1) Coin Permutation - 1

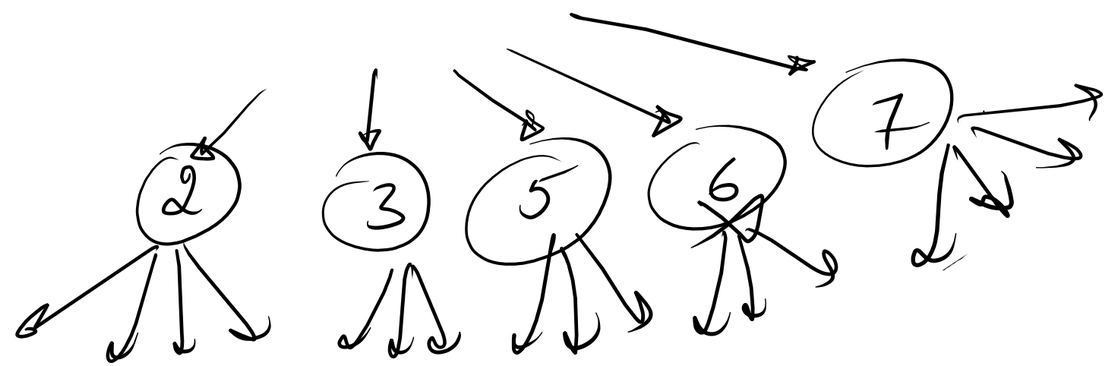
non-duplicates ✓

Input

Size → 5

coins → 2 3 5 6 7

amt → 12



Output

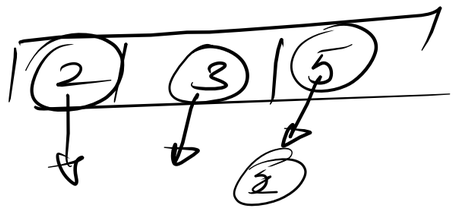
[2 3 7
 2 7 3
[3 2 7
 3 7 2
[5 7
[7 2 3
 7 3 2
 7 5

Ex 1) coins[] = {2, 3, 5}

amt = 5

op → [2, 3]
[3, 2]
[5]

How to track non-duplicates



boolean visted[]:

0	1	2
F	F	F

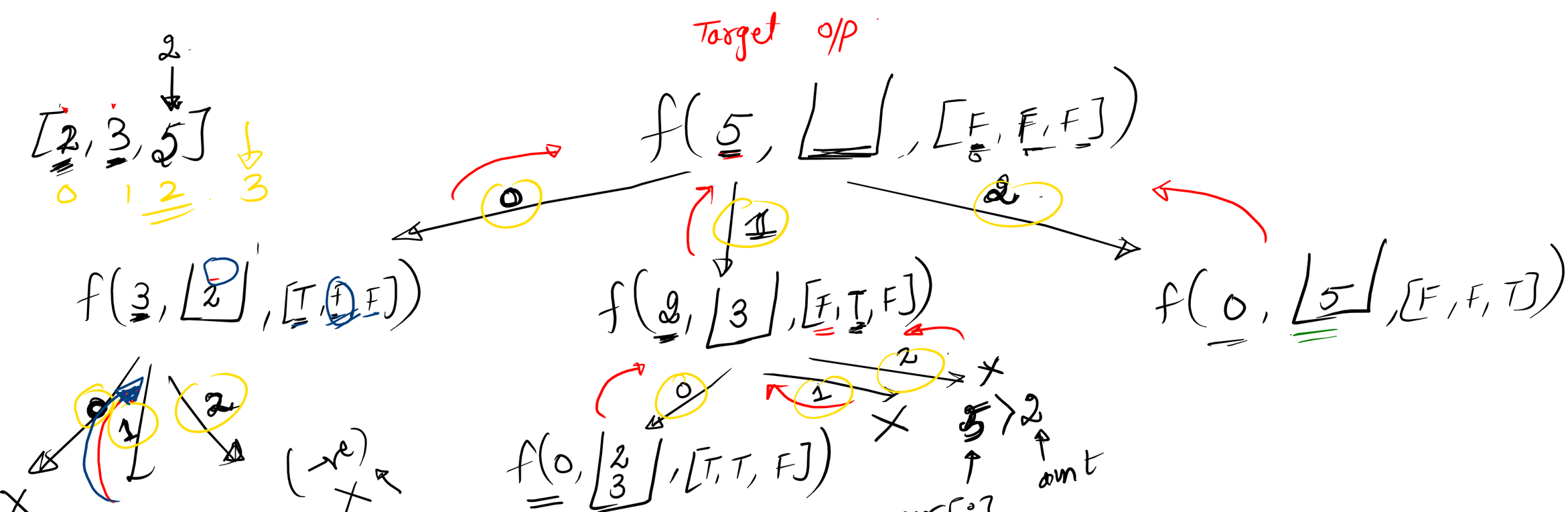
visted[0] = false.

