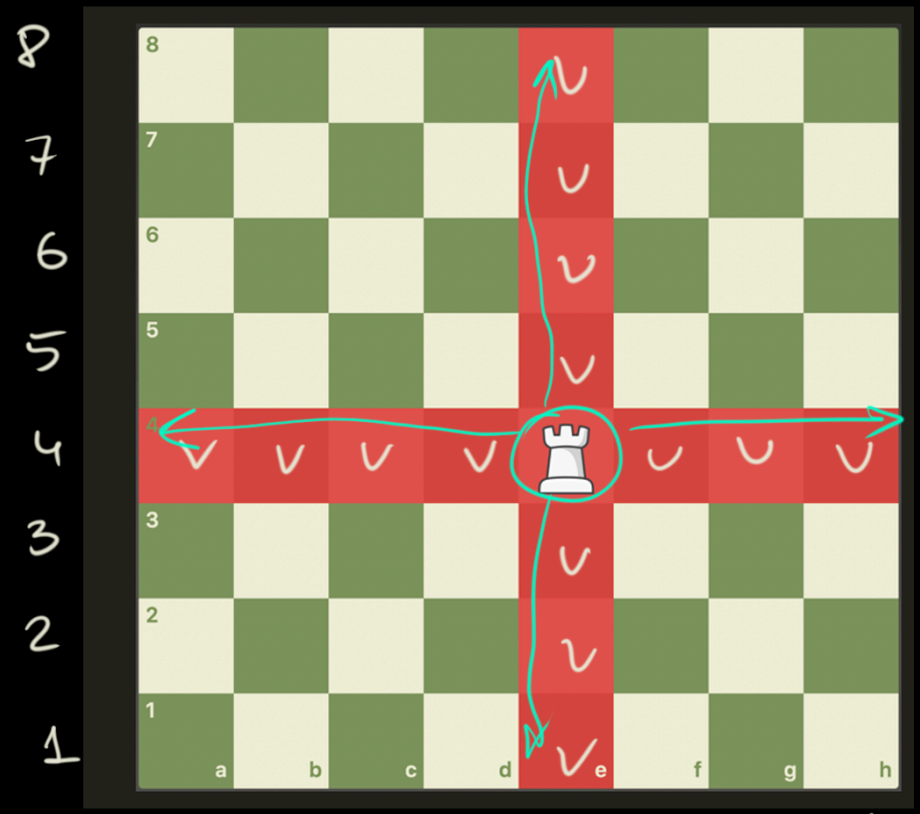


a b c d e f g h

possibleMoves("e4") →

a4, b4, c4, d4
f4, g4, h4

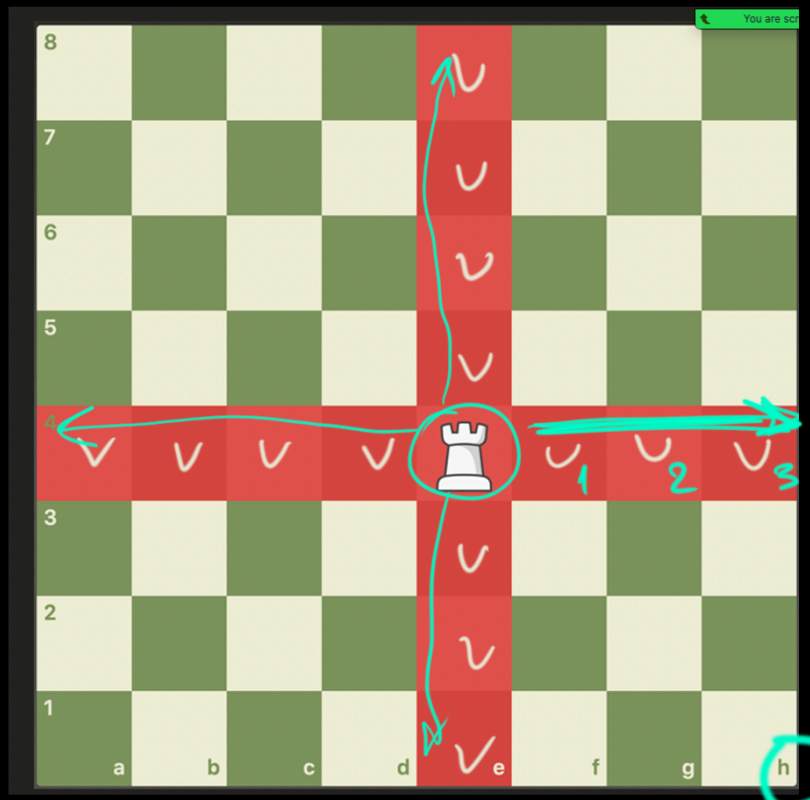
List<String> = e8, e7, e6, e5, e3, e2, e1,

```
public List<String> possibleMoves(String move){
    List<String> allMoves = new ArrayList<>();
    String[] numbers = {"1","2","3","4","5","6","7","8"};
    String[] letters = {"a","b","c","d","e","f","g","h"};

    for (String letter : letters) {
        for (String number : numbers) {
            String createdMove = letter.concat(number);
            allMoves.add(createdMove);
        }
    }
    return  allMoves
            .stream()
            .filter(c-> (c.charAt(0) == move.charAt(0) || c.charAt(1) == move.charAt(1)) &&
!c.equals(move))
            .collect(Collectors.toList());
}
```

```
  for (int i = 1; i <= 8; i++) {
      String colPos = positionStr + i;
      char pos = (char) (i + 96);
```

```
if(first != i)
    result.add("" + row.charAt(i) + (char)second);
if(second - '0' != i+1)
    result.add("" + row.charAt(first) + (i + 1));
```




```
public static IntStream Ito8(){
  return IntStream.rangeClosed(1, 8);
}
```

4 5 6 7 8

```
return Stream.of(right(p0), left(p0), up(p0), down(p0)) Stream<Stream<Pos>>
    .flatMap(a -> a) Stream<Pos>
    .filter(p -> p.isOnBoard());
```

right left up down

```
[f4, g4, h4, d4, c4, b4, a4, e5, e6, e7, e8, e3, e2, e1]
```

```
public int min(int[] xs) {
    ....
}
```

can not be implemented

because there is NO VALUE can be returned
in case of EMPTY ARRAY

public Int|Nothing min(....)

```
public int min(int[] xs) throws NoSuchElementException {
  throw new NoSuchElementException();
}
```

```
public Optional<Integer> min1(List<Integer> xs)
```

Integer | Empty

Stream<A>

