Übungen 3

# Aufgabe 1

## Teilaufgabe A

0.03125 \* 2 = 0.0625 | 0

0.0625 \* 2 = 0.125 | 0

0.125 \* 2 = 0.25 | 0

0.25 \* 2 = 0.5 | 0

0.5 \* 2 = 1 | 1 = 0.00001

V = 1

Exponent = -5, also 127 + -5 = 122 = 0111'1010

Mantisse = <000>

## Teilaufgabe B

66048 / 2 = 33024 | 0

33024 / 2 = 16512 | 0

16512 / 2 = 8256 | 0

8256 / 2 = 4128 | 0

4128 / 2 = 2064 | 0

2064 / 2 = 1032 | 0

1032 / 2 = 516 | 0

516 / 2 = 258 | 0

258 / 2 = 129 | 0

129 / 2 = 64 | 1

64 / 2 = 32 | 0

32 / 2 = 16 | 0

16 / 2 = 8 | 0

8 / 2 = 4 | 0

4 / 2 = 2 | 0

2 / 2 = 1 | 0

1 / 2 = 0 | 1 = 1'0000'0010'0000'0000

V = 0

Exponent = 16, also 127 + 16 = 143 = 1000'1111

Mantisse = 0000001<000>

# Aufgabe 2

## Teilaufgabe A

V = 1

Exponent = 1001’0110 = 150, also 23

Mantisse = 2^-1 + 2^-3 + 2^-4 = 0.6875

Zahl = -1 \* 1.6875 \* 2^23 = -14155776

## Teilaufgabe B

V = 0

Exponent = 0111’1101 = 125, also -2

Mantisse = 2^-2 + 2^-3 + 2^-5 = 0.40625

Zahl = 1 \* 1.40625 \* 2^-2 = 0.3515625

# Aufgabe 3

## Teilaufgabe A

* Unendlich \* negativer Wert = Negativ unendlich

## Teilaufgabe B

* Double.MAX\_VALUE, da 1 zu klein ist (In die Ungenauigkeit fällt), um addiert zu werden und den Wert unendlich werden zu lassen

## Teilaufgabe C

* Die zu subtrahierende Zahl ist am Anfang noch sehr klein, man subtrahiert sie aber gleich von einer sehr grossen Zahl – die Ungenauigkeit tritt hier ein und gewisse Werte werden gar nicht wirklich subtrahiert
* Man müsste also zuerst die grössten Werte subtrahieren und dann immer kleiner werden, da mit kleineren Zahlen auch die Breite der Ungenauigkeit abnimmt

# Aufgabe 4

|  |
| --- |
| package ch.swaechter.fhnw.algd1.substraction;  public class Substraction {  public static void main(String[] args) {  substractFalse();  substractTrue();  }  public static void substractFalse() {  int n = 80;  double x = 1, s = Math.pow(2, n + 1);  while (n >= 0) {  s = s - x;  x = x \* 2;  n--;  }  System.out.println("s: " + s + " " + (s == 1.0) + ", " + (s <= 2.0));  }  public static void substractTrue() {  int n = 80;  double x = Math.pow(2, n), s = Math.pow(2, n + 1);  while (n >= 0) {  s = s - x;  x = x / 2;  n--;  }  System.out.println("s: " + s + " " + (s == 1.0) + ", " + (s <= 2.0));  }  } |

# Aufgabe 5

|  |
| --- |
| public static String toString(int value) {  if (value < 0) {  value += 256;  }  StringBuilder builder = new StringBuilder();  for (int i = 0; i < 8; i++) {  builder.insert(0, (char) ('0' + (value % 2)));  value /= 2;  }  return builder.toString();  } |

# Aufgabe 6

## Teilaufgabe A

UTF-8 hexadezimal: E2 85 98 C3 97 34

UTF-8 binär: 1110'0010 \ 1000'0101 \ 1001'1000 \ 1100'0011 \ 1001'0111 \ 0011'0100

1. Zeichen à 3 Bytes in UTF-8

UTF-8 hexadezimal: E2 85 98

UTF-8 binär: 1110'0010 \ 1000'0101 \ 1001'1000

UTF-8 Bytes: 3

Unicode binär: 0010'0001 0101'1000

Unicode Bits: 14

Unicode hexadezimal: 21 58

Unicode Zeichen: 4/5 (Bruch)

2. Zeichen à 2 Bytes in UTF-8

UTF-8 hexadezimal: C3 97

UTF-8 binär: 1100'0011 1001'0111

UTF-8 Bytes: 2

Unicode binär: 0000'1101'0111

Unicode Bits: 11

Unicode hexadezimal 0D7

Unicode Zeichen: x (Multiplication)

3. Zeichen à 1 Bytes in UTF-8

UTF-8 hexadezimal: 34

UTF-8 binär: 0011'0100

UTF-8 Bytes: 1

Unicode binär: 0011'0100

Unicode Bits: 6

Unicode hexadezimal: 34

Unicode Zeichen: 4 (Zahl)

## Teilaufgabe B

* Zeichen:
  + Unicode Zeichen: A
  + Unicode hexadezimal: 41
  + Unicode binär: 0100’0001
  + Anzahl Bits: 7
  + UTF-8 Bytes: 1
  + UTF-8 binär: 0100’0001
  + UTF-8 hexadezimal: 41
  + UTF-8 dezimal: 65
* Zeichen: Ā
  + Unicode Zeichen: Ā
  + Unicode hexadezimal: 100
  + Unicode binär: 0001’0000’0000
  + Anzahl Bits: 9
  + UTF-8 Bytes: 2
  + UTF-8 binär: 1100’0100 / 1000’0000
  + UTF-8 hexadezimal: C4 80
  + UTF-8 dezimal: 196 128
* Zeichen: 伶
  + Unicode Zeichen: 伶
  + Unicode hexadezimal: 4F36
  + Unicode binär: 0100’1111’0011’0110
  + Anzahl Bits: 15
  + UTF-8 Bytes: 3
  + UTF-8 binär: 1110’0100 / 1011’1100 / 1011’0110
  + UTF-8 hexadezimal: E4 BC B6
  + UTF-8 dezimal: 228 188 182

# Aufgabe 7

Noch zu erledigen