Informatik II (D-ITET)
FS 2009
Prof. Dr. F. Mattern
Philipp Bolliger (bolligph@inf.ethz.ch)

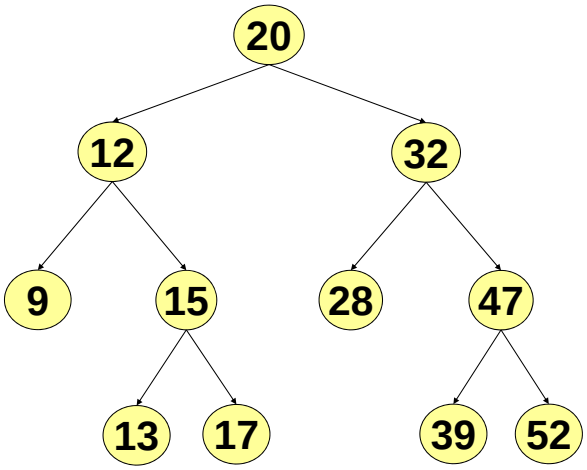
ETH Zürich
Institut für Pervasive Computing
Verteilte Systeme
http://www.vs.inf.ethz.ch

Übungsserie Nr. 7

Ausgabe: 8. April 2009
Abgabe: 22/23. April 2009

1. Aufgabe: (10 Punkte) Binäre Suchbäume

(1a) *(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 wenn möglich die Strategie "Ersetzen durch kleinstes Element des rechten Teilbaums" an (vgl. Folie 243).



Nun sollen Sie einen binären Suchbaum zum Verwalten von Integer-Zahlen implementieren. Beachten Sie, dass verglichen mit der Vorlesung (Folie 232 ff.) in dieser Teilaufgabe eine leicht abgeänderte Implementation verlangt wird.

(1b) *(1 Punkt)* Schreiben Sie eine Klasse `IntTree` zum Verwalten eines (Teil-)baums. Die Klasse soll folgende Attribute enthalten:

- **key** das Knotenattribut (vom Typ `int`) der Wurzel des (Teil-)baums

- **left** den rechten Unterbaum (vom Typ `IntTree`)
- **right** den linken Unterbaum (vom Typ `IntTree`)

Dabei müssen Sie den „leeren Baum" nicht berücksichtigen: Der Einfachheit halber nehmen Sie an, dass jeder Baum zumindest seinen Wurzelknoten enthält. Schreiben Sie dafür einen Konstruktor, der einen Baum mit einem `int` (dem Knotenattribut der Wurzel) initialisiert.

Zudem schreiben Sie noch folgende **rekursive** Methoden:

(1c) *(1 Punkt)* **insert** zum Einfügen eines Integers in diesen (Teil-)baum.

(1d) *(1 Punkt)* **height** zum Errechnen der Höhe des Baumes. Dabei soll ein Baum, der nur aus einem Wurzelknoten besteht, die Höhe 1 haben.

(1e) *(2 Punkte)* **toBracedString** gibt die Klammerdarstellung des Baumes als String zurück.

(1f) *(2 Punkte)* **toString** gibt den Inhalt des Baumes in einem String sortiert zurück. Die Elemente im String sollen dabei mit Leerzeichen getrennt sein. Beachten Sie, dass die sortierte Ausgabe im binären Suchbaum sehr leicht zu erreichen ist (siehe Vorlesung).

Die obigen Methoden sollen alle auf den Instanzen eines (Teil-)baums aufgerufen werden und sollen daher nicht als `static` deklariert werden.

(1g) *(1 Punkt)* Schreiben Sie eine Testmethode `main`, die einen Baum erstellt, einige zufällige Zahlen einfügt, die Höhe des Baumes angibt, und den Baum sowohl in Klammerdarstellung als auch sortiert ausgibt. Fügen Sie die Ausgabe des Programms Ihrer Abgabe bei.

2. Aufgabe: (12 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 Web-Seite[1], um Weiteres zu erfahren. Dort finden Sie auch die genauen Spielregeln zu Reversi. Es ist wichtig, dass sich alle Programme an diese Regeln halten, um ein Spiel überhaupt zu ermöglichen.
Zuerst sollen die Grundprinzipien des Spiels implementiert werden: das Ausführen von Spielzügen und die Überprüfung der Regelkonformität. Zudem wird diese Grundausstattung in ein Gerüst integriert, das es erlaubt, Programme gegeneinander antreten zu lassen und sie beim Spiel zu beobachten. Später werden Strategien entwickelt, welche die Qualität des Spielers verbessern sollen.

