UNIVERSIDAD DE MATANZAS

# MÉTODOS Y FUNCIONES

Octubre / 2019

INTRODUCCIÓN A LA PROGRAMACIÓN
INGENIERÍA EN INFORMÁTICA

# Objetivos

Caracterizar los métodos y funciones como parte de la construcción de un programa computacional.

# Objetivos

Utilizar los métodos comunes de Math disponibles en la API de Java.

# Objetivos

Comprender los mecanismos para pasar información entre métodos.

# Objetivos

Comprender cómo se soporta el mecanismo de llamada/retorno de los métodos mediante la pila de llamadas a métodos y los registros de activación.

# Sumario

- Módulos de programas en Java
- Métodos
  - Declaración
  - Invocación
- Métodos de la clase *Math*

# Bibliografía


- *Como programar en Java.*
- *Aprenda Java como si estuviera en primero.*

# Introducción

La mayoría de los programas de cómputo que resuelven los problemas reales son mucho más extensos que los programas que hemos desarrollado hasta el momento.

# Introducción

La experiencia ha demostrado que la mejor manera de desarrollar y mantener un programa extenso es construirlo a partir de pequeñas piezas sencillas, o **módulos**.

A esta técnica se le llama **divide y vencerás**.

# Módulos

Existen tres tipos de módulos en Java:

- Métodos
- Clases
- Paquetes

# Métodos

Los métodos (también conocidos como funciones o procedimientos en otros lenguajes) permiten al programador dividir un programa en módulos, por medio de la separación de sus tareas en unidades autónomas.

# Métodos

Una razón para dividir un programa en módulos mediante los métodos es la metodología **divide y vencerás**, que hace que el desarrollo de programas sea más fácil de administrar, ya que se pueden construir programas a partir de piezas pequeñas y simples.

# Métodos

Otra razón es la **reutilización de software** (usar los métodos existentes como bloques de construcción para crear nuevos programas). A menudo se pueden crear programas a partir de métodos estandarizados, en vez de tener que crear código personalizado.

# Métodos

Un método se invoca mediante una llamada, y cuando el método que se llamó completa su tarea, devuelve un resultado, o simplemente el control al método que lo llamó.

# Métodos

Aunque la mayoría de los métodos se ejecutan en respuesta a las llamadas a métodos en objetos específicos, éste no es siempre el caso. Algunas veces un método realiza una tarea que no depende del contenido de ningún objeto. ...

# Métodos

... Dicho método se aplica a la clase en la que está declarado como un todo, y se conoce como **método static** o **método de clase**.

Es común que las clases contengan **método static** convenientes para realizar tareas comunes.

# Métodos

## Sintaxis de declaración

```
<visibilidad> static <TDR> <nombre> (<argumentos>){

        <Conjuntos de Instrucciones>

}
```

Analicemos cada una de las partes de forma detallada.

# Métodos

## Sintaxis de declaración

- <**visibildad**>: Hace referencia a la visibilidad que tenga el método con respecto al resto de los módulos que participan en el programa.

# Métodos

## Sintaxis de declaración

- <**visibildad**>: Los posibles valores son las palabras reservadas del lenguaje **private**, **protected** o **public**.

# Métodos

## Sintaxis de declaración

- **static**: Da la posibilidad que el método sea invoca o llamado sin tener que crear un objeto o instancia de la clase donde fue declarado previamente.

# Métodos

## Sintaxis de declaración

- **static**: Puede ser omitido de la declaración pero entonces para usar o llamar el método sería a través de una instancia u objeto de la clase.

# Métodos

## Sintaxis de declaración

- <**TDR**>: **Tipo de dato de retorno** hace referencia al tipo de dato que va devolver el método una vez terminadas sus operaciones u instrucciones.

# Métodos

## Sintaxis de declaración

- <**TDR**>: Los métodos para retornar o devolver algún valor, dato o variable hace uso de la instrucción **return**.

# Métodos

## Sintaxis de declaración

- <**TDR**>: El tipo de dato debe coincidir con el tipo de dato del valor, dato o variable que se va retornar. En caso de que no se retorne nada el valor debe ser **void**.

# Métodos

## Sintaxis de declaración

- <**nombre**>: Va ser el identificador del método y para su elaboración se cumple con las mismas reglas que para la conformación de los identificadores de las variables.

# Métodos

## Sintaxis de declaración

- <**argumentos**>: Este término puede ser nulo en caso que para realizar sus instrucciones el método no necesite información extra.

# Métodos

## Sintaxis de declaración

- <**argumentos**>: Pero a menudo los métodos requieren más de un dato de información para realizar sus instrucciones.

# Métodos

## Sintaxis de declaración

- <**argumentos**>: Por cada dato que se necesite pasar al método se debe especificar el tipo de dato y el identificador. Si hay mas de un dato separalos por coma.

