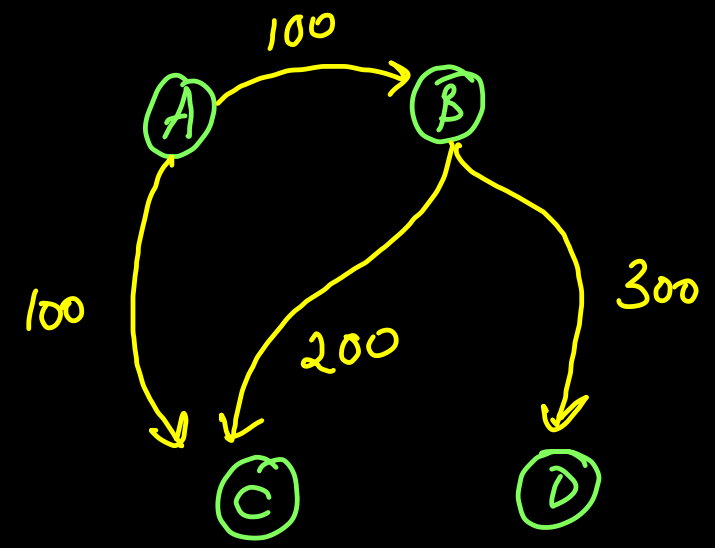
Design Splitwise

splitwise
khatabook
paytm (split)

Problem Statement

Create an expense sharing application.

An expense sharing application is where you can add your expenses and split it among different people. The app keeps balances between people as in who owes how much to whom.



simplify/minimize

#

Example

You live with 3 other friends.

You: User1 (id: u1)

Flatmates: User2 (u2), User3 (u3), User4 (u4)

This month's electricity bill was Rs. 1000.

Now you can just go to the app and add that you paid 1000,

select all the 4 people and then select split equally.

Input: u1 1000 4 u1 u2 u3 u4 EQUAL

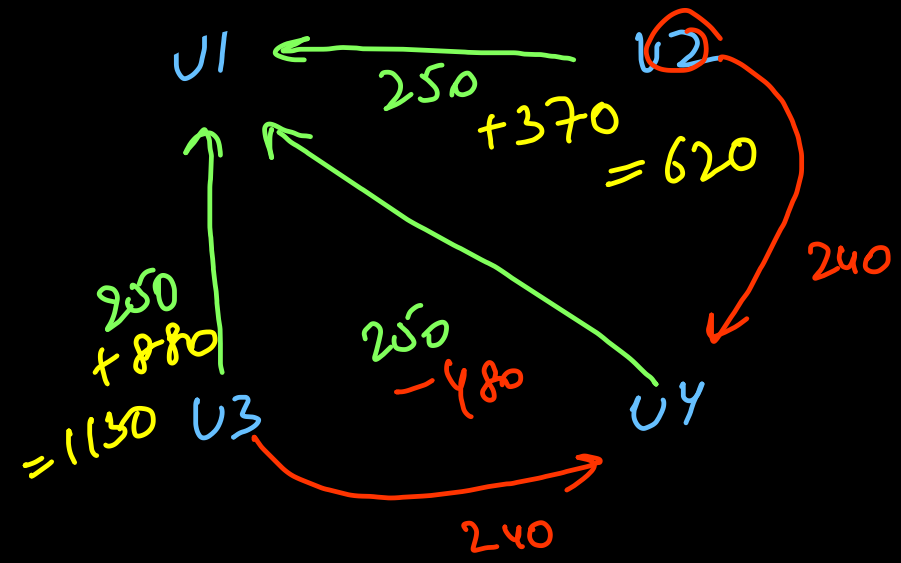
For this transaction, everyone owes 250 to User1.

The app should update the balances in each of the profiles accordingly.

User2 owes User1: 250 (0+250)

User3 owes User1: 250 (0+250)

User4 owes User1: 250 (0+250)



Now, It is the BBD sale on Flipkart and there is an offer on your card.

You buy a few stuffs for User2 and User3 as they asked you to.

The total amount for each person is different.

