

## 138 - Street Numbers

[Enlace original](#)

Una programadora de computadoras vive en una calle con casas numeradas consecutivamente (desde 1) por un lado de la calle. Cada noche ella sale a pasear su perro dejando su casa y girando al azar a la izquierda o a la derecha, camina hasta el final de la calle y vuelve. Una noche suma los números de las casas que pasa (excluyendo la suya). La siguiente vez que camina, comienza para el otro lado repitiendo la suma y encuentra, para su asombro, que las dos sumas son iguales. Aunque esto se determina en parte por su número de casa y en parte por el número de casas en la calle, ella sin embargo siente que esta es una propiedad deseable para su casa y decide que todas sus casas subsecuentes tendrán esa propiedad. Escribe un programa para encontrar los pares de números que satisfagan esta condición. Para comenzar su lista los dos primeros pares son: (número de casa, último número en la calle):

6,8  
35,49

Se piden 10 pares de números.

## StreetNumbers

Encuentra la posición de la casa de una persona asumiendo que las casas están numeradas consecutivamente. La casa es el punto de partida, por lo que la diferencia entre la suma de la numeración de todas las casas entre la primera y la casa de la persona y la suma de todas las casas después de la casa de la persona debe ser la misma.

## TARJETAS-CRC

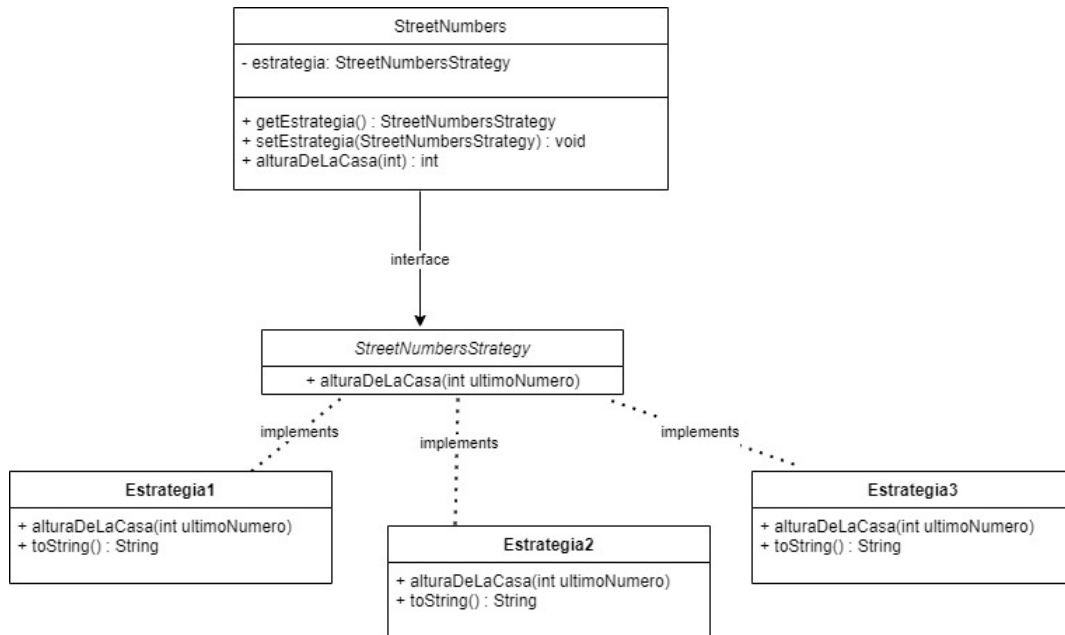
StreetNumbers: Interfaz que permite probar distintas estrategias.

StreetNumbersStrategy

Estrategia1: Proporciona el cálculo al recorrer de izquierda a derecha obteniendo la sumatoria de las mismas a medida que incrementa la posición. Tiempo de ejecución elevado.

Estrategia2: Proporciona el cálculo al recorrer de izquierda a derecha según el teorema de sumas consecutivas a medida que incrementa la posición. Tiempo de ejecución medio.

