Chapter 4

Number Representation

Under the covers of numbers in Java

How (Unsigned) Integers Work

Base 10 – Decimal (People)

…	10^2	10^1	10^0
	2	3	4

$$234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

Base 2 – Binary (Computer)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

$$234 = \begin{array}{c} 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + \\ 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + \\ 1 \times 2^1 + 0 \times 2^0 \end{array}$$

Signed (Two's Complement) Numbers

- If left-most bit is 1, ***interpret*** bits as unsigned, but subtract 2^N

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

$$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 234$$

$$234 - 2^8 = 234 - 256 = -22$$

- N is the Number of bits (8 in the example)
- Allows us to specify any integer, x: $-2^{(N-1)} <= x <= (2^{(N-1)} - 1)$

Integer Division and Truncation

- Integer division (byte, short, int, long) discards remainders:
- 23 / 4 is 5, but 4*5 is 20! (23 % 4 is 3).

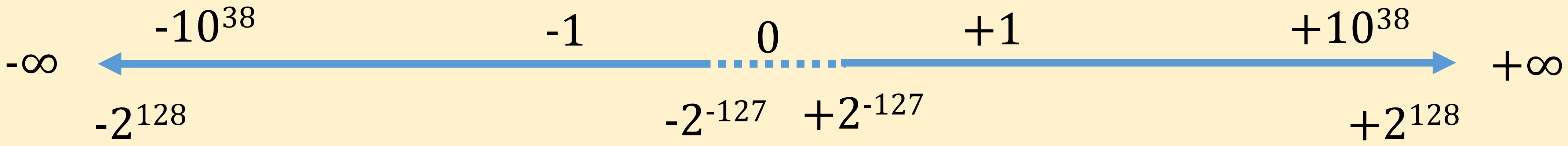
```
System.out.println(4000000000L / 1234567); // 3240
System.out.println(3240 * 1234567L); // 3999997080
System.out.println(4000000000L % 1234567); // 2920
System.out.println(4000000000L / 1234567.0); // 3240.0023652017267
```

IEEE Floating Point Standard (32 bit)

- First normalize the number to the form:

$$value = -1^{S} \times SIG \times 2^{exp}$$

- S = 0 (positive) or 1 (negative)
- $1 \leq SIG < 2$ (expressed in 24 bit precision)
- $-127 \leq \exp \leq 127$



IEEE 754 – 32 bit float

- Value Representation:
  - Decimal: [+/-]<digit>.<fraction> x $10^{\text{<exponent>}}$ e.g. 6.022 x 10^{23}
  - Binary: [+/-]1.<fraction> x $2^{\text{<exponent>}}$ e.g 1.11111110000101… x 2^{78}
  - Special case for 0, +/- ∞ (INFINITY), "Not a Number" (NAN)
- Bit Representation (float)

S	EXP								FRAC																						
b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Floating Point Approximation

- 4.35 has an infinite binary expansion that is truncated
  01000000100010110011001100110011 (float)
  0100000000010001011001100110011001100110011001100110011001100110 (double)
- 4.3499999999999993 through 4.35 all give the same double

- Weird effects of approximation:
  4.35F*100 prints as 435.0
  4.35*100 prints as 434.99999999999994
  4.05F*100 prints as 405.00003

Truncation and Rounding

```
System.out.println(4.35*100);
```
- 434.99999999999994

```
System.out.println((int)4.35*100);
```
- 400

```
System.out.println((int)(4.35*100));
```
- 434

```
System.out.println(Math.round(4.35*100));
```
- 435 (Note… this is of type “long”)

Range v. Precision v. Space

Type	Range	Precision	Space
boolean	true/false	Exact	8 bits
byte	+/- 127	Exact	8 bits
short	+/- ~32K	Exact	16 bits
int	+/- ~2M	Exact	32 bits
long	+/- ~ 10^{18}	Exact	64 bits
float	+/- ~ 10^{38}	~15 digits	32 bits
double	+/- ~10^{308}	~23 digits	64 bits

Declaring Constants

- When working with numbers in programs it is of huge benefit to give names to constants
- By introducing named constants, code becomes more transparent to readers and if a change is needed, the change is only made in one place.
- Example:

Section 4.1.2

```
final double QUARTER_VALUE = 0.25;
```

Compiler does not allow QUARTER_VALUE to be modified.

- Convention – constants are all upper case

Constants in Library

- In Math:

```
public static final double E = 2.7182818284590452354;
public static final double PI = 3.14159265358979323846;
```

access as: `Math.E` or `Math.PI`

- In the default sRGB space

```
public final static Color yellow = new Color(255,255,0);
```

- In the default sRGB space since 1.4

```
public final static Color YELLOW = yellow;
```