6 Math Class and Constants

Read the first part of this section carefully, the second part is a list of available functions which you will use as reference.

6.1 Introduction

We saw in the last section that we can have *classes* that contain *methods* that display panels and format text, in this section we look at *classes* that contain *methods* to calculate mathematical functions, for example $\sqrt{x}$, $\cos()$, etc. and return their numerical value.

6.2 The JAVA `Math` class

All the mathematical functions are defined to take arguments of type `double` and return a `double` *value*. They are simply used by `<Class.method(<variable>)>` so for example to use the `sqrt()` method you write,

```
double x = 56.90;
double y = Math.sqrt(x);
```

which will set `y` to the square root of `x`.

Trigonometric Function

Method	Action
`cos(x)`	$\cos(x)$
`sin(x)`	$\sin(x)$
`tan(x)`	$\tan(x)$
`acos(x)`	$\cos^{-1}(x)$
`asin(x)`	$\sin^{-1}(x)$
`atan(x)`	$\tan^{-1}(x)$
`atan2(y,x)`	$\tan^{-1}(y/x)$
`toRadians(d)`	Converts degrees `d` to radians
`toDegrees(r)`	Converts radians `r` to degrees

All angles in radians. Only complication is the two $\tan^{-1}()$ functions. `atan(x)` return the normal $\tan^{-1}(x)$ in the range $-\pi/2 \rightarrow \pi/2$ while `atan2(y,x)` return the $\tan^{-1}(y/x)$ in the range $-\pi \rightarrow \pi$ where the quadrant is given by the signs of *both* `x` and `y`.

Exponential, Power and Hyperbolic Functions

Method	Action
`exp(x)`	$\exp(x)$
`expm1(x)`	$\exp(x)-1$ vaild for *small x*
`log(x)`	$\log_e(x)$
`logp1(x)`	$\log_e(x+1)$ valid for *small x*
`log10(x)`	$\log_{10}(x)$
`pow(x,y)`	x^y
`sqrt(x)`	$\sqrt{x}$
`cbrt(x)`	$\sqrt[3]{x}$
`cosh(x)`	$\cosh(x)$
`sinh(x)`	$\sinh(x)$
`tanh(x)`	$\tanh(x)$
`hypot(x,y)`	$\sqrt{x^2+y^2}$

All as expected.

Absolute values, Nearest Integers, Max/Mins

Method	Action
`abs(x)`	$\|x\|$ (any numeric data type)
`ceil(x)`	Smallest integer value not less than x
`floor(x)`	Largest integer value not greater than x
`rint(x)`	x rounded to nearest integer (returns `double`)
`round(x)`	x rounder to nearest integer (returns `long`)
`max(x,y)`	Larger of x and y (any numeric data type)
`min(x,y)`	Smaller of x and y (any numeric data type)
`signum(x)`	Sign of x, being ± 1.0 or 0.0 if $x = 0.0$.

A few complications here:

1. `abs`, `min` and `max` work for all numeric data types, `int` **or** `double` in this course, and return the same data type that they were passed.
2. `round` return a `long` which is a 64 bit integer which you need to convert to normal `int`.

Random Numbers

`random()`	Random number between 0 for 1.

Simple random number generator, see class `java.util.Random` for a better version.

The maths functions have become more extensive with each release of JAVA but are still less extensive than most other numerical lanugaues.

6.3 Constants

In addition to numeric variables, each data type can have *Constants.* These can be used in arithmetic and logical expressions exactly like *variable* but their value **cannot** be altered. Such constants are extremely useful for numerical calculation, for example involving π, checking for errors, or ranges of loops etc. without having to "hard code" explicit numbers into your program. The use of such constants make code much more readable, and also more likely to be correct. There are two types of constants, these being **pre-defined** and **user-defined**.

Pre Defined Constants

The useful predefined constants for novice users are detailed below. In addition there are many additional constants used to control operation of various class types especially if you want to use the full JAVA graphics or menu libraries. There are detailed in the relevant books.

Defined in Math Class:

The following `double` constants are defined in the `Math` class.

`Math.E`	$e = 2.718\ldots$
`Math.PI`	π

This rather limited range of constants *will* be expanded in future versions of JAVA as it becomes a more established scientific language.

Maximum and Minimum Constants:

Each primitive data type has a Maximum and Minimum storable value, for the `int` and `double` these are:

`Integer.MIN_VALUE`	Minimum `int` value	-2147483648
`Integer.MAX_VALUE`	Maximum `int` value	2147483647
`Double.MIN_VALUE`	Minimum `double` value	4.9E-324
`Double.MAX_VALUE`	Maximum `double` value	1.7976931348623157E308

Double Error Constants:

In addition for the `double` type there are three additional constants that are normally used to signify an error condition, these being:

`Double.POSITIVE_INFINITY`	$+\infty$
`Double.NEGATIVE_INFINITY`	$-\infty$
`Double.NaN`	Not-a-Number

The $\pm\infty$ results if the $\pm$ range is exceeded, this is normally the result of dividing by zero. The "Not-a-Number" constant is used to represent an "illegal value", for example if you call `Math.sqrt` with a negative number.

Printing variables set to these values will result in the character strings `Infinity`, `-Infinity` and `NaN` respectively.

Note: Trying to do any other arithmetic on a `double` variable set to any of these constants will result in your program crashing, a good thing as it signifies that something has gone horribly wrong!

User Defined Constants

Any data type can be defined as being *Constant* by use of the `final` qualifier, for example to declare a constant `double` or `int` you simple use.

```
final double LAMBDA = 550e-9;
final int MAXIMUMVALUE = 1000;
```

The value **must** be set when the constant is declared and this value **cannot** be altered. If you try and overwrite this constant you program will crash with an error.

Where and why should you use *constants*,

1. When you have a fixed physical constant, for example "charge on electron", ε_0, h etc. It saves you typing long strings of digits multiple times, (with associated typing errors!).

2. When you have a fixed variable, for example in an optical modeling program you may have a fixed λ. Using a *Constant* prevents fixed numbers appearing in the code. This makes the code **much** easier to modify later on. For example to change to model to a different λ you only need to change one line.

3. It makes the code much more readable, especially if you call your constants by *sensible* names, like `MAXIMUMVALUE`. This results if pieces of code like,

   ```
   if (currentValue < MAXIMUMVALUE)
   ```

   which is easier and understand and debug.

By "convention" *Constants* is JAVA are in "UPPERCASE". This makes them easy to find in the code and also ensures that the colour coding in editors (such as `emacs`) are correct.

Examples

The following on-line source examples are available

- Calculate Plank radiation distribution

$$\rho(\lambda) = \frac{8\pi hc}{\lambda^5}\left[\frac{1}{\left(\exp\left(\frac{ch}{k\lambda T}\right)-1\right)}\right]$$

  PlankRadiation

What Next?

Continue and read the next section on conditional statements.