TECHNISCHE HOCHSCHULE DEGGENDORF

Java Programmierung

Zeichen, Bits und große Zahlen

Die Klasse Math und Zufallszahlen

Prof. Dr. rer. nat. Andreas Berl

Fakultät für Angewandte Informatik

Technische Hochschule Deggendorf

Lernziele

- Die Klasse Math
  - Math und StrictMath
  - Mathematische Konstanten und Methoden
  - Überlaufkontrolle und die Genauigkeit von Gleitkommatypen
- Zufallszahlen
  - Zufallszahlengeneratoren
  - Klassen zum Erzeugen von Zufallszahlen
  - Die Klasse Random

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math

java.lang.Math

- Bietet grundlegende numerische Berechnungen
- Typische Utility-Klasse (**Klassenmethoden** und **Konstanten**)
- Aufrufe erfolgen über den Klassennamen

```
double d = Math.PI / 2;
```

java.lang.StrictMath

- Bietet viele Methoden die auch in **`Math`** zu finden sind
- Hält sich bei Berechnungen strikt an den **IEEE-Standard**
- **`Math`** darf im Gegensatz zu **`StrictMath`** plattformspezifische Optimierungen nutzen (z.B. mit Hilfe von „Intrinsics“)
  - **`Math`** ist ggf. performanter als **`StrictMath`**
  - **`Math`** liefert ggf. andere Ergebnisse als **`StrictMath`**

Inhalte von Math

Mathematische Konstanten
- Die Kreiszahl pi
- Die eulersche Zahl e

Mathematische Funktionen
- Absolutwerte und Vorzeichen
- Runden von Werten
- Division und Restwert mit Abrunden
- Maximum/Minimum
- Wurzel, Exponentialfunktion und Logarithmus
- Winkelfunktionen

Berechnungen mit Überlaufkontrolle

Abstände zwischen Gleitkommatypen

Die Klasse Math
- Mathematische Konstanten und Funktionen

Konstanten

- Die Kreiszahl **pi**

```
static final double PI = 3.14159265358979323846;
```

- Die eulersche Zahl **e**

```
static final double E = 2.7182818284590452354;
```

Absolutwerte und Vorzeichen

- Absolute Werte ohne Vorzeichen

```
static int abs(int i)
static long abs(long l)
static float abs(float f)
static double abs(double d)
```

Beispiel: Konstanten und Absolutwerte

```
System.out.println(Math.PI)
System.out.println(Math.E);
System.out.println(Math.abs(-3000));
```

Ausgabe:

```
3.141592653589793
2.718281828459045
3000
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Mathematische Konstanten und Funktionen

Vorzeichen

- Kopieren des Vorzeichens aus `sign` auf den Wert `magnitude`

```
static double copySign(double magnitude, double sign)
static float copySign(float magnitude, float sign)
```

- Feststellen des Vorzeichens (Ergebnis ist **`1.0`** oder **`-1.0`**)

```
static float signum(float f)
static double signum(double d)
```

  - Die Signums-Methode für **`int`** und **`long`** findet sich in den jeweiligen Wrapper-Typen
    - Beispiel: Methode der Klasse **`Integer`**

```
static int signum(int i)
```

Beispiel: Vorzeichen

```
System.out.println(Math.copySign(500, -1))
System.out.println(Math.signum(-333.33));
System.out.println(Integer.signum(5));
```

Ausgabe:

```
-500.0
-1.0
1
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Mathematische Konstanten und Funktionen

Runden von Werten

- Auf- und Abrunden auf die nächste Ganzzahl

```
static double ceil(double d)
static double floor(double d)
```

- Kaufmännisches Runden

```
static int round(float f)
static long round(double d)
```

- Gerechtes Runden zur nächsten Ganzzahl

```
static double rint(double d)
```

Beispiel: Runden

```
System.out.println(Math.ceil(2.5));
System.out.println(Math.floor(2.5));
System.out.println(Math.round(2.5));
System.out.println(Math.rint(2.5));
```