Input: u1 1250 2 u2 u3 EXACT 370 880

For this transaction, User2 owes 370 to User1 and User3 owes 880 to User1.

The app should update the balances in each of the profiles accordingly.

User2 owes User1: 620 (250+370)

User3 owes User1: 1130 (250+880)

User4 owes User1: 250 (250+0)

Now, you go out with your flatmates and take your brother/sister along wit

User4 pays and everyone splits equally. You owe for 2 people.

Input: u4 1200 4 u1 u2 u3 u4 PERCENT 40 20 20 20

For this transaction, User1 owes 480 to User4, User2 owes 240 to User4 and

The app should update the balances in each of the profiles accordingly.

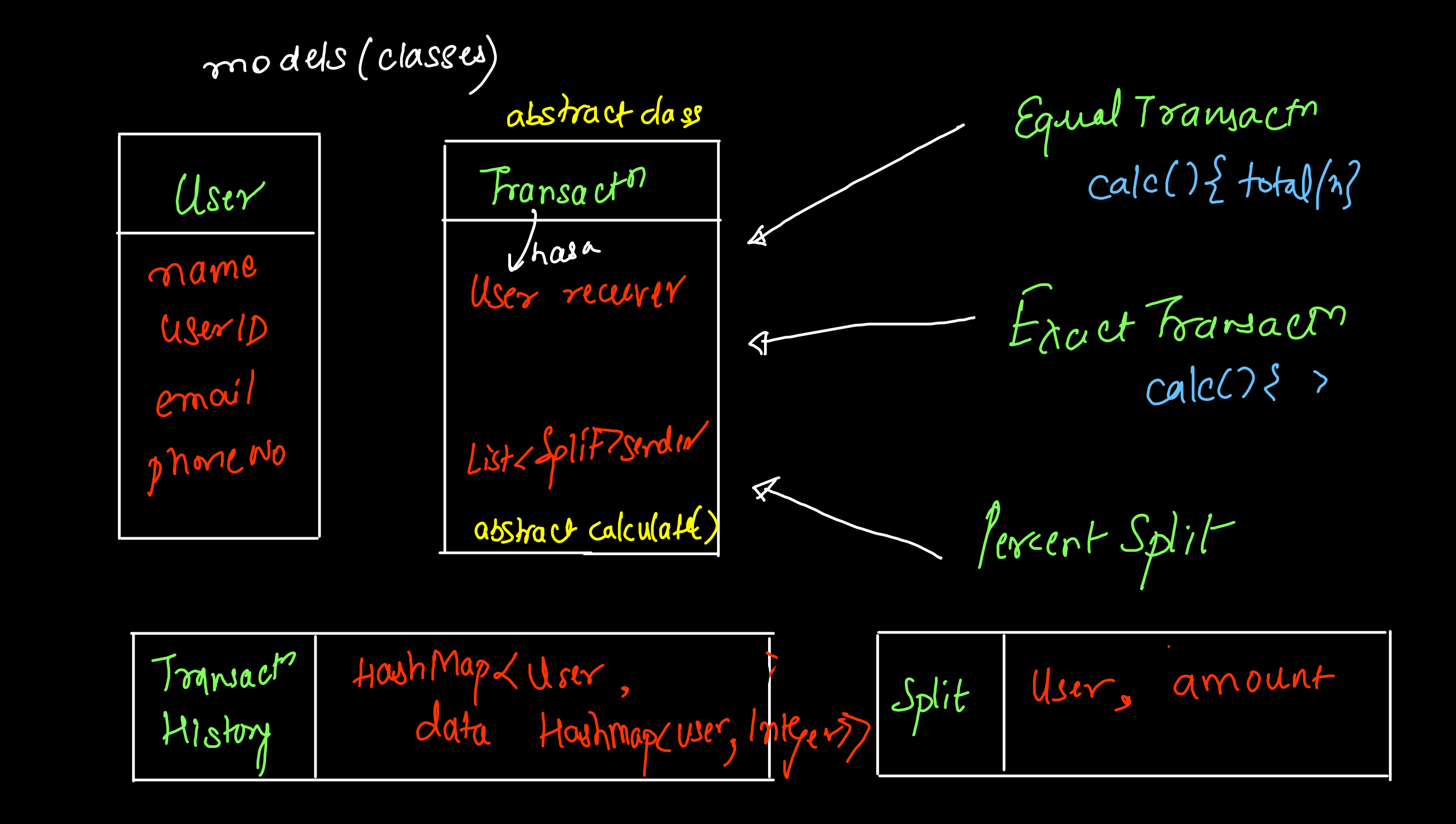
User1 owes User4: 230 (250-480)

