Praktikum D: Fehlersuche und Debugging

Programmierung III

Wintersemester 2021/22
Prof. Dr.-Ing. Marco Block-Berlitz
Prof. Dr.-Ing. Dietrich Kammer
Dipl.-Ing. Jan Roeper

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Debugging

D01

In einer früheren Übung sollte die Summe $\sum_{i=1}^{100} \frac{i \cdot (i+1)}{2}$ in einer Funktion berechnet werden. Leider liefert der folgende Programmcode nicht das gewünschte Ergebnis. Finden Sie die Fehler:

```
public static double sum1() {
   int i, startwert = 1;
   double d, h;
   for (i=--startwert; i>100; i++)
      System.out.println(d);
      {h=(i*i*i)/2;
      d=d+h;
   }
   return d;
}
```

D02

Welches Ergebnis liefern die beiden Programmzeilen `c = c++;` und `c = ++c;` und warum?

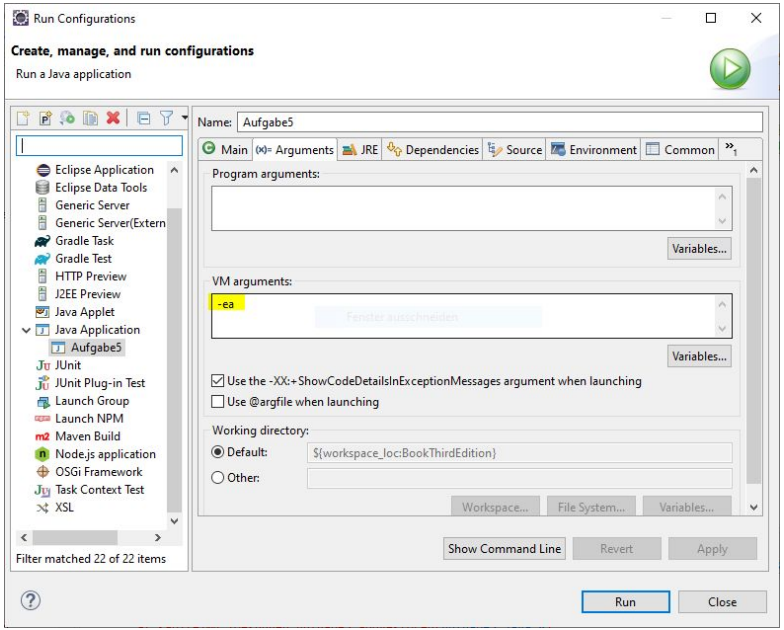
Assertions und Exceptions

D03

Ergänzen Sie geeignete Assertions in Ihrem Programm zur Vektorarithmetik, um eine korrekte Eingabe für eine der Funktionen sicherzustellen!.

D04

Ergänzen Sie nun Exceptions in den anderen Funktionen, falls die Eingabeparameter nicht korrekt (gleich lang) sind.
Diskutieren Sie Vor- und Nachteile der beiden Varianten zur Fehlerbehandlung!



Stellen Sie sicher, dass in der “Run Configuration” das VM-Argument “-ea” gesetzt ist, damit Java die Assertions auch auswertet. Sie rufen diesen Dialog über den Pfeil neben dem “Play”-Button auf.

Näherungsverfahren für PI

D05

Das **Wallis-Produkt** wurde von John Wallis 1655 formuliert und gibt eine Näherung für $\pi/2$ mit der folgenden Vorschrift an:

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \ldots = \prod_{i=1}^{\infty} \frac{4i^2}{(2i-1) \cdot (2i+1)}$$

Implementieren Sie dieses Verfahren und geben Sie damit eine Näherung für π an. Wenn Ihnen das keine Schwierigkeiten bereitet hat, dann erweitern Sie Ihre Funktionssammlung zur Näherung von π doch noch um die beiden folgenden:

$$1 + \frac{1}{3} - \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \frac{1}{11} - \ldots = \frac{\pi}{2 \cdot \sqrt{2}} \, (Newton)$$

$$1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{5} + \frac{1}{7} - \frac{1}{8} + \ldots = \frac{\pi}{3 \cdot \sqrt{3}} \, (Euler)$$

Wenn Sie das Thema der Annäherungen von π noch weiter fesselt, dann schauen Sie sich die sehenswerte Wikipedia-Seite dazu an und implementieren Sie weitere Verfahren.

Laufzeiten

D06

Wir haben die Madhava-Leibniz-Reihe als Funktion bereits implementiert. Leider ist die Konvergenzgeschwindigkeit der Reihe sehr langsam, denn *n* korrekte Nachkommastellen erhalten wir nach ½10^n Iterationen. Erweitern Sie den folgenden Programmabschnitt zum Abgleich mit der bereits definierten Konstanten `Math.PI`:

```
public static void main(String[] args) {
   int[] durchlauf = {5, 50, 5000, 50000, 500000, 5000000,
                      50000000, 500000000};
   char[] pi_leibniz, pi_biblio;
   for (int j=0; j<durchlauf.length; j++) {
      long startZeit   = System.currentTimeMillis();
      double piLeibniz = piMadhaveLeibniz(durchlauf[j]);
      long stopZeit    = System.currentTimeMillis();
      // Umwandlung der double Werte in char-Arrays
      pi_leibniz = (Double.toString(piLeibniz)).toCharArray();
      pi_biblio  = (Double.toString(Math.PI)).toCharArray();
      // TODO: Vergleich der beiden Listen und Speichern der
      // Übereinstimmungen in s

      // Ausgabe der Iterationsanzahl und benötigter Zeit
      System.out.println("Iterationen: " + durchlauf[j] +
         " (" + (stopZeit-startZeit) + ") ms");

      // Ausgabe des gemeinsamen Teils
      System.out.println("Genauigkeit: " +
         (String)(Double.toString(Math.PI)).subSequence(0, s));
      System.out.println();
   }
}
```

Arbeit mit dem Hoare-Kalkül

WICHTIG

Schauen Sie sich die Vorlesung zum Hoare-Kalkül aufmerksam an und vollziehen Sie die Beispiele sorgfältig nach.