Ausgabe:

```
3.0
2.0
3
2.0
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Mathematische Konstanten und Funktionen

Maximum/Minimum

- Maximum und Minimum von zwei Zahlen lassen sich für **`int`**, **`long`**, **`float`** und **`double`** bestimmen
  - Beispiel: Methoden für den Typ **`double`**

```
static double max(double x, double y)
static double min(double x, double y)
```

Ganzzahldivision und Restwert mit Abrunden

- Division mit Abrunden

```
static int floorDiv(int x, int y)
static long floorDiv(long x, long y)
static long floorDiv(long x, int y)
```

- Restwert mit Abrunden
  - Entspricht: **`x - (floorDiv(x, y) * y)`**, mit dem Vorzeichen von **`y`**

```
static int floorMod(int x, int y)
static long floorMod(long x, long y)
static int floorMod(long x, int y)
```

Beispiel: Minimum, Maximum, Division, Restwert

```
System.out.println(Math.min(2.0, 5.0));
System.out.println(Math.max(2.0, 5.0));
System.out.println(Math.floorDiv(11, 2));
System.out.println(Math.floorMod(11, 2));
```

Ausgabe:

```
2.0
5.0
5
1
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Mathematische Konstanten und Funktionen

Wurzel, Exponentialfunktion und Logarithmus

- Quadratwurzel $\sqrt{x}$ und Kubikwurzel $\sqrt[3]{x}$

```
static double sqrt(double x)
static double cbrt(double x)
```

- Hypothenuse $\sqrt{x^2 + y^2}$

```
static double hypot(double x, double y)
```

- Exponentialwert e^x und x^y

```
static double exp(double x)
static double pow(double x, double y)
```

- Logarithmus $\log_e(x)$ und $\log_{10}(x)$

```
static double log(double x)
static double log10(double x)
```

Beispiel: Wurzeln, Hypothenuse, Exponentialwerte, Logarithmus

```
System.out.println(Math.sqrt(16));
System.out.println(Math.cbrt(27));
System.out.println(Math.hypot(4, 16));
System.out.println(Math.exp(3));
System.out.println(Math.pow(10, 3));
System.out.println(Math.log10(1000));
```

Ausgabe:

```
4.0
3.0
16.492422502470642
20.085536923187668
1000.0
3.0
```

Die Klasse Math
- Mathematische Konstanten und Funktionen

Winkelfunktionen

- Sinus, Cosinus, Tangens
  - **Achtung**: Winkel werden im Bogenmaß übergeben, nicht im Gradmaß!

```
static double sin(double radians)
static double cos(double radians)
static double tan(double radians)
```

- Arcus-Sinus, Arcus-Cosinus, Arcus-Tangens
  - **Achtung**: Ergebnis ist ein Winkel im Bogenmaß

```
static double asin(double x)
static double acos(double x)
static double atan(double x)
```

- Umrechnen zwischen Bogenmaß und Gradmaß

```
static double toRadians(double degrees)
static double toDegrees(double radians)
```

Beispiel: Winkelfunktionen

```
System.out.println(Math.sin(Math.PI / 2));
System.out.println(Math.cos(2 * Math.PI));
System.out.println(Math.asin(1));
System.out.println(Math.acos(1));
System.out.println(Math.toRadians(180));
System.out.println(Math.toDegrees(Math.PI));
```

Ausgabe:

```
1.0
1.0
1.5707963267948966
0.0
3.141592653589793
180.0
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Überlaufkontrolle

Überlauf

- Der Wertebereich für primitive Datentypen ist durch die Anzahl der Bits im Arbeitsspeicher begrenzt
- Bei Rechenoperationen kann ein Wert aus diesem Wertebereich herauslaufen
  - Überläufe werden in Java **nicht** automatisch geprüft!
  - Weder der Compiler noch die Laufzeitumgebung bieten eine Überlaufkontrolle

Beispiel: Überlauf wird nicht geprüft

