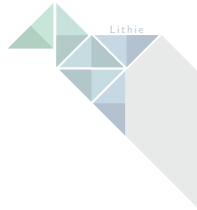


Das Ende Naht aht aht aht ht t...

Ohren hab ich gesagt g



Florian Sihler ◦ KW 3



Präsenzaufgabe

1

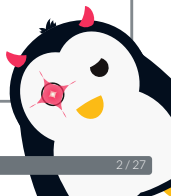
Wenn du dein Kinderzimmer nicht aufräumst, ...

In dieser Aufgabe sollen Sie die *Binäre Suche* implementieren, welche nützlich ist um ein sortiertes Array effizient zu durchsuchen. Schreiben Sie die Methoden-Signatur  `public static int`  `find(int[] array, int element)`. Diese Methode soll ein *aufsteigend* sortiertes Array und das gesuchte Element als Parameter übernehmen, und daraufhin den Index des Elements im Array zurückgeben, sofern dieses vorhanden ist. Sollte das Element mehrfach vorhanden sein, soll der kleinste Index zurückgegeben werden. Falls das Element nicht im Array vorhanden ist, soll `-1` zurückgegeben werden. Implementieren Sie die eigentliche *binäre Suche* rekursiv. Hilfsmethoden sind gestattet.

4 13 21 29 33 34 56 68 71 74 78



...

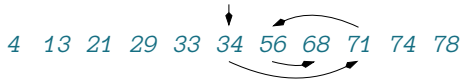


Präsenzaufgabe

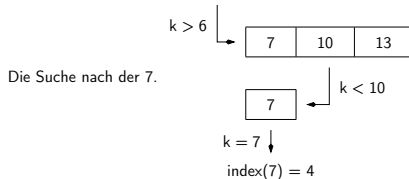
1

Wenn du dein Kinderzimmer nicht aufräumst, ...

Schreiben Sie die Methode `public static int find(int[] array, int element)` rekursiv. Sie soll für ein *aufsteigend* sortiertes Array mittels der *binären Suche* den Index des gesuchten Elements im Array zurückgeben, sofern dieses vorhanden ist. Wenn das Element mehrfach vorhanden ist, soll der kleinste Index, wenn es nicht vorhanden ist, soll `-1` zurückgegeben werden. Hilfsmethoden sind gestattet.



Index:	0	1	2	3	4	5	6
Array:	1	3	4	6	7	10	13



Präsenzaufgabe - Lösung

- Wir erschaffen eine Suche: `BinarySearch.java`. (Eine binäre Suche)

- Die Idee:

- Wir markieren das zu durchsuchende Fenster mit `left` und `right`.
- Ist das Fenster leer, haben wir das Element nicht gefunden.
- Sonst prüfen wir: Ist das mittlere Element das Gesuchte?
- Ist es kleiner, suchen wir links, ist es größer, rechts weiter.



1. Basisfall: keine Elemente mehr.
2. Basisfall: Element gefunden.
Sonst: Suche links/rechts.

- Dafür nutzen wir die Hilfsmethode:

```
binarySearch(int[] array, int left, int right, int element).
```

- Wir beginnen mit der Suche im gesamten Array:

```
binarySearch(array, 0, array.length - 1, element);
```



Präsenzaufgabe - Lösung

```
static int binarySearch(int[] array, int left, int right, int element) {  
    // 1) Fenster ist leer  
    if(right < left)  
        return -1;  
  
    int middle = left + (right - left) / 2;  
    // 2) Ist das Mittlere das Gesuchte?  
    if(array[middle] == element)  
        return middle;  
    // 3) Kleiner: suche links weiter  
    else if(array[middle] > element)  
        return binarySearch(array, left, middle - 1, element);  
    // 4) Größer: suche rechts weiter  
    else  
        return binarySearch(array, middle + 1, right, element);  
}
```

