ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Dr. Jochen Hoenicke
Alexander Nutz

23.6.2014
Abgabe: Montag 30.6.2014
via Daphne/Subversion

8. Übungsblatt zur Vorlesung
Einführung in die Informatik

Aufgabe 1: Sortierte Verkettete Listen ... mit Generics
Rufen Sie sich die Implementierung der sortierten verketteten Liste letzte Woche in Erinnerung. (Eine vereinfachte Implementierung als Ausgangspunkt für diese Aufgabe steht auf der Vorlesungswebsite.) Ein Problem bestand darin, dass bezüglich der eingefügten Objekte nur garantiert wird, dass sie Comparable sind, nicht aber, dass sie alle vom gleichen Typ sind.
Benutzen Sie Generics, um die verkettete sortierte Liste so zu erweitern, dass sie nur noch einen Typ von Daten enthält, der zum Zeitpunkt der Erstellung der Liste festgelegt wird. Dieser Typ soll beliebig wählbar sein, solange er das Interface `Comparable` implementiert.

Aufgabe 2: Generics – Hintergrund
Betrachten Sie folgendes Codefragment:

```
..
ArrayList<String> someStrings = new ArrayList<String>();
ArrayList<Object> someObjects = (ArrayList<Object>) someStrings;
..
```

Wie Sie wissen, erbt `String` von `Object`, was ja ungefähr bedeutet „jeder `String` ist auch ein `Object`“. Wieso kompiliert obiger Code dennoch nicht? Was wäre das Problem, wenn man zulassen würde, dass eine Liste eine Typs `T` immer wie eine Liste eines Supertyps von `T` behandelt werden kann?
Betrachten Sie nun folgendes Codefragment:

```
..
String[] someStrings = new String[] { "bla", "blub" };
Object[] someObjects = (Object[]) someStrings;
..
```

Dieser Code kompiliert. Was könnte bei der weiteren Verwendung von `someObjects` dennoch problematisch werden?

Aufgabe 3: Alapo (3) – Berechnung aller möglichen Spielzüge
Schreiben eine Klasse `AIPlayer`, die von `Player` erbt. Erstellen Sie darin eine Methode `ArrayList<Move> computePossibleMoves(Board board)`, die alle möglichen (d.h.

gültigen) Züge des entsprechenden Spielers zurückgibt. (Damit erhalten Sie schon eine – wenn auch recht dumme – KI, wenn Sie einfach immer das erste Element der Liste in `getMove(..)` zurückliefern.)
Zusätzlich zu den schon bekannten Einschränkungen sollen auch Züge ungültig sein, in denen der Gegner eine Figur auf der eigenen Grundlinie stehen hat, und wo diese Figur nicht sofort geschlagen wird. (Gewinnbedingung ist dann, dass der andere Spieler keine gültigen Züge mehr hat.)
Beachten Sie, dass sich das Interface für Spieler noch einmal geändert hat. `Player` ist jetzt ein Interface, keine abstrakte Klasse mehr. Die Implementierung des Spielbretts ist nicht mehr Teil des Interfaces, so dass die KIs eine beliebige eigene Repräsentation verwenden können. Die entsprechenden Änderungen in `doc/public/Alapo` wurden gemacht.
Verwenden Sie in Zukunft die dort vorgeschlagene (geänderte) Ordnerstruktur. D.h. die Klassen für Ihre Alapo-KI sollten im Verzeichnis
`(daphneSVN)/ab123/uebung08/Alapo/alapo/ab123` (Package `alapo.ab123`)
liegen.
Im Unterverzeichnis `an37` stehen Skelettklassen für die Implementierung dieser Aufgabe bereit. Im Unterverzeichnis `jh957` ist ein KI-Spieler als `.class` Dateien vorhanden. Mit einem Befehl der Art `java alapo.Alapo alapo.jh957.JohoPlayer alapo.HumanPlayer` können Sie zwei Spieler gegeneinander spielen lassen.