Guia4 [BACKEND1]

Subprogramas

Material de

lectura

## 

## OBJETIVOS DE LA GUÍA

En esta guía aprenderemos a:

* Definir y utilizar funciones, manejo de retornos
* Definir y utilizar de procedimientos
* Realizar llamados de funciones y procedimientos

## 

# Subprogramas

Un método muy útil para solucionar un problema complejo es dividirlo en subproblemas —problemas más sencillos— y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar “divide y vencerás”.

El problema principal se soluciona por el correspondiente programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, conocidos como **procedimientos** o **funciones**. Un subprograma es un como un mini algoritmo, que recibe los *datos,* necesarios para realizar una tarea, desde el programa y devuelve los *resultados* de esa tarea.

## Funciones

Las funciones o métodos son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente según los parámetros definidos en la función, esta recibe argumentos, cuyos valores se utilizan para efectuar operaciones y adicionalmente retornan un valor. En otras palabras, una función según sus parámetros puede recibir argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y retorna un valor usando la instrucción return, si no retorna es otro tipo de función. Los tipos que pueden usarse en la función son: int, doble, long, boolean, String y char.

A estas funciones les vamos a asignar un tipo de acceso y un modificador. Estos dos conceptos los vamos a ver mejor más adelante, pero por ahora siempre vamos a crear las funciones con el acceso public y el modificador static. **Para saber más sobre estos dos temas, leer la explicación del método main.**

[acceso][modificador][tipo] nombreFuncion([tipo] nombreArgumento, …….){

/\*

\* Bloque de instrucciones

\*/

return valor;

}

**Consejos acerca de return:**

· Cualquier instrucción que se encuentre **después de la ejecución de return NO será ejecutada**. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.

· El tipo de valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.

| public static void main(String[] args) {    Scanner leer = new Scanner(System.in);  int num1 = 5;  int num2 = 7;    //Puedo invocar el retorno de esta funcion de esta manera  System.out.println("La suma de ambos es: " + sumar(num1, num2));    int retorno = sumar(num1, num2);   System.out.println("La suma de ambos es: " + retorno);  }  // Pero, recomendamos hacerlo de esta manera, ya que los retornos deben alojarse en variables para su posterior uso   public static int sumar(int num1, int num2) {  int suma;  suma = num1 + num2;  return suma;  } |
| --- |

## Ejercicio 11

Escribir un programa que procese una secuencia de caracteres ingresada por teclado y terminada en punto, y luego codifique la palabra o frase ingresada de la siguiente manera: cada vocal se reemplaza por el carácter que se indica en la tabla y el resto de los caracteres (incluyendo a las vocales acentuadas) se mantienen sin cambios.

| a | e | i | o | u |
| --- | --- | --- | --- | --- |
| @ | # | $ | % | \* |

Realice un subprograma que reciba una secuencia de caracteres y retorne la codificación correspondiente. Utilice la estructura “según” para la transformación.

Por ejemplo, si el usuario ingresa: Ayer, lunes, salimos a las once y 10.

La salida del programa debería ser: @y#r, l\*n#s, s@l$m%s @ l@s %nc# y 10.

===================================================

package codificadorvocales;

// @author Horacio

import java.util.Scanner;

public class CodificadorVocales {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Ingrese una secuencia de caracteres terminada en punto: ");

        String entrada = sc.nextLine();

        String salida = codificar(entrada); //llama al subprograma codificar

        System.out.println("La secuencia codificada es: " + salida);

    }

    public static String codificar(String entrada) {

        String salida = "";

        for (int i = 0; i < entrada.length() - 1; i++) { //itera sobre cada caracter de la entrada excepto el punto

            char caracter = entrada.charAt(i);

            switch (caracter) { //utiliza la estructura "switch" para transformar cada caracter según corresponda

                case 'a':

                case 'A':

                    salida += '@';

                    break;

                case 'e':

                case 'E':

                    salida += '#';

                    break;

                case 'i':

                case 'I':

                    salida += '$';

                    break;

                case 'o':

                case 'O':

                    salida += '%';

                    break;

                case 'u':

                case 'U':

                    salida += '\*';

                    break;

                default:

                    salida += caracter;

                    break;

            }

        }

        salida += "."; //agrega el punto al final

        return salida;

    }

}

El programa comienza pidiendo al usuario que ingrese una secuencia de caracteres terminada en punto. Luego, llama al subprograma codificar() para procesar la entrada y devolver la secuencia codificada.

