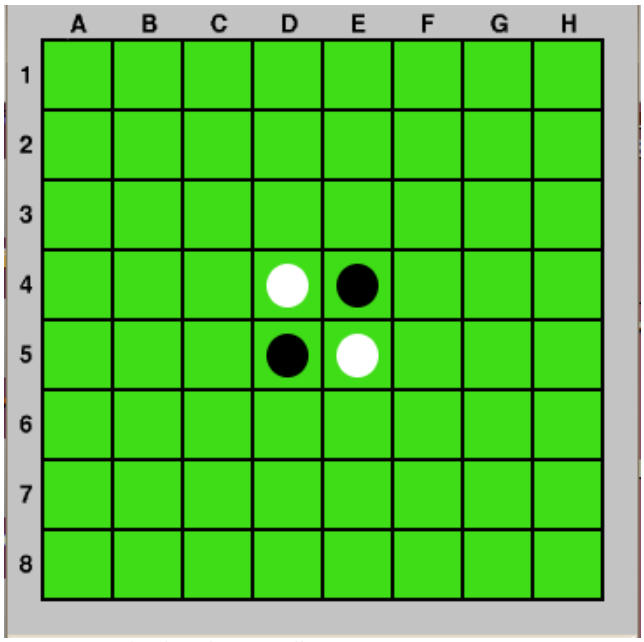
Othello

In dieser Abschlussaufgabe sollen Sie *Othello* implementieren.

Othello ist ein Brettspiel für zwei Spieler (im Folgenden:„Spieler B“ für *black/schwarz* und „Spieler W“ für *white/weiß*). Gespielt wird auf einem Brett, das m Felder breit und n Felder hoch ist. Im Folgenden werden die Regeln am Beispiel m=n=8 erklärt, Ihre Implementierung soll aber allgemeinere Spielbrettgrößen zulassen.

Jeder Spielstein hat zwei Seiten, von denen eine schwarz (zu Spieler B gehörend) und eine weiß (zu Spieler W gehörend) ist. Die beiden Spieler legen abwechselnd Spielsteine mit der eigenen Farbe nach oben auf das Brett. Das Spiel endet, sobald keiner der Spieler mehr einen Zug machen kann. Ziel des Spiels ist es, am Ende möglichst viele Steine mit der eigenen Farbe nach oben auf dem Brett zu haben.

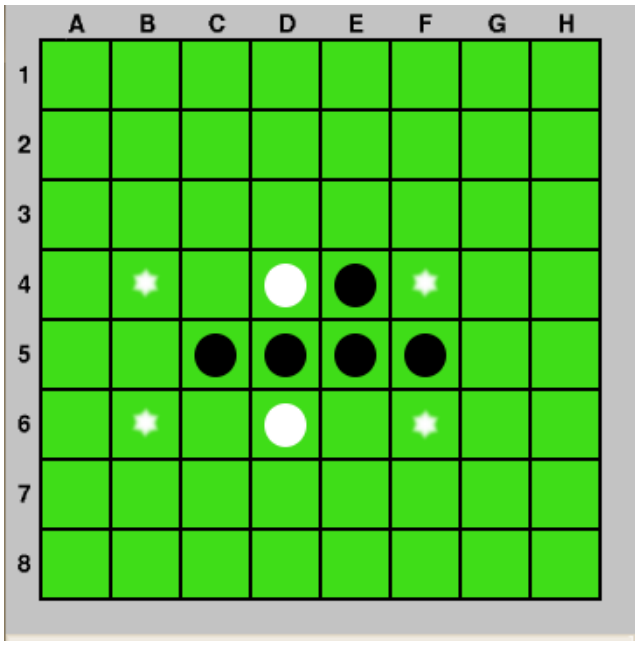
Es wird von folgender Anfangsstellung ausgegangen:



Standard-Anfangsstellung

Ein Spieler macht einen Zug, indem er einen Stein mit der eigenen Farbe nach oben auf ein freies Feld legt, das horizontal, vertikal oder diagonal an ein bereits belegtes Feld angrenzt. Anschließend wird jede horizontale, vertikale oder diagonale Linie von gegnerischen Steinen, die durch den neuen Stein und einen beliebigen anderen eigenen Stein begrenzt wird, durch Umdrehen in eigene Steine umgewandelt. Spielzüge, die zu keinem Umdrehen von gegnerischen Steinen führen, sind nicht erlaubt.

