6.- Métodos (subrutinas), arreglos y paso de parámetros.

6.1.- Declaración de arreglos de tipos primitivos.

Un arreglo es un conjunto de datos homogéneo que puede ser utilizado por un solo nombre de variable y un índice que indica la posición del dato dentro del conjunto.
Para declarar un arreglo utilizamos la siguiente sintaxis:

```
<tipo>[] <nombreArreglo> = new <tipo>[<tamañoDelArreglo>];
```

El tamaño del arreglo puede ser una variable entera o un valor entero explícito, observe que el tipo se repite dos veces una del lado izquierdo de la asignación y otra del lado derecho. La palabra **new** es una solicitud de alojamiento de memoria para el sistema Java y observe también los paréntesis cuadrados que indican la repetición de la celda.
Cuando el tipo es primitivo o **String** , el arreglo puede utilizarse inmediatamente después de su declaración, no si es de un tipo complejo ya sea declarado por Java o por el programador, los cuales quedan fuera del alcance de este curso.
Para utilizar un arreglo podemos utilizar la siguiente sintaxis en cualquier lugar ´donde se utilice una variable:

<nombreArreglo>[<valorDelIndice>]

El índice puede ser una variable entera o un valor entero explícito.

Por ejemplo el siguiente código declara un arreglo de números enteros de nombre id de 100 elementos y otro de números flotantes de nombre precio con 100 elementos:

```
int  id[]=new int[100];
float precio[]=new float[100];
```

Para utilizar leer el primer arreglo podemos utilizar el siguiente código:

```
int i,n;
boolean r;
// Capturando los arreglos de id y precio
    for(i=0; i<100; i++)
       {
        id[i]=Ip.wlee("Id Articulo: ",id[i]);
        precio[i]=Ip.wlee("Precio  : ",precio[i]);
        r=Ip.wleeRespuesta(
         "Desea capturar otro articulo");
        if (!r)
            break;
       }
```

```
n=i+1;
```

Para mostrar el arreglo podemos utilizar las siguientes instrucciones:

```
// Imprime lista leida
String mensaje="Lista de articulos leida";
for(i=0; i<n; i++)
    {
       mensaje += "\n " + Ip.formatoNumero(id[i])+
"   " + Ip.formatoDinero(precio[i]);
     }
Ip.escribeMensaje(mensaje);
```

Para ejemplificar el uso de los arreglos utilice los programas: 620ArreglosNumeros.java, 621ArreglosStrings.java y 622MuestraArregloIp.java.

6.2.- Flujo de control durante la ejecución de un método o subrutina.

Para implementar los procesos o módulos Java cuenta con métodos (en otros lenguajes se les llama subrutinas) hay dos tipos: los métodos de clase y los métodos de instancia.

El flujo de control durante la ejecución de un método es como se muestra en la figura 6.1. Suponiendo que del programa principal se le llama al método A, es necesario invocar el método para que al llegar a la instrucción de invocación se transfiera el control al método llamado, al transferirse el control se ejecutan las instrucciones contenidas en el método y al terminar continúa el proceso llamador o invocador con la siguiente instrucción después de la invocación.

