
- List<Card> cards

+ void addPlayer(Player p)
+ void shuffle()
+ void dealInitialCards()

Player

- Hand hand
- int totalBets
- Int bets

+ void joinGame(Deck d)
+ void placeBets(int amount)
+ void insertCard (Card c)

Dealer

- Hand hand

+ void insertCard (Card c)

Card

Hand

- List<Card> cards

+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Deal

- Player decides whether they want to get another card

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard()

Player
- Hand hand - int totalBets - Int bets
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard (Card c)

Dealer
- Hand hand
+ void insertCard (Card c)

Card

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard()

Player
- Hand hand - Int bets - int totalBets - Deck d
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard (Card c)

Dealer
- Hand hand
+ void insertCard (Card c)

Card

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck

- Dealer dealer
- List<Player> players
- List<Card> cards

+ void addPlayer(Player p)
+ void shuffle()
+ void dealInitialCards()
+ Card dealNextCard()

Player

- Hand hand
- int totalBets
- Int bets
- Deck d

+ void joinGame(Deck d)
+ void placeBets(int amount)
+ void insertCard(Card c)
+ void dealNextCard()

Dealer

- Hand hand

+ void insertCard (Card c)

Card

Hand

- List<Card> cards

+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard()

Player
- Hand hand - int totalBets - Int bets - Deck d
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard()

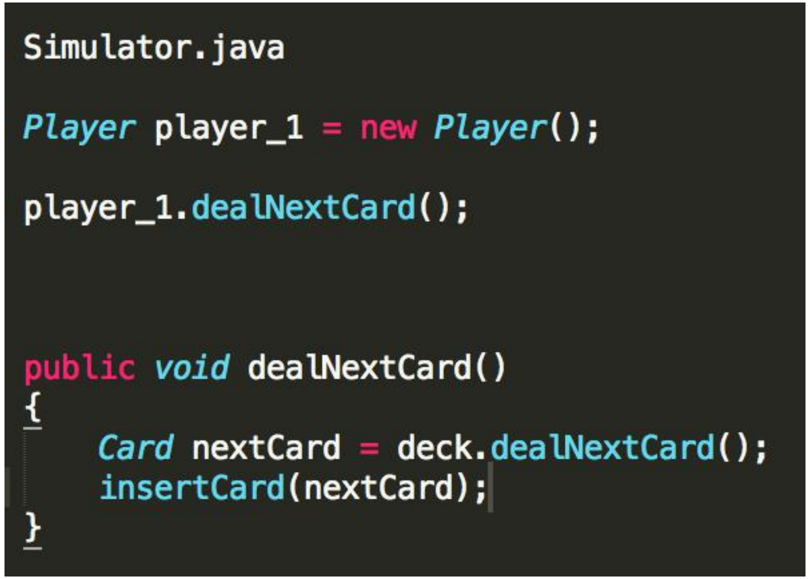
Hand
- List<Card> cards
+ void insertCard(Card c)

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard()

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes



```
Simulator.java

Player player_1 = new Player();

player_1.dealNextCard();



public void dealNextCard()
{
    Card nextCard = deck.dealNextCard();
    insertCard(nextCard);
}
```

Stop dealing

- A player calls stop and not get any new cards

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard()

Player
- Hand hand - int totalBets - Int bets - Deck d
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing()

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard()

Card

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing()

Hand
- List<Card> cards
+ void insertCard(Card c)

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard()

Card

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Compare results

- Player compare results with Dealer

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing()

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard()

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing()

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard()

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing()

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard() + boolean largerThan(Player p)

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Win/Lose bets

- Update player's bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing() + void updateBets(int amount)

Dealer
- Hand hand - Deck d
+ void insertCard (Card c) + void dealNextCard() + boolean largerThan(Player p)

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing() + void updateBets(int amount)

Dealer
- Hand hand - Deck d - Int bets
+ void insertCard (Card c) + void dealNextCard() + boolean largerThan(Player p)

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void stopDealing() + void updateBets(int amount)

Dealer
- Hand hand - Deck d - Int bets
+ void insertCard (Card c) + void dealNextCard() + boolean largerThan(Player p) + void updateBets(int amount)

Card
- Int value

Hand
- List<Card> cards
+ void insertCard(Card c)

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes

Deck
- Dealer dealer - List<Player> players - List<Card> cards
+ void addPlayer(Player p) + void shuffle() + void dealInitialCards() + Card dealNextCard() + void compareResults()

Player
- Hand hand - int totalBets - Int bets - Deck d - Boolean stopDealing
+ void joinGame(Deck d) + void placeBets(int amount) + void insertCard(Card c) + void dealNextCard() + void updateBets(int amount) + void stopDealing() + int getCurrentBets()

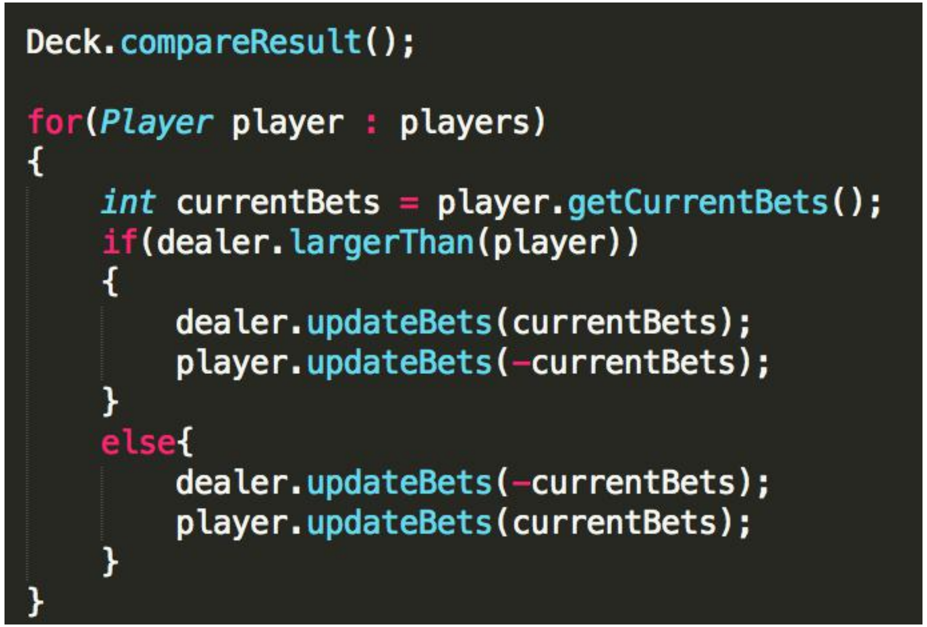
Hand
- List<Card> cards
+ void insertCard(Card c)

Dealer
- Hand hand - Deck d - Int bets
+ void insertCard (Card c) + void dealNextCard() + booleanlargerThan(Player p) + void updateBets(int amount)

Card
- Int value

Use cases
Join table
Place bets
Get initial cards
Deal
Stop dealing
Compare scores
Take/Lose bets

Classes



```
Deck.compareResult();

for(Player player : players)
{
    int currentBets = player.getCurrentBets();
    if(dealer.largerThan(player))
    {
        dealer.updateBets(currentBets);
        player.updateBets(-currentBets);
    }
    else{
        dealer.updateBets(-currentBets);
        player.updateBets(currentBets);
    }
}
```

棋牌类总结

- Clarify : 玩家，规则，胜负，积分

棋牌类总结

- Clarify : 玩家，规则，胜负，积分
- Core object: Hand, Board, Deck/Table, Suit, …

棋牌类总结

- Clarify : 玩家，规则，胜负，积分
- Core object: Hand, Board, Deck/Table, Suit, …
- Use cases: Initialization / Play / Checkout

棋牌类总结

- Clarify : 玩家，规则，胜负，积分
- Core object: Hand, Board, Deck/Table, Suit, …
- Use cases: Initialization / Play / Checkout

- 对于牌类，需要从Player的角度出发

Exception

- https://www.geeksforgeeks.org/exceptions-in-java/

Design pattern 总结

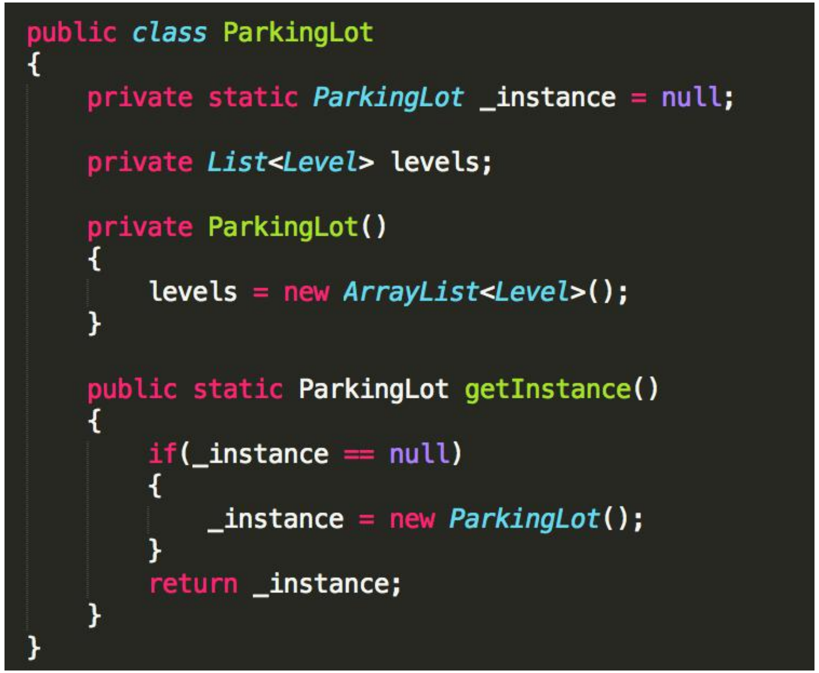
- Singleton
- Strategy
- Adapter
- State
- Decorator
- Factory

