Projekt RAM

Inhalt

[Aufgabe 1 – Erste Gedanken über Fibonacci 2](#_Toc151091551)

[Wie lauten die nächsten drei Zahlen nach 21 in der Fibonacci-Folge? 2](#_Toc151091552)

[Beschreibe die Gesetzmässigkeit in der Zahlenfolge. 2](#_Toc151091553)

[Aufgabe 2 – Arithmetische Operationen in Java 3](#_Toc151091554)

[Zu verwendende Funktionen (Methoden) mit PAPs 3](#_Toc151091555)

[Conditional Statements 3](#_Toc151091556)

[Loops 4](#_Toc151091557)

[Erstelle eine neue Klasse Operation (Operation.java) in einem neuen Package mit dem Namen ram. 4](#_Toc151091558)

[Initialisiere die int Variablen x mit 14 und y mit 42 und gebe die Resultate von Addition, Subtraktion, Multiplikation und Division auf der Konsole aus. 4](#_Toc151091559)

[Resultate 5](#_Toc151091560)

[Implementiere die Multiplikation, indem du nur Addition und eine For Schleife verwendest. Kommentiere deine Lösung. 5](#_Toc151091561)

[Aufgabe 3 – Fibonacci in Java 6](#_Toc151091562)

[Implementiere ein Programm (Fibonacci.java), welches die n-te Fibonacci Zahl berechnet und auf die Konsole ausgibt. 6](#_Toc151091563)

[Wie lautet die 34-te Fibonacci Zahl? 6](#_Toc151091564)

[Resultat 7](#_Toc151091565)

[Ergänze das Programm so, dass das Verhältnis zwischen der 34-ten und der 33-ten Fibonacci Zahl berechnet und ausgegeben wird. 7](#_Toc151091566)

[Resultat 8](#_Toc151091567)

[Weisst du wie dieses Verhältnis auch genannt wird? 8](#_Toc151091568)

[Aufgabe 4 - RAL-Programmierung: Subtraktion 9](#_Toc151091569)

[Eigene erste Gedanken zu RAM 9](#_Toc151091570)

[Kurze Erklärung der RAM 9](#_Toc151091571)

[Erklärung eines RAM am Beispiel Addition 9](#_Toc151091572)

[Anzuwendende Befehle 10](#_Toc151091573)

[Implementiere die Subtraktion zweier ganzen Zahlen mit RAL (z.B. 7 – 3 = 4) 11](#_Toc151091574)

[Aufgabe 5 - RAL-Programmierung: Multiplizierung 11](#_Toc151091575)

[Implementiere die Multiplikation zweier ganzen Zahlen mit RAL (Orientiere dich an den Code von Aufgabe 2) 11](#_Toc151091576)

[Erster Loop 12](#_Toc151091577)

[Zweiter Loop 13](#_Toc151091578)

[Überblick 13](#_Toc151091579)

[Aufgabe 6 - RAL-Programmierung: Fibonacci-Folge 14](#_Toc151091580)

[Implementiere die Berechnung der n-ten Fibonacci Zahl mit RAL (Orientiere dich an den Code von Aufgabe 3) 14](#_Toc151091581)

[Erster Loop 16](#_Toc151091582)

[Zweiter Loop 17](#_Toc151091583)

[Überblick 18](#_Toc151091584)

[Zusatzaufgabe – Bubble-Sort 19](#_Toc151091585)

[Prinzip des Bubble-Sort erklärt 19](#_Toc151091586)

[Implementiere den Bubble-Sort in Java und in RAL. 19](#_Toc151091587)

[Java 19](#_Toc151091588)

[RAL 20](#_Toc151091589)

# Aufgabe 1 – Erste Gedanken über Fibonacci

## Wie lauten die nächsten drei Zahlen nach 21 in der Fibonacci-Folge?

Die folgende nächsten drei Zahlen, die nach dieser Zahlenfolge kommen (1,1,2,3,5,8,13,21) sind **34, 55, 89**.

## Beschreibe die Gesetzmässigkeit in der Zahlenfolge.

Auf dem ersten Blick und den ersten Eindruck merkt man, dass die Zahlen in irgendeiner regelmässigen Abhängigkeit zu der vorherigen Zahl stehen, dem Fall gibt es eine Gesetzmässigkeit und deshalb kann die Fibonacci-Folge sogar anhand einer Formel beschrieben werden.

Nach ein bisschen längerem Nachdenken oder sogar nach recherchieren merkt man, dass die Fibonacci-Folge ist eine Zahlenfolge ist, bei der jede Zahl die Summe der beiden vorherigen Zahlen bildet.

