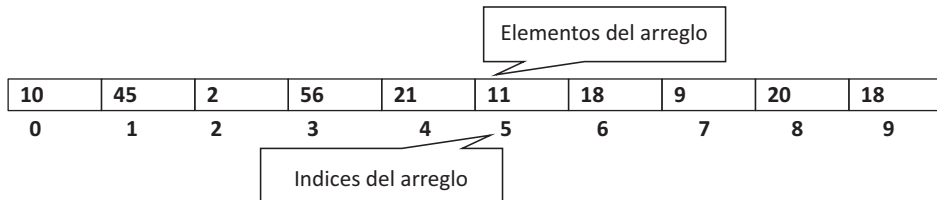


5. ARREGLOS

Un arreglo es una estructura de almacenamiento de un número determinado de variables primitivas o referencias a objetos, todas del mismo tipo y que cada elemento está identificado por uno o varios índices. Con los arreglos se pueden construir vectores, matrices y demás estructuras repetitivas de datos; las existencias de estas estructuras han sido de gran utilidad, solo limitadas por el hecho de que su dimensión debe estar previamente declarada y no puede ser ampliado en tiempo de ejecución [5]. Un arreglo lo podemos definir básicamente como un conjunto de datos o variables del mismo tipo referenciadas bajo un mismo nombre. Para poder acceder a cada uno de los elementos que conforman este conjunto es indispensable hacer uso de una variable de tipo entero que hace las veces de índice.

El siguiente ejemplo ilustra la definición de un arreglo y sus componentes:

Arreglo **Numeros**



En la ilustración anterior se pueden identificar las tres características esenciales que debe contener todo arreglo:

- **Nombre:** El nombre bajo el cual se identifican cada uno de los elementos del arreglo. Para este caso **Numeros**.
- **Elementos:** Cada una de las variables individuales agrupadas bajo el nombre **Numeros**. Note que para este caso es un arreglo de tipo entero, por lo cual todos sus elementos son de tipo entero.

- **Índice:** Representa la posición que ocupa un elemento dentro del arreglo. Es importante no llegar a confundir la posición con el contenido dentro del arreglo; en arreglo **Numeros** se observa que en la posición 0 contiene 10, la posición 1 contiene 45, la posición 2 contiene 2, la posición 3 contiene 56 y así sucesivamente.

5.1 DECLARACIÓN DE UN ARREGLO

La declaración de un arreglo se hace especificando el tipo de datos que agrupará el arreglo seguido por los símbolos [] y el nombre del arreglo. En el ejemplo siguiente se crean dos arreglos de tipo enter:

```
int conjunto[];  
int [] lista;;
```

5.2 INICIALIZACIÓN DE UN ARREGLO

La instrucción anterior permite la declaración de un arreglo, pero para poder inicializarlo, es necesario recurrir al operador new:

```
int conjunto[] = new int[10];;
```

Esta instrucción inicializa un arreglo de tipo entero compuesto por 10 elementos. Es importante puntualizar que los índices de un arreglo se inician en 0 y no en 1. Para este ejemplo los índices del arreglo van desde 0 hasta 9 y no desde 1 hasta 10.

Cuando se inicializa un arreglo de tipo entero, cada uno de sus elementos contendrá como valor predefinido 0.

5.3. REFERENCIACIÓN DE ELEMENTOS

Para poder hacer referencia a cada uno de los elementos de un arreglo es necesario utilizar el índice que representa su posición dentro del arreglo.

```
void imprimirVector()  
{  
    for(int i = 9;i>=0;i--)  
    {  
        System.out.print(conjunto[i]+" ");  
    }  
}
```

El método anterior permite la impresión del contenido de cada uno de los elementos del arreglo **conjunto**. Tenga en cuenta que el índice máximo del arreglo es 9 y no 10. Si se pretendiera tener acceso a *conjunto [10]* se generaría un error al momento de ejecutar el programa.

Java permite conocer la longitud de un arreglo a través de la propiedad `length` del arreglo, cuya sintaxis es:



NombreArreglo.**length**

Un ejemplo de su utilización es el siguiente:

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

Este método, aunque en forma ascendente, también hace la impresión del contenido de cada uno de los elementos del arreglo **conjunto**.