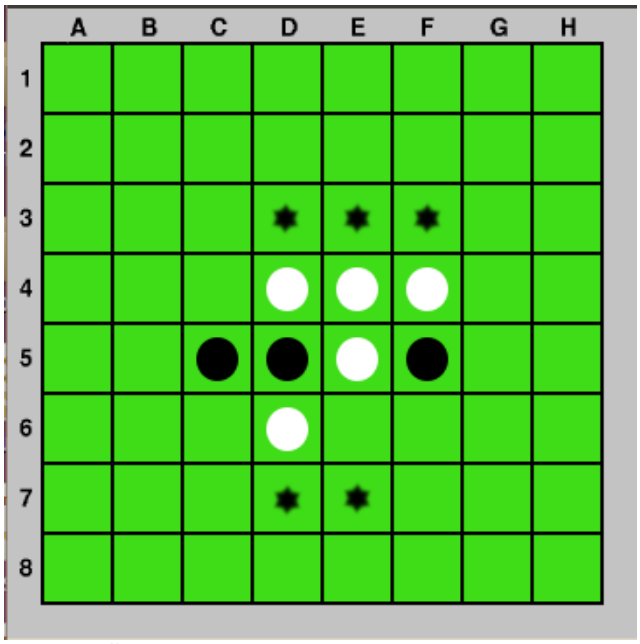
Betrachten Sie die folgende Situation, in der Spieler W am Zug ist (die weißen Sterne deuten die möglichen Züge von Spieler W an):



Stellung 1

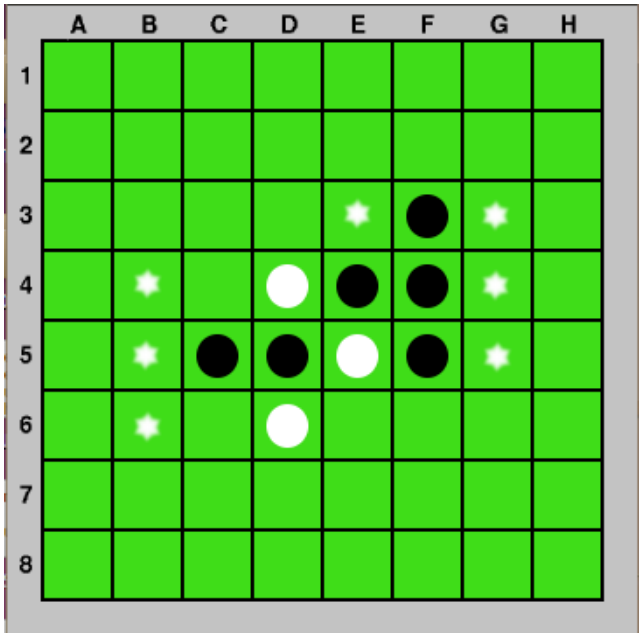
Setzt Spieler W nun seinen nächsten Stein an die Position F4, so werden alle Steine von Spieler B, die auf einer Linie zwischen F4 und einem beliebigen anderen Stein von Spieler W sind, umgedreht. In diesem Fall sind dies die Steine auf den Positionen E4 (horizontale

Linie zwischen D4 und F4) und E5 (diagonale Linie zwischen F4 und D6). Die Folgestellung sieht dann so aus (die schwarzen Sterne deuten die möglichen Züge von Spieler B an):



Stellung 2

Beachten Sie, dass neben horizontalen und vertikalen Linien auch Diagonalen betrachtet werden. Insbesondere hat Spieler B in Stellung 2 die Möglichkeit, einen Stein auf F3 zu legen, weil auf der Diagonalen zwischen F3 und D5 ein weißer Stein liegt, der umgedreht werden kann. Auch auf der vertikalen Linie zwischen F3 und F5 liegt ein weißer Stein. Setzt Spieler B also seinen nächsten Stein auf F3, werden sowohl der Stein auf F4 als auch der Stein auf E4 umgedreht:

