BigNumber

ÍNDICE

SEGUNDA PRÁCTICA

La segunda práctica obligatoria de Programación tiene el siguiente objetivo:

Crear el objeto y la clase BigNumber para evitar los problemas de capacidad y precisión que tienen los tipos primitivos en Java a la hora de almacenar números. Para ello trataremos a los números como cadenas o Strings, y tendremos que utilizar los algoritmos que se emplean para realizar las operaciones en papel.

Una de las operaciones que tenemos que realizar es la de comparar números. El programa tiene que devolver un '1' si la primera String pasada por test es mayor, un '0' si son iguales y un '-1' si la de abajo es mayor. Realmente, lo que comparamos no es la String en sí, sino los números que están almacenados en ellas.

Para ello, he creado la función quitarCeros(); que contiene un int (primerCharNoCero) que empieza teniendo el valor de -1, y creamos un bucle que recorra la cadena que le pasamos hasta encontrar un número diferente a 0. Si el primer número ya es diferente a 0, enviaremos la String igual pero sino, utilizando el método charAt(i) cambiaremos el valor de primerCharNoCero para crear una substring a partir del primer caracter de la cadena que no sea 0. De esta forma, por ejemplo, '00056' y '56' serán iguales.

Creando b1 y b2

```
BigNumber b1, b2;

b1 = new BigNumber("2");
b2 = new BigNumber("02");
```

Lo que hacemos con el new es llamar al constructor, que tiene la siguiente estructura:

```
public BigNumber(String s) {
    this.valor = s;
}
```

Dentro de this.valor guardas el valor que le pasas a s con los dos objetos BigNumber

Con el método assertEquals, el primer objeto (b1) llama al método y es el valor que se guarda en this.valor. El object other es el otro objeto, el que no llama a la comparación sino el que es llamado. Ese valor lo guardamos en b y corresponde a b2.

```
public boolean equals(Object other) {
    BigNumber b = (BigNumber) other;
    if (b.valor.equals(this.valor)) return true;
    return false;
}
```

Para pasar los tests tenemos que meter las funciones en el constructor, por ejemplo la de quitar ceros de esta forma:

This.valor = quitaceros(s)

SUMAS

PARA QUE LAS STRINGS TENGAN LA MISMA LONGITUD, DEBEMOS PONER TANTOS 0 A LA IZQUIERDA COMO SEA NECESARIO.

Para ello hemos creado la función añadirCeros(String valor, int longitudFinal); Dentro de la funcion de sumar crearemos dos condiciones:

```
if (s1.length() > s2.length()) {
    s2 = añadirCeros(s2, s1.length());
}
```

```
if (s1.length() < s2.length()) {
    s1 = añadirCeros(s1, s2.length());
}
```

De esta forma, comparamos cual es la mayor y le pasamos siempre como primer argumento la String cuya longitud es menor, y el valor(int) de la longitud de la String mayor. Creamos tambien un int que será el resultado de la longitud de la String mayor menos la longitud de la String menor y ese será el número de ‘0’ que habrá que añadir a la izquierda de la String menor.

Las sumas tienen que hacerse generando un bucles que itere sobre las dos Strings, empezando desde el último carácter hasta el primero.

Creamos una String vacía (stringResult), un int para guardar el resultado y un int (carry) que empezará valiendo 0 para el acarreo. Luego creamos dos variables int (d1, d2), cuyo valor respectivamente será el resultado de s1.parseInt y s2.parseInt , cuyos valores habremos guardado en un char con el método charAt. Como el método parseInt solo funciona con Strings, pasaremos el char a String.

```
for (int i = s1.length() - 1; i >= 0; i--) {
    int d1 = Integer.parseInt(s1.charAt(i) + "");
    int d2 = Integer.parseInt(s2.charAt(i) + "");

    result = d1 + d2 + carry;
```

Falta una condición para los casos en los que haya acarreo:

```
if (result > 9) {
                carry = 1;
                result -= 10;
            } else {
                carry = 0;
            }

            stringResult = result + stringResult;
        }
//En caso de que no haya más dígitos a los que sumarle el carry pero
éste valga 1, le sumamos el 1 a la String.
// De esta forma, no hará la suma como a un int, sino que
literalmente añadirá ese '1' a lo que hubiera detrás.
        if (carry > 0) {
            stringResult = carry + stringResult;
        }
```

De esta manera, si el resultado es mayor a 9, carry vale 1 y se le resta 10 al resultado de d1 + d2 + carry. Si no es así, carry valdrá 0 en la próxima iteración.

Y por último, devolvemos el resultado en forma de BigNumber(stringResult);

RESTAS

Para el correcto funcionamiento de nuestro programa también debemos igualar las Strings.

Las restas tienen el punto positivo de que los Strings que le pasamos al programa via test ya están listos para restar: El grande arriba y el pequeño abajo.

