

TECNICATURA
UNIVERSITARIA
EN PROGRAMACIÓN
UTN-FRC



UTN 
Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN
PROGRAMACIÓN

LABORATORIO DE COMPUTACIÓN III

Unidad Temática 2:
Arreglos y Colecciones

Material de Estudio

2^{do} Año - 3^{er} Cuatrimestre

2020



V.0.1

Índice

1. ARREGLOS	2
Arreglos	2
2. COLECCIONES	9
Características de los arreglos	9
Framework de Colecciones	9
Uso de arraylist.....	11
Generics	13
Composición	14
BIBLIOGRAFÍA	26

1. ARREGLOS

Arreglos

Cuando se requiere almacenar un conjunto de datos de un tamaño considerable, la opción de guardarlos en variables resulta impráctica. En una situación en la que se necesiten guardar solamente 10 datos, ya no es una opción razonable declarar 10 variables, cargarlas por separado y luego manipularlas. Lógicamente con tamaños mayores, del orden de las centenas o miles de datos, esta opción resulta virtualmente imposible.

En tales escenarios se necesita poder guardar el conjunto de datos en una unidad conceptual que los agrupe, denominada estructura de datos. La estructura de datos más simple es el arreglo.

Un arreglo es una estructura de datos que permite con un único identificador almacenar múltiples datos todos del mismo tipo. Los arreglos pueden imaginarse como una secuencia lineal de casilleros, cada uno de los cuales funciona como una variable, por tanto, el arreglo puede almacenar tantos datos como casilleros posea. Pero como el arreglo sólo tiene un nombre, para identificar a cada casillero individual se los enumera con números naturales llamados índices. El primer índice lleva el número 0 y los siguientes con 1, 2, 3, etc. A causa de lo anterior, el último índice es equivalente al tamaño del arreglo $- 1$.

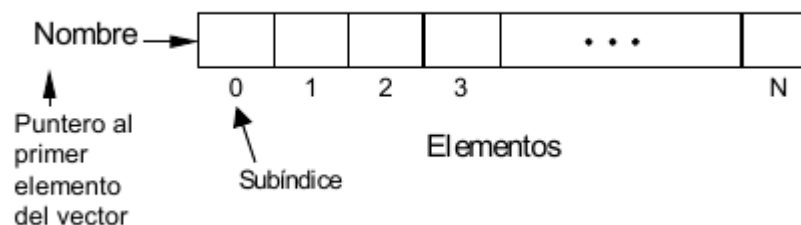


Gráfico 1: Elaboración propia

Los arreglos poseen un tamaño estático el cual es asignado antes de utilizarlo. Tal tamaño no puede ser modificado por lo tanto debe prestarse especial atención en el momento de la creación para no seleccionar un tamaño inadecuado. Si se lo crea con una cantidad de casilleros menor a la cantidad de datos que se necesitan guardar, el arreglo se llena y algunos datos quedan afuera de la estructura. De modo contrario si se lo crea demasiado grande, por un lado se desperdicia memoria en los casilleros que no se utilicen y por otro aumenta la complejidad de los algoritmos que deben verificar continuamente cuales casilleros están ocupados y cuáles no.

Existe la posibilidad de crear arreglos cuyos elementos se identifiquen mediante un par ordenado de números, tomando conceptualmente una forma de tabla de doble entrada. Tradicionalmente se denomina vectores a los arreglos con índices simples y matrices a los arreglos con índices de pares ordenados. En esta materia únicamente se analizarán y practicarán los arreglos simples, y durante el texto se utiliza la denominación de vector como sinónimo.

Para mayor detalle puede consultarse Ceballos (2010) y Corso (2012).

Sintaxis

Para utilizar un vector se requiere declararlo, luego asignarle tamaño y finalmente asignarle valores a sus casilleros.

Declaración

La declaración de un vector es similar a la de una variable simple. Únicamente requiere agregar un par de corchetes vacíos antes o después del nombre.

```
tipo[] nombre;  
tipo nombre[];
```

Primero se indica el tipo de los elementos del vector, que pueden ser tipos simples como variables o nombres de clases. A continuación se especifica el identificador, el cual debe seguir las mismas reglas de nomenclatura que las variables. Los corchetes indican que se desea un arreglo en lugar de una variable.

