Primitives (Data and Operations)

- Types and Sizes
- Numeric Constants
- Casting
- Math Operations

See Also

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

https://en.wikipedia.org/wiki/Primitive_data_type

Memory

- Computer "memory" is a collection of bytes (8 bits)
- CPUs work naturally with groups of bytes (2, 4, etc.)
- In the JVM sizes are the same on every platform

```java
public class Tinker {
    public static void main(String [] args) {

        int a = 0;
        System.out.println(a); // "0"

        int b = 2147483647;
        System.out.println(b); // "2147483647"

        b = b + 1;
        System.out.println(b); // "-2147483648" What??

    }
}
```

01111111	11111111	11111111	11111111

10000000	00000000	00000000	00000000

Built-ins (Primitives)

- No “unsigned” keyword
  - (Java8 has functions for unsigned processing)
- Always the same size on every platform
- One other built-in type: pointer (later in the course)

```
int a = -25;          // 32-bit (4 byte) -2,147,483,648 to 2,147,483,637

byte b = 100;         // 1 byte -128 to 127
short c = -1;         // 2 bytes -32,768 to 32,767
long d = 0;           // 8 bytes -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float e = -1.3E-5;    // 4 bytes IEEE 754 floating point
double f = 1.22E-5;   // 8 bytes IEEE 754 floating point

char x = 'A';         // 2 bytes UNICODE
boolean y = true;     // Not numeric. Values "true" or "false".
```

Constants

```
        int a = 23456;      // Whole numbers are "int" by default
        long b = 1234L;     // Append "L" to say "this is a long"

        double d = 3.2e-2;  // Decimal numbers are "double" by default
        float f = 3.2F;     // Append "F" to say "this is a float"

        a = 'X';          // Converted to numerical value (88)
        a = '\u0108';     // Unicode sequence ('C' with circumflex)

JAVA 7  a = 1_000_000;    // Underscores in constants are ignored

        a = 010;          // Leading 0 means "octal" (value 8 here)
        a = 0xFACE;       // Leading 0x means "hex" (value 64206 here)

JAVA 7  a = 0b1101011;    // Leading 0b means "binary" (value 106 here)
```

Numeric Casts

- Assignment means "copy"

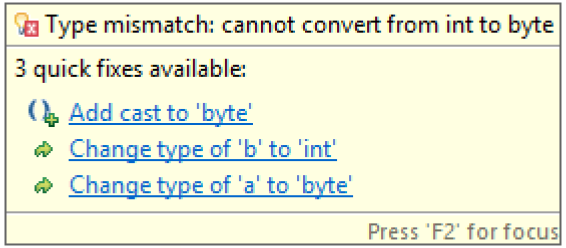
```
int a = 1340;
int b = a;
```

- Copying to larger

```
byte a = 123;
int b = a;
```

- Copying to smaller

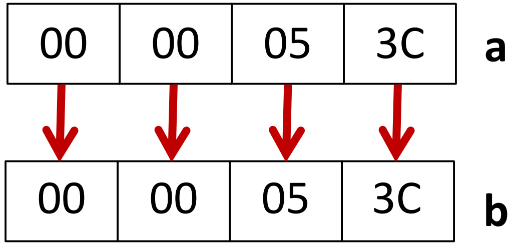
```
int a = 1340;
byte b = a;
byte b = (byte) a;
```

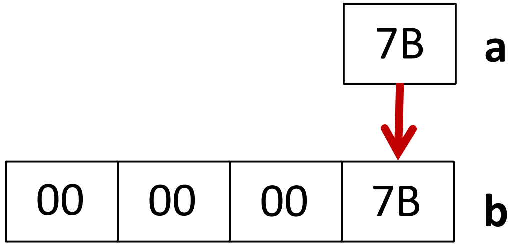

Type mismatch: cannot convert from int to byte

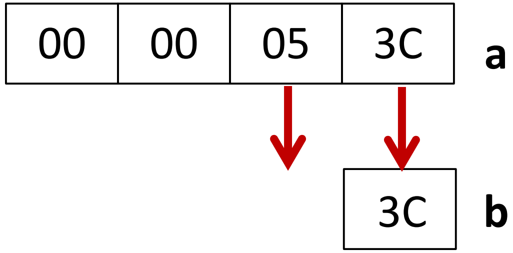
3 quick fixes available:

- Add cast to 'byte'
- Change type of 'b' to 'int'
- Change type of 'a' to 'byte'

