Programación

PLG
Planning and Learning Group

Universidad Carlos III de Madrid

Estructuras de Datos Simples

En este tema

Estructuras de Datos Simples

- Arrays

Estructuras de Datos

- **Estructura de Datos**: Conjunto de datos agrupados en una sola variable
  - Reservamos varias celdas de memoria, una para cada dato
  - Asignamos un nombre al grupo completo de datos
- Estructuras de Datos Básicas:
  1. Arrays: para agrupar datos del mismo tipo
  2. Registros: para agrupar datos de distinto tipo
     - Java no tiene registros. En su lugar utiliza objetos. Una especie de “super registros” que veremos más adelante.

Arrays

- **Arrays**: Conjunto de datos del mismo tipo que ocupan posiciones sucesivas de memoria y a los que se accede mediante una única variable.
- Definición de arrays
  - Definir la variable

```
int[] numeros;

int otroNumeros[];
```

  - Definir el número de elementos y reservar memoria para ellos

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

  - Multiples definiciones y asignaciones, igual que en tipos básicos

```
int[] a, b = new int[5], c = new int[8];
```

Arrays en la memoria

- Un array es un puntero, una referencia a otra celda de memoria
- Cuando se declara un array su contenido es el valor especial `null` (no apunta a ningún sitio).
- Cuando reservamos memoria nueva y la asignamos a un array, los datos individuales tienen valor por defecto.
  - 0 para los números
  - `false` para los `boolean`
  - la cadena vacía para los `char`
  - `null` para los `String`

Acceso a Elementos

- Podemos asignar valores iniciales a los elementos de un array

```
int[] nums = new int [] {2,4,8,10};
```

- Para acceder a los elementos de un array indicamos la posición entre corchetes

```
int a = 7, b;
int[] nums =new int[5];
nums[0] = 15;
nums[2] = 10;
b = nums[2];
```

- Las posiciones van de 0 a (num. elementos - 1)

Arrays y Bucles

- Se suele utilizar un bucle cuando tenemos que asignar o acceder a varios elementos de un array
- `nombreArray.length` nos indica el tamaño del array para reconocer el número de iteraciones que necesitamos.

```
// Los 10 primeros numeros de la serie Fibonacci
int i;
int [] fibonacci = new int [10];

fibonacci[0] = 1;
fibonacci[1] = 1;

for (i = 2; i < 10; i++)
{
    fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
}

for (i = 0; i < fibonacci.length; i++)
{
    System.out.print(fibonacci[i] + " ");
}
```

Asignaciones y Copias de Arrays

- Las variables de un tipo no básico almacenan siempre un puntero, una referencia a otra celda de memoria
- ¡Ojo! la asignación directa (con “=”) de un array a otro copia el puntero
- ¡Ojo! dos arrays son iguales (con “==”) solamente cuando apuntan a la misma dirección de memoria

Asignaciones y Copias de Arrays

- Para copiar el contenido de dos arrays podemos
  1. Copiar elemento a elemento utilizando un bucle
  2. Utilizar el `System.arraycopy(origen, pos, destino, pos, n_elementos)`
- Para ver si el contenido de dos arrays es igual hay que comparar elemento a elemento

Ejemplo copia de arrays

```
int[] nums = new int [] {2,4,8,10};

int[] numsCopy = new int[4];

System.arraycopy(nums, 0, numsCopy, 0, nums.length);
```

Más sobre Arrays

- Una vez definido el número de elementos, este no se puede cambiar. Hay que crear un nuevo array y copiar los elementos
- Si declaramos un array como `final`, lo que es constante es su dirección de memoria. Los elementos de dicho array se pueden cambiar

Bucles `for each` para Arrays

- A partir de Java 5 hay una forma alternativa de iterar sobre los elementos de un array
- La instrucción se lee como **for each**, o para cada elemento del array repetir el bucle.
- Sintaxis:

```
for(tipo_array variable_elemento : nombreArray)
    { sentencias }
```

- No vale para cambiar valores del array

Ejemplo `for` each para arrays

```
int[] nums = new int [] {2,4,8,10};
int[] numsCopy = new int[4];

System.arraycopy(nums, 0, numsCopy, 0, nums.length);

System.out.println("array nums");

for (int i = 0; i< nums.length; i++){
    System.out.println(nums[i]);
}

System.out.println("array numsCopy");

for (int elem : numsCopy){
    System.out.println(elem);
}
```

Matrices

- Matrices: Arrays de 2 dimensiones
- Declaración y creación en memoria similar a los de una dimensión

```
int[][] matriz = new int[3][3];
int[][] matriz2 = {{2,4},{3, 2}};
```

- La primera posición corresponde a las filas y la segunda a las columnas
- *En memoria cada fila es un array*

Matrices

- Matrices: Arrays de 2 dimensiones
- Declaración y creación en memoria similar a los de una dimensión

```
int[][] matriz = new int[3][3];
int[][] matriz2 = {{2,4},{3, 2}};
```

- La primera posición corresponde a las filas y la segunda a las columnas
- *En memoria cada fila es un array*
- Ejemplo: memoria para `int [][] matriz = new int [4][3];`

Matrices: acceso y recorrido

- Acceso a elementos (ejemplo): `matriz[1][1]`
- Recorrido: for anidados

Matrices: acceso y recorrido

- Acceso a elementos (ejemplo): `matriz[1][1]`
- Recorrido: for anidados

```
int i,j;

int[][] matriz1 = new int [3][2];

int numeroInicial = 1;

for ( i = 0; i < matriz1.length; i++){
     for (j = 0; j < matriz1[i].length; j++){
       matriz1[i][j] = numeroInicial ;
       numeroInicial++;
     }
}
```

Matrices: recorrido `for` each

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

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

System.out.println();
for (int [] filas : matriz){
   for (int elemento: filas){
      System.out.print(elemento+" ");
   }
System.out.println();
}
```

Matrices: filas de distinta longitud

```
int [][] a;
a = new int [3][];

a[0] = new int [1];
a[1] = new int [3];
a[2] = new int [2];
```

Arrays multidimensionales

- Se pueden construir arrays de más de dos dimensiones

```
int multi[][][] = new int[3][3][3];
```

- Aunque la tendencia es usarlos poco para no complicar el código