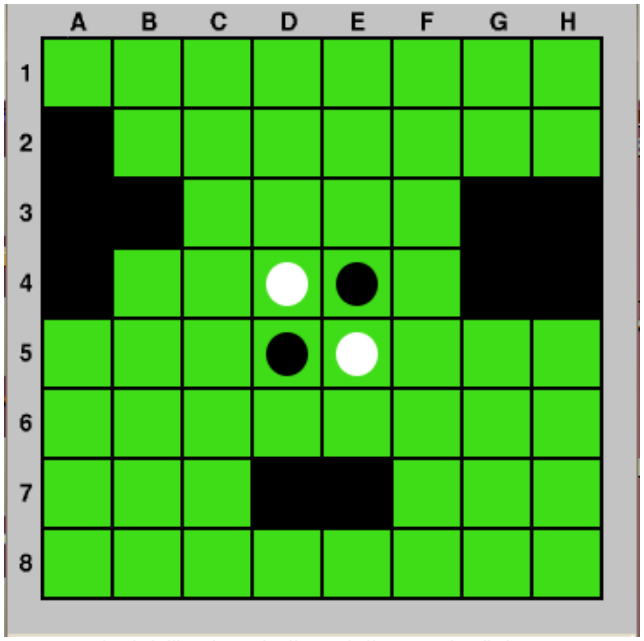


Stellung 3

Ein Othello-Spiel endet, wenn keiner der Spieler einen Stein setzen kann. Es gewinnt dann der Spieler, der mehr Steine mit der eigenen Seite nach oben auf dem Brett hat. Liegen am Ende von jeder Sorte gleichviele Steine auf dem Brett, geht das Spiel unentschieden aus.

Ihre Aufgabe ist es nun, ein Othello-Spiel zu implementieren und eine Kommandoschnittstelle bereitzustellen, über die zwei Spieler miteinander spielen können. Sie müssen keinen Computergegner implementieren. Außerdem soll Ihr Programm die folgenden Modifikationen der normalen Othello-Variante unterstützen:

- Die Höhe und Breite des Spielbretts sollen vom Benutzer einstellbar sein. Der Einfachheit halber sind dabei nur gerade Zahlen zwischen 2 und 26 (für die Breite) bzw. 2 und 98 (für die Höhe) möglich. Die Standardausgangsstellung ist dann als die Stellung zu verstehen, in der die vier mittleren Felder so belegt sind wie im ersten Bild gezeigt.
- Es soll möglich sein, Bereiche zu definieren, in denen das Spielfeld „Löcher“ hat.



Beispiel für ein Othello-Spielbrett mit Löchern

In diesen Bereichen dürfen keine Steine gesetzt werden. Felder innerhalb von solchen Bereichen unterbrechen, wie freie Felder, Linien zwischen Steinen derselben Farbe.

Bitte beachten Sie:

- Da wir automatische Tests dieser Schnittstelle machen, müssen die Ausgaben Ihrer Shell **exakt** den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Geben Sie auch keine zusätzlichen Informationen aus. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung zu testen, und verwenden Sie das Forum, um Unklarheiten zu klären.
- Um Tests zu erleichtern, soll es möglich sein ein Spiel mit beliebigen Stellungen zu beginnen und vorzeitig abzubrechen. Details dazu finden Sie in den Anforderungsbeschreibungen der Befehle.

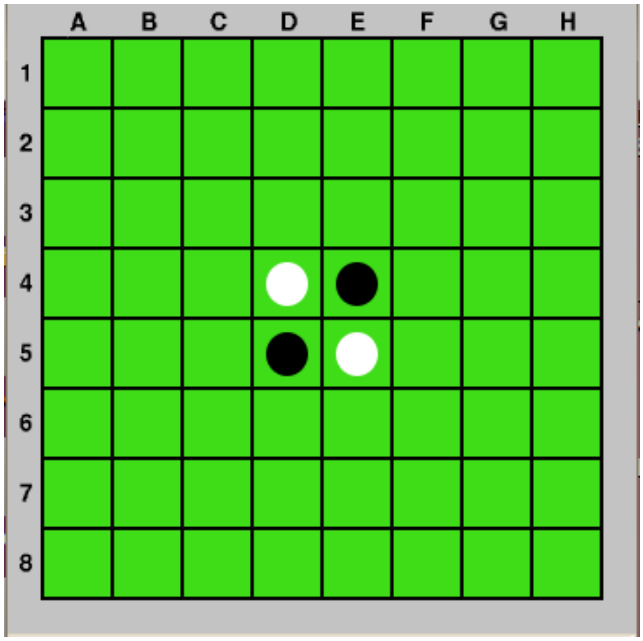
Formate der Ein- und Ausgaben

Im folgenden werden Benutzereingaben **`im fetten Schreibmaschinenstil`** dargestellt und die Ausgaben des Programms `im normalen Schreibmaschinenstil`.

