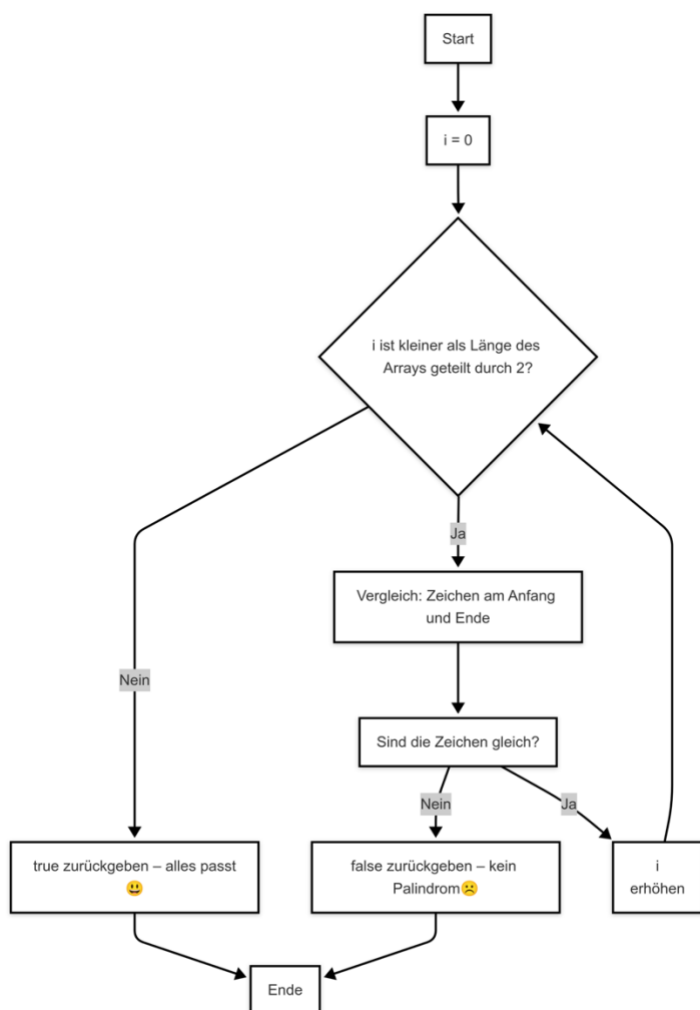


Aufgabe a3x1

```
public boolean isPalindrome(char[] arrayOfChars) {

    for (int i = 0; i < arrayOfChars.length/2; i++) {
        if(arrayOfChars[i]== arrayOfChars[arrayOfChars.length-1-i]) {
        }
        else return false;
    }

    return true;
}
```



Beschreibung:

- Nach der Definition ist ein Wort ein Palindrom, wenn die Buchstaben an den Stellen n und $(\text{Wortlänge} - 1 - n)$ übereinstimmen. Daraus folgt, dass man nur die ersten $(\text{Wortlänge} / 2)$ Buchstabenpaare vergleichen muss;
- Sollte an den genannten Positionen kein gleiches Buchstabenpaar stehen, ist dieses Wort kein Palindrom und somit wird sofort falsch ausgegeben, ohne die restlichen Buchstabenpaare zu überprüfen.