User2 owes User1: 620 (620+0)

User2 owes User4: 240 (0+240)

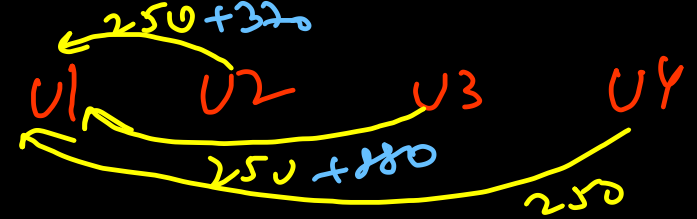
User3 owes User1: 1130 (1130+0)

User3 owes User4: 240 (0+240)


models (classes)
User
name
UserID
email
phoneNo
abstract class
Transactn
has a
User receiver
List<Split?sender
abstract calculate()
Equal Transactn
calc(){ total/n}
Exact Transactn
calc(){ >
Percent Split
Transactn History
HashMap<User,
data Hashmap(user, Integer>>
Split
User, amount

Sample Input

```
SHOW
SHOW u1
EXPENSE u1 1000 4 u1 u2 u3 u4 EQUAL
SHOW u4
SHOW u1
EXPENSE u1 1250 2 u2 u3 EXACT 370 880
SHOW
EXPENSE u4 1200 4 u1 u2 u3 u4 PERCENT 40 20 20 20
SHOW u1
SHOW
```



"No balances"

"No balances"

"U4 owes U1 : 250"

"U2 owes U1 : 250
U3 owes U1 : 250
U4 owes U1 : 250"

"U2 owes U1 : 620
U3 owes U1 : 1130
U4 owes U1 : 250"

Models

```java
public class User {
    private String id;
    private String name;
    private String email;
    private String phone;
}
```

```java
public class Split {
    private String userID;
    private double amount;
}
```

sender

Expense

u1 1000 2 u2 u3 PERCENT

50 20

u2 → 500 → u1

u3 → 200 → u1

validate "false"

```java
public enum ExpenseType {
    EQUAL {
        @Override
        public double calcAmount(double amount, double countSplit) {
            return amount / countSplit;
        }
    },
    EXACT {
        @Override
        public double calcAmount(double amount, double exactSplit) {
            return exactSplit;
        }
    },
    PERCENT {
        @Override
        public double calcAmount(double amount, double percentSplit) {
            return (amount * percentSplit) / 100.0;
        }
    };

    public abstract double calcAmount(double totalAmount, double split);
```

You, 4 months ago • Added Design Splitwise

open for extension closed for modification

```java
public boolean validate(double totalAmount, List<Split> splits) {
    double sumSplit = 0.0;
    for (Split split : splits) {
        sumSplit += split.getAmount();
    }

    return ((sumSplit - totalAmount) <= 0.1);
}
```

service/controller

```
public class Dao {
    private Map<String, User> users;
```

```
private Map<String, Map<String, Double>> balances;
```

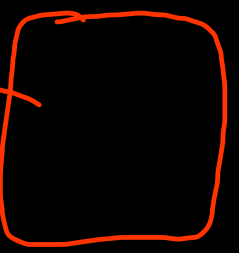
receiver

Sender

Amount

data (remove

Database



Requirement : Only single instance should be used

⇒ Singleton design pattern

⇒ DB connection (single)

⇒ Logging systems

```
class Singleton{
    static Singleton instance;
    private Singleton(){} //private constructor

    public static Singleton getInstance(){
        if(instance == null){
            instance = new Singleton();
        }
        return instance;
    }

    public void fun(){
        System.out.println(this);
    }
}
```

```
Singleton obj1 = Singleton.getInstance();
obj1.fun();
Singleton obj2 = Singleton.getInstance();
obj2.fun();
```

