Allgemeine Hinweise: Fügen Sie Kommentare hinzu, sobald Sie es für sinnvoll halten. Der Code auf dem `main`-Branch wird als Abgabe gewertet. Abgaben, bei denen die Kompilierung fehlschlägt, werden mit 0 Punkten bewertet. Melden Sie sich bei Fragen und Problemen in den Tutorien oder in Teams. Die Designeentscheidungen in den Übungsaufgaben entsprechen aus Gründen der Vereinfachung nicht immer denen, die man in der Realität antreffen würde.

1 Repository einrichten (0 P)

Ab dieser Woche erfolgt die Abgabe in Kleingruppen. Sie finden das Abgaberepository Ihrer Gruppe (in Kürze) unter `https://gitlab.rlp.net/eis22`. Dieses Repository ist eine Fork des zentralen Repositorys, in das ab jetzt Code und Tests zu den Übungsblättern gepusht werden. Mehr Details zum Abgabeworkflow finden Sie auf Blatt 1.

2 Wiederholung (0 P)

Lesen Sie zur Wiederholung der dieswöchigen Vorlesung den Artikel unter `https://docs.scala-lang.org/scala3/book/taste-vars-data-types.html`.

3 Mehr Funktionen für Brüche (25 P)

Ergänzen Sie die **unveränderliche** Bruchklasse aus der Vorlesung um die unten genannten Funktionen. Schreiben Sie außerdem einen sinnvollen Scaladoc-Kommentar für jede der Funktionen!

(a) Wenn der Nenner den Wert 1 hat, soll die vorhandene `print`-Funktion nur den Zähler printen.

(b) `def unary_- : Rational` soll den Bruch negieren.

(c) `def reciprocal: Rational` soll den Kehrwert des Bruchs zurückgeben.

(d) `reciprocal` soll eine `ArithmeticException` mit sinnvoller Fehlermeldung werfen, wenn der Zähler den Wert 0 hat.

(e) `def -(other: Rational): Rational` soll zwei Brüche subtrahieren.

(f) Es soll möglich sein, Ganzzahlen (`Int`) von Brüchen abzuziehen.

(g) Es soll möglich sein, Brüche von Ganzzahlen (`Int`) abzuziehen.

(h) `def *(other: Rational): Rational` soll zwei Brüche multiplizieren.

(i) `def /(other: Rational): Rational` soll zwei Brüche dividieren.

(j) Die Division soll eine `ArithmeticException` mit sinnvoller Fehlermeldung werfen, wenn der Divisor den Wert 0 hat.

(k) `def min(other: Rational): Rational` soll das Minimum zweier Brüche zurückgeben.

(l) `def toString: String` soll den Bruch als Zeichenkette zurückgeben. Wenn der Nenner den Wert 1 hat, soll nur der Zähler als Zeichenkette zurückgegeben werden. Beachten Sie, dass der Bruch hier *nicht* geprintet werden soll!

Codestil-Konventionen

1. Methodennamen und Variablennamen starten mit einem Kleinbuchstaben. Worttrennungen werden dadurch kenntlich gemacht, dass der erste Buchstabe des Folgeworts großgeschrieben wird (*Camel-Case-Konvention*). Unterstriche sollen (bis auf Ausnahmen) vermieden werden.
   Beispiele: `def cartesianProduct` oder `val lengthSquared`
2. Alle Methodennamen und Variablennamen sollten so gewählt werden, dass sich aus dem Namen auf den Sinn der Methode oder Variable schließen lässt.
3. Klassennamen und Typbezeichner werden wie Variablen benannt, starten aber immer mit einem Großbuchstaben (*Pascal-Case-Konvention*). Beispiele: `class Table` oder `trait AutoClosable` oder `type KeyType`
4. Programmkonstanten werden wie Variablen benannt, aber starten ebenfalls mit einem Großbuchstaben. Programmkonstanten sind Werte, die vom Programmierer fest einkodiert werden und sich während der Ausführung des Programms nie verändern. Beispiele: `math.Pi` oder `val UserLimit = 10`
5. In Scala werden zwei Leerzeichen pro Einrückung genutzt (nicht 4 oder 8)!
6. Alle binären Operatoren (`=`, `+`, `+=`, usw.) sollen von einem Leerzeichen auf jeder Seite umgeben werden. Hinter einem Komma soll ein Leerzeichen stehen, davor aber nicht. Dasselbe gilt für Doppelpunkte zur Typannotation!
   Beispiele: `(x + 1) / 2` oder `def add(a: Int, b: Int): Int`
7. Halten Sie (sofern möglich) Ihre Zeilen unter 120 Zeichen.
8. Normale Kommentare werden mit `//` (einzeilig) oder `/*` (mehrzeilig) eingeleitet. Scaladoc-Kommentare werden mit `/**` eingeleitet und müssen eine bestimmte Formvorgabe einhalten. Nutzen Sie normale Kommentare für informelle Dokumentation und zur Beschreibung von Implementierungsdetails.
9. Sie dürfen sich aussuchen, ob Sie Blöcke durch Klammern (VL-Stil) oder nur durch Einrückungen (neuer Scala3-Stil) kenntlich machen. Sie sollen allerdings nicht in einer Abgabe zwischen beiden Stilen hin- und herwechseln!
10. Wenn Sie die Wahl zwischen `val` und `var` haben, nutzen Sie `val`.
11. Bei Methoden ohne Argumente lassen Sie die Klammern immer dann weg, wenn die Funktion keine Seiteneffekte hat. Seiteneffekte umfassen das Lesen oder Schreiben in eine Datei, Prints und Interaktion mit der Standardeingabe, und das Modifizieren von nicht-lokalen Variablen.
    Beispiele: `def toString: String` oder `def clearBuffer(): Unit` oder `def printMaximum(): Unit`