Singleton

- 常见的Design pattern
- 三种常见写法

Design pattern

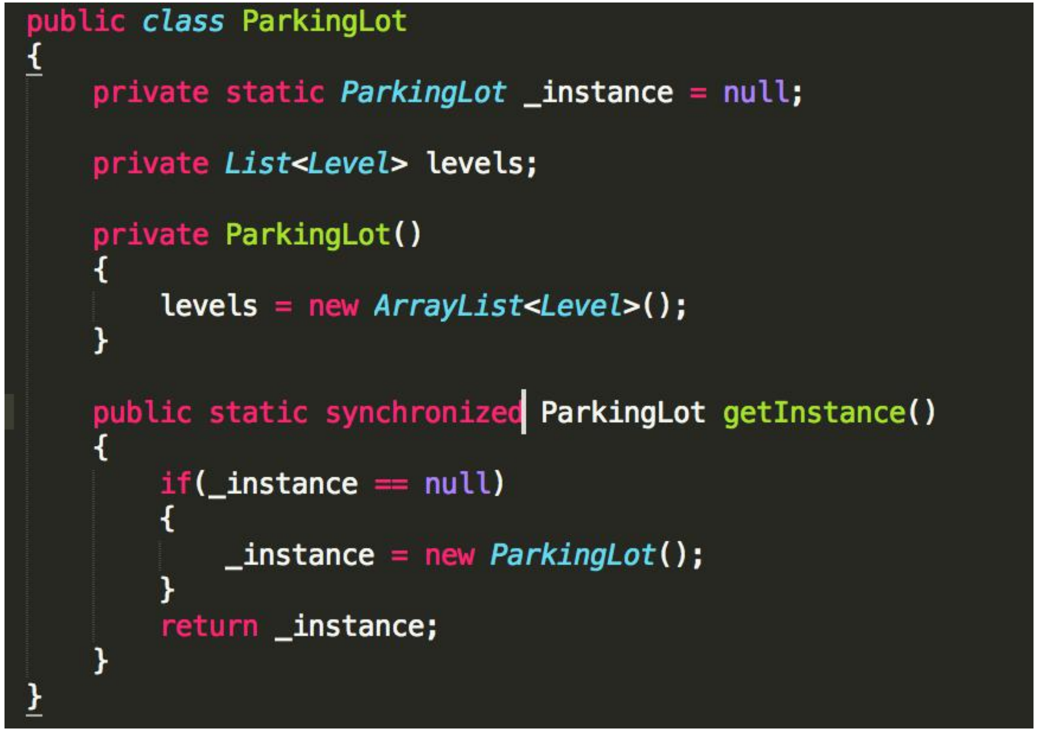
- Singleton – 基本式



```
public class ParkingLot
{
    private static ParkingLot _instance = null;

    private List<Level> levels;

    private ParkingLot()
    {
        levels = new ArrayList<Level>();
    }

    public static ParkingLot getInstance()
    {
        if(_instance == null)
        {
            _instance = new ParkingLot();
        }
        return _instance;
    }
}
```

Design pattern

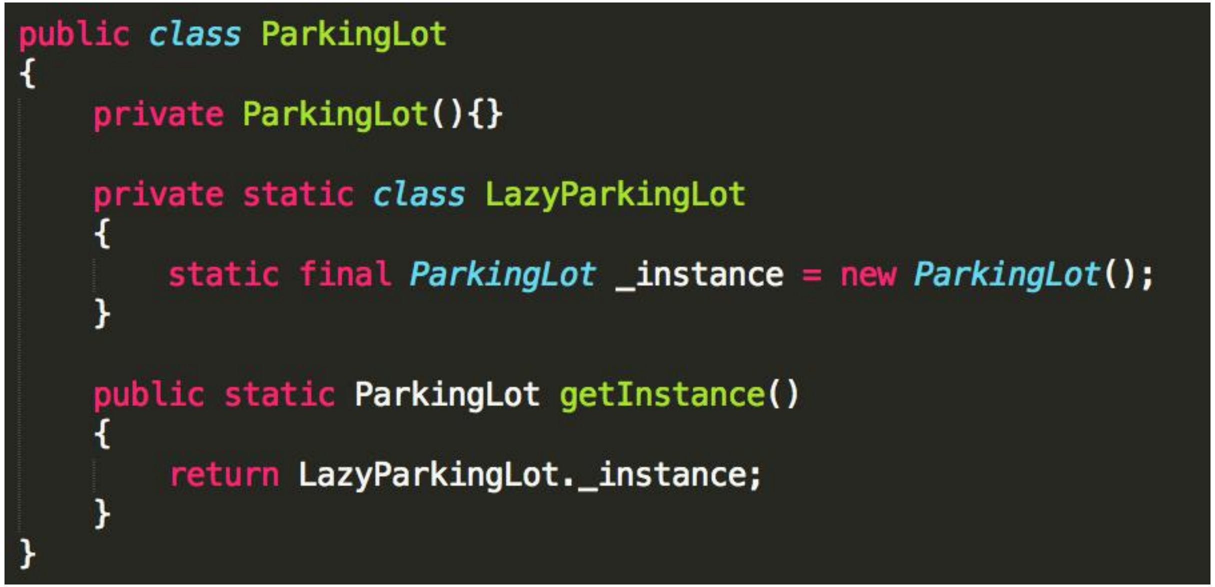
- Singleton – 线程安全式



```
public class ParkingLot
{
    private static ParkingLot _instance = null;

    private List<Level> levels;

    private ParkingLot()
    {
        levels = new ArrayList<Level>();
    }

    public static synchronized ParkingLot getInstance()
    {
        if(_instance == null)
        {
            _instance = new ParkingLot();
        }
        return _instance;
    }
}
```

Design pattern

- Singleton – 静态内部类式



```
public class ParkingLot
{
    private ParkingLot(){}

    private static class LazyParkingLot
    {
        static final ParkingLot _instance = new ParkingLot();
    }

    public static ParkingLot getInstance()
    {
        return LazyParkingLot._instance;
    }
}
```

Singleton

- 用途：

考虑你设计的东西，是否应该只有一个实例

- ElevatorSystem vs. Elevator

Singleton

- 用途：

考虑你设计的东西，是否应该只有一个实例

- ElevatorSystem vs. Elevator
- 象棋大厅 vs. 象棋 / Deck / Table

Singleton

- 用途：

考虑你设计的东西，是否应该只有一个实例

- ElevatorSystem vs. Elevator
- 象棋大厅 vs. 象棋 / Deck / Table
- Kindle 内部的 ReaderFactory

Singleton

面试中：

不需要一上来就考虑Singleton.

做完class diagram之后：

- So I was thinking maybe we can apply singleton pattern to this ReaderFactory as well, because…
- Do you think there should be only one instance of the Elevator System?

State

- 出现频率不高
- 特别适合于特殊类型的题目

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State: OPEN v.s. CLOSE

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State: OPEN v.s. CLOSE

24Hr Parking Lot?

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State: OPEN v.s. CLOSE

Park vehicle
Get available counts
Free spot

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State: OPEN v.s. CLOSE

Park vehicle
Get available counts
Free spot

以上use case，的确受Open/Close的影响

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. Management类型 -> Parking Lot

State: OPEN v.s. CLOSE

Park vehicle
Get available counts
Free spot

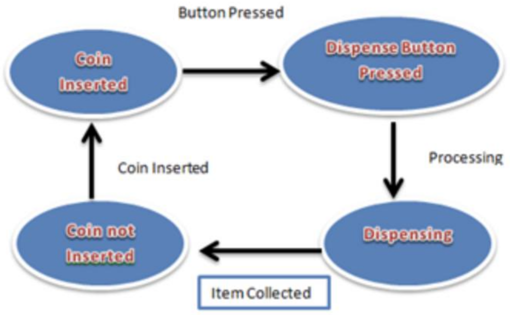
以上use case，的确受Open/Close的影响
但是以上的use case，并不会导致State的转换

State

- 出现频率不高
- 特别适合于特殊类型的题目

e.g. 实物类 -> Vending Machine

http://ydtech.blogspot.com/2010/06/state-design-pattern-by-example.html



State

State Pattern思考示例

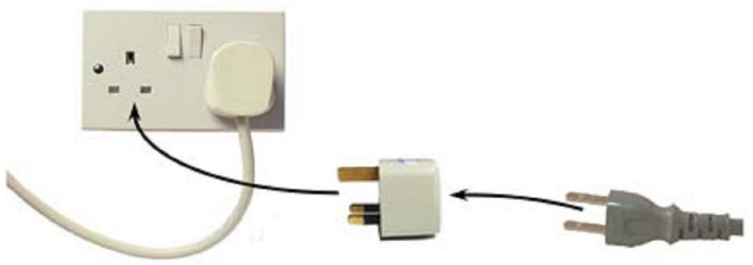
1. 有哪些State?
2. 有哪些function会受到上诉State的影响
3. 写State class以及所有子类
4. 在主体（vending machine）加上必要的函数和变量

