Capítulo

07

Funciones

Ania Cravero Leal

Departamento de Ingeniería de Sistemas
Facultad de Ingeniería, Ciencias y Administración

"Proyecto financiado por el Fondo de Desarrollo Educativo de la Facultad de Ingeniería, Ciencias y Administración de la Universidad de La Frontera"

Versión

1.0

TEMARIO

Funciones

En este capítulo introduciremos un nuevo elemento, la función, que te permitirá afrontar cualquier tarea compleja mediante la descomposición de ésta en las unidades de diseño y organización de datos. A partir de ahora, un programa será una organización de una o varias funciones relacionadas entre sí.

Esta nueva técnica, llamada diseño modular, te aportará dos ventajas: por una parte, completará y ampliará al diseño descendente como método de resolución de problemas algorítmicos; por otra, permitirá proteger la estructura de la información asociada a un determinado problema, limitando las operaciones que puedan actuar sobre ellas.

En primer lugar, te presentamos los conceptos básicos y los aplicamos a un ejemplo. Después veremos el uso de las variables para introducir datos (acceder) a los proceso de la función. Finalmente revisaremos dos metodologías reconocidas para crear programas modulares mediante funciones.

7.1 Conceptos Básicos

En los primero capítulos de este libro te hemos proporcionado una colección de utilidades que permiten escribir algoritmos claros y correctos para una amplia gama de problemas. Sin embargo, cuando la tarea a realizar es suficientemente larga y compleja, las herramientas que hasta ahora hemos conocido son insuficientes para satisfacer criterios de verificabilidad, legibilidad, reusabilidad y modificabilidad.

a. Verificabilidad:

Un software es verificable si sus propiedades pueden ser verificadas fácilmente. Por ejemplo, la correctitud o la performance de un software son propiedades que interesa verificar. El diseño modular, prácticas de codificación disciplinadas, y la utilización de lenguajes de programación adecuados contribuyen a la verificabilidad de un software.

La verificabilidad es en general una cualidad interna pero a veces también puede ser externa, por ejemplo, en muchas aplicaciones de seguridad crítica, el cliente requiere la verificación de ciertas propiedades.

b. Legibilidad:

Es posible tener dos versiones diferentes de un programa de software que funcionen exactamente de la misma manera, y que tengan el mismo nivel interno de diseño y construcción desde una perspectiva técnica, pero los cuales son vastamente diferentes en su legibilidad humana. Si un sistema de software no es legible, es difícil (o imposible) depurarlo, modificarlo, ampliarlo, escalarlo, etc. Hay otros dos aspectos concernientes a la legibilidad del software: la claridad que está construida dentro del código, y los comentarios que acompañan al código. La primera incluye nombres significativos para las variables y constantes, buen uso del espaciado y la indentación, estructuras de control transparentes, y rutas de ejecución normales y directas. La práctica de comentar bien el código fuerza a incluir comentarios que eduquen al próximo programador sobre los tópicos que no pueden ser inferidos del código mismo.

c. Reusabilidad:

La noción de módulo permite que programas que traten las mismas estructuras de información reutilicen las funciones empleadas en otros programas. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de funciones ya definidos donde estos están relacionados de una manera particular.

d. Modificabilidad:

Un software es modificable si permite la corrección de sus defectos con una carga limitada de trabajo. En otros campos de la ingeniería puede ser más barato cambiar un producto entero o una buena parte del mismo que repararlo, por ejemplo televisores, y una técnica muy utilizada para lograr modificabilidad es usar partes estándares que puedan ser cambiadas fácilmente. Sin embargo, en el software las partes no se deterioran, y aunque el uso de partes estándares puede reducir el costo de producción del software, el

concepto de partes reemplazables pareciera no aplicar a la modificabilidad del software. Otra diferencia es que el costo del software está determinado, no por partes tangibles sino por actividades humanas de diseño. Un producto de software consistente en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, sin embargo, el solo aumento del número de módulos no hace que un producto sea más modificable. Una modularización adecuada con definición adecuada de interfaces que reduzcan la necesidad de conexiones entre los módulos promueve la modificabilidad ya que permite que los errores estén ubicados en unos pocos módulos, facilitando la localización y eliminación de los mismos.

Consideremos un programa para calcular las remuneraciones de un empleado. En un primer análisis podemos diferenciar diversas tareas casi independientes:

- El acceso a los datos personales y profesionales de un empleado: por lo general estos datos, se encuentran almacenados en archivos o en bases de datos (colección de archivos administrados por un sistema). Los datos personales están asociados a su Rut, nombre, dirección, entre otros; y los datos profesionales están asociados al título profesional, el grado que le asigna la empresa, los años de experiencia; que generalmente determinan su sueldo base.
- Calcular los bonos del empleado: hay veces en que un empleado dispone de bonos especiales, como por ejemplo: bonos por cargas familiares, bonos de producción, bonos profesionales, bonos por zona de trabajo, bonos por aguinaldo (fiestas patrias y navidad), entre otros.
- Calcular los descuentos: es normal que al sueldo base de un empleado se le apliquen ciertos descuentos, como por ejemplo: descuentos por adelantos que solicitó el empleado, descuento por salud, descuento por AFP, descuentos por pagos, etc.
- Calcular pagos por horas extras de trabajo: cuando un empleado trabaja horas extras, la organización que lo contrata debe otorgarle un pago por ello. Normalmente la cantidad que se l pague está asociado a las políticas que aplica la empresa. Por ejemplo, una política de la empresa es que se pagará $6.000 pesos por hora extra trabajada si el empleado es un técnico.
- Calcular remuneración: Esta tarea consiste en calcular la remuneración de un empleado en particular para un determinado mes del año. Esta tarea debe considerar la información que le proporcionen las otras actividades descritas en los párrafos anteriores.
- Imprimir la liquidación de sueldo: consiste en imprimir todos los datos que tiene que ver con el empleado en particular, su sueldo base, bonos y descuentos; en algún formato que determine la empresa.

Es fácil observar que lo más razonable es resolver estos problemas por separado, estableciendo los canales de comunicación apropiados entre todos ámbitos de trabajo para combinar estas soluciones y lograr el objetivo final.

e. Módulo:

Un módulo es una colección de declaraciones, en principio ocultas respecto a toda acción o declaración ajena al propio módulo, es decir, definidas en un ámbito de visibilidad cerrado.

Emplearemos el concepto de módulo para materializar cada una de las unidades en las que se tendrá que descomponer toda tarea de programación mínimamente importante, de modo que una vez conectados convenientemente estos módulos resulte una estructura que resuelva el problema en cuestión.

f. Diseño modular:

El uso de módulos se acompaña de una metodología de desarrollo de problemas basada en la descomposición de éstos en unidades independientes y la resolución separada de los subproblemas que surgen. Denominamos "diseño modular" a tal estrategia.

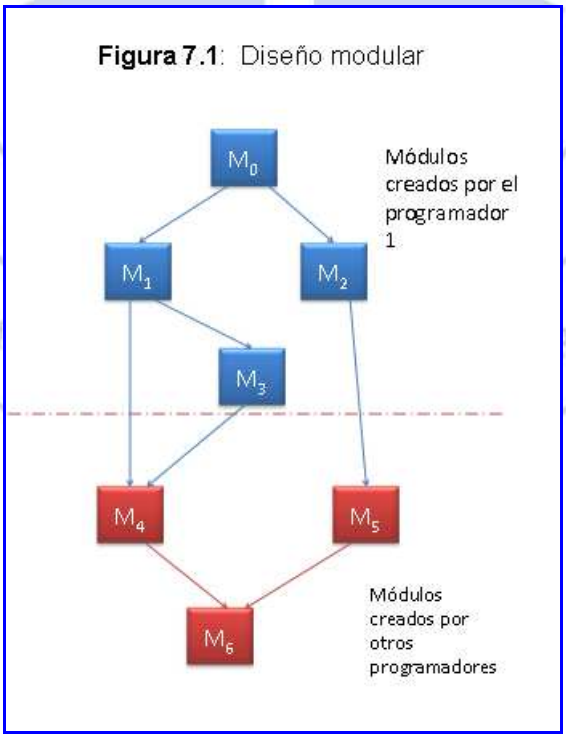
Algunas ventajas que podemos asociar al diseño modular son:

- Permite resolver el problema por partes. Esto nos obliga a determinar la estructura de cada parte o módulo. Cada módulo del problema será resuelto por una función.
- Permite el trabajo en equipo. Ya que cada módulo o función es desarrollada por un determinado programador. La solución completa estará determinada por la integración de todas las funciones del programa.
- Permite la reutilización. Esto se debe a que una función puede ser utilizada por varios programas. Esto permite acelerar el tiempo de desarrollo del software ya que podemos utilizar código fuente que ya está probado y funcionando.

El fundamento de un buen diseño modular consiste en contemplar nuestros futuros algoritmos como una jerarquía de módulos perfectamente comunicados entre sí, donde cada uno de ellos cuenta con un objetivo diferenciado, como si fueran piezas de una máquina que pudiesen ser utilizadas para construir otras máquinas.