El siguiente ejemplo crea un arreglo de 10 elementos de tipo entero y lo llena con los números del 1 al 10 a través de un bucle for. También con un bucle for, se imprime el contenido de cada uno de los elementos del arreglo **Lista**. En esta impresión es posible ver el comportamiento de la variable que hace de índice así como su contenido.

```
import java.io.*;
public class Arreglos
{
    public static void main(String[]args) throws IOException
    {
        Vectores vector1 = new Vectores();
        vector1.llenarVector();
        vector1.imprimirVector();
        System.out.println();
        System.out.println("Finalizó el programa!");
    }
}
```

```
class Vectores
{
    int Lista[] = new int[10];
    void llenarVector()
    {
        for(int i = 0;i<10;i++)
        {
            Lista[i]=i+1;
        }
    }
    void imprimirVector()
    {
        for(int i = 0;i<Lista.length;i++)
        {
            System.out.println("Lista ["+i+"] = "+Lista[i]+" ");
        }
    }
}
```

Al ejecutar el programa, su salida por consola es la siguiente:

```
Lista [0] = 1
Lista [1] = 2
Lista [2] = 3
Lista [3] = 4
Lista [4] = 5
Lista [5] = 6
Lista [6] = 7
Lista [7] = 8
Lista [8] = 9
Lista [9] = 10
```

¡Finalizó el programa!

5.4 INICIALIZACIÓN DE ARREGLOS INDETERMINADOS

En muchas oportunidades, debido a que se hace dispendioso conocer la longitud de un arreglo, se hace necesario definir e inicializar arreglos de tamaño indeterminado. Java calcula automáticamente el tamaño del arreglo dependiendo de los valores que lo conforman. Considere el siguiente ejemplo en el que se desea construir un arreglo para almacenar códigos tipo carácter para diferentes mensajes correspondientes a fallos que se presentan en un sistema:

```
public class CodigosFalla
{

    public static void main(String[]args)
    {
        char letras[] = {'a', 'b', 'c', 'd'};
        for(int i = 0; i < letras.length; i++)
            System.out.println(letras[i]);
    }
}
```

Se puede observar que la definición y el llenado del arreglo se hacen en una misma línea de código y sin utilizar la palabra reservada **new**.

5.5 COPIAR UN ARREGLO EN OTRO

Es posible que necesitemos manipular los elementos contenidos en un arreglo pero conservando una copia de sus valores originales. En este caso lo más lógico es crear un arreglo del mismo tipo y tamaño que el original y crear un algoritmo que nos permita pasar uno a uno los elementos contenidos en el vector inicial hacia un vector destino, con lo cual, tendríamos lo siguiente:

```
import java.io.*;
public class CopiaArreglos
{
    public static void main(String[]args)
    {
        Pares pares1 = new Pares();
        System.out.println("Arreglos originales!");
        pares1.imprimirListas();
        pares1.trasladarLista();
        System.out.println("Arreglos modificados!");
        pares1.imprimirListas();
    }
}

class Pares
{
    int lista1 [] = {2,4,6,8,10};
    int lista2 [] = new int[5];
    void trasladarLista()
    {
```

```
        int i;
        for (i=0; i<lista1.length; i++)
            lista2[i]=lista1[i];
    }
    void imprimirListas()
    {
        System.out.println();
        for (int i=0; i<lista1.length; i++)
            System.out.println("Lista1 ["+i+"] = "+lista1[i]);
        System.out.println();
        for (int i=0; i<lista2.length; i++)
            System.out.println("Lista2 ["+i+"] = "+lista2[i]);
    }
}
```

Pero teniendo en cuenta que estamos incursionando en la Programación Orientada a Objetos, donde una de sus principios es la reutilización, es bueno que hagamos uso del paquete del sistema, en donde se nos ofrece un método para copiar un arreglo y cuya sintaxis general es la siguiente:

```
System.arraycopy(arregloOri,iniarregloOri,arregloDest,iniarregloDest,numelementosCopiar);
```

Donde ***arregloOri*** corresponde al nombre del arreglo origen, ***iniarregloOri*** corresponde a la posición del arreglo origen desde la que se inicia la copia, ***arregloDest*** corresponde al nombre del arreglo en el que se hará la copia*, ***iniarregloDest*** corresponde a la posición del arreglo destino en que se inicia la copia y ***numelementosCopiar*** corresponde al número de elementos del arreglo origen que serán copiados en el arreglo destino.