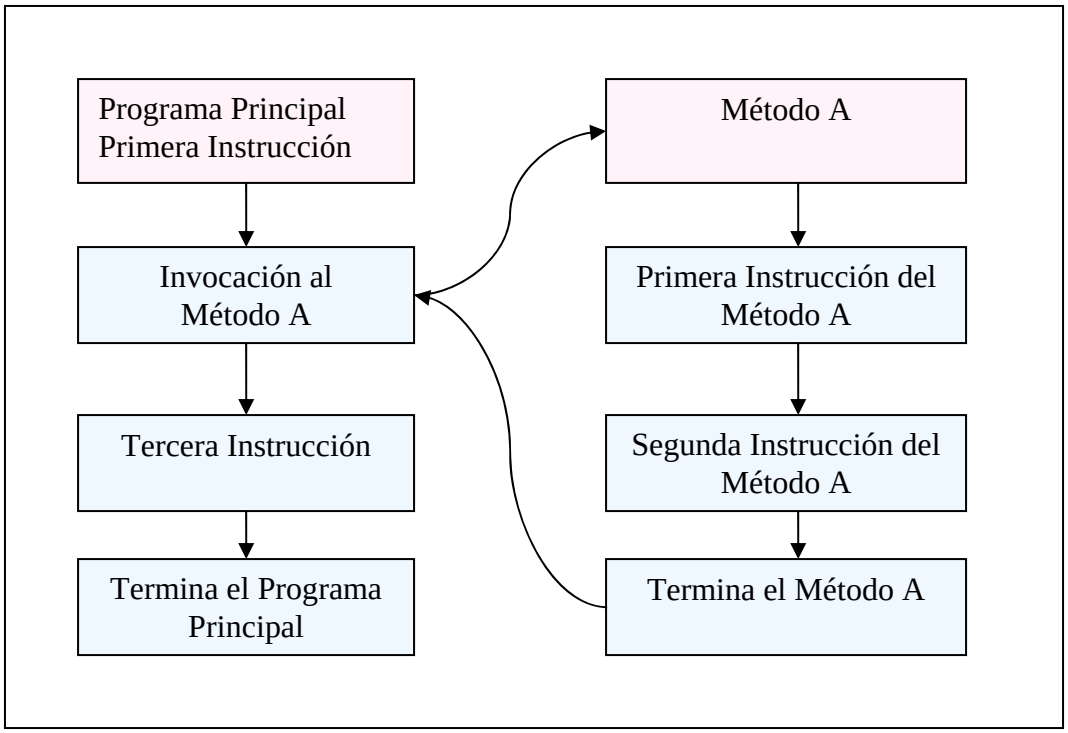

Figura 6.1 Flujo de control durante la ejecución de un método

En un mismo programa puede haber varias invocaciones a un mismo método o a varios métodos distintos. Un método puede invocar a otro método.

Los métodos de clase tiene la sintaxis siguiente:

```
class MetodoDeClase{
//Declaracion de variables del programa
public static void <nombreDelMetodo>() {
        //instrucciones del método
    }
//Proceso principal para ejecutar metodos
   public static void main(String[] args) {
        //Instrucciones primera parte
        //Invocación al método
        <nombreDelMetodo>();
      //Instrucciones segunda parte
  }//Termina el proceso
 }//Termina el programa
```

Vea el ejemplo 651MetodosDeClase.java que ilustra el funcionamiento de un método de clase de éste tipo.
Note el atributo ***`static`*** del método, eso es lo que lo hace un método de clase.

Cuándo el método de clase regresa un valor en el nombre se tiene:

```
class MetodoDeClase{
//Declaracion de variables del programa
//definición de los método que sean necesarios uno o más
Public static <tipoQueRegresa> <nombreDelMetodo>() {
        //instrucciones del método
        return <valorARegresar>;
   }
//Proceso principal para ejecutar metodos
  public static void main(String[] args) {
        //Instrucciones primera parte
        //Invocación al método
        <variableParaRecibirElValorRegresado> =
              <nombreDelMetodo>();
     //Instrucciones segunda parte
  }//Termina el proceso
 }//Termina el programa
```

Note: la necesidad de especificar un tipo que el método regresa en su encabezado, la instrucción ***`return`*** para especificar el valor a regresar y en el programa llamador la variable que recibe el valor que el método regresa en su nombre, el ejemplo adjunto 660RegresaValorEnElNombre.java muestra como se puede regresar un valor en el nombre del método.

Los métodos de instancia requieren que se aloje memoria para ellos, la sintaxis que se utiliza es la siguiente:

```
class <nombreClase>{
```

```
//Declaración de variables del programa
//definición de los métodos que sean necesarios uno o más
//El siguiente metodo no regresa valor en el nombre
public void <nombreDelMetodo1>() {
        //instrucciones del método
    }
//El siguiente metodo regresa un valor en el nombre
public <tipoQueRegresa> <nombreDelMetodo2>() {
        //instrucciones del método
        return <valorARegresar>;
    }
 //Proceso principal para ejecutar métodos requiere el atributo
//static
//Proceso principal para ejecutar metodos
   public static void main(String[] args) {
      //Solicita alojamiento de memoria para la instancia
        <nombreClase> <nombreInstancia> =
                                    new <nombreClase();
        //Instrucciones primera parte
        //Invocación al método que no regresa valor en el nombre
        <nombreInstancia>.<nombreDelMetodo>();
      //Instrucciones segunda parte
//Invocación al método que regresa un valor en el nombre
        variableParaRecibirElValorRegresado> =
              <nombreInstancia>.<nombreDelMetodo>();
//Instrucciones tercera parte
  }//Termina el proceso
} //Termina el programa
```

