

# Desarrollo Avanzado de Aplicaciones I

## Introducción a Java III



i

inicio

d

desarrollo

a

aplicación

t

término



idat

# Inicio

Logro de aprendizaje – Introducción



# Logro de Aprendizaje

"Al finalizar la sesión, el participante podrá utilizar métodos de clases con sus modificadores, parámetros y argumentos, datos de retorno, variables globales y locales, y clases envoltorio (wrappers)."

# Introducción



- Revisión Rápida de Temas de Sesión Anterior
- Revisión de Ejercicios de Sesión Anterior
- Inquietudes y/o Preguntas
- Agenda de Sesión
  - Métodos de una clase
  - Modificadores de acceso y comportamiento
  - Variables globales y locales
  - Parámetros y argumentos
  - Métodos sobrecargados
  - Métodos de clases propias de java
  - Clases envoltorio - wrappers



# Desarrollo

Desarrollo del Contenido de la Sesión



# Métodos de una Clase



```
[modificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws listaExcepciones] {  
    // sentencias  
    [return valor;]  
}
```

- ✓ *modificadores* (opcional): determinan el tipo de acceso al método y el tipo de método.
- ✓ *tipoDevuelto*: indica el tipo de dato que devuelve el método. El dato se devuelve mediante la instrucción `return`. Si el método no devuelve ningún valor este tipo será `void`.
- ✓ *nombreMetodo*: es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.
- ✓ *Lista de parámetros* (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros, separados por comas.
- ✓ *throws listaExcepciones* (opcional): indica las excepciones que puede generar y manipular el método.
- ✓ *return*: se utiliza para devolver un valor y finalizar el método.



# Consideraciones



- ✓ Los paréntesis a continuación del nombre del método son obligatorios aunque estén vacíos.
- ✓ La ejecución de un método termina cuando se llega a su llave final “}” o cuando se ejecuta la instrucción “return”, la cual puede aparecer en cualquier lugar dentro del método.





# Métodos sin Valor de Retorno



- ✓ Cuando el método no va a devolver ningún valor se debe indicar "void" en "tipoDevuelto".
- ✓ Si el método no devuelve ningún valor la instrucción "return" es opcional.
- ✓ El método "main" siempre se declara con "void".

```
public class Saludar {  
  
    public static void main(String[] args) {  
        mostrarSaludo();  
    }  
  
    static void mostrarSaludo() {  
        System.out.println("Hola a Todos");  
    }  
}
```



# Métodos con Valor de Retorno



- ✓ Si el método va a devolver un valor de retorno se debe indicar el tipo de dato en "tipoDevuelto".
- ✓ Para devolver un valor se usa "return" seguida de una expresión. La expresión puede ser compleja o simple, pero el valor que retorne debe ser igual a lo declarado en "tipoDevuelto".

```
import java.util.Scanner;
public class Saludar {
    public static void main(String[] args) {
        String nombre = ingresarNombre();
        System.out.println("Hola " + nombre);
    }
    static String ingresarNombre() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingresa tu nombre: ");
        return scanner.next();
    }
}
```

← tipoDevuelto

← Valor de retorno String



# Modificadores de Acceso



- ✓ Los modificadores de acceso nos permiten establecer el alcance y la accesibilidad o visibilidad de un método. Aplica también para clases y variables.

Modificador de acceso	Palabra reservada	Visibilidad
Público	public	Visible para todas las clases.
Protegido	protected	Visible para las clases de un mismo paquete y para las clases derivadas en otro paquete.
Sin modificador de acceso (predeterminado)		Visible para las clases de un mismo paquete.
Privado	private	Visible solo en la misma clase. No es accesible fuera de la clase.



