MANEJO DE GRUPOS DE ATRIBUTOS

03

1. Objetivos Pedagógicos

Al final de este nivel el lector será capaz de:

- Utilizar las estructuras contenedoras de tamaño fijo como elementos para modelar una característica de un elemento del mundo que permiten almacenar una secuencia de valores (simples u objetos).
- Utilizar las estructuras contenedoras de tamaño variable como elementos de modelado que permiten manejar atributos cuyo valor es una secuencia de objetos.
- Utilizar las instrucciones iterativas para manipular estructuras contenedoras y entender que dichas instrucciones se pueden utilizar en otro tipo de problemas.
- Crear una clase completa en Java utilizando el ambiente de desarrollo Eclipse.
- Entender la documentación de un conjunto de clases escritas por otros y utilizar dicha documentación para poder incorporar y usar adecuadamente dichas clases en un programa que se está construyendo.

2. Motivación

Cuando nos enfrentamos a la construcción del modelo conceptual del mundo del problema, en muchas ocasiones nos encontramos con el concepto de colección o grupo de cosas de la misma clase. Por ejemplo, si retomamos el caso de estudio del empleado presentado en el nivel 1 y lo generalizamos a la administración de todos los empleados de la universidad, es claro que en alguna parte del diagrama de clases debe aparecer el concepto de grupo de empleados. Además, cuando planteemos la solución, tendremos que definir un método en alguna clase para añadir un nuevo elemento a ese grupo (ingresó un nuevo empleado a la universidad) o un método para buscar un empleado de la universidad (por ejemplo, quién es el empleado que tiene mayor salario). De manera similar, si retomamos el caso de estudio del nivel 2 sobre la tienda, lo natural es que una tienda manipule un número arbitrario de productos, y no sólo cuatro de ellos como se definió en el ejemplo. En ese caso, la tienda debe poder agregar un nuevo producto al grupo de los que ya vende, buscar un producto en su catálogo, etc.

En este capítulo vamos a introducir dos conceptos fundamentales de la programación:

1. Las estructuras contenedoras, que nos permiten manejar atributos cuyo valor corresponde a una secuencia de elementos.
2. Las instrucciones repetitivas, que son instrucciones que nos permiten manipular los elementos contenidos en dichas secuencias.

Además, en este nivel estudiaremos la manera de crear objetos y agregarlos a una contenedora, la manera de crear una clase completa en Java y la forma de leer la descripción de un conjunto de clases desarrolladas por otros, para ser capaces de utilizarlas en nuestros programas.

Vamos a trabajar sobre varios casos de estudio que iremos introduciendo a lo largo del nivel.

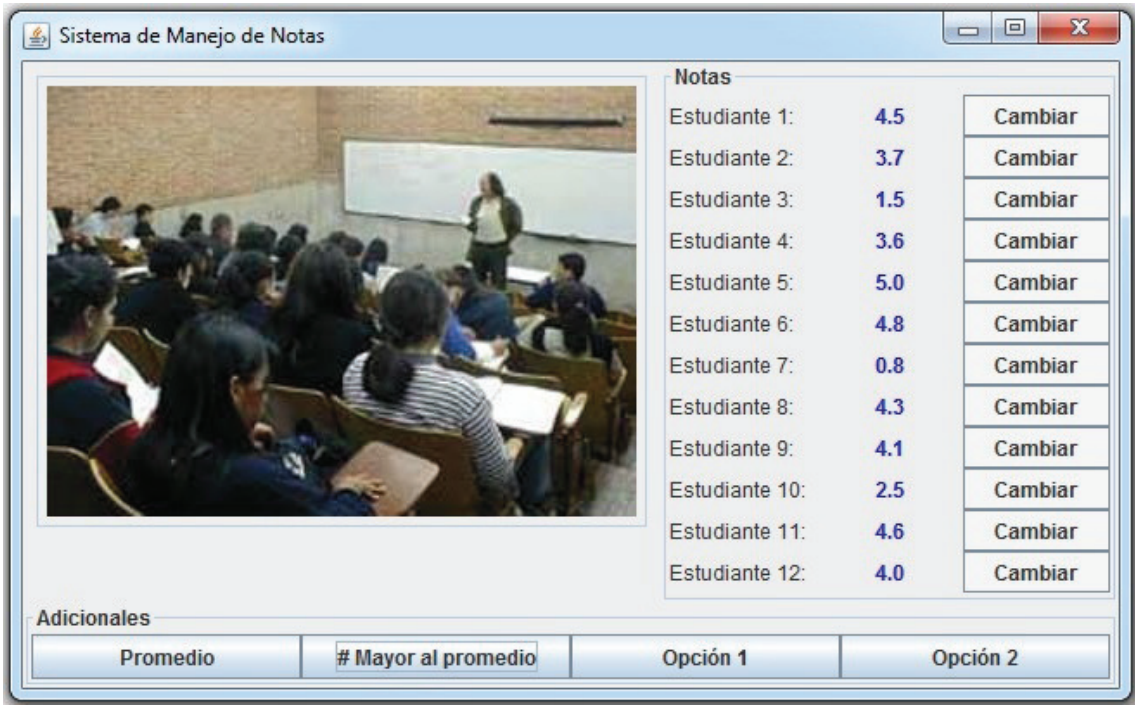
3. Caso de Estudio Nº 1: Las Notas de un Curso

Considere el problema de administrar las calificaciones de los alumnos de un curso, en el cual hay doce estudiantes, de cada uno de los cuales se tiene la nota definitiva que obtuvo (un valor entre 0,0 y 5,0). Se quiere construir un programa que permita:

1. Cambiar la nota de un estudiante.
2. Calcular el promedio del curso.
3. Establecer el número de estudiantes que está por encima de dicho promedio.