↓

i did not use that element before.

Vice Versa.

Target o/p


[2, 3, 5]
0 1 2 3
f(5, [ ], [F, F, F])
f(3, [2], [T, T, F])
f(2, [3], [F, T, F])
f(0, [5], [F, F, T])
f(0, [2 3], [T, T, F])
f(0, [3 2], [T, T, F])
5 > 2
arr[i]
ans
5 > 3
arr[i]
amt
-ve
[2, 3]
[3, 2]
[5]
output

* Base case

```
if(amt == 0) {
        s.o.p(output);
        return;
}
```

arr | 2 | 3 | 5 |

vis | F | F | F |

op | |

```
* for(int i=0 ; i < arr.length ; i++){
        if( vis[i] == True ) continue;
```

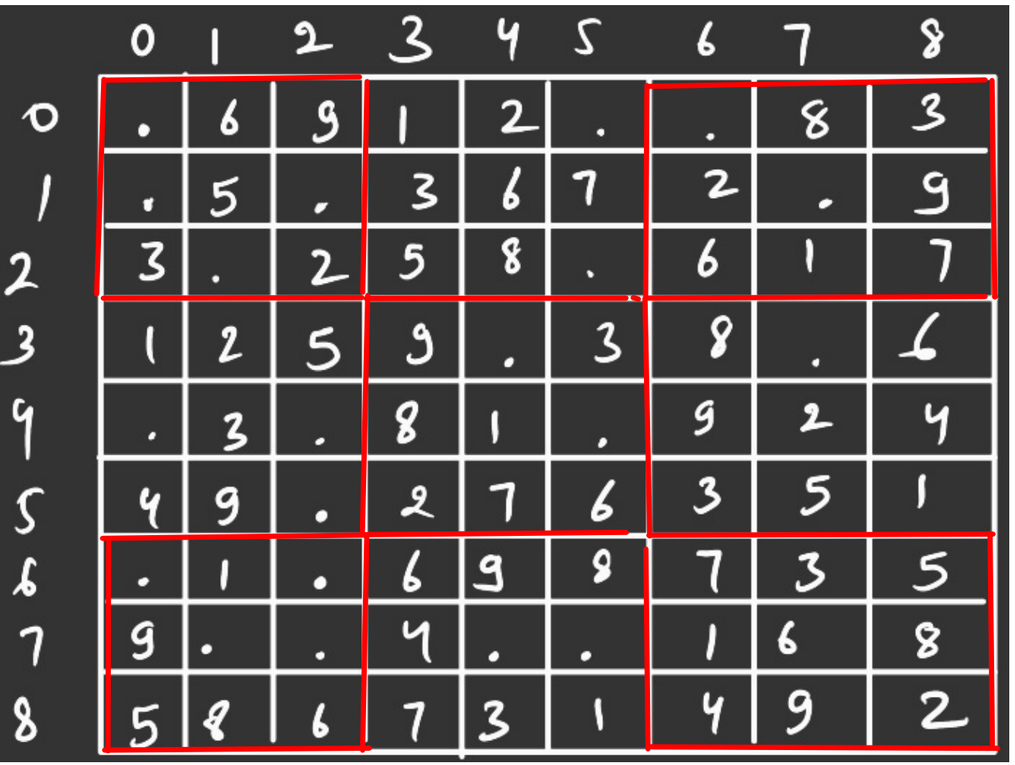
```java
public static void coinChange(int[] coins, int amt){
    ArrayList<Integer>output = new ArrayList<>();
    boolean[] visited = new boolean[coins.length];
    solve(coins,amt,output,visited);
}
public static void solve(int[] arr,int T,ArrayList<Integer>output,boolean[] visited){
   //base case
   if(T == 0 ){
     for(int nums : output){
        System.out.print(nums+" ");
     }
     System.out.println();
    //  return;
   }

   //process
   for(int i = 0 ; i<arr.length;i++){
     if(visited[i] == false && arr[i] <= T){
        visited[i] = true;
        output.add(arr[i]);
        solve(arr,T-arr[i],output,visited);
        output.remove(output.size() - 1); // element remove
        visited[i] = false;
     }
   }
}
```