Durch folgendes Beispiel wird es ersichtlich:

1, 1, 2, 3, 5 -> 3 + 5 = 8

1, 2, 3, 5, 8 -> 5 + 8 = 13

2, 3, 5, 8, 13 -> 8 + 13 = 21

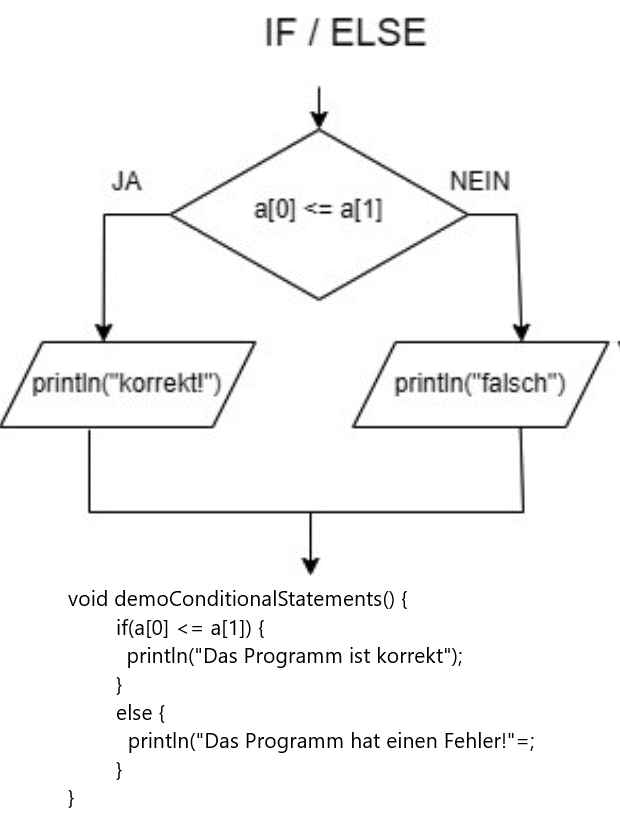
… und immer so weiter (Gesetzmässigkeit).

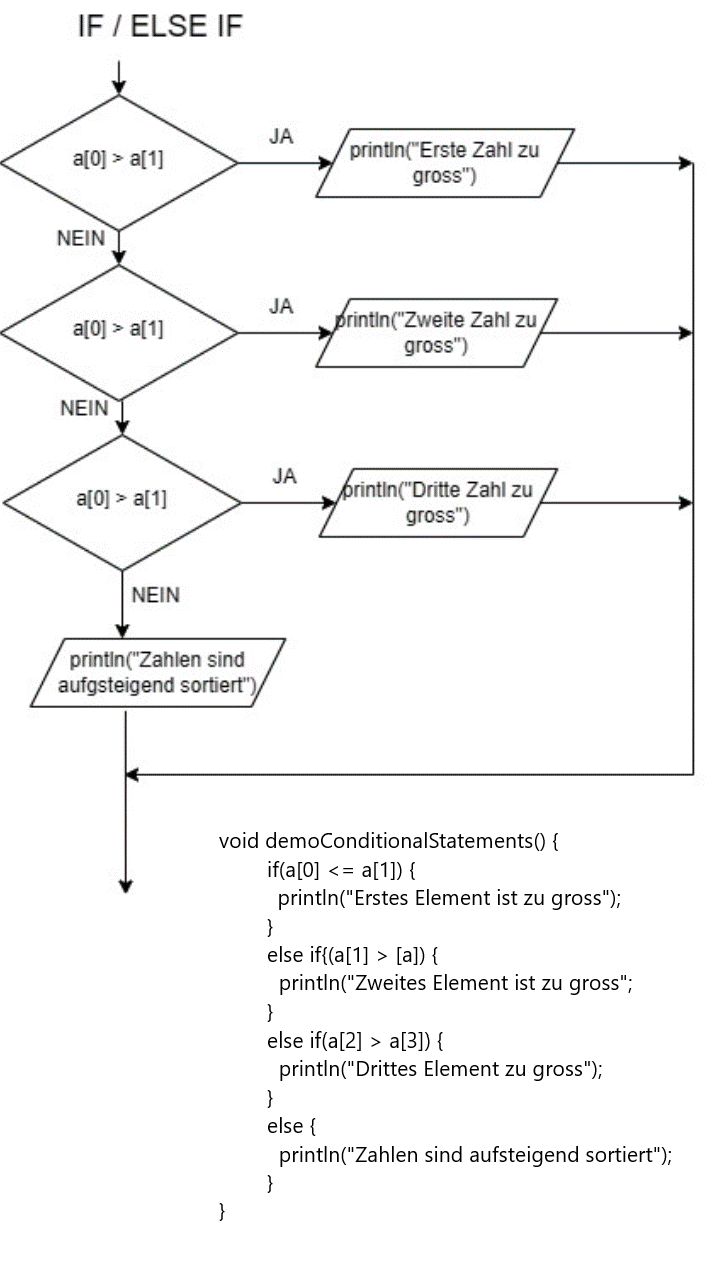
# Aufgabe 2 – Arithmetische Operationen in Java

