OBJETOS BIGINTEGER

Los tipos primitivos int, long, float, double son muy eficientes (en uso de memoria y rapidez), pero tienen limitaciones.

El mayor número entero representado con long puede ser para algunos cálculos "pequeño". Para representar enteros "enormes" disponemos de la clase BigInteger que permite utilizar enteros de tamaño "arbitrario", es decir, cualquier tamaño, el límite viene impuesto por la memoria disponible y alguna otra cuestión de representación interna pero en principio, nos olvidamos de que exista un límite en la práctica.
En el siguiente ejemplo tenemos un BigInteger que vemos que es mucho más grande que el long más grande permitido. Y todavía podía ser mucho mayor. Además vemos que a ese número enorme le sumo 1 y la suma se hace correctamente.

```
import java.math.BigInteger;

class Unidad2 {
   public static void main(String[] args) {
     long miLong=                          9223372036854775807l;
     BigInteger bigInt=new BigInteger("99999999999999999999999999999999999999999999999999999999999999999999999999999");
     System.out.println(miLong);
     System.out.println(bigInt);
     BigInteger suma=bigInt.add(new BigInteger("1"));
     System.out.println(suma);
   }
}
```

CREACIÓN DE OBJETOS BIGINTEGER

Si consultas el API observas que el constructor está sobrecargado. Nosotros no nos complicamos y utilizamos sólo 3 formas como ilustra el código del siguiente apartado "operaciones con BigInteger":

- A partir de un String por ejemplo `BigInteger numberA = new BigInteger("20");`
- A partir de una Constante del API, por ejemplo `BigInteger numberB = BigInteger.TEN;`
- A partir de un método. Por ejemplo a partir de un método que implementa una operación con BigInteger y que devuelve el resultado en un nuevo integer, por ejemplo `BigInteger numberC=numberA.add(numberB);`

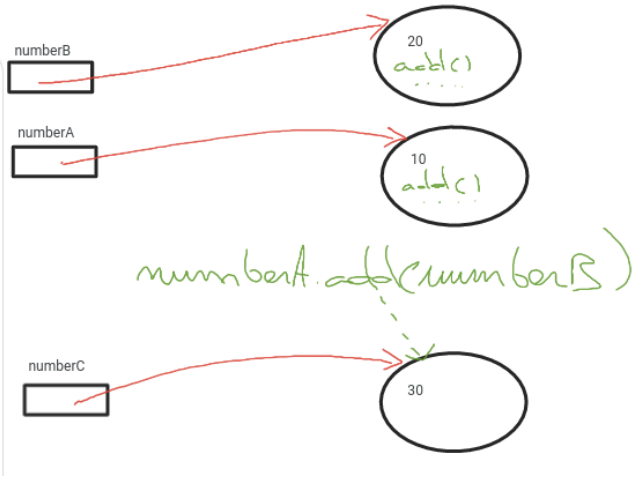
Además en el último ejercicio veremos también que se puede obtener un BigInteger del método NextBigInteger()

OPERACIONES CON BIGINTEGER

Consulta el API y observa que todas las operaciones devuelven un nuevo BigInteger que contiene el resultado, por ejemplo experimentamos con la operación add()

```
import java.math.BigInteger;

class Unidad2{
  public static void main(String[] args) {
    BigInteger numberA = new BigInteger("20");
```

```
    BigInteger numberB = BigInteger.TEN;


    System.out.println("numberA antes de add() vale:  " + numberA);
    numberA.add(numberB);//esto no modifica numberA si no que devuelve un nuevo BigInteger
    System.out.println("numberA despues de hacer  numberA.add(numberB) NO CAMBIA. numberA=" + numberA);

    BigInteger numberC=numberA.add(numberB);
    System.out.println("numberC es un nuevo bigInteger que tiene la suma de A y B. numberC= " + numberC);


  }
}
```



Ejemplo: más operaciones con BigInteger

```
import java.math.BigInteger;

class Unidad2{
  public static void main(String[] args) {
    BigInteger numberA = new BigInteger("98765432123456789");
    BigInteger numberB = BigInteger.TEN;
    BigInteger numberC;

    numberC = numberA.add(numberB);
    System.out.println("numberC = " + numberC);

    numberC = numberA.multiply(numberB);
    System.out.println("numberC = " + numberC);

    numberC = numberA.subtract(numberB);
    System.out.println("numberC = " + numberC);

    numberC = numberA.divide(numberB);
    System.out.println("numberC = " + numberC);

    numberC = numberA.mod(numberB);
    System.out.println("numberC = " + numberC);

    numberC = numberA.pow(2);
    System.out.println("numberC = " + numberC);

    numberC = numberA.negate();
```

```
    System.out.println("numberC = " + numberC);


  }
}
```

LOS OBJETOS BIGINTEGER SON INMUTABLES

Un objeto es inmutable si una vez creado no se puede modificar su estado. En java los objetos String y otros muchos son inmutables. Más adelante entenderemos porqué es conveniente que algunos objetos sean inmutables. Los objetos BigInteger también son inmutables. En el siguiente ejemplo, realmente no estoy modificando el objeto referenciado por numberA si no que el método add() generó un nuevo objeto y a la referencia numberA se le asigna la referencia de este nuevo objeto. No se ha modificado el objeto original, si no que se creo uno nuevo que recoge los cambios.

```
import java.math.BigInteger;

class Unidad2{
 public static void main(String[] args) {
   BigInteger numberA = new BigInteger("20");
   BigInteger numberB = BigInteger.TEN;


   System.out.println("numberA antes de add() vale:  " + numberA);
   numberA.add(numberB);//esto no modifica numberA si no que devuelve un nuevo BigInteger
   System.out.println("numberA despues de hacer  numberA.add(numberB) NO CAMBIA. numberA=" + numberA);

   numberA=numberA.add(numberB);//numberA apunta a un nuevo objeto devuelto por add()
   System.out.println("numberA ahora referencia a  un nuevo bigInteger que tiene la suma de A y B.  " + numberA);


 }
}
```

EJERCICIO U2_B9_E1:

Comprueba en el api que Scanner tiene un método nextBigIntegerl(). Modifica el ejemplo anterior pidiendo los valores de numberA y numberB por teclado.

EJERCICIO U2_B9_E2: La Galaxia de Andrómeda está a 2,5 millones de años luz de la tierra. Si me empeño en hablar en kilómetros en lugar de años luz, y quiero calcular los kilómetros de ida y vuelta a Andrómeda, observo, por el resultado negativo obtenido, que sobrepasé el rango de long (y recuerda que el resultado puede ser positivo y también haber sobrepasado dicho rango). Se pide hacer el cálculo con BigInteger.

```
class Unidad2 {
    public static void main(String[] args) {
        long kilometrosEnLong = (long)365 * 24 * 60 * 60 * 300000 * 2500000*2;
        System.out.println("Kilometros a Ándromeda en Long ida y vuelta: " + kilometrosEnLong);
    }
}
```