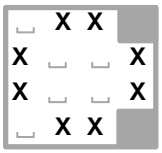
Aufgabe A3.1 Muster addieren

Entpacken Sie das Code-Template `A3x1.zip` und lösen Sie die Aufgabe A3.1.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Vorweg: In den folgenden Array-Darstellungen ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt links oben an".

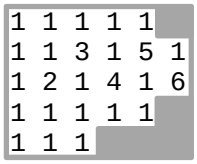
Durchlaufen Sie ein als Parameter gegebenes Array **`long[][]`** mit dem nachfolgendenMuster



und addieren Sie jeweils alle Grundelemente im als Parameter übergebenen Array die jeweils dem obigen Muster (die mit X markierten Felder) genügen.
Achtung! Weder das Muster noch das als Parameter übergebene Array muss echt zweidimensional sein. Wir erinnern uns, in Java gibt es keine echt zweidimensionale Arrays. Es gibt nur eindimensionale Arrays über eindimensionale Arrays.
Es wird zugesichert, dass das Ergebnis im Wertebereich von long liegt – dies müssen sie also nicht überprüfen.

Beispiel:

Das als Parameter übergebene Array:

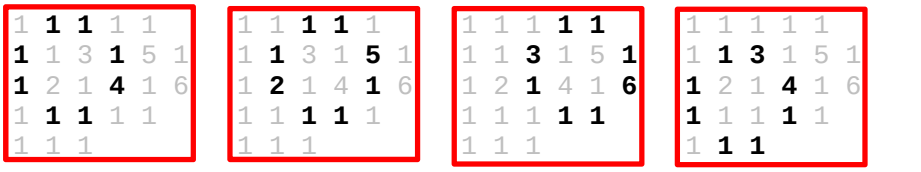


Das Array hat in der ersten Dimension 5 Einträge und in der zweiten Dimension ist die Anzahl der Einträge unterschiedlich (zunächst 5, dann 6, 6, 5 und zuletzt 3).
Das Muster hat in der ersten Dimension 4 und in der zweiten Dimension jeweils 3, 4, 4, 3 Einträge.

Für obiges Beispiel-Array und das zuvor gegebene Muster lautet das Ergebnis:

52

Da:



Aufgabe A3.2 Karten in Kartenmatrix einsortieren

Entpacken Sie das Code-Template `A3x1.zip` und lösen Sie die Aufgabe A3.1.
Nach dem Entpacken finden Sie 2 Packages/Folder vor Klassen vor:

- cards enthält die Spielkarten – fassen Sie dieses Package/diesen Folder nicht an.
- implementingCardMatrix ist das Package/der Folder in dem Sie arbeiten müssen

Im Package/Folder implementingCardMatrix finden Sie drei Klassen vor

- CardProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- Tester unterstützt das Testen von CardProcessor.
- TestFrame startet den Test.

Kleine Kinder benutzen oft zum Sortieren von Karten (auf dem Tisch/Fußboden) eine gedachte Kartenmatrix bei der die Zeilen über die (Karten-)Farben und die Spalten über die (Karten-)Ränge laufen.

Typischer Aufbau dieser Kartenmatrix:

	2	3	4	5	6	7	8	9	10	Bube	Dame	König	Ass
Kreuz	C2	C3	C4	C5	C6	C7	C8	C9	CT	CJ	CQ	CK	CA
Pik	S2	S3	S4	S5	S6	S7	S8	S9	ST	SJ	SQ	SK	SA
Herz	H2	H3	H4	H5	H6	H7	H8	H9	HT	HJ	HQ	HK	HA
Karo	D2	D3	D4	D5	D6	D7	D8	D9	DT	DJ	DQ	DK	DA

Vermutlich ist für Sie die folgende Kartenmatrix für die gegebene Karten-Klasse angenehmer:

	2	3	4	5	6	7	8	9	10	Bube	Dame	König	Ass
Kreuz	C2	C3	C4	C5	C6	C7	C8	C9	CT	CJ	CQ	CK	CA
Karo	D2	D3	D4	D5	D6	D7	D8	D9	DT	DJ	DQ	DK	DA
Herz	H2	H3	H4	H5	H6	H7	H8	H9	HT	HJ	HQ	HK	HA
Pik	S2	S3	S4	S5	S6	S7	S8	S9	ST	SJ	SQ	SK	SA

Schreiben Sie nun eine Methode `generateCardMatrix()`, die eine gewünschte Anzahl Karten vom Kartenstapel zieht und in eine Kartenmatrix einsortiert. Vor dem Befüllen durch die gezogenen Karten soll die Kartenmatrix leer sein. Nach dem Befüllen soll die Kartenmatrix als Ergebnis von der Methode zurückgegeben werden.
Die Kartenmatrix soll als `Card[4][13]`-Array modelliert werden. Den ersten Index bestimmen Sie über die Farbe und den 2.Index über den Rang der jeweiligen einzusortierenden Karte.
Optional soll es möglich sein, die Ausgabe jeder gezogenen Karte unmittelbar nach der Ziehung zu veranlassen.
Die Methode `generateCardMatrix()` muss wie folgt beschaffen sein:

