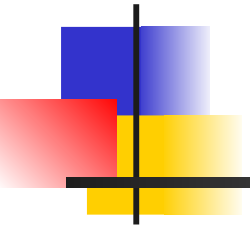


Mathematical Functions

Operations on numeric data types,
esp. functions in the Math class.

James Brucker

Mathematical Functions

- The **Math** class contains methods for common math functions.
- They are *static* methods, meaning you can invoke them using the "Math" class name (more on "static" later).

```
// compute the square root of x
double x = 50.0;
double y = Math.sqrt( x );
```

This means: the sqrt( ) *method in the* Math *class.*

```
// raise x to the 5th power ( = x*x*x*x*x)
double x = 50.0;
double y = Math.pow( x , 5 );
```

Mathematical Functions

Common Math Functions

abs(x)	absolute value of x
cos(x), sin(x), tan(x)	cosine, sine, etc. x is in *radians*
acos(y), asin(y), atan(y), ...	inverse cosine, sine, etc.
toDegrees(radian)	convert radians to degrees
toRadians(degree)	convert degrees to radians
ceil(x)	ceiling: round *up to nearest int*
floor(x)	floor: round *down to nearest int*
round(x)	round to the nearest integer
exp(x)	exponential: $y = e^x$
log(y)	natural logarithm of y ($y = e^x$)
pow(a, b)	a^b (a to the power b)
max(a, b)	max of a and b
min(a, b)	min of a and b

Examples of Using Math Functions

Expression	Result	Type of result
`Math.sqrt( 25.0 );`	5.0	double
`Math.sqrt( 25 );`	5.0	double
`Math.log( 100 );`	4.60517018	double
`Math.log10( 100.0 );`	2.0	double
`Math.sin( Math.PI/2 );`	1.0	double
`Math.cos( Math.PI/4 );`	0.70710678	double
`Math.abs( -2.5 );`	2.5	double
`Math.abs( 12 );`	12	int
`Math.max( 8, -14);`	8	int
`Math.min( 8L, -14L);`	-14L	long
`Math.max( 8.0F, 15);`	15F	float
`Math.pow( 2, 10 );`	1024.0	double
`Math.toRadians( 90 );`	1.5707963	double
`Math.E;`	2.7182818...	double
`Math.PI;`	3.1415926...	double

Overloaded Math Functions

- Some methods in Math have multiple implementations for different parameter types.

`abs(x)`	returns "int" if x is "int"; returns "long" if x is long; returns "float" if x is float; returns "double" if x is "double".
`max(a,b)`	returns "int" if a *and* b are "int"; returns "long" if a *and* b are "long"; etc.
`round(x)`	returns "float" if x is float; returns "double" if x is "double".
`but...`	
`sqrt(x)`	*always* promotes x to double and returns a double. Most math functions are like this (sin, cos, tan, log, log10, ...).

overload: using the same name for functions that have different parameters.

Example: Math.abs(int) has int parameter and returns an int result.

Math.abs(double) has double parameter and returns a double

Overloaded Functions Example

Example	Returns
`Math.max( 2, 10 )`	`(int) 10`
`Math.max( -1L, -4L )`	`(long) -1L`
`Math.max( 2F, 10.0F )`	`(float) 10.0F`
`Math.max( -4.0, 0.5 )`	`(double) 0.5`

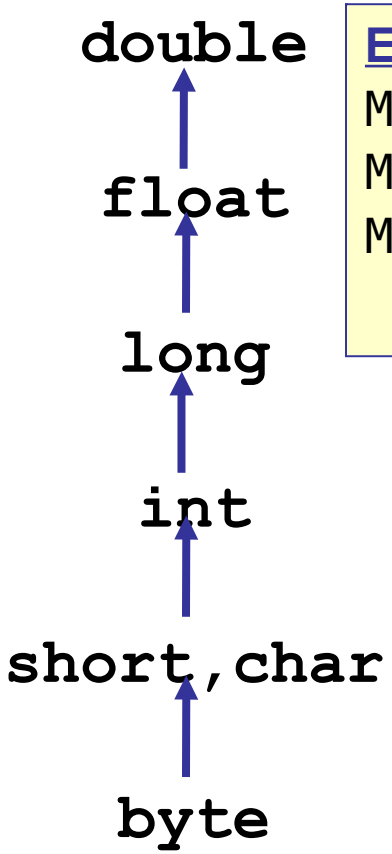
What if the arguments are of different date types?