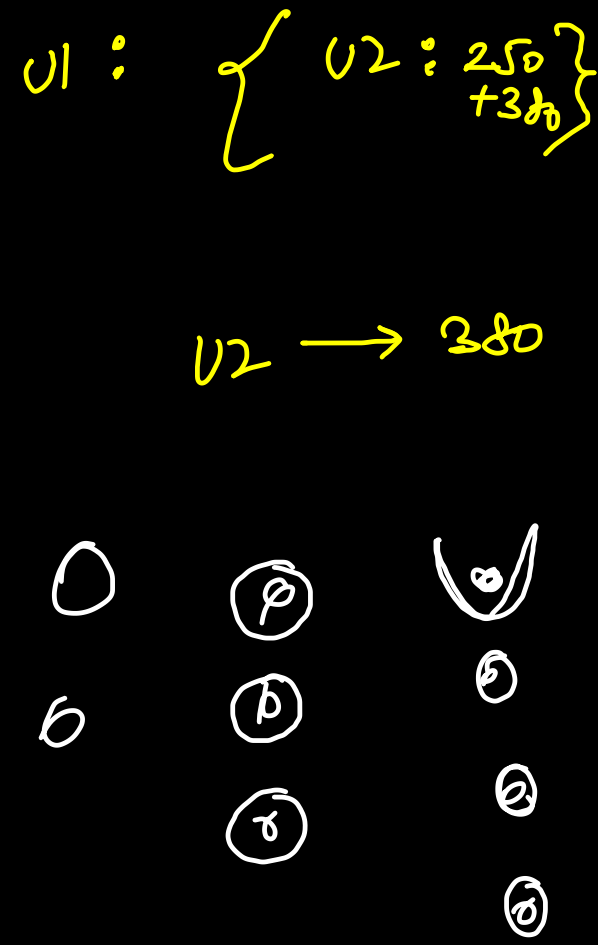
same hashcode

```java
public class Dao {
    private Map<String, User> users;
    private Map<String, Map<String, Double>> balances;

    private static Dao instance = null;

    private Dao() {
        users = new HashMap<>();
        balances = new HashMap<>();
    }

    public static Dao getInstance() {
        if (instance == null) {
            instance = new Dao();
        }
        return instance;
    }
```