- Eine **Position** besteht immer aus einer Spalte und einer Zeile. Eine Spalte wird durch einen einzelnen Großbuchstaben dargestellt, wobei die erste Spalte stets mit `A` bezeichnet wird. Ein Zeilenbezeichner ist eine Zahl zwischen 1 und 99.
- Ein **Rechteck** wird durch zwei durch einen Doppelpunkt getrennte Positionen dargestellt. Die erste Position gibt dabei die obere linke und die zweite Position die untere rechte Ecke des Rechtecks an. Beispielsweise ist `A3:B4` ein Rechteck, welches aus den Feldern `A3,A4,B3,B4` besteht.

  Es ist erlaubt, dass die beiden Ecken eines Rechtecks übereinstimmen, in diesem Fall umfasst das Rechteck genau ein Feld.
- Eine **Stellung** durch die Liste der Zeilen des Spielfeldes dargestellt. Eine Zeile darf die Zeichen `-` (für "freies Feld"), `B` (für "von Spieler B (black) besetzt"), `W` (für "von Spieler W (white) besetzt") oder `#` (für "Loch") enthalten. In einer Stellung dürfen keine Leerzeichen vorkommen und jede Zeile hat dieselbe Länge. Je zwei aufeinander folgende Zeilen werden durch ein Trennzeichen getrennt. Zum Einlesen einer Stellung verwenden Sie bitte Kommata als Trennzeichen, und zur Ausgabe Zeilenvorschubszeichen(`\n`).

  Beispielsweise soll der Befehl



durch

```
newGame 8 8 --------,--------,--------,---WB---,---BW---,--------,--------,--------
```

ein neues Spiel der Größe 8x8 mit der Standard-Anfangsstellung starten. Die Ausgabe dieser Stellung sieht folgendermaßen aus:

```
--------
--------
--------
---WB---
---BW---
--------
--------
--------
```

Sie dürfen sich auch eine anschaulichere Ausgabe überlegen und zusätzlich umsetzen. Für die automatischen Tests ihres Programms wird aber auf das eben beschriebene Format zurückgegriffen.

- Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung **muss** aber mit `"Error! "` beginnen und darf keine Zeilenumbrüche enthalten.

Die Kommandoschnittstelle

Sie sollen auch eine Shell implementieren, über die ein Spiel gestartet und gespielt werden kann.

Der Prompt der Shell ist

```
othello>
```

Ein möglicher Programmablauf sieht so aus:

```
othello> newGame 8 8
othello> move D6
Move not possible.
othello> move E6
othello> print
--------
--------
--------
---WB---
---BB---
----B---
--------
--------
turn: white
othello> newGame
```

```
Error! There is already an active game.
othello> possibleMoves
Possible moves: D6,F4,F6
othello> move D6
othello> abort
Game has ended in a draw.
othello> move A1
Error! No active game.
othello> quit
```

Ihre Shell muss mindestens folgende Befehle kennen:

newGame <Breite> <Höhe> [<Stellung>]

Startet ein neues Spiel, in dem Spieler B den ersten Zug hat. Die Parameter haben folgende Bedeutung:

- **<Breite>** ist eine gerade Zahl zwischen 2 und 26, die die Breite des Spielbretts angibt.
- **<Höhe>** ist eine gerade Zahl zwischen 2 und 99, die die Höhe des Spielbretts angibt.
- **<Stellung>** ist ein optionales Argument, mit dem die Anfangsstellung des neuen Spiels bestimmt werden kann. Wird dieses Argument weggelassen, so wird die normale Anfangsstellung in der angegebenen Spielfeldgröße verwendet. Ihr Programm muss nicht prüfen, ob es sich bei der angegebenen Stellung um eine von der normalen Anfangsstellung erreichbare Stellung handelt. Kann Spieler B keinen Zug machen, so soll dies direkt behandelt werden, ebenso wenn es sich bei der angegebenen Stellung um eine Endstellung handelt, in der kein Spieler einen Zug machen kann. Näheres dazu entnehmen Sie der Beschreibung des Befehls **move**.