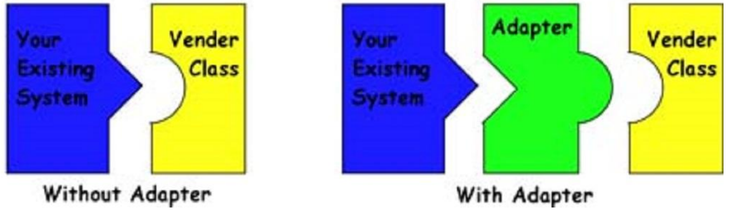
Adapter

- 面试中频率低
- 现实Coding中很实用

Adapter

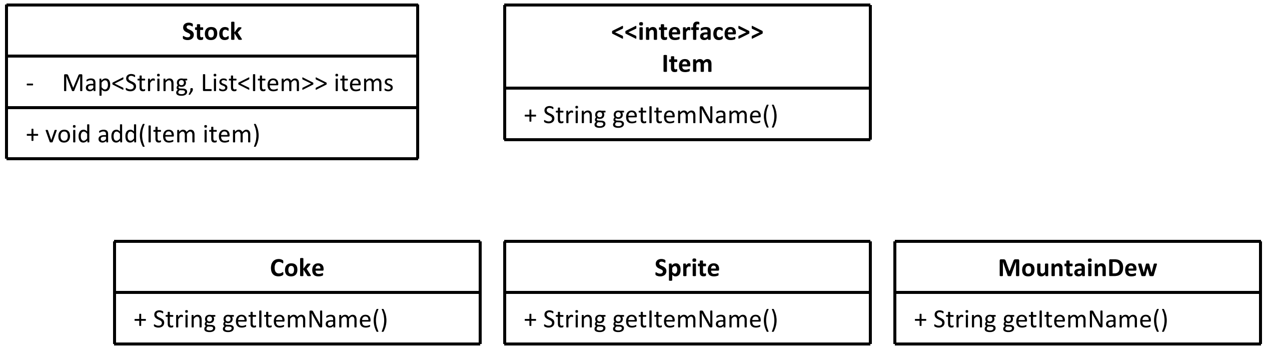
- 面试中频率低
- 现实Coding中很实用





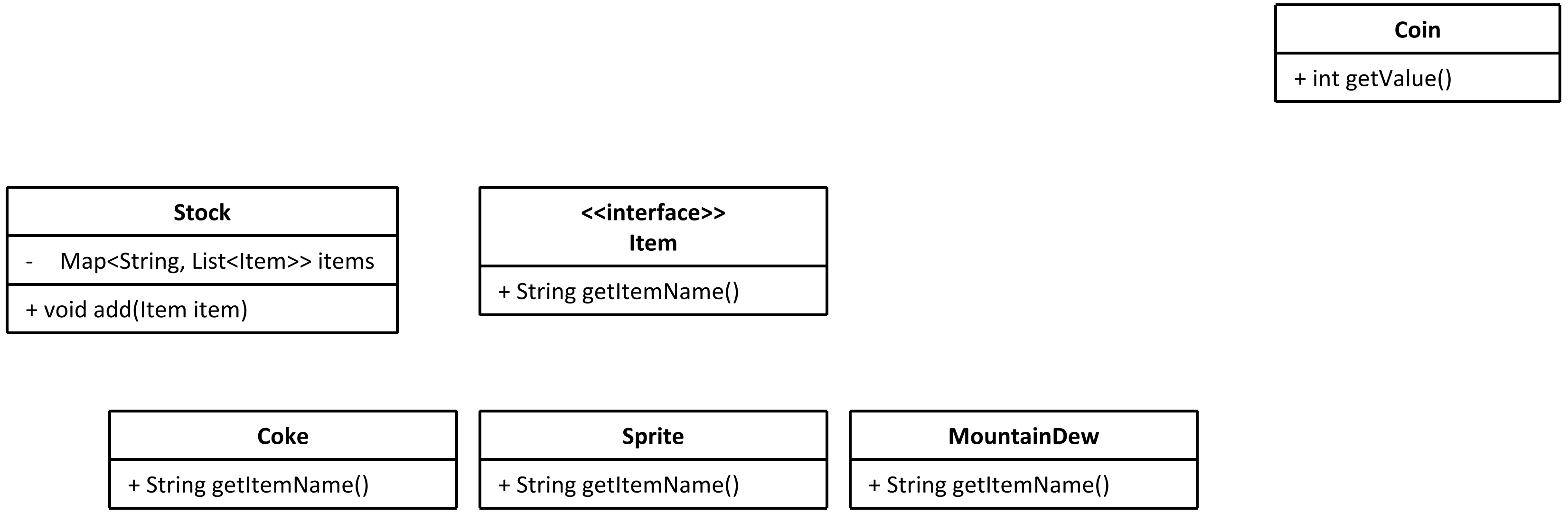
Adapter

- 例子:


Stock
- Map<String, List<Item>> items
+ void add(Item item)
<<interface>>
Item
+ String getItemName()
Coke
+ String getItemName()
Sprite
+ String getItemName()
MountainDew
+ String getItemName()

Adapter

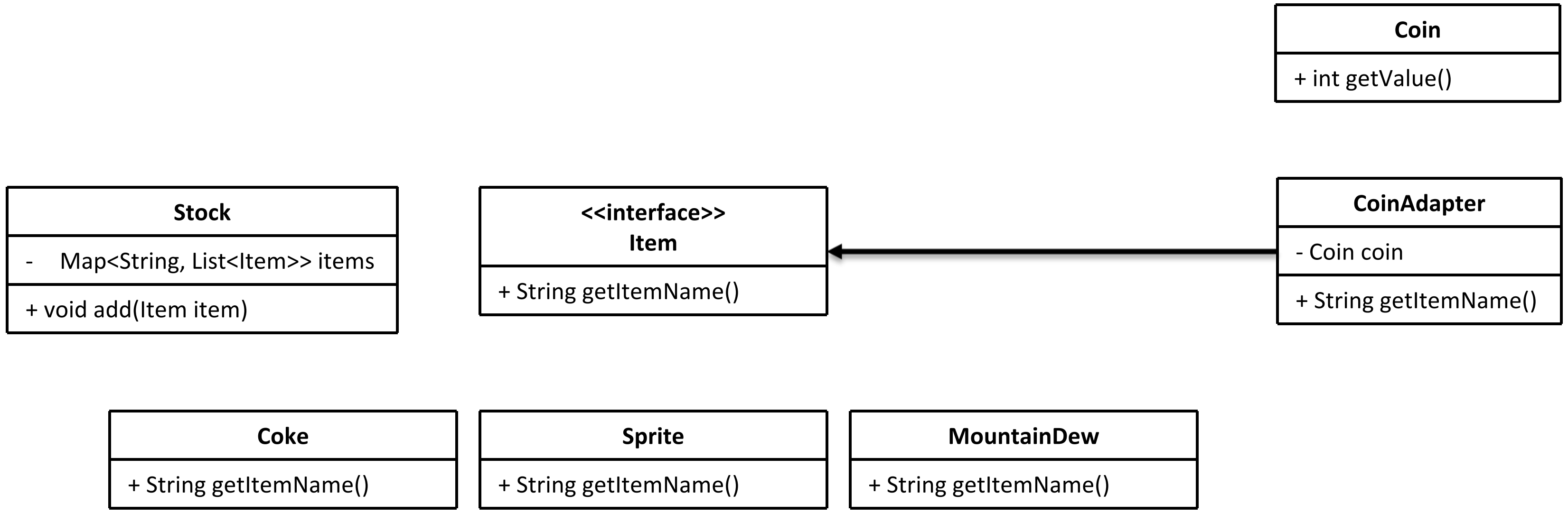
- 例子：



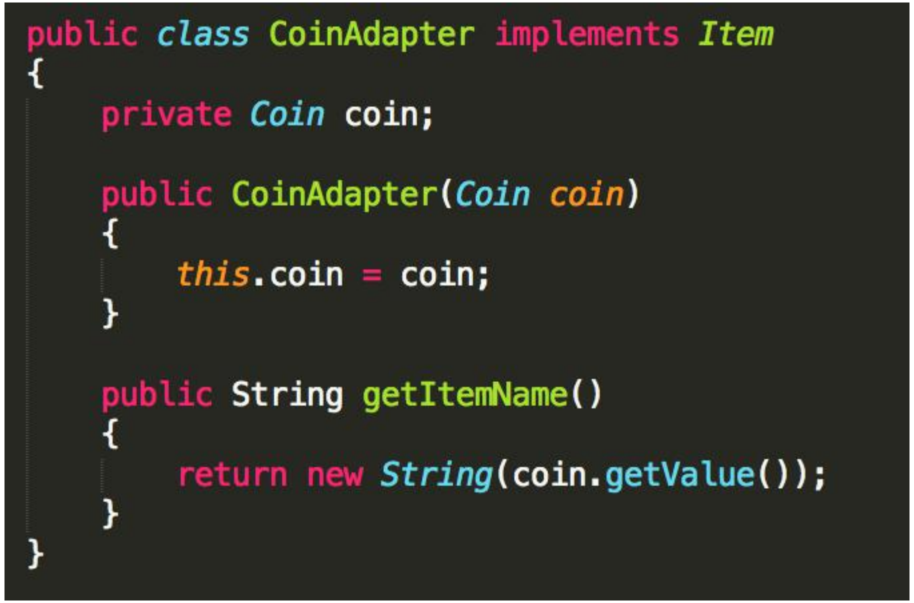
Coin
+ int getValue()
Stock
- Map<String, List<Item>> items
+ void add(Item item)
<<interface>>
Item
+ String getItemName()
Coke
+ String getItemName()
Sprite
+ String getItemName()
MountainDew
+ String getItemName()

