Vectores , matrices y funciones.

Vectores y matrices

Arrays

- Declaración
- Creación
- Acceso a los elementos de un array
- Manipulación de vectores y matrices

Algoritmos de ordenación

- Ordenación por selección
- Ordenación por inserción
- Ordenación por intercambio directo (método de la burbuja) Ordenación rápida (QuickSort)

Algoritmos de búsqueda

- Búsqueda lineal
- Búsqueda binaria

Apéndice: Cadenas de caracteres

Arrays

Un array es una estructura de datos que contiene una colección de datos del mismo tipo

Ejemplos

Temperaturas mínimas de los últimos treinta días…
Valor de las acciones de una empresa durante la última semana…

Propiedades de los arrays

- Los arrays se utilizan como contenedores para almacenar datos relacionados (en vez de declarar variables por separado para cada uno de los elementos del array).
- Todos los datos incluidos en el array son del mismo tipo. Se pueden crear arrays de enteros de tipo int o de reales de tipo float, pero en un mismo array no se pueden mezclar datos de tipo int y datos de tipo float.
- El tamaño del array se establece cuando se crea el array (con el operador new, igual que cualquier otro objeto).
- A los elementos del array se accederá a través de la posición que ocupan dentro del conjunto de elementos del array.

Terminología

Los arrays unidimensionales se conocen con el nombre de **vectores**. Los arrays bidimensionales se conocen con el nombre de **matrices**.

Declaración

Para declarar un array, se utilizan corchetes para indicar que se trata de un array y no de una simple variable del tipo especificado.

Vector (array unidimensional):

```
tipo identificador[];
```

o bien

```
tipo[] identificador;
```

donde **tipo** es el tipo de dato de los elementos del vector
identificador es el identificador de la variable.

Matriz (array bidimensional):

```
tipo identificador[][];
```

o bien

```
tipo[][] identificador;
```

NOTA: No es una buena idea que el identificador del array termine en un dígito, p.ej. vector3

Creación

Los arrays se crean con el operador new.

Vector (array unidimensional):

```
vector = new tipo[elementos];
```

Entre corchetes se indica el tamaño del vector.
tipo debe coincidir con el tipo con el que se haya declarado el **vector**. vector debe ser una variable declarada como **tipo[]**

Ejemplos

```
float[] notas = new float[ALUMNOS];
```

```
int[] temperaturas = new int[7];
```

Matriz (array bidimensional):

```
matriz = new tipo[filas][columnas];
```

Ejemplo

```
int[][] temperaturas = new int[12][31];
```

Uso

Para acceder a los elementos de un array, utilizamos índices (para indicar la posición del elemento dentro del array)

Vector (array unidimensional):

```
vector[índice]
```

- En Java, el índice de la primera componente de un vector es siempre 0.
- El tamaño del array puede obtenerse utilizando la propiedad **vector.length**
- Por tanto, el índice de la última componente es **vector.length-1**

Ejemplo

```
float[] notas = new float[3];
```

Matriz (array bidimensional):

```
matriz[índice1][índice2]
```

Una matriz, en realidad, es un vector de vectores:

- En Java, el índice de la primera componente de un vector es siempre 0, por lo que **matriz[0][0]** será el primer elemento de la matriz.

- El tamaño del array puede obtenerse utilizando la propiedad **array.length**:
  - **matriz.length** nos da el número de filas
  - **matriz[0].length** nos da el número de columnas
  - Por tanto, el último elemento de la matriz es **matriz[matriz.length-1][matriz[0].length-1]**

Inicialización en la declaración

Podemos asignarle un valor inicial a los elementos de un array en la propia declaración

```
int vector[] = {1, 2, 3, 5, 7};
int matriz[][] = { {1,2,3}, {4,5,6} };
```

El compilador deduce automáticamente las dimensiones del array.

MIND HUB.