Beachten Sie außerdem:

- Es kann kein Spiel gestartet werden, wenn momentan ein noch nicht beendetes Spiel läuft. In diesem Fall soll die Eingabe von **newGame** zu einem Fehler führen.
- Wird die initiale Stellung mit angegeben, so muss diese mit den Breiten- und Höhenangaben des Spielbretts kompatibel sein. Es müssen also in der Stellung genau so viele Zeilen angegeben werden, wie von der Höhe angegeben und in jeder Zeile müssen genauso viele Zeichen stehen, wie die Breite angibt.

hole <Rechteck>

Fügt dem Spielbrett den durch **<Rechteck>** definierten Lochbereich hinzu. **<Rechteck>** beschreibt dabei ein gültiges Rechteck innerhalb der Spielbrettgrenzen, in dem sich keine Steine befinden. Ist das Rechteck ungültig oder befinden sich dort irgendwo Steine, so wird eine Fehlermeldung ausgegeben und das Spielbrett bleibt unberührt. Handelt es sich um ein gültiges Rechteck, so werden alle darin enthaltenen Felder als Loch markiert. Bestehende Lochbereiche bleiben dabei unberührt, d.h. der neue Lochbereich wird den bestehenden Lochbereichen hinzugefügt. Beachten Sie, dass ein **hole**-Befehl nur dann abgesetzt werden darf, wenn ein Spiel aktiv ist, in dem noch kein gültiger **move**-Befehl abgesetzt wurde. In allen anderen Situationen wird eine Fehlermeldung ausgegeben.

move <Position>

Wir gehen davon aus, dass Spieler B am Zug ist. Ist Spieler W am Zug, so soll der Befehl Entsprechendes leisten.

Bei diesem Befehl wird versucht, den nächsten Stein von Spieler B auf die gegebene Position zu setzen. Dabei ist folgendes zu beachten:

- Ist der Zug nicht möglich, wird

  ```
  Move not possible.
  ```
  ausgegeben und Spieler B ist immer noch am Zug.

- Ist der Zug möglich, so wird der Zug ausgeführt. Anschließend verhält sich das Programm folgendermaßen:
  - Kann Spieler W einen Zug machen, ist dieser am Zug und es geht normal weiter.
  - Kann Spieler W keinen Zug machen und Spieler B hat Möglichkeiten, dann wird

    ```
    White passes.
    ```

    ausgegeben und Spieler B ist am Zug.

  - Kann keiner der beiden Spieler einen Zug machen, so ist das Spiel vorbei. In diesem Fall wird, falls es einen Gewinner gibt,

    ```
    Game Over! <Z> has won (<A>:<B>)!
    ```

    ausgegeben, wobei <Z> der Gewinner des Spiels ist, <A> die Zahl der Steine des Gewinners auf dem Spielbrett und <B> die Zahl der Steine des Verlierers. Geht das Spiel unentschieden aus, so wird

    ```
    Game has ended in a draw.
    ```

    ausgegeben.

print
Gibt die aktuelle Stellung und den Spieler aus, der am Zug ist. Die aktuelle Stellung wird dabei zeilenweise ausgegeben und in

der letzten Zeile wird

```
turn: <aktueller Spieler>
```

ausgegeben.

Wurde zum Beispiel gerade ein neues Spiel mit Spielbrettgröße 8x8 in der Standard-Anfangsstellung gestartet, so führt die Eingabe von **print** zur Ausgabe von

```
--------
--------
--------
---WB---
---BW---
--------
--------
--------
turn: black
```

abort
Bricht das aktuelle Spiel ab. Es wird so vorgegangen, als ob das Spiel regulär beendet worden ist, also so, als ob gerade ein Zug ausgeführt worden wäre und nun keiner der Spieler mehr einen Zug machen kann. Beachten Sie hierzu die Beschreibung des Befehls **move**.

possibleMoves
Zeigt die möglichen Züge des aktuellen Spielers an. Genauer wird `Possible moves:` gefolgt von einer sortierten, komma-separierten Liste (ohne Leerzeichen) der möglichen Positionen ausgegeben, auf die der aktuelle Spieler seinen nächsten Stein setzen kann. Die möglichen Positionen sollen aufsteigend zuerst nach Spalte und dann nach Zeile sortiert sein, d.h. sind die Positionen C4, B6, B5 und D4 möglich, so sieht die Ausgabe folgendermaßen aus:
`Possible moves: B5,B6,C4,D4`

quit
Beendet das Programm.

