UNIVERSIDAD AUTÓNOMA DE SINALOA

Facultad de Informática Culiacán

Programación Modular en Java

Instructor:

MC. Gerardo Gálvez Gámez

gerardo.galvez@uas.edu.mx

Diciembre de 2016

Métodos y Parámetros

JAVA

Notas Generales

Es recomendable descomponer la lógica del programa en unidades funcionales, ya que facilita la reutilización del código.

1. Uso de métodos.
2. Uso de parámetros.
3. Uso de métodos sobrecargados.

Módulos en C#• FIUAS

Uso de Métodos

Uno de los principios básicos del diseño de aplicaciones es que deben estar divididas en unidades funcionales, ya que las secciones pequeñas de código son más fáciles de entender, diseñar, desarrollar y depurar.

La división de una aplicación en unidades funcionales permite también la reutilización de componentes funcionales en toda la aplicación.

- Definición de métodos.
- Llamadas a métodos.
- Uso de la instrucción return.
- Uso de variables locales.
- Devolución de valores.

Uso de Métodos

- **Un método de clase es aquel que puede ser invocado sin existir una instancia.**
- Una aplicación Java está estructurada en clases que contienen bloques de código con nombre llamados métodos.
- Un *método* es un miembro de una clase que lleva a cabo una acción o calcula un valor.
- Se utilizan para estructurar el código de un programa.

Módulos en C#• FIUAS

Definición de Métodos

Un método es una serie de instrucciones Java que han sido agrupadas bajo un nombre determinado.

Un método de clase se define agregando la palabra clave static antes del tipo en la signatura del método.

sintaxis:

```
//Comentario descriptivo de qué hace el método tipo función
    static  TipoValorDevuelto  NombreDelMétodo (tipo parámetro1, tipo parámetro2…) {
        Código del método
        return ResultadoQueDevuelveElMétodo;
    }
```

```
//Comentario descriptivo de qué hace el método tipo procedimiento
      static void nombreDelMétodo (tipo parámetro1, tipo parámetro2…) {
Código del método
}
```

Creación de Métodos

Cuando se crea un método hay que especificar lo siguiente:

- **Nombre**
  - Un método no puede tener el mismo nombre que una variable, una constante o cualquier otro elemento que no sea un código y haya sido declarado en la clase.
- **Lista de parámetros**
  - A continuación del nombre del método viene una lista de parámetros para el método. Esta lista aparece entre paréntesis que deben estar presentes aunque no hay ningún parámetro.
- **Cuerpo del método**
  - Después de los paréntesis viene el cuerpo del método, que debe estar entre llaves ({ y }) aunque no contenga más que una instrucción.

Módulos en C#• FIUAS

Llamadas a Métodos

Una vez definido un método, se puede:

- **Llamar a un método desde dentro de la misma clase**:
  - Se usa el nombre del método seguido de una lista de parámetros entre paréntesis.
- **Llamar a un método que está en una clase diferente**:
  - Hay que indicar al compilador cuál es la clase que contiene el método que se desea llamar.
  - El método llamado se debe declararse con la palabra clave **public.**
- **Usar llamadas anidadas**:
  - Unos métodos pueden hacer llamadas a otros, que a su vez pueden llamar a otros métodos, y así sucesivamente

Uso de la instrucción return

La instrucción **return** se puede emplear para hacer que un método se devuelva inmediatamente al llamador. Sin una instrucción **return**, lo normal es que la ejecución se devuelva al llamador cuando se alcance la última instrucción del método.

- Return inmediato
- Return con una instrucción condicional

Módulos en C#• FIUAS

Return inmediato

Por defecto, un método es devuelto a su llamador cuando se llega al final de la última instrucción del bloque de código.

La instrucción **return** se utiliza cuando se quiere que un método sea devuelto inmediatamente al llamador.

Ejemplo:

```
static void ExampleMethod( )
 {
    Console.WriteLine("Hola");
    return;
    Console.WriteLine("mundo");
 }
```

Return con una instrucción condicional

Es mucho más habitual, y mucho más útil, utilizar la instrucción **return** como parte de una instrucción condicional como **if** o **switch**.

Esto permite que un método sea devuelto al llamador si se cumple cierta condición.

Ejemplo:

```
static void ExampleMethod( )
{
   int Numero;
   Console.WriteLine("Hello");
   if (Numero < 10)
   {
      return;
   }
   Console.WriteLine("World");
}
```

Módulos en C#• FIUAS

Uso de variables locales

- **Variables locales:**
  - Se crean cuando comienza el método.
  - Son privadas para el método.
  - Se destruyen a la salida.
- **Variables compartidas:**
  - Para compartir se utilizan variables de clase.
  - Se definen como static.
- **Conflictos de ámbito:**
  - El compilador no avisa si hay conflictos entre nombres locales y de clase.

Variables Locales

```
static void MethodWithLocals( )
 {
    int x = 1; // Variable con valor inicial
   ulong y;
   string z;
    ...
}
```

Módulos en C#• FIUAS

Variables Compartidas

```
class CallCounter_Good
 {
   static int nCount;
   static void Init( )
   {
      nCount = 0;
    }
    static void CountCalls( )
    {
      ++ nCount;
      Console.Write("Método llamado " + nCuenta + " veces.");
    }
   static void main( )
    {
       Init( );
       CountCalls( );
       CountCalls( );
    }
}
```

Devolución de valores

- El método se debe declarar con un tipo que no sea void.
- Se añade una instrucción return con una expresión donde:
  - Se Fija el valor de retorno.
  - Se devuelve al llamador.
- Los métodos que no son void deben devolver un valor.

```
static int DosMasDos( ) {
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
static void main( )
{
    int x;
    x = DosMasDos( );
    System.out.println(x);
}
```


Los métodos que no son void

Deben devolver valores:

1. **Si se declara un método con un tipo distinto de void, es obligatorio añadir al menos una instrucción return.**

   El compilador intenta comprobar que cada uno de estos métodos devuelve siempre un valor al método de llamada.

