Denken Sie an "Aufgabe V2.3 Vorbereitungsaufgabe: char[] versus String Demonstrator" von AZ#2 (bzw. Aufgabenzettel Nr.2). Arrays waren inzwischen Thema und wurden bereits abschießend behandelt.

Aufgabe A3.1 (Wort-)Palindrom erkennen (`char[]` basiert)

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `a3x1.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt a3x1 lösen.
Nach dem Entpacken finden Sie im Package `simpleCharacterArrayBasedPalindromeTester` 4 Klassen vor:

- PalindromeTester ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- UnitTestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und zu testen.
- TestFrameAndStarter ist eine Möglichkeit Ihre Lösung anzustarten und interaktiv zu testen.
- ProposalForAlternativeTestFrameAndStarter 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 ist 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 bzw. als Vereinfachung ist es ok, wenn Kleinbuchstaben als verschieden von Großbuchstaben gewertet werden.

Aufgabe A3.2 Karten-Array In-Place sortieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `a3x2.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt a3x2 lösen.
Nach dem Entpacken finden Sie 3 Packages/Folder vor Klassen vor:

- cards enthält die Spielkarten – fassen Sie dieses Package/diesen Folder nicht an.
- cardProcessor ist das Package/der Folder in dem Sie arbeiten müssen.
- stuffBeginnersDontHaveToUnderstand muss Sie nicht weiter interessieren.

Im Package/Folder cardProcessor finden Sie zwei Klassen vor

- CardProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- TestFrameAndStarter startet "den Test" in Dialog-Form.
- ProposalForAlternativeTestFrameAndStarter bietet die Möglichkeit eigene Tests zu schreiben.

Vorweg: In der Vorlesung wurde die Idee einer Kartenmatrix als Hilfsmittel zum Sortieren von Karten besprochen. Nutzen Sie das in der Vorlesung erworbene Wissen zur Lösung der Aufgabe.
Wir erinnern uns aus der Vorlesung an:
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.

Aufgaben-Kern: Sortieren Sie in einem Array gegebene Karten.

Zur Lösung der Aufgabe dürfen Sie nicht unnötig "Dinge" doppelt tun. In diesem Zusammenhang daher der Tipp: Vermutlich ist für Sie die folgende Kartenmatrix für die gegebene Karten-Klasse am angenehmsten:

	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: `Card[][] generateCardMatrix( Card[] givenCards )`, die die in einem Array gegebenen Karten (bis zu max. 52 unterschiedliche Karte) in eine Kartenmatrix einsortiert, die für diesen Zweck zuvor zu erzeugen ist. Nach dem Befüllen soll die Kartenmatrix als Ergebnis von der Methode zurückgegeben werden.
Die Kartenmatrix soll als `Card[4][13]`-Array modelliert werden. Mit dem 1.Index bestimmen Sie über die Farbe und mit dem 2.Index über den Rang die Position der jeweiligen einzusortierenden Karte in der Kartenmatrix.

Schreiben Sie eine Methode `void sortCards( Card[] cardsToBeSorted )`, die beliebig viele (jedoch niemals mehr als 52) unterschiedliche Karten in Form eines Arrays entgegen nimmt und die so gegeben Karten In-Place sortiert. D.h. nach dem Aufruf der Methde sortCards soll das als Parameter übergebene Karten-Array sortiert sein und zwar konsequent absteigend mit 1.Priorität nach den Rängen und mit 2.Priorität nach den Farben. Sofern 4 Asse und 4 Könige sowie eine Karo-2 in dem als Parameter übergeben Array enthalten sind, dann sind die ersten 8 Karten (von der ersten bis zur achten Karte) CA, SA, HA, DA, CK, SK, HK, DK und die letzte Karte D2.
Die Methode `sortCards()` soll intern (also in ihrem Methoden-Rumpf) die zuvor eingeforderte Methode `generateCardMatrix()` nutzen.

Optionale(!) zusätzlich Teilaufgabe:
Schreiben Sie eine Methode `sortCardsMyWay()`, die wieder beliebig viele viele (jedoch niemals mehr als 52) unterschiedliche Karten entgegen nimmt und die so gegeben Karten In-Place sortiert.
Diesmal dürfen Sie die Kartenmatrix jedoch nicht nutzen.
Es geht hierbei nicht darum, dass Sie ein besonders effizientes Sortierprogram schreiben. Effiziente Sortierprogramme sind Thema in Algorithmen und Datenstrukturen (typischer Weise im 3.Semester). Mit dem Entwickeln von eigenen Sortier-Algorithmen lassen sich die bisher besprochenen Themen gut einüben. Wichtig ist, dass Sie selbst verstehen, was Sie tun und eine klare Idee erfolgreich umsetzen.
Tipp: Eine Methode zum Vergleichen bzw. Ordnen von 2 Karten ist bestimmt hilfreich. Etwa
`int compare( Card a, Card b )` mit
`compare( x, y ) < 0` falls Karte x vor Karte y kommt
`compare( x, y ) > 0` falls Karte y vor Karte y kommt
`compare( x, y ) == 0` falls die Karten x und y gleich sind. Dieser Fall wird wegen der Zusicherung nicht benötigt, macht aber ansonsten Sinn.

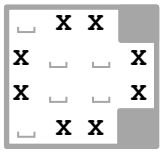
Aufgabe A3.3 Muster addieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `a3x3.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt a3x3 lösen oder A3xX sofern sie auch die Zusatzaufgaben Z3.1 bis Z3.4 angehen.
Nach dem Entpacken finden Sie u.a. 3 Klassen vor:

- ArrayProcessor ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- ProposalForAlternativeTestFrameAndStarter soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrameAndStarter 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.
- Weiterhin finden Sie auch den Code (die UnitTestFrames) für die Zusatzaufgaben vor

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". Dort sind die "Koordinaten (0,0)".