Press 'F2' for focus





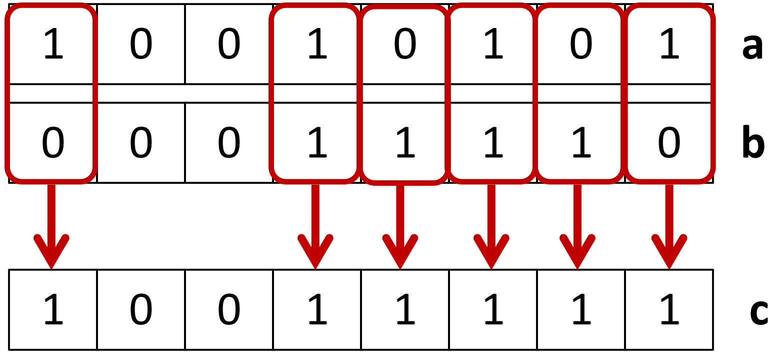


Basic Math Operations

```
int a = 2+3*2-5; // Addition, subtraction, multiplication

int b = (2+3) * (2-5); // Use parenthesis liberally

int c = 5/2;  // Integer division here. Result is 2.

int d = 9%4; // Modulo (remainder) of division. Result is 1.
```

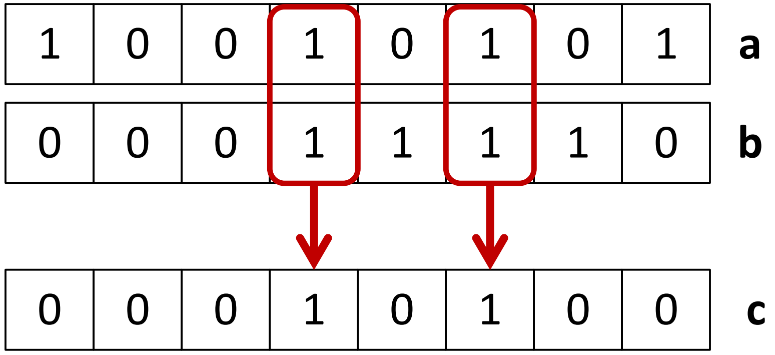
Bitwise Math

```
int a = 0b0_1001_0101; // 149
int b = 0b0_0001_1110; // 30

// Bitwise OR (result is 159 1001_1111)
int c = a | b;
```



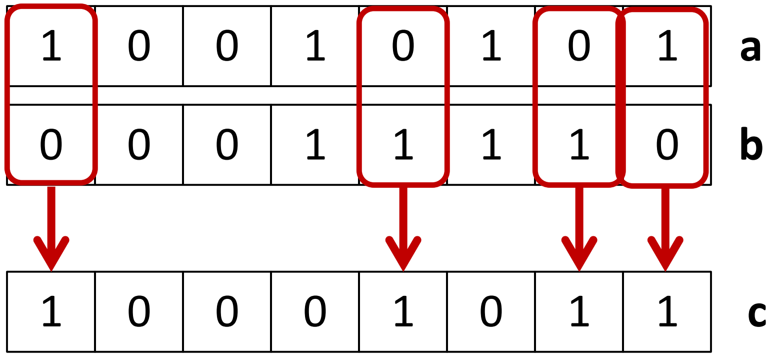
Bitwise Math

```
int a = 0b0_1001_0101; // 149
int b = 0b0_0001_1110; // 30

// Bitwise OR (result is 159 1001_1111)
int c = a | b;

// Bitwise AND (result is 20 0001_0100)
int d = a & b;
```



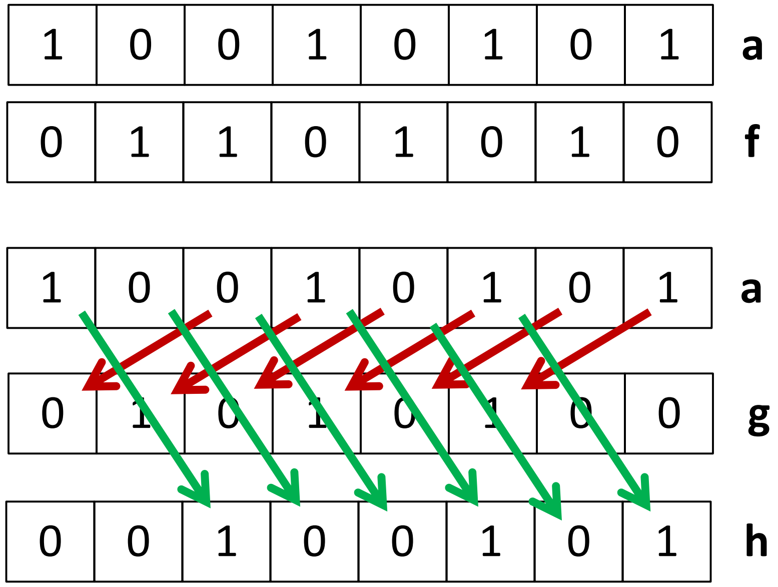
Bitwise Math

```
int a = 0b0_1001_0101; // 149
int b = 0b0_0001_1110; // 30

// Bitwise OR (result is 159 1001_1111)
int c = a | b;

// Bitwise AND (result is 20 0001_0100)
int d = a & b;

// Bitwise XOR (result is 1000_1011)
int e = a ^ b;
```



Bitwise Math

```
int a = 0b0_1001_0101; // 149
int b = 0b0_0001_1110; // 30

// Bitwise OR (result is 159 1001_1111)
int c = a | b;

// Bitwise AND (result is 20 0001_0100)
int d = a & b;

// Bitwise XOR (result is 1000_1011)
int e = a ^ b;

// Bitwise complement (result is 0110_1010)
int f = ~a;

// Shift-left 2 (result is 0101_0100)
int g = a<<2;

// Shift-right 2 (result is 1110_0101)
int h = a>>2;
```