(2a) *(2 Punkte)* Laden Sie sich die Datei `reversi-1.2.2-eclipse.zip` (oder `reversi-1.2.2.zip`, oder `reversi-1.2.2.tar.gz`) von der Reversi-Webseite herunter (siehe "Downloads"). Entpacken Sie den Inhalt der Datei in ein Verzeichnis Ihrer Wahl.
In dem gewählten Verzeichnis werden drei Unterverzeichnisse `reversi`, `humanPlayer` und `javadoc` angelegt. Das Verzeichnis `reversi` enthält das Paket `reversi` mit Klassen und

[1] http://www.vs.inf.ethz.ch/edu/i2/reversi/

Interfaces, die wir auch für weitere Reversi-Aufgaben benutzen werden. Es enthält weiterhin das Programm `Arena`, welches es erlaubt, die von Ihnen in dieser und in zukünftigen Aufgaben entwickelten Reversi-Computerspieler gegeneinander antreten zu lassen. Im Verzeichnis `javadoc` befindet sich die für diese Serie relevante Dokumentation zum Reversi-Paket im HTML-Format, so wie Sie es von *Javadoc* gewohnt sind.
Das Verzeichnis `humanPlayer` enhält einen "Computerspieler" `HumanPlayer`, der seine Züge berechnet, indem er sie über die Tastatur einliest. Sie können zunächst, als Test, zwei menschliche Spieler wie folgt gegeneinander spielen lassen (mehr Informationen erhalten Sie auf der Reversi-Webseite):

```
java -classpath . reversi.Arena -t 0 SpielName humanPlayer.HumanPlayer
humanPlayer.HumanPlayer
```

`SpielName` ist ein beliebiger Name für das Spiel. Die Option `-t 0` setzt die Dauer, die sich beide Spieler für einen Zug nehmen dürfen, auf unendlich.
Jetzt spielen Sie eine Partie gegen Ihren Gruppenkollegen (oder sich selbst :-)) und geben Sie einen *Screenshot* des Endzustandes des Reversi-Arena-Fensters sowohl als gedruckte Kopie als auch per E-Mail ab.

Beachten Sie, dass jeder Spieler das folgende Interface implementieren muss:

```
package reversi;

public interface ReversiPlayer {

    /**
     * Uebergibt dem Spieler seine zugewiesene Farbe und die fuer beide
     * Spieler identische Zeitbeschraenkung in Millisekunden. Die
     * Methode wird ein einziges Mal zu Beginn eines Spieles aufgerufen.
     * ...
     * ...
     **/
    public void initialize(int myColor, long timeLimit);

    /**
     * Berechnet auf der Basis des uebergebenen Spielfeldes den
     * naechsten Spielzug.
     * ...
     * ...
     **/
    public Coordinates nextMove(GameBoard gb);
}
```

Schauen Sie sich dazu auch die Implementation in `humanPlayer.HumanPlayer` an.

(2b) *(5 Punkte)* Ergänzen Sie die Klasse `HumanPlayer` um eine Methode `checkMove` zum Prüfen, ob ein bestimmter Zug `zug` für "rot" bzw. "grün" erlaubt ist:

```
public boolean checkMove(Coordinates zug, GameBoard gb)
```

Die aktuelle Spielsituation wird als Objekt `gb` übergeben, welches das Interface `GameBoard` implementiert. Sehen Sie nötigenfalls in der Dokumentation von GameBoard nach, wie Sie auf Informationen über das Spielbrett zugreifen können.

Ändern Sie die Implementation so, dass solange wiederholt ein Zug von der Tastatur eingelesen wird, bis ein erlaubter Zug eingegeben wurde. Überprüfen Sie auch, ob überhaupt ein erlaubter Zug möglich ist, anderfalls soll `nextMove` den Wert `null` zurückgeben.

(2c) *(5 Punkte)* Schreiben Sie einen Computerspieler "`RandomPlayer`", der zufällig einen korrekten Zug auswählt. Schreiben Sie dazu eine Klasse, die das Interface `ReversiPlayer` implementiert. Orientieren Sie sich bei der Entwicklung Ihres Computerspielers an der Klasse `HumanPlayer`.

Ihre Klasse soll die folgenden Anforderungen erfüllen:

- Ihre Klasse muss einen Konstruktor mit leerer Parameterliste besitzen.
- Die Methode `nextMove()` soll einen zufälligen, aber gültigen Zug für die in `initialize()` angegebene Farbe zurückliefern (und `null`, falls es keinen gültigen Zug gibt). Die aktuelle Spielsituation wird Ihnen über das Interface `GameBoard` zugänglich gemacht, siehe oben. Selbstverständlich können Sie hier auf Ihre Implementation in Teilaufgabe *a)* zurückgreifen. Sie können die Methode `checkMove` entweder in die Klasse des Computerspielers kopieren oder eine effizientere Lösung (Vererbung) finden, die mit einem Zusatzpunkt belohnt wird.
- Ihre Klasse (und gegebenenfalls benötigte Hilfsklassen) sollen in einem eigenen Paket (und damit in einem eigenen Unterverzeichnis) untergebracht sein, dessen Namen Sie beliebig wählen können.

Testen Sie Ihre Klasse, indem Sie (mittels `HumanPlayer`) gegen Ihren Computer-Zufallsspieler spielen. Starten Sie dazu die `Arena` wie folgt:

```
java -classpath . reversi.Arena -t 0 SpielName mypack.myclass
humanPlayer.HumanPlayer
```

Ersetzen Sie dabei "`mypack.myclass`" durch den Paket-/Klassennamen Ihres Computerspielers. Beachten Sie, dass Ihr Paket in einem Unterverzeichnis des aktuellen Verzeichnisses liegen muss, also `./mypack/...`. Sie können den Computer-Zufallsspieler natürlich auch gegen ihn selbst spielen lassen.

Summe: 22 Punkte