.map(f: A => B) Stream<B>

.map(A => Stream<B>) Stream<Stream<B>>

.flatMap(A => Stream<B>) Stream<B>

.map(□ → △) △ △ △

.flatMap(□ → Stream(△, △, △)) → △△△△△△△△△

Stream<Stream<A>> .flatMap(x → x) → Stream<A>

```java
// all elements > 0
public boolean something1(int[] xs) {
  for (int x: xs) {
    if (!(x > 0)) return false;
  }
  return true;
}

// NONE elements > 0
public boolean something2(int[] xs) {
  for (int x: xs) {
    if (x > 0) return false;
  }
  return true;
}

// AT least one element > 0
public boolean something3(int[] xs) {
  for (int x: xs) {
    if (x > 0) return true;
  }
  return false;
}
```

```java
// all elements > 0
public boolean something1(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (!p.test(x)) return false;
  }
  return true;
}

// NONE elements > 0
public boolean something2(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (p.test(x)) return false;
  }
  return true;
}

// AT least one element > 0
public boolean something3(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (p.test(x)) return true;
  }
  return false;
}
```

```java
// all elements > 0
public boolean something1(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (!p.test(x)) return false;
  }
  return true;
}

// NONE elements > 0
public boolean something2(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (p.test(x)) return false;
  }
  return true;
}

// AT least one element > 0
public boolean something3(int[] xs, Predicate<Integer> p) {
  for (int x: xs) {
    if (p.test(x)) return true;
  }
  return false;
}
```

```java
IntStream.range(1, 10)
.noneMatch(x -> x < 0)

IntStream.range(1, 10)
.allMatch(x -> x > 0)

IntStream.range(1, 10)
.anyMatch(x -> x > 0)

IntStream.range(1, 10)
.anyMatch(x -> x < 0)
```