

Trabajo JUnit



Entornos de Desarrollo
1º Desarrollo de Aplicaciones Web
Zahira Zamora Camacho
Carlos Domínguez Merchán
Diciembre 2012


Índice

1.-Descripción del trabajo.....	3
1.1.-Proyecto Intervalos.....	3
1.2.-Proyecto Rectangulador	3
2.-Desarrollo del trabajo	4
2.1.- Proyecto Intervalos.....	4
2.1.1.- Nombres de los métodos implementados.....	4
2.1.2.- Casos implementados.....	5
2.1.3.- Código del solapamiento de intervalos.....	6
2.1.4.- Resultados Test Intervalos.....	8
2.2.- Proyecto Rectangulador	9
2.2.1.- Funcionamiento del código	9
2.2.2.- Resultados Test Rectangulador	11

1.-Descripción del trabajo

1.1.-Proyecto Intervalos

Se pide diseñar todos los casos de prueba para el proyecto Intervalos e implementarlos mediante el módulo JUnit para Eclipse. Además el código de intervalos tiene errores y hay que corregirlos.

 **int Intervalos.solapamiento(int x1, int x2, int y1, int y2)**


Calcula el solapamiento de dos intervalos [x1,x2) y [y1,y2) Por ejemplo, [5, 10) y [0,9) solapan en los elementos 5, 6, 7 y 8, ya que el 9 no esta incluido.

Parameters:
x1 Inicio del intervalo x
x2 Fin del intervalo x
y1 Inicio del intervalo y
y2 Fin del intervalo y

Returns:
El solapamiento de los intervalos

1.2.-Proyecto Rectangulador

Para el proyecto Rectangulador se entregan los casos de prueba implementados en Junit, y se pide implementar el contenido del método para que pase los casos de prueba y además realice lo siguiente:

 **int Rectangulador.rectangulosDiferentes(long nOri)**

Calcula el número de rectángulos distintos que se pueden formar con varios elementos. Se considera que el rectángulo 5x7 y el rectángulo 7x5 son iguales. No se considera rectángulos a los que tienen lado 1.

Parameters:
i numero de elementos
nOri

Returns:
el numero de rectángulos diferentes que se pueden formar con este número de elementos