- * Es importante tener en cuenta que si se quiere hacer una copia total del arreglo origen, los dos arreglos deben ser del mismo tipo y el arreglo destino debe tener como mínimo la misma cantidad de elementos que el arreglo origen.

Aplicando esta sintaxis, el programa anterior queda transformado de la siguiente forma:

```
import java.io.*;
public class CopiaArreglo1
{
    public static void main(String[] args)
    {
        OtrosPares otrosPares1 = new OtrosPares();
        //IMPRIMIR ARREGLOS INICIALES
        otrosPares1.imprimirLista();
        otrosPares1.trasladarLista();
        System.out.println();
        //IMPRIMIR ARREGLOS DESPUES DE LA COPIA
        otrosPares1.imprimirLista();
    }
}
class OtrosPares
{
    int lista1[] = {12,14,16,18,20};
    int lista2[] = new int[5];
    void trasladarLista()
    {
        System.arraycopy(lista1,0,lista2,0,lista1.length);
    }
    void imprimirLista()
    {
        for(int i=0;i<lista1.length;i++)
            System.out.println("lista1 ["+i+"] = "+lista1[i]);
        System.out.println();
        for(int i=0;i<lista2.length;i++)
            System.out.println("lista2 ["+i+"] = "+lista2[i]);
    }
}
```

Con lo cual la salida por consola estaría dada de la siguiente forma:

Impresión de los arreglos antes de hacer la copia de lista1 en lista2:

lista1 [0] = 12	lista2 [0] = 0
lista1 [1] = 14	lista2 [1] = 0
lista1 [2] = 16	lista2 [2] = 0
lista1 [3] = 18	lista2 [3] = 0
lista1 [4] = 20	lista2 [4] = 0

y la impresión de los arreglos luego de copiar lista1 en lista2:

```
lista1 [0] = 12  
lista1 [1] = 14  
lista1 [2] = 16  
lista1 [3] = 18  
lista1 [4] = 20
```

```
lista2 [0] = 12  
lista2 [1] = 14  
lista2 [2] = 16  
lista2 [3] = 18  
lista2 [4] = 20
```

En java el método **arraycopy()** ofrece una forma rápida y fácil para copiar un arreglo de cualquier tipo en otro, pero es importante tener en cuenta que este método nos puede generar las siguientes excepciones:

[NullPointerException:](#)

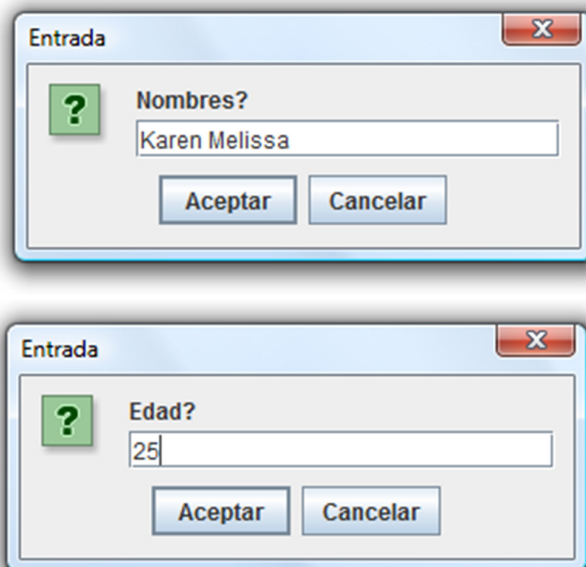
Cuando alguno de los arreglos es nulo, es decir, no ha sido inicializado.

[ArrayStoreException:](#)

Cuando se intenta copiar en un arreglo de diferente tipo.

[IndexOutOfBoundsException:](#)

Cuando si intenta copiar fuera del área reservada para el arreglo.