```
int i = 2147483647;
int mult = i * i;
System.out.println(mult); // → 1
```

`i * i` ist zu groß und kann nicht mehr in einem `int` gespeichert werden.

Bei Überläufen kommt es zu falschen Ergebnissen. Der Fehler wird **nicht** angezeigt.

Wertebereiche von Zahlentypen

Name	RAM	Wertebereich
`byte`	8 Bit	-128 ... +127
`short`	16 Bit	-32768 ... +32767
`int`	32 Bit	-2147483648 ... +2147483647
`long`	64 Bit	-9223372036854775808 ... +9223372036854775807
`float`	32 Bit	$\pm 1{,}4^{-45}$ ... $\pm 3{,}4028235^{38}$
`double`	64 Bit	$\pm 4{,}9^{-324}$ … $\pm 1{,}79769313486231^{308}$

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Überlaufkontrolle

Verhindern von Überläufen

- Die Anpassung des Ergebnistyps auf den größeren Datentyp **`long`** reicht nicht aus!
  - Bei der Multiplikation zweier **`int`** ist das Ergebnis wieder ein **`int`**
- Zusätzlich muss einer der beiden Operanden der Berechnung auf den Datentyp **`long`** gecasted werden

Überlauf und Stolperfallen

- Ganzzahlige Literale werden in Java immer als **`int`** interpretiert
- Auch in Zwischenergebnissen kann ein Überlauf stattfinden

Beispiel: Überlauf in einer Berechnung

```
int i = 2147483647;
long mult = i * i;
System.out.println(mult); // → 1
```

`int * int` wird zu einem `int`.

Beispiel: Überlauf verhindern

```
int i = 2147483647;
long mult = (long) i * i;
System.out.println(mult); // → 4611686014132420609
```

Richtiges Ergebnis.

Beispiel: Ganzzahlige Literale sind vom Typ `int`

```
long result = 2147483647 * 2147483647;
System.out.println(result); // → 1
```

`int * int`.

Beispiel: Überlauf im Zwischenergebnis

```
int result = 100000 * 100000 / 100000;
System.out.println(result); // → 14100
```

Zwischener-gebnis zu groß!

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Überlaufkontrolle

Methoden zur Erkennung von Überläufen

- Bei einigen Fragestellungen muss man Überläufe erkennen können
- Die Klasse **`Math`** bietet spezielle Methoden an
  - Für Operationen, bei denen es zu einem Überlauf kommen kann
  - Falls ein Überlauf stattfindet wird zur Laufzeit eine **`ArithmeticException`** ausgelöst

Beispiel: Überlauf bei Multiplikation

```
int i = 2147483647;
int mult = Math.multiplyExact(i, i);
System.out.println(mult);
```

Eine **`ArithmeticException`** wird ausgelöst.

- Rechenoperationen und Negation

```
static int addExact(int x, int y)
static long addExact(long x, long y)
static int subtractExact(int x, int y)
static long subtractExact(long x, long y)
static int multiplyExact(int x, int y)
static long multiplyExact(long x, long y)
static int negateExact(int a)
static long negateExact(long a)
```

- Inkrement und Dekrement

```
static int incrementExact(int a)
static long incrementExact(long a)
static int decrementExact(int a)
static long decrementExact(long a)
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Überlaufkontrolle

Überlauf beim Typecast

- Java konvertiert Datentypen automatisch vom „niederwertigen“ zum „höherwertigen“ Typ
- Konvertierungen in die andere Richtung erfordern einen Typecast

Beispiel: Konvertierung von `long` nach `int` ist nur mit Typecast möglich

```
long l = 100L;
int i = l;
```

Kompiliert nicht.

```
long l = 100L;
int i = (int) l;
```

Kompiliert.