Adapter

- 例子：

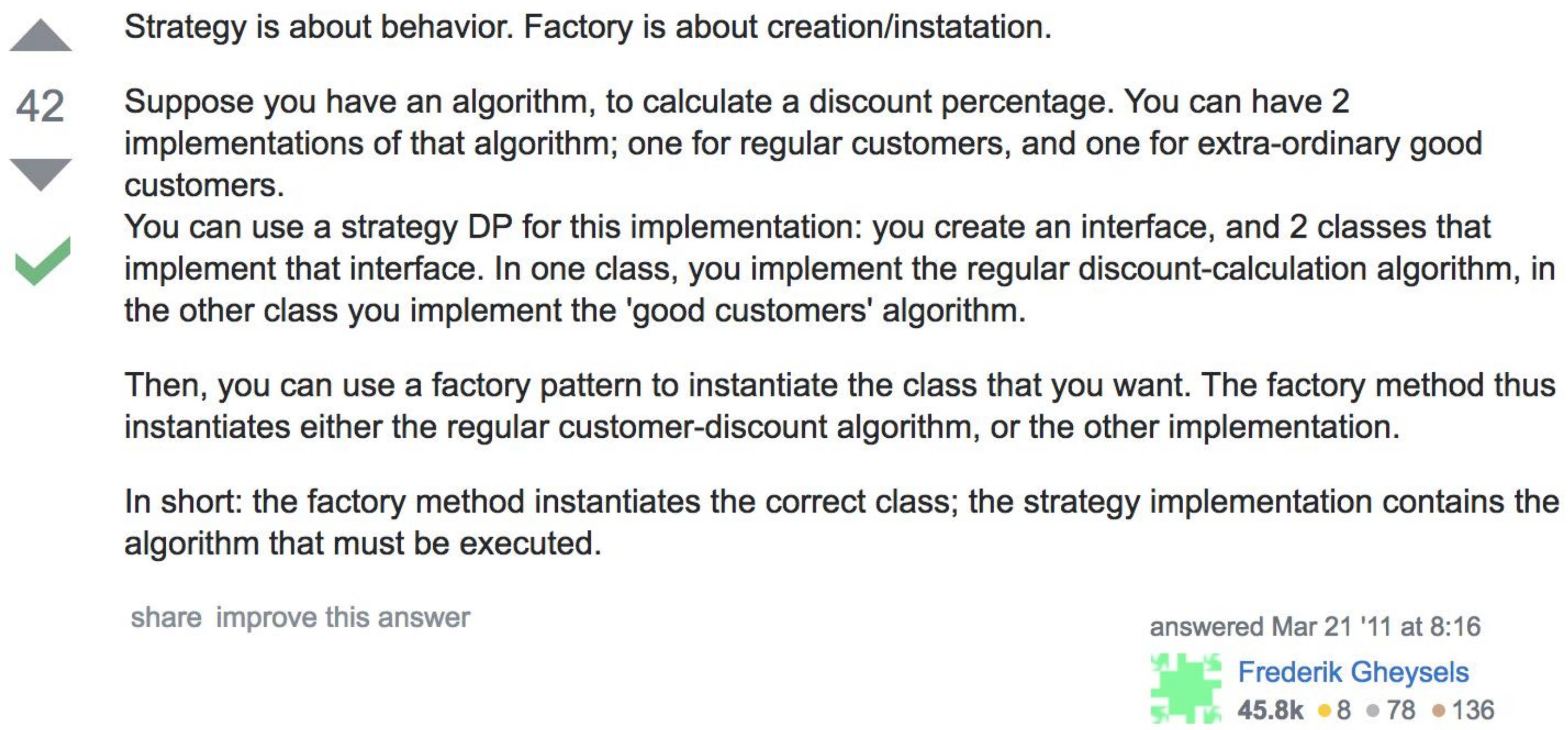

Coin
+ int getValue()
Stock
- Map<String, List<Item>> items
+ void add(Item item)
<<interface>>
Item
+ String getItemName()
CoinAdapter
- Coin coin
+ String getItemName()
Coke
+ String getItemName()
Sprite
+ String getItemName()
MountainDew
+ String getItemName()

Adapter


```
public class CoinAdapter implements Item
{
    private Coin coin;

    public CoinAdapter(Coin coin)
    {
        this.coin = coin;
    }

    public String getItemName()
    {
        return new String(coin.getValue());
    }
}
```

Strategy v.s. Factory

Strategy v.s. Factory



42

Strategy is about behavior. Factory is about creation/instatation.

Suppose you have an algorithm, to calculate a discount percentage. You can have 2 implementations of that algorithm; one for regular customers, and one for extra-ordinary good customers.
You can use a strategy DP for this implementation: you create an interface, and 2 classes that implement that interface. In one class, you implement the regular discount-calculation algorithm, in the other class you implement the 'good customers' algorithm.

Then, you can use a factory pattern to instantiate the class that you want. The factory method thus instantiates either the regular customer-discount algorithm, or the other implementation.

In short: the factory method instantiates the correct class; the strategy implementation contains the algorithm that must be executed.

share improve this answer

answered Mar 21 '11 at 8:16
Frederik Gheysels
45.8k ● 8 ● 78 ● 136

Strategy v.s. Factory

BookingSystem

- Strategy strategy

+ void pay(Payment payment)
- Void setStrategy(Strategy s)

<<interface>>
Strategy

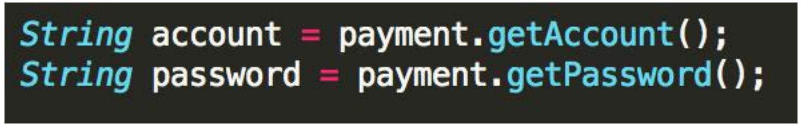
+ void pay(Payment payment)

<<interface>>
PaypalStrategy

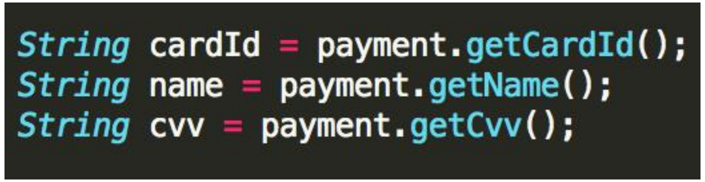
+ void pay(Payment payment)

<<interface>>
CreditCardStrategy

+ void pay(Payment payment)

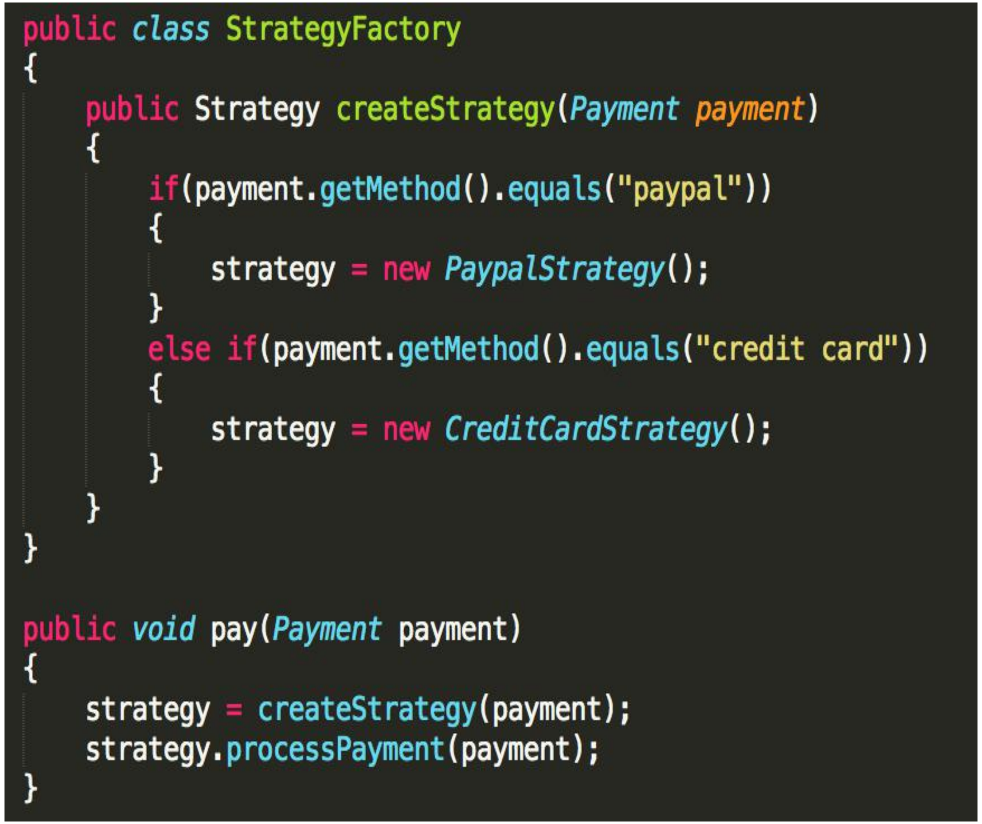


```
String account = payment.getAccount();
String password = payment.getPassword();
```

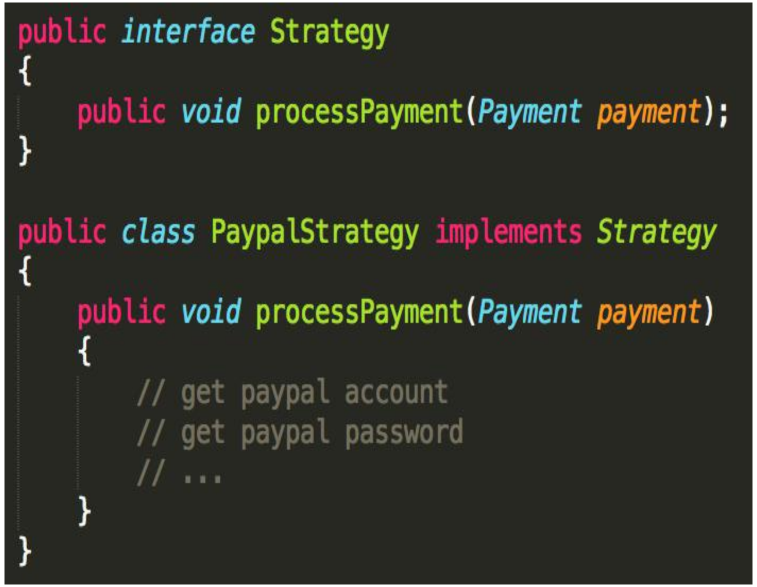


```
String cardId = payment.getCardId();
String name = payment.getName();
String cvv = payment.getCvv();
```

Strategy v.s. Factory

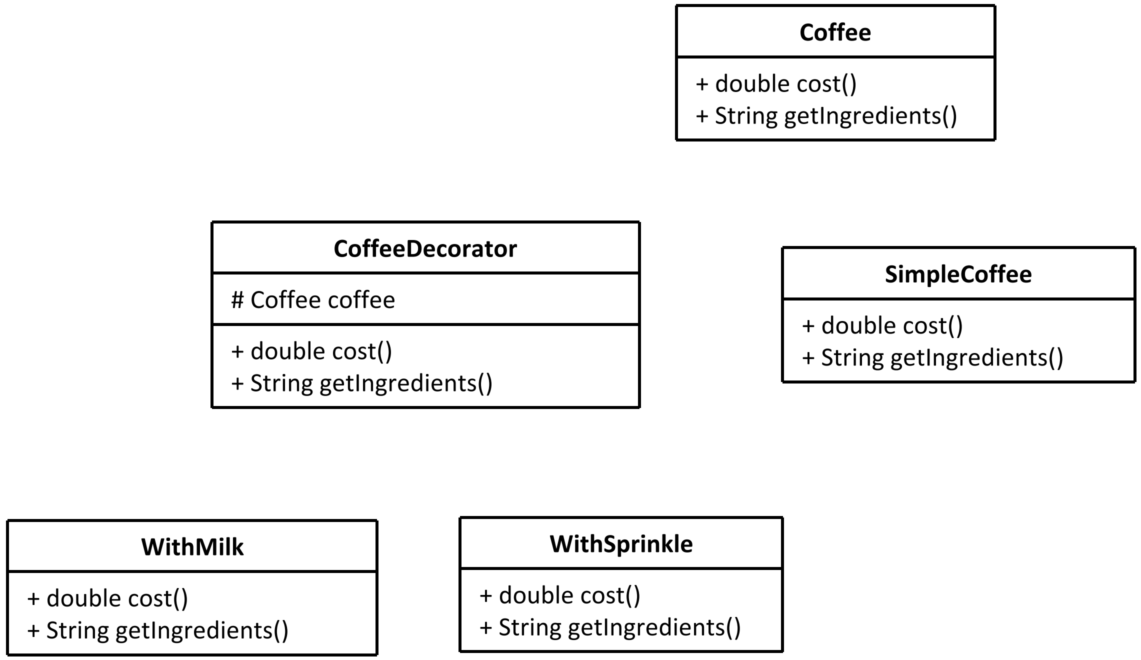


```
public class StrategyFactory
{
    public Strategy createStrategy(Payment payment)
    {
        if(payment.getMethod().equals("paypal"))
        {
            strategy = new PaypalStrategy();
        }
        else if(payment.getMethod().equals("credit card"))
        {
            strategy = new CreditCardStrategy();
        }
    }
}

public void pay(Payment payment)
{
    strategy = createStrategy(payment);
    strategy.processPayment(payment);
}
```



```
public interface Strategy
{
    public void processPayment(Payment payment);
}

public class PaypalStrategy implements Strategy
{
    public void processPayment(Payment payment)
    {
        // get paypal account
        // get paypal password
        // ...
    }
}
```