En la figura 3.1 aparece la interfaz de usuario que se quiere que tenga el programa.

Fig. 3.1 Interfaz de usuario del programa del primer caso de estudio



- En la ventana del programa aparece la nota de cada uno de los doce estudiantes del curso. La nota con la que comienzan es siempre cero.
- Con el respectivo botón es posible modificar la nota. Al oprimirlo, aparece una ventana de diálogo en la que se pide la nueva nota.
- En la parte de abajo de la ventana se encuentran los botones que implementan los requerimientos funcionales: calcular el promedio e indicar el número de estudiantes que están por encima de dicha nota.

3.1. Comprensión de los Requerimientos

Requerimiento funcional 1

Nombre	R1 – Cambiar una nota.
Resumen	Permite cambiar la nota definitiva que tiene asignado un estudiante del curso.
Entradas	(1) El estudiante a quien se le quiere cambiar la nota. (2) La nueva nota del estudiante.
Resultado	Se le ha asignado al estudiante la nueva nota.

Requerimiento funcional 2

Nombre	R2 – Calcular el promedio.
Resumen	Se quiere calcular el promedio del curso, utilizando la nota que tiene cada estudiante.
Entradas	Ninguna.
Resultado	Promedio de las notas de los doce estudiantes del curso.

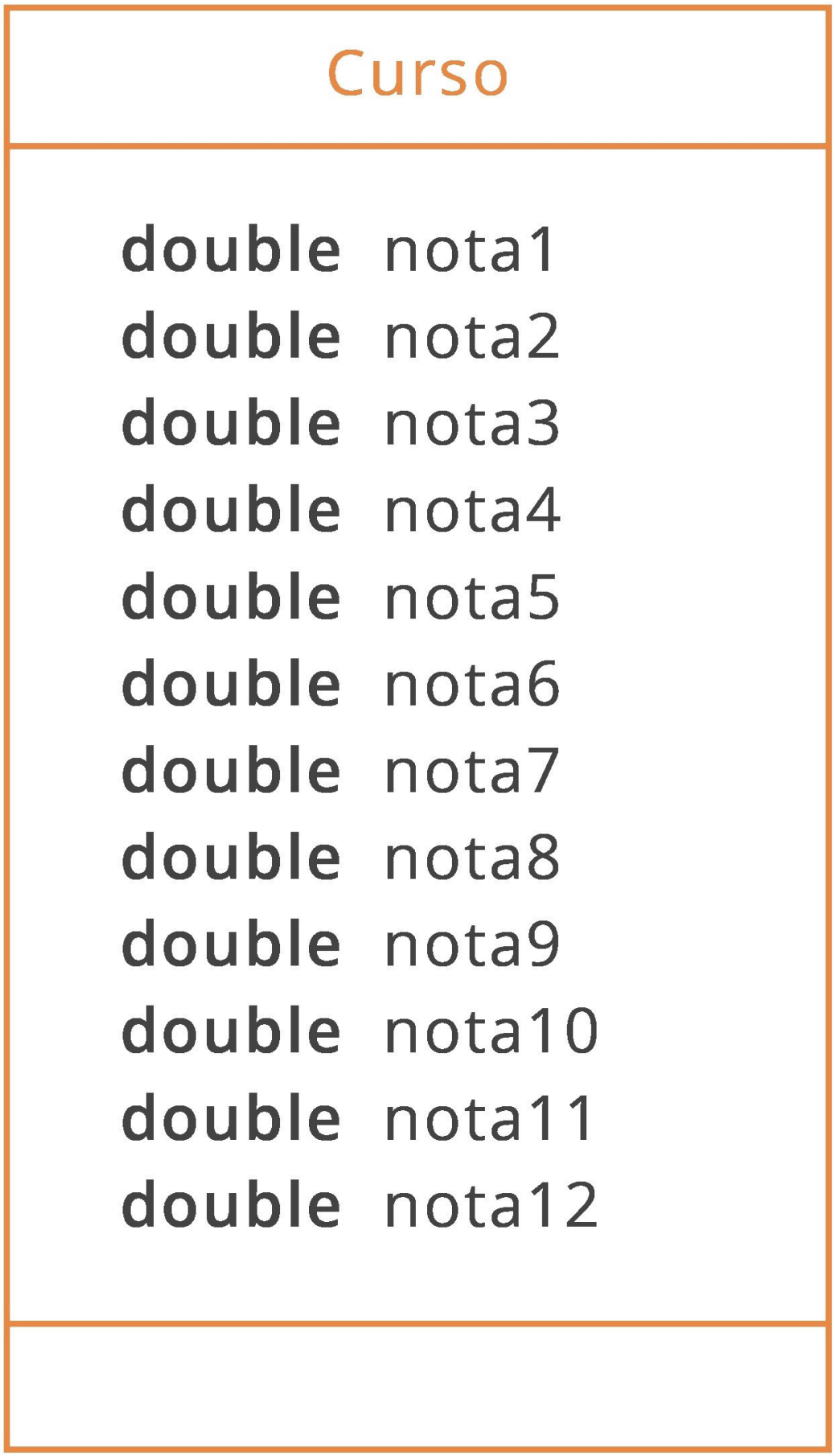
Requerimiento funcional 3

Nombre	R3 – Calcular el número de estudiantes por encima del promedio.
Resumen	Se quiere saber cuántos estudiantes tienen una nota superior a la nota promedio del curso.
Entradas	Ninguna.
Resultado	Número de estudiantes con nota mayor al promedio del curso.

3.2. Comprensión del Mundo del Problema

Dado el enunciado del problema, el modelo conceptual se puede definir con una clase llamada Curso, la cual tendría doce atributos de tipo double para representar las notas de cada uno de los estudiantes, tal como se muestra en la figura 3.2.

Fig. 3.2 Modelo conceptual de las calificaciones de los estudiantes

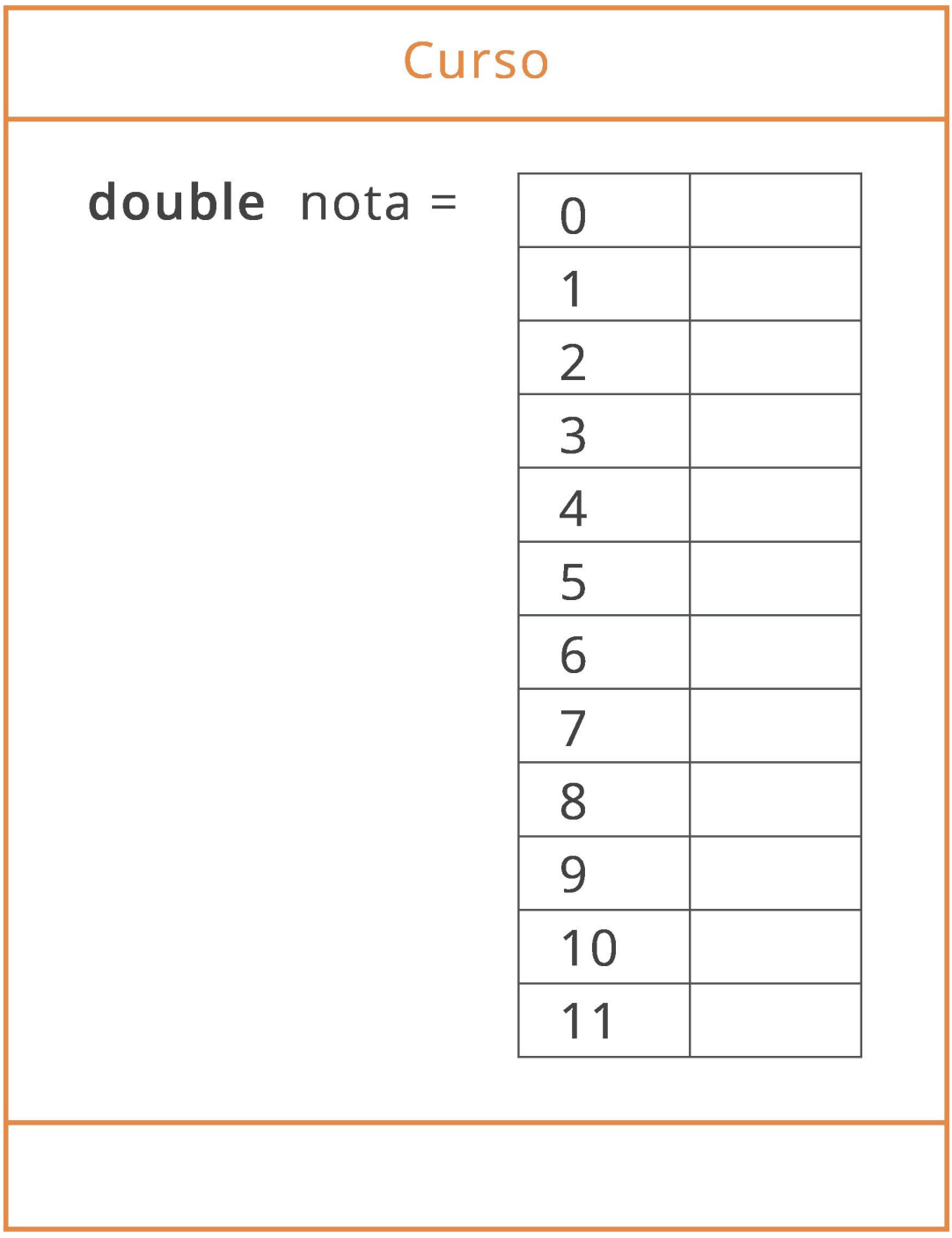

Curso
double nota1
double nota2
double nota3
double nota4
double nota5
double nota6
double nota7
double nota8
double nota9
double nota10
double nota11
double nota12