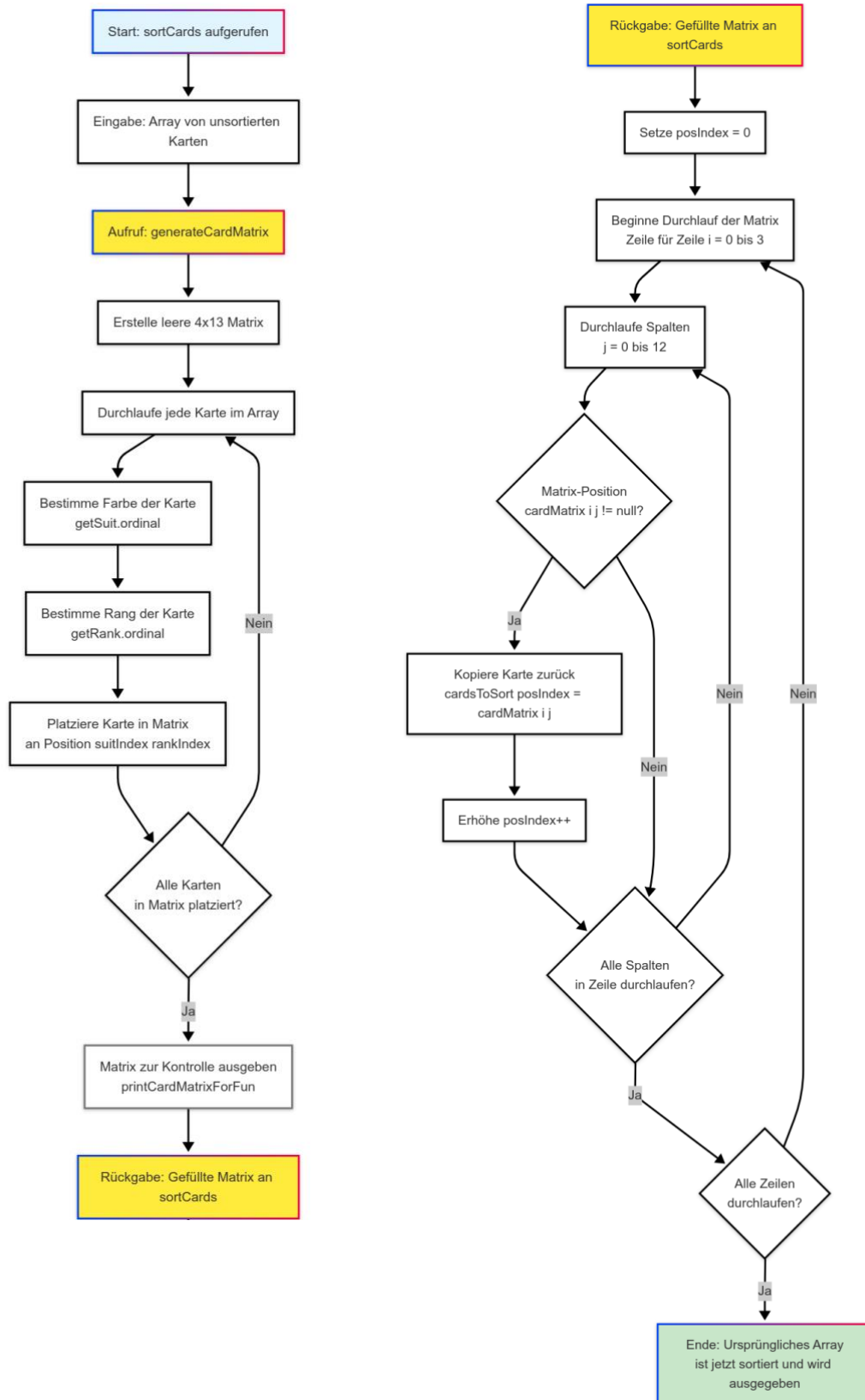
Aufgabe a3x2

```
public static Card[][] generateCardMatrix (Card[] givenCards) {  
    Card[][] cardMatrix = new Card[4][13];  
  
    for (Card card : givenCards) {  
        int suitIndex = card.getSuit().ordinal();  
        int rankIndex = card.getRank().ordinal();  
        cardMatrix[suitIndex][rankIndex] = card;  
    }  
    printCardMatrixForFun(cardMatrix);  
    return cardMatrix;  
}  
  
public static void sortCards(Card[] cardsToSort) {  
  
    Card[][] cardMatrix = generateCardMatrix(cardsToSort);  
    int posIndex = 0;  
    for (int i = 0; i < 4; i++) {  
        for (int j = 0; j < 13; j++) {  
            if (cardMatrix[i][j] != null) {  
                cardsToSort[posIndex] = cardMatrix[i][j];  
                posIndex++;  
            }  
        }  
    }  
}
```

```
}//method
```

