

INTRODUCCIÓN A LA PROGRAMACIÓN

Semana 4

ING. MARTIN POLIOTTO

Docente a cargo del módulo

Agosto 2020



DIPLOMATURA EN

**NUEVAS
TECNOLOGÍAS**

SECRETARÍA DE
EXTENSIÓN
UNIVERSITARIA

SEU

UTN
Facultad Regional Córdoba

Ministerio de
PROMOCIÓN DEL EMPLEO
Y DE LA ECONOMÍA FAMILIAR

Ministerio de
CIENCIA Y
TECNOLOGÍA

CBA
ENTRE TODOS

Oficina de
Montevideo
Organización
de las Naciones Unidas
para la Educación,
la Ciencia y la Cultura

Semana 04

1. Arreglos

1.1 Estructura de datos

Cuando nos iniciamos en el mundo de la programación, uno de los conceptos más difíciles de entender son las estructuras de datos y las definiciones que usualmente se encuentran son un tanto enredadas. Supongamos un problema en el que necesitamos procesar un volumen importante de datos y que a dicho conjunto de datos lo necesitamos acceder varias veces para realizar diferentes consultas o cálculos. Una primera aproximación de solución podría ser pensar en utilizar un *ciclo de carga de datos*. El problema aquí radica en que cada vez que el ciclo termina una vuelta se carga un nuevo lote, y se pierde con ello el lote anterior. Piense por un momento que tiene manejar información de una agenda de contactos bajo este criterio.

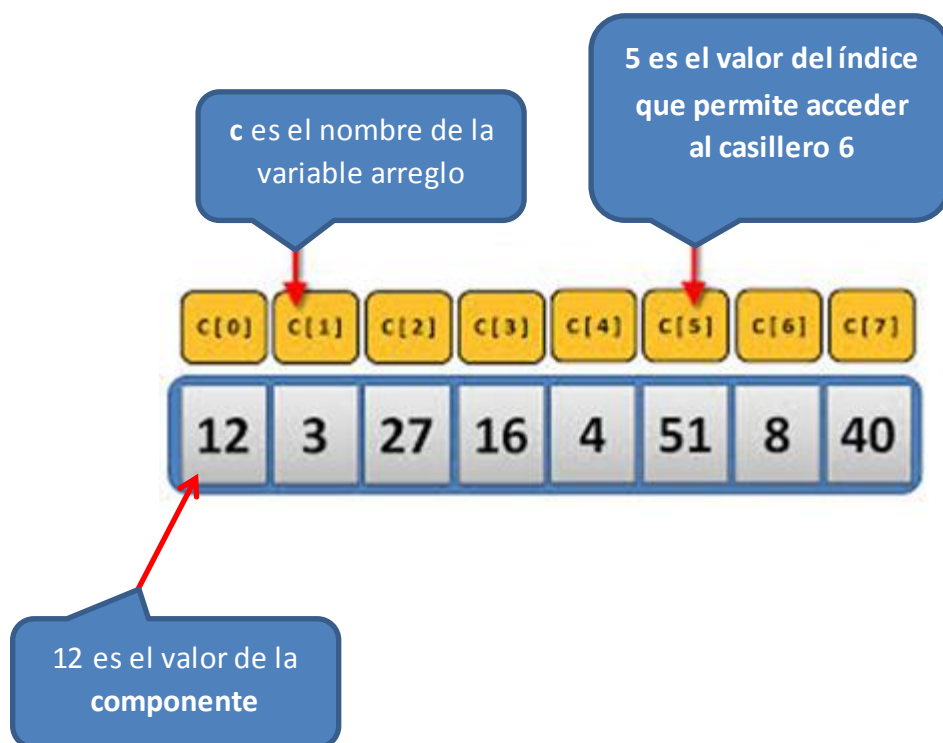
Para poder resolver esta situación necesitamos pensar de qué manera se podrían almacenar en memoria los datos de manera que su posterior acceso y procesamiento sea relativamente sencillo. Dicho en otras palabras, necesitamos hablar de **estructuras de datos**.