## Zu verwendende Funktionen (Methoden) mit PAPs

Zurzeit arbeiten wir jediglich mit den konditionalen Statements (if-, if-else, if-else-if) und mit den Schleifen (for-, while- und do-while) um unser Programm zu schreiben.

### Conditional Statements





### Loops

## Erstelle eine neue Klasse Operation (Operation.java) in einem neuen Package mit dem Namen ram.

Um eine neue Klasse namens "Operation" in einem Package namens "ram" in Java zu erstellen, muss man die folgenden Schritte befolgen:

1. Erstelle einen neuen Ordner für dein Projekt und wähle es als Java Project aus. Dies könnte zum Beispiel "Aufgabe\_2" heissen.
2. Innerhalb dieses Ordners erstelle ein neues Package namens "ram". Dies wird das Package für deine Klasse sein.
3. Innerhalb des "ram"-Ordners erstelle eine Class namens "Operation.java".
4. Öffne die "Operation.java"-Datei mit einem Texteditor und füge den folgenden Code ein.

## Initialisiere die int Variablen x mit 14 und y mit 42 und gebe die Resultate von Addition, Subtraktion, Multiplikation und Division auf der Konsole aus.

**package** ram;

**public** **class** Operation {

**public** **static** **void** main(String[] args) {

// Initialisiere die int Variablen x mit 14 und y mit 42

**int** x = 14;

**int** y = 42;

// Addition

**int** addition = x + y;

System.***out***.println("Addition: " + addition);

// Subtraktion

**int** subtraktion = x - y;

System.***out***.println("Subtraktion: " + subtraktion);

// Multiplikation

**int** multiplikation = x \* y;

System.***out***.println("Multiplikation: " + multiplikation);

// Division (Achtung: Überprüfe auf Division durch Null)

**if** (y != 0) {

**int** division = x / y;

System.***out***.println("Division: " + division);

} **else** {

System.***out***.println("Division durch Null ist nicht erlaubt.");

}

}

}

### Resultate

Addition: 56

Subtraktion: -28

Multiplikation: 588

Division: 0

## Implementiere die Multiplikation, indem du nur Addition und eine For Schleife verwendest. Kommentiere deine Lösung.

**package** ram;

**public** **class** Multiplikation {

**public** **static** **void** main(String[] args) {

// Initialisiere die int Variablen x mit 14 und y mit 42

**int** x = 14;

**int** y = 42;

// Rufe die Multiplikationsmethode auf

**int** ergebnis = *multipliziere*(x, y);

// Gib das Ergebnis auf der Konsole aus

System.***out***.println("Multiplikation: " + ergebnis);

}

// Multiplikationsmethode mit Addition und For-Schleife

**static** **int** multipliziere(**int** a, **int** b) {

// Initialisiere das Ergebnis als 0

**int** ergebnis = 0;

// Führe die Addition b Mal durch

**for** (**int** i = 0; i < b; i++) {

// Addiere a zu dem bisherigen Ergebnis

ergebnis = ergebnis + a;

}

// Gib das Endergebnis zurück

**return** ergebnis;

}

}

Hier wird die Multiplikation durch eine For-Schleife durchgeführt, die b-Mal durchläuft und in jedem Schritt a zu einem Zwischenergebnis (ergebnis) addiert. Dies spiegelt die Idee der Multiplikation durch Addition wider.