Präsenzaufgabe - Lösung

- Einen Teil haben wir noch nicht gemacht:
Sollte das Element mehrfach vorhanden sein, soll der kleinste Index zurückgegeben werden
- Idee: Wir verringern den Index, solange das Element immer noch das Gesuchte ist:

```
public static int find(int[] array, int element) {  
    int index = binarySearch(array, 0, array.length - 1, element);  
    if(index == -1) return -1;  
  
    for(int i = 1; i <= index; i++) {  
        if(array[index - i] != element)  
            return index - i + 1;  
    }  
    return 0;  
}
```

Rekursive Variante (etwas zu lang)

```
// return findLower(array, element, index, 1);  
int findLower(int[] arr, int e, int idx, int i) {  
    if(i > idx)  
        return 0;  
    if(arr[idx - i] != e)  
        return idx - i + 1;  
    return findLower(arr, e, idx, i + 1);  
}
```

```
> int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1;  
    int mid = l + (r - l) / 2;  
    if (arr[mid] == e) return mid;  
    else if (arr[mid] > e)  
        return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

```
int find(int[] arr, int e) {  
> int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

```
> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);
```



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1; r=6, l=0  
    int mid = l + (r - l) / 2; mid=3  
    if (arr[mid] == e) return mid; e=-14  
    else if (arr[mid] > e) 24 > -14  
        > return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

```
int find(int[] arr, int e) {  
> int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);




```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1; r=2, l=0  
    int mid = l + (r - l) / 2; mid=1  
    > if (arr[mid] == e) return mid; e=-14, mid=1  
    else if (arr[mid] > e)  
        > return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

```
int find(int[] arr, int e) {  
    > int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1;  
    int mid = l + (r - l) / 2;  
    if (arr[mid] == e) return mid;  
    else if (arr[mid] > e)  
        return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}  
  
int find(int[] arr, int e) {  
    int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1; idx=1  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

```
find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);
```



```
int search(int[] arr, int l, int r, int e) {
    if(r < l) return -1;
    int mid = l + (r - l) / 2;
    if (arr[mid] == e) return mid;
    else if (arr[mid] > e)
        return search(arr, l, mid - 1, e);
    else
        return search(arr, mid + 1, r, e);
}

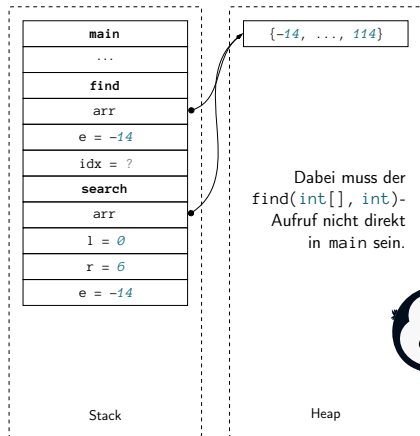
int find(int[] arr, int e) {
    int idx = search(arr, 0, arr.length - 1, e);
    if(idx == -1) return -1;
    for(int i = 1; i <= idx; i++) {
        if(arr[idx - i] != e)
            return idx - i + 1;
    }
    return 0; //Wir sind ganz links
}
```

`find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);` < // → 0



```
> int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1;  
    int mid = l + (r - l) / 2;  
    if (arr[mid] == e) return mid;  
    else if (arr[mid] > e)  
        return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}  
  
int find(int[] arr, int e) {  
> int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

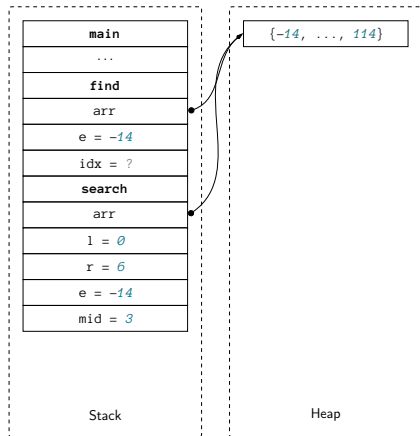
```
> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);
```



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1; r=6, l=0  
    int mid = l + (r - l) / 2; mid=3  
    if (arr[mid] == e) return mid; e=-14  
    else if (arr[mid] > e) 24 > -14  
        > return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