El subprograma codificar() itera sobre cada caracter de la entrada (excepto el punto) y utiliza la estructura "switch" para transformar cada vocal en su respectivo caracter codificado. Si el caracter no es una vocal, se mantiene sin cambios. Al final, se agrega el punto al final de la secuencia codificada y se devuelve.

Por ejemplo, si el usuario ingresa "Ayer, lunes, salimos a las once y 10.", la salida del programa sería "@y#r, l\*n#s, s@l$m%s @ l@s %nc# y 10."

Ejemplo: Del método concat de la clase String.

**concat**en Java, usando**StringBuilder**paraca**+**, especialmente cuando se realizan muchas concatenaciones en un bucle

public class OptimizedConcat {

    public static String concat(String... strings) {

        StringBuilder sb = new StringBuilder(); // Crear un StringBuilder

        for (String s : strings) {

            sb.append(s); // Agregar cada string al StringBuilder

        }

        return sb.toString(); // Convertir el contenido del StringBuilder a un string

    }

    public static void main(String[] args) {

        String s1 = "Hola";

        String s2 = "mundo";

        String s3 = "!";

        String resultado = concat(s1, " ", s2, s3); // Llamar al método concat con los strings a concatenar

        System.out.println(resultado); // Imprimir el resultado: "Hola mundo!"

    }

}

En esta implementación**StringBuilder**para concatenar los**concat**. El **StringBuilder**permite**String**temporales en el proceso de concatenación. Esto puede mejorar**toString()**del**StringBuilder**para convertir el**StringBuilder**en una cuerda final

Revisemos lo aprendido hasta aquí:

* Diseñar funciones.
* Nombrarlas correctamente.
* Determinar y definir retornos.
* Comprender el funcionamiento de los parámetros.
* Poder mostrar el retorno de la función, alojar su retorno y llamarlo desde el Algoritmo principal.
* Saber elegir cuándo debe utilizarse una función.

## =================================================

## Procedimientos (Funciones sin retorno)

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo, en la definición no hay nada que se lo impida. En el contexto de Java un procedimiento es básicamente una función cuyo tipo de retorno es void, los que indica que devuelven ningún resultado.

[acceso][modificador] void nombreFuncion([tipo] nombreArgumento){

/\*

\* Bloque de instrucciones

\*/

}

**Acerca de los argumentos o parámetros:**

· Hay algunos detalles respecto a los argumentos de un método, veamos:

· Una función, un método o un procedimiento pueden tener una cantidad infinita de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.

· Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.

· Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo del método.

· Al recibir un argumento nada nos obliga a hacer uso de éste al interior del método, sin embargo, para que recibirlo si no lo vamos a usar.

· En java los argumentos que sean variables de tipos primitivos (int, double, char, etc.) **se van a pasar por valor**, mientras que los objetos (String, Integer, etc.) y los arreglos se van a pasar **por referencia**. Nota: el concepto de objetos lo vamos a ver más adelante.

Ejemplo:

| public static void main(String[] args) {   String nombre = "Mariano";    int edad = 29;    mostrarInfo(nombre, edad);  }     public static void mostrarInfo(String nombre, int edad) {    System.out.println("El nombre del usuario es: " + nombre + "y su edad:" + edad);    } |
| --- |

Manos a la obra:

## Ejercicio 12

Crea un procedimiento *EsMultiplo* que reciba los dos números pasados por el usuario, validando que el primer número múltiplo del segundo e imprima si el primer número es múltiplo del segundo, sino informe que no lo son.

package multiplo;