- **Aber**: Beim Typecast findet **keine** Prüfung des Überlaufs statt

Beispiel: Überlauf beim Typecast

```
long l = 10000000000000L;
int i = (int) l;
```

Auch dieser Cast kompiliert → Überlauf ohne Fehlermeldung!

- Methode in **`Math`** zur Überlauferkennung beim Typecast von **`long`** zu **`int`**
  - Falls ein Überlauf stattfindet wird zur Laufzeit eine **`ArithmeticException`** ausgelöst

```
static int toIntExact(long value)
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Genauigkeit von Gleitkommatypen

Grenzen von primitiven Gleitkommatypen

- Die Typen **`float`** und **`double`** sind sehr ungenau
  - Es kommt leicht zu Rundungsfehlern
  - Weder **`float`** noch **`double`** sollten für Währungen verwendet werden!

Beispiel: Rundungsfehler

```
/* Ausgabe von 0.02 mit 20 Nachkommastellen */
System.out.printf("%.20f%n", 0.01f + 0.01f); // 0.02f -> 0,01999999955296516400
System.out.printf("%.20f%n", 0.03d - 0.01d); // 0.02d -> 0,01999999999999999700
```

Abstände zwischen Gleitkommazahlen

- Der Abstand von einer Gleitkommazahl zur Nächsten ist **nicht** immer gleich
- Methoden der Klasse **`Math`** erlauben das arbeiten mit genauen Abständen
  - Abstand zur nächsten Gleitkommazahl berechnen
  - Nächsthöhere / nächstniedrigere Gleitkommazahl finden

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Die Klasse Math
- Genauigkeit von Gleitkommazahlen

Unit in the last place (ulp)

- Abstand zur nächsten Gleitkommazahl

```
static float ulp(float f)
static double ulp(double d)
```

  - Typische Operationen (+, -, *, /) haben ½ ulp
  - Methoden in **`Math`**: ulp siehe Javadoc

Beispiele: Abstände zur nächsten Gleitkommazahl

Methodenaufruf	Rückgabewert
`Math.ulp(-100000.0)`	`0,00000000001455191522836685 2`
`Math.ulp(-100.0)`	`0,00000000000001421085471520 2004`
`Math.ulp(-1.0)`	`0,00000000000000022204460492503 13`
`Math.ulp(1.0)`	`0,00000000000000022204460492503 13`
`Math.ulp(100.0)`	`0,00000000000001421085471520 2004`
`Math.ulp(100000.0)`	`0,00000000001455191522836685 2`

Nächste Gleitkommazahl finden

- Nächste Gleitkommazahl (in Richtung `direction`)

```
static float nextAfter(float start, float direction)
static double nextAfter(double start, double direction)
```

- Nächste Gleitkommazahl (größer/kleiner)

```
static float nextUp(float f)
static double nextUp(double d)
static float nextDown(float f)
static double nextDown(double d)
```

Beispiel: Zusammenhang ulp und nextUp(…)

```
double d = Math.nextUp(1.0) - Math.ulp(1.0);
System.out.println(d); // -> 1.0
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen

Zufallszahlengeneratoren

- Verfahren zur Erzeugung von **Zufallszahlen** bzw. **Folgen von Zufallszahlen**
- **Echte Zufallszahlen**
  - Erzeugung mithilfe physikalischer Phänomene
    - Münzwurf, Würfel, Roulette, Rauschen elektronischer Bauelemente, radioaktive Zerfallsprozesse oder quantenphysikalische Effekte
    - Nachteil: Zeitlich und technisch aufwändig
- **Pseudozufallszahlen** (werden in Java verwendet)
  - Scheinbar zufällige Zahlen → lassen sich vorhersagen
  - Werden einem festen, reproduzierbaren Verfahren erzeugt
  - Haben ähnliche statistische Eigenschaften wie echte Zufallszahlenfolgen
    - Gleichmäßige Häufigkeitsverteilung
    - Geringe Korrelation