La diferencia es que en este segundo tipo de métodos es necesario alojar memoria para el programa y los **`métodos`**, a esto se le llama instancia y se hace por medio de la instrucción **`new`**. Además se requiere llamar a los métodos anteponiendo el nombre de la instancia seguida por un punto, en este caso el prefijo de llamada a los métodos queda como **`prog.`**

El programa adjunto 652MetodosDeInstancia.java muestra un ejemplo completo de este tipo de métodos.
Pos su sencillez hemos optado por utilizar durante este curso los métodos de clase.

6.3.- Uso de parámetros en los métodos.

Un método sirve para implementar un proceso, en programación estructurada esto se le llama módulo, un módulo es un proceso que se puede identificar como unidad, por ejemplo: cálculo de un área, ordenamiento de datos, consulta de una base de datos, escritura de la información en un archivo, validación de información, lectura de información y muchos otros.

La llamada a un método puede ser representada por un solo bloque con el nombre del método, eso indica que se llevará a cabo la ejecución de dicho método.

Los métodos reciben parámetros, los parámetros representan información de entrada al método con la cual se llevará a cabo la ejecución. El valor de los parámetros generalmente varía de una invocación o llamada a otra. Un parámetro se considera como una variable que dentro del método y se puede utilizar como una variable más, hay parámetros de entrada en ellos el método recibe información del programa llamador y hay parámetros de salida en los cuales los métodos envían información al programa llamador.

La sintaxis para definir parámetros es la siguiente:

```
class MetodoConParametros{
//Declaracion de variables del programa
public static void <nombreDelMetodo>
        (
        <tipoDelParámetro1> <nombreParámetro1>,
         <tipoDelParámetro2> <nombreParámetro2>,
                ....,
                ....,
                ....,
        <tipoDelParámetroN> <nombreParámetroN>
        )
    {
        //instrucciones del método
    }
```


```
//Proceso principal para ejecutar metodos
  public static void main(String[] args) {
    //Declaración de variables
    // que se enviarán como parámetros
      <tipoDelParámetro1> <nombreVariable1>;
      <tipoDelParámetro2> <nombreVariable2>;
      ....;
      ....;
      ....;
      <tipoDelParámetroN> <nombreVariableN>;
      //Instrucciones primera parte
      //Invocación al método con parámetros
      <nombreDelMetodo>
            (
            <nombreVariable1>,
            <nombreVariable2>,
            ....,
            ....,
            ....,
             <nombreVariableN>
            );
       //Instrucciones segunda parte
 }//Termina el proceso
}//Termina el programa
```

Es muy importante notar lo siguiente

1. Los parámetros se definen al definir el método, son parte de su encabezado y van entre paréntesis después del nombre del método, los parámetros podrán utilizarse como una variable más dentro del método y como ya se dijo en ellos se puede recibir y enviar información del o al programa llamador según sea el caso.
2. En el programa llamador se requiere definir una serie de variables del mismo tipo que los parámetros del método para utilizarlos en la instrucción de invocación, aunque esto no es indispensable, ya que en lugar de una variable puede enviarse un valor del tipo que espera el método, las variables que se utilizan en la invocación del método deben ser del mismo tipo que los parámetros definidos en el encabezado del método.

3. En la invocación la lista de parámetros no lleva el tipo, solo las variables o valores que se asignarán a cada parámetro, su tipo debe necesariamente coincidir con el encabezado del método.

6.4.- Paso de parámetros a los métodos por valor y por referencia.

Existen dos formas de pasar los parámetros: Por Valor y por Referencia.

Cuando se pasan los parámetros por valor, lo que hace el compilador de Java es que reserva un espacio en memoria para cada variable que se haya declarado como parámetro, luego al momento de la invocación del método copia los valores que la invocación este pasando a ese espacio que fue asignado para los parámetros, de tal forma que si los parámetros son alterados dentro del código del método, dicha alteración no se refleja en el programa llamador, ya que esencialmente se esta trabajando en otra parte de la memoria. Este es el método que Java utiliza por defecto para el paso de parámetros. La figura 6.2 nos muestra este método de pasar parámetros.

