Chapter 4

Constants

Some things Never Change.

Literal Values

- Primitive Data Types:
  boolean, char, byte, short, int, long, float, double
- We very often type literal values in programs:

```
int x = 17;
```

Section 4.1, Table 1

- But we need to remember, computer numbers have limits

```
int x = 6726382901; // error… number too big
```

- Oops… that number is bigger than an integer

```
long x = 6726382901; // error… number STILL too big
```

Literal Value Types

- If Java sees a number like 67263829, it assumes it is an int
  - Any literal value that starts with 1-9 and has no decimal point

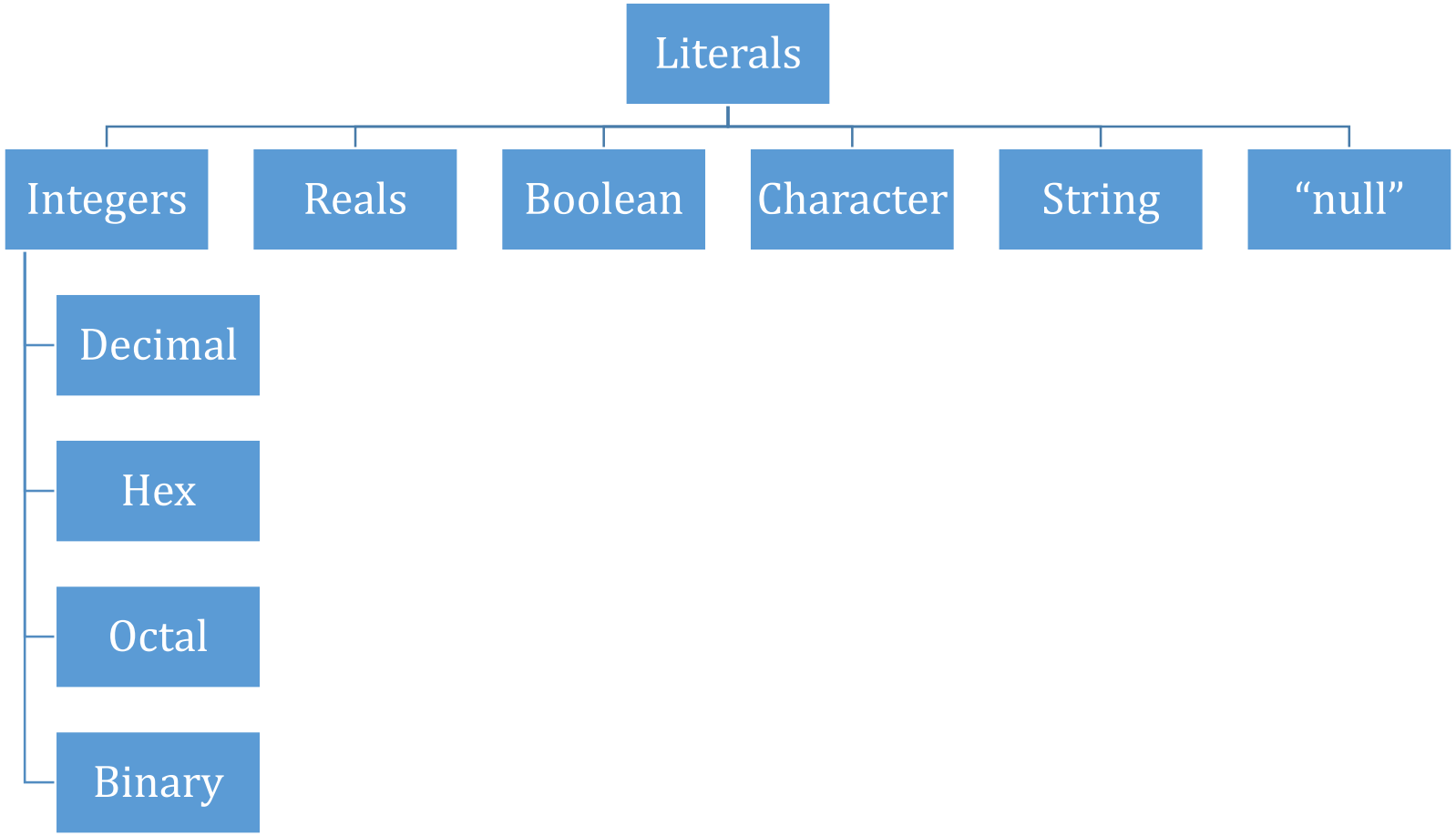
- We can tell Java treat this literal as a long by adding an "L" suffix

```
long x = 6726382901L; // but fits in 64 bits!
```