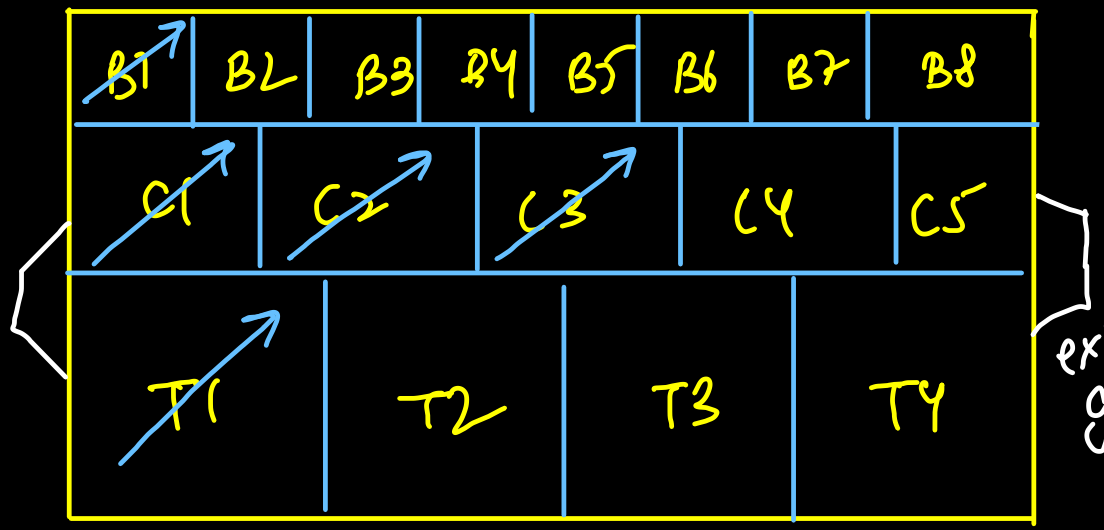
Singleton class design pattern

Top Bottom

```java
public void addExpense(String creditor, Split split) {
    String debtor = split.getUser();

    Map<String, Double> balanceMap = balances.get(creditor);
    balanceMap.put(debtor, balanceMap.getOrDefault(debtor, defaultValue: 0.0)
            + split.getAmount());

    balanceMap = balances.get(debtor);
    balanceMap.put(creditor, balanceMap.getOrDefault(creditor, defaultValue: 0.0)
            - split.getAmount());
}

public void showBalance(String user1) {
    boolean isEmpty = true;

    for (String user2 : balances.get(user1).keySet()) {
        double balance = balances.get(user1).get(user2);
        if (balance != 0) {
            printBalance(user1, user2, balance);
            isEmpty = false;
        }
    }

    if (isEmpty == true) {
        System.out.println(x: "No balances");
    }

    System.out.println();
}
```

Parking lot



abstract class
Vehicle

Bike Car Truck

Ticket
→ vehicle(bj)
→ slot (id)
→ entryTime

① Vehicle ② Parking slots ③ Parking floors & buildings

④ Entry Gate & Exit Gate ⑤ Payment method ⑥ Ticket

- The functions that the parking lot system can do:
  - Create the parking lot.
  - Add floors to the parking lot.
  - Add a parking lot slot to any of the floors.
  - Given a vehicle, it finds the first available slot, books it, creates a ticket, parks the vehicle, and finally returns the ticket.
  - Unparks a vehicle given the ticket id.
  - Displays the number of free slots per floor for a specific vehicle type.
  - Displays all the free slots per floor for a specific vehicle type.
  - Displays all the occupied slots per floor for a specific vehicle type.
- Details about the Vehicles:
  - Every vehicle will have a type, registration number, and color.
  - Different Types of Vehicles:
    - Car
    - Bike
    - Truck

Service

```java
public class ParkingLot {
    List<Slot> bikes, cars, trucks;

    ParkingLot() {
        bikes = new ArrayList<>();
        cars = new ArrayList<>();
        trucks = new ArrayList<>();
    }
```

```java
void initSlots() {
    for (int i = 1; i <= 8; i++)
        bikes.add(new Slot("B" + i));
    for (int i = 1; i <= 5; i++)
        cars.add(new Slot("C" + i));
    for (int i = 1; i <= 4; i++)
        trucks.add(new Slot("T" + i));
}

Ticket park(Vehicle vehicle) {
    Slot freeSlot = null;
    if (vehicle instanceof Bike) {
        freeSlot = findSlot(type: "BIKE");
    } else if (vehicle instanceof Car) {
        freeSlot = findSlot(type: "CAR");
    } else if (vehicle instanceof Truck) {
        freeSlot = findSlot(type: "TRUCK");
    }

    if (freeSlot == null)
        return null;
    return freeSlot.parkVehicle(vehicle);
}
```

```java
Slot findSlot(String type) {
    if (type.equals(anObject: "BIKE")) {
        for (Slot slot : bikes) {
            if (slot.isEmpty == true) {
                return slot;
            }
        }
        return null;
    } else if (type.equals(anObject: "CAR")) {
        for (Slot slot : cars) {
            if (slot.isEmpty == true) {
                return slot;
            }
        }
        return null;
    } else if (type.equals(anObject: "TRUCK")) {
        for (Slot slot : trucks) {
            if (slot.isEmpty == true) {
                return slot;
            }
        }
    }
    return null;
}
```

models

```java
public abstract class Vehicle {
    String numberPlate;
    String color;

    public Vehicle(String numberPlate, String color) {
        this.numberPlate = numberPlate;
        this.color = color;
    }

    public String getNumberPlate() {
        return numberPlate;
    }

    public void setNumberPlate(String numberPlate) {
        this.numberPlate = numberPlate;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

