```java
package uebung1;

import java.math.BigInteger;
import java.util.Comparator;

public class Bruch implements Comparable<Bruch> {
    private int z;
    private int n;

    // Getters
    public int getZ() {
        return this.z;
    }

    public int getN() {
        return this.n;
    }

    /**
     * Bruch-Konstruktor
     * @param z Zähler
     * @param n Nenner
     * @throws IllegalArgumentException Nenner darf nicht 0 sein
     */
    public Bruch(int z, int n) throws IllegalArgumentException{
        if(n == 0){
            throw new IllegalArgumentException("Nenner darf nicht 0 sein");
        }

        this.z = z;
        this.n = n;
    }

    @Override
    public int compareTo(Bruch b){
        if(this.ausrechnen() < b.ausrechnen()){
            return -1;
        } else if(this.ausrechnen() == b.ausrechnen()){
            return 0;
        } else return 1;
    }

    /**
     * Generiert einen String aus dem Bruch
     * @return String im Format "(Zähler/Nenner)"
     */
    public String toString(){
        return "(" + this.z + "/" + this.n + ")";
    }

    public String toStringPlusDecimal(){
        return "(" + this.z + "/" + this.n + ")" + " (" + this.ausrechnen() + ")";
    }

    /**
     * Multipliziert zwei Brüche.
     * @param b Bruch mit dem multipliziert wird
     * @return Neuer Bruch
     */
    public Bruch multiplizieren(Bruch b){
```

```
61          return new Bruch(this.z * b.z, this.n * b.n);
62      }
63
64      public float ausrechnen(){
65          return (float)this.z/this.n;
66      }
67
68      public Bruch kuerzen(){
69          BigInteger biZ = BigInteger.valueOf(this.z);
70          BigInteger biN = BigInteger.valueOf(this.n);
71          int gcd = biZ.gcd(biN).intValue();
72          return new Bruch(this.z/gcd, this.n/gcd);
73      }
74
75      public Bruch kehrwert(){
76          return new Bruch(this.n, this.z);
77      }
78
79      public Bruch dividieren(Bruch b){
80          return this.multiplizieren(b.kehrwert());
81      }
82  }
83
84  /**
85   * Comparator-Implementierung für Vergleich der Dezimalwerte von Brüchen
86   */
87  class WertComparator implements Comparator<Bruch> {
88      @Override
89      public int compare(Bruch a, Bruch b){
90          if(a.ausrechnen() < b.ausrechnen()){
91              return -1;
92          }
93          else if(a.ausrechnen() == b.ausrechnen()){
94              return 0;
95          }
96          else{
97              return 1;
98          }
99      }
100 }
101
102 /**
103  * Comparator-Implementierung für Vergleich der Zähler von Brüchen
104  */
105 class ZaehlerComparator implements Comparator<Bruch> {
106     @Override
107     public int compare(Bruch a, Bruch b){
108         if(a.getZ() < b.getZ()){
109             return -1;
110         }
111         else if(a.getZ() == b.getZ()){
112             return 0;
113         }
114         else{
115             return 1;
116         }
117     }
118 }
```