Manipulación de vectores y matrices

Las operaciones se realizan componente a componente

Ejemplo: Suma de los elementos de un vector

```
1  static float media (float datos[])
2  {
3   int i;
4   int n = datos.length;
5   float suma = 0;
6   for (i=0; i<n; i++)
7   suma = suma + datos[i];
8   return suma/n;
9  }
```

No es necesario utilizar todos los elementos de un vector, por lo que, al trabajar con ellos, se puede utilizar una variable entera adicional que nos indique el número de datos que realmente estamos utilizando:

El tamaño del vector nos dice cuánta memoria se ha reservado para almacenar datos del mismo tipo, no cuántos datos del mismo tipo tenemos realmente en el vector.

Ejemplo: Suma de los n primeros elementos de un vector

```
1  static float media (float datos[], int n)
2  {
3   int i;
4   float suma = 0;
5   for (i=0; i<n; i++)
6   suma = suma + datos[i];
7   return suma/n;
8  }
```

MIND HUB.

Ejemplo

```
1  public class Vectores
2  {
3   public static void main (String[] args)   {
4   int pares[] = { 2, 4, 6, 8, 10 };  int impares[] = { 1, 3, 5, 7, 9 };
5   mostrarVector(pares);
6   System.out.println("MEDIA="+media(pares));
7   mostrarVector(impares);
8   System.out.println("MEDIA="+media(impares));  }
9   static void mostrarVector (int datos[])  {
10  int i;
11  for (i=0; i<datos.length; i++)
12  System.out.println(datos[i]);  }
13  static float media (int datos[])
14  {
15  int i;
16  int n = datos.length;
17  int suma = 0;
18  for (i=0; i<n; i++)
19  suma = suma + datos[i];
20  return suma/n;
21  }
22 }
```

```
1   static int[] leerVector (int datos)  {
2    int i;
3    int[] vector = new int[datos];
4    for (i=0; i<datos; i++)
5    vector[i] = leerValor();
6    return vector;
7    }
```

IMPORTANTE:
Cuando se pasa un array como parámetro, se copia una referencia al array y no el conjunto de valores en sí.

Por tanto, tenemos que tener cuidado con los efectos colaterales que se producen si, dentro de un módulo, modificamos un vector que recibimos como parámetro.

Ejemplo

El siguiente método lee los elementos de un vector ya creado

```
1   static void leerVector (int[] datos)  {
2   int i;
3   for (i=0; i<datos.length; i++)  datos[i] = leerValor();
4   }
```

Copia de arrays

La siguiente asignación sólo copia las referencias, no crea un nuevo array:

```
int[] datos = pares;
```

Para copiar los elementos de un array, hemos de crear un nuevo array y copiar los elementos uno a uno

```
int[] datos = new int[pares.length];

 for (i=0; i<pares.length; i++)
 datos[i] = pares[i]
```

También podemos utilizar una función predefinida en la biblioteca de clases estándar de Java:

```
System.arraycopy(from,fromIndex,to,toIndex,n);
```

```
int[] datos = new int[pares.length];
System.arraycopy(pares,0,datos,0,pares.lengt
h);
```

EXTRA:

La biblioteca de clases de Java incluye una clase auxiliar llamada **java.util.Arrays** que incluye como métodos algunas de las tareas que se realizan más a menudo con vectores:

- **Arrays.sort(v)** ordena los elementos del vector.

- **Arrays.equals(v1,v2)** comprueba si dos vectores son iguales.
- **Arrays.fill(v,val)** rellena el vector **v** con el valor **val**.
- **Arrays.toString(v)** devuelve una cadena que representa el contenido del vector
- **Arrays.binarySearch(v, k)** busca el valor k dentro del vector v (que previamente ha de estar ordenado).

Ejemplos

Un programa que muestra los parámetros que le indicamos en la línea de comandos:

```
1  public class Eco
2  {
3   public static void main(String args[])
4   {
5   int i;
6   for (i=0; i<args.length; i++)
7   System.out.println(args[i]);
8   }
9  }
```

Un método que muestra el contenido de una matriz:

```
1  public static void mostrarMatriz (double matriz[][]) {
2   int i,j;
3   int filas = matriz.length;
4   int columnas = matriz[0].length;
5   // Recorrido de las filas de la matriz
6   for (i=0; i<filas; i++) {
7   // Recorrido de las celdas de una fila  for (j=0; j<columnas; j++) {
8   System.out.println ( “matriz[”+i+“][”+j+“]=”   + matriz[i][j] );
9   }
10  }
11 }
```

FUNCIONES

Métodos en Java, funciones y procedimientos. Cómo hacerlos y usarlos

Los métodos en Java, las funciones y los procedimientos, especialmente en Java, son una herramienta indispensable para programar. Java nos permite crear o hacer nuestros propios métodos y usarlos sencillamente como también nos facilita hacer uso de los métodos de otras librerías (funciones matemáticas, aritméticas, de archivos, de fechas, etc.)

