Allgemeines Programmierpraktikum **Sommersemester 2017**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Institut für Informatik

Übung 2

Abgabe der Prüfsumme bis Mi., 10.05., 23:59 Uhr.
Testat von Do., 11.05., 12-14 Uhr bis Mi., 17.05., 16-18 Uhr.

Vorbereitung

Java

Lesen Sie das Kapitel *5.5 Abstrakte Datentypen* aus dem Skript *Informatik I, Wintersemester 16/17*, siehe in der Stud.IP-Veranstaltung *Dateien→ Vorlesung*.

Informieren Sie sich in der *Java Platform, Standard Edition 8 API Specification* `https://docs.oracle.com/javase/8/docs/api/` über die Collection `java.util.Stack` und die Schnittstelle `java.lang.Comparable`. Betrachten Sie jeweils die nicht typsichere Version.

Bemerkung

Bei den Aufgaben dieser Übung können *unchecked warnings*, z.B. die Folgende, ignoriert werden.

```
Note: xxx.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Aufgabe 1 – 25 Punkte

Rationale Zahlen

1. Implementieren Sie im Package `app.exercise.algebra` eine Klasse `Rational`.
   a) Die Klasse `Rational` speichert rationale Zahlen in einer Normalform für die Folgendes gilt.
      - Der Zähler und Nenner sind teilerfremd, das wird erreicht indem beide durch ihren größten gemeinsamen Teiler (ggt) geteilt werden. Der ggt kann z.B. mit dem Euklidischen Algorithmus ermittelt werden.
      - Der Nenner ist immer positiv, d.h. negative Brüche besitzen einen negativen Zähler.

b) Überschreiben Sie die von der Superklasse `java.lang.Object` geerbten Methoden `clone` und `toString` sinnvoll.

c) Überschreiben Sie die von der Superklasse `java.lang.Object` geerbten Methoden `equals` und `hashCode`. Stellen Sie die in der API für `equals` geforderten Eigenschaften (reflexive, symmetric, transitive, consistent) sicher. Sorgen Sie weiterhin dafür, dass für zwei Objekte, die von `equals` als gleich erkannt werden, `hashCode` denselben Wert zurückliefert, wie in der API gefordert.

d) `Rational` implementiert die Methoden der Schnitstelle `Arithmetic`.