```java
public class Slot {
    String slotNo;
    Vehicle vehicle;
    boolean isEmpty = true;

    public Slot(String slotNo) {
        this.slotNo = slotNo;
    }

    public Ticket parkVehicle(Vehicle vehicle) {
        isEmpty = false;
        this.vehicle = vehicle;
        return new Ticket(vehicle, this);
    }
}
```

```java
public class Ticket {
    Vehicle vehicle;
    Slot slot;
    LocalTime entryTime;

    public Ticket(Vehicle vehicle, Slot slot) {
        this.vehicle = vehicle;
        this.slot = slot;
        this.entryTime = LocalTime.now();
    }

    @Override
    public String toString() {
        return slot.slotNo + "-" + vehicle.numberPlate + "-" + entryTime;
    }
}
```

Design ATM

1) deposit (4, 0, 4, 0, 2)

2) withdraw (740)

res = { 0, 0, 0, 0, 0 }

2 2 1

withdraw (~~820~~) ~~320~~ ~~120~~ 80

{ ~~0~~, 0, ~~0~~, 0, ~~0~~ }

2 2 1

{-1}

0 1 2 3 4

20, 50, 100, 200, 500

int[] = { ~~4~~, 0, ~~4~~, 0, ~~2~~ }

2 2 1

```java
class ATM {
    int[] notes = {0, 0, 0, 0, 0};
    int[] types = {20, 50, 100, 200, 500};

    public void deposit(int[] change) {
        for(int i = 0; i < notes.length; i++){
            notes[i] += change[i];
        }
    }

    public int[] withdraw(int amount) {
        int[] change = {0, 0, 0, 0, 0};

        for(int i = change.length - 1; i >= 0; i--){
            change[i] = Math.min(notes[i], amount / types[i]);
            amount -= change[i] * types[i];
        }

        if(amount != 0) return new int[]{-1};