Los siguientes son todas declaraciones válidas:

```
int[] v;  
float[] temperatura;  
Persona[] personas;
```

La primera línea declara un vector de elementos de tipo int; la segunda, un vector de elementos de tipo float; y la tercera un vector de objetos Persona. En el momento de la declaración no se especifica el tamaño del vector, esto se realiza en el momento de la creación.

Dado que los corchetes pueden estar indicados a continuación del nombre, las siguientes declaraciones son válidas y equivalentes a las anteriores:

```
int v[];  
float temperatura[];  
Persona personas[];
```

Creación

Al declarar el arreglo no se indica el tamaño, por lo tanto no tiene lugar para almacenar datos ni ocupa memoria. Para poder utilizarlo se lo debe crear o construir. La creación sí reserva memoria para todos los casilleros que indiquen. La sintaxis de la creación es la siguiente:

```
nombre = new tipo[tamaño];
```

El tamaño debe ser un valor constante o variable de tipo int. Si se crea un vector con el tamaño indicado por una variable, el mismo poseerá tantos casilleros como el valor de la variable en el momento de la creación, pero esto no significa que si posteriormente la variable cambia de valor el vector pueda cambiar de tamaño.

Para darle tamaño a los vectores declarados anteriormente se puede ejecutar las siguientes líneas:

```
v = new int[10];  
temperatura = new float[31];  
personas = new Persona[25];
```

Muy habitualmente se declara y crea el vector en la misma línea:

```
tipo[] nombre = new tipo[tamaño];  
tipo nombre[] = new tipo[tamaño];
```

Por ejemplo:

```
int[] v = new int[10];  
float[] temperatura = new float[31];  
Persona[] personas = new Persona[25];
```

La utilidad de indicar el tamaño mediante una variable permite calcular o pedirle al usuario el tamaño exacto que se requiera, en la porción de código presentada a continuación se le solicita al usuario la cantidad de datos que quiere ingresar y con esa cantidad se crea el vector con el tamaño exacto.

```
int cantidad;  
System.out.println("Cuántos números desea ingresar: " );  
cantidad = sc.nextInt();
```

```
int[] v = new int[cantidad];
```

Inicialización

Al crear los arreglos todos sus casilleros quedan con un valor vacío que depende del tipo de datos (0 para los numéricos, cadena vacía para los String, null para los objetos, etc.) Si se necesita que el arreglo posea valores iniciales se los puede indicar en forma explícita separados por comas y envueltos en un par de llaves.

```
float[] temperatura = (10.2f, 12.3f, 3.4f, 14.5f, 15.6f, 16.7f);
```

En este caso no se utiliza new ni se indica el tamaño, el arreglo queda con tantos casilleros como valores se hayan indicado.

Acceso

Cuando se necesita acceder a un casillero individual del arreglo para asignar o consultar el valor del mismo, se escribe el nombre del arreglo y a continuación un par de corchetes indicando dentro de los mismos el índice del casillero que se necesita.

Las siguientes líneas presentan distintos accesos individuales, en ellas se evidencia que cada casillero funciona como una variable individual.

```
int[] v = new int[10];  
int i = 0, s=0;  
s = v[1] + v[9];  
i=5;  
v[i]++;  
v[i+1] = v[i];
```

De la misma forma que al crear al vector, para acceder a sus casilleros el índice puede especificarse con un valor constante, una variable o una expresión, siempre de tipo entero.

No es posible acceder a un elemento con un índice menor a 0 o mayor o igual al tamaño, ya que por definición no hay ningún casillero con tales índices. Si se intenta esta operación se recibe un error (IndexOutOfBoundsException) que interrumpe la ejecución del programa.

```
int[] v = new int[10];  
v[11] = 2334; //Excepción: los índices van de 0 a 9, no hay casillero con índice 11.
```

Para evitar la posibilidad de acceder a un índice fuera de rango, se puede verificar la longitud del vector mediante el atributo `length` que poseen por los vectores, de la siguiente manera:

```
int n = v.length; // número de elementos del vector v
```

Recorrido secuencial

Es muy habitual necesitar recorrer un arreglo para efectuar alguna tarea en todos los elementos almacenados. Para ello el algoritmo más habitual y sencillo es el de utilizar un ciclo `for`, en el que la variable contadora de vueltas se inicie en 0 y finalice en el tamaño del arreglo. Dentro del ciclo, cualquier operación que se realice con la variable del ciclo `for` se va a repetir en cada uno de los elementos del arreglo.