Decorator


Coffee
+ double cost()
+ String getIngredients()
CoffeeDecorator
# Coffee coffee
+ double cost()
+ String getIngredients()
SimpleCoffee
+ double cost()
+ String getIngredients()
WithMilk
+ double cost()
+ String getIngredients()
WithSprinkle
+ double cost()
+ String getIngredients()

Decorator

```
// The interface Coffee defines the functionality of Coffee implemented by decorator
public interface Coffee {
    public double getCost(); // Returns the cost of the coffee
    public String getIngredients(); // Returns the ingredients of the coffee
}

// Extension of a simple coffee without any extra ingredients
public class SimpleCoffee implements Coffee {
    @Override
    public double getCost() {
        return 1;
    }

    @Override
    public String getIngredients() {
        return "Coffee";
    }
}
```

Decorator

```java
// Abstract decorator class - note that it implements Coffee interface
public abstract class CoffeeDecorator implements Coffee {
    protected final Coffee decoratedCoffee;

    public CoffeeDecorator(Coffee c) {
        this.decoratedCoffee = c;
    }

    public double getCost() { // Implementing methods of the interface
        return decoratedCoffee.getCost();
    }

    public String getIngredients() {
        return decoratedCoffee.getIngredients();
    }
}
```

Decorator

```java
// Decorator WithMilk mixes milk into coffee.
// Note it extends CoffeeDecorator.
class WithMilk extends CoffeeDecorator {
    public WithMilk(Coffee c) {
        super(c);
    }

    public double getCost() { // Overriding methods defined in the abstract superclass
        return super.getCost() + 0.5;
    }

    public String getIngredients() {
        return super.getIngredients() + ", Milk";
    }
}

// Decorator WithSprinkles mixes sprinkles onto coffee.
// Note it extends CoffeeDecorator.
class WithSprinkles extends CoffeeDecorator {
    public WithSprinkles(Coffee c) {
        super(c);
    }

    public double getCost() {
        return super.getCost() + 0.2;
    }

    public String getIngredients() {
        return super.getIngredients() + ", Sprinkles";
    }
}
```

Decorator

```
public class Main {
    public static void printInfo(Coffee c) {
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());
    }

    public static void main(String[] args) {
        Coffee c = new SimpleCoffee();
        printInfo(c);

        c = new WithMilk(c);
        printInfo(c);

        c = new WithSprinkles(c);
        printInfo(c);
    }
}
```

The output of this program is given below:

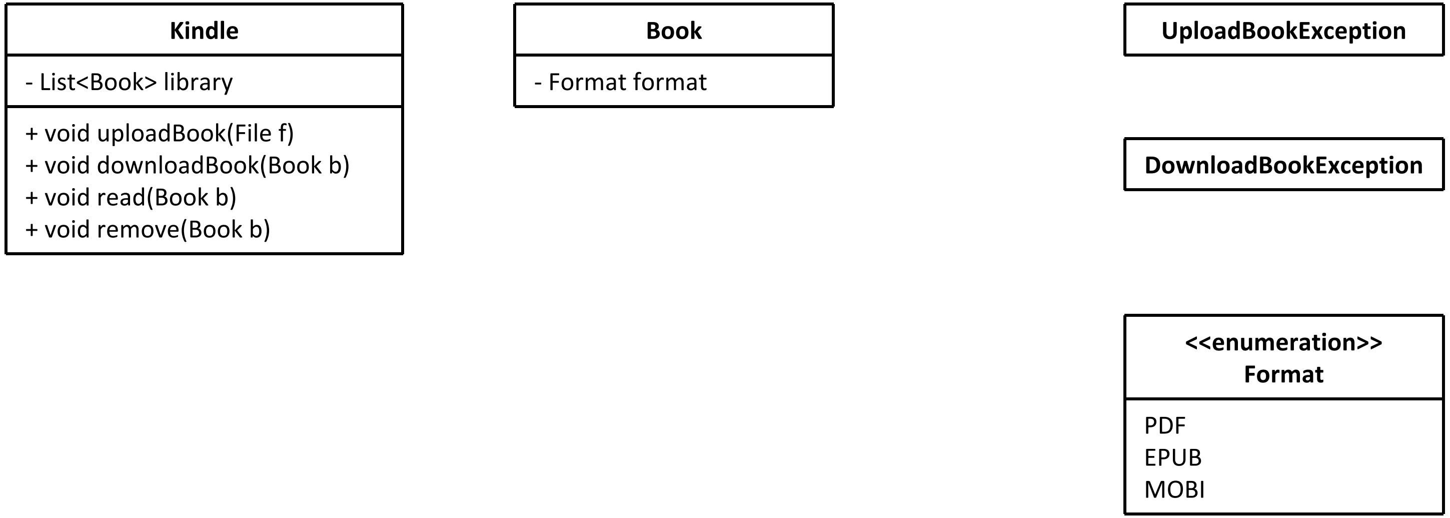
```
Cost: 1.0; Ingredients: Coffee
Cost: 1.5; Ingredients: Coffee, Milk
Cost: 1.7; Ingredients: Coffee, Milk, Sprinkles
```

Q & A

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
知乎专栏： http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com

Classes



Kindle

- List<Book> library

+ void uploadBook(File f)
+ void downloadBook(Book b)
+ void read(Book b)
+ void remove(Book b)

Book

- Format format

UploadBookException

DownloadBookException

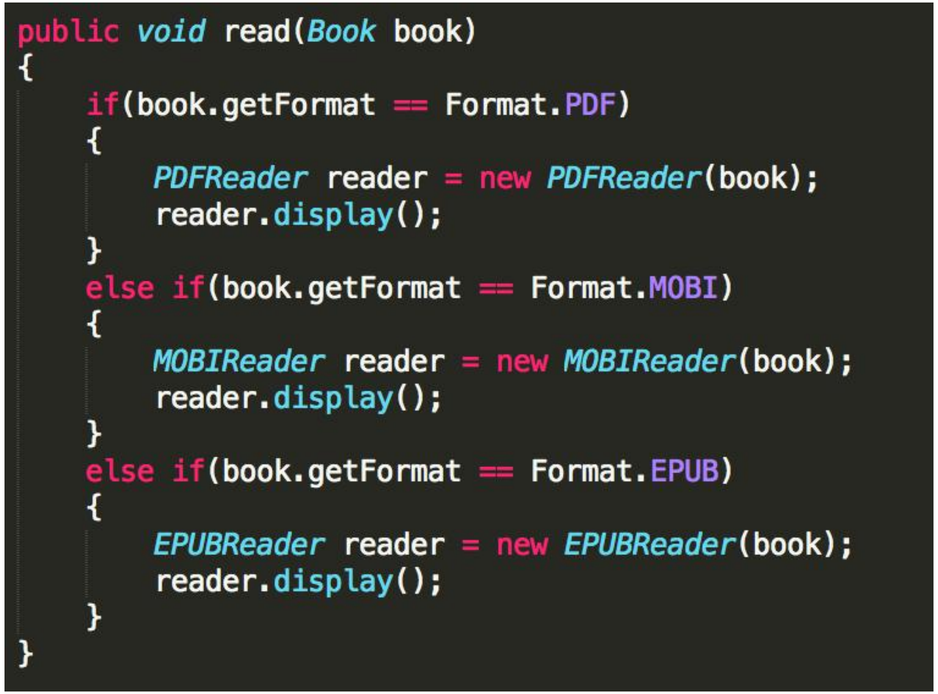
<<enumeration>>
Format

PDF
EPUB
MOBI

Use cases
Upload book
Download book
Read book
Remove book

Challenge

- How would read book work?

