6. ARRAYS DINÁMICOS

Un array dinámico (arreglo dinámico), es un arreglo de elementos que crece o decrece dinámicamente conforme se adicionan o eliminan elementos. Por lo general esta clase es suministrada como librería estándar por muchos lenguajes de programación, entre ellos java.

Si creamos un arreglo normal de 10 posiciones, no será posible para este arreglo adicionar una nueva posición una vez el arreglo haya sido utilizado en su totalidad. Por lo tanto, si queremos agregar más elementos, tendríamos que crear un arreglo de mayor tamaño y copiar allí inicialmente los elementos que tenemos en nuestro arreglo inicial y posteriormente agregar los nuevos elementos. En la figura siguiente podemos ver de forma gráfica el proceso de adicionar más elementos a un arreglo de tipo entero y de tamaño 10 al cual deseamos agregarle 5 posiciones más.

Arreglo A

1	2	3	4	5	6	7	8	9	10

Arreglo B

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Copiando el Arreglo A en el Arreglo B:

Arreglo B

1	2	3	4	5	6	7	8	9	10	0	0	0	0	0

Se tendrían 5 posiciones disponibles para agregar nuevos elementos. Para el caso del Arreglo B, estas posiciones disponibles están llenas con el valor 0 ya que el Arreglo B es de tipo entero.

Al pasar del ejemplo gráfico a un ejemplo codificado en lenguaje de programación java quedaría así:

```
//Clase ArregloFijo

public class ArregloFijo {
   int ArregloA[]=new int[10];
   int ArregloB[]=new int[15];

   public void llenarArreglo()
   {
      for(int i=0;i<ArregloA.length;i++)
      {
         ArregloA[i]=i+1;
      }
   }

   public void adicionarElemento(int elemento, int posicion)
   {
      ArregloA[posicion]=elemento;
   }

   public void imprimirArreglo()
   {
      for(int i=0;i<ArregloA.length;i++)
      {
         System.out.print(ArregloA[i]+" ");
      }
   }

}
```

La clase **ArregloFijo** contiene dos arreglos de tipo entero y longitud fija: **ArregloA[]** y **Arreglo[B]** y los métodos **llenarArreglo()** en el cual se asignan los valores del uno al diez al **ArregloA[]**, el método **adicionarElemento(int elemento, int posición)** que recibe como parámetros un elemento y una posición con los que se le asigna al **ArregloA** en la posición recibida el elemento **recibido** y el método **imprimirArreglo[]** el cual haciendo un recorrido a través del arreglo imprime cada uno de sus elementos.

A continuación en la clase Main se crea el objeto ArregloFijoA de tipo ArregloFijo y de inmediato se procede a llamar al método llenarArreglo con el que se asignan valores a cada una de las posiciones del arreglo, luego se imprime, se le adiciona un valor a la posición cero del arreglo y se vuelve a imprimir el arreglo una vez hecho el cambio en arreglo.

```
/Clase Main

public class Main {

   public static void main(String[] args) {
     ArregloFijo A = new ArregloFijo();
     A.llenarArreglo();
     A.imprimirArreglo();
     A.adicionarElemento(15,0);
     System.out.println("Después de hacer un cambio: ");
     A.imprimirArreglo();
   }

}
```

Si miramos en la Clase Main la línea que aparece en negrita (**A.adicionarElemento(15,0);**), esta línea de código funcionará siempre y cuando no agreguemos un valor para posiciones mayores a las dadas en el arreglo inicial. El método *adicioanrElemento()* de la clase ***ArregloFijo*** contiene dos parámetros de tipo entero: el primero equivale al valor numérico que se quiere incorporar en el vector y el segundo hace referencia a la posición en la cual se ingresará dicho valor (Para nuestro caso no puede ser superior a 9 que es el índice máximo del arreglo).

Ejecutando el programa anterior, la salida para el programa sería:

```
run:
1 2 3 4 5 6 7 8 9 10
Después de hacer un cambio:
15 2 3 4 5 6 7 8 9 10 BUILD SUCCESSFUL (total time: 2 seconds).
```

Pero si lo ejecutásemos llamando al método adicionarElemento de la siguiente forma: **A.adicionarElemento(15,10);**

La salida nos producirá será:

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
1 2 3 4 5 6 7 8 9 10        at ArregloFijo.adicionarElemento(ArregloFijo.java:16)
          at Main.main(Main.java:8)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

Esto debido a que no es posible adicionar nuevos elementos (en este caso la posición 10) a un arreglo que se ha definido de forma fija. Por lo tanto como ya se había mencionado anteriormente, la solución será crear o utilizar un arreglo del mismo tipo y de mayor tamaño para poder así adicionar un nuevo elemento.

Haciendo las siguientes modificaciones al programa inicial:

```
//Clase Arreglo Fijo:

public class ArregloFijo {
   int ArregloA[]=new int[10];
   int ArregloB[]=new int[15];

   public void llenarArreglo()
   {
      for(int i=0;i<ArregloA.length;i++)
      {
         ArregloA[i]=i+1;
      }
   }

   public void adicionarElemento(int elemento, int posicion)
   {
      ArregloA[posicion]=elemento;
   }

   public void imprimirArreglo(int Arreglo[])
   {
      for(int i=0;i<Arreglo.length;i++)
      {
         System.out.print(Arreglo[i]+" ");
      }
   }

   public void copiarArreglo()
   {
      for (int i=0;i<ArregloA.length;i++)
      {
         ArregloB[i]=ArregloA[i];
      }
   }
```