Q-2 Coin Permutation-2.

↳ Done

Sudoku Solver



Sudoku rules:

- Each of the digits 1-9 must occur exactly once in each row.
- Each of the digits 1-9 must occur exactly once in each column.
- Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.
- The '.' character indicates empty cells.

1-9 digit . → True .

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	.	.	8	3
1	.	5	.	3	6	7	2	.	9
2	3	.	2	5	8	.	6	1	7
3	1	2	5	9	.	3	8	.	6
4	.	3	.	8	1	.	9	2	4
5	4	9	.	2	7	6	3	5	1
6	.	1	.	6	9	8	7	3	5
7	9	.	.	4	.	.	1	6	8
8	5	8	6	7	3	1	4	9	2

col = col+1

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	4	.	8	3
1	.	5	.	3	6	7	2	.	9
2	3	.	2	5	8	.	6	1	7
3	1	2	5	9	.	3	8	.	6
4	.	3	.	8	1	.	9	2	4
5	4	9	.	2	7	6	3	5	1
6	.	1	.	6	9	8	7	3	5
7	9	.	.	4	.	.	1	6	8
8	5	8	6	7	3	1	4	9	2

return back to c0

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	4	5	8	3
1	.	5	.	3	6	7	2	.	9
2	3	.	2	5	8	.	6	1	7
3	1	2	5	9	.	3	8	.	6
4	.	3	.	8	1	.	9	2	4
5	4	9	.	2	7	6	3	5	1
6	.	1	.	6	9	8	7	3	5
7	9	.	.	4	.	.	1	6	8
8	5	8	6	7	3	1	4	9	2

True

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	4	5	8	3
1	8	5	1	3	6	7	2	4	9
2	3	4	2	5	8	9	6	1	7
3	1	2	5	9	4	3	8	7	6
4	6	3	7	8	1	5	9	2	4
5	4	9	8	2	7	6	3	5	1
6	2	1	4	6	9	8	7	3	5
7	9	7	3	4	5	2	1	6	8
8	5	8	6	7	3	1	4	9	2

True

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	4	5	8	3
1	8	5	1	3	6	7	2	.	9
2	3	.	2	5	8	.	6	1	7
3	1	2	5	9	.	3	8	.	6
4	.	3	.	8	1	.	9	2	4
5	4	9	.	2	7	6	3	5	1
6	.	1	.	6	9	8	7	3	5
7	9	.	.	4	.	.	1	6	8
8	5	8	6	7	3	1	4	9	2

True

	0	1	2	3	4	5	6	7	8
0	7	6	9	1	2	4	5	8	3
1	8	5	.	3	6	7	2	.	9
2	3	.	2	5	8	.	6	1	7
3	1	2	5	9	.	3	8	.	6
4	.	3	.	8	1	.	9	2	4
5	4	9	.	2	7	6	3	5	1
6	.	1	.	6	9	8	7	3	5
7	9	.	.	4	.	.	1	6	8
8	5	8	6	7	3	1	4	9	2

return True

board → solution exists ✓ → True.

