Universität Leipzig **WS 14/15**
Institut für Informatik **Dr. Monika Meiler**

Modellierung und Programmierung 1
Übungsserie 3

Abgabetermin: 30.11.2014, 23:55 Uhr

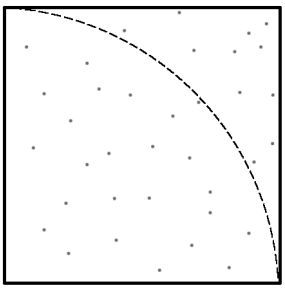
Grundsätzlich sind Nebenrechnungen anzugeben und Antworten zu begründen.
Einzureichen sind, bei mehreren Dateien als .zip-Archiv:
Lösungen als .pdf-Datei, Programme als Quellcode, Ergebnisdateien.

Die ersten 100 Nachkommastellen der Ludolphschen Zahl[1] π:
π periphereia (Randbereich), perimeter (Umfang)

$\pi \approx$ **3, 141 592 653 589 793 238 462 643 383 279 502 884 197 169 399 375 105 820 974 944 592 307 816 406 286 208 998 628 034 825 342 117 067 9**

1. **Anweisungen - Monte Carlo Verfahren**

   Gegeben sei ein Quadrat mit den Eckpunkten $(0,0), (0,1), (1,0)$ und $(1,1)$, in dem ein Kreisviertel mit dem Radius 1.0 einbeschrieben ist. Innerhalb des Quadrates werden nun n Punkte mit zufälliger x- und y-Koordinate ($(x,y) \in [0,1] \times [0,1]$) verteilt.



   a)
   - Welchem Verhältnis entspricht die Menge aller Punkte unterhalb des Kreisbogens zu der Gesamtpunktmenge?
   - Welche Bedingung erfüllt ein Punkt (x,y) unterhalb des Kreisbogens bzgl. seines euklidischen Abstandes zum Nullpunkt $(0,0)$?
   - Formulieren Sie mittels der vorrangegangenen Überlegungen einen Algorithmus als Grobstruktur für ein Programm **MonteCarlo.java**, welcher mit n zufällig erzeugten Punkten die Zahl π näherungsweise bestimmt.

   b) Implementieren Sie durch schrittweise Verfeinerung Ihre Grobstruktur **MonteCarlo.java**. Testen Sie es mit $n = 10^3, 10^6$ und 10^9 und berechnen Sie jeweils den *absoluten* Fehler zu dem Wert der Java-Konstanten **Math.PI**. Leiten Sie die Ausgaben in eine Datei **MonteCarlo.out** um.
   Hinweise:
   - **double Math.random()** liefert eine Zufallszahl $r \in [0,1)$ zurück.
   - Die Berechnung für sehr große n kann längere Zeit in Anspruch nehmen.

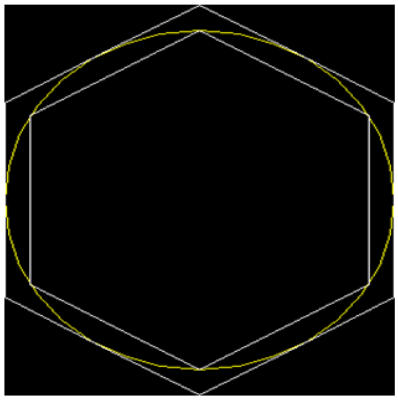
[1] Ludolf van Ceulen (1540-1610)

