Práctica de iteraciones

Vale la pena codificar una clase `Mod3Lecc2` para probar los métodos que estarás usando para practicar. La primera línea sería simplemente

```
1 /* Pruebas de iteraciones */
2 public class Mod3Lecc2 {
```

No lo pones en el paquete `discos` porque no es realmente parte del proyecto. También te aseguras que sea el único en el IDE de DrJava para que no trate de compilar algún otro que esté todavía sin terminar, como `Catalogo.java`. Procedes ahora a practicar con el enunciado `for`.

Por ejemplo, si quieres un método que imprima en una lista los primeros `n` múltiplos de 3, podrías hacerlo de la siguiente forma usando un `for`:

```
4   public static void mult3 (int n) {
5      for ( int i = 1; // inicializacion
6            i <= n;    // expresion booleana
7            i ++)      // actualizacion
8         System.out.println( i * 3 ); // bloque
9   }
```

Si este método es invocado como `mult3( 5 )`, obtendrás 5 renglones, cada uno con un múltiplo de 3, en orden desde 3 hasta 15.

```
3
6
9
12
15
```

Cambiando ligeramente el método para que escriba en un único renglón, separando los múltiplos con un tabulador, codificas un método `mult3B`:

```
11  public static void mult3B (int n) {
12    for ( int i = 1; // inicializacion
13          i <= n;    // expresion booleana
14          i ++)      // actualizacion
15      System.out.print( i * 3 + "\t");  // bloque
16    /*  ... continua ... no esta ya en el for */
17    System.out.println();
18  }
```

El único enunciado dentro de la iteración es el de la línea 15, que escribe los múltiplos en una misma línea, seguidos cada uno por un tabulador para separarlos. Una vez terminada la iteración se ejecuta la impresión de un cambio de línea (línea 17).

```
3   6   9   12   15
```

Como estás trabajando en un ambiente integrado para Java, la sangría de los renglones te indican su nivel de anidamiento o a qué bloque pertenecen. En el segundo ejemplo la impresión del cambio de línea está fuera del rango del `for`.

Para la tercer versión del `for` tienes varias formas de organizarlo. En el ejemplo, la inicialización `int i = 1` declara una variable **local** al bloque del `for` y la inicializa. La variable `i` no va a ser

conocida fuera del ⟨bloque⟩ del for. En ocasiones quieres usar el valor de la variable más allá del final de la ejecución del for, por lo que la declaración de la variable debe estar fuera de la inicialización:

```
1  public static void mult3C (int n) {
2    /* Tercera version del for */
3    int i;
4    for (i = 1; i <= n ; i ++)
5      System.out.print ( i * 3 + "\t");
6    System.out.println( "\ni=" + i );
7  }
```

La impresión de esta ejecución con el uso del valor de i fuera del bloque es la siguiente:

```
3    6    9    12   15
i=6
```

i queda valiendo 6 porque este es el primer valor para el cual la expresión booleana deja de cumplirse cuando incrementas a la i de 1 en 1 mientras sea menor o igual a n, que en este caso es 5.

Aunque en teoría podrías tener una variable i declarada dentro del bloque del for y otra afuera (la del for "tapa" a la local), el compilador, una vez que se declara la i fuera del for, no va a permitir una declaración dentro del for de i. Puedes sacar la inicialización del encabezado del for y simplemente colocar un punto y coma (;); también puedes quitar del encabezado la actualización y moverla al final del bloque del for. En el método mult3D ejemplificamos esta implementación. Están en un cuadro rojo aquellas partes del encabezado del for que se movieron:

```
26 public static void mult3D ( int n ) {
27   int i = 1;
28   for ( ; i <= n ; ) {
29     System.out.print ( i * 3 + "\t");
30     i++;
31   }
32   System.out.println( "\ni=␣" + i );
33 }
```

La inicialización está vacía ya que cuando se llega al for ya está hecha, afuera, la declaración de un variable y su inicialización. Si la ⟨expr bool⟩ se encuentra vacía, se sustituye por true, lo que quiere decir que se repita por siempre o hasta que se salga del ⟨bloque⟩ de alguna otra manera que no sea porque deje de cumplirse la expresión booleana. Como no has visto condicionales, todavía no vamos a dejar vacía la expresión booleana, pues el compilador detecta un error de lógica y no lo permite.