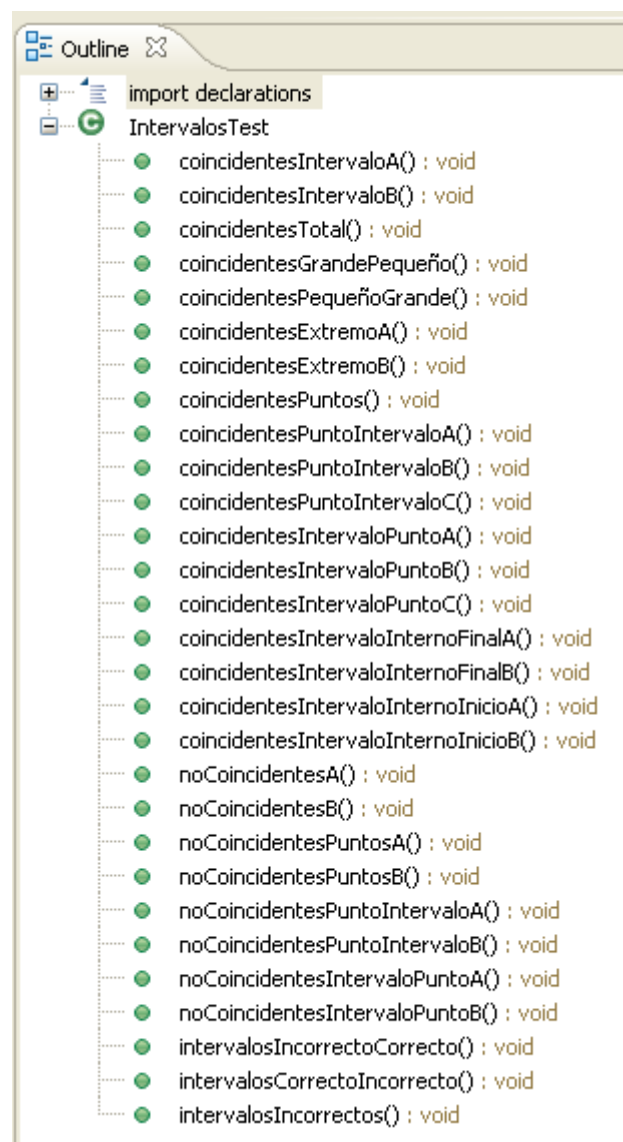
2.-Desarrollo del trabajo

2.1.- Proyecto Intervalos

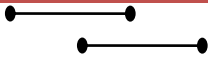
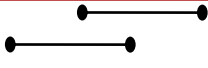
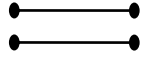
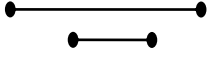
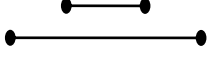
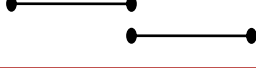



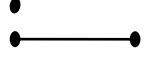
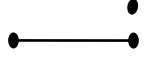
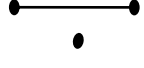
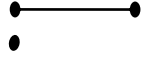
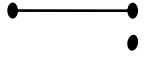
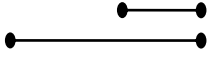
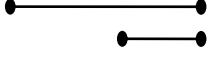
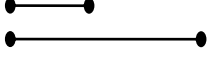
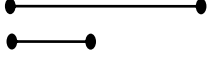
Para la realización de los casos hemos diseñado 29 casos de prueba que hemos implementado en el JUnit. Teniendo en cuenta dos grandes grupos, cuando los dos intervalos son coincidentes y cuando no lo son.

Dentro de estos casos hemos subdividido en solapamientos entre intervalos, entre intervalos y puntos, y entre puntos.

2.1.1.- Nombres de los métodos implementados



2.1.2.- Casos implementados

Coincidentes				
Nº	Nombre	Diagrama	[x1,x2) [y1,y2)	Salida
1	coincidentesIntervaloA		[-6,3) [1,7)	2
2	coincidentesIntervaloB		[5,15) [0,8)	3
3	coincidentesTotal		[4,8) [4,8)	4
4	coincidentesGrandePequeño		[2,9)[5,8)	3
5	coincidentesPequeñoGrande		[3,5)[0,12)	2
6	coincidentesExtremoA		[2,5) [5,11)	0
7	coincidentesExtremoB		[7,12)[3,7)	0
8	coincidentesPuntos		[2,2)[2,2)	1
9	coincidentesPuntoIntervaloA		[3,3)[0,10)	1
10	coincidentesPuntoIntervaloB		[5,5)[5,9)	1
11	coincidentesPuntoIntervaloC		[7,7)[3,7)	0
12	coincidentesIntervaloPuntoA		[2,15)[8,8)	1
13	coincidentesIntervaloPuntoB		[1,8)[1,1)	1
14	coincidentesIntervaloPuntoC		[3,10)[10,10)	0
15	coincidentesIntervaloInternoFinalA		[6,9)[0,9)	3
16	coincidentesIntervaloInternoFinalB		[0,8)[6,8)	2
17	coincidentesIntervaloInternoInicioA		[2,4)[2,7)	2
18	coincidentesIntervaloInternoInicioB		[1,9)[1,4)	3

No Coincidentes				
19	noCoincidentesA		[1,6][8,13)	0
20	noCoincidentesB		[9,20)[1,7)	0
21	noCoincidentesPuntosA		[2,2)[5,5)	0
22	noCoincidentesPuntosB		[7,7)[3,3)	0
23	noCoincidentesPuntoIntervaloA		[1,1)[4,8)	0
24	noCoincidentesPuntoIntervaloB		[9,9)[2,7)	0
25	noCoincidentesIntervaloPuntoA		[2,6)[7,7)	0
26	noCoincidentesIntervaloPuntoB		[3,8)[2,2)	0

Intervalos Erróneos				
27	intervalosIncorrectoCorrecto		[3,0)[4,5)	Error
28	intervalosCorrectoIncorrecto		[1,2)[5,1)	Error
29	intervalosIncorrectos		[3,1)[6,2)	Error

2.1.3.- Código del solapamiento de intervalos

Una vez creados todos los casos en JUnit nos damos cuenta de que no todos se cumplen con el código, por tanto el código tiene fallos y hay que arreglarlos. A continuación presentamos el código corregido:

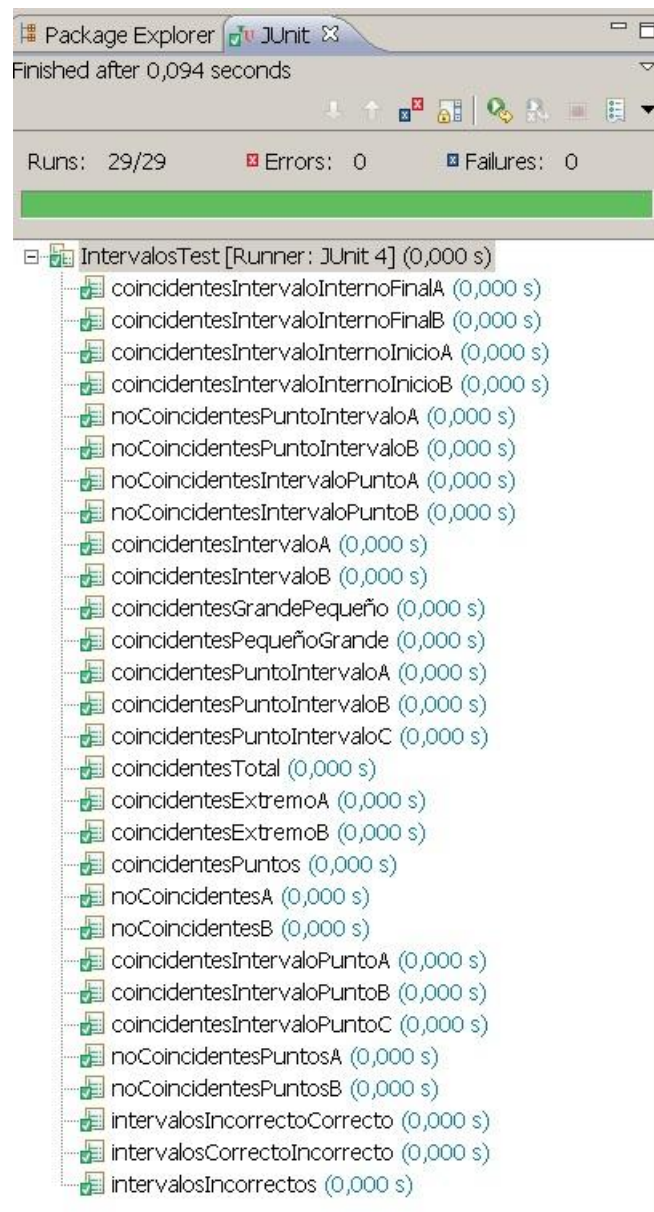
```
public static int solapamiento(int x1, int x2, int y1, int y2) {  
    if (x2 < x1 || y2 < y1) throw new IllegalArgumentException(); (*6)  
    int ret;  
    if (x1 > y1) {  
        if (x2 < y2)  
            if (x1 == x2) ret = 1; //Para tener en cuenta un caso pto-  
            intervalo (*3)  
            else  
                ret = x2 - x1;  
        else {  
            if (x1 > y2)  
                ret = 0;  
            else  
                ret = y2 - x1;  
        }  
    } else {  
        if (x2 > y2)  
            if (y2 == y1) ret = 1; //Para tener en cuenta un caso pto-  
            intervalo (*4)  
            else  
                ret = y2 - y1; //Estaba mal, antes: x2-y1 (*1)  
        else {  
            if (x2 > y1)  
                ret = x2 - y1; //Estaba mal, antes: antes:y2-y1 (*2)  
            else  
                //Para tener en cuenta un caso pto-intervalo y ptos coincidentes (*5)  
                if (x1 == x2 && x1 == y1) ret = 1;  
                else  
                    ret = 0;  
        }  
    }  
    return ret;  
}
```

(*1) y (*2) → Las soluciones estaban intercambiadas, por lo que la salida de estos casos era errónea.

(*3), (*4) y (*5) → Estos casos los hemos tenido que implementar ya que para intervalos que son un punto, es decir, un número sólo (ejemplo [2,2)), no funcionaba el código. Incluyendo estos tres casos se soluciona ese problema.

(*6) → Hemos añadido este código para que no se puedan introducir intervalos de manera incorrecta, es decir, cuando el extremo inicial del intervalo es mayor que el extremo final del intervalo.