Aunque este modelado es correcto, los métodos necesarios para resolver el problema resultarían excesivamente largos y dispendiosos. Cada expresión aritmética para calcular cualquier valor del curso tomaría muchas líneas de código. Además, imagine si en vez de 12 notas tuviéramos que manejar 50 ó 100. Terminaríamos con algoritmos imposibles de leer y de mantener. Necesitamos una manera mejor de hacer este modelado y ésta es la motivación de introducir el concepto de estructura contenedora.

4. Contenedoras de Tamaño Fijo

Lo ideal, en el caso de estudio, sería tener un sólo atributo (llamado por ejemplo notas), en donde pudiéramos referirnos a uno de los valores individuales por un número que corresponda a su posición en el grupo (por ejemplo, la quinta nota). Ese tipo de atributos que son capaces de agrupar una secuencia de valores se denominan contenedoras y la idea se ilustra en la figura 3.3). Vale la pena aclarar que la sintaxis usada en la figura no corresponde a la sintaxis de UML, sino que solamente la usamos para ilustrar la idea de una estructura contenedora.

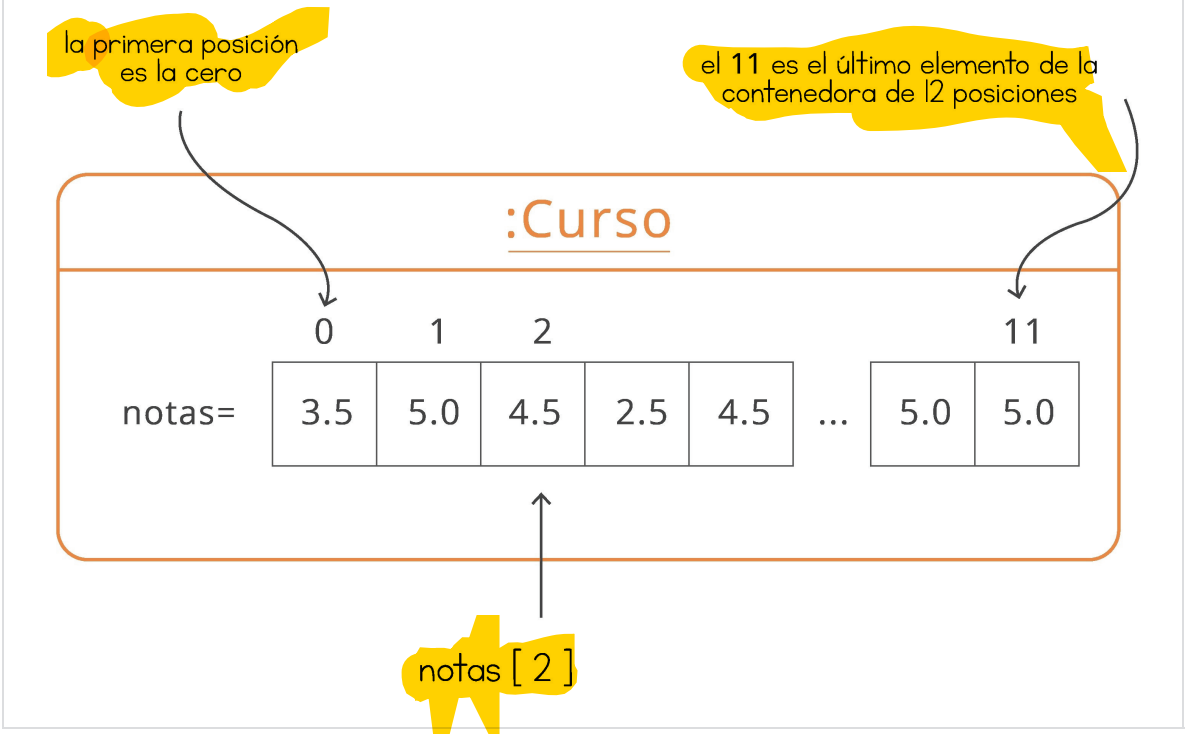
Fig. 3.3 Modelo conceptual de las calificaciones con una contenedora



- En lugar de tener 12 atributos de tipo real, vamos a tener un sólo atributo llamado "notas" el cual contendrá en su interior las 12 notas que queremos representar.
- Cada uno de los elementos del atributo "notas" se puede referenciar utilizando la sintaxis notas[x], donde x es el número del estudiante a quien corresponde la nota (comenzando en 0).
- Con esta representación podemos manejar de manera más simple y general el grupo de notas de los estudiantes.

Un objeto de la clase Curso se vería como aparece en la figura 3.4. Allí se puede apreciar que las posiciones dentro de una contenedora se comienzan a numerar a partir del valor 0 y que los elementos individuales se referencian a través de su posición. Cada nota va en una posición distinta de la contenedora de tipo `double` llamada notas.