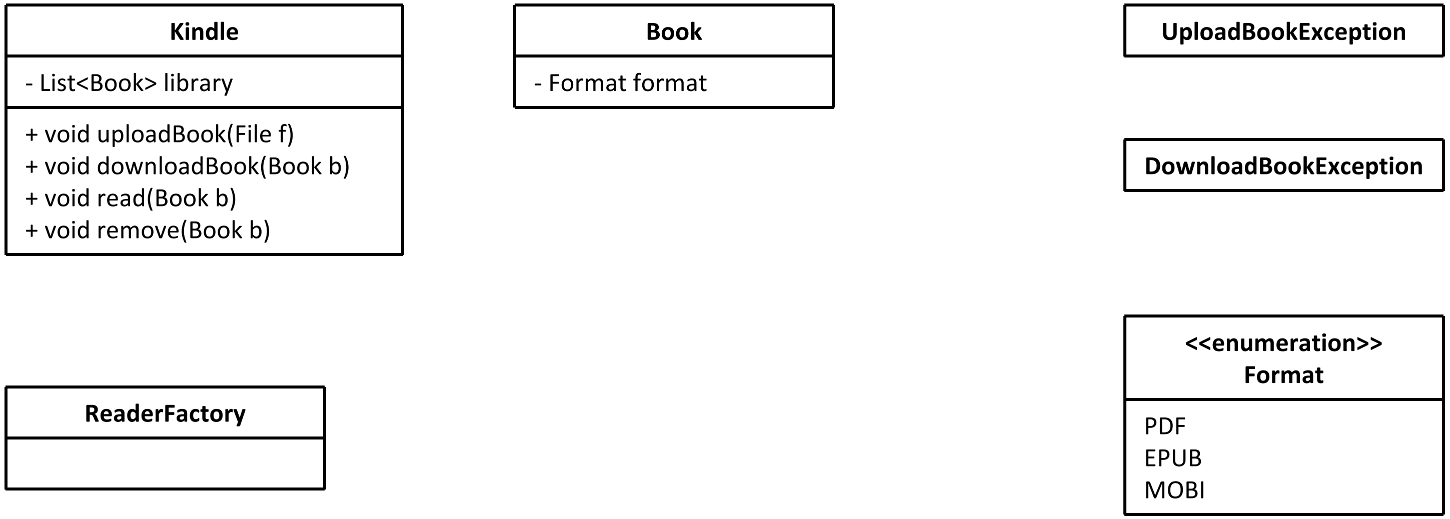


```java
public void read(Book book)
{
    if(book.getFormat == Format.PDF)
    {
        PDFReader reader = new PDFReader(book);
        reader.display();
    }
    else if(book.getFormat == Format.MOBI)
    {
        MOBIReader reader = new MOBIReader(book);
        reader.display();
    }
    else if(book.getFormat == Format.EPUB)
    {
        EPUBReader reader = new EPUBReader(book);
        reader.display();
    }
}
```

Challenge

- Solution: Factory design pattern

Factory design pattern



Kindle

- List<Book> library

+ void uploadBook(File f)
+ void downloadBook(Book b)
+ void read(Book b)
+ void remove(Book b)

Book

- Format format

UploadBookException

DownloadBookException

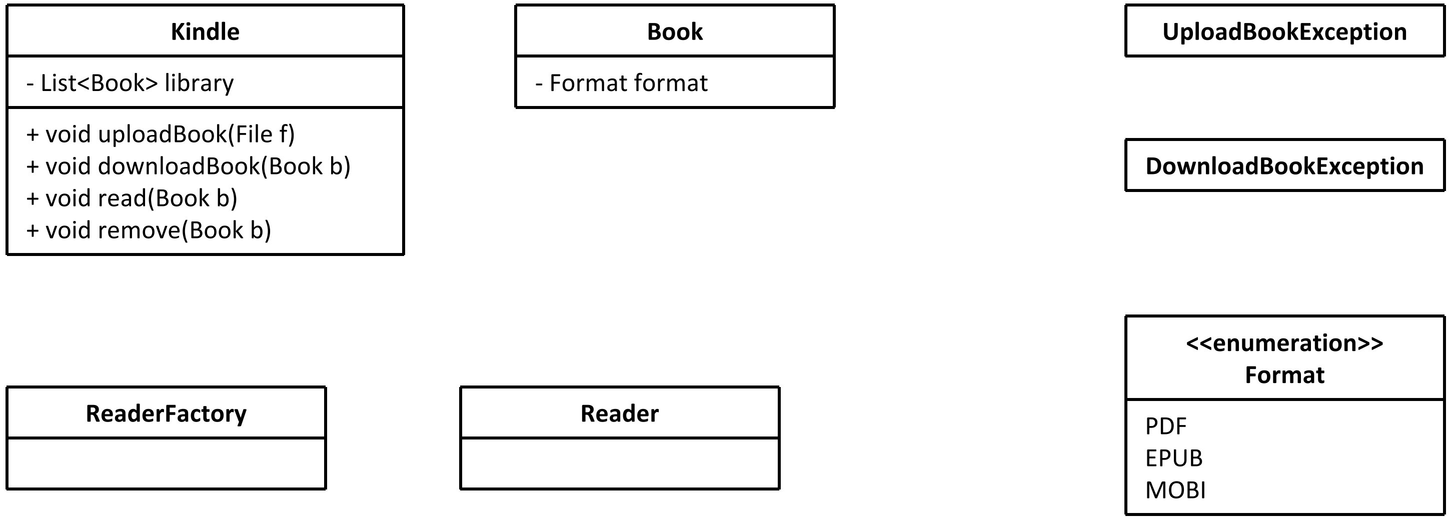
ReaderFactory

<<enumeration>>
Format

PDF
EPUB
MOBI

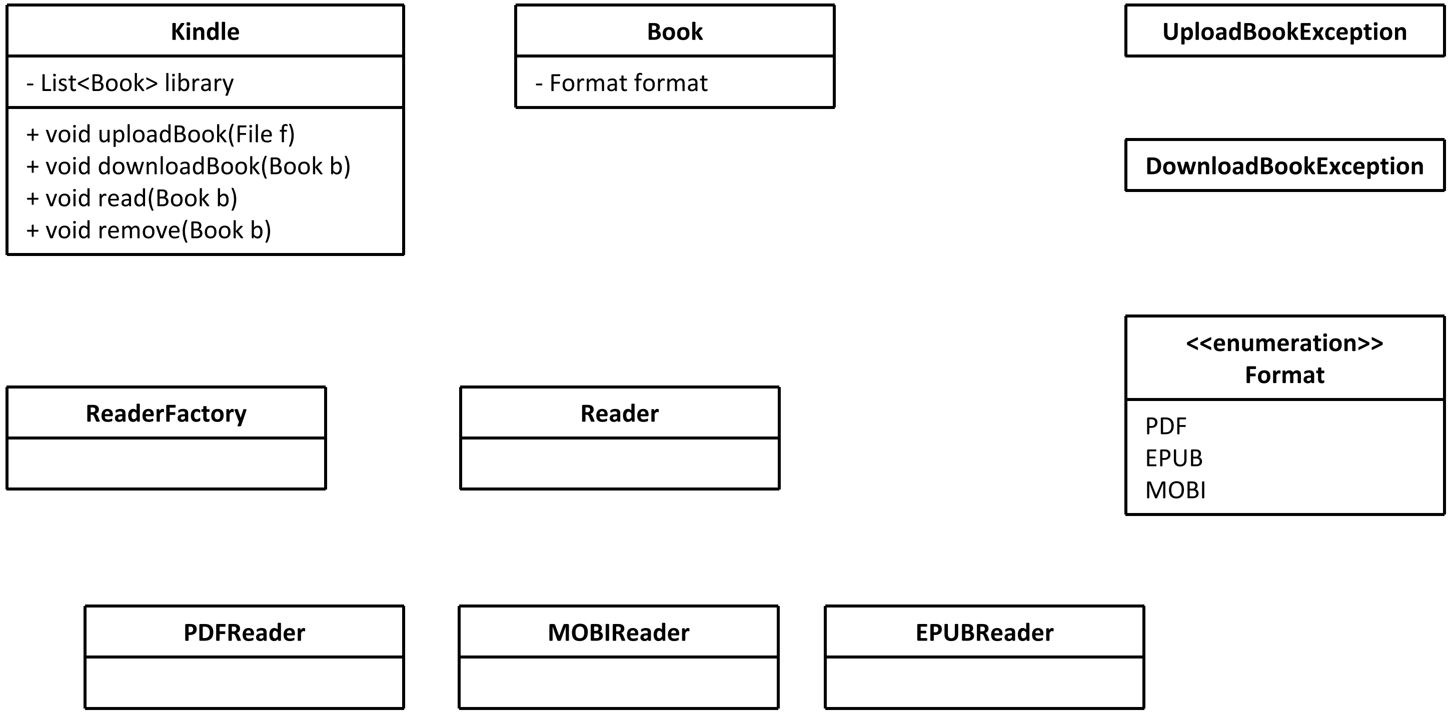
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



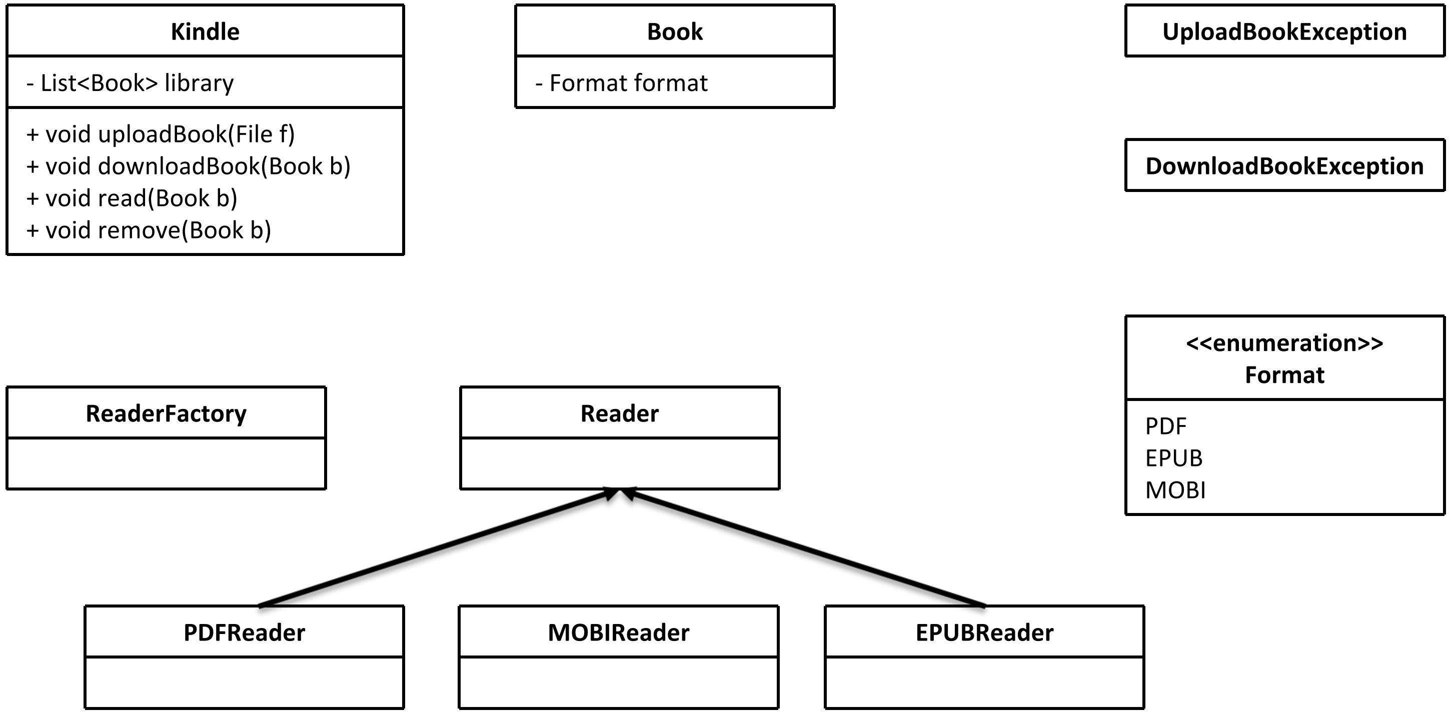
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



