Programación en Lenguaje Java

Tema 4. Datos compuestos

Michael González Harbour

Mario Aldea Rivas

Departamento de Matemáticas,
Estadística y Computación

Programación en Java

1. Introducción a los lenguajes de programación
2. Datos y expresiones
3. Estructuras algorítmicas
4. *Datos compuestos*
   - Arrays y tablas unidimensionales. Algoritmos de recorrido y búsqueda. Arrays multidimensionales. Tablas de tamaño variable. Tipos enumerados.
5. Entrada/salida
6. Clases, referencias y objetos
7. Modularidad y abstracción
8. Herencia y polimorfismo
9. Tratamiento de errores
10. Entrada/salida con ficheros
11. Pruebas

Construcción de tablas mediante arrays

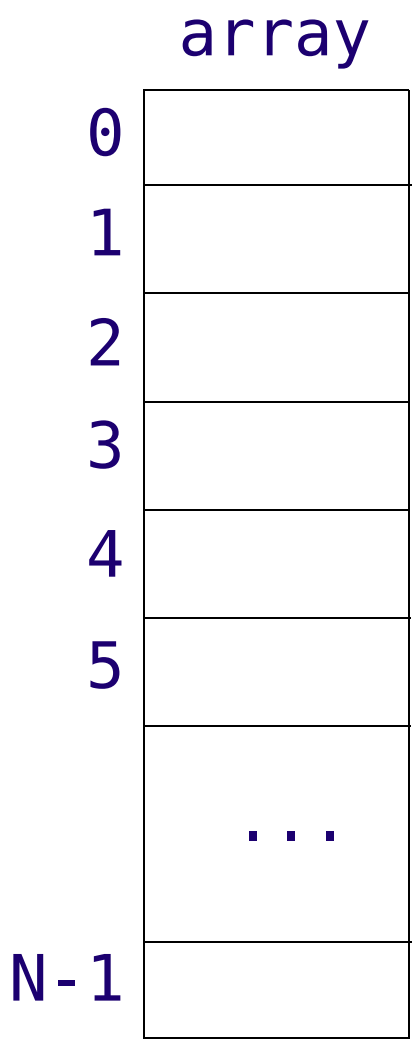
Los arrays permiten guardar muchos datos del mismo tipo

- se agrupan bajo un nombre común
- se utiliza un índice numérico para referirse al dato individual
  - en Java el índice se restringe: `[0..N-1]`
- o varios índices en el caso de arrays multidimensionales
- los datos pueden ser variables o referencias a objetos

El tamaño es fijo: no puede cambiar

El array en Java se usa como un objeto

- se usa a través de una referencia
- se crea con new



Arrays unidimensionales

Declaración:

```
tipo nombre[];
```

O también:

```
tipo[] nombre;
```

- El tipo es el tipo base de los elementos del array
  - un tipo primitivo (double, int, char, etc.)
  - o una clase
- El nombre representa una referencia a un objeto del tipo array, que aún no existe
  - inicialmente vale `null`

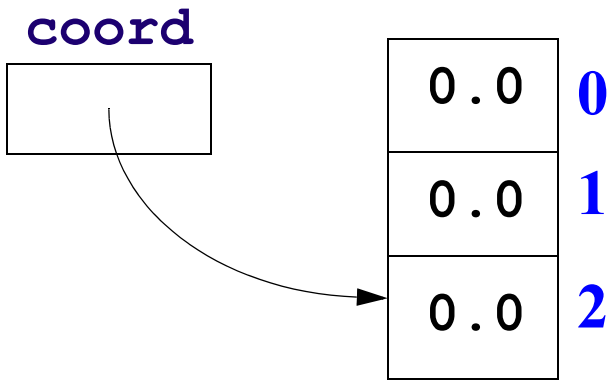
Para crear un objeto del tipo array

```
nombre=new tipo[tamaño];
```

Ejemplos de creación de arrays

Declaración y posterior creación

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



O todo junto:

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

O equivalentemente:

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

El array referenciado por `coord` tiene tres elementos enteros

- numerados 0,1, y 2.

Ejemplos de creación de arrays (cont.)