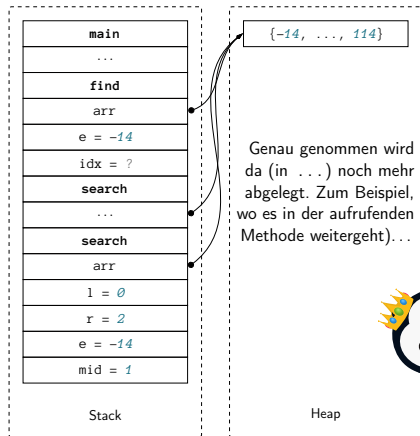
```
int find(int[] arr, int e) {  
    > int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1; r=2, l=0  
    int mid = l + (r - l) / 2; mid=1  
    > if (arr[mid] == e) return mid; e=-14, mid=1  
    else if (arr[mid] > e)  
        > return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}  
  
int find(int[] arr, int e) {  
    > int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

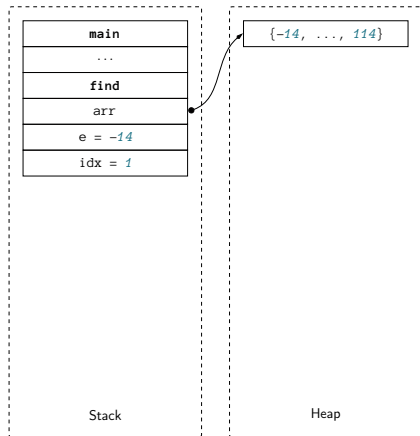
> find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1;  
    int mid = l + (r - l) / 2;  
    if (arr[mid] == e) return mid;  
    else if (arr[mid] > e)  
        return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

```
int find(int[] arr, int e) {  
    int idx = search(arr, 0, arr.length - 1, e);  
    > if(idx == -1) return -1; idx=1  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0;  
}
```

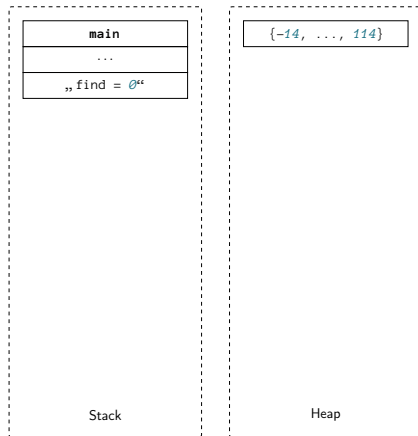
```
find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);
```



```
int search(int[] arr, int l, int r, int e) {  
    if(r < l) return -1;  
    int mid = l + (r - l) / 2;  
    if (arr[mid] == e) return mid;  
    else if (arr[mid] > e)  
        return search(arr, l, mid - 1, e);  
    else  
        return search(arr, mid + 1, r, e);  
}
```

```
int find(int[] arr, int e) {  
    int idx = search(arr, 0, arr.length - 1, e);  
    if(idx == -1) return -1;  
    for(int i = 1; i <= idx; i++) {  
        if(arr[idx - i] != e)  
            return idx - i + 1;  
    }  
    return 0; //Wir sind ganz links  
}
```

`find(new int[]{-14, -14, 24, 24, 42, 109, 113}, -14);` < // → 0



Übungsblatt 9 - Aufgabe 1

In dieser Aufgabe sollen Sie eine Methode implementieren, welche die Summe errechnet die ein Startkapital in einer bestimmten Anzahl an Jahren mit einem fixen Jahreszinssatz erreicht. Die Methode soll die folgende Signatur Gestalt haben: **public static** \leftarrow **double** compoundInterest(**double** capital, **double** interestRate, **int** years)
Hierbei soll der Parameter capital das Startkapital angeben, interestRate den Zinssatz in Prozent und years die Laufzeit. Verwenden Sie bei ihrer Implementierung keine Schleifen, sondern rekursive Methoden-Aufrufe!

Iterativer Ansatz:

```
for(int i = 0; i < years; i++) {  
    capital *= (1 + interestRate);  
}
```

```
for(int i = years; i > 0; i--) {  
    capital *= (1 + interestRate);  
}
```

Übungsblatt 9 - Aufgabe 1

- Die Datei befindet sich hier: [CompoundInterest.java](#)

