

Práctica #6 – Programación modular



RESULTADOS DE APRENDIZAJE:

Al finalizar esta sesión de práctica el estudiante deberá:

- ✓ Crear código que permita la captura de datos.
- ✓ Realizar operaciones de asignación de datos.
- ✓ Comprender el uso de la clase MATH de Java.
- ✓ Realizar operaciones matemáticas en Java.

PROGRAMACIÓN MODULAR

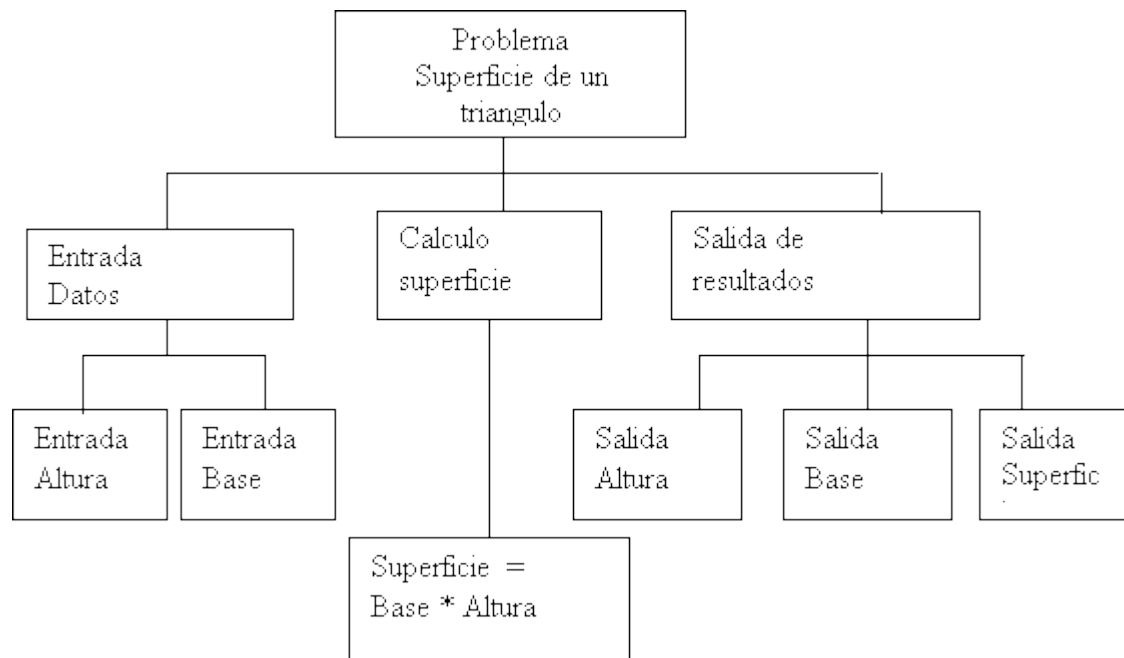
La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que esta puede resolver.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Esta técnica se llama refinamiento sucesivo, divide y vencerás o análisis descendente (Top-Down).

Un 'módulo' es cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema complejo original. Cada uno de estos módulos tiene una tarea bien definida y algunos necesitan de otros para poder operar. En caso de que un módulo necesite de otro, puede comunicarse con éste mediante una interfaz de comunicación que también debe estar bien definida.

Si bien un módulo puede entenderse como una parte de un programa en cualquiera de sus formas y variados contextos, en la práctica se los suele tomar como sinónimos de procedimientos y funciones. Pero no necesariamente ni estrictamente un módulo es una función o un procedimiento, ya que el mismo puede contener muchos de ellos. No debe confundirse el término "módulo" (en el sentido de programación modular) con términos como "función" o "procedimiento", propios del lenguaje que lo soporta.