En la mayoría de los lenguajes de programación (incluido Java) existen dos tipos de variables: las **variables de tipo simple** o primitivo, cuya característica esencial es que **son capaces de almacenar solo un valor a la vez**, mientras que existen otras

variables que **pueden contener varios valores a la vez**. Esta segunda categoría de variable recibe el nombre de variable estructurada o simplemente **estructura de datos**.

1.2 Arreglos unidimensionales

Una de las estructuras más conocidas en el ámbito de la programación son los **arreglos unidimensionales**. Un arreglo (array) o vector es un conjunto de datos, todos del mismo tipo, almacenado secuencialmente en memoria. Cada valor o componente individual es asociado con un número entero llamado **índice**. Gráficamente:



En **Java**, los arreglos son estructuras estáticas cuya declaración es la siguiente:

```
int vector [];
```

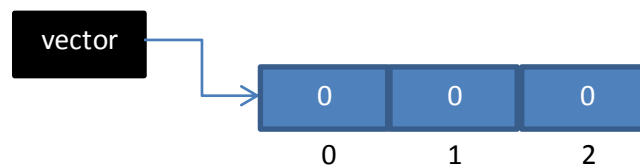
- Esa declaración es totalmente equivalente **int [] vector**. Es decir los corchetes pueden colocarse delante o detrás del identificador de variable.
- Técnicamente la declaración anterior se conoce como una referencia a un vector, que Java inicializará automáticamente en **null**.



- Para pedir memoria para las componentes es necesario utilizar el operador **new**:

`vector = new int[3];`

- Gráficamente lo anterior:



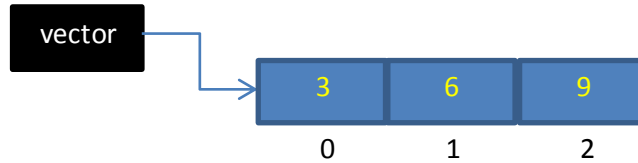
- Es posible de declarar y dimensionar la variable en la misma instrucción:

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

- Algunas consideraciones importantes sobre los vectores:
 - ✓ al ser estáticos, su tamaño queda definido en tiempo de compilación. Durante la ejecución del programa no es posible modificar su dimensión.
 - ✓ Notar que si el vector tiene 3 componentes la última casilla tiene índice **2** debido a que los índices siempre comienzan en 0.
 - ✓ Es posible *declarar*, dimensionar e inicializar sus componentes en una sola línea:

```
int vector[] = {3,6,9};
```

Resultando:



- Una vez creado el arreglo es posible conocer su cantidad de componentes o largo mediante:

```
vector.length; //cantidad de casillas
```

- Para acceder a cada casilla o componente:

```
vector[índice]
```

Por ejemplo para mostrar el contenido de la primera posición:

```
System.out.println(vector[0]);
```

Para modificar el valor de la última posición:

```
vector[2] = 10;
```

1.3 Algoritmos básicos

Las operaciones básicas más comunes que necesitamos aprender para manejar arreglos son:

- Recorrido
- Búsqueda
- Ordenamiento

1.3.1 Recorrido

Para comenzar a trabajar con vectores vamos a plantear el siguiente problema:

Se necesita cargar en un arreglo la cantidad de reproducciones diarias de un video de youtube en una semana de estudio. Luego mostrar en el contenido del vector una componente al lado de otra.