Una visualización de esa idea se muestra en la figura 7.1, en la que se incorpora una típica organización de módulos.

Figura 7.1: Diseño modular



La figura presenta dos secciones. En la mitad superior observamos los módulos que el programador 1 ha diseñado para resolver el problema:

- Las declaraciones y acciones de nivel superior se hallan en M0
- M0 ha requerido de dos módulos M1 y M2 para funcionar
- Por su parte, M1 importa elementos de M3

Algunos de los problemas que ha surgido ya estaban resueltos por otros programadores, por lo que se necesitará acceder o llamar a los módulos de la parte inferior.

- M1 y M3 importan elementos de M4 y M2 de M5, respectivamente
- Por otro lado, M4 y M5 importan elementos de M6

Una flecha entre dos módulos **A → B** significa, por lo tanto, "**A emplea elementos de B**", o bien "**A llama al módulo B para que le entregue sus datos**".

Ejemplo 7.1:

Para el caso de cálculo de las remuneraciones de un empleado necesitamos varios módulos de programa. La Figura 7.2 muestra la relación de cada módulo.

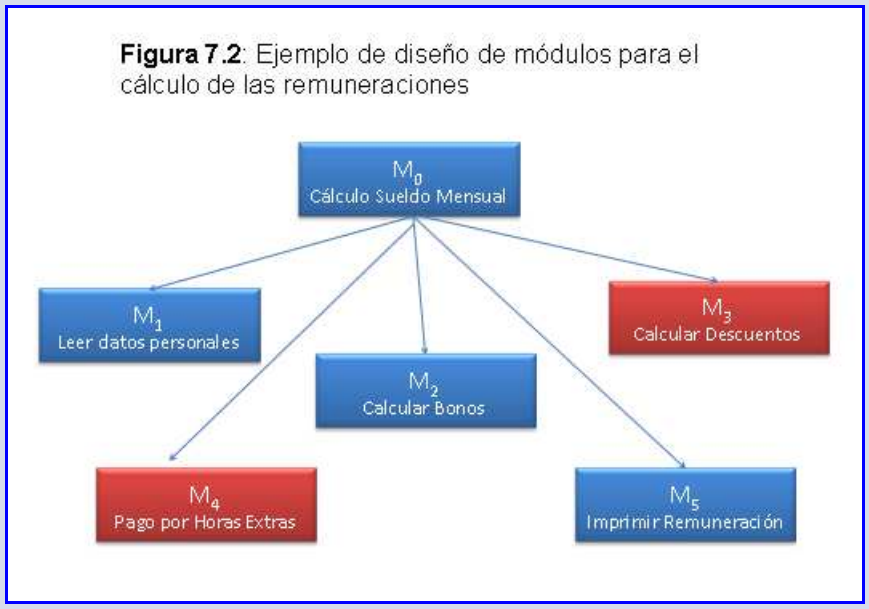


Figura 7.2: Ejemplo de diseño de módulos para el cálculo de las remuneraciones

Observamos que el Módulo Mo realiza la llamada a los otros módulos para calcular la remuneración considerando el sueldo base, bonos, descuentos y pago por horas extras. El módulo M2 es un módulo que está a cargo de calcular los bonos que recibe el empleado para el mes actual.

Observemos también que los módulos M3 y M4 han sido programados previamente por otro programador.

Cada módulo de la figura 7.2 representará un pequeño programa que será invocado por otro módulo. Para que no tengamos problemas con los datos que debe recibir o enviar cada módulo, será necesario seguir ciertas reglas que mencionaremos en la siguiente sección.

7.2 Introducción a los Subprogramas

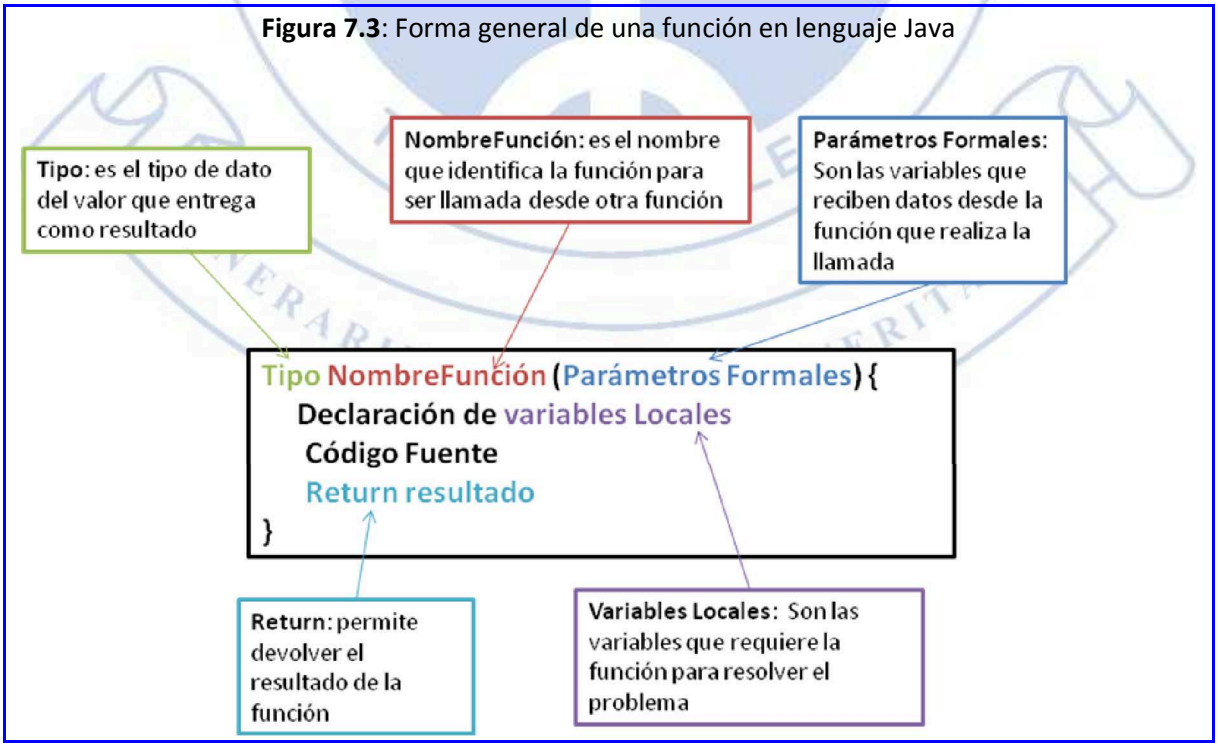
a. Subprograma:

Un subprograma es una colección de sentencias que posee varias características:

- Posee su propio nombre: El conjunto de sentencias considerado como subprograma debe disponer de un identificador exclusivo. Este identificador bastará para que el compilador o traductor sepa a qué subprograma se alude cuando un identificador aparezca en una llamada.
- Se puede invocar como un todo: Se puede solicitar la ejecución de esa colección de sentencias mencionando el nombre acordado para ellas. En la solicitud de ejecución, denominada llamada a la función, se proporcionan a la misma ciertos valores denominados argumentos. Cuando finaliza la ejecución de las sentencias, el control de flujo del programa vuelve a la primera sentencia posterior a la llamada al subprograma.
- Puede recibir y devolver información: El subprograma recibe, cuando es invocada, una colección de valores denominados argumentos. Y las variables que reciben los datos desde los argumentos se denominan parámetros formales.
- Dispone de sus propios espacios de memoria: Los subprogramas permiten declarar variables en su interior. Estas variables se denominan variables locales, y se caracterizan porque sólo se puede acceder a ellas desde el subprograma que las crea.

La figura 7.3 muestra la forma general de una función en lenguaje Java.

Figura 7.3: Forma general de una función en lenguaje Java



Ejemplo 7.2:
Llamada a una función que está almacenada en una librería
El siguiente código fuente muestra la llamada a la función sqrt para calcular la raíz de x. "x" es una variable declarada con algún tipo de dato, por ejemplo, de tipo int

```
resultado = sqrt(x); /*llamada a la función sqrt con x como argumento*/
```

Por tanto "x" sería el argumento, y contiene el dato del que necesitamos calcular la raíz cuadrada.
Note que el parámetro formal que se debe definir en la función sqrt debe ser de tipo compatible con el dato que se le almacenará, ya que debe ser capaz de almacenar el dato almacenado en la variable "x".

Otra forma de llamar a la función sqrt es así:

```
resultado = sqrt(5); /*llamada a la función sqrt con el dato 5 como argumento*/
```

En este caso, el argumento es el dato "5" que será recibido por el parámetro formal creado en la función sqrt.

Ejemplo 7.3:
Parámetro Formal para la función sqrt
El siguiente código fuente muestra el parámetro formal para la función sqrt

```
double sqrt (double num)
```

En este caso la variable "num" servirá para recibir el dato que envía el argumento.

De manera breve definiremos los conceptos que aparecen en la figura 7.3.

b. Argumentos:

Son los datos que enviaremos a la función cuando esta es invocada desde otra función. Observemos el ejemplo 7.2.

c. Parámetro Formal:

Los parámetros son variables que permiten a la función recibir datos cuando ésta es invocada (o llamada). Existen tres reglas para crear los parámetros de una función.