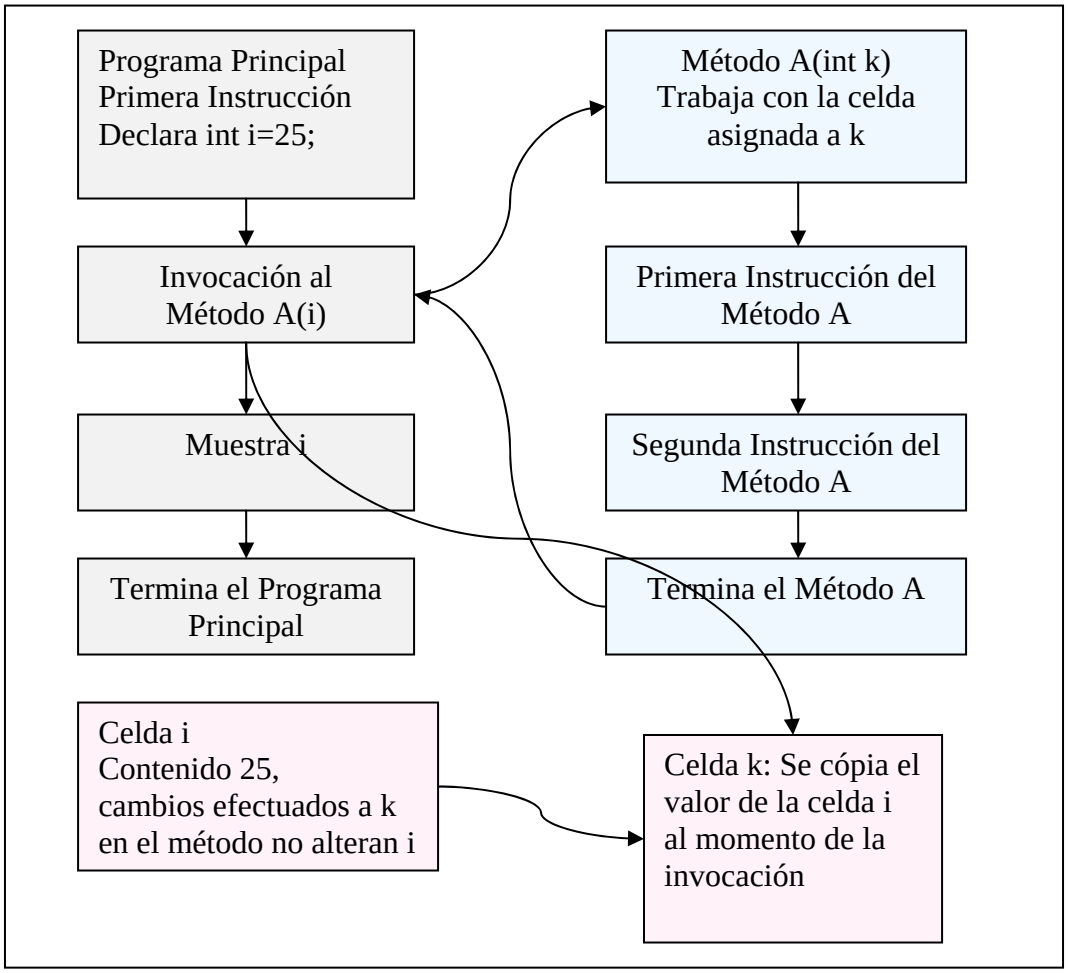

Figura 6.2 Paso de parámetros por valor.

El programa 655ParametrosPorValor.java adjunto demuestra como se comporta el paso de parámetros por valor.

Por otra parte cuándo se pasan los parámetros por referencia lo que sucede es que en lugar de llevarse a cabo una copia, lo que se pasa al método es la dirección de la memoria que ha sido asignada a las variables del programa llamador, de ésta forma el efecto logrado es que el programa llamado trabaja en la misma memoria que usa el programa llamador y cualquier cambio que haga en las variables se ve reflejado en el programa llamador.