```
for(int i = years; i > 0; i--) {  
    capital *= (1 + interestRate);  
}
```

```
public static  
double compoundInterest(double capital, double interestRate, int years) {  
    if(years == 0)  
        return capital;  
    else  
        return compoundInterest(capital * (1 + interestRate), interestRate,  
                                years - 1);  
}
```

Warum die herunterzählende Variante? Nun, sie spart uns eine weitere Variable in der Rekursion. So müssen wir nur prüfen, wann years den Wert 0 erreicht. Andernfalls wäre eine weitere Variable notwendig um das ursprüngliche years zu halten.



Übungsblatt 9 - Aufgabe 2

In dieser Aufgabe sollen Sie die Methode `public static boolean isPalindrome (String s)` implementieren, welche überprüfen soll ob es sich bei dem als Parameter übergebenen String `s` um ein Palindrom handelt. Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen identisch ist. Die Überprüfung soll **nicht case-sensitive** sein, d.h. das Wort *Kajak* soll zum Beispiel ein gültiges Palindrom sein. Verwenden Sie bei ihrer Implementierung **keine Schleifen**, sondern **rekursive Methoden-Aufrufe**!

Idee:

- Prüfe für $i = 0$ bis $i = \lfloor s.length/2 \rfloor$, ob „`s[i] == s[s.length - i - 1]`“.
- `String::toLowerCase()` entweder für jeden Vergleich oder einmal per Hilfsmethode.
- Noch einfacher: Anstelle `i` zu inkrementieren, löschen wir das erste und letzte Zeichen nach dem Vergleich (via `String::substring(int, int)`—das Ende ist exklusiv)!

Übungsblatt 9 - Aufgabe 2

- Die Datei befindet sich hier: [Palindrome.java](#)

```
public static boolean isPalindrome(String s) { // die "Hilfsmethode"
    return isPalindromeRecursive(s.toLowerCase());
}

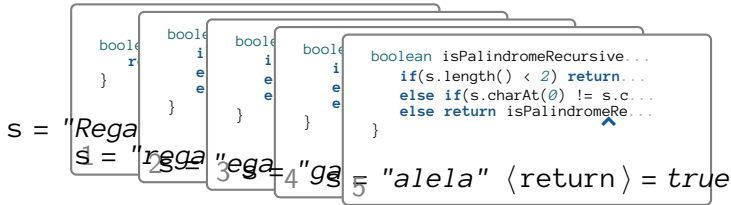
private static boolean isPalindromeRecursive(String s) {
    if(s.length() < 2) // Basisfall: weniger als zwei Zeichen (Abrunden)
        return true;
    else if(s.charAt(0) != s.charAt(s.length() - 1)) // Basisfall
        return false;
    else // Rekursionsfall
        return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

```
private static boolean isPalindrome(String s) {  
    return isPalindromeRecursive(s.toLowerCase());  
}  
private static boolean isPalindromeRecursive(String s) {  
    if(s.length() < 2) return true;  
    else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;  
    else return isPalindromeRecursive(s.substring(1, s.length() - 1));  
}
```

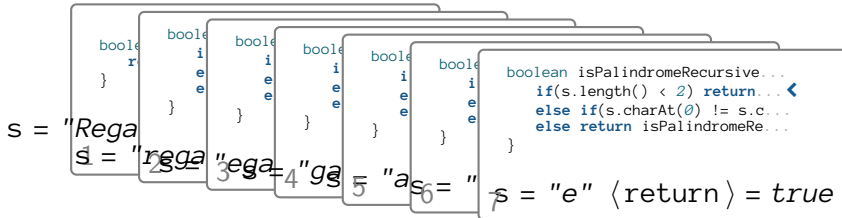
„RegalelaGEr“

(Wer sieht auch ein e, dass die Arme hochwirft?)

```
private static boolean isPalindrome(String s) { s="RegalelaGEr"
    return isPalindromeRecursive(s.toLowerCase()); "regalelager"
}
private static boolean isPalindromeRecursive(String s) { s="alela"
    if(s.length() < 2) return true;
    > else if(s.charAt(0) != s.charAt(s.length() - 1)) return false; 'a'!='a'
    else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```