2. **Si detecta que un método que no es void no incluye ninguna instrucción return, el compilador mostrará el siguiente mensaje de error: "No todas las rutas de código devuelven un valor."**

   Este mensaje también aparecerá si el compilador detecta que es posible ejecutar un método que no es void sin devolver un valor.

Declaración y llamadas a parámetros

- Declaración de parámetros:
  - Se ponen entre paréntesis después del nombre del método
  - Se definen el tipo y el nombre de cada parámetro
- Llamadas a métodos con parámetros:
  - Un valor para cada parámetro
- El mecanismo del paso de parámetro es por valor

```
static void  MethodWithParameters(int n, string y)
{ ... }

MethodWithParameters(2, "Hola, mundo");
```

Módulos en C#• FIUAS

Llamadas a métodos con parámetros

El código de llamada debe indicar los valores de los parámetros en la llamada al método.

Ejemplos:

```
MethodWithParameters(2, "Hola, mundo");

int p = 7;
String s = "Mensaje de prueba";
MethodWithParameters(p, s);
```

Paso por valor

Mecanismo predeterminado para el paso de parámetros:

- Se copia el valor del parámetro.
- Se puede cambiar la variable dentro del método.
- No afecta al valor fuera del método.
- El parámetro debe ser de un tipo igual o compatible.

```
static void SumaUno(int x)
{
      x++; // Incrementar x
}
static void Main( )
{
      int k = 6;
      SumaUno(k);
      Console.WriteLine(k); // Muestra el valor 6, no 7
}
```


Uso de métodos recursivos

- Un método el aquel que puede hacer una llamada a sí mismo.
  - Directamente.
  - Indirectamente.
- Útil para resolver ciertos problemas
  - Árboles.
  - Listas.

Ejemplo:

```
static int potencia (int x, int n)
{
  if (n==0)                          // Caso base
     return 1;
  else                               // Caso general
     return x * potencia(x,n-1);
}
```

Llamada recursiva

Uso de métodos sobrecargados

Los métodos no pueden tener el mismo nombre que otros elementos en una clase.

Sin embargo, dos o más métodos en una clase sí pueden compartir el mismo nombre. A esto se le da el nombre de sobrecarga.

- Declaración de métodos sobrecargados
- Signaturas de métodos
- Uso de métodos sobrecargados

Declaración de métodos sobrecargados

- Métodos que comparten un nombre en una clase
  - Se distinguen examinando la lista de parámetros
  - Java diferencia los métodos sobrecargados con base en el número y tipo de argumentos que tiene el método y no por el tipo que devuelve.

```
class OverloadingExample
{
    static int Suma(int a, int b)
    {
        return a + b;
    }
    static int Suma(int a, int b, int c)
    {
        return a + b + c;
    }
    static void Main( )
    {
        Console.WriteLine(Suma(1,2) + Suma(1,2,3));
    }
}
```

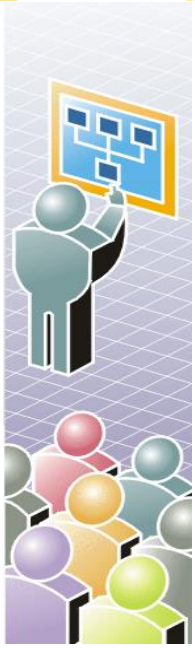

Uso de métodos sobrecargados

- Conviene usar métodos sobrecargados si:
  - Hay métodos similares que requieren parámetros diferentes.
  - Se quiere añadir funcionalidad al código existente.
- No hay que abusar, ya que:
  - Son difíciles de depurar.
  - Son difíciles de mantener.

Codificación de Algoritmos con propuesta de solución modular.

Actividad #1

- Descripción:
  - Codificar el siguiente pseudocódigo que calcula e imprime:
    - El área de un triangulo (Base * Altura /2), y



Propuesta Algoritmo Modular

```
//Objetivo: Calcular el área de un Triangulo
//Programador: MC. Gálvez
//Fecha: ___de Noviembre de 2015
INICIO
   //Definición de Constantes  y  Variables Globales

   PRINCIPAL ()
   INICIO
         //Definición de Constantes  y  Variables Locales
         REAL Area, Base, Altura
         LecturaDatos(Altura,Base)
         Area= CalcularArea(Altura, Base)
         ImprimirArea(Area)
   FIN

   SINVALOR  LecturaDatosTriangulo(REAL Altura, REAL Base)
   INICIO
        IMPRIMIR "Proporciona el valor de la Base: "
        LEER Base
        IMPRIMIR "Proporciona el valor de la Altura: "
        LEER Altura
        IMPRIMIR "Fin de lectura…"
   FIN
```


Continuación …

```
   REAL  CalcularArea(REAL Altura, REAL Base)
   INICIO
        //Definición de Constantes  y  Variables Locales
        REAL Area
        ///Proceso, calcular el área
        Area=Base * Altura / 2
        //Regresar el resultado obtenido
         REGRESAR Area
   FIN

   SINVALOR  ImprimirArea(REAL Area)
   INICIO
         IMPRIMIR "El Área es: ", Area
   FIN

FIN
```

Preguntas?