What should be the data type of the returned value?

Example	Returns
`Math.max( 2, 10.0F )`	?
`Math.max(-1, -4L )`	?
`Math.max( 3, 1.25 )`	?

Functions and Data Types

Java *promotes* one of the arguments until it finds a matching function *prototype*.



Example	Promotion	Then Call
`Math.max( 2, 10.0F )`	`2 to 2.0F`	`max(2F, 10F)`
`Math.max(-1, -4L )`	`-1 to -1L`	`max(-1L, -4L)`
`Math.max( 3, 2.236 )`	`3 to 3.0`	`max(3.0,2.236)`

Automatic Conversions

When necessary, Java automatically "promotes" an argument to a higher data type according to the diagram. These widening conversions will never "overflow" the data type, but may result in lose of precision

Analyzing an Expression

How would you write this in Java syntax?

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Hint: use Math.sqrt(desc)

Your answer:

Analyzing an Expression

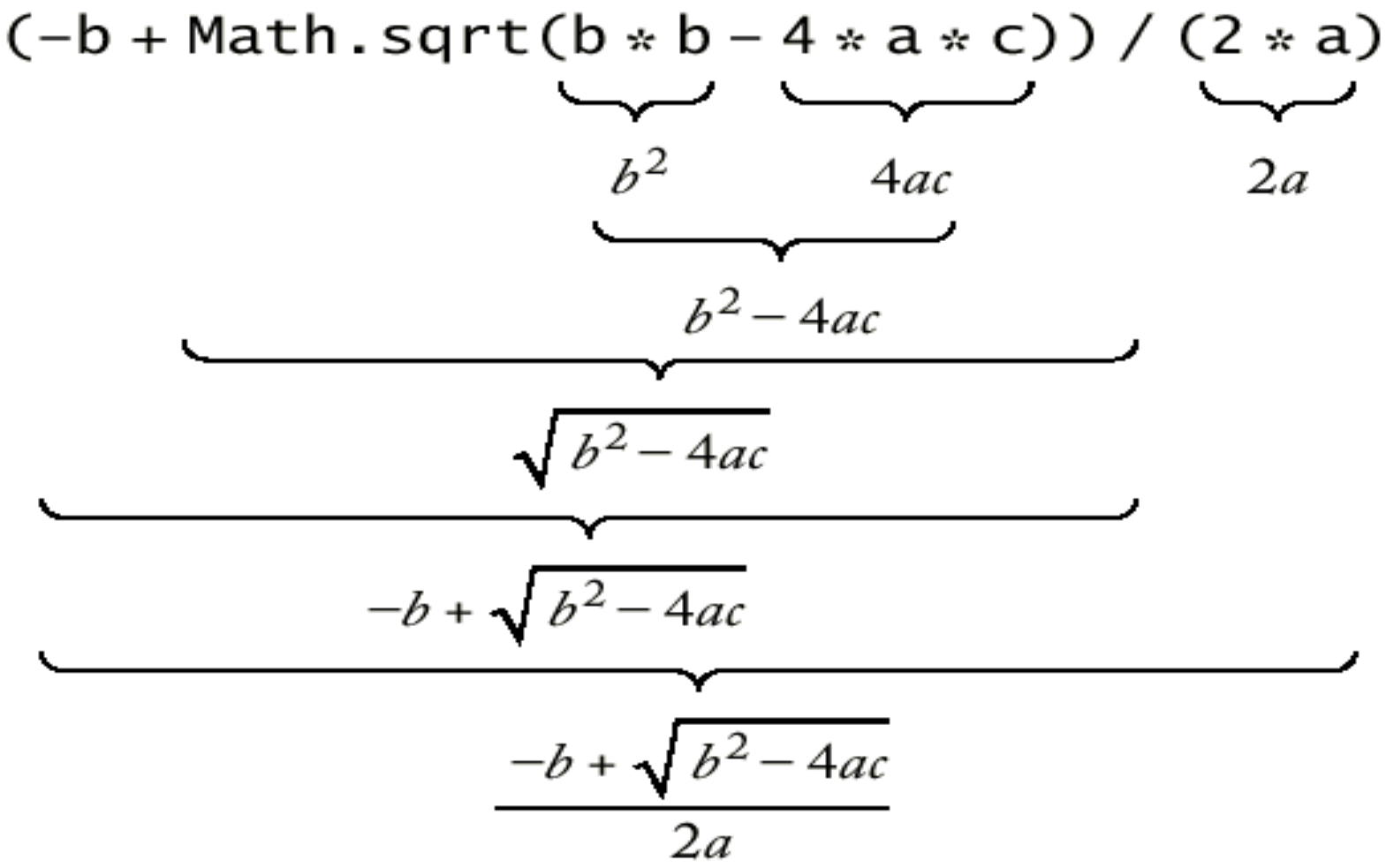
How would you write this in Java syntax?