1.Parameter:
Der Kartenstapel von dem die Karten gezogen werden.
2.Parameter:
Die Anzahl der zu ziehenden Karten.
3.Parameter:
Ein Wahrheitswert für die Aufforderung zur Ausgabe der jeweils gezogenen Karte unmittelbar nach der Ziehung. `true` soll die Ausgabe veranlassen und `false` nicht.
Rückgabewert:
Die Kartenmatrix mit den einsortierten Karten.

Schreiben Sie eine weitere Methode `printCardMatrix()`, die als Parameter eine Kartenmatrix entgegen nimmt und ausgibt. In der Ausgabe sollen alle leeren Felder mit "--" gekennzeichnet sein.
Beispielsweise könnte die Ausgabe einer mit 5 Karten befüllten Kartenmatrix wie folgt aussehen:

```
--    --   [C4]   --    --    --    --   [C9]   --    --    --    --    --
--    --    --    --    --    --    --    --    --   [DJ]   --    --    --
--    --    --    --    --    --    --    --    --    --    --    --    --
--    --    --    --   [S6]   --    --    --    --    --    --   [SK]   --
```

Optional muss diese Methode (zur Kontrolle) jede Karte, die gerade gezogen wurde, sofort nach der Ziehung(!) auf dem Bildschirm (mit "print") ausgegeben können.

Optionale zusätzliche Teilaufgabe:
Nun soll `printCardMatrix()` die Reihenfolge der Ränge umdrehen und die Farben in der Reihenfolge Kreuz, Pik, Herz und Karo ausgeben.
Beispielsweise würde die Ausgabe vorheriger Beispiel-Kartenmatrix wie folgt aussehen:

```
--    --    --    --    --   [C9]   --    --    --    --   [C4]   --    --
--   [SK]   --    --    --    --    --    --   [S6]   --    --    --    --
--    --    --    --    --    --    --    --    --    --    --    --    --
--    --    --   [DJ]   --    --    --    --    --    --    --    --    --
```

Die gedachte Kartenmatrix hat also nun folgenden Aufbau:

	ASS	König	Dame	Bube	10	9	8	7	6	5	4	3	2
Kreuz	CA	CK	CQ	CJ	CT	C9	C8	C7	C6	C5	C4	C3	C2
Pik	SA	SK	SQ	SJ	ST	S9	S8	S7	S6	S5	S4	S3	S2
Herz	HA	HK	HQ	HJ	HT	H9	H8	H7	H6	H5	H4	H3	H2
Karo	DA	DK	DQ	DJ	DT	D9	D8	D7	D6	D5	D4	D3	D2

Für 52 Karten muss die Ausgabe wie folgt aussehen:

```
[CA]  [CK]  [CQ]  [CJ]  [CT]  [C9]  [C8]  [C7]  [C6]  [C5]  [C4]  [C3]  [C2]
[SA]  [SK]  [SQ]  [SJ]  [ST]  [S9]  [S8]  [S7]  [S6]  [S5]  [S4]  [S3]  [S2]
[HA]  [HK]  [HQ]  [HJ]  [HT]  [H9]  [H8]  [H7]  [H6]  [H5]  [H4]  [H3]  [H2]
[DA]  [DK]  [DQ]  [DJ]  [DT]  [D9]  [D8]  [D7]  [D6]  [D5]  [D4]  [D3]  [D2]
```

Aufgabe A3.3 Karten sortieren

Entpacken Sie das Code-Template `A3x1.zip` und lösen Sie die Aufgabe A3.1.
Nach dem Entpacken finden Sie 2 Packages/Folder vor Klassen vor:

- cards enthält die Spielkarten – fassen Sie dieses Package/diesen Folder nicht an.
- implementingCardMatrix ist das Package/der Folder in dem Sie arbeiten müssen

Im Package/Folder implementingCardMatrix finden Sie drei Klassen vor

- CardProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- Tester unterstützt das Testen von CardProcessor.
- TestFrame startet den Test.

Vorweg: Nutzen Sie `varargs` um beliebig viele Karten sowohl einzeln als auch als Array in den eingeforderten Methoden entgegen nehmen zu können.

A3.3 baut auf Aufgabe A3.2 auf. In A3.3 soll die Lösung von A3.2 wie folgt modifiziert werden.