Array que contendrá el número de días de cada mes de un año no bisiesto:

```
int[] diasMes = new int[12];
```

El array referenciado por `coord` tiene tres elementos enteros

- numerados 0,1, y 2.
- al crear un array todos sus elementos se inicializan
  - los números a cero
  - los booleanos a `false`
  - las referencias a objetos a `null`

Nota: En otros lenguajes (por ejemplo en C) los arrays no se inicializan

Uso de arrays

Observar que un array no es una clase, sino un objeto

- no tiene operaciones
- pero sí el atributo `length`, que contiene el tamaño

```
coord.length  // es 3
```

Uso de un elemento de un array:

```
nombre[indice]
```

Si se intenta acceder a una casilla que no existe, se produce un error

Uso de arrays (cont.)

Ejemplos de uso

```
final int ENERO=0, FEBRERO=1; // etc
diasMes[1] = 28;  // mejor FEBRERO, no 1
diasMes[ENERO] = 31;
System.out.println
    ("Febrero tiene "+diasMes[FEBRERO]+" dias");
```

Se puede escribir un literal de array

- solo en la *declaración*, al crear la referencia al array:

```
int[] diasMes =
   {31,28,31,30,31,30,31,31,30,31,30,31};
```

Esquema de recorrido en tablas

El recorrido para hacer algo con todas sus casillas es un algoritmo muy frecuente en tablas

Podemos recorrer con una variable de control todos los índices

- por ejemplo para un array de n números reales

```
real[0..n-1] tabla

para i desde 0 hasta n-1 hacer
    trabajar con tabla[i]
fin para
```

También podemos recorrer todas las casillas

```
para cada x en tabla hacer
    trabajar con x
fin para
```

Implementación del recorrido en Java

Array de números reales

```
double[] tabla=new double[n];
```

Recorriendo los índices

```
for (int i=0; i<tabla.length; i++) {
    //trabajar con tabla[i]
}
```

Recorriendo los elementos del array

```
for (double x: tabla) {
    //trabajar con x
}
```

Veremos más adelante que esta última modalidad del bucle `for` también funciona con otras tablas además de los arrays

Ejemplo de recorrido en tablas: tabla de enteros

```
/**
 *  Clase que contiene un atributo que es un array
 *  de enteros y métodos para calcular la media y
 *  mostrarla en pantalla
 */
public class TablaEnteros {

  private int[] nums;
```

Ejemplo de recorrido en tablas: tabla de enteros (cont.)

```
/**
 * Constructor, al que se le pasa un array
 * Copia todos sus elementos
 */
public TablaEnteros(int[] numeros) {

  // crea el array del mismo tamaño que el parámetro
  nums=new int[numeros.length];

  // copia todos los elementos
  // debemos usar el bucle con índices
  for (int i=0; i<nums.length; i++) {
    nums[i]=numeros[i];
  }
}
```

Ejemplo de recorrido en tablas: tabla de enteros (cont.)

```
    /**
     * Retorna la media de los números almacenados
     */
    public double media() {
        int suma=0;
        // for (int i=0; i<nums.length; i++) {
        //     suma=suma+nums[i];
        // }

        // podemos usar el bucle "para cada"
        for (int x : nums) {
            suma=suma+x;
        }
        return(double)suma / nums.length;
    }
}
```

Ejemplo de recorrido en tablas: tabla de enteros (cont.)

```
/**
 * Programa que usa TablaEnteros
 */
public class Media {

   public static void main(String[] args) {

      int[] n = {3,4,7,8,4,5,6};
      TablaEnteros t=new TablaEnteros(n);

      System.out.println("La media es "+t.media());
      System.out.println("Longitud es "+n.length);
   }
}
```

Uso de arrays como parámetros

Se pueden pasar a un método parámetros del tipo array

En el constructor del ejemplo anterior se pasa un array de enteros llamado `numeros`

```
public TablaEnteros(int[] numeros)
```

Al igual que con los objetos pasados como parámetros, el método puede modificar el array original

Recorrido parcial de una tabla

En ocasiones nos interesa un recorrido parcial

- no podemos usar el bucle "*para cada*"
- debemos recorrer los índices que nos interesen