```
private static boolean isPalindrome(String s) { s="RegalelaGEr"
    return isPalindromeRecursive(s.toLowerCase()); "regalelager"
}
private static boolean isPalindromeRecursive(String s) { s="alela"
    if(s.length() < 2) return true; <
    else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;
    else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```



```
private static boolean isPalindrome(String s) { s="RegalelaGEr"
    return isPalindromeRecursive(s.toLowerCase()); < true
}
private static boolean isPalindromeRecursive(String s) {
    if(s.length() < 2) return true;
    else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;
    else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

```
boolean isPalindrome(String s) {
    return isPalindromeRecurs...
}
```

$s = \text{"RegalelaGEr"} \quad \langle \text{return} \rangle = t$
1

Mit `<return>` haben wir hier die zurückzugebenden anonymen Variablen referenziert. Generell ist hier das „Speichern“ der Position für den Aufstieg der Rekursion informal dargestellt.



- Die ersten beiden Aufgaben lassen sich *einfacher* iterativ prüfen.
- Für das Palindrom schauen wir für jedes Zeichen i der „linken Hälfte“ ob es mit dem gespiegelten $\text{length} - i - 1$ der „rechten Hälfte“ übereinstimmt:

```
public static boolean isPalindromeIterative(String s) {  
    s = s.toLowerCase();  
    for(int i = 0; i < s.length() / 2; i++) {  
        if(s.charAt(i) != s.charAt(s.length() - i - 1))  
            return false;  
    }  
    return true;  
}
```

- Dies können wir als Tail-Rekursion umschreiben ([Palindromelterative.java](#)):

```
public static boolean isPalindrome(String s) {  
    return helper(s.toLowerCase(), 0);  
}
```

```
private static boolean helper(String s, int i) {  
    if (i >= s.length() / 2)  
        return true;  
    if (s.charAt(i) != s.charAt(s.length() - i - 1))  
        return false;  
    return helper(s, i + 1);  
}
```

```
s = s.toLowerCase();  
for(int i = 0; i < s.length() / 2; i++) {  
    if(s.charAt(i) != s.charAt(s.length() - i - 1))  
        return false;  
}  
return true;
```

Übungsblatt 9 - Aufgabe 3

In dieser Aufgabe sollen Sie die Methode `public static void printCombinations(int[] array, int n)` implementieren, welche ein Integer Array und eine natürliche Zahl `n` als Parameter übernimmt. Die Methode soll rekursiv alle möglichen Kombinationen aus `n` Elementen des Arrays ausgeben. Hierbei soll `n` maximal so groß sein wie die Länge des Arrays. Verwenden Sie bei ihrer Implementierung **rekursive Methoden-Aufrufe** wann immer möglich. Sie benötigen **maximal eine Schleife** für die Lösung.

Hinweis: Sie können sich entscheiden ob Sie die Ausgabe der Kombinationen mit oder ohne Wiederholung von Elementen implementieren.

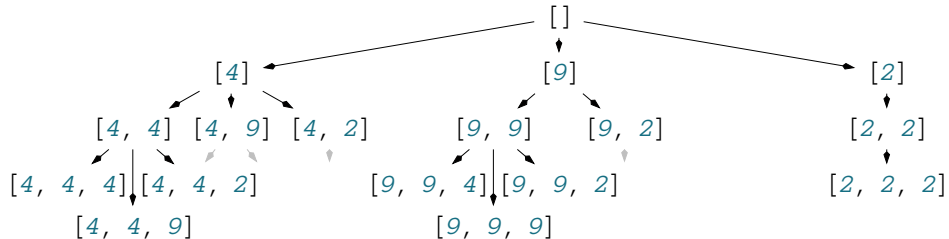
Übungsblatt 9 - Aufgabe 3

Da wir die Maximalgröße der Ausgabekombinationen kennen, wird das Hinzufügen eher in der Form $[0,0,0] \rightarrow [1,0,0] \rightarrow [1,2,0] \rightarrow [1,2,3]$ erfolgen.

■ Zuerst die Idee:

- Wir verwenden ein Array um die Ausgabekombination zu halten.
- Basisfall: das Array hat die gewünschte Länge, dann geben wir es aus.
- Sonst: erweitern wir das Array für jedes Symbol einmal und erzeugen so eine baumartige Rekursion.

■ Ein Beispiel mit $[4, 9, 2]$ und $n = 3$, mit Wiederholung:



Übungsblatt 9 - Aufgabe 3

- Die Datei befindet sich hier: [Combinations.java](#) und [CombinationsRepetition.java](#).
- Zunächst das rekursive Ausgeben eines Arrays.
- Idee: wir inkrementieren rekursiv einen Zähler anstelle der for-Schleife.