# Aufgabe 3 – Fibonacci in Java

## Implementiere ein Programm (Fibonacci.java), welches die n-te Fibonacci Zahl berechnet und auf die Konsole ausgibt.

**package** ram;

**public** **class** Fibonacci {

**public** **static** **void** main(String[] args) {

// Setze die gewünschte Position der Fibonacci-Zahl (z.B. 10)

**int** n = 10;

// Rufe die Fibonacci-Methode auf und gib das Ergebnis auf der Konsole aus

System.***out***.println("Die " + n + ". Fibonacci-Zahl ist: " + *berechneFibonacci*(n));

}

// Methode zur Berechnung der n-ten Fibonacci-Zahl

**static** **int** berechneFibonacci(**int** n) {

// Initialisiere die ersten beiden Fibonacci-Zahlen

**int** a = 0;

**int** b = 1;

// Spezialfall für n = 0 und n = 1

**if** (n == 0) {

**return** a;

} **else** **if** (n == 1) {

**return** b;

}

// Berechne die n-te Fibonacci-Zahl durch Iteration

**for** (**int** i = 2; i <= n; i++) {

**int** temp = a + b;

a = b;

b = temp;

}

// Gib die n-te Fibonacci-Zahl zurück

**return** b;

}

}

## Wie lautet die 34-te Fibonacci Zahl?

Um die 34-te Fibonacci-Zahl zu berechnen, kann ich die zuvor gegebene Java-Implementierung des Fibonacci-Programms verwenden. Ich ändere einfach den Wert der Variable n in der main-Methode auf 34 und führe das Programm aus.

**package** ram;

**public** **class** Fibonacci\_34 {

**public** **static** **void** main(String[] args) {

// Setze die gewünschte Position der Fibonacci-Zahl auf 34

**int** n = 34;

// Rufe die Fibonacci-Methode auf und gib das Ergebnis auf der Konsole aus

System.***out***.println("Die " + n + ". Fibonacci-Zahl ist: " + *berechneFibonacci*(n));

}

// Methode zur Berechnung der n-ten Fibonacci-Zahl

**static** **int** berechneFibonacci(**int** n) {

// Initialisiere die ersten beiden Fibonacci-Zahlen

**int** a = 0;

**int** b = 1;

// Spezialfall für n = 0 und n = 1

**if** (n == 0) {

**return** a;

} **else** **if** (n == 1) {

**return** b;

}

// Berechne die n-te Fibonacci-Zahl durch Iteration

**for** (**int** i = 2; i <= n; i++) {

**int** temp = a + b;

a = b;

b = temp;

}

// Gib die n-te Fibonacci-Zahl zurück

**return** b;

}

}

### Resultat

Die 34. Fibonacci-Zahl ist: 5702887

## Ergänze das Programm so, dass das Verhältnis zwischen der 34-ten und der 33-ten Fibonacci Zahl berechnet und ausgegeben wird.

**package** ram;

**public** **class** Fibonacci\_Verhaltnis {

**public** **static** **void** main(String[] args) {

// Setze die gewünschte Position der Fibonacci-Zahl auf 34

**int** n = 34;

// Rufe die Fibonacci-Methode auf und gib das Ergebnis auf der Konsole aus

**int** fibonacciN = *berechneFibonacci*(n);

System.***out***.println("Die " + n + ". Fibonacci-Zahl ist: " + fibonacciN);

// Berechne das Verhältnis zur vorherigen Fibonacci-Zahl

**if** (n > 1) {

**int** fibonacciNMinus1 = *berechneFibonacci*(n - 1);

**double** verhaeltnis = (**double**) fibonacciN / fibonacciNMinus1;

System.***out***.println("Das Verhältnis zur vorherigen Fibonacci-Zahl ("

+ (n - 1) + ". Fibonacci-Zahl) ist: " + verhaeltnis);

}

}

// Methode zur Berechnung der n-ten Fibonacci-Zahl

**static** **int** berechneFibonacci(**int** n) {

// Initialisiere die ersten beiden Fibonacci-Zahlen

**int** a = 0;

**int** b = 1;

// Spezialfall für n = 0 und n = 1

**if** (n == 0) {

**return** a;

} **else** **if** (n == 1) {

**return** b;

}

// Berechne die n-te Fibonacci-Zahl durch Iteration

**for** (**int** i = 2; i <= n; i++) {

**int** temp = a + b;

a = b;

b = temp;

}

// Gib die n-te Fibonacci-Zahl zurück

**return** b;

}

}

### Resultat

Die 34. Fibonacci-Zahl ist: 5702887

Das Verhältnis zur vorherigen Fibonacci-Zahl (33. Fibonacci-Zahl) ist: 1.618033988749859

## Weisst du wie dieses Verhältnis auch genannt wird?

Das Verhältnis zwischen aufeinanderfolgenden Fibonacci-Zahlen, insbesondere wenn man sich den Grenzwert für sehr große Fibonacci-Zahlen anschaut, nähert sich dem sogenannten "Goldenen Schnitt" an. Der Goldene Schnitt ist ein irrationaler mathematischer Wert, der etwa 1,618 beträgt. Dieses Verhältnis findet in vielen natürlichen Phänomenen, Kunstwerken und sogar in der Architektur Anwendung.