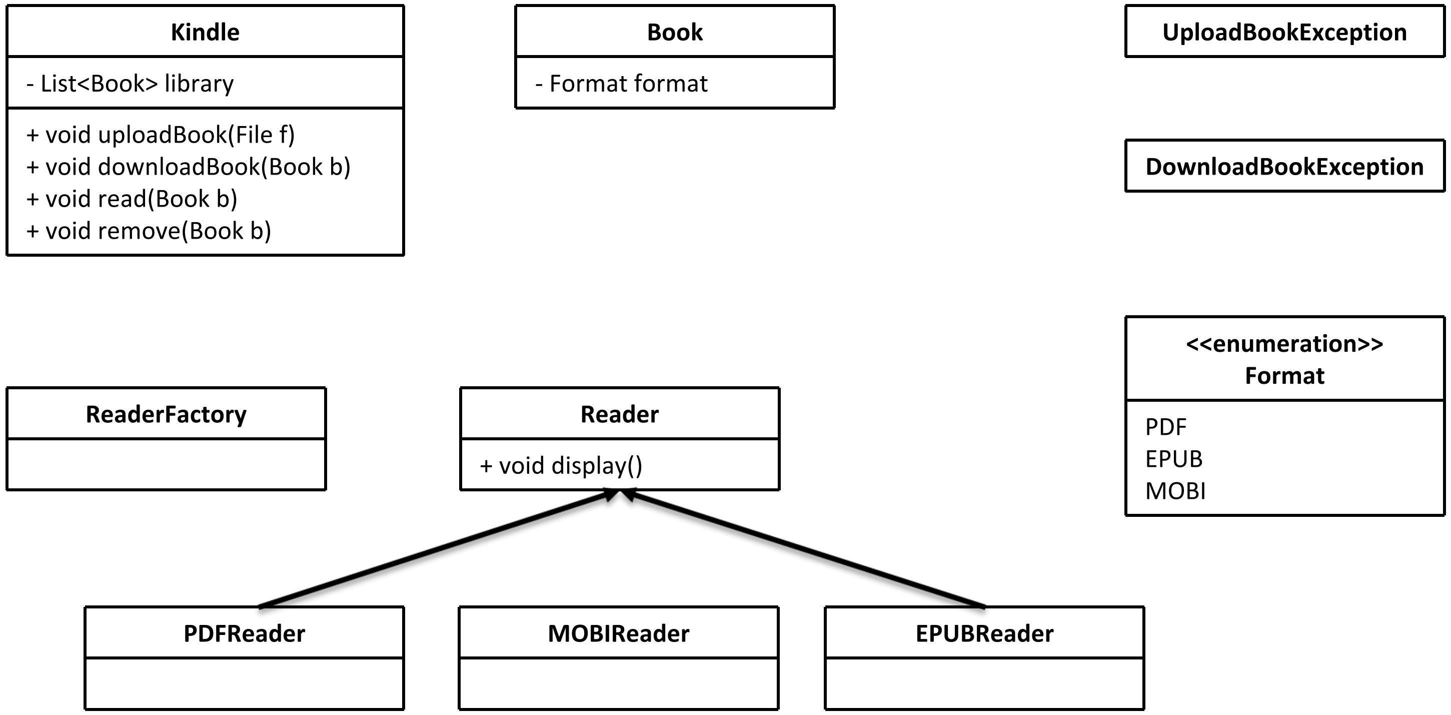
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



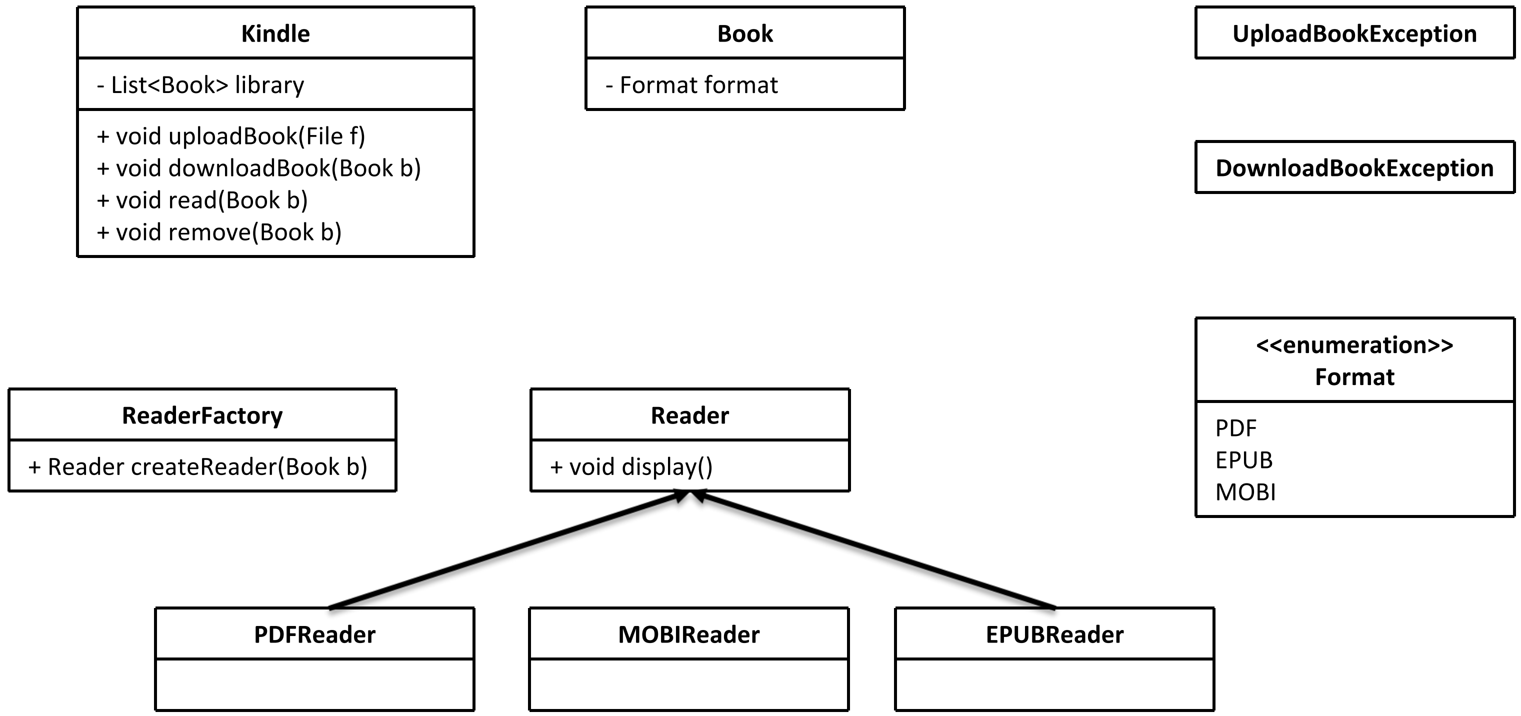
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



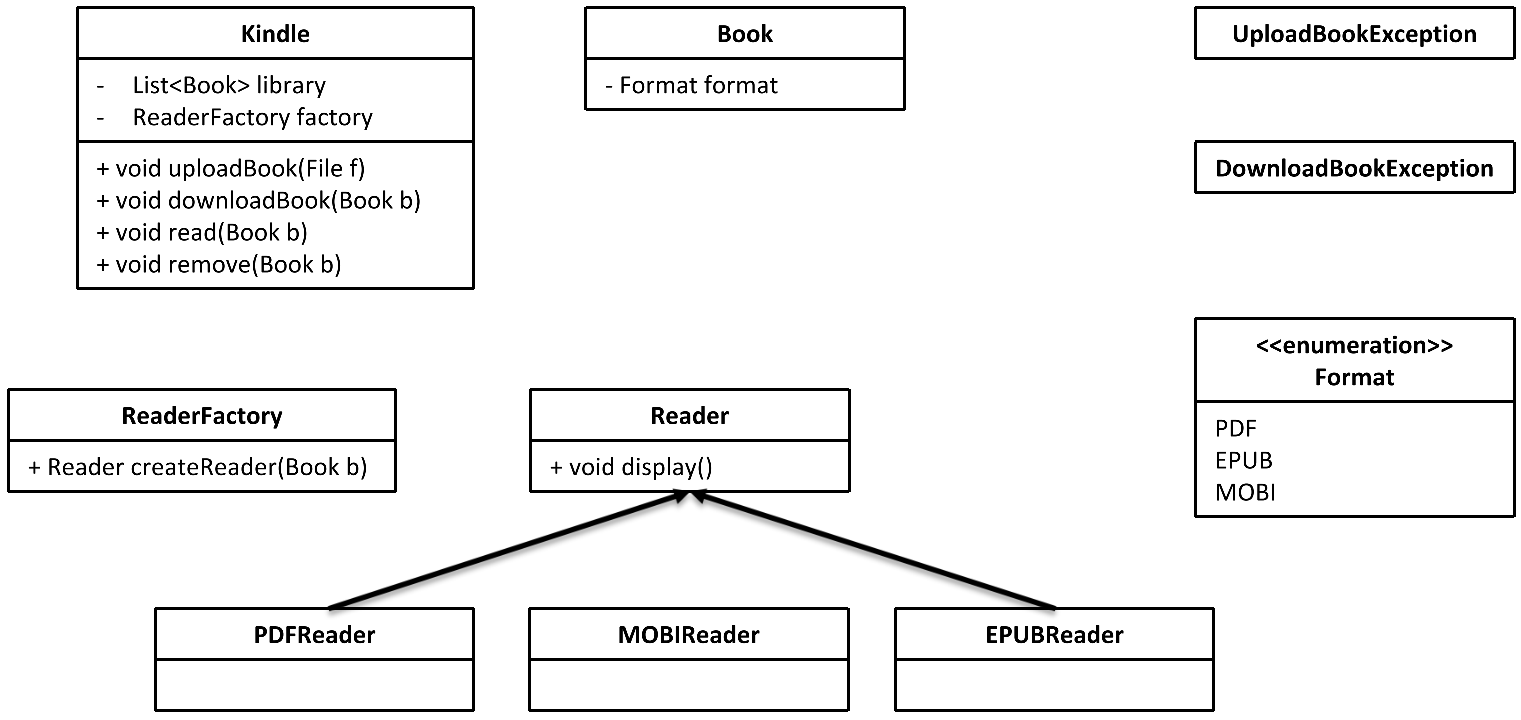
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