Shortcuts

- op-equals works with any operator
- Produces the same code

```
int a = 2;

a = a + 4;

a += 4; // Means a=a+4
a /= 3; // Means a=a/3
a >>= 2; // Means a=a>>2
```

Pre and Post Increment

- The ++ adds one to the variable
- The “--” subtracts one from the variable
- Order matters if these are used in expressions

```
int a = 8;
int b = 0;

b++; // Add one to b
++b; // Add one to b

// a is 8
b = ++a;
System.out.println(a); // "9"
System.out.println(b); // "9"
```

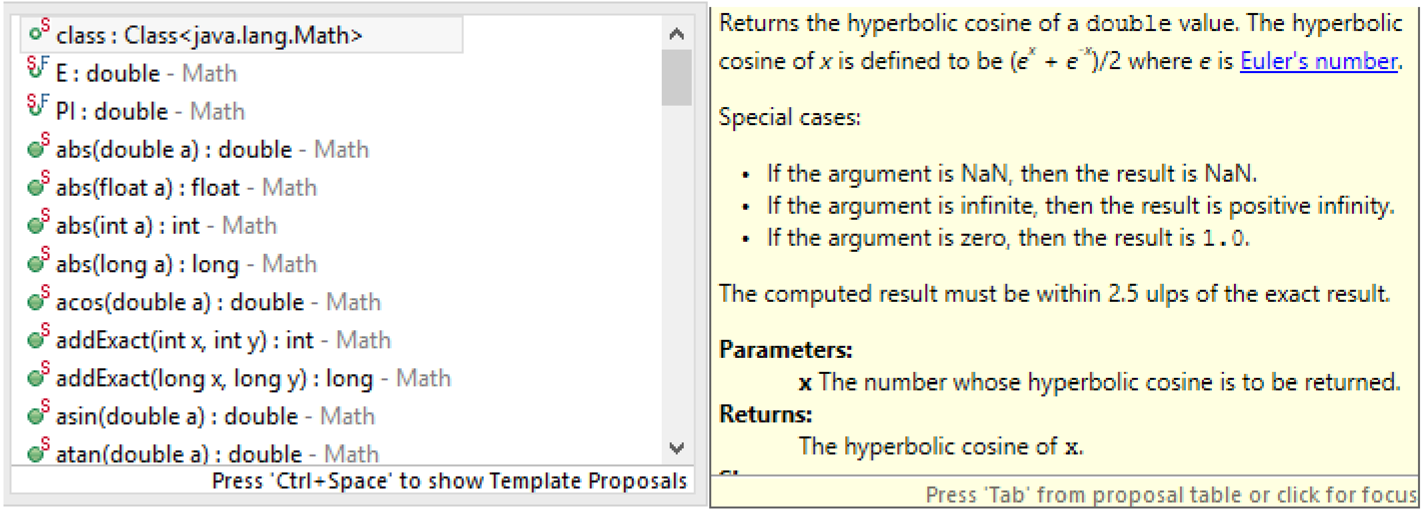
```
a = 8; // a is 8
b = a++;
System.out.println(a); // "9"
System.out.println(b); // "8"

b = a--; // Also decrement
```

Math Functions

- The “Math” class has many static functions and constants

```
double a = Math.PI * 2;
double b = Math.toDegrees( Math.acos(0.34) );
```

```
Math.
```



```
class : Class<java.lang.Math>
E : double - Math
PI : double - Math
abs(double a) : double - Math
abs(float a) : float - Math
abs(int a) : int - Math
abs(long a) : long - Math
acos(double a) : double - Math
addExact(int x, int y) : int - Math
addExact(long x, long y) : long - Math
asin(double a) : double - Math
atan(double a) : double - Math
                    Press 'Ctrl+Space' to show Template Proposals
```

Returns the hyperbolic cosine of a `double` value. The hyperbolic cosine of *x* is defined to be $(e^x + e^{-x})/2$ where *e* is Euler's number.

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is infinite, then the result is positive infinity.
- If the argument is zero, then the result is `1.0`.

The computed result must be within 2.5 ulps of the exact result.

Parameters:
 x The number whose hyperbolic cosine is to be returned.
Returns:
 The hyperbolic cosine of x.

Press 'Tab' from proposal table or click for focus

Tinkering

$$len_{ap} = len\sqrt{1 - \frac{v^2}{c^2}}$$

- Print a value before and after shifting it left or right. What math operation is shifting doing?
- A moving car shrinks in the direction it is moving. The faster it goes, the more it shrinks. Write a program to calculate the apparent length of a 4 meter (*len*) car passing you at 250,000,000 m/s (*v*).
  - The formula is in the upper right:
  - What data type should you use for "length" and "velocity"?
  - Hard code the values in main and print the result
  - c = 300,000,000 m/s