Cualquiera que sea el caso, las funciones permiten automatizar tareas que requerimos con frecuencia y que además se pueden generalizar por medio de parámetros o argumentos. Aprender a crear métodos en Java y usarlos correctamente es de gran importancia, separar nuestro código en módulos y según las tareas que requerimos.

En java una función debe contener la implementación de una utilidad de nuestra aplicación, esto nos pide que por cada utilidad básica (abrir, cerrar, cargar, mover, etc.) sería adecuado tener al menos una función asociada a ésta, pues sería muy complejo usar o crear un método que haga todo de una sola vez, por esto es muy buena idea separar cada tarea en una función o método (según corresponda).

Para estar claros en todo, en Java es mucho más común hablar de métodos que de funciones y procedimientos y esto se debe a que en realidad un método, una función y un procedimiento NO son lo mismo, veamos la diferencia:

¿Funciones, métodos o procedimientos?

Es muy común entre programadores que se hable indistintamente de estos tres términos sin embargo poseen diferencias fundamentales.

Funciones:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente retornan un valor. En otras palabras una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y retorna un valor usando la instrucción return, si no retorna algo, entonces no es una función. En java las funciones usan el modificador static.

Métodos:

Los métodos y las funciones en Java están en capacidad de realizar las mismas tareas, es decir, son funcionalmente idénticos, pero su diferencia radica en la manera en que hacemos uso de uno u otro (el contexto). Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo en método está asociado a un objeto, SIEMPRE, básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada.

Nota: Es aquí donde digo que en Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función esta debe ser static, para que no requiera de un objeto para ser llamada.

Procedimientos:

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo en la definición no hay nada que se lo impida. En el contexto de Java un procedimiento es básicamente un método cuyo tipo de retorno es **void** que no nos obliga a utilizar una sentencia return.

Crear un método en Java

La sintaxis para declarar una función es muy simple, veamos:

```
[acceso] [modificador] tipo nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento]...])
{
    /*
        * Bloque de instrucciones
    */

    return valor;
}
```

El primer componente corresponde al **modificador de acceso**, que puede ser public o private, éste es opcional, si no ponemos nada, se asume el modificador de acceso por defecto, el segundo componente es el modificador que puede ser **final o static** (o ambas), también es opcional.

Recordemos que un método o función siempre retorna algo, por lo tanto es obligatorio declararle un tipo (el tercer componente de la sintaxis anterior), puede ser entero (int), booleano (boolean), o cualquiera que consideremos, inclusive tipos complejos, luego debemos darle un nombre a dicha función, para poder identificarla y llamarla (invocarla)

durante la ejecución, después al interior de paréntesis, podemos poner los argumentos o parámetros. Luego de la definición de la "firma" del método, se define su funcionamiento entre llaves; todo lo que esté dentro de las llaves es parte del cuerpo del método y éste se ejecuta hasta llegar a una instrucción return.

Acerca de los argumentos o parámetros

Hay algunos detalles respecto a los argumentos de un método, veamos:

- Una función, un método o un procedimiento pueden tener una cantidad cualquier de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.
- Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.
- Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo del método.
- Al recibir un argumento nada nos obliga a hacer uso de éste al interior del método, sin embargo para qué recibirlo si no lo vamos a usar.
- En Java los parámetros que podemos recibir pueden ser por valor por referencia, esto implica que si modificamos los valores recibidos al interior del método, estos pueden mantener sus cambios o no después de ejecutada el método (esto lo explico con más detalla enseguida).

