Informatik II (D-ITET)
Frühjahrssemester 2014
Prof. F. Mattern
C. Beckel (beckel@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Gruppe für Verteilte Systeme
http://vs.inf.ethz.ch

Übungsserie Nr. 7

Ausgabe: 9. April 2014
Abgabe: 16. April 2014

Hinweise

Für diese Serie benötigen Sie die folgenden Archive:
http://vs.inf.ethz.ch/edu/FS2014/I2/downloads/u7.zip
http://vs.inf.ethz.ch/edu/FS2014/I2/downloads/reversi.jar

1. Aufgabe: (6 Punkte) Array-Listen und Generics

Analog zu dynamisch wachsenden Stacks gibt es manchmal auch Bedarf nach dynamisch wachsenden Arrays. In Java gibt es hierfür einen Container namens *ArrayList*[1] (siehe auch Skript Seite 141, Folie 354). In dieser Aufgabe sollen Sie unter Verwendung von *ArrayList* einen Filter implementieren, der nur Studenten mit einer Mindestanzahl an Punkten zum Testat zulässt. Die Klasse *Student* ist bereits im Programmgerüst enthalten.

(1a) *(1 Punkt)* Erstellen Sie eine leere Implementierung der Schnittstelle *IFilter* und implementieren Sie die Fabrikmethode *FilterFactory.create*.

(1b) *(3 Punkte)* Implementieren Sie in einer neuen Klasse (z.B. *Filter*) die Methode *filterRaw* anhand der Schnittstellendokumentation. Diese Methode verwendet *ArrayList* als sog. *raw type*, also als generische Array-Liste von *Object*s. Davon wird seit Java 1.5 abgeraten, worauf Sie der Compiler hinweist. Ignorieren Sie diesbezügliche Warnungen und setzen Sie in Ihrer Implementierung *keine* Generics ein.

Hinweis: In den Tests sehen Sie, wie die Liste der Studenten genau aufgebaut ist.

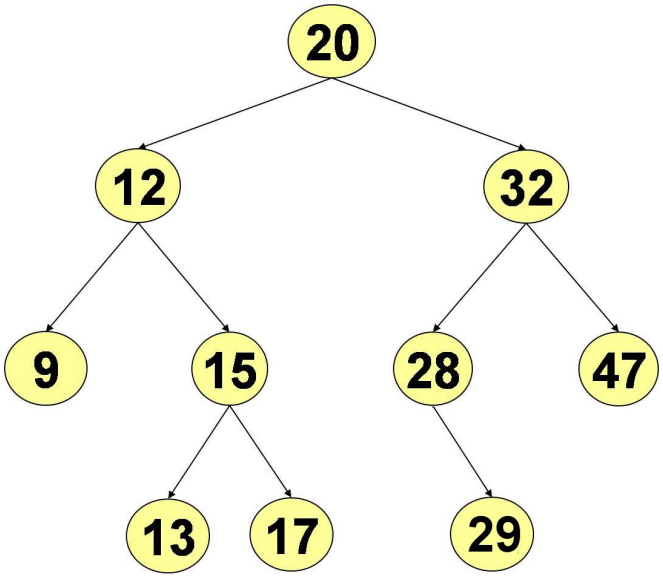
(1c) *(2 Punkte)* Implementieren Sie nun die Methode *filterGeneric* anhand der Schnittstellendokumentation. Diese Methode spezifiziert auch den Typ der in der *ArrayList* gespeicherten Objekte. Diese Technik nennt sich in Java *Generics*[2] (Skript Seite 143, Folie 357). Aus Sicht des Benutzers gibt es drei wesentliche Unterschiede:

[1] http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
[2] http://docs.oracle.com/javase/tutorial/java/generics/index.html

- Man sieht den Array-Listen an, was für Objekte darin gespeichert sind.
- Nur kompatible Objekte können zur Liste hinzugefügt werden.
- Man erhält aus der Array-Liste direkt Referenzen des richtigen Typs und muss daher nicht manuell *Object*-Referenzen in den richtigen Typ casten.

