3. Übung (Wochenendzettel) zur Veranstaltung *Proinformatik: Objektorientierte Programmierung*

Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik, SoSe 2013
Prof. Dr. Marco Block-Berlitz

1. 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 $\frac{1}{2}10^n$ 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, 500, 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 = piMadhavaLeibniz(durchlauf[j]);
      long stopZeit    = System.currentTimeMillis();

      // wir wandeln die beiden double-Werte in Arrays
      // vom Datentyp char um
      pi_leibniz = (Double.toString(piLeibniz)).toCharArray();
      pi_biblio  = (Double.toString(Math.PI)).toCharArray();

      /* TODO:
       * In diesem Abschnitt sollen die ersten Zeichen der
       * beiden Listen pi_leibniz und pi_biblio verglichen
       * und die Anzahl der Übereinstimmungen in s
       * gespeichert werden
       */

      // Ausgabe der Iterationsanzahl und der benötigen 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();
    }
}
```

Falls Sie Aufgabe 2 erfolgreich bearbeitet haben, könnten Sie die Näherung vom Wallis-Produkt in die Vergleichsklasse ebenfalls einbauen. Leider ist die Konvergenzgeschwindigkeit der Reihe sogar noch etwas langsamer, denn n korrekte Nachkommastellen erhalten wir nach $\frac{3}{5}10^n$ Iterationen.

2. In der Vorlesung haben wir die Lösung von Aufgabe 6 aus Übung 2 so verändert, dass der Speicherüberlauf im positiven Bereich erkannt wird. Die Methode soll nun ebenfalls `null` zurückliefern, wenn ein Speicherüberlauf im negativen Bereich stattfindet. Schreiben Sie zuerst einen Test, der das überprüft. Erkennen Sie, warum der Test nicht fehlschlägt? Ja, das sind die Tücken des Programmierens. Ergänzen Sie anschließend die Lösung.

3. Die Rückgabe von `null` oder ein Programmabbruch bei einem fehlerhaften Aufruf ist kein guter Programmierstil, wie wir bereits wissen. In der letzten Aufgabe hatten wir allerdings davon Gebrauch gemacht und wollen das jetzt verbessern. Sollte ein Speicherüberlauf stattfinden, soll die Methode eine Exception werfen, die von einem Test erfolgreich identifiziert wird.