Para resolver las restas es un proceso muy similar al de la suma pero con algunas diferencias:

Por eso mismo crearemos también dos variables int(d1, d2) que corresponderán al caracter en la posición de 'i' con el método charAt pero a d2 le sumaremos el acarreo.

IMPORTANTE: Cuando el número de arriba es menor que el de abajo, al de arriba se le suman 10 y nos llevamos 1. Ésta se suma al número de abajo a la izquierda. Entonces, siempre y cuando el número de arriba sea menor al de abajo, el resultado pasa a ser la suma del d1 + 10 menos el d2 más el acarreo. Guardaremos el resultado de esa resta en la variable int (result) que hayamos creado para guardar el resultado y el carry que hayamos creado ahora valdrá 1.

La condición sería así:

```
if (d2 > d1) {
    result = 10 - d2 + d1;
    carry = 1;
} else {

    result = d1 - d2;
    carry = 0;
}
```

Y por último, devolvemos el resultado en forma de BigNumber(stringResult);

MULTIPLICACIÓN

Para multiplicar tomamos el primer dígito de la derecha de b2 y lo multiplicamos por todos los digitos de b1.

Para ello, creamos una función aparte que solamente recibirá un dígito del factor inferior. Ese número empezará siendo el primero a la derecha y avanzará hacia la izquierda.

Esta función se encargará de calcular el producto del dígito que le habremos pasado por los dígitos del factor superior uno por uno. Para ello crearemos también una variable de acarreo (carry) y una condición: si el resultado es mayor a 10, lo dividimos entre 10 y guardamos el residuo en la variable del resultado y el cociente en la variable carry para sumarlo al siguiente producto. Si no hay siguiente producto, lo encadenamos delante del resultado.

```
result = (number * other) + carry;
    if (result > 9) {
        carry =  result / 10;
        result = result % 10;
    } else {
        carry = 0;
    }

    stringResult = result + stringResult;
}

if (carry > 0) {
    stringResult = carry + stringResult;
}
```

Cada vez que el programa hace una multiplicación por un dígito, le enviamos el resultado a la función de las sumas para que sume los resultados del factor inferior por todos los factores superiores y se obtiene el producto.

El problema viene con el espacio que debemos dejar aqui:

```
 154
*43
 ----
  462
 616
```

Por ello debemos crear una nueva función que se encargue de añadir un cero al final de la String con cada iteración del bucle de nuestra función principal:

```
private void añadirCerosDerecha(int cantidad) {
    for (int i = 0; i < cantidad; i++) {
        valor += '0';
    }
```

La cantidad se la pasamos por parámetro con la variable ‘ceros’ de la función principal que se irá incrementando conforme avancemos de dígito en el factor inferior. El objetivo es este:

```
 154
*43
 ____
  462
 6160
 ____
 6622
```

POTENCIAS

Una vez tenemos la multiplicación resuelta, las potencias no son muy complejas. Básicamente guardamos la base y el exponente que nos dan como nuevos BigNumber Y la base siempre empezará en 1 porque cualquier número elevado a 1 es ese mismo número.

Ahora la forma de resolver este problema es la siguiente:

```
while (i.compareTo(new BigNumber(n + "")) < 0) {
    result = result.mult(this);
    i = i.add(new BigNumber("1"));
}
```

Mientras se cumpla la condición de que ‘i’, es decir, la potencia, sea inferior al número que nos pasan como exponente, la base debe multiplicarse por sí misma. Para ello la mandaremos a la función de las multiplicaciones. Le iremos mandando ‘i’ a la función de las sumas para que le sume 1 con cada iteración. Por ejemplo, en el caso de 5 elevado a 2, el 5 se multiplicará una vez por si mismo ya que en la siguiente iteración ‘i’ valdrá 2 y ya será igual a 2.

FACTORIALES

Realizaremos un proceso similar al de las potencias pero con algunas diferencias.

En este caso, nos pasan por test un número y este tiene que multiplicarse por los que hay antes que él hasta llegar a 1 incluido. Primero creamos dos nuevos BigNumber: result, que contendrá el número del que queramos sacar su factorial, por ejemplo 5. Y el otro será el result -1, que es el número por el cual empezaremos a multiplicar nuestro 5. Y creamos la siguiente condición:

```
while (other.compareTo(new BigNumber( "0")) > 0) {
   result = result.mult(other);
   other = other.sub(new BigNumber("1"));
}
```

Mientras se cumpla la condición de que el segundo número (en este caso 5 - 1 = 4) sea superior a 0, multiplicaremos el primer número (5) por el segundo mandándolos a la función de las multiplicaciones y le restaremos 1 al segundo número mandándolo a la función de las restas.

De esta forma iríamos acumulando los productos en la variable result y quedaría esto:

5 x 4 = 20
20 x 3 = 60
60 x 2 = 120
120 x 1= 120.

Devolvemos el resultado y ya hemos acabado

La división me daba errores así que no la he podido completar, y como la división era una pieza clave para las sqrt y mcd tampoco las he hecho.