Schreiben Sie eine Methode `sort()`, die beliebig viele (jedoch niemals mehr als 52) Karten ohne Doppelte entgegen nimmt, die so gegeben Karten sortiert und als sortiertes Array über Karten zurückgibt.
Für das Sortieren **müssen** Sie die Kartenmatrix aus A3.2 nutzen. Sortieren Sie zunächst die übergebenen Karten in die Kartenmatrix ein. Stellen Sie nun das geforderte sortierte Karten-Array durch geeignetes Auslesen der Kartenmatrix zusammen und geben es dann als Ergebnis zurück.
Signatur – Deck ????

Schreiben Sie eine Methode `printCards()`, die beliebig viele Karten entgegen nimmt und auf dem Bildschirm einzeilig ausgibt.

Optionale zusätzlich Teilaufgabe:
Schreiben Sie eine Methode `sortMyWay()`, die wieder beliebig viele Karten (sowohl einzeln als auch als Array) entgegen nimmt, die so gegeben Karten sortiert und dann als sortiertes Array über Karten zurückgibt.
Diesmal dürfen Sie die Kartenmatrix aus A3.2 bzw. A3.3 nicht nutzen.
Es geht hierbei nicht darum, dass Sie das effizienteste Sortierprogram schreiben. Mit dem Entwickeln von Sortier-Algorithmen lassen sich die bisher besprochenen Themen gut einüben. Wichtig ist, dass Sie selbst verstehen, was Sie tun und eine klare Idee klar umsetzen.

Aufgabe A3.4 (Wort-)Palindrom erkennen (`char[]` basiert)

Entpacken Sie das Code-Template `A3x4.zip` und lösen Sie die Aufgabe A3.4.
Nach dem Entpacken finden Sie 3 Klassen vor:

- PalindromeTester ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- TestFrame ist eine Möglichkeit Ihre Lösung anzustarten und interaktiv zu testen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.

Achtung Studenten mit Vorkenntnissen: Die Pflicht-Lösung für diese Aufgabe muss iterativ sein! Rekursive Lösungen werden nicht akzeptiert bzw. (wenn überhaupt) nur diskutiert, wenn bereits eine iterative Lösung akzeptiert wurde. Gemäß Vorlesungswissen ist Rekursion unbekannt und alles was wir kennen iterativ.

Schreiben Sie eine Methode `isPalindrome()`, die für einen als Parameter übergebenes `char[]` überprüft, ob die im char[] enthaltenen Zeichen ein (Wort-)Palindrom bilden und einen entsprechenden Wahrheitswert zurückgibt. Der Rückgabewert ist also `true` für "ist ein Palindrom" oder `false` für "ist kein Palindrom".

Unter https://de.wikipedia.org/wiki/Palindrom finden Sie eine Erklärung, was ein Wortpalindrom ist und einige Beispiele.

Eine mögliche Unterscheidung zwischen Klein- und Groß-Buchstaben ist freigestellt.
Als Vereinfachung ist es ok, wenn Kleinbuchstaben als verschieden von Großbuchstaben gewertet werden.

Freiwillige Zusatzaufgaben

Es folgen freiwillige Zusatzaufgaben. D.h. diese Aufgabe ist freiwillig ;-).
Wenn Sie diese freiwillige Zusatzaufgabe freiwillig lösen, dann haben Sie den "Gewinn", dass Sie mehr geübt haben und dass Sie Ihre Lösung für diese freiwillige Zusatzaufgabe im Labor besprechen können (sofern Zeit ist – Pflichtaufgaben haben Vorrang).

Freiwillige Zusatzaufgabe Z3.1 Lage zweier Rechtecke zueinander bestimmen

Entpacken Sie das Code-Template `Z3x1.zip` und lösen Sie die Aufgabe Z3.1.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ContactAnalyzer ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- UnitTestFrame ist ein einfacher Unit-Test.

Die Aufgabe:
In einem *kartesischen zweidimensionalen* Koordinatensystem ist ein *achsenparalleles* Rechteck durch die Koordinaten von zwei gegenüberliegenden Eckpunkten P(p_x,p_y) und Q(q_x,q_y) bestimmt. *Bezüglich der Lage von P und Q zueinander darf es keine Einschränkungen in Ihrem Programm geben! Jedoch dürfen Sie selbstverständlich basierend auf P und Q andere Punkte oder andere Dinge berechnen, die Sie "zum Arbeiten" verwenden.* Zur Vereinfachung sind alle Kantenlängen >=1 und alle Koordinaten ganzzahlig.
Bestimmen sie für 2 solche achsenparallele Rechtecke, die sich an beliebigen Positionen in einem *kartesischen* Koordinatensystem befinden, die Lage zueinander.
Dabei wird unterschieden zwischen:

aligned — Der Durchschnitt der beiden Rechtecke ist eine Linie.
Alle gemeinsamen Punkte liegen auf einer Linie der Länge > 0.

