Technische Universität Berlin **SS 2013**
Fakultät II, Institut für Mathematik
Sekretariat MA 5–2, Frau Kiefer

Prof. Dr. Martin Skutella
Torsten Gellert
Martin Groß
Dr. Max Klimm

Aylin Acikel, Katharina Bütow, Christian Döblin,
Alexander Hopp, Daniel Kuske, Olivia Röhrig
Robert Rudow, Daniel Schmand, Hendrik Schrezenmaier,
Judith Simon, Sebastian Spies, Fabian Wegscheider, Jan Zur

Computerorientierte Mathematik II

Binärbäume

1. Programmieraufgabe

Abnahme: spätestens am 18./19.4.2013 (je nach Gruppennummer).

Ungerade Gruppennummern geben spätestens am Donnerstag,
Gerade Gruppennummern geben spätestens am Freitag ab.

In dieser Programmieraufgabe implementiert ihr einen Binärbaum, den ihr für spätere Programmieraufgaben benötigt. Viele Vorgaben sind diesmal anhand von generischen Interfaces gegeben. Dort ist auch in der Java-Dokumentation vermerkt, was die geforderten Methoden leisten sollen. In den vorgegebenen Klassen selbst muss nicht programmiert werden. Programmiert und testet euren Code sorgfältig; Fehler, die ihr in dieser Aufgabe macht, werden euch sonst bei den späteren Programmieraufgaben Probleme bereiten.

Es ist günstig, die Programmieraufgabe in Eclipse[1] zu bearbeiten. Legt den Workspace in euer Home-Verzeichnis und nennt ihn auch "workspace". Legt in Eclipse ein neues Java-Projekt `program1` (mit separaten Ordnern für Quell- und Class-Dateien) an und öffnet darin ein neues Paket `binTree`. Beachtet dabei Groß- und Kleinschreibung! Ladet euch nun von unserer Homepage das zip-File zur 1. Programmieraufgabe herunter. Entpackt es und importiert die Dateien im Ordner `vorgabe` in das Verzeichnis `program1/src/binTree/` von Eclipse.

(a) Schreibt eine abstrakte generische Klasse `BinTree<T>`, die das Interface `BinTreeInterface<T>` implementiert.

- Legt in `BinTree<T>` eine generische innere Klasse `BinTreeNode` an, die die Knoten des Binärbaumes realisiert und deklariert Sie als `protected`. Ein `BinTreeNode` soll neben seinem Datum auch seinen Elter und die beiden Nachfolger kennen.
- Legt nun ein Datenfeld `BinTreeNode dummy` an, das auf einen Dummy-Knoten des Baumes verweist und ebenfalls `protected` ist. Der Dummy-Knoten soll über der Wurzel des Baumes hängen, sodass das linke Kind des Dummys die Wurzel ist. Ein solcher Dummy-Knoten erweist sich als günstig für den Iterator, den ihr implementieren werdet.

[1] `http://www.eclipse.org`

- Implementiert nun einen Default-Konstruktor `BinTree()`, der einen leeren Baum initialisiert, sowie einen Konstruktor `BinTree(T data)`, der ein Objekt vom Typ `T` übergeben bekommt und daraus einen einelementigen Baum erzeugt.
- Legt in `BinTree<T>` eine öffentliche Klasse `InorderIterator` an. Diese soll das Interface `IteratorInterface<T>` implementieren. Beim Aufruf `increment()` soll der Iterator zum nächsten Knoten in Inorder-Traversierung springen. Diese Klasse muss also wissen, welcher Knoten aktuell ist.
- Überschreibt in `BinTree<T>` die Methode `toString()`, welche jeder Klasse automatisch gegeben ist. Diese Methode soll den Baum graphisch als String ausgeben. Es ist nicht wichtig, wie schön der Baum aussieht, es zählt hauptsächlich ob alle wesentlichen Informationen ablesbar sind. Zum Beispiel die Unterscheidung von rechtem und linkem Teilbaum, oder welches Datum ein Knoten enthält.
- Implementiert nun alle vom Interface geforderten Methoden.

  Hinweis: Nutzt rekursive Hilfsmethoden um `toString` oder `getHeight()` zu implementieren.

  Da "\" im String ein Sonderzeichen ankündigt, müsst ihr "\\" für einen "\" schreiben. "\n" ist der Character für einen Zeilenumbruch.

  Für den iterativen `InorderIterator` kann es sinnvoll sein, sich zu merken, aus welcher Richtung ein Knoten erreicht wurde. Dies kann mittels eines `enum` gespeichert werden, also z. B. `public enum Dir{left, right, above;}`.

(b) Schreibt eine nicht abstrakte Unterklasse `DefaultBinTree<T>` von `BinTree<T>`. Sie soll das Interface `DefaultBinTreeInterface<T>` implementieren.

- Implementiert auch hier zwei Konstruktoren.
- Ihr könnt eure Klasse `DefaultBinTree<T>` testen, indem ihr die Klasse `DefaultBinTreeTest` ausführt. Diese Klasse enthält sogenannte *Assertions* der Form

  ```
  assert condition : expression;
  ```

  Ein solches Statement prüft ob `condition` wahr ist. Falls `condition` nicht erfüllt ist, wird das Programm terminiert und der Ausdruck `expression` ausgegeben. Im vorliegenden Fall ist `expression` einfach ein String mit einer Fehlermeldung. Ihr müsst der Virtual Machine sagen, dass Assertions geprüft werden sollen. In Eclipse geht das unter `Run -> Run Configurations -> Arguments`. Dort könnt ihr unter `VM Arguments` mit `-ea` Assertions anschalten. Bei der Abnahme eures Programms wird getestet, ob diese Klasse ohne Fehler ausgeführt werden kann.
- Schreibt selbst eine weitere Klasse für eigene Testcases. Implementiert mindestens eine Test-Methode die euren `InorderIterator` testet. Baut dafür einen Baum mit mindestens fünf Knoten auf, speichert das Resultat der Traversierung in einer Variable und testet mittels Assertions, ob sie dem zu erwartenden Ergebnis entspricht. Gebt außerdem auf der Konsole das Resultat eines Inorderdurchlaufes und der `toString`-Methode aus.

Viel Erfolg und Spaß!