Por ejemplo, luego de ejecutar el siguiente código, el arreglo `m` tendrá todas sus posiciones asignadas con 5.

```
int[] v = new int[100];
```

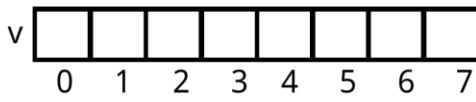
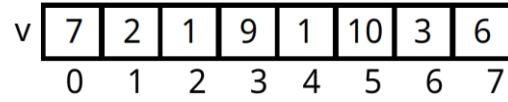
Declaración	
<pre>tipo nombre[]; tipo []nombre;</pre>	<pre>int []v; int v[];</pre> <p style="text-align: center;">V</p>
Creación	
<pre>nombre = new tipo[tamaño];</pre>	<pre>v = new int[8];</pre> 
Inicialización	
<pre>nombre = {valor1, valor2, ..., valorn};</pre>	<pre>v = {7, 2, 1, 9, 10, 3, 6};</pre> 
Acceso	
<pre>nombre[indice]</pre>	
Recorrido secuencial	
<pre>for (i = 0; i < nombre.length; i++) nombre[i]...;</pre>	<pre>for (i = 0; i < v.length; i++) System.out.println(v[i]);</pre>

Tabla 1: Elaboración propia

```
//inicializa en 5
for (int i=0; i < m.length; i++) {
    v[i] = 5;
}
```


Ejercitación

1. Desarrollar un programa que ingrese 10 números en un arreglo. Luego de la carga calcular y mostrar:

- a) Suma de todos
- b) Promedio de todos
- c) Cantidad de números mayores a 5

```
package arregloenteros;

import java.util.Scanner;

public class ArregloEnteros {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int numeros[] = new int[10];

        System.out.println("Ingrese 10 numeros");

        for (int i = 0; i < numeros.length; i++) {
            numeros[i] = sc.nextInt();
        }

        System.out.println("Los números ingresados son: ");
        for (int i = 0; i < numeros.length; i++) {
            System.out.print(numeros[i] + ",");
        }

        int suma = 0, c = 0;

        for (int i = 0; i < numeros.length; i++) {
            suma += numeros[i];
            if (numeros[i] > 5) {
                c++;
            }
        }

        float promedio = (float) suma / numeros.length;

        System.out.println("\nLa suma es: " + suma);
        System.out.println("El promedio: " + promedio);
        System.out.println("Hay " + c + " números mayores a 5");
    }
}
```

2. COLECCIONES

Características de los arreglos

Los arreglos unidimensionales son las estructuras de datos más simples y efectivos para almacenar conjuntos de datos, tanto de tipos simples como de objetos. Sin embargo poseen algunos inconvenientes que deben ser considerados:

- Tamaño fijo: El tamaño de los vectores debe ser especificado durante la creación con el operador `new` y el mismo no puede ser cambiado con posterioridad. Es decir que de alguna manera debe conocerse con antelación la cantidad de datos que se van a ingresar en el mismo. Cuando no se conoce los vectores son inconvenientes; por ejemplo porque el usuario va ingresando datos a medida que los obtiene y tampoco puede saber con antelación la cantidad. La única opción es la de crearlo muy grande desperdiciando memoria o de estimar el tamaño, con el riesgo de que resulte más chico que lo que se necesita y el programa no funcione.
- Mismo tipo: Los vectores son definidos con un tipo de datos y sólo pueden guardar datos de ese único tipo. Aun siendo discutible su necesidad, si hace falta guardar datos de diferentes tipos esto resulta imposible. La única variante es la de los vectores de objetos. En este caso sólo se pueden guardar objetos que sean instancias de la clase de la definición o de cualquiera de sus derivadas.
- Algoritmos: El lenguaje no ofrece ningún algoritmo que opere sobre vectores. Es decir que cualquier operación que se deba realizar (inserción, búsqueda, recorrido, ordenamientos, etc.) debe ser provista por el programador. Si bien estas operaciones suelen ser fáciles de programar no están incluidas en la biblioteca estándar de java. La única excepción es la del ordenamiento, que sí está disponible mediante el método `Arrays.sort`, pero que aplica un único algoritmo denominado QuickSort. Si se desea ordenar mediante otros algoritmos también deben ser programados.