```
public static void printArray(int[] combination, int i) {  
    if(i >= combination.length) { // Basisfall  
        System.out.println(); // Ende der Ausgabe  
    } else { // Rekursionsfall: ausgeben und nächstes  
        System.out.print(combination[i] + "_");  
        printArray(combination, i + 1);  
    }  
}
```

Anstelle das Array zu nutzen und das Auszugeben, können wir für diese Aufgabe auch einen String aufbauen und ausgeben. Das kommt weiter unten (mit der *#Prefixalternative*).



Übungsblatt 9 - Aufgabe 3

- Für das baumartige Antackern bauen wir uns eine Hilfe (zuerst mit Wiederholung):

```
static void helper(int[] array, int n, int[] combs, int start, int index)
```

- Dabei sind `array` und `n` die gegebene Eingabe.
- `combs` ist das zu konstruierende Array.
- `start` beschreibt ab welchem Feld aus `array` gezogen werden soll.
- `index` ist das aktuell in `combs` zu füllende Feld.

Übungsblatt 9 - Aufgabe 3

- Die Baumrekursion gestaltet sich wie folgt:

```
static void helper(int[] arr, int n, int[] combs, int start, int idx) {  
    if (idx == n) { // Basisfall  
        printArray(combs, 0);  
    } else { // Sonst: Für jedes Symbol...  
        for (int i = start; i < arr.length; i++) {  
            combs[idx] = arr[i];  
            helper(arr, n, combs, i, idx + 1);  
        }  
    }  
}
```

- Aber halt... Arrays sind komplexe Datentypen. Warum geht das? (Tipp: simulieren.)

- Allgemein ist es besser, einen eignen (/lokalen) Zustand sicherzustellen (hier iterativ!):

```
for (int i = start; i < array.length; i++) {  
    int[] newCombs = new int[combs.length];  
    for(int j = 0; j < combs.length; j++) newCombs[j] = combs[j];  
    newCombs[index] = array[i];  
    helper(array, n, newCombs, i, index + 1);  
}
```

- Oder um es anderweitig lokal zu halten – komplexe Datentypen eben:

```
for (int i = start; i < array.length; i++) {  
    int tmp = combs[index];  
    combs[index] = array[i];  
    helper(array, n, combs, i, index + 1);  
    combs[index] = tmp;  
}
```



Safety measures: on

- Jetzt haben wir aber nichts wiederhergestellt...
- Dies macht die nächste Iteration, zusammen mit einem sogenannten *Tiefendurchlauf!*
(Die Arrays rechts sind ausgegraut, da sie nur den Aufrufzustand ausgeben. Natürlich wird immer das selbe Array verändert und überschrieben. Wir werden Tiefendurchläufe bei Bäumen inspizieren.)

```
static void helper(int[] arr,
int n, int[] combs, int s, int idx) {
    if (idx == n) { 0==3
        printArray(combs, 0);
    } else {
        for (int i = s; i < arr.length; i++) {
            combs[idx] = arr[i]; combs={2, 0, 0}
            helper(arr, n, combs, idx + 1);
        }
    }
}
```

({2, 4, 6}, 3, {0, 0, 0}, 0, 0) i=0

- Jetzt haben wir aber nichts wiederhergestellt...
- Dies macht die nächste Iteration, zusammen mit einem sogenannten *Tiefendurchlauf!*
(Die Arrays rechts sind ausgegraut, da sie nur den Aufrufzustand ausgeben. Natürlich wird immer das selbe Array verändert und überschrieben. Wir werden Tiefendurchläufe bei Bäumen inspizieren.)

