Sesión 6. Conjuntos

En esta práctica se implementará un *tipo de dato modificable*, $FiniteSet<E>$, cuyas instancias son conjuntos representados mediante vectores de bits. Se trata por tanto de conjuntos finitos de elementos de tipo entero, o bien, de otros tipos que se puedan representar mediante enteros; es decir, de *tipos escalares ordinales*: *enteros*, *caracteres*, *booleanos* o *enumerados*. Por ser conjuntos finitos, habrá un *conjunto universal* que contendrá todos posibles elementos que puedan contener las instancias del tipo $FiniteSet<E>$ (conjuntos); es decir, un rango de valores de tipo E (donde E será un tipo escalar ordinal).

Crea un proyecto Java de nombre `sesión-06`, descarga el material que se proporciona para la práctica y copia los archivos al *packages estDatos*.

Material proporcionado para la práctica:

1. El archivo $Range.java$ que proporciona el *tipo de dato no modificable e iterable* $Range<E>$, cuyas instancias son rangos de *tipos escalares ordinales*. Un rango será un *conjunto universal*.
2. El archivo $FiniteSet.java$ que proporciona un esqueleto de la implementación a realizar.

En particular, en el tipo $Range<E>$, además del constructor, cabe destacar las dos operaciones siguientes:

a. $eToInt(e)$: Retorna la posición del elemento e en el rango, correspondiendo la posición 0 al primer elemento de éste.
b. $intToE(n)$: Retorna el valor del rango que ocupa la posición n especificada.

Ejemplo: Para el rango de caracteres $r = [‘a’,‘z’]$, $r.eToInt(’c’) = 2$ (posición del carácter ‘c’ en r) y $r.intToE(1) = ’b’$ (carácter que ocupa la posición 1 en r).

Apartado 1

Completar el tipo de dato modificable $FiniteSet<E>$. Las instancias de este tipo son conjuntos finitos cuyos elementos son valores de un rango r dado y que será el *conjunto universal*. Para facilitar su desarrollo esta clase extiende la clase abstracta $AbstractSet<E>$.

Un conjunto de este tipo se representará mediante una *secuencia de* $r.size()$ *booleanos* (tamaño del rango r asociado o *conjunto universal*). La secuencia representa el conjunto tal y como se vio en la teoría para los *conjuntos basados en vectores de bits*: si un valor e del rango r pertenece al conjunto, entonces el booleano de la secuencia que está en la posición $r.eToInt(e)$ será $true$ y $false$ en caso contrario. *Por ejemplo*:

x	$r.eToInt(x)$	$x \in c$
‘d’	0	$false$
‘e’	1	$true$
‘f’	2	$false$
‘g’	3	$false$
‘h’	4	$true$
‘i’	5	$false$

Para el rango $r = [’d’,’i’]$ ($r.size() = 6$), el conjunto $c = \{’e’,’h’\}$ de rango asociado r estaría representado mediante la siguiente secuencia de seis booleanos: $false\ true\ false\ false\ true\ false$. Esta secuencia se obtiene como se muestra en la tabla adjunta, indicando si cada uno de los elementos del rango r (*conjunto universal*) pertenece o no al conjunto c, en el mismo orden que establece el rango.

Para implementar el iterador de la clase se deberá tener en cuenta lo siguiente:

Para obtener los elementos de un conjunto cualquiera, el método $next()$ del iterador siempre debe conocer (incluso al inicio) cuál es la siguiente posición de la secuencia que es $true$ (información o dato requerido que denotaremos como $current$), mientras que los valores de la secuencia que son $false$ se deben ignorar porque se corresponden con elementos del rango asociado que no pertenecen al conjunto.

Por otra parte, para la operación $remove()$ se necesita conocer la posición en la secuencia del $true$ correspondiente al último elemento retornado por la operación $next()$. Esta información no es conocida salvo que se guarde específicamente, obsérvese que $current$ no es esta posición, si no la del siguiente $true$ de la secuencia (excepto si la iteración ha terminado). Además, entre las posiciones de ambos $true$ no existe relación alguna, ya que están separados por un número indeterminado de valores $false$.

Iteración para el ejemplo del documento, rango $r = ['d', 'i']$ y el conjunto $c = \{'e', 'h'\}$. El método $next()$ se invocará dos veces (marcadas en la tabla):

Repres.	$next()$	
	current	*retorna*
$false$	1 (al inicio)	
$true$	4	$'e'$
$false$		
$false$		
$true$	6 (final)	$'h'$
$false$		

Apartado 2

1. Escribir un programa en un *package* independiente (app) que haga lo siguiente:
    a. Cree el rango ri de enteros: $[-40, 60]$
    b. Cree el conjunto $c0$ con todos los múltiplos de 7 del rango ri, utilizando el iterador para obtener sus valores
    c. Quite del conjunto $c0$ los elementos: 0, 7 y 14
    d. Cree el rango rc de caracteres: [‘a’, ‘z’]
    e. Cree los conjuntos $c1 = \{$‘c’, ‘k’, ‘r’, ‘x’$\}$ y $c2 = \{$‘b‘, ‘f’, ‘k’, ‘r’, ‘z’$\}$ cuyos elementos son valores del rango rc.
    f. Cree el conjunto $c3$ copia de $c1$
    g. Muestre en consola los conjuntos: $c0$, $c1$, $c2$ y $c3$, junto con sus tamaños y el resultado de las comparaciones de igualdad ¿$c3 = c1$? y ¿$c3 = c2$?, tal y como se muestra al final en la *salida esperada*.
    h. Cree tres conjuntos de nombres: $union$, $interseccion$ y $diferencia$, de forma que: $union = c1 \cup c2$, $interseccion = c1 \cap c2$ y $diferencia = c1 - c2$
    i. Muestre en consola los conjuntos $union$, $interseccion$ y $diferencia$, así como el resultado de las comprobaciones de inclusión: ¿$c1 \subset c2$? y ¿$c1 \subset c3$?, ?, tal y como se muestra en *salida esperada*.

Salida esperada

```
Conjunto c0: [-35, -28, -21, -14, -7, 21, 28, 35, 42, 49, 56] (11 elementos)
Conjunto c1: [c, k, r, x] (4 elementos)
Conjunto c2: [b, f, k, r, z] (5 elementos)
Conjunto c3: [c, k, r, x] (4 elementos)
¿c3 = c1? true
¿c3 = c2? false
Conjunto union       : [b, c, f, k, r, x, z]
Conjunto interseccion: [k, r]
Conjunto diferencia  : [c, x]
¿c1 subconjunto de c2? false
¿c1 subconjunto de c3? true
```