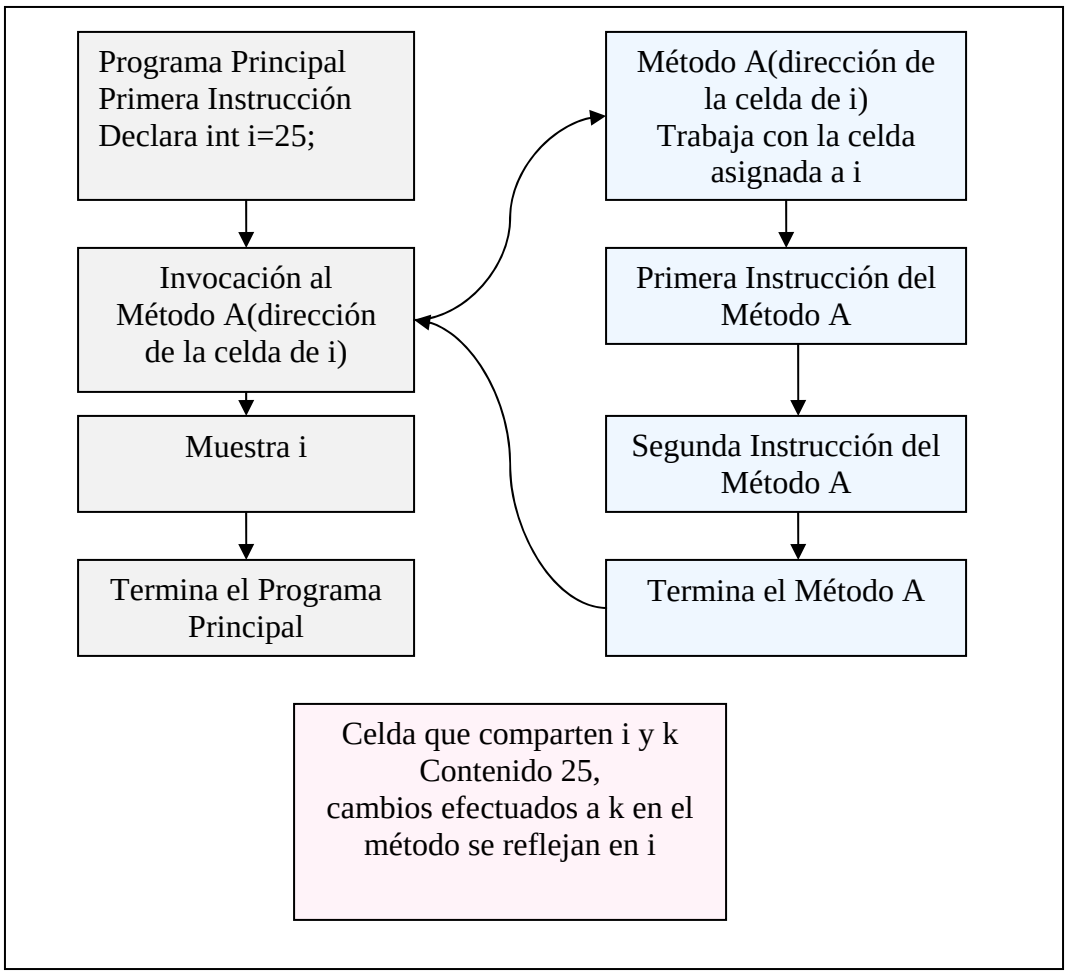

Figura 6.3 Paso de parámetros por referencia.

Todos los arreglos en Java se pasan por referencia, si se desea pasar un valor entero se puede declarar un arreglo entero de una posición y pasar el arreglo como parámetro de ésta forma se logra compartir el arreglo entre el programa llamador y el programa llamado. Vea los ejemplos adjuntos 665PasarArregloPorReferencia.java y el 666PasoParametros.java.

Otra manera de de lograr pasar parámetros por referencia es utilizar un bloque de memoria que se comparta entre los dos programas, como lo muestra el ejemplo

670PasarArregloPorReferenciaUsandoBoxing.java a ésta última técnica es la más utilizada, aunque es la más compleja.

6.5.- Firma de los métodos.

La firma de un método proporciona la identificación única del método, varios métodos pueden tener el mismo nombre, pero no la misma firma. La firma de un método esta formada por el nombre del método y la lista de los tipos de los parámetros de dicho método.