# Ejemplo de Modificadores de Acceso



```
package desarrollo.de.aplicaciones;
public class EjemploModificadores {
    public static void main(String[] args) {
        // Métodos de la misma clase
        saludarPrivado();
        saludarPublico();
        saludarProtegido();
        saludarPredeterminado();
        // Métodos de otra clase
        claseSaludar cS = new claseSaludar();
        cS.saludarPublico();
        cS.saludarProtegido();
        cS.saludarPredeterminado();
        cS.otroSaludo(); }
    private static void saludarPrivado() {
        System.out.println(⌘: "Hola Privado (misma clase)"); }
    public static void saludarPublico() {
        System.out.println(⌘: "Hola Público (misma clase)"); }
    protected static void saludarProtegido() {
        System.out.println(⌘: "Hola Protegido (misma clase)"); }
    static void saludarPredeterminado() {
        System.out.println(⌘: "Hola Predeterminado (misma clase)"); } }
```

```
class claseSaludar {
    public void saludarPublico() {
        System.out.println(⌘: "Hola Público (otra clase)");
    }
    private void saludarPrivado() {
        System.out.println(⌘: "Hola Privado (otra clase)");
    }
    protected void saludarProtegido() {
        System.out.println(⌘: "Hola Protegido (otra clase)");
    }
    void saludarPredeterminado() {
        System.out.println(⌘: "Hola Predeterminado (otra clase)");
    }
    public void otroSaludo() {
        saludarPrivado();
    }
}
```



# Modificadores de Comportamiento



- ✓ Estos tipos de modificadores se utilizan para controlar el comportamiento de los métodos.

Modificador	Palabra reservada	Comportamiento
Estático	<code>static</code>	El método pertenece a la clase y no a las instancias de la clase. Se puede utilizar de forma directa sin instanciar un objeto de la clase y solo pueden invocar directamente a otros métodos y variables estáticos.
Final	<code>final</code>	El método no se puede redefinir en subclases. Aplica también para clases y variables. Ejemplo los métodos de la Clase String.
Abstracto	<code>abstract</code>	El método solo se declara en la clase, sin código. Su código se implementa en la subclase. Para implementar un método abstracto la clase debe ser abstracta.
Sincronizado	<code>synchronized</code>	El método no se puede ejecutar de forma simultánea.



# Ejemplo de Modificadores de Comportamiento



```
public class OtrosModificadores {  
    public static void main(String[] args) {  
        metodoEstatico();  
        ClasePadre CP = new ClasePadre();  
        CP.metodoFinal();  
    }  
  
    static void metodoEstatico() {  
        System.out.println("Método Estático");  
    }  
}  
  
class ClasePadre {  
    final void metodoFinal() {  
        System.out.println("Método Final");  
    }  
}
```

```
class ClaseHija extends ClasePadre {  
    @Override  
    void metodoFinal() {  
        System.out.println("Método Final");  
    }  
}
```

metodoFinal() in ClaseHija cannot override metodoFinal() in ClasePadre  
overridden method is final  
----  
(Alt-Enter shows hints)



# Variables Globales y Locales



- ✓ Las variables locales son aquellas que se declaran dentro del cuerpo de un método o de un bloque de código (Sentencias condicionales o repetitivas).
- ✓ Las variables globales son aquellas que se declaran fuera de un método pero dentro de una clase, es decir, los atributos de una clase. Al declararlas se le asigna siempre un valor por defecto:

Tipo de Dato	Valor por Defecto
byte, short, int, long	0
float, double	0.0
boolean	false
char	'\u0000' (Caracter nulo)
String (o cualquier objeto)	null



# Ejemplo de Variables Globales y Locales



```
public class Triangulo {  
    int base;  
    int altura;  
  
    public void calcularArea() {  
        int area = base * altura / 2;  
        System.out.println("El área del triángulo es: " + area);  
    }  
}
```

← Variables Globales

↑ Variable Local

¿Y los valores de las variables base y altura para calcular su área?