Consejos acerca de return

Debes tener en cuenta dos cosas importantes con la sentencia return:

- Cualquier instrucción que se encuentre después de la ejecución de return NO será ejecutada. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.
- El tipo del valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.
- En el caso de los procedimientos (void) podemos usar la sentencia return pero sin ningún tupo de valor, sólo la usaríamos como una manera de terminar la ejecución del procedimiento.

Ejemplos de métodos

Veamos algunos ejemplos prácticos de métodos en Java.

```java
int metodoEntero()//Función sin parámetros
{
    int suma = 5+5;
    return suma; //Acá termina la ejecución del método
    //return 5+5;//Este return nunca se ejecutará
    //Intenta intercambiar la línea 3 con la 5
    //int x = 10; //Esta línea nunca se ejecutará
}
```

Como puedes ver es un ejemplo sencillo, es un método llamado metodoEntero, si ejecutas esto, la función te retornará el valor de suma que es 10 (5+5). Las líneas posteriores no se ejecutarán nunca, aunque no generan error alguno, no tienen utilidad. Puedes notar que para este caso es lo mismo haber escrito return suma que escribir return 5+5. Ambas líneas funcionan equivalentemente.

Nota: Recuerda que en Java todo debe estar al interior de una clase (o interfaz) y que debes tener al menos la función main.

Ejemplo 2:

```java
public String metodoString(int n)//método con un parámetro
{
    if(n == 0)//Usamos el parámetro en la función
    {
        return "a"; //Si n es cero retorna a
        //Notar que de aquí para abajo no se ejecuta nada más
    }
    return "x";//Este return sólo se ejecuta cuando n NO es cero
}
```

Aquí creamos un método público, hicimos uso de múltiples sentencia return y aprovechamos la característica de que al ser ejecutadas finalizan inmediatamente la

ejecución de la parte restante del método. De este modo podemos asegurar que la función retornará "a" únicamente cuando el valor del parámetro n sea cero y retornará un "x" cuando dicho valor no sea cero.

Ejemplo 3:

```
static boolean metodoBoolean(boolean n, String mensaje)//Método con dos parámetros
{
    if(n)//Usamos el parámetro en el método
    {
        System.out.println(mensaje);//Mostramos el mensaje
    }
    return n; //Usamos el parámetro como valor a retornar
}
```

Aquí ya tenemos una función (digo función y no método porque es static) que recibe dos parámetros, uno de ellos es usado en el condicional y el otro para mostrar su valor por pantalla con ***System.out.println***, esta vez retornamos valores booleanos true o false y utilizamos el valor propio recibido en el parámetro. Toma en cuenta que en esta ocasión únicamente usamos una sentencia return, pues usar una al interior del if habría sido innecesario y el resultado sería el mismo.

Hablemos un poco de los procedimientos

Los procedimientos son similares a las funciones, aunque más resumidos. Debido a que los procedimientos no retornan valores, no hacen uso de la sentencia return para devolver valores y no tienen tipo específico, sólo **void**. Veamos un ejemplo:

Ejemplo de procedimientos

```
void procedimiento(int n, String nombre) //Notar el void
{
    if(n > 0 && !nombre.equals(""))//usamos los dos parámetros
    {
        System.out.println("hola " + nombre);
        return; //Si no ponemos este return se mostraría hola y luego adiós
    }
    //También podríamos usar un else en vez del return
    System.out.println("adios");
}
```

De este ejemplo podemos ver que ya no se usa un tipo sino que se pone void, indicando que no retorna valores, también podemos ver que un procedimiento también puede recibir parámetros o argumentos.

Recuerda: Los procedimientos también pueden usar la sentencia return, pero no con un valor. En los procedimientos el **return** sólo se utiliza para finalizar allí la ejecución.

Invocando funciones y procedimientos en Java

Ya hemos visto cómo hacer funciones en Java, cómo se crean y cómo se ejecutan, ahora veamos cómo usar un método, función o procedimiento.

```
nombre([valor,[valor]...]);
```

Como puedes notar es bastante sencillo invocar o llamar funciones en Java, sólo necesitas el nombre del método, función o procedimiento y enviarle el valor de los parámetros. Hay que hacer algunas salvedades respecto a esto.