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

In what order would Java evaluate this expression:

```
x = ( -b + Math.sqrt(b*b - 4*a*c) ) / ( 2 * a)
```

Analyzing an Expression



Converting Strings to Numbers

- Many times we have a String containing a number. How can we convert it to a number?
- Java has "wrapper classes" for primitive data types. These classes perform useful services.

Convert To	Method	Example
		`String s = "1234", t = "0.52";`
`int`	Integer.parseInt( )	`int n = Integer.parseInt(s);`
`long`	Long.parseLong( )	`long m = Long.parseLong(s);`
`float`	Float.parseFloat( )	`float x = Float.parseFloat(t);`
`double`	Double.parseDouble( )	`y = Double.parseDouble(t);`

Warning: if you apply these methods to a String that *does not contain* a valid number, Java will throw an Exception at run-time.

Converting Numbers to Strings

Java *automatically* converts numbers to strings when:

- used in print & println: System.out.println(x);
- concatenated to a String: String s = "x = " + x;

To create a String from a numeric value use toString :

Datatype	to string form	Example
`int`	Integer.toString( )	`Integer.toString( 500/12 );`
`long`	Long.toString( )	`Long.toString( 2L );`
`float`	Float.toString( )	`Float.toString( 1.0F/7.0F );`
`double`	Double.toString( )	`Double.toString( Math.PI );`

For more control over the appearance, use `String.format( )`

Efficient Computation

Here are a couple of common ways to improve your code. You should use them!

Example: find the distance from point (x1, y1) to (x2, y2)

```
double length = Math.hypot(x1-x2, y1-y2);
```

You should not use **`Math.sqrt( )`** for this.
`Math.sqrt()` can overflow or underflow.

`Math.hypot` does not overflow or underflow.

Efficient Computation

Example: evaluate the polynomial:

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

where the polynomial coefficients are a0, a1, a2, a3

```
double p = a0 + x*(a1 + x*(a2 + x*a3));
```

You should not use **`Math.pow(x,n)`** to compute powers of x.

The above formula is more efficient and more accurate.

Scientific Notation

1.0E8 means 1.0×10^{8}

1.0E-9 means 1.0×10^{-9} or 0.000000001

```
final double AVAGADRO = 6.022E+23;
final double NANO = 1.0E-9;
```

Don't write Math.pow(10,-6) for this!

(waste of time, harder to read)