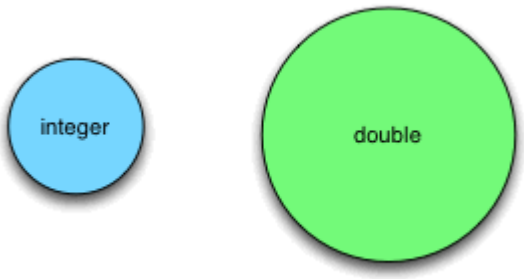
La mayor parte de las veces que trabajamos con números en Java nos es suficientes utilizar los tipos básicos como int y double que nos cubren la mayor parte de nuestras necesidades. De hecho el rango de un tipo double y un integer en Java es el siguiente:

integer: -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

double: 1.40129846432481707e-45 to 3.40282346638528860e+38

La realidad es que es suficiente para la “mayor” parte de las operaciones:



Manejando Factoriales (Java BigInteger)

Lamentablemente hay en algunas casuísticas que se nos puede quedar corto. Parece imposible pero a veces ocurre una de las situaciones más sencillas de reproducir es el calculo de un factorial. Supongamos que tenemos el siguiente código:

```
package com.arquitectura;

public class CalcularFactorial {

public static void main(String[] args) {

System.out.println(factorial(50));
}

static double factorial(double n)
{
if (n == 1.0)
return 1.0;
else
return n*factorial(n-1);
}
}

&amp;amp;amp;amp;nbsp;
```

Este bloque de código nos calculará el factorial de 50 y el resultado será :3.04140932017133 76E64 . Que es un número bastante grande. Ahora bien que sucede en el caso del factorial de 200 por ejemplo. La respuesta de Java será curiosa.

Infinity

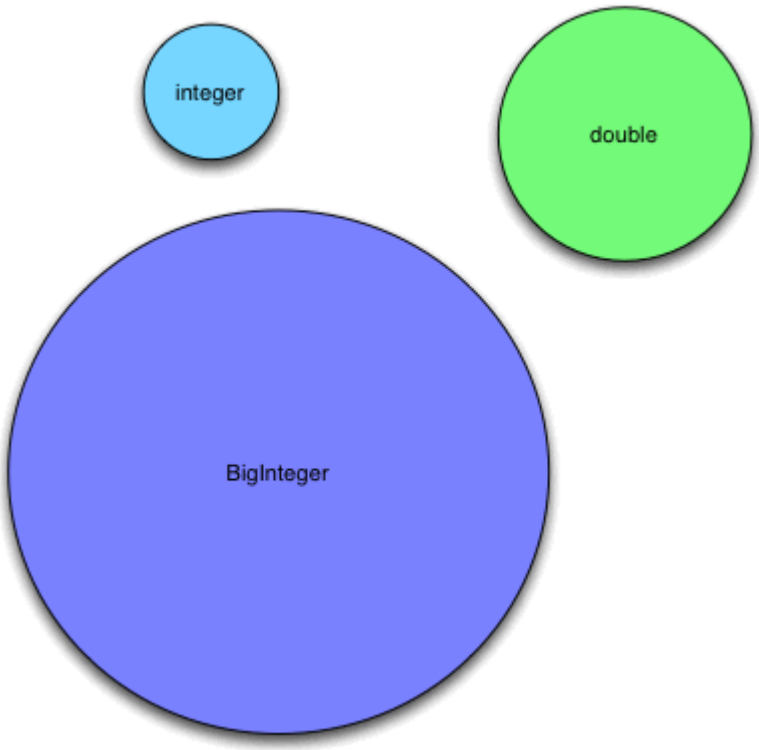
Nos hemos salido de rango y no podemos calcular el factorial de este número. La solución va a venir de una clase Java especial reservada para esta tipología de operaciones y que se denomina BigInteger. Con ella podremos calcular el factorial de números mucho más grandes vamos a verlo:

```
package com.arquitectura;

import java.math.BigInteger;

public class CalcularFactorial2 {

public static void main(String[] args) {

System.out.println(factorial(new BigInteger("1000")));

}
static BigInteger factorial(BigInteger n)
{
if (n.equals(BigInteger.ZERO))
return BigInteger.ONE;
else
return n.multiply(factorial(n.subtract(BigInteger.ONE)));
}

}
```

En este caso hemos creado un objeto de tipo BigInteger y hemos usado los métodos que tiene (multiply) para realizar las operaciones que necesitamos. Una vez procesado el resultado será el siguiente:

```
402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969404800479988610197196058631666872994808558901323829669944590
997424504087073759918823627727188732519779505950995276120874975462497043601418278094646496291056393887437886487337119181045825783647849977012476632889835955
735432513185323958463075557409114262417474349347553428646576611667797396668820291207379143853719588249808126867838374559731746136085379534524221586593201928
890878297308431392844403281231558611036976801357304216168747609675871348312025478589320767169132448426236131412508780208000261683151027341827977704784635868
170164365024153691398281264810213092761244896359928705114964975419909342221566832572080821333186116811553615836546984046708975602900950537616475847728421889
679646244945160765353408198901385442487984959953319101723355556602139450399736280750137837615307127761926849034352625200015888535147331611702103968175921510
907788019393178114194545257223865541461062892187960223838971476088506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911
903754031274622289988005195444414282012187361745992642956581746628302955570299024324153181617210465832036786906117260158783520751516284225540265170483304226
143974286933061690897968482590125458327168226458066526769958652682272807075781391858178889652208164348344825993266043367660176999612831860788386150279465955
131156552036093988180612138558600301435694527224206344631797460594682573103790084024432438465657245014402821885252470935190620929023136493273497565513958720
559654228749774011413346962715422845862377387538230483865688976461927383814900140767310446640259899490222221765904339901886018566526485061799702356193897017
860040811889729918311021171229845901641921068884387121855646124960798722908519296819372388642614839657382291123125024186649353143970137428531926649875337218
940694281434118520158014123344828015051399694290153483077644569099073152433278288269864602789864321139083506217095002597389863554277196742822248757586765752
344220207573630569498825087968928162753848863396909959826280956121450994871701244516461260379029309120889086942028510640182154399457156805941872748998094254
742173582401063677404595741785160829230135358081840096996372524230560855903700624271243416909004153690105933983835777939410970027753472000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```


Como podemos ver un número realmente grande. Así pues siempre podremos usar int o double pero para cosas que realmente se escapan de lo habitual clases como BigInteger o BigDecimal pueden venir muy bien ya que aportan el tamaño adecuado.



Otros artículos relacionados :

StringBuilder

String y Rendimiento

foreach vs Iterator