EJEMPLO

Vamos a partir de un programa que se ha escrito sin usar Programación Modular ni Programación Orientada a Objetos. Se trata de un programa que recibe un número entero, y el usuario puede decidir evaluar si el número es perfecto, abundante o deficiente. El programa original a continuación:

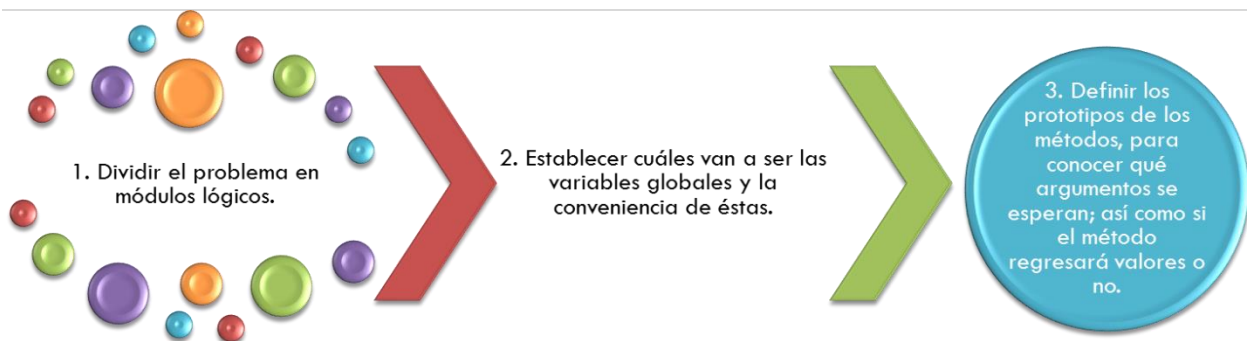
```
package modular1;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Modular1 {

    public static void main(String[] args) throws IOException{
        BufferedReader buffer = new BufferedReader(new
        InputStreamReader(System.in));
        String entrada;
        int opcion, numero, suma = 0, i;
        System.out.println("Programa que detecta los siguientes tipos de
        numeros:");
        System.out.println("1. Perfecto");
        System.out.println("2. Defectivo");
        System.out.println("3. Abundante");
        System.out.println("Seleccione su opcion 1, 2 o 3: ");
        entrada = buffer.readLine();
        opcion = Integer.parseInt(entrada);
        System.out.println("-----");
        System.out.println("Escriba el numero a evaluar: ");
        entrada = buffer.readLine();
        numero = Integer.parseInt(entrada);
        switch( opcion ){
            case 1: {
                for (i = numero-1; i>=1; i--)
                    if(numero%i == 0)
                        suma += i;
                if(numero == suma)
                    System.out.println("El numero: " + numero + " es
        perfecto");
            }
            else
                System.out.println("El numero: " + numero + " NO es
        perfecto");
            break;
            case 2: {
                for (i = numero-1; i>=1; i--)
                    if(numero%i == 0)
                        suma += i;
                if(numero > suma)
                    System.out.println("El numero: " + numero + " es
        defectivo");
            }
            else
                System.out.println("El numero: " + numero + " NO es
        defectivo");
        }
    }
}
```

```
break;
case 3: {
    for (i = numero; i>1; i--)
        if(numero%i == 0)
            suma += i;
    if(numero > 2 * numero)
        System.out.println("El numero: " + numero + " es
abundante");
    else
        System.out.println("El numero: " + numero + " NO es
abundante");
}
break;
default: System.out.println("Opcion no validad");
}
}
```

¿CÓMO CREAR UN PROGRAMA MODULAR?



PASO 1: DIVIDIR EL PROBLEMA EN MÓDULOS LÓGICOS.

Un módulo lógico es un conjunto de líneas de código que se escriben para resolver una parte del problema. Según el lenguaje de programación, un módulo puede llamarse procedimiento, función, subprograma, subrutina, etc.

En el lenguaje Java, tenemos los métodos. Un método en sí no fue diseñado para constituir un módulo, sino como parte de una clase.