↓ solution Not Exists → False.

f(board, 0, 0)

col row col

* - Base Case.

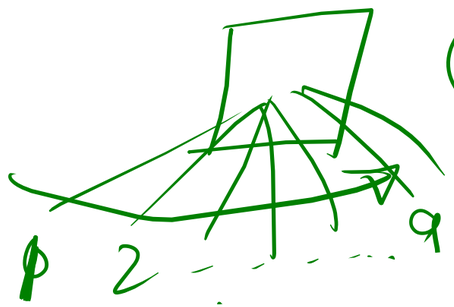
```
if(row == board.length)   return True;
```

row & col update -

```
else {
    row = row;
    col = col+1;
}
```

```
if(col == board.length){
    row = row+1;
    col = 0;
}
```

process



(1-9) digit.

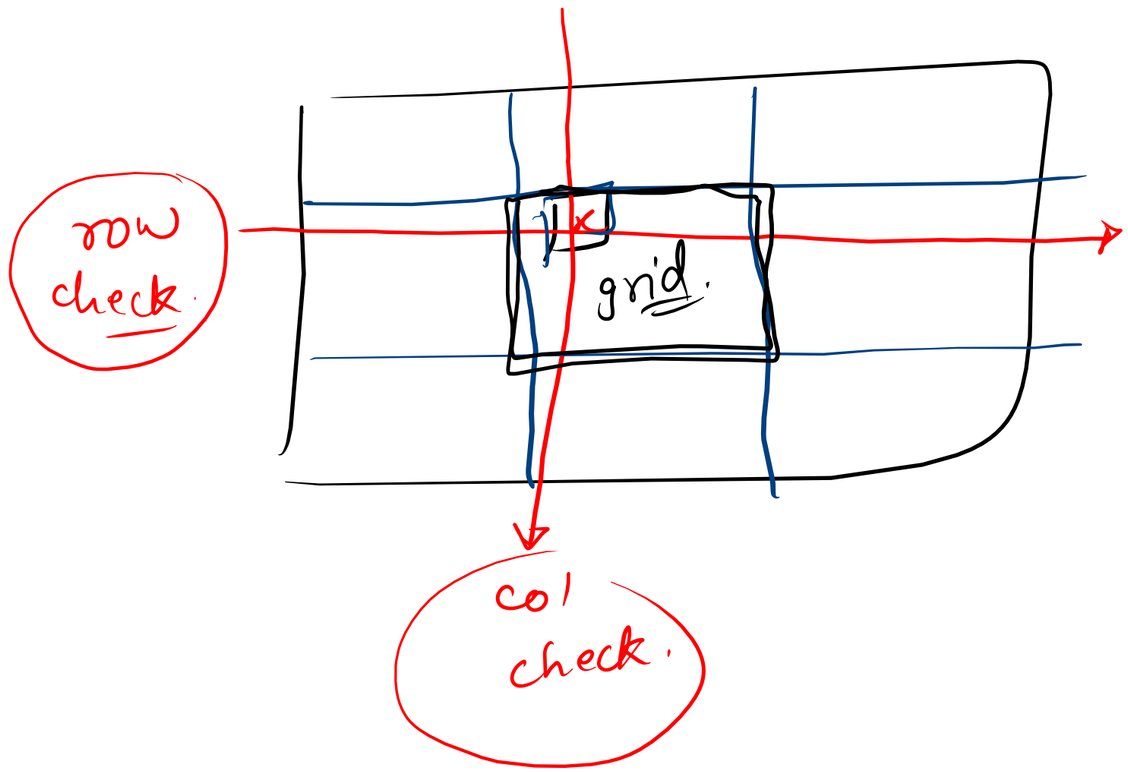
```
for(int i=1 ; i<=9 ; i++){
    if(isSafe(board, r, c, i){
        board[r][c] = i;        // pick
        solve(board, row, col); // func. call
        board[r][c] = '0';
    }
```