Framework de Colecciones

Para ofrecer una alternativa que solucione todos los inconvenientes de los arreglos Java ofrece un gran conjunto de clases denominado Framework de Colecciones (Collections Framework). El uso de estas clases es relativamente simple pero al mismo tiempo ofrecen soluciones para muchos problemas relacionados con las estructuras de datos, tales como algoritmos muy rápidos de búsqueda, ordenamientos y manipulación de los datos almacenados.

El conjunto de clases es muy grande y excede el alcance de la materia, en esta ocasión sólo se analizará el uso de la colección más simple que se incluye. Esta colección se caracteriza por ser muy fácil de usar y permitir acceso mediante índices y recorridos con la instrucción `for` mejorada, convirtiéndose en la alternativa más adecuada para reemplazar a los arreglos.

Otras clases de la librería de colecciones permiten operaciones más avanzadas, tales como inserción ordenada, búsquedas rápidas, acceso por datos clave en lugar de índices y otras.

Clase ArrayList (Extraído de Corso (2010))

Podemos definir una lista como un conjunto de elementos los cuales ocupan una determinada posición de tal forma que cada elemento tendrá un único sucesor y un único antecesor, menos el primero y el último.

El tamaño de estas estructuras es dinámico y va cambiando a lo largo del programa: se añaden elementos al final de la lista, se insertan en una determinada posición, se eliminan elementos, etc.

Las operaciones generales que se llevan a cabo en las listas son las siguientes:

Insertar, eliminar o localizar un elemento.

Determinar su tamaño: número de sus elementos.

Recorrer la lista.

Unir dos o más listas en una sola.

Dividir una lista en varias sublistas.

Copiar la lista.

Borrar la lista.

Los métodos más importantes de la clase ArrayList son los siguientes:

Constructores

ArrayList()

Construye una lista vacía.

ArrayList(Collection c)

Construye una lista que contiene los elementos de la colección, en el orden en que son devueltos al recorrerla.

ArrayList(int capacidadInicial)

Construye una lista vacía con la capacidad inicial dada. Esta capacidad inicial no limita el tamaño, si durante la ejecución del programa se necesita agregar más datos que la capacidad inicial, la lista crece de tamaño automáticamente.

boolean add(Object o): Añade un elemento al final de la lista.

void add(int in, Object o): Inserta el elemento dado en el lugar especificado.

boolean addAll(Collection c): Añade todos los elementos de la colección al final de la lista, en el orden en que son devueltos al recorrerla

boolean addAll(int in, Collection c): Inserta todos los elementos de la colección al final de la lista, comenzando por la posición dada.

void clear(): Elimina todos los elementos de la lista.

boolean contains(Object o). Devuelve true si la lista contiene el elemento.

Object get(int in): Devuelve el elemento de la lista que ocupa el lugar especificado.

int indexOf(Object o): Devuelve –1 si el elemento no pertenece a la lista, y el lugar de la primera aparición si el elemento está en la lista.

boolean isEmpty(): Devuelve true si la lista está vacía.

int lastIndexOf(Object o): Devuelve –1 si el elemento no pertenece a la lista, y el lugar de la última aparición si el elemento está en la lista.

Object remove(int in): Elimina el elemento de la posición dada.

Object set(int in, Object o): Reemplaza el elemento en la posición dada por el nuevo elemento.

int size(): Devuelve el número de elementos disponibles.

Object [] toArray(): Devuelve una tabla conteniendo todos los elementos de la lista en el mismo orden.