- Remember, we can still overflow long values

```
long x = 67263829012345678901L; // error – number too big
```

Literal Values



Literal Syntax

- Decimal: 0 or [1-9][0-9,_]*[L]
- Hex: 0[x,X][0-9,a-f,A-F,_]*[L]
- Octal: 0[0-7,_]*[L]
- Binary: 0[b,B][0,1,_]*[L]
- Reals: [0-9,_]*.[0-9_*][e,E][0-9,_]*[f,F,d,D] (Hex allowed too)
- Boolean: [true,false]
- Character: ‘<single character>’
- String: “[<single character>]*”

Escaped Characters

- \b – backspace
- \t – horizontal tab
- \n – line feed
- \f – form feed
- \r – carriage return
- \” – double quote
- \’ – single quote
- \\ - backslash

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

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

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

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

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

Literal Gotcha… Octal Numbers

- If a literal has a leading zero, Java treats as octal (base 8)!

```
int x = 0153; System.out.println(x);
// prints  107  = 1*8² +5*8+3
```

- Problem…

```
months[01]="January";
months[02]="February";
…
months[07]="July";
months[08]="August"; // number too big!
```

Integer Division and Truncation

- Division of *int* and *long* discards remainders:
- 23 / 4 is 5, (23 % 4 is 3).

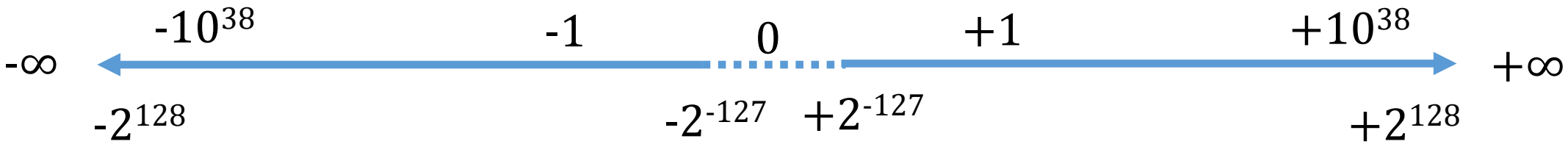
```
System.out.println(4000000000L / 1234567); // 3240
System.out.println(3240 * 1234567L); // 399997080
System.out.println(4000000000L % 1234567); // 2920
System.out.println(4000000000L / 1234567.0);
// 3240.0023652017267
```

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 $10^{<exponent>}$ e.g. 6.022 x 10^{23}
  - Binary: [+/-]1.<fraction> x $2^{<exponent>}$ 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

see: Class Web Page Examples: Number Conversion

Floating Point Literals

- If a literal contains a decimal point, Java defaults to “double” (64 bit floating point)
- Use “F” or “f” suffix to force interpretation as a “float” (32 bit)
- Note the inaccuracies of floating point:

```
System.out.println(1862500000000010000e17);
System.out.println(1862500000000001000e17);
```

- print out

```
1.8625000000000001E36
1.8625E36
```

Floating Point Approximation

- 4.35 has an infinite binary expansion that is truncated
  01000000100010**1100**11001100110011 (float)
  0100000000010001**1100**1100110011001100110011001100110011001100110 (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
float	+/- ~ 10^{38}	~15 digits	32 bits
long	+/- ~ 10^{18}	Exact	64 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 in Section 4.1.2

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

similarly DIME_VALUE, NICKEL_VALUE, PENNY_VALUE

Constants Improve Readability

```
/**
      Processes the payment received from the customer.
      @param dollars the number of dollars in the payment
      @param quarters the number of quarters in the payment
      @param dimes the number of dimes in the payment
      @param nickels the number of nickels in the payment
      @param pennies the number of pennies in the payment
   */
public void receivePayment(int dollars, int quarters,
         int dimes, int nickels, int pennies) {
         payment = dollars + quarters * QUARTER_VALUE
                  + dimes * DIME_VALUE + nickels * NICKEL_VALUE
                  + pennies * PENNY_VALUE;
}
```

Constants Isolate Changes

```
private static final double RATE1 = 0.10;
private static final double RATE2 = 0.25;
private static final double RATE1_SINGLE_LIMIT = 32000;
private static final double RATE1_MARRIED_LIMIT = 64000;
  ...
  else { // several branches similar to
        tax1 = RATE1 * RATE1_MARRIED_LIMIT;
        tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
  }
}
  return tax1 + tax2;
```

Constants in Library

- In Math:

```
public static final double E = 2.718281828459045235 4;
public static final double PI = 3.14159265358979323846;
```

- 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;
```