5.6 CAJAS DE DIALOGO PARA INGRESAR DATOS

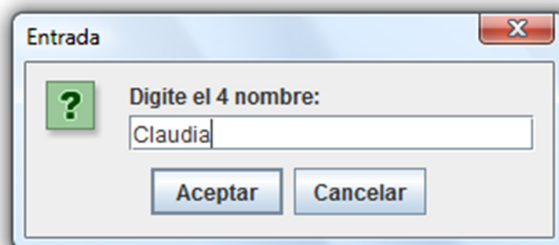
Ejemplo de programa para ingresar datos a un arreglo:


```
import javax.swing.JOptionPane;

public class ArreglosChar
{
    public static void main(String[] args)
    {
        FallaSistema fs1 = new FallaSistema();
        fs1.llenarVector();
        fs1.imprimirVector();
    }
}

class FallaSistema
{
    String Falla[] = new String[10];
    void llenarVector()
    {
        for(int i = 0; i < Falla.length; i++)
        {
            Falla[i] = JOptionPane.showInputDialog
                ("Digite el "+i+" nombre: ");
        }
    }
    void imprimirVector()
    {
        for(int i = 0; i < Falla.length; i++)
        {
            System.out.print(Falla[i] + " ");
        }
    }
}
```

Donde la caja de diálogo a través de la cual se ingresan los datos desde el teclado al arreglo es la siguiente:



Y cuya salida por consola una vez ejecutado el programa puede ser similar a la siguiente:

```
María José Paula Hernán Claudia Nicolás Rocío Juan Manuel
Fernanda
```

5.7. ARREGLOS COMO PARÁMETROS

En java es posible pasar un arreglo completo (con todos sus elementos) a un método. Una vez en el método sus elementos podrán ser utilizados para realizar diferentes operaciones. A continuación se ilustra mediante un ejemplo su uso; se implementa el método ***imprimirDatos()*** que recibe como parámetro un arreglo de tipo entero e imprime cada uno de sus valores.

```
package invertirarreglo;
```

```
public class Main {
    public static void main(String[] args) {
        Arreglo miArreglo = new Arreglo();
        miArreglo.leerDatos();
        miArreglo.imprimirDatos(miArreglo.A);
        miArreglo.invertirDatos();
        miArreglo.imprimirDatos(miArreglo.B);
    }
}
```

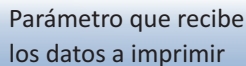
```
package invertirarreglo;
import javax.swing.JOptionPane;
```

```
public class Arreglo {
    int longitudArreglo = Integer.parseInt(JOptionPane.
        showInputDialog("Digite la longitud del arreglo: "));
    int A[] = new int[longitudArreglo];
    int B[] = new int[longitudArreglo];

    void leerDatos() {
        for (int i = 0; i < A.length; i++) {
            A[i] = Integer.parseInt(JOptionPane.showInputDialog
                ("Digite el valor de A[" + i + "] : "));
        }
    }
}
```

```
void imprimirDatos(int[] Impresion) {  
    for (int i = 0; i < Impresion.length; i++) {  
        System.out.println("Valor[" + i + "] = " + Impresion[i]);  
    }  
}  
  
void invertirDatos() {  
    for (int i = (A.length - 1); i >= 0; i--) {  
        B[(A.length - 1) - i] = A[i];  
    }  
}
```

En el ejemplo anterior, podemos observar que el método `imprimirDatos()`, recibe como parámetro un arreglo de tipo entero. A través de este parámetro el método recibe los datos del arreglo que se desea imprimir.



Parámetro que recibe
los datos a imprimir

```
void imprimirDatos(int[] Impresion)
```

5.8 ORDENAR UN ARREGLO

Es importante que una vez ingresada una cantidad de elementos en un arreglo los podamos ordenar y visualizar. Podemos hacerlo desarrollando nuestro propio algoritmo o haciendo uso el método **`Arrays.sort()`** contenido en el paquete *`java.util.Arrays`* y cuya sintaxis es la siguiente:



```
Arrays.sort(nombreArreglo);
```

A continuación se presenta el programa anterior, modificado para hacer el ordenamiento de un arreglo a través del método **`Arrays.sort()`**.

```
import java.util.Arrays;  
import javax.swing.JOptionPane;  
  
public class ArreglosChar  
{
```

```
public static void main(String[] args)
{
    FallaSistema fs1 = new FallaSistema();
    fs1.llenarVector();
    fs1.imprimirVector();
}

class FallaSistema
{
    String Falla[] = new String[10];
    void llenarVector()
    {
        for(int i = 0; i < Falla.length; i++)
        {
            Falla[i] = JOptionPane.showInputDialog
            ("Digite el "+i+" nombre: ");
        }
    }
    void imprimirVector()
    {
        Arrays.sort(Falla);
        for(int i = 0; i < Falla.length; i++)
        {
            System.out.print(Falla[i] + " ");
        }
    }
}
```