Uso de arraylist

De una manera similar a los arreglos, para poder almacenar datos en un ArrayList se requiere declarar una variable de esta clase, crearla con new y luego agregar tantos datos como se necesite. Posteriormente se puede acceder a los mismos mediante índice o recorrerlo con for o for mejorado.

Declaración	
<code>ArrayList lista;</code>	<code>ArrayList listaPersonas;</code>
Creación	
<code>lista = new ArrayList();</code>	<code>listaPersonas = new ArrayList();</code>
Inserción	
<code>lista.add(dato);</code>	<code>Persona p = new Persona("Juan","Perez");</code> <code>Persona q = new Persona("Ana","Gomez");</code> <code>listaPersonas.add(p);</code> <code>listaPersonas.add(q);</code>
Acceso por índices	
<code>lista.get(indice);</code>	
Tamaño	
<code>lista.size()</code>	
Recorrido con for	
<code>for(int i = 0; i = lista.size(); i++)</code> <code>{</code> <code> lista.get(i);</code> <code>}</code>	<code>for(int i = 0; i = listaPersonas.size(); i++)</code> <code>{</code> <code> Persona p = (Persona) listaPersonas.get(i);</code> <code> System.out.println(p.getNombreCompleto());</code> <code>}</code>
Recorrido con for mejorado	
<code>for(Object obj: lista)</code> <code>{</code> <code> Clase item = (Clase) obj;</code> <code>}</code>	<code>for(Object obj: listaPersonas)</code> <code>{</code> <code> Persona p = (Persona) obj;</code> <code> System.out.println(p.getNombreCompleto());</code> <code>}</code>

Tabla 2: Elaboración propia

Generics

Las clases de colecciones reciben cualquier objeto de la clase `Object` o de cualquiera de sus derivadas. Por este motivo se pueden mezclar dentro de misma colección objetos de diferentes tipos. Sin embargo, suele ser inconveniente mezclar tipos dado que al recorrer los objetos de la colección no se puede saber con facilidad qué métodos se pueden invocar de cada objeto porque estos son diferentes por cada clase. Además sería necesario realizar muchas condiciones con `instanceof` para determinar de qué clase es cada objeto.

Otra consecuencia de esto es que todos los métodos que permiten acceder a los elementos almacenados únicamente pueden devolver referencias a `Object`. Aún si se ingresaron todos objetos de la misma clase, el compilador no puede garantizar que no se hayan mezclado y por lo tanto el método `get` y la instrucción `for` mejorada devuelven `Object`.

Para mejorar esta situación existe una característica de lenguaje denominada Generics que permite parametrizar las colecciones con el **nombre de una clase**. Este nombre de clase permite al compilador restringir que todas las operaciones de la colección se realicen con esa clase y sus derivadas pero no con otras.

Así, si definimos un `arraylist` con un parámetro de clase en particular, tanto el método `add()` como el `get()` van a recibir y devolver referencias a objetos de esa clase y el compilador sí va a poder controlarlo.

Declaración	
<code>ArrayList<Clase> lista;</code>	<code>ArrayList<Persona> listaPersonas;</code>
Creación	
<code>lista = new ArrayList<>();</code>	<code>listaPersonas = new ArrayList<>();</code>
Recorrido con for	
<code>for(int i = 0; i = lista.size(); i++) { lista.get(i); }</code>	<code>for(int i = 0; i = listaPersonas.size(); i++) { System.out.println(listaPersonas.get(i).getNombreCompleto()); }</code>
Recorrido con for mejorado	
<code>for(Clase item: lista) { ... item ...; }</code>	<code>for(Persona p: listaPersonas) { System.out.println(p.getNombreCompleto()); }</code>

Tabla 3: Elaboración propia

Composición

Una situación muy habitual es la de representar una relación de composición utilizando arreglos o colecciones. En una relación de composición un objeto se dice que está compuesto o que contiene objetos de otra clase.

Por ejemplo un Curso está compuesto de muchos Alumnos. En este caso se requiere programar una clase Alumno cuyas instancias representen cada una a cada alumno existente y una clase Curso que represente al conjunto de alumnos que conforman un curso.

