Map

Array / Matrix

- Ein typisches Muster, mit dem wir prüfen, ob sich der Index innerhalb der Grenzen vom 2D-Array befindet.
- In der Programmierung spricht man dabei von der Matrix

Definition

```
row >= 0 && row < map.length
```

- Hiermit wird geprüft, ob der Zeilenindex größer oder gleich 0 (da die Array-Indizes bei 0 beginnen) und kleiner als die Länge des map Array. Dadurch wird sichergestellt, dass der Zeilenindex für das angegebene Array gültig ist.

```
column >= 0 && column < map[row].length
```

- Hiermit wird geprüft, ob der Spaltenindex größer oder gleich 0 und kleiner als die Länge der Zeile bei der angegebenen ist row Index. Dadurch wird sichergestellt, dass der Spaltenindex für die angegebene Zeile gültig ist.

```
&&
```

- Das && Der Bediener wird verwendet, um sicherzustellen, dass beide Bedingungen erfüllt sind. Wenn eine der beiden Bedingungen falsch ist, ist der gesamte Ausdruck falsch, was darauf hinweist, dass die angegebenen Zeilen- und Spaltenindizes nicht innerhalb der Grenzen des map Array.

Allgmemein

- In Java haben Arrays eine feste Größe, die beim Erstellen bestimmt wird. Für ein 2D-Array bedeutet dies, dass jede Zeile eine andere Anzahl von Spalten haben kann, die Anzahl der Zeilen jedoch festgelegt ist. Um sicherzustellen, dass wir auf ein gültiges Element im Array zugreifen, müssen wir daher überprüfen, ob sich sowohl der Zeilen- als auch der Spaltenindizes innerhalb der Grenzen des Arrays befinden.

- Der && Operator ist ein logisjava cher UND Operator, was bedeutet, dass beide Bedingungen wahr sein müssen, damit der gesamte Ausdruck wahr ist. In diesem Fall überprüfen wir, ob der Zeilenindex größer oder gleich 0 und kleiner als die Länge des map Array und dass der Spaltenindex größer oder gleich 0 und kleiner als die Länge der Zeile bei der angegebenen ist row Index.

```
int[][] map = new int[10][10];
    for (int row = 0; row < map.length; row++) {
        for (int column = 0; column < map[row].length; column++) {
            if (row >= 0 && row < map.length && column >= 0 && column <
map[row].length) {
                // Access the element at the given row and column indices
            int value = map[row][column];
                // Do something with the value
            }
        }
    }
```