- La cantidad de Parámetros debe ser la misma que los Argumentos: Se requiere crear un parámetro formal por cada argumento que se cree en la llamada a la función. Por ejemplo, si disponemos de 2 argumentos en la llamada, quiere decir que dos son los datos que enviaremos a la función por medio de sus parámetros.
- Argumentos y Parámetros deben ser de tipo compatible: Cada parámetro recibirá un dato específico que dependerá del dato que tiene almacenado el argumento, por tanto el tipo de dato que se defina para el parámetro formal debe permitir almacenar el dato que proviene desde el argumento.
- El traspaso de datos desde los Argumentos hacia los Parámetros debe ser en orden: Cada argumento entrega los datos a sus respectivos parámetros de manera ordenada, uno por uno, por tanto, los parámetros deben ser declarados en el miso orden que se especifican los argumentos al momento de llamar a la función.

Observa el ejemplo 7.3.

d. Variable Local:

Son las variables que se declaran al interior de una función. Estas variables sólo pueden ser utilizadas por la función que las crea, y existirán cuando la función es invocada.

Revisemos el ejemplo 7.4.

Ejemplo 7.4:
Declaración de variables locales
El siguiente código muestra las variables locales que utiliza la función sqrt

```
double sqrt (double num) {
double resultado;
sentencias;
}
```

Observamos que las variables num y resultado han sido declaradas al interior del función sqrt. Ambas variables sólo pueden ser utilizadas por la función sqrt y existirán cuando ésta función sea invocada por otra función.

Ejemplo 7.5:
Declaración de una variable global
El siguiente código muestra la declaración de una variable global.

```
import java.lang.Math
int numero;
```

En este caso, numero es una variable global y puede ser utilizada en cualquier parte del programa.

e. Variable Global:

Son variables que se declaran fuera de cualquier función del programa, normalmente después de la sección de Librerías. Estas variables pueden ser utilizadas por cualquiera de las funciones escritas en el programa ya que existen mientras se ejecute el programa (desde que se ejecuta), lo cual es cómodo y supone un grave riesgo. Por lo tanto no dependen de la ejecución de una función específica. Observemos el ejemplo 7.5.

Nota:
Debemos considerar que el uso de variables globales puede traer problemas para realizar modificaciones en las funciones. Por tanto limitaremos el uso de éstas sólo para casos excepcionales. En efecto cualquier función podría hacer uso de una variable global considerando que era local, con resultados

7.3 Declaración de Funciones

La sintaxis de una declaración en Java tiene dos aspectos:

7.3.1 Encabezado de la función

El Encabezado de la función debe declararse en la primera línea de la función. Consta de tres bloques de información:

tipo_proporcionado nombre_de_la_función (parámetros formales)

- El tipo_proporcionado es el tipo al que pertenece el valor que toma (o restorna) la función cuando concluye su ejecución. Puede ser cualquier tipo de dato válido en Java, tanto predefinido (ejemplo: int, double, char) como definido por el usuario. También puede ser de tipo void si la función no retorna un resultado.

- El nombre de la función es un identificador válido en Java. Los nombres de funciones no deben ser palabras reservadas de lenguaje de programación (por ejemplo, read, nextInt, sqrt, int). Para hacer la llamada a una función se escribe el nombre de la función y a continuación una pareja de paréntesis. Si la función permite argumentos, los parámetros se colocan dentro de los paréntesis (pero sin escribir los nombres de las variables).
- Los parámetros de entrada o parámetros formales de la función son una colección de declaraciones de tipos de datos, si es que la función admite argumentos.

El ejemplo 7.6 presenta una función prototipo.

Ejemplo 7.6:
Funciones prototipo
Los siguientes ejemplos muestran diversas funciones prototipos.

```
int suma (int num1 , int num2)
char letra (char[] letras)
```

En este caso, la función suma recibe dos valores de tipo entero, que serán almacenados en los parámetros formales cuando ésta sea llamada. Además la función retornará como resultado otro dato de tipo entero.
Por otro lado, la función letra recibe un conjunto de caracteres por medio de un parámetro que es un arreglo de tipo char. La función retorna un carácter como resultado.

7.3.2 Cuerpo de la Función

El cuerpo de la función se define a continuación del encabezado de la función con una colección de sentencias entre llaves. La función finaliza su ejecución cuando el flujo de control alcanza la sentencia return. Si la función es de tipo void, se puede omitir la sentencia return; entonces la ejecución finaliza cuando se haya ejecutado la última sentencia de la función.

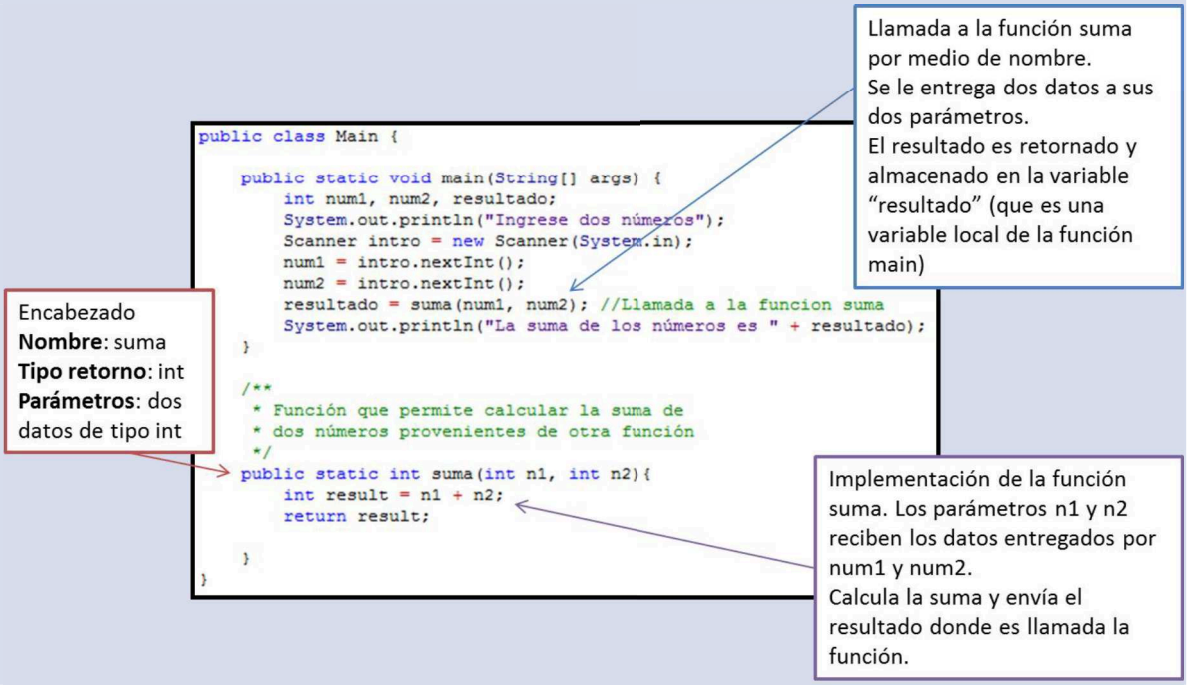
El código siguiente muestra de manera general el cuerpo de una función. Comúnmente llamamos a este hecho "implementación de la función".

```
tipo_proporcionado nombre_de_la_función (parámetros formales) {
    declaraciones de variables locales;
    sentencia1;
    sentencia2;
    ……..
    sentenciaN;
    return expresión_de_tipo_compatible_con_el_tipo_proporcionado;
}
```

Revisemos el ejemplo 7.7 que calcula la suma de dos números por medio de una función.

Ejemplo 7.7:
Programa que calcula la suma de dos números
El siguiente programa en Java (ver figura 7.4) dispone de una función para sumar dos números que ingrese el usuario.

Figura 7.4: Ejemplo de programa con una función




```
public class Main {

    public static void main(String[] args) {
        int num1, num2, resultado;
        System.out.println("Ingrese dos números");
        Scanner intro = new Scanner(System.in);
        num1 = intro.nextInt();
        num2 = intro.nextInt();
        resultado = suma(num1, num2); //Llamada a la funcion suma
        System.out.println("La suma de los números es " + resultado);
    }

    /**
     * Función que permite calcular la suma de
     * dos números provenientes de otra función
     */
    public static int suma(int n1, int n2){
        int result = n1 + n2;
        return result;

    }
}
```

La figura 7.4 presenta un ejemplo de un programa con una función que permite sumar dos números.
Lo primero, es definir el encabezado de la funcion en donde se especifica el nombre de la función, el tipo de dato que retornará y los tipos de cada parámetro formal. Esto es necesario para que el compilador conozca las características de la función. De esta manera no tenemos problemas de compilación en la línea de código cuando es llamada la función, ya que la palabra suma, no es una palabra reservada ni tampoco una variable.

Note que la llamada a la función se realiza con el nombre de la función más sus argumentos. Es aquí donde se detiene la ejecución de la función principal main y es ejecutada la función suma.