La estructura del programa original ayuda mucho a encontrar los módulos lógicos, pues se detectan tres subtarefas claves:

- La porción de código que detecta si un número es perfecto
- Las líneas de código que detectan si un número es defectivo
- Las líneas de código que detectan si un número es abundante

De esta forma ya hemos detectado los tres métodos que integrarán nuestro programa.

```
14 System.out.println("2. Defectivo");
15 System.out.println("3. Abundante");
16 System.out.println("Seleccione su opcion 1, 2 o 3: ");
17 entrada = buffer.readLine();
18 opcion = Integer.parseInt(entrada);
19 System.out.println("-----");
20 System.out.println("Escriba el numero a evaluar: ");
21 entrada = buffer.readLine();
22 numero = Integer.parseInt(entrada);
23 switch( opcion ){
24     case 1: {
25         for (i = numero-1; i>=1; i--)
26             if(numero%i == 0)
27                 suma += i;
28         if(numero == suma)
29             System.out.println("El numero: " + numero + " es perfecto");
30         else
31             System.out.println("El numero: " + numero + " NO es perfecto");
32     }
33     break;
34     case 2: {
35         for (i = numero-1; i>=1; i--)
36             if(numero%i == 0)
37                 suma += i;
38         if(numero > suma)
39             System.out.println("El numero: " + numero + " es defectivo");
40         else
41             System.out.println("El numero: " + numero + " NO es defectivo");
42     }
43     break;
44     case 3: {
45         for (i = numero-1; i>=1; i--)
46             if(numero%i == 0)
47                 suma += i;
48         if(numero > 2 * numero)
49             System.out.println("El numero: " + numero + " es abundante");
50         else
51             System.out.println("El numero: " + numero + " NO es abundante");
52     }
53     break;
54     default: System.out.println("Opcion no validad");
55 }
56 }
57 }
```

PASO 2. IDENTIFICAR Y DEFINIR VARIABLES GLOBALES

Tipos de variables

Según el paradigma de programación modular, existen dos tipos de variables:

- Variables globales: las cuales son accesibles por todos los módulos dentro del programa para usar y/o modificar su valor.

- Variables locales: las cuales solo son accesibles dentro del módulo donde fueron declaradas. De hecho, este tipo de variables solo se crean cuando el flujo del programa entra al módulo correspondiente, y se destruyen cuando se sale de él.

Ámbito de las variables globales

Java no tiene explícitamente variables globales como tal, porque sus variables pueden ser: **privadas**, **públicas** o **protegidas**. Aunque el *ámbito* de acción de una variable pública es incluso mayor que el de una variable global, este tipo de variable nos servirá para definir variables globales para programación modular.

Ahora, acerca del ámbito de una variable: simplemente establece la zona o zonas del programa donde la variable existe y puede ser accedida. De esta forma una variable global tiene un ámbito de acción que incluye todos los métodos del programa. Esta variable es reconocida y puede ser usada por cualquier método, ya sea imprimiendo su valor, usándolo para realizar algún cálculo o para evaluar alguna expresión condicional. Y, sobre todo, cualquier método tiene *derecho* de modificar el valor almacenado en la variable global.

Ámbito de las variables locales

Las cosas son diferentes cuando se trata de variables locales o variables de método. Para este tipo de variables, sucede lo siguiente:

- Solo son reconocidas por el método que las declaró. Cualquier intento de usarlas en otro método diferente, te generará error de sintaxis, indicando que la variable no ha sido declarada.
- Ocupan un espacio temporal de almacenamiento en la memoria RAM. Cuando el flujo del programa entra en el método, se crea la casilla de almacenamiento para estas variables. Una vez que el flujo del programa sale del método, estas casillas (y su contenido) son destruidas.
- Si una variable local tiene el mismo nombre que una variable global, la local se antepone a la global. Eso quiere decir que el método NO tendrá acceso a la variable global, porque su local está *ocultando* a la variable global.