Fig. 3.4 – Representación gráfica de un arreglo



En las secciones que siguen veremos la manera de declarar (en UML y en Java) un atributo que corresponda a una contenedora, lo mismo que a manipular los valores allí incluidos.

4.1 Declaración de un Arreglo

En Java, las estructuras contenedoras de tamaño fijo se denominan arreglos (arrays en inglés), y se declaran como se muestra en el ejemplo 1. Los arreglos se utilizan para modelar una característica de una clase que corresponde a un grupo de elementos, de los cuales se conoce su número. Si no supiéramos, por ejemplo, el número de estudiantes del curso en el caso de estudio, deberíamos utilizar una contenedora de tamaño variable, que es el tema de una sección posterior de este capítulo.

Ejemplo 1

Objetivo: Mostrar la sintaxis usada en Java para declarar un arreglo.

En este ejemplo se hace la declaración del arreglo de notas, como parte de la clase Curso del caso de estudio.

```
public class Curso
{
  //-----------------------------------
  // Constantes
  //-----------------------------------
  public final static int TOTAL_EST = 12;

  //-----------------------------------
  // Atributos
  //-----------------------------------
  private double[] notas;
  ...
}
```

- Es conveniente declarar el número de posiciones del arreglo como una constante (`TOTAL_EST`). Eso facilita realizar más tarde modificaciones al programa. Si en vez de 12 hay que manejar 15 estudiantes, bastaría con cambiar dicho valor.
- En el momento de declarar el atributo " `notas` ", usamos la sintaxis " `[]` " para indicar que va a contener un grupo de valores.
- El tamaño del arreglo será determinado en el momento de la inicialización del arreglo, en el método constructor. Por ahora no hay que decir nada al respecto.
- En la declaración le decimos al compilador que todos los elementos del arreglo son de tipo `double` .
- Recuerde que los elementos de un arreglo se comienzan a referenciar a partir de la posición 0.

4.2 Inicialización de un Arreglo

Al igual que con cualquier otro atributo de una clase, es necesario inicializar los arreglos en el método constructor antes de poderlos utilizar. Para hacerlo se debe definir el tamaño del arreglo, o sea el número de elementos que va a contener. Esta inicialización es obligatoria, puesto que es en ese momento que le decimos al computador cuántos valores debe manejar en el arreglo, lo que corresponde al espacio en memoria que debe reservar. Veamos en el ejemplo 2 cómo se hace esto para el caso de estudio.

> Si tratamos de acceder a un elemento de un arreglo que no ha sido inicializado, vamos a obtener el error de ejecución: *java.lang.NullPointerException*

Ejemplo 2

Objetivo: Mostrar la manera de inicializar un arreglo en Java.

En este ejemplo mostramos, en el contexto del caso de estudio, la manera de inicializar el arreglo de notas dentro del constructor de la clase Curso.

```
public Curso( )
{
  notas = new double[ TOTAL_EST ] ;
}
```

- Se utiliza la instrucción `new` como con cualquier otro objeto, pero se le especifica el número de valores que debe contener el arreglo (TOTAL_EST, que es una constante de valor 12).
- Esta construcción reserva el espacio para el arreglo, pero el valor de cada uno de los elementos del arreglo sigue siendo indefinido. Esto lo arreglaremos más adelante.

El lenguaje Java provee un operador especial (`length`) para los arreglos, que permite consultar el número de elementos que éstos contienen. En el caso de estudio, la expresión `notas.length` debe dar el valor 12, independientemente de si los valores individuales ya han sido o no inicializados, puesto que en el método constructor de la clase se reservó dicho espacio de memoria.

4.3. Acceso a los Elementos del Arreglo

Un índice es un valor entero que nos sirve para indicar la posición de un elemento en un arreglo. Los índices van desde 0 hasta el número de elementos menos 1. En el caso de estudio la primera nota tiene el índice 0 y la última, el índice 11. Para tomar o modificar el valor de un elemento particular de un arreglo necesitamos dar su índice, usando la sintaxis que aparece en el siguiente método de la clase Curso y que, en el caso general, se puede resumir como `<arreglo>[<índice>]` .

```
public void noHaceNadaUtil( double valor )
{
  int indice = 10;
  notas[ 0 ] = 3.5;
  if( valor < 2.5 && notas.length == TOTAL_EST )
  {
    notas[ indice ] = notas[ 0 ];
    notas[ 0 ] = valor + 1.0;
  }
  else
  {
    notas[ indice ] = notas[ 0 ] - valor;
  }
}
```

- Este método sólo lo utilizamos para ilustrar la sintaxis que se utiliza en Java para manipular los elementos de un arreglo.
- Para asignar un valor a una casilla del arreglo, usamos la sintaxis `notas[ x ] = valor` , donde x es el índice que nos indica una posición.
- Para obtener el valor de una casilla, usamos la misma sintaxis (`notas[ x ]`). `notas.length` nos da el número de casillas del arreglo.