Al momento de ejecutarse la función suma se crean tres zonas de memoria, una para cada variable local, **n1, n2 y resul**. Las variables **n1 y n2** son utilizadas como parámetros formales por lo que están preparadas para almacenar los datos que enviemos a l función por medio de los argumentos. En este caso, **n1** recibe el dato de **num1 y n2** recibe el dato de **num2**.

n1 ← num1
n2 ← num2

Finalmente, la función calcula la suma de los datos almacenados en n1 y n2 (que son los mismos que ingresó el usuario) y retorna este resultado por medio de la sentencia return. En este caso, el resultado es recibido por la variable resultado que ha sido declarada en el main.

resultado ← resul

Nota:
No necesariamente los parámetros deben tener distintos nombres que los argumentos. Si escogemos que tengan los mismos nombres entonces hemos declarado las variables num1 y num2 para que los use la función main, y las variables num1 y num2 para que las utilice la función suma.
En este caso tenemos zonas de memoria con el mismo nombre, pero para distintas funciones.

7.4 Funciones sin parámetros ni resultados

Se pueden crear funciones que no admiten parámetros ni proporcionan resultados. El encabezado de este tipo de funciones es el siguiente:

```
void nombre_de_la_función ()
```

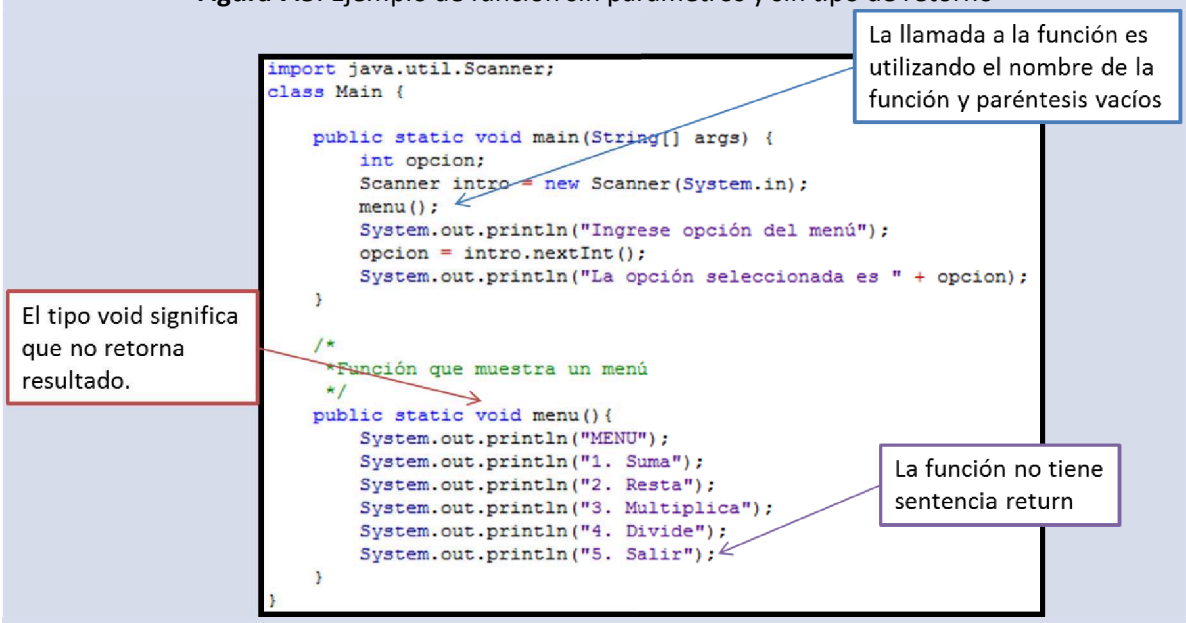
y el cuerpo será de la siguiente forma:

```
{
   variables locales;
    instrucción_1;
    instrucción _2;
   ……….
   instrucción _N;
}
```

El ejemplo 7.8 describe un programa con una función si parámetros.

Ejemplo 7.8:
Función sin parámetros ni resultados.
Es muy común crear una función para imprimir un menú. El siguiente programa muestra un ejemplo de ello.

Figura 7.5: Ejemplo de función sin parámetros y sin tipo de retorno




```
import java.util.Scanner;
class Main {

    public static void main(String[] args) {
        int opcion;
        Scanner intro = new Scanner(System.in);
        menu();
        System.out.println("Ingrese opción del menú");
        opcion = intro.nextInt();
        System.out.println("La opción seleccionada es " + opcion);
    }

    /*
     *Función que muestra un menú
     */
    public static void menu(){
        System.out.println("MENU");
        System.out.println("1. Suma");
        System.out.println("2. Resta");
        System.out.println("3. Multiplica");
        System.out.println("4. Divide");
        System.out.println("5. Salir");
    }
}
```

Observe que en la figura 7.5 hay una función **menu** del tipo **void**. Esto quiere decir que la función se ejecutará pero no retornará ningún resultado. La función tampoco dispone de parámetros.

Observe también que la llamada a la función se realiza simplemente escribiendo el nombre de la función y los paréntesis vacíos.

```
menu(); //llamada a la función menu
```

7.5 Funciones con parámetros y sin resultados

El encabezado de una función con parámetros y sin resultados es como sigue:

```
void nombre_funcion (tipo1, tipo2, …, tipoN)
```

y el cuerpo de la función será:

```
{
 variables locales;
  instrucción 1;
  instrucción 2;
 ……
instrucción n;
}
```

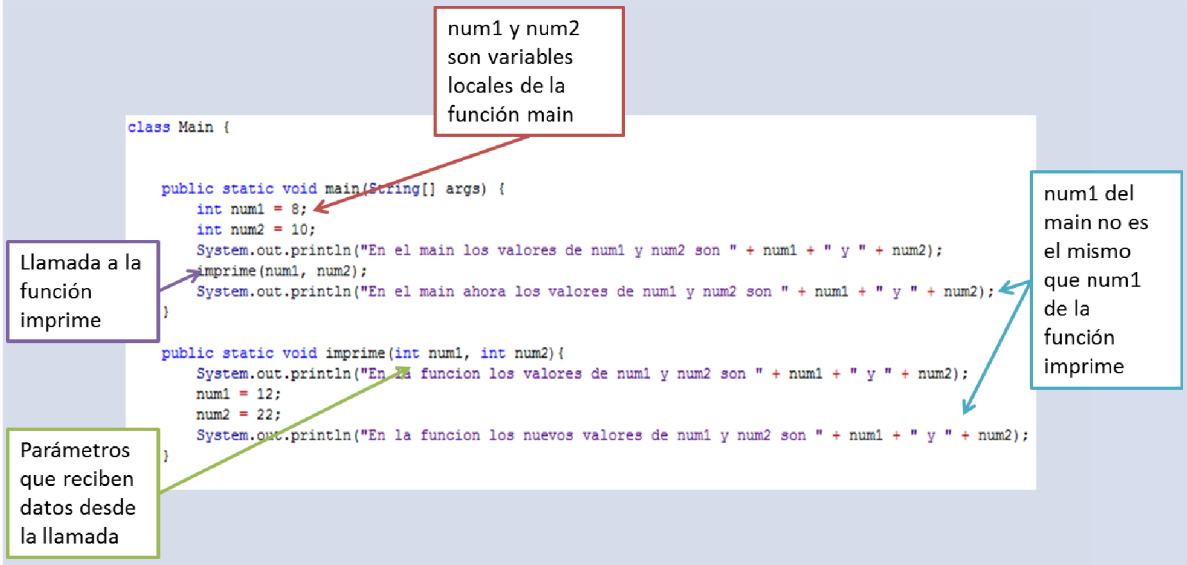
Esta función puede comunicarse con el resto del programa por medio de los argumentos (arg1, arg2, .. argN). De esta manera no es necesario recurrir a variables globales; es suficiente con los argumentos, que permiten efectuar un paso de información en ambos sentidos (de la función A que llama a la función B, y de la función B que es llamada por la función A).

7.5.1 Paso por Valor

El paso de información por valor consiste en evaluar los argumentos y traspasar el valor a los parámetros formales. Luego de ello, se ejecuta el código de la función. Esto implica que ha ocurrido un paso de información desde el exterior.

Ejemplo 7.9:
El siguiente ejemplo muestra cómo es traspasada la información desde los argumentos hacia los parámetros formales.

Figura 7.6: Función que imprime los valores de los parámetros



Observamos desde la figura 7.6 que las variables num1 y num2 declaradas en el main son variables locales que puede utilizar sólo el main. Estas variables existen durante toda la ejecución de programa ya que el main en la función principal.

Por otro lado, **num1 y num2** son parámetros formales de la función imprime. Estas son variables locales que puede ser utilizadas sólo por la función imprime y no por otra función. Cabe destacar que las variables del main no son las mismas que las variables de la función ***imprime***, a pesar de tener el mismo nombre. La razón de ello, es que se crean en diferentes zonas de memoria.