```
static void helper(int[] arr,
int n, int[] combs, int s, int idx) {
    if (idx == n) { 3==3
        printArray(combs, 0); → 2 2 2
    } else {
        for (int i = s; i < arr.length; i++) {
            combs[idx] = arr[i];
            helper(arr, n, combs, idx + 1);
        }
    }
}
```

```
({2, 4, 6}, 3, {0, 0, 0}, 0, 0) i=0
({2, 4, 6}, 3, {2, 0, 0}, 0, 1) i=0
({2, 4, 6}, 3, {2, 2, 0}, 0, 2) i=0
({2, 4, 6}, 3, {2, 2, 2}, 0, 3)
```

- Jetzt haben wir aber nichts wiederhergestellt...
- Dies macht die nächste Iteration, zusammen mit einem sogenannten *Tiefendurchlauf!*
(Die Arrays rechts sind ausgegraut, da sie nur den Aufrufzustand ausgeben. Natürlich wird immer das selbe Array verändert und überschrieben. Wir werden Tiefendurchläufe bei Bäumen inspizieren.)

```
static void helper(int[] arr,
int n, int[] combs, int s, int idx) {
    if (idx == n) {
        printArray(combs, 0);
    } else {
        for (int i = s; i < arr.length; i++) {
            combs[idx] = arr[i]; combs={2, 4, 6}
            helper(arr, n, combs, idx + 1);
        }
    }
}
```

(*{2, 4, 6}*, 3, {0, 0, 0}, 0, 0) *i=0*
(*{2, 4, 6}*, 3, {2, 0, 0}, 0, 1) *i=1*

- Jetzt haben wir aber nichts wiederhergestellt...
- Dies macht die nächste Iteration, zusammen mit einem sogenannten *Tiefendurchlauf!*
(Die Arrays rechts sind ausgegraut, da sie nur den Aufrufzustand ausgeben. Natürlich wird immer das selbe Array verändert und überschrieben. Wir werden Tiefendurchläufe bei Bäumen inspizieren.)

```
static void helper(int[] arr,
int n, int[] combs, int s, int idx) {
    if (idx == n) { 3==3
        printArray(combs, 0); → 2 4 6
    } else {
        for (int i = s; i < arr.length; i++) {
            combs[idx] = arr[i];
            helper(arr, n, combs, idx + 1);
        }
    }
}
```

```
({2, 4, 6}, 3, {0, 0, 0}, 0, 0) i=0
({2, 4, 6}, 3, {2, 0, 0}, 0, 1) i=1
({2, 4, 6}, 3, {2, 4, 6}, 1, 2) i=0
({2, 4, 6}, 3, {2, 4, 4}, 1, 3)
```

- Jetzt haben wir aber nichts wiederhergestellt...
- Dies macht die nächste Iteration, zusammen mit einem sogenannten *Tiefendurchlauf!*
(Die Arrays rechts sind ausgegraut, da sie nur den Aufrufzustand ausgeben. Natürlich wird immer das selbe Array verändert und überschrieben. Wir werden Tiefendurchläufe bei Bäumen inspizieren.)

```
static void helper(int[] arr,
int n, int[] combs, int s, int idx) {
    if (idx == n) {
        printArray(combs, 0);
    } else {
        for (int i = s; i < arr.length; i++) {
            combs[idx] = arr[i];
            helper(arr, n, combs, idx + 1);
        }
    }
}
```

Hier hatte Flo dann auch keine Zeit mehr.

Sonntags um 2:54 Uhr

```
({2, 4, 6}, 3, {0, 0, 0}, 0, 0) i=0
({2, 4, 6}, 3, {2, 0, 0}, 0, 1) i=1
({2, 4, 6}, 3, {2, 4, 6}, 1, 2) i=0
```

- Der Start der Rekursion ist nun recht einfach:

```
public static void printCombinations(int[] array, int n) {  
    int[] combination = new int[n]; // initialisiert mit 0  
    helper(array, n, combination, 0, 0);  
}  
  
private static void helper(int[] array, int n, int[] combs, int start, int index) {  
    if (index == n) { printArray(combs, 0); }  
    else {  
        for (int i = start; i < array.length; i++) {  
            combs[index] = array[i];  
            helper(array, n, combs, i, index + 1);  
        }  
    }  
}
```

- Wir inkrementieren i einfach um 1 im rekursiven Aufruf, damit es nicht wieder das selbe Feld wählen kann.