De esta manera podemos asignar cualquier valor de tipo `double` a cualquiera de las casillas del arreglo, o tomar el valor que allí se encuentra.

> Cuando dentro de un método tratamos de acceder una casilla con un índice no válido (menor que 0 o mayor o igual que el número de casillas), obtenemos el error de ejecución: *java.lang.ArrayIndexOutOfBoundsException*

Es importante destacar que, hasta este momento, lo único que hemos ganado con la introducción de los arreglos es no tener que usar atributos individuales para representar una característica que incluye un grupo de elementos. Es más cómodo tener un sólo atributo con todos esos elementos en su interior. Las verdaderas ventajas de usar arreglos las veremos a continuación, al introducir las instrucciones repetitivas.

5. Instrucciones Repetitivas

5.1. Introducción

En muchos problemas notamos una regularidad que sugiere que su solución puede lograrse repitiendo un paso que vaya transformando gradualmente el estado del mundo modelado y acercándose a la solución. Instintivamente es lo que hacemos cuando subimos unas escaleras: repetimos el paso de subir un escalón hasta que llegamos al final. Otro ejemplo posible es si suponemos que tenemos en una hoja de papel una lista de palabras sin ningún orden y nos piden buscar si la palabra "casa" está en la lista. El algoritmo que seguimos para realizar está tarea puede ser descrito de la siguiente manera:

1. Verifique si la primera palabra es igual a "casa".
2. Si lo es, no busque más. Si no lo es, busque la segunda palabra.
3. Verifique si la segunda palabra es igual a "casa".
4. Si lo es, no busque más. Si no lo es, busque la tercera palabra.
5. Repita el procedimiento palabra por palabra, hasta que la encuentre o hasta que no haya más palabras para buscar.

Tarea 1

Objetivo: Explicar el significado de la instrucción repetitiva y usarla para definir un algoritmo que resuelva un problema simple.

Suponga que en el ejemplo anterior, ya no queremos buscar una palabra sino contar el número total de letras que hay en todas las palabras de la hoja.

Escriba el algoritmo para resolver el problema:

5.2. Calcular el Promedio de las Notas

Para resolver el segundo requerimiento del caso de estudio (R2 - calcular el promedio de las notas), debemos calcular la suma de todas las notas del curso para luego dividirlo por el número de estudiantes. Esto se puede hacer con el método que se muestra a continuación:

```
public double promedio( )
{
  double suma = notas[ 0 ] + notas[ 1 ] + notas[ 2 ] +
                notas[ 3 ] + notas[ 4 ] + notas[ 5 ] +
                notas[ 6 ] + notas[ 7 ] + notas[ 8 ] +
                notas[ 9 ] + notas[ 10 ] + notas[ 11 ];
  return suma / TOTAL_EST;
}
```

- Primero sumamos las notas de todos los estudiantes y guardamos el valor en la variable suma.
- El promedio corresponde a dividir dicho valor por el número de estudiantes, representado con la constante `TOTAL_EST`.

Si planteamos el problema de manera iterativa, podemos escribir el mismo método de la siguiente manera, en la cual, en cada paso, acumulamos el valor del siguiente elemento:

```
public double promedio( )
{

  double suma = 0.0;
  int indice = 0;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  indice++;
  suma += notas[ indice ];
  return suma / TOTAL_EST;
}
```

- Esta solución también calcula el promedio del curso, pero en lugar de hacer referencia directa a las doce casillas del arreglo, utiliza un índice que va desplazando desde 0 hasta 11.
- Por supuesto que es más clara la solución anterior, pero queremos utilizar este ejemplo para introducir las instrucciones iterativas, que expresan esta misma idea de "desplazar" un índice, pero usando una sintaxis mucho más compacta.
- Lo primero que debemos notar es que vamos a ejecutar 12 veces (`TOTAL_EST` veces para ser exactos) un grupo de instrucciones.
- Ese grupo de instrucciones es: `suma += notas[ indice ]; indice++` ;
- Después de ejecutar 12 veces esas dos instrucciones, en la variable suma tendremos el valor total, listo para dividirlo por el número de estudiantes.
- El índice comienza teniendo el valor 0 y termina teniendo el valor 11. De esta manera, cada vez que hacemos referencia al elemento `notas[indice]` , estamos hablando de una casilla distinta del arreglo.

Allí repetimos 12 veces una pareja de instrucciones, una vez por cada elemento del arreglo. Basta un poco de reflexión para ver que lo que necesitamos es poder decir que esas dos instrucciones se deben repetir tantas veces como notas haya en el arreglo. Las instrucciones repetitivas nos permiten hacer eso de manera sencilla. En el siguiente método se ilustra el uso de la instrucción `while` para el mismo problema del cálculo del promedio.

```
public double promedio( )
{
  double suma = 0.0;
  int indice = 0;
  while( indice < TOTAL_EST )
  {
    suma += notas[ indice ]; indice++;
  }
  return suma / TOTAL_EST;
}
```