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Erzeugen von Zufallszahlen

Klassen zur Erzeugung von Zufallszahlen

`java.lang.Math`

Erzeugt Zufallszahlen zwischen 0 und 1

`java.math.BigInteger`

Konstruktoren für die Erzeugung von großen Zufallszahlen

`java.util.Random`

Standardklasse zur Erzeugung von Zufallszahlen

`java.security.SecureRandom`

Bessere Zufallszahlen, die auch kryptographisch genutzt werden können

`java.util.SplittableRandom`

Erzeugt Folgen von Zufallszahlen die den sogenannten „dieharder-Test“ bestehen

`java.util.concurrent.ThreadLocalRandom`

Generieren von Zufallszahlen bei nebenläufiger Programmierung

Diese drei Klassen werden im Folgenden näher betrachtet.

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Die Klassen Math und BigInteger

java.lang.Math

- Die Methode **`random()`**
  - Erzeugt gleichverteilte Zufallszahlen im Intervall `[0, 1[`
  - Nutzt die Klasse **`java.util.Random`** um die Zufallszahlen zu erstellen

```
static double random() // Zufallszahl z, mit 0.0 ≤ z < 1.0
```

java.math.BigInteger

- Es gibt zwei Konstruktoren um große Zufallszahlen zu erzeugen
  - Erzeugen von gleichverteilten Zufallszahlen im Intervall `[0, ` $2^{numBits}$ `- 1]`

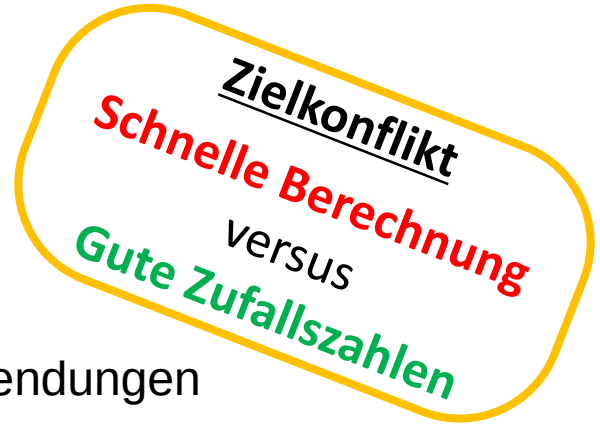
```
public BigInteger(int numBits, Random random) // Zufallszahl z, mit 0.0 ≤ z ≤ 2^numBits - 1
```

  - Erzeugen von zufällige **Primzahlen** mit der angegebenen Wahrscheinlichkeit in der gegebenen Bitlänge

```
public BigInteger(int bitLenght, int certainty, Random random) // Siehe Javadoc
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Die Klasse Random



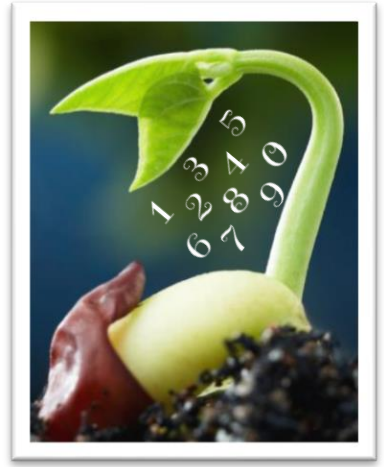
java.util.Random

- Standardklasse zur Erzeugung von Zufallszahlen
  - Random bietet **schnelle Berechnung**
  - **Aber:** die Zufallszahlen sind **nicht ausreichend gut** für kryptographische Anwendungen

→ **`java.security.SecureRandom`**

Der Seed

- Jedes **`Random`**-Objekt benötigt einen sogenannten „Seed“ (Samen)
  - **Seed**: 48-Bit Startwert für die Berechnung von Folgen von Zufallszahlen
  - **Reproduzierbarkeit**: Gleicher Seed ⇒ gleiche Folge von Zufallszahlen
- Erzeugen eines **`Random`**-Objekts ohne explizite Angabe eines Seed
  - Die aktuelle Systemzeit wird als Seed verwendet: **`System.nanoTime()`**



Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Die Klasse Random

Erzeugen der Klasse Random

- Konstruktoren