# Métodos

## Ejemplos. Con argumentos y retorno

```
// devuelve el maximo de sus tres parametros int
public static int maximo( int x, int y, int z ){
  int valorMaximo = x;
  if(y>valorMaximo){
    valorMaximo=y;
  }
  if(z>valorMaximo){
    valorMaximo=z;
  }
  return valorMaximo;
}
```

# Métodos

## Ejemplos. Sin argumentos y ni retorno

```
public static void determinarMaximo(){
    Scanner entrada = new Scanner( System.in );
    System.out.print(
          "Escriba tres valores enteros, separados
             por espacios: ");
    int nu1 = entrada.nextInt();
    int nu2 = entrada.nextInt();
    int nu3 = entrada.nextInt();
    /*determina el valor maximo llamando al metodo
    anterior asumiendo que ambos metodos estan
    declarado en la misma clase*/
    int resultado=maximo(nu1,nu2,nu3);
    System.out.println( "El maximo es: "+resultado);
}
```

# Métodos

## Sintaxis de declaración

Lo anterior no es una *camisa de fuerza* porque pueden existir:

- Métodos con retorno y con argumentos.
- Métodos sin retorno y con argumentos.
- Métodos con retorno y sin argumentos.
- Métodos sin retorno y sin argumentos.

# Métodos

## Sintaxis de invocación

Puede llamar a cualquier método *static* especificando el nombre de la clase en la que está declarado el método, seguido de un punto (.) y del nombre del método, como sigue:

```
<NombreClase>.<nombreMetodo>( <argumentos> )
```

Analicemos cada una de las partes de forma detallada.

# Métodos

## Sintaxis de invocación

- <**NombreClase**>: Se va nombrar el nombre de la clase donde está el método que queremos invocar.

# Métodos

## Sintaxis de invocación

- <**NombreClase**>: En caso de queramos invocar un método que esta en la misma clase (*class*) no es necesario ponerla ni el punto (.) pero no esta mal hacerlo de hecho el lenguaje lo reconoce.

# Métodos

## Sintaxis de invocación

- <**argumentos**>: De igual que sucede en la declaración esto puede ser nulo en caso que el método no necesite datos adicionales.

# Métodos

## Sintaxis de invocación

- <**argumentos**>: En caso que necesite datos adicionales los mismo se ponen separados por comas pero sin especificar el tipo de dato como si se hace en la declaración.

# Métodos

## Sintaxis de invocación

- <**argumentos**>: Si es importante que los datos que se les pase en la invocación coincidan en ese mismo orden con los tipos de datos de los argumentos de la declaración.

# Métodos

## Ejemplos

Aquí utilizaremos un método de la clase Math para presentar el concepto de invocación.

```
Math.sqrt( 900.0 );
/*Pero tambien pudo ser*/
double a=25;
Math.sqrt(a);
```

# Métodos

## Ejemplos

La expresiónes anteriores se evalúan como 30.0 para 900 y 5.0 para *a*.

El método *sqrt* (halla la raíz cuadrada) recibe un argumento de tipo double y devuelve un resultado del mismo tipo.

# Métodos

## Ejemplos

Para imprimir el valor de la llamada anterior al método en una ventana de comandos, podríamos escribir la siguiente instrucción:

```
System.out.println(Math.sqrt(900.0));
```

En esta instrucción, el valor que devuelve *sqrt* se convierte en el argumento para el método *println* .

# Clase Math

La clase **Math** cuenta con una colección de métodos y constantes que nos permiten realizar cálculos matemáticos comunes. La siguiente tabla sintetiza de los métodos y constantes de la clase **Math**.

# Clase Math

| **Método** | **Descripción** |
|---|---|
| abs(x) | valor absoluto de x |
| ceil(x) | redondea x al entero más pequeño que no sea menor de x |
| cos(x) | coseno trigonométrico de x (x está en radianes) |
| exp(x) | método exponencial $e^x$ |

# Clase Math

| **Método** | **Descripción** |
|---|---|
| floor(x) | redondea x al entero más grande que no sea mayor de |
| log(x) | logaritmo natural de x (base e) |
| max(x,y) | el valor más grande de x y y |
| min(x,y) | el valor más pequeño de x y y |

# Clase Math

| **Método** | **Descripción** |
|---|---|
| pow(x,y) | x elevado a la potencia y $(x^y)$ |
| sin(x) | seno trigonométrico de x (x está en radianes) |
| sqrt(x) | raíz cuadrada de x |
| tan(x) | tangente trigonométrica de x (x está en radianes) |

# Conclusiones

- Cada método debe limitarse de manera que realice una sola tarea bien definida, y su nombre debe expresar esa tarea con efectividad.

# Conclusiones

- Si no puede elegir un nombre conciso que exprese la tarea de un método, tal vez esté tratando de realizar diversas tareas en un mismo método. Por lo general, es mejor dividirlo en varias declaraciones de métodos más pequeños.

# Conclusiones

- Una mala práctica o una evidencia de que no existe una correcta implementación del método es que el mismo tenga como **TDR void** y haga uso del sentencia **return** en su implementación.

# Conclusiones

- No pueden existir dos métodos iguales y para que esto no ocurra tienen que diferir en al menos uno de los siguientes aspectos.
  - Paquete o clase que lo contiene.
  - Nombre
  - Tipo de dato de retorno
  - Cantidad de argumentos
  - Orden de los tipos de datos de los argumentos

UNIVERSIDAD
DE MATANZAS

cosechando el saber

FIN