Detalles para invocar métodos funciones y procedimientos

- No importa si se trata de un método en Java o de una función o de un método, sólo debes ocuparte de enviar los parámetros de la forma correcta para invocarlos.
- El nombre debe coincidir exactamente al momento de invocar, pues es la única forma de identificarlo.
- El orden de los parámetros y el tipo debe coincidir. Hay que ser cuidadosos al momento de enviar los parámetros, debemos hacerlo en el mismo orden en el que fueron declarados y deben ser del mismo tipo (número, texto u otros).
- Cada parámetro enviado también va separado por comas.
- Si una función no recibe parámetros, simplemente no ponemos nada al interior de los paréntesis, pero SIEMPRE debemos poner los paréntesis.
- Invocar una función sigue siendo una sentencia común y corriente en Java, así que ésta debe finalizar con ';' como siempre.
- El valor retornado por un método o función puede ser asignado a una variable del mismo tipo, pero no podemos hacer esto con un procedimiento, pues no retornan valor alguno.
- Una función puede llamar a otra dentro de sí misma o incluso puede ser enviada como parámetro a otra (mira el siguiente ejemplo).

Ejemplos de uso de funciones

En el siguiente código vamos a hacer un llamado a algunas de las funciones y al procedimiento, que declaramos anteriormente.

```
public class Ejemplos
{
    public static void main(String args[])//Siempre necesitamos un main
    {
        Ejemplos ejemplo = new Ejemplos(); //Cuando no es estático, debe usarse u

        //Llamando a un método sin argumentos, usando el objeto
        ejemplo.metodoEntero();

        //Asignando el valor retornado a una variable
        boolean respuesta = metodoBoolean(true, "hola");

        // El procedimiento no es static, así que debe llamarse desde el objeto.
        ejemplo.procedimiento(0, "Juan");//Invocando el procedimiento

        //Usando una función como parámetro
        ejemplo.procedimiento(metodoBoolean(1, "hola"), "Juan");
        //Lo que retorne metodoBoolean (en este caso 1) se envía al procedimiento
    }
}
```

En el código anterior podemos ver cómo todo ha sido invocado al interior la función main (la función principal), esto nos demuestra que podemos hacer uso de funciones al interior de otras. También vemos cómo se asigna el valor retornado por el método a la variable 'respuesta' y finalmente, antes del return, vemos cómo hemos usado el valor retornado por 'funcionBool' como parámetro del procedimiento.

Ejercicios resueltos de métodos en Java

Vamos a realizar un par de ejercicios sobre métodos y los vamos a resolver y explicar.

Funciones anidadas

Para este ejercicio vamos a crear una función que llama a otra al interior de ella (por eso las quise llamar anidadas). Es un sistema de validación de un usuario que recibe un usuario y una contraseña y según sean válidos o no, muestra un mensaje al usuario.

MIND HUB.

```java
public class Ejercicios
{
    public static String saludar(String nombre)
    {
        //Se crea el mensaje de saludo
        String saludo = "Hola. Bienvenido " + nombre;

        return saludo;//Se retorna el saludo
    }

    public static String error(String nombre)
    {
        //Se crea el mensaje de error
        String error = "Ups. No pudimos validar tus datos. " + nombre + " es tu usuario?";

        return error; //Se retorna el error
    }

    public static void verificar(String usuario, String contrasenia)
    {
        String usuarioValido = "JuanDMeGon";

        String contraseniaValida = "MiPass";

        //Se validan los datos
        if(usuarioValido.equals(usuario) && contraseniaValida.equals(contrasenia))
        {
            //Si son validos se llama ala función saludar y se muestra el mensaje retornado por pantalla
            System.out.println(saludar(usuario));
            return; //Terminamos la ejecución
        }
        //Si no son válidos entonces mostramos el mensaje de error de la funcion error.
        System.out.println(error(usuario));
    }

   public static void main(String[] args)
   {
       String usuario = "Juan";
       String contrasenia = "pass";

       //Se hace la verificación
       verificar(usuario, contrasenia);

       //Mostrará el mensaje error.
   }
}
```