La figura 7.7 muestra un ejemplo de ejecución del programa de la figura 7.6

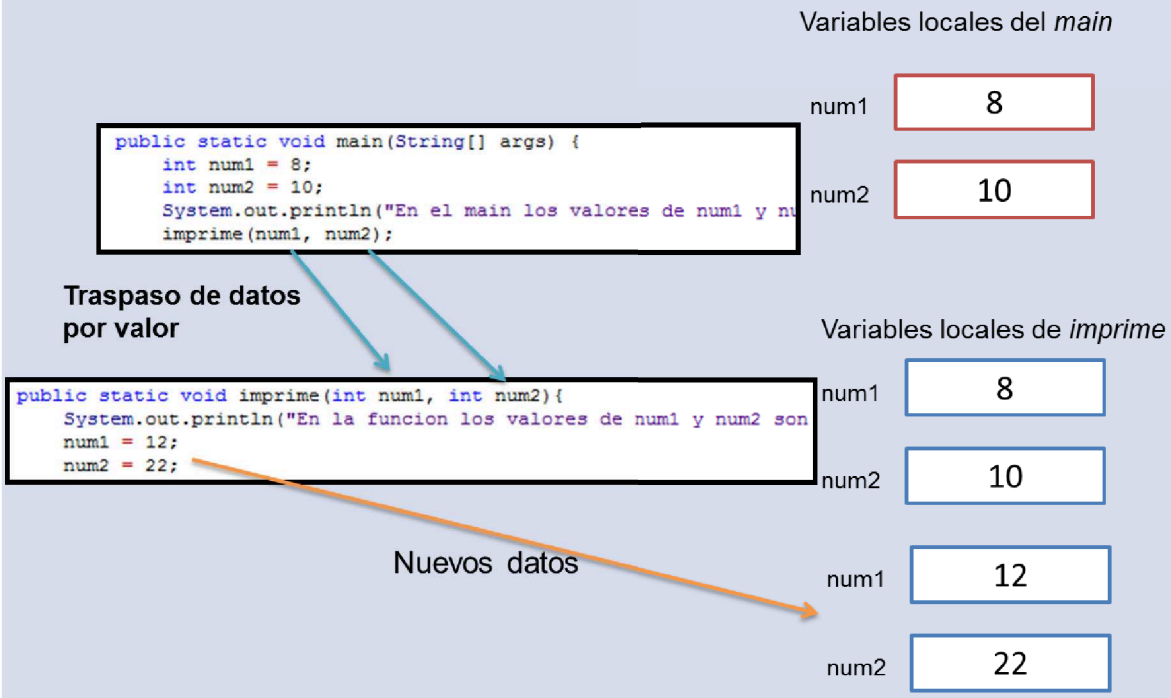
Figura 7.7: Ejemplo de ejecución del programa en la figura 7.6

```
En el main los valores de num1 y num2 son 8 y 10
En la funcion los valores de num1 y num2 son 8 y 10
En la funcion los nuevos valores de num1 y num2 son 12 y 22
En el main ahora los valores de num1 y num2 son 8 y 10

Process completed.
```

De la figura 7.7 observamos que los parámetros num1 y num2 de la función imprime, sí almacenan los datos traspasados por las variables num1 y num2 del main. Pero al cambiar los datos de num1 y num2 por 12 y 22 respectivamente, observamos que sólo ocurren dentro de la función, y no modifica los datos de las variables (también num1 y num2) en el main.

La siguiente figura representa lo descrito anteriormente.

Figura 7.8: Representación de las variables locales en las funciones main e imprime



En la figura 7.8 las zonas de memoria de las variables num1 y num2 del main están representadas en color rojo, y las variables locales (parámetros) en la función ***imprime*** están representadas en color azul, ya que utilizan distintas zonas de memoria.

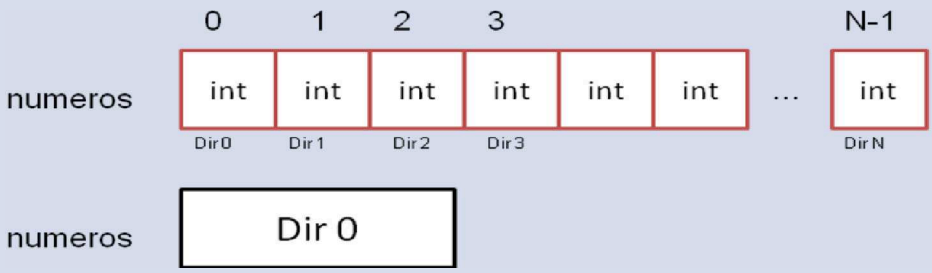
7.5.1.1 Paso de Arreglos

Vimos en la sección anterior que no es posible cambiar el valor de una variable externa desde una función.

Sin embargo, existe un caso especial cuando se tiene un arreglo como parámetros de la función, ya que todo arreglo dispone de un puntero implícito (ya que es definido por defecto).

En este capítulo no utilizaremos de manera directa los punteros para cambiar los datos de las variables externas (argumentos), puesto que no es parte de los objetivos de este libro. Pero, ya que un arreglo dispone de un puntero implícito, aprovecharemos esta ventaja para manipular un conjunto de datos durante toda la ejecución de un programa.

Figura 7.9: Representación del arreglo *numeros* de tipo int y su puntero implícito



Observamos en la figura 7.9, que el arreglo numeros contiene N celdas que se representan por su índice (desde cero hasta N). Además, cada celda está ubicada en alguna dirección de memoria (Dir 0 hasta Dir N). En este caso, el puntero implícito del arreglo, que también se llama numeros, almacena la dirección de memoria de la celda cero. De esta manera es posible acceder a los datos del arreglo desde cualquier parte (funciones) del programa, ya que accedemos de manera directa a los datos almacenados.

Revisemos el siguiente ejemplo para comprender de mejor manera este hecho.

Ejemplo 7.10:

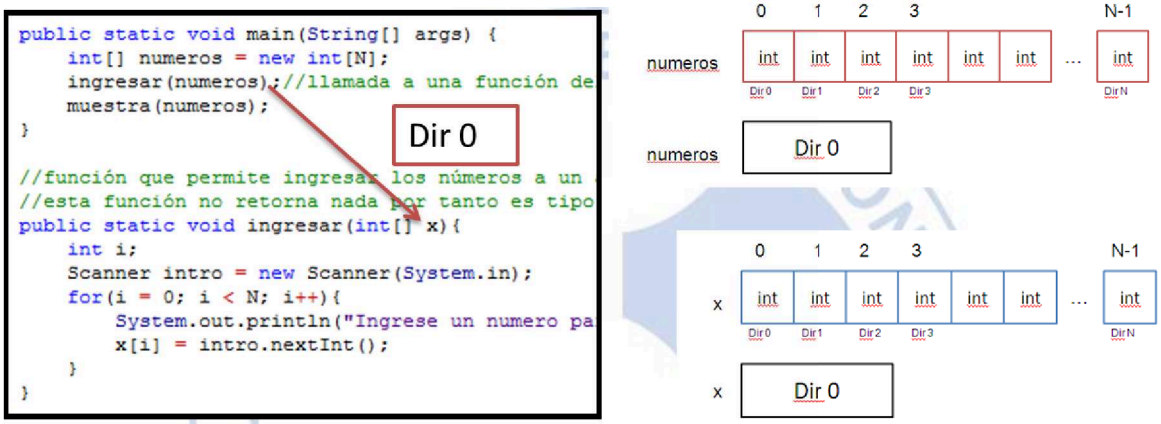
El siguiente programa permite ingresar datos a un arreglo por medio de una función con parámetros de referencia.

```
import java.util.Scanner;
class Main {

    final static int N = 8;//cantidad de celdas para el arreglo

    public static void main(String[] args) {
        int[] numeros = new int[N];
        ingresar(numeros);//llamada a una función del tipo void
        muestra(numeros);
    }

    //función que permite ingresar los números a un arreglo
    //esta función no retorna nada por tanto es tipo void
    public static void ingresar(int[] x){
        int i;
        Scanner intro = new Scanner(System.in);
        for(i = 0; i < N; i++){
            System.out.println("Ingrese un numero para la celda " + i);
            x[i] = intro.nextInt();
        }
    }

    //función que muestra el arreglo
    //no retorna resultado por tanto es de tipo void
    public static void muestra(int[] x){
        int i;
        System.out.println("Números en el arreglo");
        for(i = 0; i < N; i++){
            System.out.println("X[" + i + "] = " + x[i]);
        }
    }
}
```

En el programa anterior, el arreglo numeros es una variable local que sólo puede utilizar la función principal main. Este arreglo es traspasado a la función ingresar por medio del parámetro formal x, que es un arreglo definido dentro de la función, por tanto, es un variable local que puede ser utilizada sólo dentro de la función ingresar. Note que el arreglo declarado en la función mostrar no es el mismo que el de la función ingresar, a pesar de tener el mismo nombre. Esto es, ya que ambos arreglos se crean en momentos distintos (sólo cuando se ejecuta la función) y en diferentes zonas de memoria.