Das Verhältnis aufeinanderfolgender Fibonacci-Zahlen kann als Annäherung des Goldenen Schnitts betrachtet werden, und es zeigt sich, dass je weiter man in der Fibonacci-Folge fortschreitet, desto näher kommt das Verhältnis dem Goldenen Schnitt.

# Aufgabe 4 - RAL-Programmierung: Subtraktion

## Eigene erste Gedanken zu RAM

### Kurze Erklärung der RAM

Zunächst einmal will ich verstehen, was ein RAM ist und wie solches funktioniert.

Grundsätzlich ist ein RAM eine vereinfachte Darstellung eines Algorithmus oder sogar eines ganzen Programms, damit man später dann einschätzen kann, wie gut und qualitativ es ist.

Ein minimales Modell zur Untersuchung der Komplexität von Algorithmen (RAM-Modell) sollte unbedingt folgendes beinhalten:

* Speicher
* Ein- und Ausgabe
* Programmierbarkeit

### Erklärung eines RAM am Beispiel Addition

### Anzuwendende Befehle

|  |  |  |
| --- | --- | --- |
| Befehl | Argument (Adresse) | Bedeutung |
| LDA | X | Lade den AC mit dem Inhalt von Speicheradresse X (Überschreiben des AC) |
| LDI | X | Lade den AC indirekt mit dem Inhalt von Adresse X (Überschreiben des AC) |
| STA | X | Speichere den Inhalt des AC an Speicheradresse X (Überschreiben der Speicheradresse) |
| STI | X | Speichere den Inhalt des AC indirekt an Adresse X (Überschreiben der Speicheradresse) |
| ADD | X | Addiere den Inhalt von Adresse X zum AC |
| SUB | X | Subtrahiere den Inhalt von Adresse X vom AC |
| JMP | X | Springe zur mit X markierten Instruktion |
| JMZ | X | Springe zur mit X markierten Instruktion, wenn der AC 0 enthält |
| HLT |  | Halt |

Der nullte Schritt ist bei uns immer ein Halt. Wenn man fertig ist mit dem Programm springt man zurück zu 0.(HLT). Dem Fall beginnt jedes Programm erst mit dem ersten Schritt und jedes hat in etwa folgenden Beginn in sich:

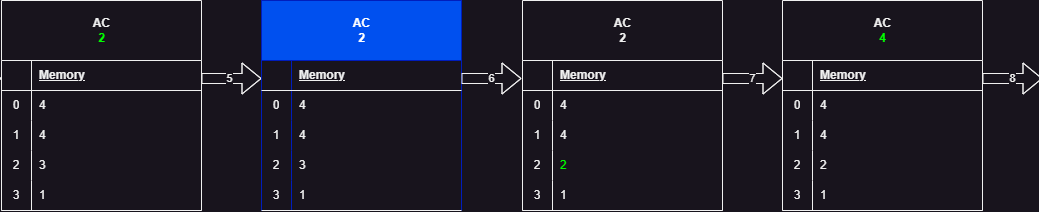
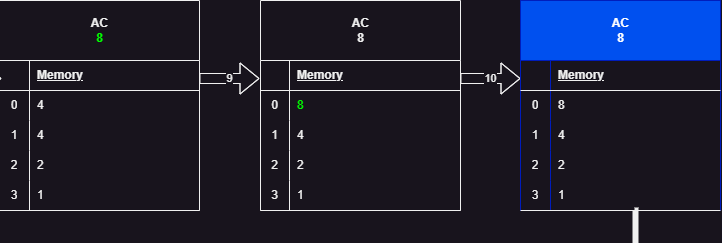
* 0. HLT
* 1. LDA 1
* (2.- …) Der Rest des Programms…

Jump-Befehle (wie JMP und JMZ) orientiere sich nicht an den Akkumulator und das Memory, sondern an das RAL-Programm selbst, sie springen im RAL-Programm herum und berühren den Akkumulator/Memory gar nicht einmal mit diesen Befehlen.