// @author HoraXio

 import java.util.Scanner; // Importación de la clase Scanner para leer entrada del usuario

public class Multiplo {

    public static void EsMultiplo(int num1, int num2) { // Declaración del procedimiento EsMultiplo

        if (num1 % num2 == 0) { // Verificación si num1 es múltiplo de num2

            System.out.println(num1 + " es múltiplo de " + num2); // Impresión del resultado si num1 es múltiplo de num2

        } else {

            System.out.println(num1 + " no es múltiplo de " + num2); // Impresión del resultado si num1 no es múltiplo de num2

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in); // Creación de un objeto Scanner para leer entrada del usuario

        // Lectura de los dos números ingresados por el usuario

        System.out.println("Ingrese el primer número: ");

        int num1 = sc.nextInt();

        System.out.println("Ingrese el segundo número: ");

        int num2 = sc.nextInt();

        EsMultiplo(num1, num2); // Llamada al procedimiento EsMultiplo con los dos números ingresados por el usuario

    }

}

=========================================================================================================================

En este programa, se define un procedimiento llamado EsMultiplo que recibe dos números enteros num1 y num2.

Dentro del procedimiento, se utiliza la condición num1 % num2 == 0 para verificar si num1 es múltiplo de num2.

Si es cierto, se imprime un mensaje indicando que num1 es múltiplo de num2, de lo contrario se imprime un mensaje

indicando que num1 no es múltiplo de num2. En el método main,

se crea un objeto Scanner para leer los dos números ingresados por el usuario,

y luego se llama al procedimiento EsMultiplo con los dos números ingresados.

Recuerda que utilizaremos los procedimientos [Funciones y Procedimientos] **únicamente** cuando el objetivo del mismo, es mostrar información.

Revisemos lo aprendido hasta aquí:

* Diseñar procedimientos.
* Nombrarlos correctamente.
* Comprender el funcionamiento de los parámetros.
* Llamar al procedimiento desde el algoritmo principal.
* Saber elegir cuándo debe utilizarse un procedimiento.

Para más información ver vídeo de enlace: https://youtu.be/4VH6-i7g1Bk

\*  En ésta clase se aprende a ver cómo es una estructura básica de una función en java.

 \*/

package ejemplovideos;

// @author Horacio

import java.util.Scanner;

public class EjemploVideos { //2º Este método [método main] está contenido en una clase.

    public static void main(String[] args) { //1º Metodo main, 1er método que se va a ejecutar cuando ejecutemos nuestra aplicación.

// 10° llamar a las funciones desde nuestro método "main". Capturamos el valor generando variable de tipo entero "int" llamado "resultado"

 int resultado = suma(10, 10);  // 11° le asignamos el producto de la función.

        System.out.println(resultado);// 12° Mostramos el resultado por pantalla.

        saludoPersonalizado("Chiquito"); //13° llamamos al otro método o función "saludo personalizado", e imprimimos por pantalla.

        Scanner leer = new Scanner(System.in); //14° Si quisiéramos introducir por teclado algo, creamos una variable de tipo Scanner y la importamos.

        String nombre = leer.next (); // 15° Creamos una variable de tipo String y mandamos a la función.

        saludoPersonalizado(nombre); //16° De esta manera se pasa el "nombre" a la función.

        saludoPersonalizado(leer.next()); //17° Otra forma de llamar a la función es crear.

    }             //4º [int] lo alberga el valor. 5º suma(recibe dos variables de tipo entero)

public static int suma(int a, int b) {

//3º [static] hace que no este instanciado mediante un objeto, sin no que se encuentra instanciado de forma estatic en la memoria.

//6º lógica (creamos una variable de tipo entero "resultado" y guardamos la suma de a + b.

int resultado = a + b;

return resultado;// 7º para retornar "resultado".

}

// las funciones sirven para reutilizar códigos

public static void saludoPersonalizado(String nombre){ // 8º creamos un método y agregamos (String con la variable nombre)

    System.out.println("Hola " + nombre + " ¿cómo estas?.");// 9º llamamos al System out.println y concatenamos el saludo.

}

}