* isSafe (board, row, col, i)

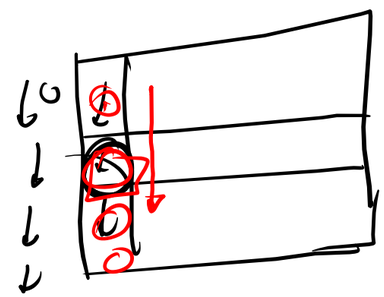
// row check

// col check

// grid check (3X3)



* Row isSafe(board, row, col, i) 2 i=2



```
for(int n2 = 0; n2 < bod.length; n2++){
    if(board[n2][col] == i) return false;
}
```

2.

* col isSafe (board, row, col, i)

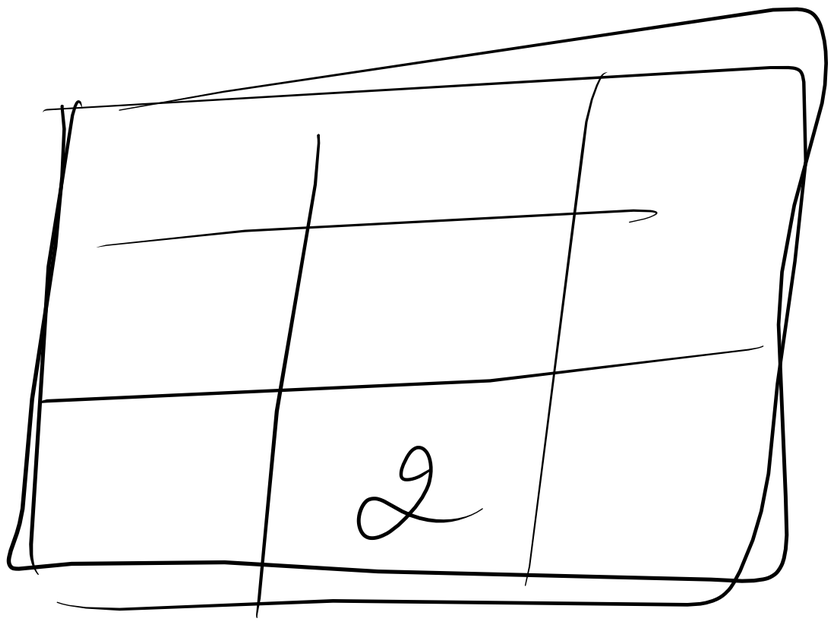
```
for (int n2 = 0;  n2 < board.length ;  n2++) {
    if ( board[row][n2] == i ) return false;
}
```

board[row] → 2

i → 2

* 3×3 Grid

isSafe (board, row, col, i)




```
sr
for( ↓  ↓ ) {
        sc
        ↓
    for( ↓  ↓ )
        if( b[sr+i][sc+j] == 2)
            return false
}
```

How to find out the starting row & starting col of a particular cell.