2. **Anweisungen - Methode von Archimedes**

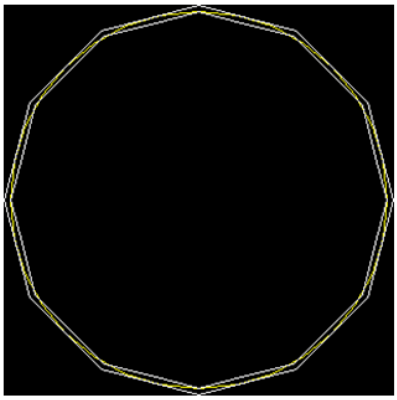
Archimedes[2] berechnete den Umfang eines n-Ecks, welches einen Kreis mit dem Durchmesser $d = 1$ umschreibt, und den eines n-Ecks, welches vom Kreis umschrieben wird. Je größer n wird, desto näher liegt das arithmetische Mittel dieser beiden Werte am wahren Wert von π.
Bei dem hier vorgestellten Iterationsverfahren handelt es sich um eine Variante, die für $n = 6 * 2^i$ - Ecken optimiert ist (a_i - Umfang des äußeren n-Eck, b_i - Umfang des inneren n-Eck):

Iterationsanfang	$a_0 = 2\sqrt{3}$	$b_0 = 3$
Iterationsschritt	$a_{i+1} = \frac{2a_ib_i}{a_i+b_i}$	$b_{i+1} = \sqrt{a_{i+1}b_i}$

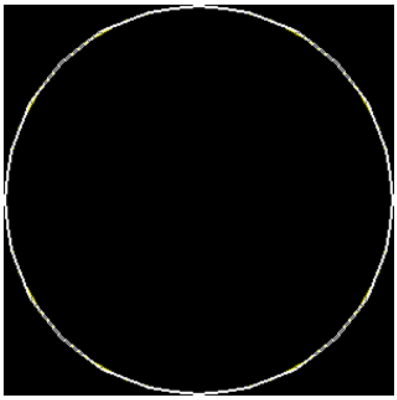
Archimedes fing mit einem 6-Eck an. Er verdoppelte diesen Wert viermal bis zum 96-Eck und schachtelte damit den Wert von π auf das Intervall $(3\frac{10}{71}, 3\frac{1}{7})$ ein. Er erhielt (handschriftlich!) den Schätzwert von $\pi \approx$ **3, 141**635. Bis Mitte des 17. Jahrhunderts griffen fast alle Versuche π zu berechnen auf diese Methode zurück. Ludolf van Ceulen berechnete als Näherung $\pi \approx$ **3, 141 592 653 589 793 238 462 643 383 279 502 88**.



6-Eck ($i = 0$)
$3{,}00 < \pi < 3{,}46$



12-Eck ($i = 1$)
$3{,}11 < \pi < 3{,}22$.



24-Eck ($i = 2$)
$3{,}13 < \pi < 3{,}16$.

a) Entwickeln Sie ein Programm ***Archimedes.java*** zur Berechnung eines Näherungswertes von π mittels des Algorithmus von Archimedes. Ermitteln Sie die notwendige Schrittanzahl und berechnen Sie den ***absoluten*** Fehler des berechneten Wertes zu dem Wert der Java-Klassenkonstanten **Math.PI**. Leiten Sie die Ausgaben in eine Datei ***Archimedes.out*** um. Hinweis: Das Verfahren bricht ab, wenn sich a_i und b_i rechentechnisch nicht mehr unterscheiden.

b) Analysieren Sie die Methode von Archimedes im Vergleich mit dem Monte Carlo Verfahren: Wie verläßlich sind die zu erwartenden Ergebnisse?

3. **Felder**

Eine Klasse **RandomArray** soll Klassenmethoden zur statistischen Auswertung von Zufallsfeldern zusammenstellen:

a) Schreiben Sie eine Klassenmethode **randomArray**, welche n natürliche Zufallszahlen x_i aus einem Bereich von 1 bis m in ein eindimensionales Feld vom Typ **long** abspeichert.
Implementieren Sie eine Klassenmethode **ausgabe** zum Ausgeben von eindimensionalen Feldern.
Schreiben Sie als Hauptmethode **main** ein Testprogramm, welches n und m einliest, ein Zufallsfeld erzeugt und dieses anschließend ausgibt.

b) Implementieren Sie eine Klassenmethode **bubbleSort**, welche ein Feld aufsteigend sortiert. Nutzen Sie hierfür den *Bubble-Sort*-Algorithmus.