En estas situaciones se identifica la clase Contenedora (Curso) y la clase Contenida (Alumno) y de entre todas las formas posibles de programarlas las más sencillas implican que en la clase Contenedora se incluya un vector o una colección, la cual va a contener una o más instancias de la clase Contenida.

Composición con arreglos

- En la clase contenedora se declara un arreglo de la clase contenida
- En la clase contenedora se agrega un método para insertar nuevos objetos de la clase contenida
- En el constructor de la clase contenedora se necesita un parámetro que indique la cantidad máxima de objetos contenidos

Clase contenedora

- Es una clase común, con atributos y métodos habituales
- Debe poseer al menos un arreglo donde se almacenan los objetos contenidos
- No debe poseer métodos get y set para el arreglo

Clase contenida

- Es una clase común, con atributos y métodos habituales
- No va a poseer ninguna referencia a la clase contenedora

Declaración

```
public class Contenedora {  
  
    private Contenida[] v;  
  
}
```

Constructor

```
public Contenedora (parámetros, int cantidad) {  
  
    ...asignación de atributos...  
    v = new Contenida[cantidad];  
}
```

- El parámetro cantidad indica el tamaño del arreglo de objetos contenidos.
- El arreglo debe estar creado antes de agregar elementos, por lo tanto se lo crea durante la construcción.

Inserción

- La clase contenedora debe poseer un método para agregar un nuevo elemento contenido
- El método de inserción debe recibir como parámetro un objeto de la clase contenida
- Por simplicidad el método no va a devolver nada
- Para insertar se necesita recorrer el arreglo buscando el primer elemento que sea nulo
- Cuando se lo encuentra, se asigna el nuevo objeto en ese lugar
- Es imprescindible interrumpir el ciclo, si no, el nuevo objeto va a ocupar todo el arreglo porque se lo asignaría a todos los casilleros vacíos.

```
public void agregar(Contenida nuevo){  
    for (int i=0; i < v.length; i++)  
        if(v[i] == null) {  
            v[i] = nuevo;  
            break;  
        }  
}
```

Recorridos

- Cuando se necesite obtener información de los elementos almacenados, debe recorrerse el arreglo con for o for mejorado
- Para ello se agrega a la clase contenedora un método por cada resultado que se requiera
- Los resultados nunca deben mostrarse en esta clase, únicamente deben ser retornados.
- Si necesita un dato externo (por ejemplo, para una búsqueda) debe ser recibido por parámetro
- En el ciclo que recorra el arreglo, debería verificarse que no haya elementos nulos.

```
public int recorrido() {  
    for (Contenida c: v) {  
        if (c != null) {  
            operaciones con c  
        }  
    }  
    return ...;  
}
```

Ejercitación

2. Desarrollar un programa que permita administrar la información de un curso y de sus alumnos. De cada curso se conoce su nombre y el listado de sus alumnos. Por otro lado, de cada alumno se conoce su nombre, su legajo y su promedio.

El programa deberá solicitar por teclado los datos del curso y la cantidad de alumnos inscriptos al mismo. A continuación debe solicitar todos los datos de los alumnos. Finalmente se debe presentar al usuario la siguiente información:

- a) Listado de alumnos
- b) Promedio general del curso
- c) Cantidad de alumnos con promedio mayor a 8.

```
package curso;  
  
public class Alumno {  
    private String nombre;  
    private int legajo;  
    private int[] notas;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getLegajo() {  
        return legajo;  
    }  
}
```

```
public void setLegajo(int legajo) {
    this.legajo = legajo;
}

public Alumno(String nombre, int legajo, int cantidadNotas) {
    this.nombre = nombre;
    this.legajo = legajo;

    this.notas = new int[cantidadNotas];
    for (int i = 0; i < cantidadNotas; i++) {
        this.notas[i] = -1;
    }
}

public void agregarNota(int nota)
{
    for (int i = 0; i < notas.length; i++) {
        if(notas[i] == -1)
            notas[i] = nota;
    }
}

public float getPromedio()
{
    int total = 0, cantidad = 0;
    for (int i = 0; i < notas.length; i++)
    {
        if(notas[i] > -1)
        {
            total += notas[i];
            cantidad++;
        }
    }
    return (float)total/cantidad;
}

@Override
public String toString() {
    return "Alumno " + nombre + " - Legajo: " + legajo + " - Promedio: " + getPromedio();
}

}

package curso;

public class Curso {

    private String nombre;
    private Alumno[] alumnos;

    public String getNombre() {
```

```
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Curso(String nombre, int cantidad) {
        this.nombre = nombre;
        this.alumnos = new Alumno[cantidad];
    }

    public void agregarAlumno(Alumno nuevo) {
        for (int i = 0; i < alumnos.length; i++) {
            if (alumnos[i] == null) {
                alumnos[i] = nuevo;
                return;
            }
        }
    }

    public float promedioGeneral() {
        float suma = 0f;
        int c = 0;
        for (int i = 0; i < alumnos.length; i++) {
            if (alumnos[i] != null) {
                c++;
                suma += alumnos[i].getPromedio();
            }
        }
        return suma / c;
    }

    public int cantidadMayor8() {
        int c = 0;
        for (int i = 0; i < alumnos.length; i++) {
            if (alumnos[i] != null && alumnos[i].getPromedio() > 8) {
                c++;
            }
        }
        return c;
    }

    public String listadoOpcionString()
    {
        String listado = "";
        for (int i = 0; i < alumnos.length; i++) {
            if(alumnos[i] != null)
            {
                listado += alumnos[i].toString() + "\n";
            }
        }
    }
}
```

```
        }
        return listado;
    }

    public String listadoOpcionStringForEach()
    {
        String listado = "";
        for (Alumno alumno : alumnos) {
            if (alumno != null) {
                listado += alumno.toString() + "\n";
            }
        }
        return listado;
    }
}

package curso;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String nombreCurso;
        int cantidadAlumnos;

        System.out.println("Ingrese el nombre del curso");
        nombreCurso = sc.next();
        System.out.println("Ingrese la cantidad de alumnos");
        cantidadAlumnos = sc.nextInt();

        Curso c = new Curso(nombreCurso, cantidadAlumnos);

        for (int i = 0; i < cantidadAlumnos; i++) {
            System.out.println("Ingrese el nombre del alumno");
            String nombre = sc.next();
            System.out.println("Ingrese el legajo");
            int legajo = sc.nextInt();
            System.out.println("Ingrese la cantidad de notas");
            int notas = sc.nextInt();

            Alumno nuevo = new Alumno(nombre, legajo, notas);

            System.out.println("Ingrese " + notas + " notas");
            for (int j = 0; j < notas; j++) {
                System.out.println(j + " ");
                int nota = sc.nextInt();
                nuevo.agregarNota(nota);
            }
        }
    }
}
```

```
        c.agregarAlumno(nuevo);
    }

    System.out.println("Listado de alumnos ingresados");
    System.out.println(c.listadoOpcionStringBuilderForEach());
    System.out.println("El promedio general del curso es: " + c.promedioGeneral());
    System.out.println("Hay " + c.cantidadMayor8() + " alumnos con promedio > 8");
}
}
```

Composición con colecciones

Si en lugar de utilizar un arreglo para representar la composición se lo hace con una colección, por ejemplo un arraylist, la mayoría de los métodos se simplifican. Inicialmente el constructor ya no requiere un parámetro indicando la cantidad de elementos contenidos ya que los arraylist no necesitan una indicación del tamaño para ser creados.

Por otro lado, las inserciones no necesitan el recorrido para buscar lugar libre como en el caso de los arreglos, una simple llamada al método add logra insertar un nuevo elemento sin necesidad de ningún otro control.

De la misma manera, los recorridos con for o for mejorado no requieren verificar nulos ya que nunca se va a insertar un nulo y no se recorre más allá de la cantidad de elementos insertados, como sí ocurre con un arreglo si se insertaron menos elementos que la capacidad del mismo.