2.1.4.- Resultados Test Intervalos



2.2.- Proyecto Rectangulador

El método “rectángulosDiferentes” tiene como parámetro “nOri” que es el número con el cuál debemos calcular los rectángulos posibles. Para la implementación del código hemos seguido los siguientes pasos:

- 1) Calculamos los divisores de nOri y los guardamos en un arrayList llamado “divisores”.
- 2) Creamos dos arrayList llamados “aux” y “aux1”. Los rectángulos vienen determinados por “nOri = X * Y”. De este modo guardamos en “aux” las “X” y en “aux1” las “Y”.
- 3) Por último el tamaño del arrayList “aux” nos determina los rectángulos diferentes que se pueden formar con “nOri”.

2.2.1.- Funcionamiento del código

El cálculo de divisores está basado en cómo se calcula los divisores de un número entero en la realidad con papel y lápiz. Además para mejorar la velocidad de búsqueda de divisores, una vez que hemos controlado todas las veces que es divisible entre 2, nos saltamos todos los pares.

120		2	divisores = {2,2,2,3,5} n = 5 = divisores.size()
60		2	
30		2	
15		3	
5		5	
1			

```
Long num = nOri;
if (num<=0) throw new IllegalArgumentException();
ArrayList<Long> divisores = new ArrayList<Long>();
long i=2; //calcula de divisores
while(num>1){
    while(num%i==0){
        num = num/i;
        divisores.add(i);
    }
    if (i==2) i--;
    i=i+2;
}
```

Para calcular todas las combinaciones posibles y no olvidarnos de ninguna, generamos un número binario con tantos bits como número de divisores tiene “nOri” (es decir n). Desechamos las combinaciones “todo ceros” y “todo unos” ya que estas dos posibilidades no nos genera ningún rectángulo. Multiplicamos cada bit por el divisor de su misma posición, éstos los multiplicamos entre si y los guardamos en una variable de tipo long llamada “mul”.

Comprobamos que el valor de “mul” no esté ya en “aux” ni en “aux1” para no repetir combinaciones. En el caso de que no esté lo guardamos en “aux” y en “aux1” (nOri/mul).

Nº binario	Divisores → {2,2,2,3,5}
00000 → No nos interesa	Combinación binaria → 1 0 0 1 1
00001 → 1	
...	
11110 → $2^n - 2$ (30 en decimal)	$(2*1)*(2*0)*(2*0)*(3*1)*(5*1) = (2*3*5) \rightarrow \text{aux}$
11111 → No nos interesa	$\text{nOri}/(2*3*5) \rightarrow \text{aux1}$

```
// calculo combinaciones de divisores
ArrayList<Long> aux = new ArrayList<Long>();
ArrayList<Long> aux1 = new ArrayList<Long>();
for(int j=1;j<(Math.pow(2,divisores.size()))-1;j++){
    String bin = Integer.toBinaryString(j);
    StringBuffer binBuffer = new StringBuffer(bin);
    bin = binBuffer.reverse().toString();
    long mul=1;
    for(int z=0; z<bin.length();z++)
        if ((int) bin.charAt(z)-48==1)
            mul = mul * divisores.get(z);

    if (aux.contains(mul)==false && aux1.contains(mul)==false){
        aux.add(mul); aux1.add(nOri/mul);
    }
}
```

Por último devolvemos el tamaño de “aux”, que serán las combinaciones diferentes de rectángulos a partir de “nOri”.

Código completo:

```
public static int rectangulosDiferentes(long nOri) {
    Long num = nOri;
    if (num<=0) throw new IllegalArgumentException();
    ArrayList<Long> divisores = new ArrayList<Long>();
    long i=2; //calcula de divisores
    while(num>1){
        while(num%i==0){
            num = num/i;
            divisores.add(i);
        }
        if (i==2) i--;
        i=i+2;
    }
    // calcula combinaciones de divisores
    ArrayList<Long> aux = new ArrayList<Long>();
    ArrayList<Long> aux1 = new ArrayList<Long>();
    for(int j=1;j<(Math.pow(2,divisores.size()))-1;j++){
        String bin = Integer.toBinaryString(j);
        StringBuffer binBuffer = new StringBuffer(bin);
        bin = binBuffer.reverse().toString();
        long mul=1;
        for(int z=0; z<bin.length();z++)
            if ((int) bin.charAt(z)-48==1)
                mul = mul * divisores.get(z);
        if (aux.contains(mul)==false && aux1.contains(mul)==false){
            aux.add(mul); aux1.add(nOri/mul);
        }
    }
    return aux.size();
}
```

2.2.2.- Resultados Test Rectangulador