Ejemplo: algoritmo que calcula el máximo de los elementos de una tabla

```
real[0..n-1] tabla
// max contendrá el máximo encontrado hasta
// el momento; inicialmente es la primera casilla
real max=tabla[0]
// comparamos max con el resto de casillas
para i desde 1 hasta n-1 hacer
   si tabla[i]>max entonces
      max=tabla[i]
   fin si
fin para
```

Ejemplo de recorrido parcial: Máximo de unas edades de personas

```
/**
 * Clase que contiene la edad y el nombre
 * de una persona
 */
public class Persona {

    private int edad;
    private String nombre;

    /** Constructor al que se le pasa la edad
     * y el nombre */
    public Persona(int edad, String nombre) {
        this.edad=edad;
        this.nombre=nombre;
    }
```

Ejemplo de recorrido parcial (cont.)

```
    /**
     * Retorna la edad
     */
    public int edad() {
        return edad;
    }

    /**
     *Retorna el nombre
     */
    public String nombre() {
        return nombre;
    }
}
```

Ejemplo de recorrido parcial (cont.)

```
/**
 * Programa que calcula la edad máxima a partir de
 * una lista de personas
 */
public class ListaPersonas {
```

Ejemplo de recorrido parcial (cont.)

```
/**
 * Calcula la edad máxima de un array de personas,
 * con un recorrido desde el segundo al ultimo
 */
public static int edadMaxima(Persona[] perso) {
  int max=perso[0].edad();

  // comenzamos por el segundo elemento
  for (int i=1;i<perso.length; i++) {
    if (perso[i].edad()>max) {
        max=perso[i].edad();
    }
  }
  return max;
}
```

Ejemplo de recorrido parcial (cont.)

```
   /**
    * Programa de prueba; crea una tabla de personas
    *  e invoca al metodo anterior
    */
   public static void main (String[] args) {
     Persona[] lista=new Persona[5];
     lista[0]=new Persona(10,"juan");
     lista[1]=new Persona(14,"pedro");
     lista[2]=new Persona(18,"andres");
     lista[3]=new Persona(8,"ana");
     lista[4]=new Persona(13,"lucia");
     System.out.println("Edad máxima:"+
                        edadMaxima(lista));
   }
}
```

Algoritmo de búsqueda en tablas

La búsqueda de un elemento que cumple una determinada propiedad es otro algoritmo muy habitual en tablas

Se parece al recorrido

- pero cuando encontramos el elemento buscado cesamos el recorrido

Hay que prever el caso de que no encontremos lo buscado

Esquema de búsqueda

Buscaremos en: tabla[0..n-1]:

```
entero i=0
booleano encontrado=false
mientras i<n y no encontrado hacer
   si tabla[i] cumple la propiedad entonces
      encontrado=true
   si no
      i++
   fin si
fin mientras

si encontrado entonces
   trabajar con la casilla buscada: tabla[i]
si no
   resolver el caso no encontrado
fin si
```

Ejemplo de búsqueda

Añadimos a la clase `ListaPersonas` una operación para buscar una persona en la tabla de personas

- La búsqueda es por el nombre

```
/**
 * Busca en perso la persona cuyo nombre
 * se indica, y retorna su edad,
 * o -1 si no se encuentra
 */
public static int edadDe
 (Persona[] perso, String nombre)
{
  // inicializaciones
  boolean encontrado=false;
  int i=0; // índice para el array
```

Ejemplo de búsqueda (cont.)

```
    // recorremos el array mientras no hayamos
    // encontrado la persona buscada
    while (i<perso.length && !encontrado) {
      if (perso[i].nombre().equals(nombre)) {
        encontrado=true;
      } else {
        i++;
      }
    }
    // retorna edad encontrada o -1 si no encontrado
    if (encontrado) {
      return perso[i].edad();
    } else {
      return -1;
    }
  }
```

Alternativa “de los vagos” para la búsqueda

Cuando se escribe un método cuyo objetivo es retornar lo buscado se puede usar un esquema más sencillo para la búsqueda:

```
método …
    para i desde 0 hasta n-1 hacer
        si tabla[i] cumple la propiedad entonces
            retornar dato asociado a tabla[i]
        fin si
    fin para
    // si el bucle acaba, no se encontró lo buscado
    retornar dato que indique "no encontrado"
fin método
```