## Implementiere die Subtraktion zweier ganzen Zahlen mit RAL (z.B. 7 – 3 = 4)

# Aufgabe 5 - RAL-Programmierung: Multiplizierung

## Implementiere die Multiplikation zweier ganzen Zahlen mit RAL (Orientiere dich an den Code von Aufgabe 2)



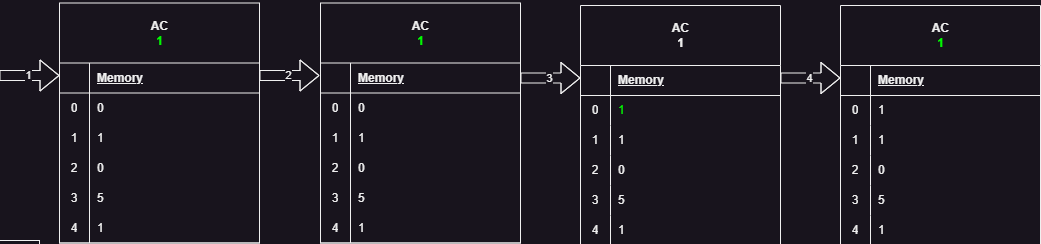
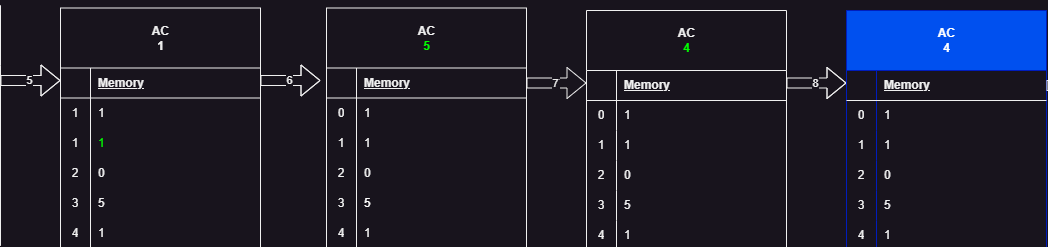
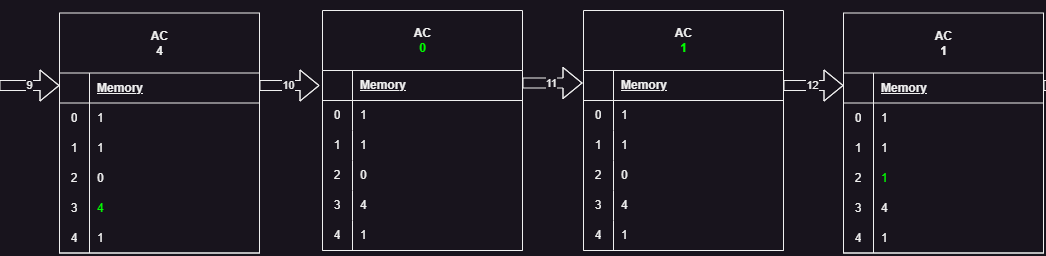
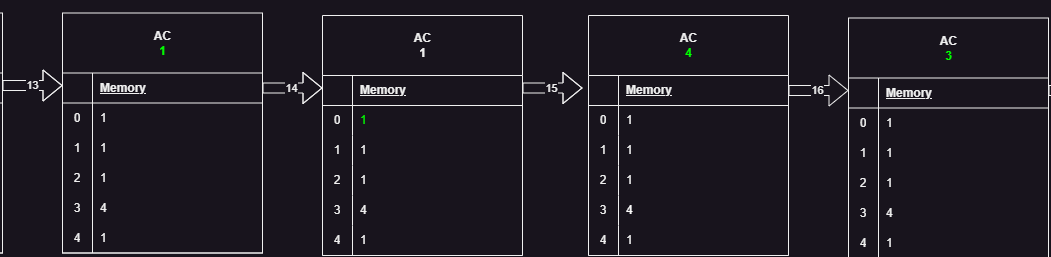
### Erster Loop