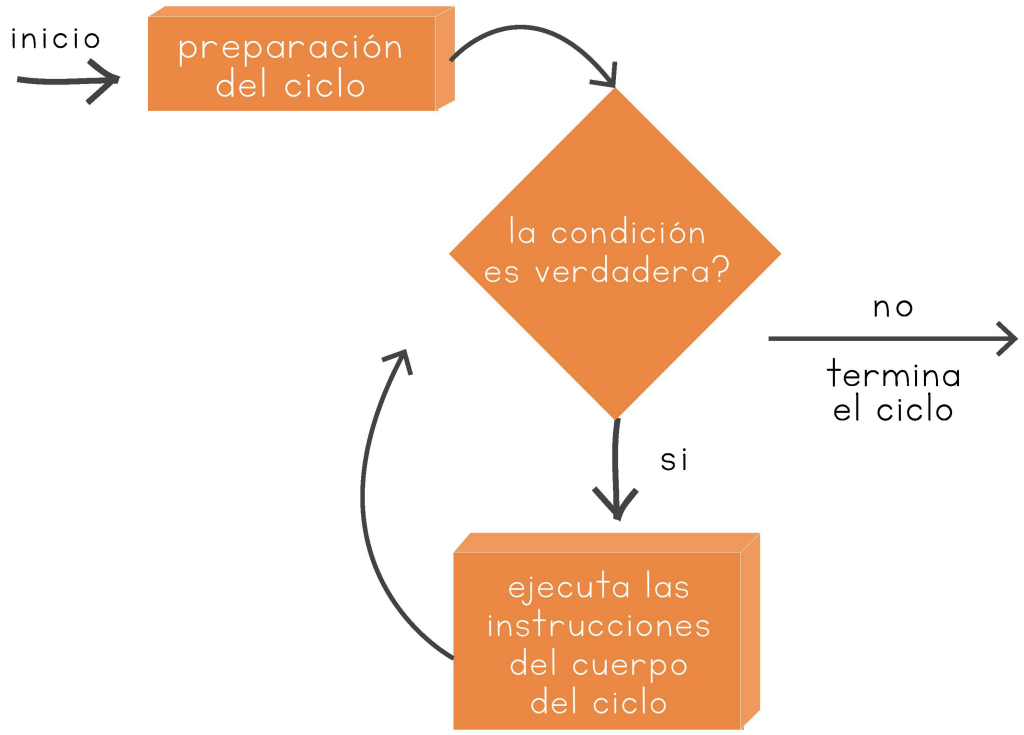
- La estructura del método sigue siendo la misma, con la única diferencia de que en lugar de repetir 12 veces la pareja de instrucciones, las incluimos dentro de la instrucción `while`, que se encarga de ejecutar repetidamente las instrucciones que tiene en su interior.
- La instrucción `while` sirve para decirle al computador que "mientras que" una condición se cumpla, siga ejecutando las instrucciones que están por dentro.
- La condición en el ejemplo es `indice < TOTAL_EST`, que equivale a decirle que "mientras que" el índice no llegue a 12, vuelva a ejecutar la pareja de instrucciones que tiene asociadas.

Ahora veremos las partes de las instrucciones repetitivas y su significado.

5.3. Componentes de una Instrucción Repetitiva

La figura 3.5 ilustra la manera en que se ejecuta una instrucción repetitiva. Primero, y por una sola vez, se ejecutan las instrucciones que vamos a llamar de inicio o preparación del ciclo. Allí se le da el valor inicial al índice y a las variables en las que queremos acumular los valores durante el recorrido. Luego, se evalúa la condición del ciclo. Si es falsa, se ejecutan las instrucciones que se encuentran después del ciclo. Si es verdadera, se ejecutan las instrucciones del cuerpo del ciclo para finalmente volver a repetir el mismo proceso. Cada repetición, que incluye la evaluación de la condición y la ejecución del cuerpo del ciclo, recibe el nombre de **iteración** o **bucle**.

Fig. 3.5 Ejecución de una instrucción repetitiva



Usualmente en un lenguaje de programación hay varias formas de escribir una instrucción repetitiva. En Java existen varias formas, pero en este libro sólo vamos a presentar dos de ellas: la instrucción `for` y la instrucción `while` .

5.3.1. Las Instrucciones for y while

Una instrucción repetitiva con la instrucción `while` se escribe de la siguiente manera:

```
<inicio>
while( <condición> )
{
  <cuerpo>
  <avance>
}
```

- Las instrucciones de preparación del ciclo van antes de la instrucción repetitiva.
- La condición que establece si se debe repetir de nuevo el ciclo va siempre entre paréntesis.
- El avance del ciclo es una parte opcional, en la cual se modifican los valores de algunos de los elementos que controlan la salida del ciclo (avanzar el índice con el que

se recorre un arreglo sería parte de esta sección).

Una instrucción repetitiva con la instrucción `for` se escribe de la siguiente manera:

```
<inicio1>
for( <inicio2>; <condición>; <avance> )
{
  <cuerpo>
}
```

- El inicio va separado en dos partes: en la primera, va la declaración y la inicialización de las variables que van a ser utilizadas después de terminado el ciclo (la variable `suma`, por ejemplo, en el método del promedio). En la segunda parte de la zona de inicio van las variables que serán utilizadas únicamente dentro de la instrucción repetitiva (la variable `índice`, por ejemplo, que sólo sirve para desplazarse recorriendo las casillas del arreglo).
- La segunda parte del inicio, lo mismo que el avance del ciclo, se escriben en el encabezado de la instrucción `for`.

