5. Praktikum "Algorithmen und Programmierung II"
SS 2014

Auf der Web-Seite `www.gm.fh-koeln.de/ehses/ap/` finden Sie die Dateien `vorlage5a.zip` und `vorlage5b.zip`.

Aufgabe 1. *(vorlage5a)* Lösen Sie das N-Damen Problem. Auf einem *N*N* großen Schachbrett sind *N* Damen so aufzustellen, dass sie sich gegenseitig nicht schlagen können. Zwei Damen können sich schlagen, wenn sie in der gleichen Zeile, gleichen Spalte oder gleichen diagonal auf- oder absteigenden Linie stehen.

Die Lösung ist schon in soweit vorbereitet, als eine Klasse zur Erzeugung eines Schachbretts mit Damen existiert, die alle nötigen Operationen unterstützt (`BoardWithQueens`). Es geht nur noch um die Umsetzung der Lösungssuche mit einem rekursiven Backtracking-Algorithmus (in `Queens.java`). Dieser sieht in Pseudocode wie folgt aus:

```
boolean loesbar(brett):
  falls brett vollständig mit N-Damen besetzt ist:
    return true
  andernfalls:
     für zeile 0 bis N-1:
       falls in zeile eine neue Dame sicher ist:
           setze eine neue Dame in zeile
           mache den aktuellen Zustand sichtbar
           falls loesbar(brett): // Rest war lösbar
                  return true // wird sind (momentan) fertig
           andernfalls:
               entferne die gesetzte Dame wieder
               mache den aktuellen Zustand sichtbar
               // und probiere die naechste zeile
     // wenn wir hierher kommen, hatten wir keinen Erfolg
     return false
```

Hinweise:

- Die Damen werden automatisch fortlaufend von links nach rechts gesetzt.
- Im Internet finden Sie hilfreiche Informationen.
- Fertige Algorithmen sind aber hier nicht brauchbar.
- Sorgen Sie dafür, dass der Lösungsvorgang graphisch dargestellt wird.

Fragen:

- Warum heißt der Algorithmus *Backtracking*?
- Was muss man tun, wenn man alle möglichen Lösungen finden will?

Lernziel: Verständnis einer Pseudcode-Lösung in Verbindung mit fertigen dokumentierten Klassen. Erlernen von „Backtracking“.

Aufgabe 2. *(vorlage5b)* Die Klasse `Application` erlaubt das wahlweise Erzeugen von Objekten der Klassen `Circle` und `Rectangle`. Soweit es sich um das Erzeugen von Kreisen handelt, ist das Programm (und der Test) lauffähig.

Schreiben Sie die Klasse `Rectangle`, die das Interface `IShape` sinnvoll (die Klasse soll halt Rechtecke beschreiben) implementiert. Schreiben Sie auch eine dazugehörige Klasse `RectangleTest`, die festlegt, wie die Ergebnisse der Methoden aussehen sollen. Orientieren Sie sich an den Klasse Circle und `CircleTest`. Schreiben Sie sinnvolle Javadoc-Kommentare.

Schreiben Sie eine Comparator-Klasse namens `ShapesAlphabetisch`, die die Namen zweier IShape-Objekte mittels `String.compareTo` vergleicht. Testen Sie die Klasse mit dem Test `ShapesAlphabetischTest` und führen Sie danach auch die Klasse `Application` aus.

Hinweis: Zeilen, die wegen der noch fehlenden Klassen zu Compilerfehlern führen würden, sind auskommentiert. Diese Kommentare müssen natürlich letztlich entfernt werden. Diese Zeilen sind mit TODO: gekennzeichnet.

Hinveis: Abgeben sollen Sie den Endzustand - nicht die Zwischenschritte. Beachten Sie die Kommentare im Interface `IShape` und die Verwendung der neuen Klassen in den bereits vorhandenen Klassen.

Lernziele: Entwicklung von JUnit-Tests, Interface, Implementierung des Schnittstelle `Comparator`.