Supongamos que queremos obtener el promedio final de un estudiante que se calcula a partir de calificaciones de distintas actividades (tareas, exámenes, participación, entre otros) y cada una de estas actividades aporta a la calificación final un cierto porcentaje, que lo pensamos como sigue:

	0	1	2	3			
0	8.75	10.00	4.25	6.67	...	...	calificaciones
1	0.25	0.15	0.30	0.20	...	...	porcentajes (deben sumar 1.00)

Supone que declaras el arreglo como

```
double[ ][ ] califs = { {8.75, 10.00, 4.25, 6.67, 8.3}, // calificaciones
                        {0.25, 0.15, 0.30, 0.20, 0.1} // porcentajes
                      };
```

Para obtener la calificación final basta sumar el producto de `califs[0][k] * califs[1][k]`, para $k = 0, \ldots, 4$. Supone que tienes un método estático que hace esto. El encabezado es como sigue:

```
public static double promedio( double[ ][ ] nums ) {
```

Usarás la iteración `for` para obtener la calificación final. Como vas a sumar, dentro del `for`, a cada uno de los productos, necesitas ir acumulando cada producto en una variable que inicie en 0.00. También tienes que saber el número de elementos en cada renglón del arreglo, que esperas sea el mismo en los dos renglones:

```
double prom = 0; // acumulador
int tam     = nums[0].length; // para usar en el for
for (int k = 0 ; k < tam; k ++) { // los arreglos empiezan en 0
  prom += nums[0][k] * nums[1][k]; // renglon 0 y 1
}
return prom;
```

Aunque únicamente tienes un enunciado en el bloque del `for`, es conveniente ponerlo en un bloque (entre llaves) por si después quieres agregar algo más o quieres vigilar el curso de la iteración imprimiendo alguno de los valores. Esto es usual si la iteración no está funcionando como quieres.

Puedes invocar a este método (desde `main` o cualquier otro método de la clase), con un arreglo "anónimo" e imprimiendo su resultado al momento de invocarlo, de la siguiente manera:

```
System.out.println("Promedio="
                   + promedio(new double[ ][ ] { // anonimo
  {8.75, 10.00, 4.25, 6.67, 8.3}, // calificaciones
  {0.25, 0.15, 0.30, 0.20, 0.1} // porcentajes
}));  // arreglo anonimo
```

Que un arreglo sea "anónimo" quiere decir que no hay un identificador asociado a él. Se construye como cualquier otro arreglo. Los arreglos anónimos sólo tienen sentido como argumentos para métodos, ya que dentro del método el arreglo tiene asociado el identificador del parámetro correspondiente. El resultado de esta invocación es el siguiente:

```
Promedio= 7.1265
```

Anotaciones sobre arreglos

Tienes ya todo lo que necesitas para programar el tercer constructor, pero antes de hacerlo haremos énfasis en algunos aspectos de los arreglos que fungen como parámetros formales o argumentos.

(a) Cuando se desea un arreglo como **parámetro formal**, se debe indicar el tipo de sus elementos, el número de dimensiones colocando tantas parejas de corchetes como dimensiones tenga el arreglo que se va a manejar dentro del método y el identificador con el que se asocia el arreglo dentro del método.

(b) En un arreglo que es parámetro formal el tamaño de cada dimensión está dado por el argumento con el que se invoca; no se puede dar en la declaración del parámetros formal.

(c) Cuando se desea pasar un arreglo como **argumento** (en la llamada de un método) se usa el identificador de un arreglo del mismo número de dimensiones que el que aparece como parámetro formal.

(d) Si el método tiene como parámetro formal un arreglo con k dimensiones, se le puede pasar como argumento uno de los arreglos de un arreglo de $k + 1$ dimensiones. En el ejemplo que vimos respecto a las calificaciones, podemos suponer que tenemos una tabla por cada alumno de un grupo:

```
public static void main (String[ ] args) {
  double [ ][ ][ ] alumnos;
    /* Se llenan las tablas con los valores correspondientes
       a cada alumno */
  for (int i = 0; i < alumnos.length; // da el numero de alumnos
       i++) {
    System.out.println("Promedio del alumno[" + i + "]: "
                     + promedio ( alumnos[i] ));
  }
```

En este caso el método `promedio` espera un arreglo de dos dimensiones y le pasas cada uno de los arreglos de dos dimensiones que tiene el arreglo de tres dimensiones.

(e) Como **argumento** se puede pasar un arreglo anónimo que se construya en el momento de invocar al método. Como en cualquier caso, tienen que coincidir el tipo del arreglo y el número de dimensiones que espera el método.

Con estas anotaciones procedes a resolver el problema que nos trajo a este ejercicio.