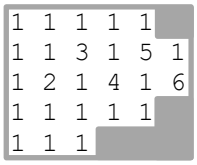
Durchlaufen Sie ein als Parameter gegebenes Array **`long[][]`** mit dem nachfolgenden Muster



- das Muster muss zum jeweiligen Additions-Zeitpunkt in das als Parameter übergebene Array passen -
und addieren Sie jeweils alle Grundelemente im als Parameter übergebenen Array, die jeweils dem obigen Muster (die mit X markierten Felder) genügen. Die aus der Addition resultierende Summe ist das Ergebnis.
Wie Sie das Muster in Ihrem Code umsetzen/implementieren - ob z.B. als Array etwa `boolean[][]` (oder anders) - ist Ihnen freigestellt.
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.
Sollte es gemäß Aufgabenstellung keine Werte/Grundelemente zum Aufaddieren geben, so ist das Ergebnis 0.

Beispiel:

Das als Parameter übergebene Array soll wie folgt aussehen:

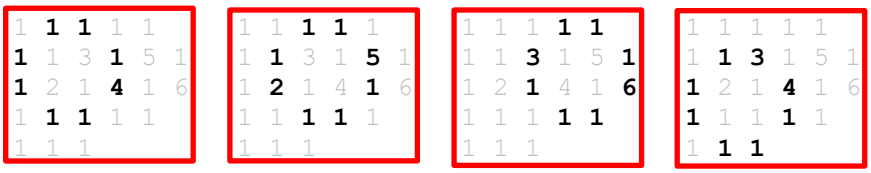


Das Array hat in der ersten Dimension 5 Einträge und in der zweiten Dimension ist die Anzahl der Einträge unterschiedlich (zunächst 5, dann 6, 6, 5 und zuletzt 3).
Das Muster hat in der ersten Dimension 4 und in der zweiten Dimension jeweils 3, 4, 4, 3 Einträge.

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

52

Da das Muster in diesem Beispiel an vier Stellen in das Array passt:



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 Muster addieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `z3x1.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z3x1 lösen.
Nach dem Entpacken finden Sie u.a. die 3 bereits bei A3.3 beschriebenen Klassen vor.

Analog zur Pflichtaufgabe A3.3 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 Z3.2 Muster addieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `z3x2.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z3x2 lösen.
Nach dem Entpacken finden Sie u.a. die 3 bereits bei A3.3 beschriebenen Klassen vor.

Analog zur Pflichtaufgabe A3.3 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
```

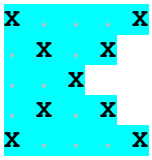
und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt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

Freiwillige Zusatzaufgabe Z3.3 Muster addieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `z3x3.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z3x3 lösen.
Nach dem Entpacken finden Sie u.a. die 3 bereits bei A3.3 beschriebenen Klassen vor.

Analog zur Pflichtaufgabe A3.3 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster



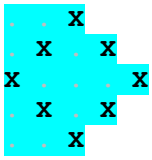
und addieren Sie jeweils alle Felder des Musters (die mit X markierten) Felder).
Achtung! Das als Parameter übergebene Array muss nicht echt Zwei-Dimensional sein.
Wieder gilt: In der Array-Darstellung ist die erste Dimension als Y-Achse und die zweite Dimension als X-Achse dargestellt. Das Array "fängt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

Freiwillige Zusatzaufgabe Z3.4 Muster addieren

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `z3x4.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z3x4 lösen.
Nach dem Entpacken finden Sie u.a. die 3 bereits bei A3.3 beschriebenen Klassen vor.

Analog zur Pflichtaufgabe A3.3 sollen Sie "ein Muster addieren". Jedoch ist es für diese Aufgabe ein anderes Muster.

Durchlaufen Sie ein als Parameter gegebenes Array mit dem Muster



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

Sie haben jetzt A3.3, Z3.1, Z3.2, Z3.3 und Z3.4 gelöst. U.U. mit wirklich individuellen Lösungen im Sinne von fünf unterschiedlichen Klassen/ArrayProcessoren. Für Anfänger wäre das keine Schande! Jedoch in dieser Zusatzaufgabe nun: Bekommen Sie das auch mit einem(!) geeignet parameterisierten ArrayProcessor hin, der die Muster aus A3.3, Z3.1, Z3.2, Z3.3 und Z3.4 und beliebige andere/weitere Muster unterstützt?

Freiwillige Zusatzaufgabe Z3.6 (Wort-)Palindrom erkennen (`String` basiert)

Vorweg: Erinnern Sie sich an Ihre Erkenntnisse aus V2x3.

An der abgesprochenen "Ablagestelle" finden Sie die Zip-Datei `z3x6.zip`. Entpacken Sie die Zip-Datei, binden Sie den gestellten Code in Eclipse ein und beachten Sie dabei, dass Sie diese Aufgabe in einem eigenen Eclipse-Projekt z3x6 lösen.

Diese Aufgabe ist analog zu A3.1. Der Unterschied besteht darin, dass das zu untersuchende Wort **nicht** als char[] sondern als String übergeben wird.
Achtung! Der Einsatz von Konvertierungsmethden wie `toCharArray()` untergräbt den Lerneffekt.

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 Z3.7 Fraction

Sobald die Fraction-Klasse in der Vorlesung besprochen wurde, implementieren Sie diese. Kürzen Sie die Brüche automatische nach jeder Operation (auf den jeweiligen Bruch und wie in der Vorlesung vorgeschlagen mit einer Metode reduce()). Dafür müssen Sie u.a. eine GGT-Funktion implentieren.
Testen Sie Ihren anschließend Ihren Code.