Zu jedem Zeitpunkt soll höchstens ein Spiel aktiv sein. Weiter sollen die folgenden Anforderungen umgesetzt werden:

- Am Anfang ist kein Spiel aktiv.
- Durch Eingabe des Befehls **newGame** wird ein neues Spiel gestartet, wenn gerade kein Spiel aktiv ist. Dieses neue Spiel ist dann aktiv. Ist bereits ein Spiel aktiv, so führt die Eingabe von **newGame** zu einem Fehler.
- Ein Spiel endet in einem der folgenden Fälle:
  - Kein Spieler kann mehr einen Zug machen.
  - Das Spiel wird abgebrochen (durch Eingabe von **abort**).

  Nach Beendigung des Spiels ist kein Spiel mehr aktiv.

- Die Befehle **move**, **print**, **possibleMoves** und **abort** können nur dann ausgeführt werden, wenn ein Spiel aktiv ist. Ist gerade kein Spiel aktiv, so führen diese Befehle zu einem Fehler.
- Die Eingabe von **quit** ist immer möglich, egal ob gerade ein Spiel aktiv ist oder nicht.

Um die Spezifikation zu illustrieren, zeigen wir Ihnen zwei weitere Ablaufbeispiele.

Beispiel 1

```
othello> newGame 8 8 WW-B----,WWB-----,WWB-B-BW,WWBBBBBB,WWBWBWB-,WWWWWBBB,BWWBBWWW,WWWWWWWW
othello> move H2
othello> move H5
black passes.
othello> print
WW-B----
WWB----B
WWB-B-BB
WWBBBBBB
WWBWBWWW
WWWWWBWW
BWWBBWWW
WWWWWWWW
turn: white
othello> move H1
black passes.
othello> move E2
othello> move D2
othello> print
WW-B---W
WWBBW--W
WWB-B-BW
```

```
WWBBWBBW
WWBWWWWW
WWWWWBWW
BWWBBWWW
WWWWWWWW
turn: white
othello> move G2
othello> move F3
othello> move D3
othello> move F2
othello> print
WW-B---W
WWWBBBWW
WWWWBBWW
WWWBWBWW
WWBWWWWW
WWWWWBWW
BWWBBWWW
WWWWWWWW
turn: white
othello> move G1
othello> move F1
othello> move C1
black passes.
othello> move E1
Game Over! white has won (58:6)!
```

Beispiel 2

```
othello> newGame 8 8
othello> hole C3:C6
othello> hole F3:F6
othello> print
--------
--------
--#--#--
--#WB#--
--#BW#--
--#--#--
--------
--------
turn: black
othello> possibleMoves
Possible moves: D3,E6
othello> move D3
othello> hole A7:A7
Error! Cannot add hole area. Game has already started!
```

Beispiel 3

```
othello> newGame 4 6
othello> print
----
----
-WB-
-BW-
----
----
turn: black
othello> move C5
othello> move B5
othello> move A5
othello> move D3
othello> print
----
----
-WWW
-BB-
BBB-
----
turn: black
othello>
```

Allgemeine Hinweise zur Implementierung

Überlegen Sie sich, welche Klassen Sie sinnvollerweise benötigen.

Setzen Sie die in der Vorlesung gelernten Prinzipien sinnvoll um.

In Ihrer Implementierung dürfen Sie Klassen aus den Paketen `java.lang.*` und `java.util.*` verwenden. Insbesondere sind also Klassen aus dem Java Collections Framework erlaubt, z.B. `LinkedList`, `TreeMap` oder `HashMap`. Beachten Sie auch, dass Java eine einfache Möglichkeit zum Sortieren von Listen über `Collections.sort()` zur Verfügung stellt.