La figura 7.10 muestra una representación del traspaso de datos desde los argumentos a los parámetros.

Figura 7.10: Ejemplo de traspaso de datos de arreglos



De la figura podemos observar que:

- Para incluir un arreglo como argumento en la llamada a una función, se debe escribir sólo el nombre del arreglo sin los corchetes. Esto quiere decir, que en realidad estamos pasando como argumento el puntero implícito de arreglo.
- Lo que se traspasa desde el argumento numeros al parámetro x, es la dirección de memoria de la primera celda de numeros (ver representación a la derecha de la figura).
- Por lo tanto, cuando el arreglo x se cree, utilizará la misma zona de memoria del arreglo numeros.

De lo anterior, podemos afirmar que cualquier cambio que ocurra en el arreglo *x* (parámetro formal) será reflejado automáticamente en el arreglo *numeros* (argumento) ya que se lleva a cabo un traspaso de datos por referencia y no por valor.

El proceso de ejecución del programa en el ejemplo 7.10 es el siguiente. Primero se ejecuta la función principal main, creando el arreglo numeros en alguna zona de memoria del computador. Cuando la función ingresar es llamada, la ejecución del main se detiene; en ese momento se crean las variables locales de la función ingresar (arreglo x y la variable i). El arreglo x se crea en la misma zona de memoria del arreglo numeros ya que recibe la dirección de memoria de la primera celda del arreglo. La función se ejecuta y el usuario ingresa los N datos al arreglo x. Cada dato ingresado es automáticamente almacenado en el arreglo numeros. Una vez finalizada la ejecución de la función ingresar, el arreglo x y la variable i desaparecen, pero los datos ingresados por el usuario han quedado almacenados en el arreglo declarado en el main. Luego continúa la ejecución de las siguientes instrucciones en la función main.

7.6 Funciones sin parámetros y con resultados

Las funciones que proporcionan un resultado y carecen de parámetros tienen un encabezado como el que sigue:

tipo nombre_funcion ()

Donde tipo, es el tipo de dato de lo que retorna la función al finalizar la ejecución de sus instrucciones.

> **Nota**:
> Tenga en cuenta que una función es capaz de devolver un solo resultado. Esto quiere decir, que no se admiten funciones de la forma:
>
> **tipo1, tipo2 nombre_funcion ();**

El cuerpo de la función será como el que sigue:

```
{
  instrucción 1;
  instruccón 2;
  instrucción 3;
  ……..
  instrucción N;
  return resultado //valor compatible con el tipo proporcionado
}
```

Las funciones que retornan un resultado se pueden llamar de varias formas:

- Desde un System.out.println para imprimir el resultado después de ejecutar la función
  System.out.println(llamada_función ());
- Almacenando el resultado retornado en otra variable
  Otra_var = llamada_función ();
- Dentro de una condición lógica
  if (llamada_función () > 0)

Observa el ejemplo 7.11.

Ejemplo 7.11:
Programa que retorna un número entero y no tiene parámetros formales.
El siguiente programa utiliza una función que selecciona el número mayor entre 5 números generados de manera aleatoria.

```
import java.util.Random;
class Main {

    public static void main(String[] args) {
        System.out.println("El mayor de 5 números generados de manera aleatoria es " +  mayorNum());
    }

    /*
     * Función que retorna el mayor de 5 números
     * generados de manera aleatoria
     */
    public static int mayorNum(){
        int numeros[] = new int[5];
        int mayor;
        int i;
        Random rand = new Random();

        //Primero generamos los 5 números
        for(i = 0; i < 5; i++){
            //rand genera un número aleatorio. En esta caso de 1 a 100
            numeros[i] = rand.nextInt(100);
            System.out.println("Número generado: " + numeros[i]);
        }

        //Despues buscamos el mayor
        mayor = numeros[0];
        for(i = 0; i < 5; i++){
            if(mayor < numeros[i]){
                mayor = numeros[i];
            }
        }
        return mayor;
    }
}
```

En la figura anterior observamos que la función mayorNum es llamada sin utilizar argumentos (mayorNum()). La función fue creada con el fin de generar 5 números aleatorios y luego retornar el mayor de ellos, por lo que no necesita datos externos que sean recibidos por medio de parámetros formales. Una vez generados los 5 números y almacenados en el arreglo numeros, la función busca el número mayor y lo retorna al main (línea de código que la invocó), donde es impreso el resultado.

La figura 7.13 muestra un ejemplo de ejecución del programa anterior.

Figura 7.13: Ejemplo de ejecución de una función sin parámetros y que retorna un resultado

```
Número generado: 17
Número generado: 25
Número generado: 12
Número generado: 19
Número generado: 96
El mayor de 5 números generados de manera aleatoria es 96
```

7.7 Funciones con parámetros y resultados

Son las de uso más frecuente. Su prototipo es como sigue:

```
tipo nombreFunción (parámetros formales);
```

Y su cuerpo tiene el siguiente aspecto:

```
tipo nombreFunción (parámetros formales) {
  variables locales;
  instrucción 1;
   instrucción 2;
   instrucción 3;
   .....
   instrucción N;
   return valor_compatible_con_tipo;
}
```

Recuerde que las funciones que retornan un resultado se pueden llamar de varias formas:

- Desde un System.out.println para imprimir el resultado después de ejecutar la función
  System.out.println(llamada_función (parámetros));
- Almacenando el resultado retornado en otra variable
  Otra_var = llamada_función (parámetros);
- Dentro de una condición lógica
  if (llamada_función (parámetros) > 0)

Observa el ejemplo 7.12.

Ejemplo 7.12:
Incluiremos un ejemplo que muestra el uso de varias funciones con parámetros formales, que retornan resultado y que no retornan nada.
Una persona posee una cuenta de ahorro en el banco "Ahorra Feliz". Para mantener información detallada de su cuenta le solicita a usted crear un programa en C++ que permita por medio de **un arreglo y funciones**:

a) Ingresar los abonos mensuales a la cuenta de ahorro durante el año 2010.
b) Mostrar el mes en que se ingresó el mayor abono a la cuenta.
c) Mostrar el promedio de su cuenta de ahorro en el año.
d) Mostrar la varianza de la cuenta de ahorro.
e) Mostrar la desviación estándar de su cuenta de ahorro anual.

Promedio, Varianza y Desviación estándar:

Esta medida nos permite determinar el promedio aritmético de fluctuación de los datos respecto a su punto central o media. La desviación estándar nos da como resultado un valor numérico que representa el promedio de diferencia que hay entre los datos y la media. Para calcular la desviación estándar basta con hallar la raíz cuadrada de la varianza, por lo tanto su ecuación sería: $S = \sqrt{S^2}$

Para comprender el concepto de las medidas de distribución vamos a suponer que la persona que solicita el programa desea saber que tanto varían los abonos mensuales en su cuenta de ahorro. Como ejemplo, supongamos que los abonos en dólares para los primeros 5 meses son los que se muestran en la siguiente fórmula que calcula el promedio:

$$\overline{X} = \frac{490 + 500 + 510 + 515 + 520}{5} = \frac{2535}{5} = 507$$

La varianza sería:

$$S_X^2 = \frac{\sum_{i=1}^{n} \left(X_i - \overline{X}\right)^2}{n-1}$$

$$S^2 = \frac{(490-507)^2+(500-507)^2+(510-507)^2+(515-507)^2+(520-507)^2}{(5-1)}$$

$$S^2 = \frac{(-17)^2+(-7)^2+(3)^2+(8)^2+(13)^2}{4} = \frac{289+49+9+64+169}{4} = \frac{580}{4} = 145$$

Por lo tanto la desviación estándar de los 5 primeros abonos sería: $S = \sqrt{145} = 12.04 \cong 12$

Con lo que concluiríamos que el abono promedio de los 5 primeros meses es de 507 dólares, con una tendencia a variar por debajo o por encima de dicho abono en 12 dólares.

Nota: para solucionar el problema, considere que el promedio, varianza y desviación estándar se calculan en funciones por separado, por lo que debemos añadir parámetros formales para pasar los resultados del promedio y varianza.