r = 0 c = 5

sr = r/3 ⇒ 0

sc = c/3 ⇒ 1 × 3 ⇒ 3

r = 5, c = 7

sr = (5/3) ⇒ 1

1 × 3 ⇒ 3

sc = (7/3) × 3 ⇒ 2 × 3 ⇒ 6

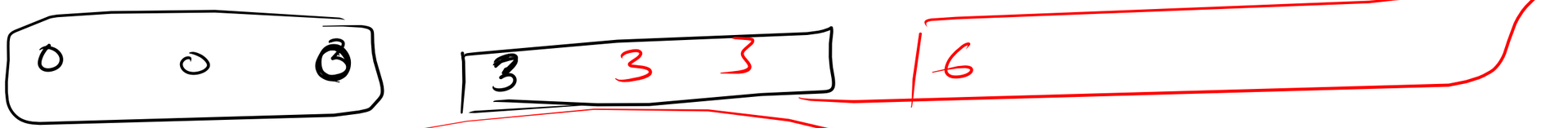
M1 M2 M3

0 1 2 3 4 5 6 7 8

6,C

$\frac{0}{3}$ $\frac{1}{3}$ $\frac{2}{3}$ $\frac{3}{3}$ $\frac{4}{3}$ $\frac{5}{3}$ $\frac{6}{3}$

0x3 0x3 0x3 1x3 1x3 1x3 2x3



SR, SC

$Sx = (x/3) \times 3$;

$Sc = (c/3) \times 3$;

int → char

1 → '1'

(1 + '0')

(char) (1 + 48)

char(49) → '1'

int → char

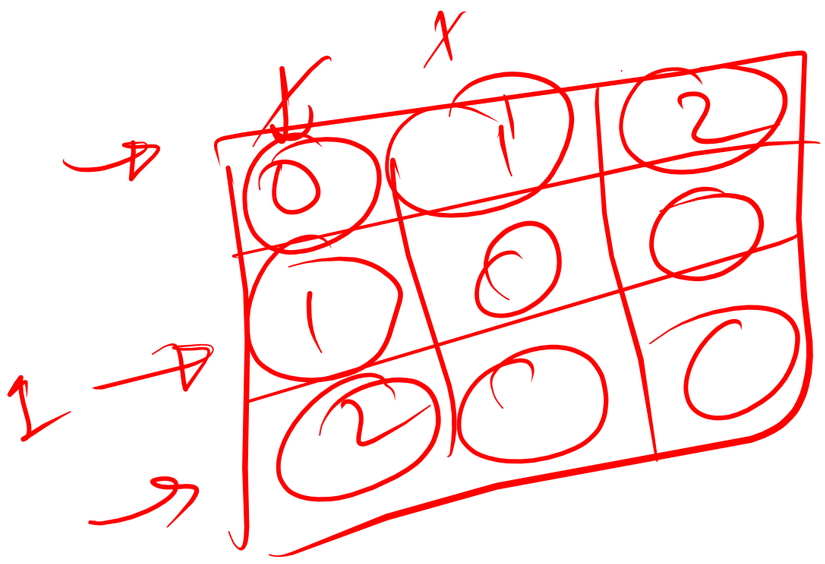
2 → '2'

2 + '0'

2 + 48

char(50) → '2'

```
for(int sr = 0 ; sr < 3; sr++){
    for(in sc = 0 ; sc < 3 ; sc++){
        if(board[sr][sc] == (char)(number+'0')) return false;
    }
}
```



```java
public static boolean isValidSudoku(char[][] board) {
  // your code here
  return solve(board,0,0);
}
public static boolean solve(char[][]board,int row , int col){
  //base case
  if(row == board.length){
      return true;
  }
  //update row , col
  int nrow = 0 ;
  int ncol = 0 ;

  if(col == board.length-1){
      nrow = row + 1 ;
      ncol = 0;
  }
  else{
      nrow = row ;
      ncol = col + 1 ;
  }

  //process
  if(board[row][col] != '.'){
      return solve(board,nrow,ncol);
  }
  else{
      for(int number = 1 ; number<=9 ; number++){
          if(isSafe(board,row,col,number)){
          board[row][col]= (char)(number+'0') ;
          if(solve(board,nrow,ncol)) return true ;
          board[row][col] ='.';
          }
```

```java
          }
      }
    }
   return false;
  }

  public static boolean isSafe(char[][]board,int row , int col,int number){


    //row check
    for(int r = 0 ; r < board.length ; r++){
        if(board[r][col] == (char)(number+'0')) return false;
    }


    //col check
    for(int c = 0 ; c < board.length ; c++){
        if(board[row][c] == (char)(number+'0')) return false;
    }

    //3 * 3 GRID
    int sr = (row/3) * 3 ;
    int sc = (col/3) * 3 ;

    for(int i = sr ; i< sr +3; i++){
        for(int j = sc ; j < sc + 3 ; j++){
            if(board[i][j] == (char)(number+'0')) return false;
        }
    }
    return true ;
  }
```

What is OOP?