5.9 ARREGLOS MULTIDIMENSIONALES

Un arreglo multidimensional es un arreglo de arreglos y se caracterizan por necesitar más de un índice para acceder a sus elementos, los más usados son las **matrices** o **tablas**, en los que el acceso se hace a través de dos índices: uno que nos ubica en la fila y otro que nos ubica en la columna[]. Para hacer referencia a un objeto se hace dando el nombre de la matriz y la posición de intersección fila-columna, cuya forma de referencia es la siguiente:



Matriz[fila][columna]

La siguiente figura ilustra una matriz de 3 filas por 3 columnas con sus respectivos índices.

Matriz M:

M[0][1]	M[0][1]	M[0][2]
M[1][0]	M[1][1]	M[1][2]
M[2][0]	M[2][1]	M[2][2]

5.9.1 DECLARACIÓN DE UNA MATRIZ

La forma de declarar una matriz es similar a la de declarar un arreglo, pero con la diferencia que se adicionan un par de corchetes más en los que se indica el número de columnas que contendrá dicha matriz.

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

5.9.2 INICIALIZACIÓN DE UNA MATRIZ

De la misma forma que es posible inicializar arreglos, también se pueden inicializar las matrices

```
Int M[][] =  
{  
    {1,2,3},  
    {4,5,6},  
    {7,8,9}  
};
```

Una vez inicializada la matriz, el acceso a cada uno de sus elementos se hace a través del nombre de la matriz y los correspondientes índices del elemento. Por ejemplo la línea de código ***System.out.println(M[0][2]0;***, arrojará como resultado por consola el número 3, que es el valor almacenado en la posición (0,2) de la matriz **M**.

El siguiente programa muestra detalladamente la forma como se crea, inicializa y se tiene acceso a los elementos de una matriz. El método **imprimirMatriz()** utiliza los índices **i** (para manejar el acceso entre filas) y **j** (para manejar el acceso entre columnas).

```
public class Multidimensionales
{
    public static void main(String[] args)
    {
        Matrices matriz1 = new Matrices();
        matriz1.imprimirMatriz();
    }
}
class Matrices
{
    int i,j,k;
    int M[][]=
    {
        {1,2,3},
        {4,5,6},
        {7,8,9}
    };
    void imprimirMatriz()
    {
        for(i=0;i<3;i++)
        {
            for(j=0;j<3;j++)
            {
                System.out.print(M[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

Al ejecutar el programa, su salida por consola será:

```
1 2 3
4 5 6
7 8 9
```

También es posible acceder a cada una de las posiciones de memoria de una matriz para asignar valores, tarea que también se puede hacer a través de dos bucles que manejen sus respectivos índices. El siguiente ejemplo crea una matriz **Tablas[9][10]** para almacenar las tablas de multiplicación desde el 1 hasta el 9.

```
public class MatTablas
{
    public static void main(String[] args)
    {
        Multiplicacion multiplicacion1 = new
        Multiplicacion();
        multiplicacion1.inicializarMatriz();
        multiplicacion1.imprimirMatriz();
    }
}

class Multiplicacion
{
    int i,j,k;
    int MatrizMultiplicacion[][];
    void inicializarMatriz()
    {
        MatrizMultiplicacion = new int [9][10];
        for(i=0;i<9;i++)
        {
            for(j=0;j<10;j++)
            {
                MatrizMultiplicacion[i][j]=
                (i+1)*(j+1);
            }
        }
    }
    void imprimirMatriz()
    {
        for(i=0;i<9;i++)
        {
            for(j=0;j<10;j++)
            {
                System.out.print
                (MatrizMultiplicacion[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

5.9.3 MATRICES IRREGULARES

No siempre los arreglos bidimensionales son regulares, algunas veces es necesario definir un arreglo bidimensional donde no todos sus arreglos son del mismo tamaño, tal es el caso que se presenta en la siguiente figura.