Sie dürfen ebenfalls die aus der Übung bekannte Klasse `Terminal` verwenden.

Abgabe der Lösung im Praktomat

Reichen Sie bei der Abgabe Ihrer Lösung in jedem Fall folgende Dateien mit ein:

- Eine Datei `Shell.java`, die die main-Methode enthält.
- Eine Datei `Tests.txt`, die eigene Testfälle enthält. Dokumentieren Sie die Testfälle in Form von Programmabläufen, wie im obigen Beispiel.

Reichen Sie die Klasse `Terminal` **nicht** mit ein.

Achten Sie bei der Bearbeitung dieser Abschlussaufgabe auf

- korrekte Java-Syntax
- Einhaltung der Java Code Conventions
- Einrückungen
- Einhaltung vorgegebener Interfaces
- Umsetzung des Geheimnisprinzips
- ausführliche Dokumentation des Quelltextes
- adäquates Verhalten öffentlicher Methoden bei ungültigen Parametern
- sinnvolle Modellierung der Klassenstruktur

Praktomat wird für diese Abschlussaufgabe folgende automatische Prüfungen auf Ihren Lösungen ausführen. **Fett** markierte Prüfungen müssen bestanden werden, damit die Lösung eingereicht werden kann:

- **Maximale Zeilenbreite:** 120 Zeichen
- **Übersetzen (Java)**
- **Checkstyle WS (1/2)**: Überprüfen korrekter Whitespace-Setzung
- **Checkstyle JCC (2/2)**: Überprüft, ob Namenskonventionen eingehalten wurden und ob die eingereichten Klassen mit Javadoc-Kommentaren versehen wurden.
- Funktionalitätstest: Testet die Funktionalität Ihres Programmes.
  - **Einfache Tests**: Testet das Programm auf minimale Funktionalität.
  - Erweiterte Tests: Schwierigere und umfassendere Tests.

Bewertung Ihrer Abgabe

Beide Aufgaben werden jeweils in den Kategorien *Funktionalität* und *Programmiermethodik* mit einer Punktezahl von 7 bis 0 bewertet. Zum Bestehen der Abschlussaufgaben müssen

- beide Aufgaben erfolgreich in Praktomat eingereicht sein (d.h. die oben fett markierten Prüfungen müssen bestanden werden) **und**
- der Durchschnitt über beide Aufgaben in *Funktionalität* besser oder gleich 4 Punkte sein **und**
- der Durchschnitt über beide Aufgaben in *Programmiermethodik* besser oder gleich 4 Punkte sein.

Bei der Berechnung der *Endnote* werden die Punkte in *Funktionalität* **doppelt** gewichtet.

Bei Abschreiben werden beide Abschlussaufgaben mit **nicht bestanden** bewertet. Auch dann, wenn nur eine der Aufgaben abgeschrieben wurde.

Beispiel 1:

- **Aufgabe 1**: Funktionalität: 7, Programmiermethodik: 5
- **Aufgabe 2**: Funktionalität: 5, Programmiermethodik: 3

Es gilt also für den Durchschnitt: **Funktionalität**: 6, **Programmiermethodik**: 4 ⇒ Bestanden!

Beispiel 2:

- **Aufgabe 1**: Funktionalität: 7, Programmiermethodik: 3
- **Aufgabe 2**: Funktionalität: 7, Programmiermethodik: 4

Es gilt also für den Durchschnitt: **Funktionalität**: 7, **Programmiermethodik**: 3,5 ⇒ **Nicht** bestanden!

Beispiel 3:

- **Aufgabe 1**: Funktionalität: 5, Programmiermethodik: 1
- **Aufgabe 2**: Funktionalität: 6, Programmiermethodik: 3

Es gilt also für den Durchschnitt: **Funktionalität**: 5,5, **Programmiermethodik**: 2 ⇒ **Nicht** bestanden!

Beispiel 4:

- **Aufgabe 1**: Funktionalität: 6, Programmiermethodik: 7
- **Aufgabe 2**: Funktionalität: 7, Programmiermethodik: 2

Es gilt also für den Durchschnitt: **Funktionalität**: 6,5, **Programmiermethodik**: 4,5 ⇒ Bestanden!