Nuestra clase Ejecutable quedaría:

```
public class Ejecutable {

    public static void main(String[] args) {

        Scanner entrada = new Scanner(System.in);

        int v[] = new int[7]; // 7 componentes, una por cada día de la semana

        // Recorrido de carga:

        for (int i = 0; i < 7; i++) {

            System.out.println("Ingrese cantidad de reproducciones día " + (i+1));

            v[i] = entrada.nextInt();

        }

        // Recorrido para mostrar el contenido.

        System.out.println("\nReproducciones semanales:");

        System.out.print("["); // print no agrega un salto de línea

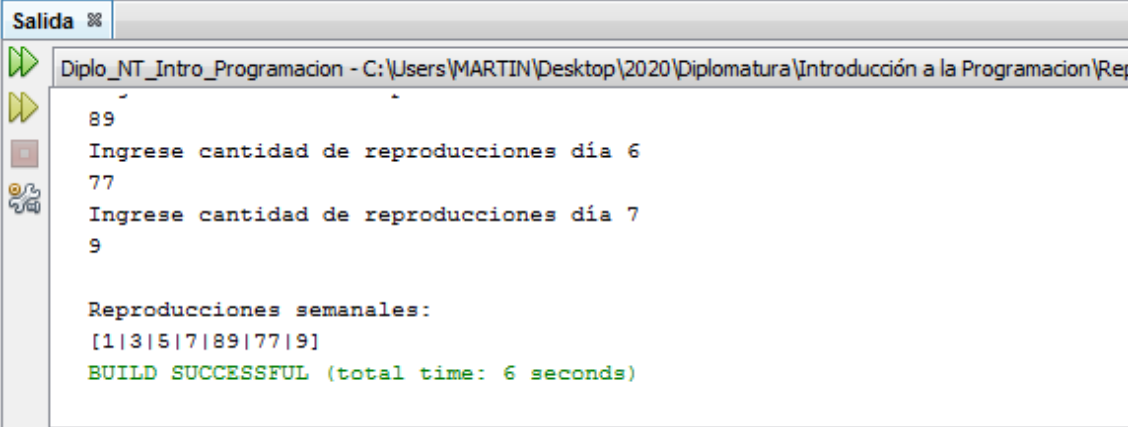
        for (int i = 0; i < v.length; i++) {

            if(i == 6)

                System.out.print(v[i]);

            else
```

```
        System.out.print(v[i]+ "|");  
  
    }  
  
    System.out.println("");  
  
}  
  
}
```



The screenshot shows a Java IDE's output window titled "Salida". The text inside the window is as follows:

```
Diplo_NT_Intro_Programacion - C:\Users\MARTIN\Desktop\2020\Diplomatura\Introducción a la Programacion\Rep  
89  
Ingrese cantidad de reproducciones día 6  
77  
Ingrese cantidad de reproducciones día 7  
9  
  
Reproducciones semanales:  
[1|3|5|7|89|77|9]  
BUILD SUCCESSFUL (total time: 6 seconds)
```

- Notar que para trabajar con arreglos, como se conoce de antemano su tamaño, utilizaremos una estructura de control **for**.
- Con el atributo **length** es posible recorrer todo un vector si necesidad de conocer su tamaño. Siempre el barrido será desde 0 hasta $\text{length} - 1$.

1.3.2 Búsqueda secuencial

Uno de los problemas más comunes en programación es determinar si determinado valor se encuentra presente entre las componentes de un vector. Para ejemplificar esta situación vamos a plantear el siguiente problema:

*Se necesita cargar en un arreglo unidimensional con **n** códigos de producto. Determinar si el código **cod** se encuentra en el arreglo. Si se encuentra indicar la posición. En caso contrario informar con un mensaje que el código no se encuentra presente.*

*Tanto **n** como **cod** son ingresados por teclado. Se deberá validar que **n** sea un número positivo.*

Nuestra clase Ejecutable quedaría:

```
public class Ejecutable {  
  
    public static void main(String[] args) {  
  
        int n, cod;  
  
        boolean encontrado = false; //bandera auxiliar  
  
        Scanner entrada = new Scanner(System.in);  
  
        do {  
  
            System.out.println("Ingrese una cantidad de códigos (>0)");  
  
            n = entrada.nextInt();  
  
        } while (n <= 0);  
  
        // Declaramos el arreglo de n componentes:  
  
        int v[] = new int[n];
```



```
//ciclo de carga:

for (int i = 0; i < n; i++) {

    System.out.println("Ingrese código de producto " + (i+1));

    v[i] = entrada.nextInt();

}

//Solicitamos el ingreso del código 'cod' buscado:

System.out.println("Ingrese el código de producto a buscar: ");

cod = entrada.nextInt();

//ciclo de búsqueda:

for (int i = 0; i < v.length; i++) {

    if(v[i] == cod){

        System.out.println("Producto encontrado!");

        System.out.println("Posición: " + i);

        encontrado = true;

        break;

    }

}

if(encontrado == false){

    System.out.println("El codigo: " + cod + " no se encuentra registrado!");

}

}

}
```