        for(int i = 0; i <= 4; i++)
            notes[i] -= change[i];
        return change;
    }
}
```

→ edge overflow handling.

{Greedy Algorithm}

$n = 3$

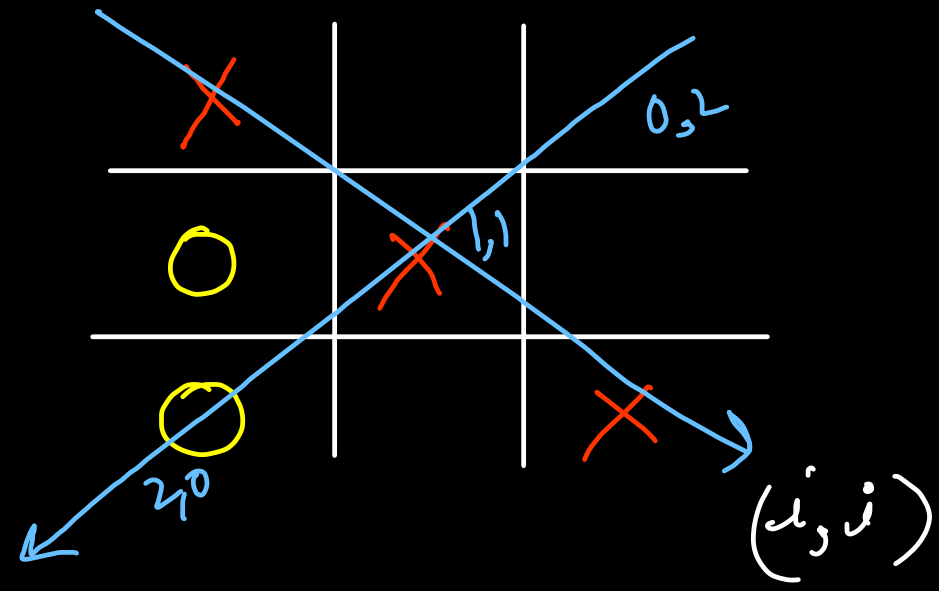
move(0, 0, 1) = 0

move(1, 0, 2) = 0

move(1, 1, 1) = 0

move(2, 0, 2) = 0

move(2, 2, 1) = 1



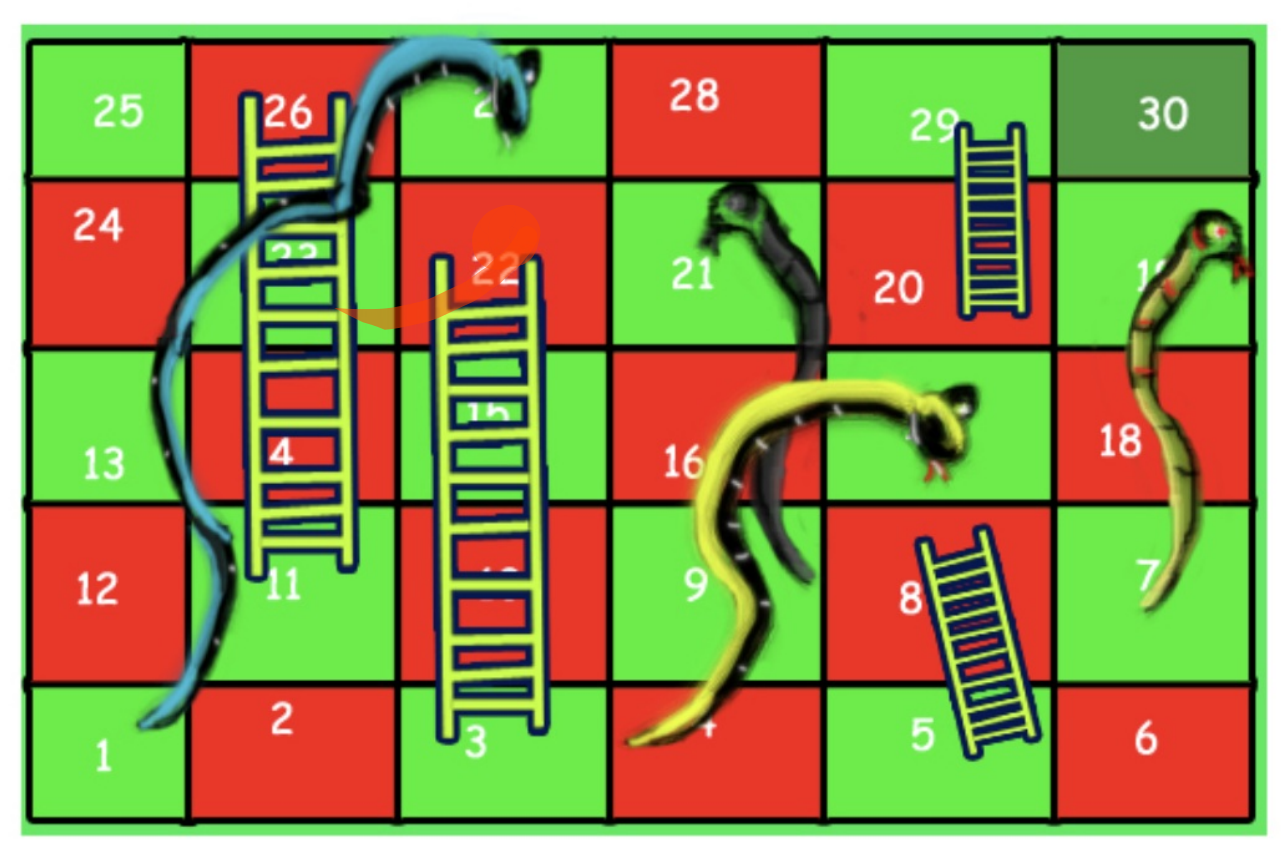
ideone.com/93jvf7