Encontrando las variables globales

Para elegir qué variables serán locales y cuáles globales, es necesario revisar el código nuevamente y observar en qué porciones del código se utilizan las variables.

```
package modular1;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Modular1 {


    public static void main(String[] args) throws IOException{
        BufferedReader buffer = new BufferedReader(new
        InputStreamReader(System.in));
        String entrada;
```


```
int opcion, numero, suma = 0, i;
System.out.println("Programa que detecta los siguientes tipos de
numeros:");
System.out.println("1. Perfecto");
System.out.println("2. Defectivo");
System.out.println("3. Abundante");
System.out.println("Seleccione su opcion 1, 2 o 3: ");
entrada = buffer.readLine();
opcion = Integer.parseInt(entrada);
System.out.println("-----");
System.out.println("Escriba el numero a evaluar: ");
entrada = buffer.readLine();
numero = Integer.parseInt(entrada);
switch( opcion ){
    case 1: {
        for (i = numero-1; i>=1; i--)
            if(numero%i == 0)
                suma += i;
        if(numero == suma)
            System.out.println("El numero: " + numero + " es
perfecto");
        else
            System.out.println("El numero: " + numero + " NO es
perfecto");
    }
    break;
    case 2: {
        for (i = numero-1; i>=1; i--)
            if(numero%i == 0)
                suma += i;
        if(numero > suma)
            System.out.println("El numero: " + numero + " es
defectivo");
        else
            System.out.println("El numero: " + numero + " NO es
defectivo");
    }
    break;
    case 3: {
        for (i = numero; i>1; i--)
            if(numero%i == 0)
                suma += i;
        if(numero > 2 * numero)
            System.out.println("El numero: " + numero + " es
abundante");
        else
            System.out.println("El numero: " + numero + " NO es
abundante");
    }
    break;
    default: System.out.println("Opcion no validad");
}
}
```

Como puedes observar, se ha subrayado con rojo las variables que serán globales. ¿Por qué esas? Si te fijas, las variables `buffer` y `entrada` las utilizamos para habilitar la entrada desde el flujo del teclado. Podrían declararse locales, pero entonces luego tendría que enviar como parámetro el dato obtenido, lo cual es más eficiente, pero no es el tema de este post. Por otro lado, la variable `número`, se utiliza en todas las partes del programa que identificamos como nuestros módulos. Entonces, como regla, procuraré que las variables que se definan globales sean aquellas que son utilizadas por todos los módulos del programa.

Las variables globales las declararemos fuera de los métodos, como se observa en el código final. Las establecemos de tipo *public* o públicas, para que otras clases también puedan usarlas. Y agregamos el modificador *static*, ya que el método `main` está definido también de esa forma, y una variable no *static* no puede ser utilizada por un método *static*. El modificador *static* indica que sólo se creará una copia de estas variables en la máquina virtual de Java.

PASO 3. DEFINIR LOS MÉTODOS

Encabezado del método  `tipoDeAcceso modificador tipoMétodo nombreMétodo(tipoArg arg){`

Cuerpo del método  `// declaración de variables globales`
`// código del método`
`}`

Como se puede observar, cada método se “encapsula” usando sus propias llaves de apertura y cierre. También tiene su zona de declaración de sus propias variables (locales).

Encabezado del método

Lo que define cómo será usado por otros métodos (incluyendo `main`), es su encabezado. He colocado en un color diferente cada una de las partes de este, y a continuación las explico:

- **tipoAcceso:** Define qué clases y métodos que podrán hacer uso de éste método. Existen tres tipos de modificadores de acceso: `public`, `private` y `protected`. Por el momento, utilizaremos el tipo `public`.
- **modificador:** Este modificador puede ser `static` o `final`, que técnicamente definen los atributos de los miembros de una clase. Por el momento establecemos una regla muy simple: si el método que lo llamará es *static*, entonces este método también debe ser *static*.
- **tipoMétodo:** El tipo de método se define por el tipo de valor que regresa un método. Si el éste no regresa un valor, será de tipo `void`. Si el método regresa un valor, deberá usarse cualquiera de los tipos de datos primitivos de Java: `int`, `float`, `char`, `double`, `float` y `boolean`. También incluso puede usarse una clase como `String`, `BufferedReader`, etc.
- **nombreMétodo:** Obviamente es el nombre que cada programador le dará al método. Se deben seguir las mismas reglas para definir identificadores en Java para crear estos nombres (recuerda: empezar con letra, continuar con letras, números y/o el carácter `_`, no usar caracteres especiales ni palabras reservadas)
- **tipoArg:** si el método lo requiere, en esta sección se definen los argumentos o parámetros del método. Específicamente, `tipoArg` define el tipo del argumento, el cual puede ser cualquiera de