```
public Random()                 // Seed wird implizit auf die Systemzeit gesetzt.
public Random(long seed)        // Seed wird explizit gesetzt (reproduzierbare Zufallszahlen).
```

- Seed neu setzen

```
public void setSeed(long seed) // Seed wird neu gesetzt. Verhält sich wie ein neues Random-Objekt.
```

Erzeugen von Zufallszahlen

- Erzeugen der jeweils nächsten Zufallszahl einer Folge (gleichverteilt)

```
public int nextInt()           // Nächste Zufallszahl im gesamten Wertebereich (auch negativ).
public int nextInt(int bound) // Nächste Zufallszahl im Intervall [0, bound[.
public long nextLong()         // Nächste Zufallszahl im gesamten Wertebereich (auch negativ).
public float nextFloat()       // Nächste Zufallszahl im Intervall [0.0, 1.0[.
public double nextDouble()     // Nächste Zufallszahl im Intervall [0.0, 1.0[.
public double nextBoolean ()   // Nächster zufälliger Boolean (true oder false).
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Die Klasse Random

Zufallszahlen mit Normalverteilung

- Erzeugen der jeweils nächsten Zufallszahl einer **normalverteilten** Folge von Zufallszahlen
  - Mittelpunkt 0 und Standardabweichung 1

```
public double nextGaussian()  // Nächste Zufallszahl der normalverteilten Folge.
```

Folgen von Zufallszahlen

- Erzeugen von Zufallszahlen in einer Schleife
- Erzeugen eines Arrays mit zufälligen Bytes

```
public byte[] nextBytes(byte[] bytes)  // Füllt das Array mit zufälligen Bytes.
```

- Erzeugen eines **Stream** von Zufallszahlen (für die Datentypen **`int`**, **`long`** oder **`double`**)

```
public IntStream ints() // Unendlicher Strom im gesamten Wertebereich.
public IntStream ints(int start, int bound) // Unendlicher Strom im Intervall [start, bound[.
public IntStream ints(long size) // Strom der Länge size im gesamten Wertebereich.
public IntStream ints(long size, int start, int bound) // Strom der Länge size im Intervall
```

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zufallszahlen
- Die Klasse Random

Beispiel: Anwendung der Klasse `Random`

```
static void main(String[] args) {
    int seed = 1234;
    Random random = new Random(seed); // Initialisierung mit Seed mit 1234

    int bound = 50;
    System.out.println(random.nextInt(bound)); // = 28
    System.out.println(random.nextInt(bound)); // = 33
    System.out.println(random.nextDouble());   // = 0.9513577109193919

    random.setSeed(seed); // Zurücksetzen des Seed auf 1234
    System.out.println(random.nextInt(bound)); // = 28
    System.out.println(random.nextInt(bound)); // = 33
    System.out.println(random.nextDouble());   // = 0.9513577109193919

    long size = 3;
    int start = 50;
    int bound = 100;
    random.ints(size, start, bound).forEach(System.out::println); // 60 93 97
}
```

Nach dem Rücksetzen des Seed wiederholen sich die Zufallszahlen.

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de

Zusammenfassung

- Die Klasse Math
  - Math und StrictMath
  - Mathematische Konstanten und Methoden
  - Überlaufkontrolle und die Genauigkeit von Gleitkommatypen
- Zufallszahlen
  - Zufallszahlengeneratoren
  - Klassen zum Erzeugen von Zufallszahlen
  - Die Klasse Random

Prof. Dr. rer. nat. Andreas Berl – www.andreas-berl.de