[2] Archimedes von Syrakus(um 287-212 v.u.Z.)

Erweitern Sie Ihr Testprogramm **main**, indem es eine *Kopie* des Zufallsfeldes sortiert und diese ausgibt.

Hinweis: *Bubble-Sort* ist ein vergleichsbasierter Sortieralgorithmus. Durchlaufen Sie Ihr Feld in aufsteigender Richtung und vergleichen Sie dabei sukzessive alle direkt benachbarten Komponenten. Stehen zwei benachbarte Komponenten in falscher Reihenfolge, werden diese einfach miteinander vertauscht. Diese Vergleiche werden solange fortgesetzt, bis das Feld sortiert ist. Dabei endet jeder Durchlauf immer eine Komponente früher.

c) Berechnen Sie in einer Klassenmethode **mittelWert** den Mittelwert $\overline{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i$ und in einer weiteren Klassenmethode **varianz** die Varianz $s^2 = \frac{1}{n-1} \sum_{i=0}^{n-1} (x_i - \overline{x})^2$ eines Feldes.

Vervollständigen Sie Ihr Testprogramm **main**, indem es den Mittelwert und die Varianz des Zufallsfeldes berechnet und ausgibt.
Erzeugen Sie mit Ihrem Testprogramm drei Zufallsfelder für drei selbstgewählte Werte n und m und geben sie jeweils das unsortierte Feld, das sortierte Feld, den Mittelwert und die Varianz mittels Ausgabeumlenkung in eine Datei **RandomArray.out** aus.

RandomArray.java (Grobstruktur)

```
public class RandomArray
{
   public static long[] randomArray( int n, int m){ }
   public static void ausgabe( long[] x){ }
   public static long[] bubbleSort( long[] x){ }
   public static double mittelWert( long[] x){ }
   public static double varianz( long[] x){ }
   public static void main(String[] args)
   {
      // a) Erzeugen und Ausgabe eines Zufallsfeldes
      // b) Sortieren und Ausgabe einer Kopie der Feldes mit Bubble-Sort
      // c) Berechnen und Ausgabe des Mittelwertes und der Varianz
   }
}
```

4. **Strukturen (Klassen)**

Gegeben seien die folgenden zwei Klassen und ein Objekt:

```
public class Pruefung
{
   String fach;
   float note;
   int wiederholungen;
}

public class Student
{
   String name, vorname;
   int matrikelnr;
   Pruefung[] abgelegtePruefungen;
}

Student balthasar;
```

a) Beschreiben Sie ***inhaltlich*** die Bedeutung des folgenden Java-Konstrukts:

```
balthasar.abgelegtePruefungen.length == 0
```

b) Welcher Wert wird in der Variablen x berechnet? Begründen Sie ***inhaltlich*** Ihre Vermutung.

```
float x = 0.0f;

for( int i = 0; i < balthasar.abgelegtePruefungen.length; i++)
{
   x += balthasar.abgelegtePruefungen[ i].note;
}

if( balthasar.abgelegtePruefungen.length != 0)
   x /= balthasar.abgelegtePruefungen.length;
```

c) Schreiben Sie analog zu den gegebenen Klassen eine Klasse **Mitarbeiter** und eine Klasse **Firma** für die Verwaltung der Mitarbeiter einer Firma. Für jeden Mitarbeiter sind Name, Wohnort und Gehalt zu speichern. Die Firmen-Klasse soll einen Namen, eine Anschrift und eine Liste aller Mitarbeiter beinhalten.
Schreiben Sie anschließend ein Java-Konstrukt, welcher den Mitarbeiter einer Firma mit dem größten Gehalt ermittelt, dessen Namen sowie sein Gehalt ausgibt.