La solución sería la siguiente:

```
import java.util.Scanner;
class Main {
    static final int N = 12;

    /* Función principal */
    public static void main(String[] args) {
        int[] ahorro = new int[N];
        float prom;
        float vari;
        ingresar(ahorro);//Llamada a una función que no retorna resultados
        System.out.println("El mes en que se ingresó el mayor ahorro es " + mayorAbono(ahorro));
        //Llamada a funciones que si retornan resultados
        prom = promedio(ahorro);
        System.out.println("El promedio de dinero en el cuenta de ahorro para el 2009 es " + prom);
        vari = varianza(ahorro, prom);
        System.out.println("La varianza es " + vari);
        System.out.println("y la desviacion varianza es " + desviacion(vari));
    }

    /* Esta función permite ingresar datos al arreglo */
    public static void ingresar(int[] cuenta){
        int i;
        Scanner intro = new Scanner(System.in);
        for(i = 0; i < N; i++){
            System.out.println("Ingrese abono para el mes " + (i+1));
            cuenta[i] = intro.nextInt();
        }
    }

    /* Busca el mayor abono y retorna el mes en que ocurre */
    public static int mayorAbono(int[] cuenta){
        int i;
        int mayor = cuenta[0];
        int mes = 0;
        for(i = 1; i < N; i++){
            if(cuenta[i] > mayor){
                mayor = cuenta[i];
                mes = i;
            }
        }
        return mes;
    }

    /* Calcula el promedio de los abonos ingresados */
    public static float promedio(int[] cuenta){
        int i;
        int suma = 0;
        float prom;
        for(i = 1; i < N; i++){
            suma = suma + cuenta[i];
        }
        prom = (float) suma/N;
        return prom;
    }

    /* Calcula la varianza de los abonos */
    public static float varianza(int[] cuenta, float prom){
        int i;
        int suma = 0;
        float resultado;
        for(i = 0; i < N; i++){
            suma = suma + (cuenta[i] - (int)prom) * (cuenta[i] - (int)prom);
        }
        resultado = (float) suma/N-1;
        return resultado;
    }

    /* Calcula la desviación en base a la varianza calculada */
    public static double desviacion(float vari){
        return Math.sqrt(vari);
    }
}
```

Observamos que las funciones *mayor_abono, promedio, varianza* y *desviación* tienen parámetros formales para recibir valores desde el *main*, y además retornan un resultado.

7.8 Ejercicios Resueltos

Ejercicio 7.1:
Se requiere un programa que permita mostrar el número mayor de 4 números que ingresa el usuario. Para solucionar el problema debe utilizar una función que se llama "mayor" que ya ha sido creada y que devuelve el número mayor de dos números.

```
int mayor (int num1, int num2) {
if (num1> num2)
   return num1;
else  return num2;
}
```

Para solucionar este problema, necesitaremos ingresar los 4 números por teclado y luego llamar a la función mayor las veces que sea necesario para encontrar el número más grande entre los 4 ingresados.
La solución sería comparar de a dos números e ir almacenando el mayor de esos dos en una variable auxiliar o en una variable que ya no estemos utilizando, por ejemplo, numero1 que será la variable para ingresar el primer número.

La solución sería así:

```
import java.util.Scanner;
class Main {

    public static void main(String[] args) {
        int numero1;
        int numero2;
        int numero3;
        int numero4;
        Scanner intro = new Scanner(System.in);
        System.out.println("Ingrese 4 números");
        numero1 = intro.nextInt();
        numero2 = intro.nextInt();
        numero3 = intro.nextInt();
        numero4 = intro.nextInt();
        numero1 = mayor(numero1, numero2);
        numero1 = mayor(numero1, numero3);
        numero1 = mayor(numero1, numero4);
        System.out.println("El mayor de los 4 es " + numero1);
    }

    public static int mayor(int num1, int num2){
        if(num1 > num2)
            return num1;
        else
            return num2;
    }
}
```

Ejercicio 7.2:

Se requiere un programa (con funciones) para calcular el valor de la siguiente serie:

Y ← 1 + 1/2! + 1/3! + …. + 1/n!

Donde n es un número ingresado por el usuario y debe ser mayor que 1000.

Para solucionar el problema necesitaremos calcular el factorial de los números 1 hasta n. Además debemos sumar los términos de la serie, es decir 1/ x!

Para ello, crearemos dos funciones, la primera que calcula el factorial de cualquier número mayor o igual que cero; y la segunda que calcula la serie (suma los términos)

La solución sería así:

```
import java.util.Scanner;
class Main {

    public static void main(String[] args) {
        int n;
        Scanner intro = new Scanner(System.in);
        do {
          System.out.println("Ingrese un número mayor que 1000");
           n = intro.nextInt();
        } while (n < 1000);
        System.out.println("El valor de la serie es " + serie(n));

    }

    public static double factorial(int x){
        int i;
        double factor = 1;
        if(x < 2){
            return 1;
        }else{
            for(i = 1; i <= x; i++){
                factor = factor * i;
            }
        }
        return factor;
    }

    public static double serie(int n){
        int i;
        double suma = 0;
        for(i = 1; i <= n; i++){
            suma = suma + (double) 1/factorial(i);
        }
        return suma;
    }
}
```

Note que la función principal main sólo realiza la llamada a la función serie. Por tanto es la función serie que realizará sucesivas llamadas a la función factorial.

Ejercicio 7.3:
Una empresa de servicios desea almacenar la cantidad de reclamos que realizan los clientes durante una semana. Para ello dispone de una tabla de valores como la que sigue:

	Lunes	*Martes*	*Miércoles*	*Jueves*	*Viernes*	*Sábado*	*Domingo*
10:00-13:00	4	6	2	34			
13:00 – 15:00		10		7			
15:00-20:00			29				

La cantidad de reclamos son almacenados por día de la semana de acuerdo a la jornada laboral. Por ejemplo, para el día martes en la jornada laboral de 10:00 a 13:00 hrs. se registraron 6 reclamos.
Con estos datos, el gerente general requiere que Ud. cree un programa que pueda realizar las siguientes actividades:

a) Ingresar la cantidad de reclamos por día y jornada laboral.
b) Mostrar el promedio de reclamos para la jornada laboral de 15:00 a 20:00 hrs
c) Mostrar el día de la semana que tuvo más reclamos.

Para cada uno de estos requerimientos crearemos una función.
Una solución al problema en lenguaje Java sería la siguiente:

```
import java.util.Scanner;
class Main {
    static final int JORNADAS = 3;
    static final int DIAS = 7;

    static Scanner intro;

    public static void main(String[] args) {
        intro = new Scanner(System.in);
        int[][] reclamos = new int[JORNADAS][DIAS];
        //primero realizamos la llamada a la función ingresar
        //recuerde que los datos quedarán almacenados en la matriz reclamos
        //que pertenece al main
        ingresar(reclamos);3

        //luego llamamos a la función promedioReclamos con los argumentos
        // reclamos: que es la matriz con valores
        // 2: que es la jornada que se solicita promedio
        System.out.println("El promedio de reclamos para la jornada de 15:00 " +
            "a 20:00 hrs es " + promedioReclamos(reclamos, 2));

        //finalmente llamamos a la función diaReclamos para buscar el dia de la semana
        // que tuvo más reclamos
        System.out.println("El dia de la semana que tuvo más reclamos fue " + diaReclamos(reclamos));
    }

    /* Función que permite ingresar los reclamos a la matriz */
    public static void ingresar(int[][] rcm ){
        int i;
        int j;
        for(j = 0; j < DIAS; j++){
            System.out.println("Ingrese los reclamos para el dia " + (j+1));
            for(i = 0; i < JORNADAS; i++){
                System.out.println("Ingrese el reclamo para la jornada " + (i+1));
                rcm[i][j] = intro.nextInt();
            }
        }
    }
```

```
    /* Función que calcula el promedio de reclamo para la jornada de
     * 15:00 a 20:00 hrs, es decir la jornada 2 */
    public static float promedioReclamos(int[][] rcm, int jornada){
        int i;
        int suma = 0;
        int dia = 0;
        for(i = 0; i < DIAS; i++){
            suma = suma + rcm[jornada][i];
        }
        return (float) suma/DIAS;
    }

    /* Función que busca el dia(columna) en que hay más reclamos */
    public static int diaReclamos(int[][] rcm){
        int i;
        int j;
        int suma;
        int mayor = 0;
        int dia = 0;

        for(j = 0; j < DIAS; j++){
            suma = 0;
            for (i= 0; i< JORNADAS; i++){
                suma = suma + rcm[i][j];
            }
            if(mayor < suma){
                mayor = suma;
                dia = j+1;
            }

        }
        return dia;
    }
}
```

Ejercicio 7.4:

Se requiere una función (sólo la función) que permita contar la cantidad de veces que se repite una letra en una palabra. La función debe tener el siguiente encabezado:

int vecesLetra (String palabra, char letra)

Donde "palabra" es el parámetro que contiene la palabra que ingresa el usuario y "letra" es el parámetro que contiene la letra a contar en la palabra.

Debemos considerar que el parámetro **palabra** contendrá la cadena de caracteres completa, y que **letra** contiene el carácter a contar. Así debemos comparar ese carácter con cada carácter de **palabra**.

```
public static int vecesLetra(String palabra, char letra){
    int cont = 0;
    for(int i = 0; i < palabra.length(); i++){
        if(letra == palabra.charAt(i)){
            cont++;
        }
    }
    return cont;
}
```

Ejercicio 7.5:

Se ingresan a un arreglo las ventas diarias (total $ de cada día) que realiza un vendedor de una tienda de CDs, se pide crear un programa que permita:

a) Ingresar las ventas diarias del vendedor realizadas durante un mes (30 días) al arreglo. (Nota: recuerde validar que los números ingresados sean mayores o iguales a cero)
b) Calcular el promedio de ventas logradas durante los primeros 15 días del mes.
c) Mostrar las ventas diarias mayores a $345.000
d) Mostrar el día en que logró la mayor venta.