```
static void helper(int[] array, int n, int[] combs, int start, int index) {
    if (index == n) {
        printArray(combs, 0);
    } else {
        for (int i = start; i < array.length && array.length - i >= n - index; i++) {
            combs[index] = array[i];
            helper(array, n, combs, i + 1, index + 1);
        }
    }
}
```

- start von unten: nach der Wahl eines i -ten Elements nur noch nach $i + 1$ wählen.
- $array.length - i >= n - index$ sichert optional zu, dass noch mindestens $n - index$ Elemente („weiter rechts“) im Array wählbar bleiben. (Bonusfrage fürs Tutorium:

Warum optional?)

- Aus den Gründen, die uns die Abwesenheit der Sicherheitsmaßnahmen erlauben, können wir auch direkt einen `String` bauen (`CombinationsString.java`).
- Die Idee (ohne Wiederholung, mit ist aber vergleichbar) ist wie folgt:
 - Für jede Stelle im Array können wir entscheiden ob sie Teil der Kombination ist oder nicht.
 - Am Ende („akzeptieren“) geben wir alle Entscheidungen aus, die n-lange Kombinationen produzieren.
- Wir nutzen die rekursive helper Funktion:

```
public static void helper(int[] array, int n, int index, String prefix);
```

- Der `index` markiert dabei, die Stelle für die Kombination im `array`. `n` die verbleibenden Felder der Kombination.
- `prefix` markiert den Teil, den wir bisher für die Kombination halten.

$$\text{helper}(a, n, i, p) = \begin{cases} \text{gebe aus: „p“} & n = 0 \\ \text{verwerfe Versuch} & i = a.\text{length} \\ \underbrace{\text{helper}(a, n - 1, i + 1, p + a[i] + " ")}_{\text{nehme } a[i] \text{ mit}} \& \underbrace{\text{helper}(a, n, i + 1, p)}_{\text{nehme } a[i] \text{ nicht mit}} & \text{sonst.} \end{cases}$$

$$\text{helper}(a, n, i, p) = \begin{cases} \text{gebe aus: „p“} & n = 0 \\ \text{verwerfe Versuch} & i = a.\text{length} \\ \underbrace{\text{helper}(a, n-1, i+1, p + a[i] + " ")}_{\text{nehme } a[i] \text{ mit}} \& \underbrace{\text{helper}(a, n, i+1, p)}_{\text{nehme } a[i] \text{ nicht mit}} & \text{sonst.} \end{cases}$$

- Damit beläuft sich der gesamte Code auf (die Skizze...):

```
public static void printCombinations(int[] array, int n) {
    helper(array, n, 0, "");
}

private static void helper(int[] array, int n, int index, String prefix) {
    if (n == 0) { System.out.println(prefix); }
    else if (index == array.length) { return; }
    else {
        helper(array, n - 1, index + 1, prefix + array[index] + "_");
        helper(array, n, index + 1, prefix);
    }
}
```

- Wir können das endständige Leerfeld aus dem String entfernen oder die Rekursion bereits eine Stufe früher „abbrechen“. (Ignorieren wir den Sonderfall $n \leq 0$ mal.)
- Das findet sich in `CombinationsStringShort.java`.
- Kurz gesagt: In diesem Fall muss die letzte Iteration gesondert behandeln:

```
private static void helper(int[] array, int n, int index, String prefix) {  
    if (index == array.length) return;  
  
    if (n == 1) System.out.println(prefix + array[index]);  
    else helper(array, n - 1, index + 1, prefix + array[index] + "_");  
  
    helper(array, n, index + 1, prefix); // Für die letzte Iteration  
}
```



You expected... a recursion gag? Read this line again!