Declaración

```
public class Contenedora {

    private ArrayList<Contenida> lista;

}
```

Constructor

```
public Contenedora (parámetros) {
    ...asignación de atributos...
    lista = new ArrayList<>();
}
```

Inserción

```
public void agregar(Contenida nuevo){
    lista.add(nuevo);
}
```

```
}
```

Recorridos

```
public int recorrido() {  
    for (Contenida c: v) {  
        operaciones con c  
    }  
    return ...;  
}
```

Ejercitación

3. Modificar el ejercicio de Cursos y Alumnos reemplazando el uso de arreglos por ArrayList y revisando todos los métodos afectados.

```
package curso;  
  
import java.util.ArrayList;  
  
public class Alumno {  
    private String nombre;  
    private int legajo;  
    private ArrayList notas;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getLegajo() {  
        return legajo;  
    }  
  
    public void setLegajo(int legajo) {  
        this.legajo = legajo;  
    }  
  
    public Alumno(String nombre, int legajo) {  
        this.nombre = nombre;  
        this.legajo = legajo;  
        this.notas = new ArrayList();  
    }  
}
```

```
        public void agregarNota(int nota)
        {
            notas.add(nota);
        }

        public float getPromedio()
        {
            int total = 0;
            for (Object o : notas) {
                Integer n = (Integer)o;
                total += n;
            }
            return (float)total/notas.size();
        }

        @Override
        public String toString() {
            return "Alumno " + nombre + " - Legajo: " + legajo + " - Promedio: " + getPromedio();
        }
    }

package curso;

import java.util.ArrayList;

public class Curso {

    private String nombre;
    private ArrayList alumnos;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Curso(String nombre) {
        this.nombre = nombre;
        this.alumnos = new ArrayList();
    }

    public void agregarAlumno(Alumno nuevo) {
        alumnos.add(nuevo);
    }
}
```



```
        public float promedioGeneral() {
            float suma = 0f;
            for (Object o : alumnos) {
                Alumno a = (Alumno) o;
                suma += a.getPromedio();
            }
            return suma / alumnos.size();
        }

        public int cantidadMayor8() {
            int c = 0;
            for (Object o : alumnos) {
                Alumno a = (Alumno) o;
                if (a.getPromedio() > 8) {
                    c++;
                }
            }
            return c;
        }

        public String listadoOpcionStringBuilderForEach() {
            StringBuilder sb = new StringBuilder();
            for (Object o : alumnos) {
                sb.append(o.toString());
                sb.append("\n");
            }
            return sb.toString();
        }
    }

package curso;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String nombreCurso;
        int cantidadAlumnos;

        System.out.println("Ingrese el nombre del curso");
        nombreCurso = sc.next();
        System.out.println("Ingrese la cantidad de alumnos");
        cantidadAlumnos = sc.nextInt();

        Curso c = new Curso(nombreCurso);
```

```
        for (int i = 0; i < cantidadAlumnos; i++) {
            System.out.println("Ingrese el nombre del alumno");
            String nombre = sc.next();
            System.out.println("Ingrese el legajo");
            int legajo = sc.nextInt();
            System.out.println("Ingrese la cantidad de notas");
            int notas = sc.nextInt();

            Alumno nuevo = new Alumno(nombre, legajo);

            System.out.println("Ingrese " + notas + " notas");
            for (int j = 0; j < notas; j++) {
                System.out.println(j + " ");
                int nota = sc.nextInt();
                nuevo.agregarNota(nota);
            }

            c.agregarAlumno(nuevo);
        }

        System.out.println("Listado de alumnos ingresados");
        System.out.println(c.listadoOpcionStringBuilderForEach());
        System.out.println("El promedio general del curso es: " + c.promedioGeneral());
        System.out.println("Hay " + c.cantidadMayor8() + " alumnos con promedio > 8");
    }
}
```

BIBLIOGRAFÍA

- Ceballos Sierra, F. (2010). *Java 2. Curso de Programación*. 4ta Edición. Madrid, España. RA-MA Editorial.
- Corso, C; Colaccioppo, N. (2012). “*Apunte teórico-práctico de Laboratorio de Computación III*”. Córdoba, Argentina. Edición digital UTN-FRC.



Atribución-NoComercial-SinDerivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterarse su contenido, ni se comercializarse. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Regional Córdoba (2020). Material para la Tecnicatura en Programación Semipresencial de Córdoba. Argentina.