Cada requerimiento será una función. Una solución en Java sería la siguiente:

```
import java.util.Scanner;
class Main {
    static final int N = 30;

    public static void main(String[] args) {
        int[] ventaCd = new int[N];

        //Primero ingresamos las ventas en el arreglo
        //Estas ventas quedaran almacenadas en el arreglo
        //ventaCd que pertenece al12 main
        ingresar(ventaCd);

        //Luego debemos llamar a la función promedio15Dias
        //para calcular el promedio de las ventas durante
        //los primeros 15 dias
        System.out.println("El promedio de ventas durante los primeros 15 dias: " +

            23promedio15Dias(ventaCd));

        //luego llamamos a la función mostrar_ventas para mostra las ventas  diarias
        //mayores a 345000
        mostrarVentas(ventaCd);

        //finalmente llamamos a la función mayor_venta para mostrar el dia en que se
        //produce la mayor venta
        System.out.println("El de mayor venta es: " + mayorVenta(ventaCd));
    }

    //función que permite ingresar las ventas al arreglo
    public static void ingresar(int[] x){
        int i;
        Scanner intro = new Scanner(System.in);
        for(i = 0; i < N; i++){
            do{
                System.out.println("Ingrese venta de cd para dia " + (i+1) +
                    " Ingrese número <=0");
                x[i] = intro.nextInt();
            }while(x[i] < 0);
```

```
        }
    }

    //función que calcula el promedio de las primeras 15 ventas
    public static float promedio15Dias(int[] x){
        int i;
        int suma = 0;
        for(i = 0; i < 15; i++){
            suma = suma + x[i];
        }
        return (float) suma/15;
    }

    //función que muestra las ventas mayores a 345000
    public static void mostrarVentas(int[] x){
        int i;
        for(i = 0; i < N; i++){
            if(x[i] > 345000){
                System.out.println("Venta dia " + (i+1) + " es " + x[i]);
            }
        }
    }

    public static int mayorVenta(int[] x){
        int i = 0;
        int dia = 0;
        int mayor = x[0];
        for(i = 1; i < N; i++){
            if(mayor < x[i]){
                mayor = x[i];
                dia = i;
            }
        }
        return dia+1;
    }
}
```

Ejercicio 7.6:

Una **empresa de seguros** necesita organizar los pagos de sus empleados. En una matriz de **30x12** almacena las remuneraciones mensuales de cada uno de ellos. En la matriz, las filas representan los empleados y las columnas los meses del año. Cree un programa en lenguaje Java que permita realizar las siguientes acciones a través de funciones:

a) Ingresar las remuneraciones de los empleados en la matriz.
b) Encontrar el número del empleado (fila) que ganó más dinero en el año.
c) Encontrar el número del mes en el que la empresa pagó menos dinero a sus empleados. (Total del mes más pequeño).
d) Calcular el promedio anual de remuneraciones.

Cada requerimiento es posible implementarlo en una función. Una posible solución en Java sería la siguiente:

```
import java.util.Scanner;
class Main {
    static final int EMPLEADOS = 2;
    static final int MESES = 12;

    public static void main(String[] args) {
        int[][] pagos = new int[EMPLEADOS][MESES];
        //primero llamamos a la función ingresar para que el usuario ingrese los
        //pagos a la matriz
        //estos pagos quedarán almacenados en la matriz pagos que está en el main
        ingresar(pagos);

        //ahora llamamos a la función gana_mas para encontrar el empleado que
        //gana mas en el año
        System.out.println("El empleado que gana mas en el año es " + ganaMas(pagos));

        //finalmente llamamos a la función promedio_anual para calcular el promedio anual
        //de remuneraciones
        System.out.println("El promedio anual de remuneraciones es " + promedioAnual(pagos));
    }

    //función que permite ingresar los pagos en la matriz
    public static void ingresar(int[][] p){
        int i;
        int j;
        Scanner intro = new Scanner(System.in);
        for(i = 0; i < EMPLEADOS; i++){
            System.out.println("Ingrese remuneracions para el empleado " + (i+1));
            for(j = 0; j < MESES; j++){
                System.out.println("Ingrese pago del mes " + (j+1));
                p[i][j] = intro.nextInt();
            }
        }
    }
```

```
    //function que busca el empleado que gana mas
    public static int ganaMas(int[][] p){
        int suma = 0;
        int mayor = 0;
        int i;
        int j;
        int emple = 0;
        for(i = 0; i < EMPLEADOS; i++){
            suma = 0;
            for(j = 0; j < MESES; j++){
                suma = suma + p[i][j];
            }
            if(mayor < suma){
                mayor = suma;
                emple = i;
            }
        }
        return emple+1;
    }

    // function que busca en el mes de menor pago
    public static int menorPago(int[][] p){
        int suma = 0;
        int menor = 9999999;
        int i;
        int j;
        int mes = 0;
        for(j = 0; j < MESES; j++){
            suma = 0;
            for(i = 0; i < EMPLEADOS; i++){
                suma = suma + p[i][j];
            }
            if(menor > suma){
                menor = suma;
                mes = j;
            }
        }
        return mes;
    }

    // función que calcula el promedio anual de remuneraciones
    public static float promedioAnual(int[][] p){
        int suma = 0;
        int i;
        int j;

        for(i = 0; i < EMPLEADOS; i++){
            for(j = 0; j < MESES; j++){
                suma = suma + p[i][j];
            }
        }
        return (float) suma/(30*12);
    }
}
```

7.9 Ejercicios Propuestos

Ejercicio 7.7:
En un arreglo de 30 celdas se almacena las ventas diarias logradas por un vendedor de artículos de cocina durante el mes de abril. La empresa requiere un programa que permita responder a los siguientes requerimientos:

a) Ingresar las ventas diarias del vendedor
b) Mostrar el día del mes en que logró la mayor venta
c) Calcular el total de ventas del mes
d) Mostrar los días del mes en el que se logró ventas inferiores a los $10.000
e) Calcular la remuneración del empleado para el mes de abril si se sabe que se le paga una comisión del 1% de las ventas totales logradas.

Ejercicio 7.8:
Escribir un programa con funciones, de tal manera que calcule y saque por pantalla los salarios de los 10 empleados de una empresa. Cada empleado se identificará por un número desde el 1 hasta el total de empleados en la empresa. El salario es función de las horas trabajadas, que serán distintas para cada trabajador y se introducirán por teclado. Cuando un trabajador trabaje más de 40 horas, la empresa pagará las horas extras de acuerdo a la siguiente tabla de valores:

Cantidad horas extras	Pago por hora ($)
Entre 1 y 3	$2.000
Entre 4 y 7	$3.000
Más de 7	$3.500

Nota: el pago por hora ordinaria es de $5.000 para cada trabajador

Considere tres funciones, uno para ingresar las horas totales trabajadas en un arreglo de 10 celdas; la segunda para calcular el pago de horas extras guardando el resultado en otro arreglo, y otro para el cálculo de los salarios de todos los empleados mostrándolos por pantalla.

Ejercicio 7.9:
Cree un programa con funciones, que permita a un usuario realizar las siguientes acciones con una matriz de 10 x 10
a. Ingresar sólo números positivos entre 0 y 250
b. Calcular el promedio de los números para una fila que ingresa el usuario
c. Sumar los números de la diagonal.

Ejercicio 7.10:
Cree un programa que realice lo siguiente:
En el **programa principal** el usuario debe ingresar una palabra (de hasta 30 caracteres, no se permite una palabra mas larga). Luego debe **imprimir** por pantalla la cantidad de vocales (cantidad de “a”, “e”, “i”, “o” y “u”) que hay en la palabra, y la cantidad de caracteres totales.

La función **Cuenta_Caracter**, entrega la cantidad de veces que se repite una letra en una palabra cualquiera. El encabezado es el siguiente:

Int Cuenta_Caracter (char carac, String pal)

Donde **carac** es la variable que recibe el caracter a contar(es decir vocal) en la palabra **pal**.
Especificación: Llame desde el programa principal la función Cuenta_Caracter cuantas veces sea necesario (una vez para cada vocal).

La función **Cuenta_Letras**, entrega la cantidad de letras que contiene la palabra ingresada.
El encabezado es el siguiente:

Int Cuenta_Letra(String pl)

Ejemplo:

```
Ingrese una palabra:   Temuco
Cantidad de a: 0
Cantidad de e: 1
Cantidad de i: 0
Cantidad de o: 1
Cantidad de u: 1
Cantidad de caracteres: 6
```

7.10 Comentarios Finales

En este capítulo te introdujimos un nuevo elemento, la función, que permite afrontar cualquier tarea compleja mediante la descomposición de ésta en las unidades de diseño y organización de datos.

El diseño modular que hemos repasado, te ha aportado dos ventajas: completar y ampliar al diseño descendente como método de resolución de problemas algorítmicos; y proteger la estructura de la información asociada a un determinado problema, limitando las operaciones que puedan actuar sobre ellas.

Hemos presentado los conceptos básicos de funciones, como estructura de la función, parámetros formales, variables locales, y tipo de la función; además de la forma como llamar a una función, en los casos en que retorna un valor o no.