los tipos de datos primitivos de Java así como también cualquier clase definida en la API de Java. Al tipo de dato del parámetro, le sigue el nombre del argumento, que una vez más, deberá seguir las reglas para construir identificadores.

Definiciones de los métodos

A continuación, se muestra el programa completo, con las variables globales al inicio y con las definiciones de cada uno de los métodos que se han detectado.

```
package modular2;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Modular2 {
    public static BufferedReader buffer = new BufferedReader(new
InputStreamReader(System.in));
    public static String entrada;
    public static int numero;

    public static boolean perfecto(){
        int i;
        int suma = 0;

        for (i = numero-1; i>=1; i--)
            if(numero%i == 0)
                suma += i;
        return suma == numero;
    }

    public static boolean defectivo(){
        int i;
        int suma = 0;

        for (i = numero-1; i>=1; i--)
            if(numero%i == 0)
                suma += i;
        if (numero > suma)
            return true;
        else
            return false;
    }

    public static boolean abundante(){
        int i;
        int suma = 0;

        for (i = numero; i>1; i--)
            if(numero%i == 0)
                suma += i;
        if(suma > 2 * numero)
            return true;
        else
            return false;
    }
}
```

```
}  
  
public static void main(String[] args) throws IOException{  
  
    boolean p, d;  
    System.out.println("Programa que detecta los siguientes tipos de  
numeros:");  
    System.out.println("Es Perfecto, Defectivo y/o Abundante");  
    System.out.println("Escriba el numero a evaluar: ");  
    entrada = buffer.readLine();  
    numero = Integer.parseInt(entrada);  
  
    p = perfecto();  
    if (p == true)  
        System.out.println("El numero: " + numero + " es perfecto");  
    else  
        System.out.println("El numero: " + numero + " NO es perfecto");  
  
    d = defectivo();  
    if (d)  
        System.out.println("El numero: " + numero + " es defectivo");  
    else  
        System.out.println("El numero: " + numero + " NO es defectivo");  
    if (abundante())  
        System.out.println("El numero: " + numero + " es abundante");  
    else  
        System.out.println("El numero: " + numero + " NO es abundante");  
}  
}
```

Una vez que ya tenemos definidos los métodos, lo último que falta es hacer que main llame a los métodos para que éstos se ejecuten. Si un método ha sido definido, pero no llamado, no se ejecutará.

La forma de llamar a los métodos depende del tipo de método, y también de si éste recibe o no argumentos. En este caso, se han definido los métodos tipo boolean, lo que quiere decir que regresarán un valor booleano. Por ello, el llamado a los métodos quedó de la forma en la que se muestra en el código. Se muestran dos formas de llamar a los métodos. En la llamada al método perfecto, en la línea:

```
p = perfecto();
```

La variable p es variable local de main tipo boolean, y en ella se guarda el resultado del método perfecto, el cual es regresado a main a través de la instrucción return. Esta línea es el llamado típico de los métodos, pues sigue la forma general.

Ahora, en el llamado al método abundante, se tiene la línea:

```
if (abundante())
```

En este caso, el método abundante no regresa su valor a una variable específica, sino que es evaluado directamente por la condicional del if. Esto se puede hacer debido a que el método abundante es tipo boolean y como recordamos, las condicionales evaluadas dentro del if son valores booleanos.