```java
public boolean isWinning(int row, int col){
    char ch = grid[row][col];

    boolean ans = true;
    for(int j = 0; j < grid.length; j++){
        if(grid[row][j] != ch) {
            ans = false;
            break;
        }
    }

    if(ans == true) return true;

    ans = true;
    for(int i = 0; i < grid.length; i++){
        if(grid[i][col] != ch) {
            ans = false;
            break;
        }
    }

    if(ans == true) return true;

    ans = true;
    for(int i = 0; i < grid.length; i++){
        if(grid[i][i] != ch) {
            ans = false;
            break;
        }
    }

    if(ans == true) return true;

    ans = true;
    for(int i = 0; i < grid.length; i++){
        if(grid[i][grid.length - 1 - i] != ch){
            ans = false;
            break;
        }
    }

    return ans;
}
```

0 references

```java
public int move(int row, int col, int player) {
    char ch = (player == 1) ? 'X' : 'O';

    grid[row][col] = ch;
    if(isWinning(row, col) == true){
        return player;
    } else return 0;
}
```

Snakes & Ladder



Whatsapp

D_1

Key: Array & Strings

value: 9

D_{10}

Array & String

value: 10

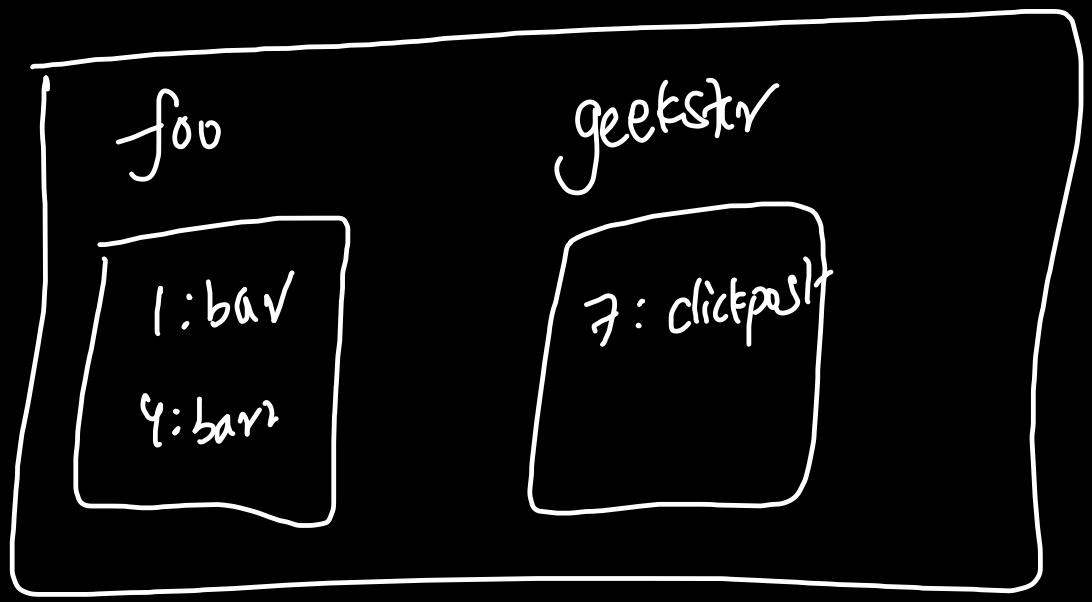
D_{20}

Algorithms

D_{30}

Advanced DSA

```
Input
["TimeMap", "set", "get", "get", "set", "get", "get"]
[[], ["foo", "bar", 1], ["foo", 1], ["foo", 3], ["foo", "bar2", 4], ["foo", 4], ["foo", 5]]
Output
[null, null, "bar", "bar", null, "bar2", "bar2"]
```



Hashmap < String,
TreeMap < Int, String>>
sort based on time

floor value

{ 1, 3, 4, 10, 15, 17, 19 } key = 15

```java
class TimeMap {
    HashMap<String, TreeMap<Integer, String>> db = new HashMap<>();

    public void set(String key, String value, int timestamp) {
        if(db.containsKey(key) == false)
            db.put(key, new TreeMap<>());
        db.get(key).put(timestamp, value);
    }

    public String get(String key, int timestamp) {
        if(db.containsKey(key) == false) return "";

        if(db.get(key).get(timestamp) != null)
            return db.get(key).get(timestamp);

        Integer floor = db.get(key).lowerKey(timestamp);
        System.out.println(floor);
        if(floor == null) return "";
        return db.get(key).get(floor);
    }
}
```

foo, 1 "" ""

foo bar 1

foo, 1 "bar"

foo bar2 3

foo, 4 "bar2"

foo bar3 5

foo, 0 "" ""