### Zweiter Loop

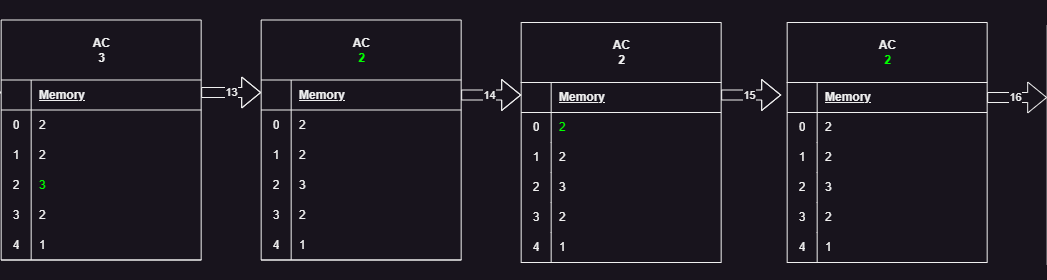
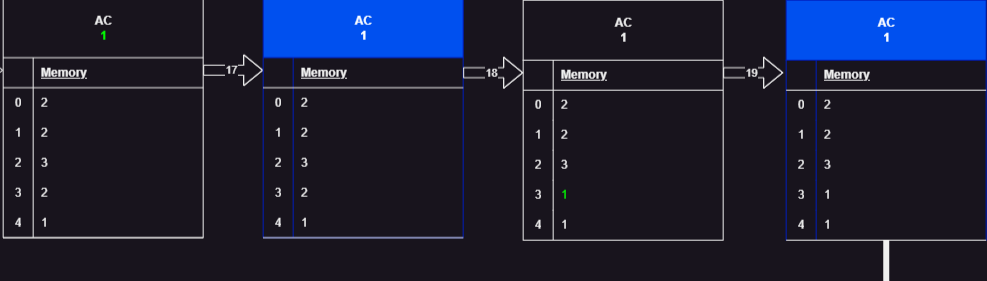
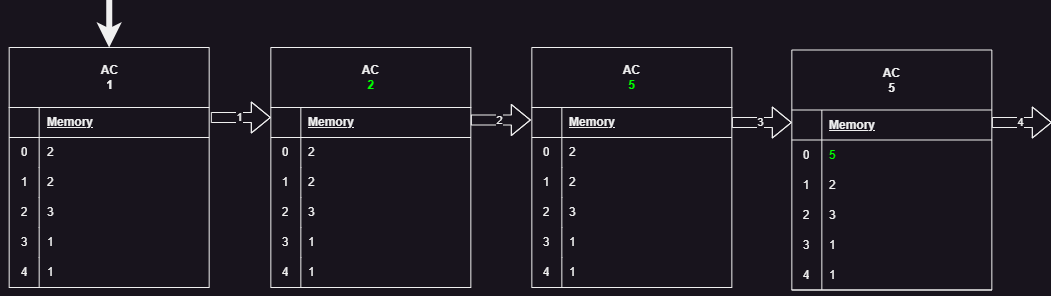
### Überblick

# Aufgabe 6 - RAL-Programmierung: Fibonacci-Folge

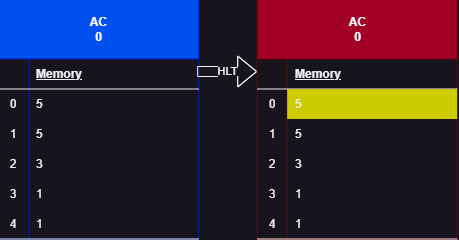
## Implementiere die Berechnung der n-ten Fibonacci Zahl mit RAL (Orientiere dich an den Code von Aufgabe 3)



### Erster Loop



### Zweiter Loop



### Überblick

# Zusatzaufgabe – Bubble-Sort

## Prinzip des Bubble-Sort erklärt

Wir vergleichen zwei benachbarte Zahlen und falls die linke Zahl grösser als die rechte Zahl ist, tauschen wir die Position der beiden Zahlen. Dieses Verfahren bewirkt, dass jeweils die grösste Zahl von links nach rechts wandert (Bubbles im Wasser nach oben schiessen). Nach mehrmaliger Wiederholung erhalten wir eine aufsteigend sortierte Liste.

## Implementiere den Bubble-Sort in Java und in RAL.

### Java

**package** ram;

**public** **class** BubbleSort {

**public** **static** **void** main(String[] args) {

**int** r1[]={2,5,11,7,24,8,9,11};

// gibt den Array in der ursprünglichen Reihenfolge aus:

**for**(**int** i=0; i<=r1.length-1; i++) {

System.***out***.print(r1[i]+" ");

}

System.***out***.println("\n");

// gibt den der Grösse nach geordneten Array aus:

**for**(**int** i=0; i<=r1.length-1; i++) {

System.***out***.print(BubbleSort.*bubbleSort*(r1)[i]+" ");

}

}

**public** **static** **int**[] bubbleSort(**int**[] a) {

**int** l=a.length;

**boolean** flagSwap=**false**;

**do** {

flagSwap=**false**;