Ejemplo 3

Objetivo: Mostrar la manera de utilizar la instrucción iterativa `for`.

En este ejemplo se presenta una implementación del método que calcula el promedio de notas del caso de estudio, en la cual se utiliza la instrucción `for`.

```
public double promedio( )
{
  double suma = 0.0;
  for(int indice = 0; indice < TOTAL_EST; indice++ )
  {
    suma += notas[ indice ];
  }
  return suma / TOTAL_EST;
}
```

- Puesto que la variable "`suma`" será utilizada por fuera del cuerpo del ciclo, es necesario declararla antes del for.
- La variable "`indice`" es interna al ciclo, por eso se declara dentro del encabezado.
- El avance del ciclo consiste en incrementar el valor del "`indice`".
- En este ejemplo, los corchetes del `for` son opcionales, porque sólo hay una instrucción dentro del cuerpo del ciclo.

Vamos a ver en más detalle cada una de las partes de la instrucción y las ilustraremos con algunos ejemplos.

5.3.2. El Inicio del Ciclo

El objetivo de las instrucciones de inicio o preparación del ciclo es asegurarnos de que vamos a empezar el proceso repetitivo con las variables de trabajo en los valores correctos. En nuestro caso, una variable de trabajo la utilizamos como índice para movernos por el arreglo y la otra para acumular la suma de las notas:

- La suma antes de empezar el ciclo debe ser cero: `double suma = 0.0;`
- El índice a partir del cual vamos a iterar debe ser cero: `int indice = 0;`

5.3.3. La Condición para Continuar

El objetivo de la condición del ciclo es identificar el caso en el cual se debe volver a hacer una nueva iteración. Esta condición puede ser cualquier expresión lógica: si su evaluación da verdadero, significa que se deben ejecutar de nuevo las instrucciones del ciclo. Si es falsa, el ciclo termina y se continúa con la instrucción que sigue después de la instrucción repetitiva.

Típicamente, cuando se está recorriendo un arreglo con un índice, la condición del ciclo dice que se debe volver a iterar mientras el índice sea menor que el número total de elementos del arreglo. Para indicar este número, se puede utilizar la constante que define su tamaño (`TOTAL_EST`) o el operador que calcula el número de elementos de un arreglo (`notas.length`).

> Dado que los arreglos comienzan en 0, la condición del ciclo debe usar el operador `<` y el número de elementos del arreglo. Son errores comunes comenzar los ciclos con el índice en 1 o tratar de terminar con la condición `indice <= notas.length`.

5.3.4. El Cuerpo del Ciclo

El cuerpo del ciclo contiene las instrucciones que se van a repetir en cada iteración. Estas instrucciones indican:

- La manera de modificar algunas de las variables de trabajo para ir acercándose a la solución del problema. Por ejemplo, si el problema es encontrar la suma de las notas de todos los estudiantes del curso, con la instrucción `suma += notas[indice]` agregamos un nuevo valor al acumulado.
- La manera de modificar los elementos del arreglo, a medida que el índice pasa por cada casilla. Por ejemplo, si queremos sumar una décima a todas las notas, lo

hacemos con la instrucción `notas[indice] += 0.1` .

5.3.5. El Avance del Ciclo

Cuando se recorre un arreglo, es necesario mover el índice que indica la posición en la que estamos en un momento dado (`indice++`). En algún punto (en el avance o en el cuerpo) debe haber una instrucción que cambie el valor de la condición para que finalmente ésta sea falsa y se detenga así la ejecución de la instrucción iterativa. Si esto no sucede, el programa se quedará en un ciclo infinito.

> Si construimos un ciclo en el que la condición nunca sea falsa (por ejemplo, si olvidamos escribir las instrucciones de avance del ciclo), el programa dará la sensación de que está bloqueado en algún lado, o podemos llegar al error: *java.lang.OutOfMemoryError*

Tarea 2

Objetivo: Practicar el desarrollo de métodos que tengan instrucciones repetitivas.

Para el caso de estudio de las notas de los estudiantes escriba los métodos de la clase Curso que resuelven los problemas planteados.

Calcular el número de estudiantes que sacaron una nota entre 3,0 y 5,0:

```
public int cuantosPasaron( )
{









}
```

Calcular la mayor nota del curso:

```
public double mayorNota( )
{









}
```

Contar el número de estudiantes que sacaron una nota inferior a la del estudiante que está en la posición del arreglo que se entrega como parámetro. Suponga que el parámetro `posEst` tiene un valor comprendido entre `0` y `TOTAL_EST - 1`.

```
public int cuantosPeoresQue( int posEst )
{









}
```

Aumentar el 5% todas las notas del curso, sin que ninguna de ellas sobrepase el valor 5,0:

```
public void hacerCurva( )
{









}
```