Aufgabenzettel 9

Datenströme und innere Klassen

Dieser Aufgabenzettel muss spätestens in Ihrer Praktikumsgruppe in **KW 27** vorgestellt werden.

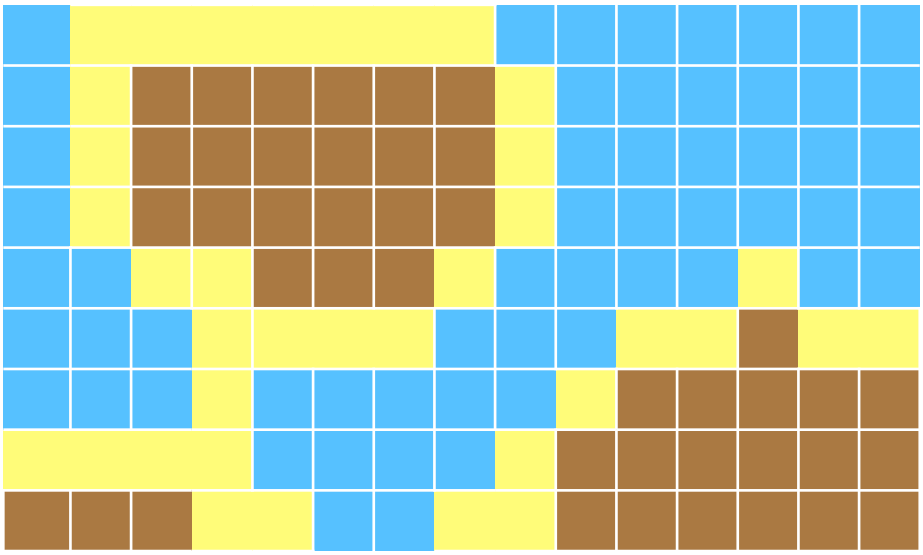
Alle für diesen Aufgabenzettel entwickelten Programme sollen sich in einem Paket **de.hsruhrwest.oop.ss2025.assignment9** oder entsprechenden **Unterpaketen** befinden.

Hinweis: Alle Klassen müssen vollständig mit JavaDoc kommentiert sein.

Aufgabe 1: Dateien lesen und schreiben

In dieser Aufgabe üben Sie das Lesen und Schreiben von Textdateien.

a) Erstellen Sie zunächst eine Klasse **Landscape**, die eine einfache zweidimensionale Landschaft repräsentiert. Eine Landschaft besteht aus einem 2D-Spielfeld, das aus einzelnen Kacheln zusammengesetzt wird. Das Spielfeld ist definiert durch eine Breite und Höhe und ein zweidimensionales Array von Kacheln, wobei als Kacheln folgende Arten unterschieden werden: Erdboden, Wasser oder Strand. Ein Spielfeld der Größe 15 Spalten x 9 Zeilen könnte z.B. so aussehen:



Kacheln sollten als **innerer** Aufzählungstyp in der Klasse **Landscape** realisiert werden. Die Klasse **Landscape** sollte einen Konstruktor besitzen, der Breite und Höhe der Landschaft

setzt. Außerdem sollte es Methoden geben, um einzelne Kacheln an einer Stelle (x,y) zu setzen oder zurückzugeben. Ungültige Werte, z.B. negative Koordinaten, sollten zu Ausnahmen führen. Die Klasse sollte außerdem eine **toString**-Methode besitzen, die die Landschaft lesbar als String repräsentiert.

b) Schreiben Sie eine Hilfsmethode, die eine Landschaft in einen OutputStream schreibt. Dabei soll folgendes Format eingehalten werden:

- Die erste Zeile der Datei sollte den String „landscape“ enthalten und dient der Identifikation von Landschaftsdateien.
- In der zweiten Zeile stehen Breite und Höhe der Landschaft, getrennt durch ein x.
- Danach erfolgt zeilenweise die Ausgabe der Landschaft.

Die Datei für das obige Beispiel könnte z.B. so aussehen, wobei eine Tilde (~) für Wasser, ein Punkt für Sand und ein X für Erde steht:

```
landscape
15x9
~.......~~~~~~~
~.XXXXXX.~~~~~~
~.XXXXXX.~~~~~~
~.XXXXXX.~~~~~~
~~..XXX.~~~~.~~
~~~....~~~..X..
~~~.~~~~~.XXXXX
....~~~~.XXXXXX
XXX..~~..XXXXXX
```

Demonstrieren Sie die Methode in einem Hauptprogramm, indem Sie eine bestehende Landschaft in eine Datei schreiben.

c) Schreiben Sie eine Hilfsmethode, die eine Landschaft aus einem InputStream liest. Werden fehlerhafte Daten gelesen (falscher String in der ersten Zeile, ungültige Breite und Länge, usw.), soll eine von Ihnen erstellte **InvalidFormatException** geworfen werden.

Demonstrieren Sie folgendes in einem Hauptprogramm:

- Das Einlesen einer Karte aus einer Datei (File).
- Das Einlesen einer Karte aus einer Resource, die in Ihrem Projekt liegt.

Aufgabe 2: Innere Klassen

In dieser Aufgabe üben Sie den Umgang mit inneren Klassen und Lambda-Ausdrücken.

a) Schreiben Sie eine Klasse **User**, die einen Nutzer darstellt. Ein Nutzer hat einen Namen und ein Profil. Ein Profil sollte als innere Klasse realisiert sein und enthält die Gesamtpunktzahl eines Nutzers und das Datum, an dem er/sie Mitglied wurde. Dieses Datum soll durch die Java-Klasse LocalDate repräsentiert werden.

b) Schreiben Sie eine Schnittstelle **UserFilter**. Die Schnittstelle besitzt nur eine einzige Methode **matches**, die einen Nutzer erhält und einen Wahrheitswert zurückgibt. Die Methode dient dazu, zu prüfen, ob Nutzer einem Kriterium entsprechen oder nicht.

c) Schreiben Sie eine Hilfsmethode **printMatchingUsers** in einer separaten Klasse. Die Methode erhält eine Liste von Nutzern und eine Referenz auf einen Filter vom Typ **UserFilter**. Die Methode gibt alle Nutzer aus, für die die **matches**-Methode des Filters true zurückgibt.

d) Legen Sie in einem Hauptprogramm ein paar Testnutzer in einer Liste an, jeweils mit eigenem Profil. Nutzen Sie nun die in c) erstellte Methode **printMatchingUsers**, um alle Nutzer der Liste auszugeben, die mindestens 300 Punkte besitzen. Nutzen Sie die Methode erneut, um alle Nutzer auszugeben, die seit maximal 100 Tagen Mitglied sind. Beim Aufruf von printMatchingUsers soll in beiden Fällen eine **anonyme Klasse** verwendet werden, die UserFilter implementiert.

e) Schreiben Sie das Hauptprogramm aus d) so um, dass es mit **Lambda-Ausdrücken** statt mit anonymen Klassen arbeitet.