2. Aufgabe: (10 Punkte) Binäre Suchbäume

(2a) *(2 Punkte)* Löschen Sie aus dem folgenden binären Suchbaum in dieser Reihenfolge die Elemente 15, 12 und 20 und skizzieren Sie den jeweils resultierenden Suchbaum. Wenden Sie dabei die Strategie "Ersetzen durch kleinstes Element des rechten Teilbaums" an.



(2b) *(8 Punkte)* Implementieren Sie die Schnittstelle *IBinarySearchTreeUtils*⟨*T*⟩ und die Fabrikmethode *UtilsFactory.create* anhand der vorhandenen Dokumentation.

Hinweis: Behandeln Sie den *Generic*-Typ *T* einfach wie einen Typ, den Sie nicht kennen. So können Sie z.B. einfach eine *ArrayList*⟨*T*⟩ anlegen und Objekte vom Typ *T* aus dem Baum nehmen und in die Array-Liste legen.

Die Tests arbeiten mit *BinarySearchTree*s von *String*s. Die Fabrikmethode muss dementsprechend diesen konkreten Typ beim Erzeugen des Objektes angeben.

Die Methode *unlinkSmallest* gibt zwei Dinge zurück. Die einzige Möglichkeit, so etwas in Java zu realisieren ist, eine Klasse zu schreiben, die diese beiden Dinge enthält, so dass die Methode ein Objekt dieser Klasse zurückgeben kann. Wir nennen diese Klasse *UnlinkSmallestResult*.

3. Aufgabe: (7 Punkte) Reversi [Teil 1]

Mit dieser Aufgabe startet eine Serie, die zum Ziel hat, einen Computerspieler für das Spiel *Reversi* zu implementieren. Gegen Ende des Semesters wird ein Turnier stattfinden, bei dem diese Computerspieler live und vor Publikum gegeneinander antreten. Die Autoren der Reversispieler können dabei tolle Preise gewinnen! Besuchen Sie die Reversi-Webseite[3] um Details über das Turnier, die Regeln und die Dokumentation des Reversi-Frameworks zu finden.

(3a) *(*3 Punkte) Das Reversi-Framework steht Ihnen als .jar-Datei zum Download zur Verfügung. Binden Sie es analog zu JUnit in Ihre Eclipseprojekte ein. Starten Sie nun ein Spiel zwischen Ihnen und Ihrem Teampartner. Gehen Sie dabei folgendermassen vor.

- Klicken Sie auf *Run* → *Run Configurations*.
- Legen Sie eine neue *Java Application* an.
- Setzen Sie *Main class* auf *reversi.Arena*.
- Klicken Sie auf den Reiter *Arguments*.
- Setzen Sie *Program arguments* auf "-t 0 TestGame u7a3.HumanPlayer u7a3.HumanPlayer".

  Dieser Befehl legt ein neues Spiel namens *TestGame* an, in welchem die Spieler-Implementierung *u7a3.HumanPlayer* gegen sich selbst antritt. Die zeitliche Begrenzung für jeden Zug wird auf *0* gesetzt und damit abgeschalten.
- Klicken Sie auf *Apply*.
- Zum Starten dieser *Run Configuration* klicken Sie auf *Run*.

Die Eingabe der Spielzüge erfolgt über die Konsole. Spielen Sie eine Runde, um ein erstes Gefühl für das Spiel zu bekommen. Schicken Sie Ihrem Tutor einen Screenshot des Spielbrettes am Ende des Spiels.

(3b) *(*4 Punkte) Implementieren Sie einen eigenen Spieler, indem Sie die Schnittstelle *ReversiPlayer* implementieren. Ihr Spieler soll unter allen möglichen Zügen einen zufälligen auswählen.

Testen Sie Ihre Implementierung, indem Sie gegen Ihren Spieler antreten. Legen Sie dazu eine neue *Run Configuration* an und geben Sie als zweiten Spieler den vollständigen Namen Ihrer Klasse an.

Hinweis: Orientieren Sie sich am Quellcode des *HumanPlayer*s.

[3] `http://informatik2.ee.ethz.ch/reversi`