2. PR - HÜ in Java - Klasse Bruch

Definieren Sie eine Klasse `Bruch` zum Verarbeiten von Bruchzahlen. Jedes Objekt dieser Klasse kapselt zwei Instanzvariable vom Typ `long` für Zähler und Nenner. Es sind mindestens die unten angegebenen Konstruktoren und Instanzmethoden zu implementieren:

```
public Bruch()                 // erzeugt das Bruchobjekt 0/1
public Bruch(long z)           // erzeugt das Bruchobjekt z/1
public Bruch(long z, long n)   // erzeugt das Bruchobjekt z/n
public Bruch(String bruch)     // parst den String
public Bruch(double wert)      // erzeugt das passende Bruchobjekt

public Bruch add(Bruch b)      // liefert this + b
public Bruch sub(Bruch b)      // liefert this - b
public Bruch mult(Bruch b)     // liefert this * b
public Bruch div(Bruch b)      // liefert this / b
public Bruch hoch(int e)       // liefert this ^ e

public void addThis(Bruch b)   // this += b
public void subThis(Bruch b)   // this -= b
public void multThis(Bruch b) // this *= b
public void divThis(Bruch b)   // this /= b
public void hochThis(int e)    // this ^= e

private void simplify()        // kürzt und macht Nenner positiv
public String toString()       // liefert Stringdarstellung
public boolean equals(Object o)   // vergleicht 2 Bruchobjekte
```

Verwenden Sie außerdem die internen Funktionen:

```
private static long ggT(long a, long b)
                    // berechnet den größten gemeinsamen Teiler
private static long kgV(long a, long b)
                    // berechnet das kleinste gemeinsame Vielfache
private void setZaehlerUndNenner(long zaehler, long nenner)
                    // setzt Zähler und Nenner des this – Objektes
```

Die Methode `toString()` liefert je nach Zustand einer privaten boolschen Klassenvariablen eine Bruchdarstellung oder eine Dezimalzahl. Diese boolsche Variable kann von außen mit Hilfe von Zugriffsfunktionen abgefragt und verändert werden.

Ungültige Bruchobjekte werden nicht erzeugt. Sollte aus welchen Gründen auch immer ein ungültiges Bruchobjekt entstehen, so wird eine eigene Exception vom Typ `BruchException` (checked Exception) geworfen. Auf eventuell auftretende `BruchException`s ist angemessen zu reagieren (sinnvolles Exception-Handling!).
Wird bei einer Rechenoperation kein echtes Bruchobjekt (`null`) übergeben, so ändert dies für die Methoden welche das `this`-Objekt verändern (z.B. `addThis`) das `this`-Objekt nicht – es bleibt unverändert (keine Exception), hingegen wird bei den echten Rechenmethoden, welche ein neues Bruch-Objekt rückliefern (z.B. `add`) eine Exception (`BruchException`) geworfen.

Testen Sie die einzelnen Funktionen mittels der entsprechenden JUnit - Tests.

Schreiben Sie auch ein geeignetes Testprogramm (Rechner) für die Klasse `Bruch`.