```
public interface Arithmetic {
    // add operand to object
    void add(Arithmetic operand);
    // subtract operand from object
    void sub(Arithmetic operand);
    // multiply object by operand
    void mul(Arithmetic operand);
    // divide object by operand
    void div(Arithmetic operand);
}
```

2. Implementieren Sie im Package `app.exercise.algebra` eine Klasse `CompRational`, die von der Klasse `Rational` aus Aufgabenteil 1. erbt und die Schnittstelle `java.lang.Comparable` implementiert.

   Implementieren Sie die Methode `compareTo` der Schnittstelle `Comparable` wie in der API beschrieben. Stellen Sie weiterhin sicher, dass `(x.compareTo(y)==0) == (x.equals(y))` für zwei beliebige Objekte `x`, `y` der Klasse `CompRational` gilt.

3. Kommentieren Sie die Klasse`Rational` und die Schnittstelle `Arithmetic`. Erstellen Sie für die Klasse selbst und alle Attribute Dokumentationskommentare für `javadoc`. Erzeugen Sie mit `javadoc` eine API-Dokumentation, inklusive `private`-Attribute, der Klasse. Spärliche und/oder schlechte Kommentierung führt zu Punktabzug. Kommentieren Sie Ihren Programmtext ausführlich.

Aufgabe 2 – 25 Punkte

Umgekehrte polnische Notation

1. Schreiben Sie im Package `app.exercise.testing` eine ausführbare Klasse `RPN` (*Reverse Polish Notation*), die Ausdrücke in umgekehrter polnischer Notation auf der Kommandozeile übergeben bekommt und das Ergebnis ausgibt.

   Zulässige Ausdrücke sind wie folgt aufgebaut.

   - Zahlen sind ganzzahlig und nicht-negativ, d.h. sie bestehen nur aus Ziffern. Das schließt nicht aus, dass das Resultat eines Ausdrucks negativ oder rational ist.

- Operatoren sind `+` (Addition), `-` (Subtraktion), `*` (Multiplikation) und `/` (Division).

Nicht zulässige Ausdrücke werden erkannt und gemeldet.

Für die Arithmetik werden ausschließlich Variablen der Schnittstelle `Arithmetic` verwendet, in denen Referenzen auf Objekte der Klasse `Rational` oder `CompRational` gespeichert sind.

Der Stack, auf dem die Operanden abgelegt werden, ist ein Objekt der Klasse `java.util.Stack`.

2. Kommentieren Sie die Klasse ausführlich. Erstellen Sie für die Klasse selbst und alle Attribute Dokumentationskommentare für `javadoc`. Erzeugen Sie mit `javadoc` eine API-Dokumentation, inklusive `private`-Attribute, der Klasse. Spärliche und/oder schlechte Kommentierung führt zu Punktabzug. Kommentieren Sie Ihren Programmtext ausführlich.

3. Verwenden Sie `ant` zum automatisierten Übersetzen des Quelltext und zum Erzeugen der Dokumentation. Nach dem Übersetzen befinden sich die `java`-, die `class`-Dateien und die Dokumentation in unterschiedlichen Verzeichnissen.

Aufgabe 3 – 50 Punkte

Binärer Suchbaum

1. Programmieren Sie im Package `app.exercise.adt` (*Abstract Data Type*) eine Klasse `BinarySearchTree`, die einen binären Suchbaum für Referenzen auf Objekte von Klassen, die `java.lang.Comparable` implementieren, realisiert.

   Programmieren Sie mindestens folgende Methoden.

   - `insert` fügt ein Element in den Baum ein. Enthält der Baum schon ein äquivalentes Element (`compareTo == 0`) wird eine Exception ausgelöst.
   - `toString` liefert einen String zurück, der eine Aufzählung aller Elemente im Baum in aufsteigende Reihenfolge repräsentiert.

2. Schreiben Sie im Package `app.exercise.testing` einen ausführbare Klasse `TestBST`, die eine Liste von rationalen Zahlen auf der Kommandozeile übergeben bekommt und sortiert ausgibt.

   Die Kommandozeile enthält eine gerade Anzahl von ganzen Zahlen. Zwei aufeinander folgende Zahlen werden als Zähler und Nenner einer vergleichbaren rationalen Zahl (`CompRational`) interpretiert.

   Speichern Sie die rationalen Zahlen der Liste in einem binären Suchbaum und geben Sie die String-Repräsentation des Suchbaums aus.

3. Kommentieren Sie die Klasse ausführlich. Erstellen Sie für die Klasse selbst und alle Attribute Dokumentationskommentare für `javadoc`. Erzeugen Sie mit `javadoc` eine API-Dokumentation, inklusive `private`-Attribute, der Klasse. Spärliche und/oder schlechte Kommentierung führt zu Punktabzug. Kommentieren Sie Ihren Programmtext ausführlich.

4. Verwenden Sie `ant` zum automatisierten Übersetzen des Quelltext und zum Erzeugen der Dokumentation. Nach dem Übersetzen befinden sich die `java`-, die `class`-Dateien und die Dokumentation in unterschiedlichen Verzeichnissen.

Umgekehrte polnische Notation

Eine Modifikation der polnischen Notation ist die umgekehrte polnische Notation (UPN, Postfix-Notation), bei der der Operator hinter die verbundenen Operaden geschrieben wird.

- Zahlen sind Operanden
- Operatoren beschreiben Verknüpfungen zwischen zwei Operanden. Ein Operator wirkt auf die beiden Operanden direkt vor ihm. Die Anwendung eines Operators auf seine Operanden wird als Einheit betrachtet und ist wieder ein Operand.

Diese Forderungen macht Klammern und Bindungsregeln für Operatoren überflüssig.

Beispiel

```
  1 2 + = 3                          UPN
  1 2 + 3 * = 9
  1 2 3 * + = 7

  1 2 + 3 / 4 * 5 6 - +              UPN
= ((((1 2 +) 3 /) 4 *) (5 6 -) +)    Klammerung durch Operatoren erzungen
= ((((1 + 2) / 3) * 4) + (5 - 6))    Infix-Notation
= 3

