NUMBERS

Álvaro Rodríguez Fresneda

Programación

1r CFGS DAW

ÍNDICE

OBJETIVOS

El objetivo de la práctica es crear un programa que tenga tres funciones principales:

-Una función llamada say, que reciba un número del 0 a 9,223,372,036,854,775,807 y que devuelva ese mismo número escrito en catalán.

-Otra función llamada words que reciba un número escrito en catalán y devuelva su valor numérico.

--La última, llamada oper que opere strings que contengan números escritos en catalán y te dé el resultado en un string escrito en catalán.

El programa tiene que superar una serie de tests que se encargan de verificar que funciona correctamente

DESCRIPCIÓN DEL PROGRAMA

NUMBERS:

Lo primero que tenemos que tener claro es que hay que dividir cada una de las tres funciones en funciones más pequeñas, dependiendo del número que nos pida el test.

Para no hacerlo tan complejo empezaremos desde el principio e iremos contemplando casos según nos aparezcan.

Primer paso es resolver todos los números únicos. En mi caso, yo decidí utilizar un array para los valores del 0 al 19 y otro del 20 al 90, ya que son los números que tienen un nombre único para cada caso.

-El array del 0 al 19 será un array de 20 elementos en el que cada posición equivale a su respectivo nombre en catalán:

```
private static String [] nombresUnics = new String[] {
     "Zero", "Un", "Dos", "Tres", "Quatre", "Cinc", "Sis",
"Set", "Vuit", "Nou", "Deu",
```

```
        "Onze", "Dotze", "Tretze", "Catorze", "Quinze",
"Setze", "Disset", "Divuit", "Dinou"
};
```

De esta forma, si queremos un número del 0 al 19 solo necesitaremos imprimir su posición en el array para tener su nombre correspondiente en catalán.

-El array del 20 al 90 funciona exactamente igual pero solo con las decenas:

```
private static String [] decenes = new String [] {
    "", "", "Vint", "Trenta", "Quaranta", "Cinquanta",
"Seixanta", "Setanta", "Vuitanta", "Noranta"
};
```

Dejamos en blanco las dos primeras posiciones, ya que estas ya están contempladas en el array anterior.
De esta manera si nos piden un número del 20 al 99 solamente tendremos que crear una variable int para dividir ese número entre 10. Al tratarse de un int, el cociente de la división nos indica las decenas, y crearemos otra variable int para guardar el resto, que serán las unidades.

Si hacemos esto, solo será necesario enviar la decena al array de decenas que acabamos de crear y la unidad al array de números del 0 al 19 y pedirle a las dos funciones que nos devuelvan la posición equivalente del array.

También debemos contemplar que entre las decenas y las unidades hay un guión ”-” y que en el caso de los números del 20 al 30 hay una “i” entre guiones. Una vez establecidas estas condiciones le añadimos estos caracteres a las strings vacías que hemos creado para guardar el resultado y ya tendremos contemplados todos los números del 1 al 99.

Un paso importante que cabe recalcar es que lo primero de todo es dividir las funciones por franjas numéricas. Por ejemplo, números entre 100 y 1000, entre 1000 y 1000000, etc. Así tenemos más controlado el programa y decidimos que el número pase por una función u otra dependiendo de su tamaño.

Para las centenas realizaremos un proceso similar. Primero necesitaremos extraer la centena, y lo haremos dividiendo el número que nos pase el test por

100. Guardaremos esa parte en una string vacía y si esta equivale a uno, le introduciremos la palabra Cent al string. Si las centenas son mayores que uno nos interesa que el programa mande al número en cuestión al array del 0 al 19 y nos lo sustituya por su respectiva posición + “-cents”, para introducirlo en nuestra string vacía. Una vez contempladas las centenas, lo que nos interesa es guardar el resto para obtener las decenas y unidades. En mi caso, yo he creado una función que analiza todos los números positivos del 0 al 99 y los envía a su array o función correspondiente para obtener la parte restante y sumarla a la string que antes estaba vacía y que ahora tiene las centenas.

Una vez llegados a este punto solo necesitaremos repetir este proceso creando funciones para los millares, los millones, los billones y los trillones pero teniendo en cuenta una serie de características:

-El dividendo y el resto de cada función se irá incrementando. Por ejemplo, en la función de los millares nos interesa dividir entre 1000, en la de los millones entre 1000000 etc.