///Clase Main:

```
public class Main {

   public static void main(String[] args) {
     ArregloFijo A = new ArregloFijo();
     A.llenarArreglo();
     A.imprimirArreglo(A.ArregloA);
     A.adicionarElemento(11,0);
     System.out.println();
     System.out.println("Después de hacer un cambio: ");
     A.imprimirArreglo(A.ArregloA);
     A.copiarArreglo();
     System.out.println();
     System.out.println("Impresión del arreglo B: ");
     A.imprimirArreglo(A.ArregloB);

   }

}
```

Y ejecutando nuevamente el programa la salida será:

```
run:
1 2 3 4 5 6 7 8 9 10
Después de hacer un cambio:
11 2 3 4 5 6 7 8 9 10
Impresión del arreglo B:
11 2 3 4 5 6 7 8 9 10 0 0 0 0 0 BUILD SUCCESSFUL (total time: 1 second)
```

Ahora se modificará el programa nuevamente de tal forma que se pueda adicionar un Nuevo elemento al ArregloB que es donde hemos copiado todos los elementos del ArregloA:

//Clase ArregloFijo:

```
public class ArregloFijo {
   int ArregloA[]=new int[10];
   int ArregloB[]=new int[15];
```

```
   public void llenarArreglo()
   {
      for(int i=0;i<ArregloA.length;i++)
      {
         ArregloA[i]=i+1;
      }
   }

   public void adicionarElemento(int elemento, int posicion,int Arreglo[])
   {
      Arreglo[posicion]=elemento;
   }

   public void imprimirArreglo(int Arreglo[])
   {
      for(int i=0;i<Arreglo.length;i++)
      {
         System.out.print(Arreglo[i]+" ");
      }
   }

   public void copiarArreglo()
   {
      for (int i=0;i<ArregloA.length;i++)
      {
         ArregloB[i]=ArregloA[i];
      }
   }

}

//Clase Main:

public class Main {

   public static void main(String[] args) {
      ArregloFijo A = new ArregloFijo();
      A.llenarArreglo();
      A.imprimirArreglo(A.ArregloA);
      A.adicionarElemento(11,0,A.ArregloA);
      System.out.println();
```

```
      System.out.println("Después de hacer un cambio: ");
      A.imprimirArreglo(A.ArregloA);
      A.copiarArreglo();
      System.out.println();
      System.out.println("Impresión del arreglo B: ");
      A.imprimirArreglo(A.ArregloB);
      A.adicionarElemento(12, 10, A.ArregloB);
      System.out.println();
      System.out.println("Impresión del arreglo B: una vez se ha modificado");
      A.imprimirArreglo(A.ArregloB);
    }

}
```

Y ejecutando nuevamente el programa la salida será:

```
run:
1 2 3 4 5 6 7 8 9 10
Después de hacer un cambio:
11 2 3 4 5 6 7 8 9 10
Impresión del arreglo B:
11 2 3 4 5 6 7 8 9 10 0 0 0 0 0
Impresión del arreglo B: una vez se ha modificado
11 2 3 4 5 6 7 8 9 10 12 0 0 0 0 BUILD SUCCESSFUL (total time: 1 second)
```

Como podemos ver, esto es costoso en términos de trabajo del programador y llegará el momento (para nuestro caso cuando el ArregloB tenga ocupadas sus 15 posiciones) en que tendremos que hacer modificaciones nuevamente al programa.

6.1 LA CLASE ARRAYLIST

En Java, ArrayList es una clase que permite almacenar de forma dinámica diferentes tipos de datos u objetos sin necesidad de declarar el tamaño al momento de su creación como sí se hace cuando creamos un Array. Su aplicación es muy variada, pero sobre todo se usan para el manejo de estructuras dinámicas como Colas, Pilas, Arboles y Grafos.

6.2 PRINCIPALES MÉTODOS DE LA CLASE ARRAYLIST

Método add: Permite añadir un nuevo elemento al ArrayList. Su sintáxis es la siguiente:

```
nombreArrayList.add("Elemento nuevo");
```

Adiciona un nuevo elemento al final del ArrayList.

```
nombreArrayList.add(posición,"Elemento nuevo");
```

Adiciona un nuevo elemento en una posición dada del ArraList.

Método size: Permite obtener el tamaño del ArrayList en un momento dado.

```
nombreArrayList.size();
```

Método get: Permite conocer el elemento que se encuentra en una posición dada del ArrayLis

```
nombreArrayList.get(posición);
```

Método contains: Permite conocer si existe dentro del ArrayList un elemento dado y que es pasado como parámetro del método.

```
nombreArrayList.contains("Elemento");
```

Método indexOf: Permite conocer la posición de la primera ocurrencia del elemento que es pasado como parámetro.

```
nombreArrayList.indexOf("Elemento");
```

Método lastIndexOf: Permite conocer la posición de la última ocurrencia del elemento que es pasado como parámetro.

```
nombreArrayList.lastIndexOf("Elemento");
```

Método remove: Permite eliminar elementos del ArrayList.

```
nombreArrayList.remove("Elemento");
```

Elimina del ArrayList el elemento que fue pasado como parámetro.

```
nombreArrayList.remove(posición);
```

Elimina del ArrayList el elemento cuya posición es pasada como parámetro.

Método clear: Elimina todos los elementos de un ArrayList dado.

```
nombreArrayList.clear();
```

Método isEmpty: Este método nos permite saber si un ArrayList está vacío o no. Si está vacío devuelve true o sino false.

```
nombreArrayList.isEmpty();
```