contained — Der Durchschnitt der beiden Rechtecke ist genau eines der beiden Rechtecke. Ein Rechteck ist also echt im anderen enthalten. D.h. auch, dass die beiden Rechtecke verschieden sind.
Jeder Punkt eines der beiden Rechtecke ist auch ein Punkt des jeweils anderen Rechtecks - aber nicht umgekehrt.

disjoint — Die beiden Rechtecke berühren sich nicht.
Sie haben keine gemeinsamen Punkte.

intersecting — Der Durchschnitt der beiden Rechtecke ist ein neues drittes Rechteck, das von den ersten beiden verschieden ist und eine Fläche > 0 enthält.
Die beiden Rechtecke haben gemeinsame Punkte, aber jedes der beiden Rechtecke hat auch Punkte, die das andere nicht hat.

same — Die Rechtecke sind gleich, sie haben die gleiche Lage und die gleiche Größe.
Der Durchschnitt der beiden Rechtecke enthält alle Punkte der beiden Rechtecke.

touching — Der Durchschnitt der beiden Rechtecke ist ein Punkt.
Die beiden Rechtecke haben genau einen gemeinsamen Punkt.

Beispiel für 2 Rechtecke:

```
int[][] r1 = {
               { 1, 2 },  // Eckpunkt P(px,py) vom 1.Rechteck
               { 7, 5 }   // Eckpunkt Q(qx,qy) vom 1.Rechteck
             };

int[][] r2 = {
               { 4, 9 },  // Eckpunkt P(px,py) vom 2.Rechteck
               { 6, 3 }   // Eckpunkt Q(qx,qy) vom 2.Rechteck
             };
```

Schreiben Sie nun eine Methode **`computeRelation`**, die zwei Rechtecke (jeweils vom Typ **`int[][]`**) als Parameter entgegen nimmt und als Ergebnis einen **`String`** abliefert, der einen Wert entsprecht der obigen Anforderung enthält.

Freiwillige Zusatzaufgabe Z3.3 (Wort-)Palindrom erkennen (`String` basiert)

Entpacken Sie das Code-Template `Z3x3.zip` und lösen Sie die Aufgabe Z3.3.
Nach dem Entpacken finden Sie 4 Klassen vor:

- PalindromeTester ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- TestFrame ist eine Möglichkeit Ihre Lösung anzustarten und interaktiv zu testen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Schreiben Sie eine Methode `isPalindrome()`, die für einen als Parameter übergebenen String überprüft, ob der String ein (Wort-)Palindrom ist und einen entsprechenden Wahrheitswert zurückgibt. Der Rückgabewert ist also `true` für "ist ein Palindrom" oder `false` für "ist kein Palindrom".
Unter https://de.wikipedia.org/wiki/Palindrom finden Sie eine Erklärung, was ein Wortpalindrom ist und einige Beispiele.

Mit **length()** lässt sich die Länge des Strings bestimmen.
Mit **charAt()** lässt sich das Zeichen an einer bestimmten Position im String bestimmen. Analog zu einem Array hat das erste Zeichen die Position 0 und das letzte Zeichen die Position length()-1.

Beispiel-Code:

```
String text = "lalilu";
char zeichen = text.charAt(1);          // "text.charAt(1)"   liefert 'a'
int textLength = text.length();         // "text.length()"    liefert 6
```

Freiwillige Zusatzaufgabe Z3.4 Muster addieren

Entpacken Sie das Code-Template `Z3x4.zip` und lösen Sie die Aufgabe Z3.4.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Analog zur Pflichtaufgabe A3.1 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster

```
X X X X
X     X
X     X
X X X X
```

und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt links oben an".

Freiwillige Zusatzaufgabe Z3.5 Muster addieren

Entpacken Sie das Code-Template `Z3x5.zip` und lösen Sie die Aufgabe Z3.5.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Analog zur Pflichtaufgabe A3.1 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster

```
X X X X
X X X
X X
X
```

und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt links oben an".

Freiwillige Zusatzaufgabe Z3.6 Muster addieren

Entpacken Sie das Code-Template `Z3x6.zip` und lösen Sie die Aufgabe Z3.6.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Analog zur Pflichtaufgabe A3.1 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster

```
X       X
  X   X
    X
  X   X
X       X
```

und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt links oben an".

Freiwillige Zusatzaufgabe Z3.7 Muster addieren

Entpacken Sie das Code-Template `Z3x7.zip` und lösen Sie die Aufgabe Z3.7.
Nach dem Entpacken finden Sie 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Analog zur Pflichtaufgabe A3.1 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster

```
    X
  X   X
X       X
  X   X
    X
```

und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt links oben an".