Informatik II (D-ITET)
FS 2010
Prof. Dr. F. Mattern
A. Bernauer (bernauer@inf.ethz.ch)

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

# Übungsserie Nr. 7

Ausgabe: 21. April 2010
Abgabe: 28. April 2010

**Hinweise**

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

## 1. Aufgabe: (6 Punkte) Vektoren und Generics

Analog zum Bedarf für dynamisch wachsende Stacks existiert manchmal auch der Bedarf nach dynamisch wachsenden Arrays. In Java heissen solche Container *Vector*[1]. Implementieren kann man Vektoren analog zu Stacks entweder durch Verdopplung und Kopieren, durch verkettete Listen oder durch Listen von Chunks. In dieser Aufgabe sollen Sie sich mit Vektoren vertraut machen.

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

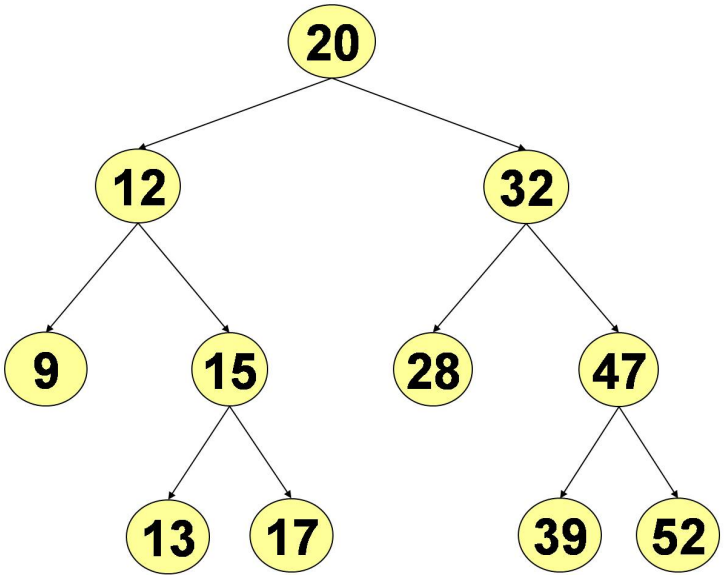
(1b) *(3 Punkte)* Implementieren Sie die Methode *filterRaw* an Hand der Schnittstellendokumentation. Diese Methode verwendet *Vector* als sog. *raw type*, also als generischen Vektor von *Object*s. Seit Java 1.5 wird das nicht mehr empfohlen, worauf Sie der Compiler hinweist. Ignorieren Sie alle diesbezüglichen Warnungen.

(1c) *(2 Punkte)* Implementieren Sie die Methode *filterGeneric* an Hand der Schnittstellendokumentation. Diese Methode spezifiziert für alle Vektoren den Typ der Objekte, die darin gespeichert werden. Diese Technik nennt sich in Java *Generics*[2]. Aus Sicht des Benutzers gibt es drei wesentliche Unterschiede. Erstens sieht man dem Vektor an, was für Objekte darin gespeichert sind. Zweitens können nur Objekte von dem angegeben Typ hinzugefügt werden. Und drittens erhält man aus dem Vektor direkt Referenzen des richtigen Typs, anstatt von Hand *Object* Referenzen in den richtigen Typ casten zu müssen.

[1] http://java.sun.com/javase/6/docs/api/java/util/Vector.html
[2] http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html

## 2. Aufgabe: (10 Punkte) Binäre Suchbäume

(2a) *(2 Punkte)* Löschen Sie aus dem folgenden binären Suchbaum nacheinander die Elemente 15, 12 und 20 (in dieser Reihenfolge!) und skizzieren Sie den jeweils resultierenden Suchbaum. Wenden Sie dabei die Strategie "Ersetzen durch kleinstes Element des rechten Teilbaums" an (vgl. Skript Folie 262).




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

Hinweis: Behandeln Sie den *Generic*-Typ *T* einfach wie einen Typ, den Sie nicht kennen. So können Sie z.B. einfach einen *Vector*⟨*T*⟩ anlegen und Objekte vom Type *T* aus dem Baum nehmen und in den Vektor 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. Diese Klasse heisst in diesem Fall *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.

- Gehen Sie zu *Run* → *Run Configurations...*.
- Legen Sie eine neue *Java Application* an.
- Setzen Sie *Main class* auf *reversi.Arena*.
- Gehen Sie auf den Reiter *Arguments*.
- Setzen Sie *Program arguments* auf "-t 0 TestGame u7a3.HumanPlayer u7a3.HumanPlayer".
- 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 ersten eigenen Spieler, in dem 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://www.vs.inf.ethz.ch/edu/I2/reversi`