Esta alternativa es más difícil de verificar

- y rompe la regla de no salir desde el interior de un bucle

Ejemplo de búsqueda: alternativa

```
public static int edadDe
 (Persona[] perso, String nombre)
{
  // Realizamos un recorrido, pero lo abandonamos
  // si encontramos la persona buscada
  for (int i=0;i<perso.length; i++) {
    if (perso[i].nombre().equals(nombre)) {
        return perso[i].edad();
    }
  }
 // Si el bucle acaba, no se ha encontrado el nombre
  return -1;
}
```

Uso de un array completo

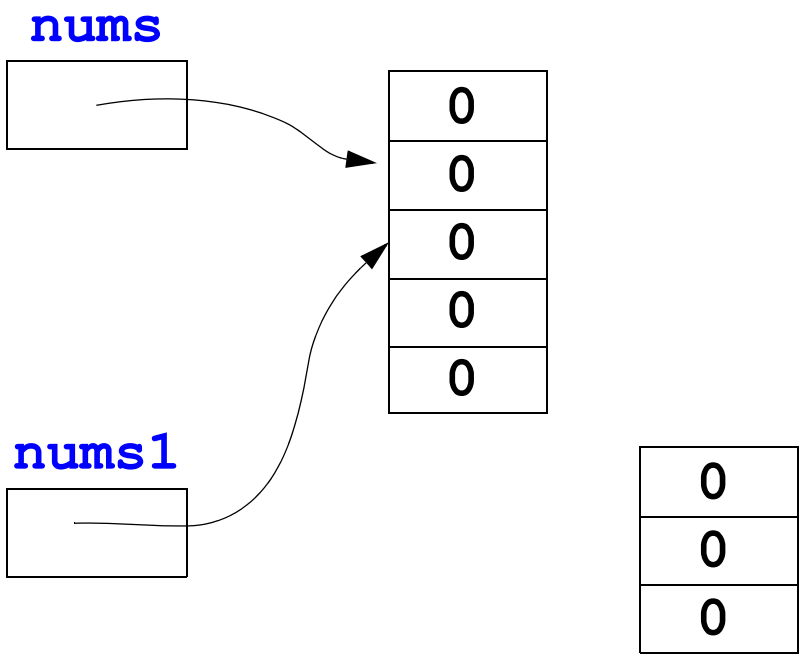
Se hace por su nombre

- Se puede cambiar la referencia al array.

```
int nums[]=new int[5];
int nums1[]=new int[3];
...
nums1=nums;
```

Esto no copia el array

- sólo cambia la referencia



Arrays multidimensionales

Los arrays multidimensionales son arrays de arrays, con algunas facilidades para hacer más simple su uso.

Declaración y creación

```
int matrizA[][] = new int[4][5];
```

El índice izquierdo representa la fila, y el derecho la columna

Los elementos se usan de la manera

```
matrizA[2][3]
```

Aunque los tamaños de cada fila pueden ser distintos, no es aconsejable (es fácil equivocarse)

- Número de filas: `matrizA.length` vale 4
- Número de columnas de la fila `i`: `matrizA[i].length` vale 5

Recorrido de un array multidimensional

Se requiere un bucle que recorre las filas y otro anidado que recorre las columnas

```
real matriz[0..m-1][0..n-1]
para i desde 0 hasta m-1 hacer
   para j desde 0 hasta n-1 hacer
      trabajar con matriz[i][j]
   fin para
fin para
```

Ejemplo en Java: mostrar en pantalla todos los elementos de la matriz `m`

```
for (int i=0; i<m.length; i++){
   for (int j=0; j<m[i].length; j++) {
      System.out.println
       ("m["+i+"]["+j+"]="+m[i][j]);
   }
}
```

Tablas de tamaño variable

Los arrays son de tamaño fijo

- Se puede guardar una tabla de tamaño variable (pero limitado)
- Basta usar sólo la primera parte del array, dejando el resto sin usar
- Hay que llevar cuenta del número de casillas útiles

Existe una clase en Java que representa tablas de tamaño variable, cuyo tamaño puede crecer

