Zufall in Java-Programmen

1. Grundbegriffe

Zufallszahlen (und andere zufällig gewählte Werte) werden in verschiedenen Programmen verwendet, z.B. in *Glücksspiel*-Programmen und in Programmen zum *Erzeugen von Testdaten.*

Für *Glücksspiele* braucht man sogenannte *nicht-reproduzierbare Zufallszahlen*, d.h. bei jeder Ausführung des Glückspiel-Programms soll der Generator andere, nicht vorhersehbare Zufallszahlen liefern.

Beim *Erzeugen von Testdaten* braucht man dagegen in aller Regel *reproduzierbare Zufallszahlen*. Falls das Testprogramm einen Fehler entdeckt und gemeldet hat, soll es bei jedem erneuten Start denselben Fehler wieder melden. Das geht aber nur, wenn der verwendete Zufallszahlen-Generator bei jeder Ausführung des Testprogramms genau die gleichen Zufallszahlen liefert.

Somit unterscheidet man zwischen *reproduzierenden* und *nicht-reproduzierenden* Zufallszahlen-Generatoren.

Ein (reproduzierender oder nicht-reproduzierender) Zufallszahlen-Generator kann mehrere Methoden enthalten, die zufällig gewählte Werte liefern. Diese Methoden können sich vor allem durch 3 Eigenschaften unterscheiden:

- durch ihren Ergebnis-*Typ*,
- ihren *Bereich* und
- ihre *Wahrscheinlichkeitsverteilung.*

Typ: Von welchem *Typ* sind die gelieferten Zufallswerte?
`int?` `float?` `double?` `boolean?` `char?` `...`

Bereich: Aus welchem *Bereich* stammen die gelieferten Zufallswerte? Aus dem *gesamten* Wertebereich des betreffenden Typs oder nur aus einem *Teilbereich*?

Wahrscheinlichkeitsverteilung: Haben alle Werte des Bereichs *die gleiche Chance*, geliefert zu werden? Dann sagt man: Die Werte sind *gleichverteilt* (engl. They are uniformly distributed).
Oder sind die Werte anders verteilt, z.B. Gauss-verteilt (engl. Gaussian distributed) oder Cauchy-Lorentz-verteilt oder ... ?

Anmerkung-1: Die *Gauss-Verteilung* wird auch als *Normalverteilung* bezeichnet und *normalverteilt* bedeutet das Gleiche wie *Gauss-verteilt*. Falls Sie Herrn Gauss einmal persönlich begegnen sollten (sehr unwahrscheinlich, er lebte von 1777 bis 1855), sollten Sie ihn aber nicht mit "Herr Normal" anreden!

Anmerkung-2: Bei der Angabe von *Bereichen* (oder: *Intervallen*) bedeutet
eine eckige Klammer *einschließlich* und
eine runde Klammer *ausschließlich.*
Somit ist z.B. `[0.0, 1.0)` der Bereich von 0.0 einschließlich bis 1.0 ausschließlich.

2. Die Klassenmethoden random() in den Klassen Math und StrictMath

Die Klassenmethoden `double random()` in
den Klassen `java.lang.Math` und `java.lang.StrictMath` liefern
nicht-reproduzierbare, gleichverteilte `double`-Werte aus dem
Bereich `[0.0, 1.0)`.

Aufgabe-01:
Kann ein Methodenaufruf wie `Math.random()` den Wert 1.0 liefern?
Kann ein Methodenaufruf wie `StrictMath.random()` den Wert 0.0 liefern?

3. Objekte der Klasse java.util.Random

Beispiel-01: Ein Random-Objekt für *reproduzierbare* Zufallszahlen:

```
Random rainer = new Random(123L);
```

Den Parameter des Random-Konstruktors (im Beispiel: 123L) bezeichnet man auch als *Keim* des Zufallszahlen-Generators rainer (engl.: seed of the random number generator rainer). Von diesem Keim hängen die Zufallswerte ab, die von den Methoden in rainer geliefert werden:
Gleicher Keim, gleiche Werte. Ungleiche Keime, ungleiche Werte.

Anmerkung: Von dem als Parameter angegebenen Keim werden (nicht alle 64 Bit, sondern) nur die ers - ten 48 Bit verwendet. Deshalb sollte man als Keim höchstens die Zahl 281.474.976.710.655 angeben (etwa 281 Billionen).

Beispiel-02: Ein Random-Objekt für *nicht-reproduzierbare* Zufallszahlen:

```
Random niko = new Random();
```

Der Keim des Generators niko wird mit Hilfe der Methode System.nanoTime() ermittelt.

4. Methoden eines Random-Objekts

In den folgenden Beispielen könnte das Random-Objekt niko auch durch das Random-Objekt rainer ersetzt werden.

Beispiel-03: Variablen werden mit Zufallswerten initialisiert

```
1 int    ilse   = niko.nextInt();
2 int    inge   = niko.nextInt(100);
3 int    ingo   = niko.nextInt(100) + 200;
4 int    irma   = niko.nextInt(101) - 50;
5 float  felix  = niko.nextFloat();
6 double dora   = niko.nextDouble();
7 double dieter = niko.nextGaussian();
```

1. ilse wird mit einem gleichverteilten Wert aus dem gesamten Bereich des Typs int: [-2_147_483_648, + 2_147_483_647] initialisiert.
In der Hälfte aller Fälle wird das ein *negativer Wert* sein!

2. inge wird mit einem Wert aus dem Bereich [0...100) gleich [0...99] initialisiert.

3. ingo wird mit einem Wert aus dem Bereich [200...300) gleich [200...299] initialisiert.

4. irma wird mit einem Wert aus dem Bereich [-50...+51) gleich [-50...+50] initialisiert.

5. felix wird mit einem gleichverteilten float-Wert aus dem Bereich [0.0F...1.0F) initialisiert.

6. dora wird mit einem gleichverteilten double-Wert aus dem Bereich [0.0...1.0) initialisiert.

7. dieter wird mit einem normalverteilten double-Wert initialisiert. Die Normalverteilung hat einen Mittelwert m von 0.0 und eine Standardabweichung s von 1.0. Das bedeutet:

Alle double-Werte haben eine Chance, von der Funktion newGaussian geliefert zu werden, aber diese Chancen sind sehr ungleich verteilt: Je näher ein double-Wert dem Mittelwert 0.0 ist, desto größer sind seine Chancen. Zwei Werte, die gleich weit vom Mittelwert entfernt sind (hier z.B. -2.73 und +2.73) haben gleich große Chancen. Von 400 Ergebnissen der Funktion newGaussian() werden im Durchschnitt etwa 399 im Bereich [-3*s, +3*s] gleich [-3.0, + 3.0] liegen und nur 1 Ergebnis ausser - halb. Werte außerhalb des Bereichs [-5*s, +5*s] haben kaum noch eine Chance, geliefert zu werden.

Das Programm Zufall04.java im Archiv BspJava.zip (auf der Netzseite zum Buch "Java ist eine Sprache") demonstriert die Methode newGaussian und die Wirkung einer Gauss-Verteilung.