A	B		
A	B	C	D
A	B	C	
A			

Esto es posible gracias a que cuando se reserva memoria para una matriz, solo es necesario especificar la cantidad de memoria para la primera dimensión. Posteriormente es posible especificar la cantidad de memoria requerida para las demás dimensiones. A continuación se presenta un programa en el que el tamaño de la segunda dimensión no está determinado al momento de definir la primera dimensión.

```
public class MatrizIndeterminada
{
    public static void main(String[] args)
    {
        Indeterminada indeterminada1 = new
        Indeterminada();
        indeterminada1.completarMatriz();
        indeterminada1.llenarMatriz();
        indeterminada1.imprimirMatriz();
    }
}

class Indeterminada
{
    char M[][] = new char[4][];
    void completarMatriz()
    {
        M[0]=new char[2];
        M[1]=new char[4];
        M[2]=new char[3];
        M[3]=new char[1];
    }
}
```



```
void llenarMatriz()
{
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<M[i].length;j++)
        {
            if(j==0)
                M[i][j]= 'A';
            else if(j==1)
                M[i][j]='B';
            else if(j==2)
                M[i][j]='C';
            else
                M[i][j]='D';
        }
    }
}

void imprimirMatriz()
{
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<M[i].length;j++)
        {
            System.out.print(M[i][j]+" ");
        }
        System.out.println();
    }
}
```

La salida generada luego de ejecutar el programa será:

```
A B
A B C D
A B C
A
```

Quizá sea creamos que es poca e incluso nula la aplicabilidad de este tipo de matrices. No obstante, su uso está expandido a muchas situaciones, tal es el caso de las matrices dispersas, en las que sabemos que no se hace uso de todos sus elementos.

Una de las formas más comunes de darle uso a la información hoy en día es presentándola en matrices conformadas por filas y columnas en las cuales se introducen, editan, procesan y visualizan datos. A través de ellas es posible presentar desde informes sencillos de gastos hasta cálculos financieros y análisis matemáticos y estadísticos muy complejos.

Entre las aplicaciones típicas de las matrices están los presupuestos, registro de ingresos, proyecciones, gastos, planificaciones de producción, juegos y hojas electrónicas entre muchos otros.

5.9.4 ARREGLOS TRIDIMENSIONALES

En muchas ocasiones es necesario declarar arreglos de más de dos dimensiones y entre los que más uso tienen están los arreglos tridimensionales, los cuales deben ser accedidos mediante tres índices. El siguiente programa crea un arreglo tridimensional y le asigna a cada elemento un valor correspondiente a la suma de los tres índices que permiten su acceso.

```
public class MatrizTridimensional
{
    public static void main(String[] args)
    {
        Cubo cubo1 = new Cubo();
        cubo1.llenarNum();
        cubo1.imprimirCubo();
    }
}
class Cubo
{
    int numsumpro[][][] = new int [3][3][3];
    void llenarNum()
    {
        for(int i=0; i<numsumpro.length; i++)
        {
            for(int j=0; j<3; j++)
            {
                for(int k=0; k<3; k++)
                {
                    numsumpro[i][j][k] = i+j+k;
                }
            }
        }
    }
}
```

```
}  
void imprimirCubo()  
{  
    for(int i=0;i<numsumpro.length;i++)  
    {  
        for(int j=0;j<3;j++)  
        {  
            for(int k=0;k<3;k++)  
            {  
                System.out.print(numsumpro[i]  
                [j][k]+ " ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}  
}
```

EJERCICIOS PROPUESTOS PARA ESTA UNIDAD

1. Cree un programa que contenga un arreglo que permita almacenar los siguientes números: 23, 24, 25, 26, 67, 90, 67, 32. Liste por consola el contenido del arreglo.
2. Elabore un programa que lea desde teclado y a través de una caja de diálogo los datos para un arreglo de tipo entero y tamaño 10, luego copie los datos de este arreglo en otro arreglo del mismo tipo y tamaño e imprímalos.
3. Elaborar un programa que inicialice dos arreglos de tipo entero y tamaño 7. El primero contiene los números desde el 1 hasta el 7 y el segundo contiene los números desde el 8 al 14. Una vez inicializados los arreglos copie los contenidos de las posiciones 0,1 y 2 del primer arreglo a partir de la segunda posición del segundo arreglo. (Haga uso del método **arraycopy()**).
4. Cuál es la salida por consola al ejecutar el siguiente programa?
5. Elabore un programa que genere la siguiente salida.

```
      1  
    1  1  
  1  2  1  
1  3  3  1  
1  4  6  4  1
```