Beschreibung:

- Die Methode *sortCards* wird aufgerufen. Dabei wird eine Referenz auf das unsortierte Kartenarray übergeben;
- Innerhalb der Methode wird die Hilfsmethode *generateCardMatrix* aufgerufen, der ebenfalls die Referenz auf das unsortierte Array übergeben wird;
- Da sowohl Rank als auch Suit einen Index besitzen, können die Karten entsprechend in einem zweidimensionalen Array (*cardMatrix*) einsortiert werden;
- Nach dem Befüllen wird die sortierte *cardMatrix* an *sortCards* zurückgegeben;
- Die Methode *sortCards* durchläuft alle Zeilen und Spalten der *cardMatrix*. Wenn sich an einer Stelle eine Karte befindet (!= null), wird sie an der aktuellen Position im ursprünglichen Array *cardsToSort* gespeichert.



Aufgabe a3x3

```
public long process( long[][] theArray ){
    int patternHeight = 2; int patternWidth = 2;
    int patternSize = patternHeight + patternWidth;
    int startPosition = 0;
    int minMiddleRowLength, currentRowLength, minEdgeRowLength;
    int limitLength;
    long totalSum = 0;

    while (startPosition+patternSize <= theArray.length) {
        currentRowLength = 0;
        limitLength = 0;
        minMiddleRowLength = Integer.MAX_VALUE;
        minEdgeRowLength = Integer.MAX_VALUE;
        for (int i = startPosition; i<patternSize+startPosition; i++) {
            if (i - startPosition == 1 || i - startPosition == 2) {
                currentRowLength = theArray[i].length;
                if (currentRowLength<minMiddleRowLength) {
                    minMiddleRowLength = currentRowLength;
                }//if inside if
            }//if inside if and for
            else {
                currentRowLength = theArray[i].length;
                if (currentRowLength < minEdgeRowLength) {
                    minEdgeRowLength = currentRowLength;
                }//if inside else
            }//else
        }//for
        if (minEdgeRowLength >= minMiddleRowLength) {
            limitLength = minMiddleRowLength;
        }//if
        else { limitLength = minEdgeRowLength+1;
        }
        if (limitLength >= 4) {
            for (int j = 0; j <= limitLength-patternSize; j++) {
```

```
    for (int i = 0; i < patternWidth; i++) {  
        totalSum += theArray[startPosition][i+j+1];  
        totalSum += theArray[startPosition+patternSize-1][i+j+1];  
        // for inside for and if(musterSiteLength)  
        for (int i = 0; i < patternHeight; i++) {  
            totalSum += theArray[startPosition+1+i][j];  
            totalSum += theArray[startPosition+1+i][j+patternSize-1];  
            // for inside for and if (musterSiteHeigth)  
        }  
        //for inside if  
    }  
    startPosition++;  
}  
return totalSum;  
}
```

Beschreibung:

- Die Größe des Musters (2 Zeilen \times 2 Spalten) wird festgelegt;
- Es wird geprüft, ob ab der aktuellen Startposition genügend Zeilen im Array vorhanden sind – also mindestens so viele wie die Musterhöhe. Ist das nicht der Fall, wird nichts weiter ausgeführt;
- Für alle Zeilen, die vom Muster betroffen sind, wird die Länge (Anzahl der Spalten) überprüft. Dabei wird die jeweils kürzeste Zeile gespeichert;
- Falls die kürzeste dieser Zeilen groß genug ist (mindestens so viele Spalten wie die Musterbreite), wird die Verarbeitung fortgesetzt;
- An den Positionen, die dem Muster entsprechen, werden die Zahlen ausgewählt und aufaddiert.