-Habrá que aplicarle condiciones al resto para enviarlo a su función correspondiente. Por ejemplo, si el resto es 0 lo que queremos es que la función nos devuelva solo los millares, ya que lo demás es 0. Pero en cambio si el resto es cualquier número entre 100 y 999 (p.ej. 137), habrá que enviarlo a la función de las centenas para que lo analice individualmente y esta se encargará de dividirlo entre 100, guardar la centena (1) y enviar el resto (37) a la función que contempla positivos de 0 a 99.

-A partir de mil millones ya no hay que pasar los números de long a int, sino utilizar el long hasta el final

-A partir de aquí solo se trata de seguir creando intervalos de números para enviar el que nos pida el test a su función correspondiente, p ej.

```
if (n > 1000000 && n < 1000000000000L){
    return  sayMillones(n);
}
```

Así envío el número a la función de los millones, que lo irá descomponiendo, añadiendo el nombre único de cada función al string vacío que creemos (milió, trilió...) y mandando el resto después a las funciones que contemplan los números inferiores.

Adjunto la función de los millares, que es la más compleja ya que es la que más condiciones contempla:

```
private static String sayMillares(long n){
    String nombreMillar = "";
    String millaress = "";
    int millar = (int)n/1000;
    if (n > 999 && n < 100000) {
        if (millar == 1) {
            nombreMillar = "Mil";
        } else {
            nombreMillar = sayPositivoDe0A99(millar) + "
mil";
        }
    }if (n >=100000 && n < 1000000){
        nombreMillar = sayCentenes(millar) + " mil";
        }
    int residu = (int)n%1000;
    if (residu >=100) {
        millaress = nombreMillar + " " +
sayCentenes(residu).toLowerCase();

    }
    if (residu < 100){
        millaress = nombreMillar + " " +
sayPositivoDe0A99(residu).toLowerCase();
    }
    if (residu == 0){
        millaress = nombreMillar;
    }
        return millaress;
}
```

Para resolver los números negativos es tan simple como crear una condición que se encargue de que si un número es menor que 0, se añada un “Menys ” al string, se multiplique ese número por -1 y se envíe el número ya convertido a positivo a la función principal.

WORDS

La parte del words es al contrario, el test nos pasa el nombre del número y nosotros tenemos que devolver el valor numérico.

Para ello empezaremos igual que antes: Escribiendo los números únicos. Yo he utilizado un switch para contemplar cada caso único y luego creamos una función que se encargue de introducir la palabra que nos envía el test en un array de strings, en el que cada posición del array estará separada por un guión.

Con el método split de los Strings. Buscamos un patrón o expresión regular para dividir el string y que cada parte de la división ocupe una posición del array.

Por ejemplo con el número Quaranta-set, la función crearía un array de dos posiciones que contenga lo siguiente:
[Quaranta, set]

De esta forma solo tendremos que enviar cada posición del array al switch que hemos hecho con los números únicos, convertirlos a un long y ir sumando posiciones. Por ejemplo, 40+7 y nos dará 47.

Con los números negativos utilizaremos el método equals de String para que, si el número que nos envían separado por posiciones en un array tiene como primera posición la palabra “Menys” automáticamente se multiplique el número por -1.

Ahora necesitaremos crear una función que separe las palabras en un array por espacios en blanco. Por ejemplo, el número 237.

Se escribe dos-cents trenta-set

Esta función lo que hace es crear un array con esto: [dos, cents, trenta, set]

Crearemos una condición que, utilizando el método equals nos diga si encuentra la palabra “cent”. En ese caso, tendrá que sumarle 100 a lo que venga despues. Si en cambio encuentra la palabra “cents”, tendrá que multiplicar lo que haya antes por cien. Utilizando el ejemplo anterior sería:

Dos = 2
Cents = 100

Multiplica la primera posición por la segunda y suma el resto:

2*100+30+7= 237

Con los millares podemos hacer algo parecido. Como la palabra “mil” no varía dependiendo de las unidades, decenas o centenas de miles que haya nos aseguraremos de si la palabra”mil” sale en la primera posición del array. En ese caso, sumaremos mil a lo que haya detrás. Si, en cambio, la palabra mil no se encuentra en la primera posición del array multiplicaremos por mil la posición que haya justo delante.

P. ej 1234

Sería algo así: 1000+2 *100+30+4

En cambio, 4567

4*1000+5*100+60+7

Con los millones, billones y trillones encontramos una complicación, y es que si multiplicamos por 1000 el número en cuestión este se agrandaría más de lo que corresponde. Con lo cual, la solución es crear una variable temporal que guarde los millones, dejar en 0 la otra variable y seguir operando con normalidad para al final sumar las variables.

De esta forma en vez de multiplicar los millones por los millares, los sumamos.

El oper no he sabido hacerlo.