Estrategia3: Proporciona el resultado al realizar el cálculo con la posición deseada. Tiempo de ejecución óptimo.



## Complejidad Computacional

Estrategia N° 1:

$O(N^2)$

Estrategia N° 2:

$O(N)$

Estrategia N° 3:

$O(1)$

Al evaluar las distintas complejidades computacionales que tienen las distintas estrategias se puede comprobar que la Estrategia 1 tiene un complejidad del tipo  $O(N * (\text{posición} + N))$  la cual hará que el

algoritmo tenga un retardo mucho mayor al aplicado en las estrategias 2 y 3 las cuales tienen una complejidad computacional lineal y constante, respectivamente. En otras palabras, obtenemos el mismo resultado con distintos algoritmos.

### Ejemplo de ejecución:


```
Estrategia utilizada: Estrategia 1
Reporte de tiempo: 121303.51 ms
Resultados:
    6,8
    35,49
    204,288
    1189,1681
    6930,9800
```

---

```
Estrategia utilizada: Estrategia 2
Reporte de tiempo: 13218.06 ms
Resultados:
    6,8
    35,49
    204,288
    1189,1681
    6930,9800
    256,131072
    7742,131528
    11707,132113
    19813,134033
    115619,134705
```


```
Estrategia utilizada: Estrategia 3
Reporte de tiempo: 8.65 ms
Resultados:
    6,8
    35,49
    204,288
    1189,1681
    6930,9800
    256,131072
    7742,131528
    11707,132113
    19813,134033
    25162,135816
```

### Solución 1




```
public class StreetNumber {  
    public static int metodo1(int n){  
        if (n < 3) return -1;  
        for (int i = 2; i <= n; i++) {  
            int sumaIzq = 0;  
            for (int j = 1; j < i; j++)  
                sumaIzq += j;  
            int sumaDer=0;  
            for (int k = i + 1; k <= n; k++)  
                sumaDer += k;  
            if (sumaDer == sumaIzq) return i;  
        }  
        return -1;  
    }  
}
```

## Solución 2



```
public static int metodo2(int n) {  
    if (n < 3) return -1;  
    for (int i = 1; i <= n; i++) {  
        int sumaIzq = i * (i - 1) / 2;  
        int sumaDer=(n*(n+1)/2-i*(i+1)/2);  
        if (sumaDer == sumaIzq) return i;  
    }  
    return -1;  
}
```

## Solución 3



```
public static int metodo3(int n) {  
    if (n<3) return -1;  
    double i = Math.sqrt((Math.pow(n, 2)+n)/2);  
    int entera = (int) i;  
    if ((i-entera) == 0) return (int) i;  
    return -1;  
}
```

```
public static void main(String[] args) {  
    //int n= 57121;  
    //int n= 332928;  
    int n= 1940449;  
  
    Calendar tIni = new GregorianCalendar();  
    System.out.println(metodo3(n));  
    Calendar tFin = new GregorianCalendar();  
    long diff = tFin.getTimeInMillis()-tIni.getTimeInMillis();  
    System.out.println("Tiempo de ejecuci n del metodo 3 " + diff);  
    System.out.println("-----");  
  
    tIni = new GregorianCalendar();  
    System.out.println(metodo2(n));  
    tFin = new GregorianCalendar();  
    diff = tFin.getTimeInMillis()-tIni.getTimeInMillis();  
    System.out.println("Tiempo de ejecuci n del metodo2 " + diff);  
    System.out.println("-----");  
  
    tIni = new GregorianCalendar();  
    System.out.println(metodo1(n));  
    tFin = new GregorianCalendar();  
    diff = tFin.getTimeInMillis()-tIni.getTimeInMillis();  
    System.out.println("Tiempo de ejecuci n del metodo 1 " + diff);  
}
```

## Ejemplos

//	6	8
//	35	49
//	204	288
//	1189	1681
//	6930	9800
//	40391	57121
//	235416	332928
//	1372105	1940449
//	7997214	11309768
//	46611179	65918161