**for** (**int** i=0; i<l-1; i++) {

**if** (a[i]>a[i+1]) {

flagSwap=**true**;

**int** temp=a[i];

a[i]=a[i+1];

a[i+1]=temp;

}

}

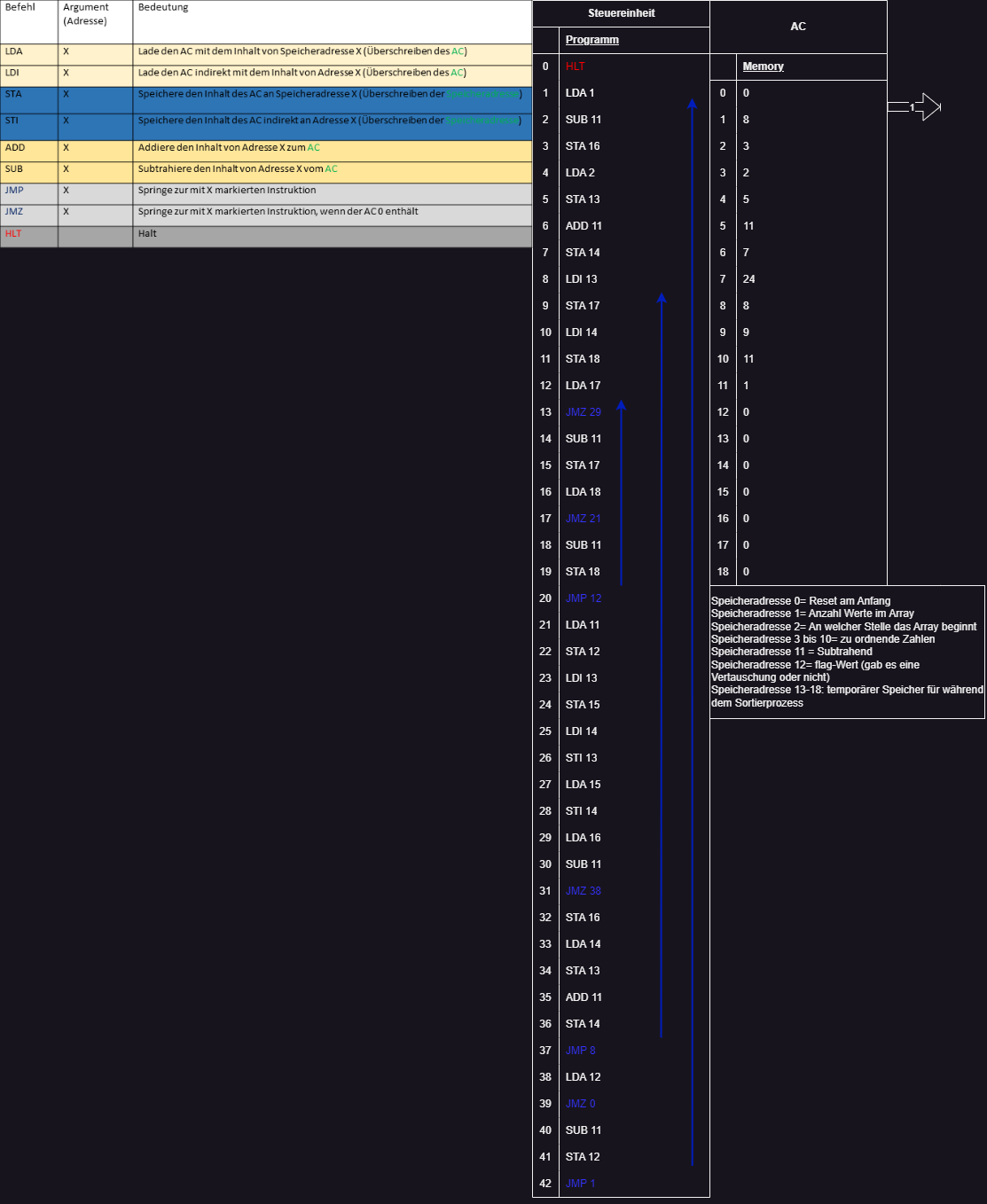
} **while**(flagSwap==**true**);

**return** a;

}

}

### RAL

Ich gehe davon aus, dass beim RAL-Programm nur positive Zahlen und die Null Elemente bearbeitet werden können. Dem Fall würden z.B. negative Zahlen einfach als Null gewertet werden. 

#### Erste Schritte

#### Überblick (in etwa)

#### Ergebnis