  5 4 3 2 1 + * - /                  UPN
= (5 (4 (3 (2 1 +) *) -) /)          Klammerung durch Operatoren erzungen
= (5 / (4 - (3 * (2 + 1))))          Infix-Notation
= -1
```

Zur Auswertung eines UPN-Ausdrucks wird die Eingabe von links nach rechts durchlaufen. Wird eine Zahl gelesen, wird diese auf den Stack gelegt. Wird ein Operator gelesen, werden zwei Zahlen vom Stack geholt, mit dem Operator verknüpft und das Ergebnis auf den Stack gelegt. Nach dem Abarbeiten der Eingabe liegt eine Zahl, das Ergebnis des Ausdrucks, auf dem Stack.

Binäre Suchbäume

In einem binären Suchbaum gelten folgende Eigenschaften.

- Jeder Knoten enthält ein Element der gespeicherten Menge und hat genau zwei, möglicherweise leere, Unterbäume.
- Es gibt einen ausgezeichneten Knoten, die Wurzel, der zu keinem Unterbaum eines Knotens gehört. Alle anderen Knoten gehören zu genau einem Unterbaum.
- Die Knoten des linken Unterbaums eines Knotens enthalten nur Elemente, die kleiner sind als das im Knoten selbst gespeicherte Element.
- Die Knoten des rechten Unterbaum eines Knotens enthalten nur Elemente, die größer sind als das im Knoten selbst gespeicherte Element.

Diese Eigenschaften gelten rekursiv für die Unterbäume und müssen z.B. bei Einfügeoperationen erhalten bleiben.

Einfügen

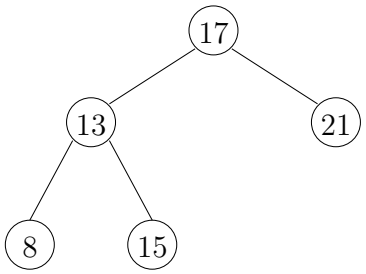
Vorraussetzung, um eine Element in einen binären Suchbaum einzufügen, ist, das der Suchbaum dieses Element (bzw. ein identisches Element) nicht bereits enthält.

Um die Einfügeposition zu bestimmen wird der Baum, beginnend mit der Wurzel als Laufknoten, durchsucht.

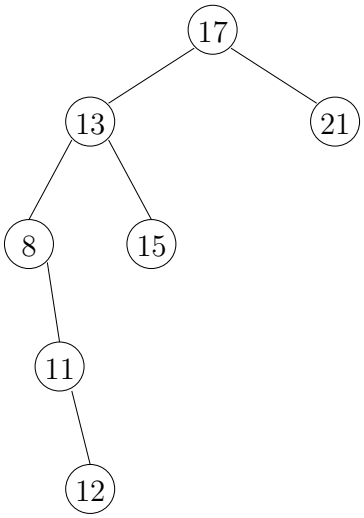
- Das einzufügende Element ist kleiner als das Element am Laufknoten.
  - Ist der linke Unterbaum des Laufknotens leer, erzeuge einen neuen Knoten mit dem einzufügenden Element und zwei leeren Unterbäumen, der zum linken Unterbaums des Laufknotens wird und das Einfügen ist beendet.
  - Ist der linke Unterbaum des Laufknotens nicht leer, setze den Laufknoten auf den ersten Knoten des linken Unterbaums und führe die Suche fort.
- Das einzufügende Element ist größer als das Element am Laufknoten.
  - Ist der rechte Unterbaum des Laufknotens leer, erzeuge einen neuen Knoten mit dem einzufügenden Element und zwei leeren Unterbäumen, der zum rechten Unterbaums des Laufknotens wird und das Einfügen ist beendet.
  - Ist der rechte Unterbaum des Laufknotens nicht leer, setze den Laufknoten auf den ersten Knoten des rechten Unterbaums und führe die Suche fort.

Beispiel

Ausgangssituation.



Ausgangssituation gefolgt von
`insert(11); insert(12);`
ergibt folgenden Baum.



Ausgangssituation gefolgt von
`insert(12); insert(11);`
ergibt folgenden Baum.