Diplo_NT_Intro_Programacion - C:\Users\MARTIN\Desktop\2020\Diplomatura\Introducción a la Programacion\Repo Git\Diplo_NT_Intro_Programacion

```
run:
Ingrese una cantidad de códigos (>0)
5
Ingrese código de producto 1
6
Ingrese código de producto 2
9
Ingrese código de producto 3
2
Ingrese código de producto 4
4
Ingrese código de producto 5
67
Ingrese el código de producto a buscar:
2
Producto encontrado!
Posición: 2
BUILD SUCCESSFUL (total time: 11 seconds)
```

- Al algoritmo utilizado se lo suele llamar **búsqueda secuencial**. Consiste básicamente en hacer una recorrido secuencial del vector con un ciclo for. En cada vuelta preguntamos si la componente **i** contiene el valor buscado. En tal caso guardamos la posición o la mostramos y cortamos el ciclo con la instrucción **break** (para no seguir buscando). Adicionalmente utilizamos una **variable centinela** para indicar que encontramos el valor deseado. Cuando terminamos el ciclo validamos si la bandera está encendida indicamos que se encontró el valor. Caso contrario podemos informar que el valor buscado no se encuentra en nuestro arreglo.
- Vale resaltar que se utiliza un ciclo **do-while** para validar que la cantidad de elementos a cargar sea positiva.
- El uso de la bandera permite detectar si el elemento buscado existe o no en el arreglo.
- Si el elemento buscado se encuentra más de una vez el proceso de búsqueda devolverá la posición de la primer ocurrencia.

1.3.3 Ordenamiento

Otra operación muy común cuando se trabaja con arreglos unidimensionales es la de ordenar sus elementos, ya sea de manera ascendente o descendente. Existen muchos métodos para proceder al ordenamiento de un arreglo de n componentes. Solo para introducir el problema nos concentraremos en comprender el modo de funcionamiento del más sencillo: **Ordenamiento de Selección**. Si se desea ordenar el arreglo de menor a mayor usando selección directa, la idea central es la adaptación y repetición sistemática de este simple algoritmo (suponemos que el arreglo a ordenar es v y que ya está cargado con valores de tipo int):

```
int aux, i = 0;

for(int j = i + 1; j < n; j++){

    if(v[i] > v[j]){

        aux = v[i];

        v[i] = v[j];

        v[j] = aux;

    }

}
```

El algoritmo anterior recorre el arreglo buscando el valor menor del mismo, y al terminar lo deja en la casilla indicada por la variable i (que en este caso vale cero). Se comienza asumiendo que el menor está en casilla la $i = 0$, y se recorre con j desde el valor $i + 1$. Cada valor en la casilla j se compara con el valor en la casilla i . Si el valor en $v[i]$ resulta mayor que $v[j]$ entonces los valores se intercambian, y así se prosigue hasta que el ciclo controlado por j termina. El algoritmo no ordena el

arreglo: sólo garantiza que el menor valor será colocado en la casilla con índice $i = 0$. Pero entonces sólo basta un pequeño cambio para que se ordene todo el arreglo: hacer que la variable i modifique su valor con otro ciclo, comenzando desde cero y terminando antes de llegar a la última casilla:

```
int aux, i;

for(i = 0; i < v.length-1; i++){

    for(int j = i + 1; j < n; j++){

        if(v[i] > v[j]){ //intercambiar

            aux = v[i];

            v[i] = v[j];

            v[j] = aux;

        } // fin if

    } // fin for-2

} // fin for-1
```

- Notar que el primer ciclo comienza en 0 y termina en el largo del arreglo – 1.
- El segundo ciclo utiliza una variable de ciclo j que se mueve desde $j = i + 1$ hasta el largo del vector.
- Cuando se cumple la condición de encontrar un nuevo menor, si realiza un proceso de intercambio entre las componentes.
- Si bien existen métodos más eficientes para ordenar un arreglo por simplicidad y para los objetivos del módulo solo se abordará este método.