Fingerübungen OOP

A1- Quadratische Funktion

Das Projekt bluej-quadratische-funktion enthält eine Klasse, die eine quadratische Funktion modelliert.

- Überlege, welche Informationen gespeichert werden müssen, um eine quadratische Funktion vollständig zu beschreiben.
- Schreibe einen Konstruktor, um eine quadratische Funktion zu erzeugen.
- Die Klasse soll die folgenden Methoden anbieten:
  - `getFunktionswert(double x)`: liefert den Funktionswert an einer Stelle x
  - `getScheitelX()`: liefert den x-Wert des Scheitels
  - `getScheitelY()`: liefert den y-Wert des Scheitels
  - `getAnzahlNullstellen()`: liefert die Anzahl der Nullstellen

Ersetzen an den Stellen, an denen noch TODO steht, den bestehenden Code durch deine Implementation. Klicke links auf "Tests starten", um automatisch 100 Testfälle ausführen zu lassen - so kannst du überprüfen, ob deine Lösung stimmt.

A2 - Notenverwaltung

Das Projekt bluej-notenverwaltung enthält eine Klasse namens Klausur, die das Ergebnis einer Oberstufenklausur modelliert. Sie bietet die folgenden Methoden an:

- `getAnzahl()` – die Gesamtzahl der erfassten Noten
- `getDurchschnitt()` – die Durchschnittsnote der erfassten Noten
- `getBesteNote()` und getSchlechtesteNote() – die beste bzw. schlechteste erfasste Note
- `noteEintragen(int note)` – fügt eine neue Note zur Erfassung hinzu
- `reset()` – löscht alle bisher erfassten Einträge aus dem Kurs

Überlege dir, wie du das Ergebnis modellieren möchtest. Es sollen wie üblich keine Variablen nach außen hin sichtbar sein – der Zugriff darf nur über die oben aufgezählten Methoden geschehen.

Implementiere die Methoden, in denen noch TODO steht.

Klicke links auf "Tests starten", um automatisch 100 Testfälle ausführen zu lassen.

03 - Brüche

Das Projekt enthält eine Klasse namens Bruch, die einen Bruch repräsentiert. Sie bietet die folgenden öffentlichen Methoden an:

- `Bruch addieren(Bruch b)`
- `Bruch subtrahieren(Bruch b)`
- `Bruch multiplizieren(Bruch b)`
- `Bruch dividieren(Bruch b)`
- `int getZaehler()`

- `int getNenner()`

Die ersten vier Methoden führen die Grundrechenarten mit dem aktuellen Bruch (`this`) und dem übergebenen Bruch b aus. Das Ergebnis ist stets ein neues Objekt, d.h. `this` ändert sich durch einen Methodenaufruf nicht. Orientiere dich an der Methode multiplizieren, um zu sehen, wie ein neues Bruch-Objekt erzeugt und zurückgegeben wird.

Dazu kommen die (überladenen) Konstruktoren:

- `public Bruch(int zaehler, int nenner)`
- `public Bruch(int wert)` – erzeugt einen Bruch mit dem Nenner 1.

Der Nenner eines Bruchs muss immer positiv sein. Dafür sollte im Konstruktor gesorgt werden. Der Aufruf `new Bruch(1,2)` sollte also den gleichen Bruch erzeugen wie `new Bruch(-1,-2)`.

Implementiere alle Stellen, an denen derzeit noch TODO steht.

Beispiele für die Verwendung:

```
Bruch a = new Bruch(1, 3); // repräsentiert die Zahl 1/3
Bruch b = new Bruch(1, 4); // repräsentiert die Zahl 1/4
Bruch c = a.addieren(b);
// c repräsentiert 1/3 + 1/4 = 7/12
// a ist weiterhin 1/3, b ist weiterhin 1/4
```

Lassen die 100 Testfälle in der Testklasse `BruchTester` ausführen, um deine Lösung zu kontrollieren.

`Bonusaufgabe für Fortgeschrittene:`

Sorge dafür, dass deine Brüche immer weitestmöglich gekürzt sind. Um kürzen zu können, benötigt man den größten gemeinsamen Teiler von Zähler und Nenner. Diesen erhält man effizient mit dem Euklidischen Algorithmus (→ Google oder Wikipedia). Lasse dann die Testfälle in der Testklasse `BruchTester2` ausführen.