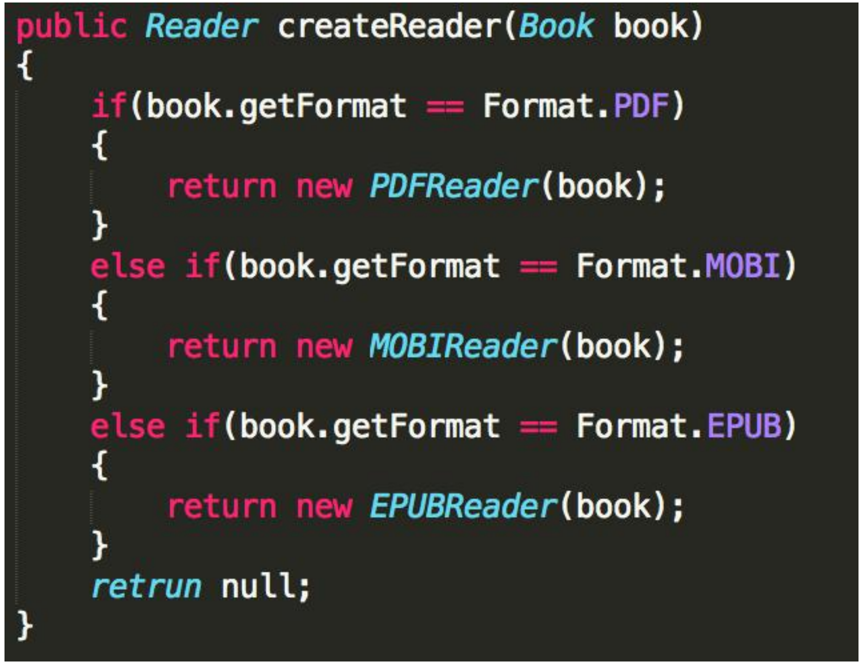
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern

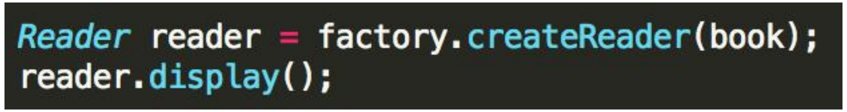


Use cases
Upload book
Download book
Read book
Remove book

Simple factory



```
public Reader createReader(Book book)
{
    if(book.getFormat == Format.PDF)
    {
        return new PDFReader(book);
    }
    else if(book.getFormat == Format.MOBI)
    {
        return new MOBIReader(book);
    }
    else if(book.getFormat == Format.EPUB)
    {
        return new EPUBReader(book);
    }
    retrun null;
}
```

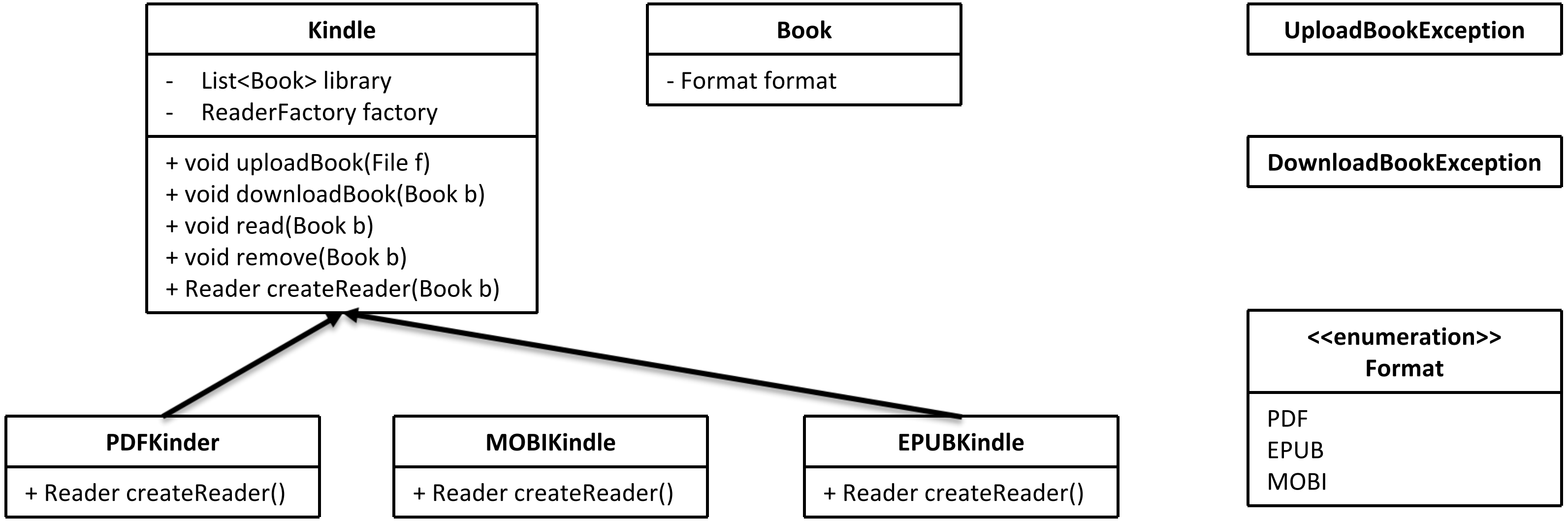


```
Reader reader = factory.createReader(book);
reader.display();
```

Factory design pattern

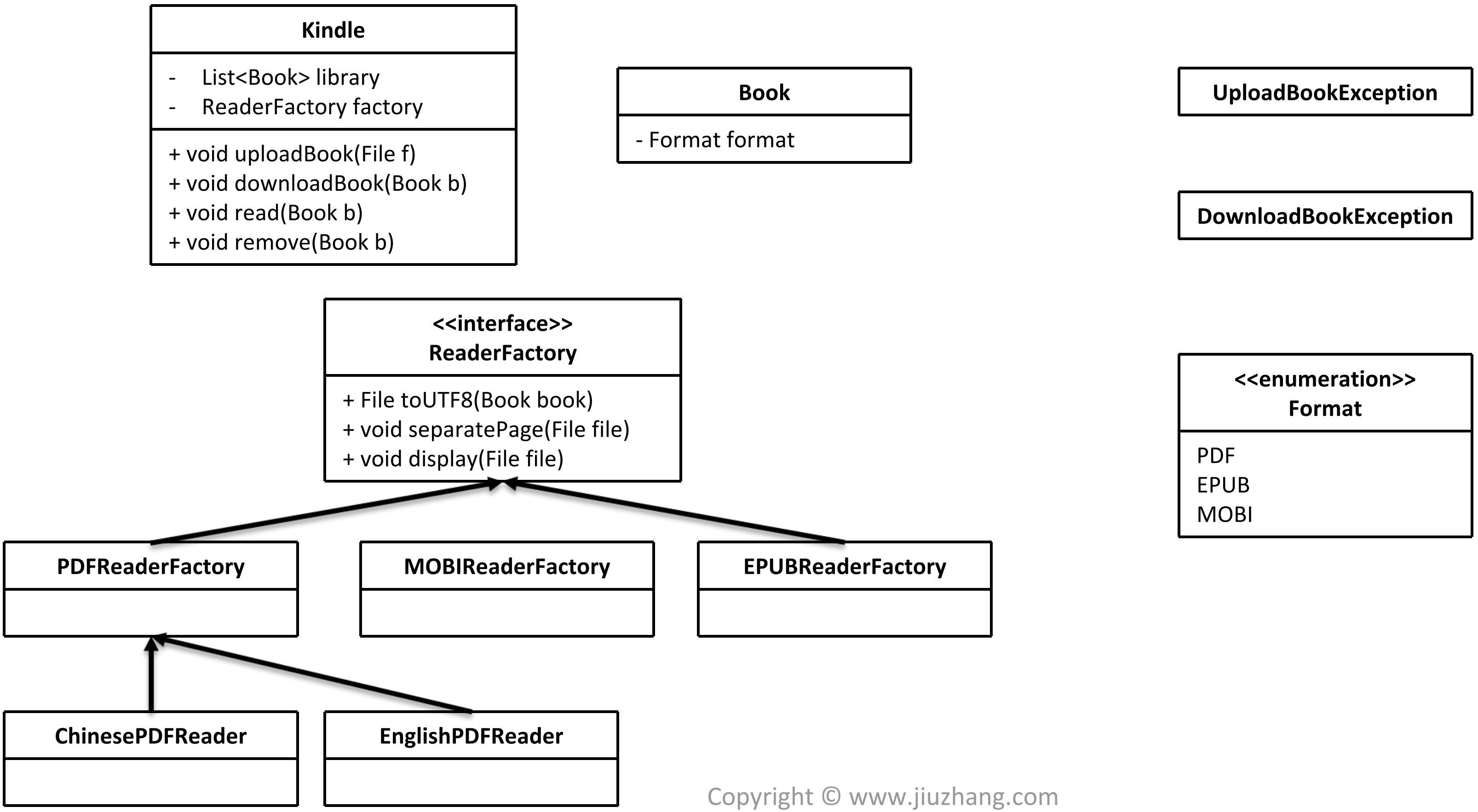
- Factory method
- Abstract factory

Factory method



Use cases
Upload book
Download book
Read book
Remove book

Abstract factory



Use cases
Upload book
Download book
Read book
Remove book