- es la clase `ArrayList`
- está en el paquete `java.util`
- pertenece a las llamadas "***Java collections***"
- sólo se pueden almacenar objetos

El uso de la clase ArrayList

Declaración y creación de una tabla cuyos elementos son objetos de la clase Elemento

```
ArrayList <Elemento> v= new ArrayList <Elemento> ();
```

Métodos para manejar la tabla

Cabecera	Descripción
`int size()`	Obtener el tamaño actual
`Elemento get(int índice)`	Obtener el elemento de la casilla indicada
`void set(int índice, Elemento e)`	Cambiar el elemento de la casilla indicada
`boolean add(Elemento e)`	Añadir el elemento al final de la lista

Ejemplo con la clase ArrayList

Es el mismo ejemplo inicial de la lista de personas

```
import java.util.ArrayList;
public class ListaPersonas1{

    public static int edadMaxima
        (ArrayList<Persona> perso)
    {
        int max=perso.get(0).edad();
        // empezamos por el segundo elemento
        for (int i=1;i<perso.size(); i++) {
          if (perso.get(i).edad()>max) {
              max=perso.get(i).edad();
          }
        }
        return max;
    }
```

Ejemplo con la clase ArrayList (cont.)

```
    public static void main (String[] args) {
        ArrayList<Persona> lista=
            new ArrayList<Persona>();
        lista.add(new Persona(10,"juan"));
        lista.add(new Persona(14,"pedro"));
        lista.add(new Persona(18,"andres"));
        lista.add(new Persona(8,"ana"));
        lista.add(new Persona(13,"lucia"));

        System.out.println("La edad máxima es "+
                           edadMaxima(lista));

    }
}
```

Bucle "para cada"

La forma "para cada" del lazo `for` para recorrer ***arrays*** sirve también para estructuras de datos Java como el `ArrayList`

```
// suma de todos los elementos del array
// nums de números enteros
for (int i : nums1) {
   suma=suma+i;
}

// recorre todas las personas de un ArrayList
// y muestra sus nombres en pantalla
for (Persona p : lista) {
   System.out.println(p.nombre());
}
```

- no se pueden hacer recorridos parciales
- sí se pueden modificar objetos contenidos en el `ArrayList`, pues la variable del bucle es una referencia

7. Tipos enumerados

Un tipo enumerado es aquel en el que los posibles valores son un conjunto finito de palabras

- por ejemplo un Color (***Rojo***, ***Verde***, ***Azul***)
- o un día de la semana (***lunes***, ***martes***,...)

En lenguajes sin enumerados se usan constantes enteras

- pero esta solución puede dar errores si se usan enteros no previstos
- además, los tipos enumerados tienen facilidades para trabajar con los valores

Declaración de clases enumeradas

Declarar una clase enumerada

```
public enum Nombre
   {palabra1, palabra2, palabra3, ...};
```

Ejemplo:

```
public enum Color {ROJO, VERDE, AZUL};
```

Elementos de una clase enumerada

La clase `Color` tiene los siguientes elementos:

- variables estáticas: `ROJO, VERDE, AZUL`
- método estático `values()`, que retorna un array con todas las constantes del tipo enumerado
- método estático `valueOf(String s)` que retorna la conversión a valor enumerado del string `s`
- métodos `equals()`, `toString()`, etc.

Los tipos enumerados pueden usarse en una instrucción `switch`

- usando directamente los valores (ej.: `ROJO, VERDE`, ...)
- sin poner delante el nombre de la clase

Ejemplo

```
import fundamentos.*;

/**
 * Clase de prueba del tipo enumerado Color
 */
public class Colores
{
    public enum Color {ROJO, VERDE, AZUL};

    public static void main (String[] args)
    {

        // mostrar la lista de todos los colores
        for (Color c : Color.values()) {
            System.out.println("Color "+c);
        }
```

Ejemplo (cont.)

```
        //usar un color
        Color c=Color.VERDE;
        System.out.println("El color c es: "+c);

        // leer un texto
        Lectura l=new Lectura("Leer un color");
        l.creaEntrada("color","");
        l.esperaYCierra();
        String s=l.leeString("color");

        // convertir texto a un color
        c=Color.valueOf(s);
        System.out.println("El color c es: "+c);
    }
}
```