Por ejemplo:

```
class FirmaDeMetodos{
static String programa="FirmaDeMetodos";
//Definiciòn de los metodos con distintas firmas y con igual nombre
    public static void usaValor(int i){
          Ip.wescribeMensaje("Usa valor de un entero");
                i = 10;
    }
   public static void usaValor(String cadena){
          Ip.wescribeMensaje("Usa valor de una cadena");
          cadena ="Nuevo Valor";
    }
    //Proceso principal para ejecutar metodos
public static void main(String[] args) {
    Ip.escribeMensaje("Inicia la ejecucion del programa "+programa);
      int k=0;
      String variableCadena="valor original";
 //El metodo que se ejecuta depende del nombre del metodo y la lista
 //de los tipos de los parámetros
      usaValor(k);
      usaValor(variableCadena);
      Ip.escribeMensaje("Termina la ejecución del programa "+programa);
      }//Termina el proceso
 //Termina el programa
}
```

Los dos método del programa anterior se llaman igual, pero tienen un parámetro de diferente tipo y eso le permite al compilador Java diferenciar entre ellos, utilice el programa 675FirmaDeMetodos.java para comprobarlo.

6.6.- Ejemplo de métodos para lectura y proceso de arreglos.

Cuando se utilizan arreglos es necesario contar con métodos capaces de llevar a cabo las operaciones de lectura y escritura de arreglos, de otra manera nos encontraremos repitiendo código constantemente. Por ejemplo un procedimiento para leer arreglos de dos dimensiones puede ser muy útil. A continuación mostramos un ejemplo para leer un renglón de dos dimensiones, esto es una matriz y mostrarla.

```
class Matrices{
 public static void leeMatriz(float[][]   matriz,int[] n,  int[] m)
   { int i,j;
```

```
     n[0]=Ip.wlee("Numero de renglones ",n[0]);
     m[0]=Ip.wlee("Numero de columnas ",m[0]);
     // Capturando matriz
    for(i=0; i<n[0]; i++)
       {for(j=0; j<m[0]; j++)
         {matriz[i][j]=Ip.wlee(
           "Captura Elemento "+i+","+j+" : ",matriz[i][j]); }
          }
    }
public static void muestraMatriz(float[][]  matriz,int[] n,  int[] m)
   { int i,j;
     String mensaje="";
    //Listado
    for(i=0; i<n[0]; i++)
       {for(j=0; j<m[0]; j++)
             { mensaje+="   "+ matriz[i][j]; }
        //Al terminar un renglon lo indica con \n
        mensaje+="\n"; }
       Ip.wescribeMensaje(mensaje);
  }
  public static void main(String[] args)
   {final String programa = "Matrices";
    Ip.escribeMensaje("Inicia la ejecucion del programa "+ programa);
     // Declarando arreglo y variables
    float[][] A =new float[3][3];
    int i,j;
    //Numero de Renglones
    int[] n = {0};
    //Numero de Columnas
    int[] m ={0};
    leeMatriz(A,n,m);
    muestraMatriz(A,n,m);
    Ip.escribeMensaje("\nTermina ejecucion del programa " + programa);
  }
}
```

En este ejemplo es importante notar que el numero de renglones n y el numero de columnas m se pasan como arreglos de un elemento, esto como ya lo vimos anteriormente se debe a que es necesario pasarlos por referencia, ya que al leer la matriz deben ser actualizados por el método de lectura. Mayores detalle puede investigar al ejecutar y compilar el programa 680ArreglosConMetodos.java.

El ejemplo adjunto 680ArreglosConMetodos.java repite el ejemplo anterior 621ArreglosStrings.java pero ahora utilizando métodos para llevar a cabo las operaciones de una manera estructurada. Los ejemplos 682SumaMatrices.java y 683MultiplicacionMatrices.java ejemplifican la suma y la multiplicación de matrices respectivamente.

6.7.- Actividades

Compile y ejecute los ejemplos adjuntos agregando operaciones matriciales tales como el cálculo de la inversa de una matriz.

6.8.- Ejercicio propuesto:

Escriba un programa que lea una matriz en la cual el contenido de la celda es la tarifa de un flete, el número de renglón es el origen del flete y el número de la columna el destino. Posteriormente pregunte origen y destino así como peso en kilogramos del flete para calcular el costo total del flete que viene dado por la tarifa que multiplica el peso en kilogramos. Utilice métodos para implementar los procesos tales como leer matriz de tarifas, calcular el costo etc.