Spicker **M411 Datenstrukturen & Algorithmen** Ceronio Justin

Arrays

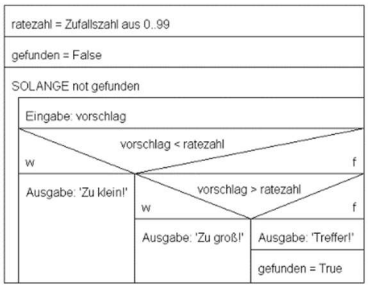
Deklarieren & Initialisieren: int[] a = new int[2]; oder int[] x = {1,2};
Zugriff: a[1] = 2; **Array Iterieren:**

```
for (i = 0; i < a.length; i++)
              x = a[i];
for (int i : ar)
              x = i;
```

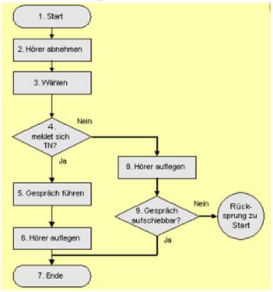
2D Array iterieren:

```
for (int row = 0; row < a.length; row++)
    for (int col = 0; col < a[row].length; col++)
       a[row][col] = row * col;
```

Struktogramm



Flussdiagramm



Sortieralgorithmen

Insertionsort:

```
for (int j = 1; j < array.length; j++) {
    int current = array[j];
    int i = j-1;
    while ((i > -1) && (array[i] > current)) {
      array[i+1] = array[i];
      i--;
    }
    array[i+1] = current;
  }
```

Komplexität: $O(n^2)$ n=>Grösse des Arrays

Selectionsort:

```
for(int i = 0; i < A.length-1; i++){
    int minIndex = i;
    for(int j = i + 1; j < A.length; j++){
        if(A[j] < A[minIndex]){
            minIndex = j;
        }
    }

    int temp = A[minIndex];
    A[minIndex] = A[i];
    A[i] = temp;
}

return A;
```

Komplexität: $O(n^2)$

Bubblesort:

```
for (int i = ar.length - 1; i > 0; i--) {
    for (int j = 0; j < i; j++) {
        if (ar[j] > ar[j + 1]) {
            temp = ar[j];
            ar[j] = ar[j + 1];
            ar[j + 1] = temp;
        }
    }
}
```

Komplexität: $O(n^2)$

```
public static void collatz_pro(long n) {
    // Process
    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        System.out.print(n + ", ");
    }
    System.out.println("Fertig!");
}
```

```
public class AB411_04_PerformanceTesting {
    public static void main(String[] args) {
        int [] liste = new int[100000];
        for (int i=0; i<liste.length; i++)
            liste[i] = (int)(100*Math.random());
        long startCloneTime = System.currentTimeMillis();
        for (int j=0; j<10000;j++) {
            int[] cloneListe = (int[]) liste.clone();}
        long stopCloneTime = System.currentTimeMillis();
        long startCopyTime = System.currentTimeMillis();
        for (int j=0; j<10000;j++) {
            int[] copyListe = new int[liste.length];
            for (int n=0; n<liste.length;n++) {
                copyListe[n] = liste[n];  } }
        long stopCopyTime = System.currentTimeMillis();
        long cloneTime = stopCloneTime - startCloneTime;
        long copyTime = stopCopyTime - startCopyTime;
        System.out.println("Dauer (clone): " + cloneTime + "ms");
        System.out.println("Dauer (elementweise Kopieren): " + copyTime + "ms");
    }
```

```
public class Archive {

    final static int DIM1 = 12;
    final static int DIM2 = 12;

    public static void main(String[] args) {
        boolean[][] welt = initWelt();
        System.out.println("Startkonstellation");
        zeigeWelt(welt);
        for (int i = 0; i <= 100; i++) {
            welt = wendeRegelnAn(welt);
            System.out.println("Generation: " + i);
            zeigeWelt(welt);
        }
    }

    private static boolean[][] initWelt() {
        boolean[][] welt = new boolean[DIM1][DIM2];
        for (int y = 1; y < DIM2 - 1; y++)
            for (int x = 1; x < DIM1 - 1; x++)
                welt[x][y] = Math.random() > 0.4;
        return welt;
    }

    public static int anzNachbarn(boolean[][] welt, int x, int y) {
        int ret = 0;
        for (int i = x - 1; i <= x + 1; ++i)
            for (int j = y - 1; j <= y + 1; ++j)
                if (welt[i][j])
                    ret += 1;
        // einen Nachbarn zuviel gezählt?
        if (welt[x][y])
            ret -= 1;
        return ret;
    }
```

```
public static void zeigeWelt(boolean[][] welt) {
    for (int y = 1; y < DIM2 - 1; y++) {
        for (int x = 1; x < DIM1 - 1; x++) {
            if (welt[x][y])
                System.out.print("x ");
            else
                System.out.print("- ");
        }
        System.out.println();
    }
}

public static boolean[][] wendeRegelnAn(boolean[][] welt) {
    boolean[][] welt_neu = new boolean[DIM1][DIM2];
    int nachbarn;
    for (int y = 1; y < DIM2 - 1; y++)
        for (int x = 1; x < DIM1 - 1; x++) {
            nachbarn = anzNachbarn(welt, x, y);
            if (welt[x][y]) {
                if ((nachbarn < 2) || (nachbarn > 3))
                    welt_neu[x][y] = false;
                if ((nachbarn == 2) || (nachbarn == 3))
                    welt_neu[x][y] = true;
            } else {
                if (nachbarn == 3)
                    welt_neu[x][y] = true;
            }
        }
    return welt_neu;
}
```

```
private static long fib(int x) {
    if (x<=2)
        return 1;
    else
        return fib( x: x-1) + fib( x: x-2);
}
```

```
static void collatz_rek(int n){
    if( n == 1 ) {
        return;
    }
    if( n % 2 == 0){
        collatz_rek( n: n/2);
    }else {
        collatz_rek( n: 3*n+1);
    }
}
```