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Class: A blueprint/template for creating objects.
It defines properties (variables) and behaviors (methods).

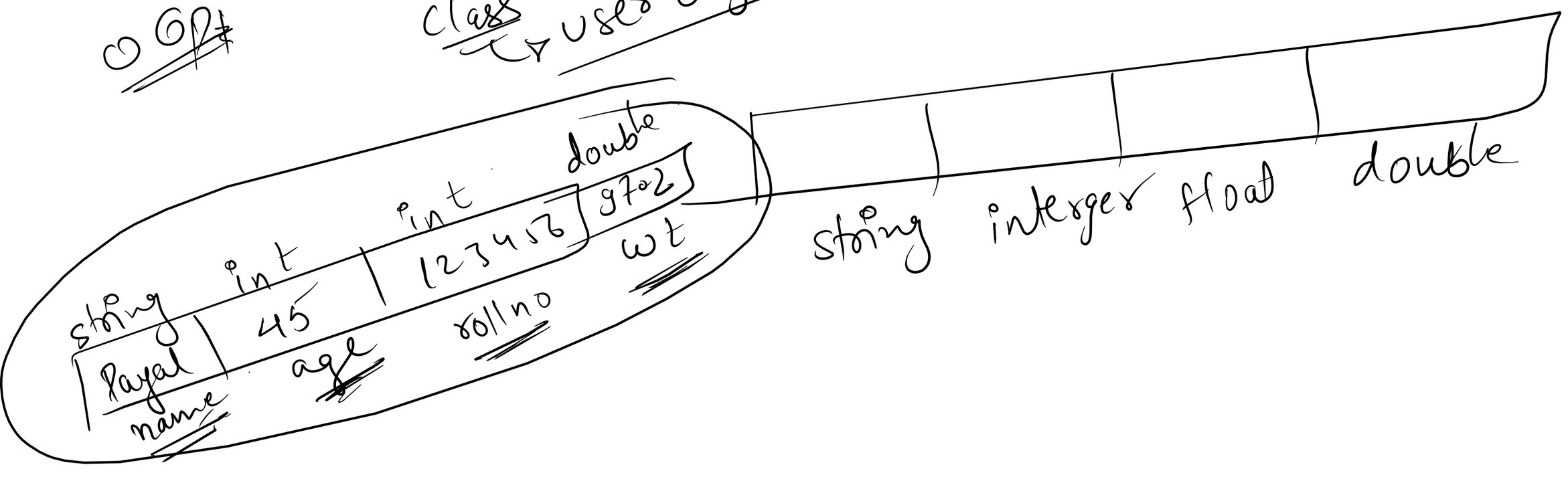
Object: A real-world entity created using a class. It holds actual values for the properties.

*

arr=[| | | |]

OOPs

class ↳ user define datatype



A constructor is a special method that:
Has the same name as the class.Is automatically called when an object is created.
Is used to initialize objects.

Types of Constructors:
1. Default Constructor – No parameters.
2. Parameterized Constructor – Takes arguments to initialize the object.

* 4-Pillers of OOPs

① Polymorphism.

② Encapsulation

③ Inheritance.

④ Abstraction

1.) Polymorphism

many form.

* Ex.
* Advantage of oops
* Data binding & Data hiding
* Diamond Problem.
* this Keyword.

this Keyword

this is a reference variable in Java that refers to the current object.
Uses of this keyword:

1. To refer to current class instance variable.
2. To call current class methods/constructors.
3. To pass the current object as an argument.