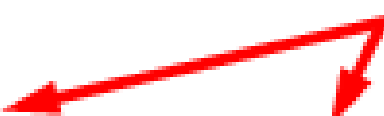




- ✓ Un parámetro es una variable que se usa en la declaración de un método.

```
public class Triangulo {  
    public void calcularArea(int base, int altura) {  
        int area = base * altura / 2;  
        System.out.println("El área del triangulo es: " + area);  
    }  
}
```

Parámetros



# Argumentos



- ✓ Un argumento es el valor que se le da a los parámetros cuando es llamado un método.

```
public class Triangulo {  
  
    public static void main(String[] args) {  
        int baseTri = 5, alturaTri = 10;  
        Triangulo tri = new Triangulo();  
        tri.calcularArea(base: 10, altura: 20);  
        tri.calcularArea(base: baseTri, altura: alturaTri);  
    }  
  
    public void calcularArea(int base, int altura) {  
        int area = base * altura / 2;  
        System.out.println("El área del triangulo es: " + area);  
    }  
}
```

Argumentos



# Métodos Sobrecargados

- ✓ Los métodos sobrecargados son métodos de una misma clase que tienen el mismo nombre pero que tienen diferentes parámetros.

```
public class Triangulo {  
    int base = 10, altura = 20;  
    public static void main(String[] args) {  
        Triangulo tri = new Triangulo();  
        tri.calcularArea();  
        tri.calcularArea(base: tri.base, altura: tri.altura);  
    }  
    public void calcularArea() { Método sin parámetros  
        int area = base * altura / 2;  
        System.out.println("El área del triangulo es: " + area);  
    }  
    public void calcularArea(int base, int altura) { Método con parámetros  
        int area = base * altura / 2;  
        System.out.println("El área del triangulo es: " + area);  
    }  
}
```



# Métodos de Clases Propias de Java



- ✓ Java proporciona un conjunto de Clases con métodos útiles para realizar diferentes funciones:
  - `Math.random()`
  - `Math.round(floatdouble)`
  - `String.indexOf(String)`
  - `String.contains(String)`
  - `String.replace(Stringtarget,Stringreplacement)`
  - `System.out.println(String)`
  - `System.out.print(String)`



# Ejemplos de Métodos de Clases Propias de Java

```
public class ClasesJava {  
    public static void main(String[] args){  
        int dado;  
        String cadena;  
        // Metodos de la Clase Math  
        dado = (int) (Math.random()*6+1);  
        System.out.println("Valor del dado: "+ dado);  
        System.out.println("Redondeo; " + Math.round( a: 15.89));  
        System.out.println("Redondeo; " + Math.round( a: 15.45));  
        // Metodos de la Clase String  
        cadena = "Cuando las aves vuelan es porque el cielo está despejado";  
        System.out.println("'aves' se encuentra en la posición: " + cadena.indexOf( str: "aves"));  
        System.out.println("'cielo' se encuentra en la oración: " + cadena.contains( s: "cielo"));  
        System.out.println("" + cadena.replace( target: "las aves", replacement: "los pajaros"));  
        System.out.println( x: cadena);  
    }  
}
```

# Clases Envoltorio - Wrappers



- ✓ Los Wrappers (envoltorios) son clases diseñadas para ser un complemento de los tipos primitivos siendo los únicos elementos de Java que no son objetos.
- ✓ Existe una clase Wrapper para cada uno de los tipos primitivos.
- ✓ Los nombres de las clases Wrappers empiezan con mayúscula.
- ✓ Cada clase Wrapper incluye métodos para conversión de datos y otras funciones.

Tipo de Dato Primitivo	Clase Wrapper	Tipo de Dato Primitivo	Clase Wrapper
byte	Byte	float	Float
short	Short	double	Double
int	Integer	boolean	Boolean
long	Long	char	Character

Java™

# Ejemplo de Clases Envoltorio - Wrappers

```
public class Envoltorios {  
    public static void main(String[] args) {  
        int i = 5; double d = 8.5; char c = 'A';  
        String cadena;  
        Integer x = 18; Double y = 19.7; Character z = 'b';  
        System.out.println("Datos Primitivos, int: " + i + " double: " + d + " char: " + c);  
        System.out.println("Envoltorios, Integer: " + x + " Double: " + y + " Character: " + z);  
        System.out.println("int concatenado con double: " + Integer.toString(i) + " - " + Double.toString(d));  
        cadena = Integer.toString(i + x); // Suma de primitivo + envoltorio  
        i = Integer.parseInt(,cadena); // Conversión de String a primitivo int  
        System.out.println("Suma de Primitivo + envoltorio int: " + i);  
        cadena = Double.toString(d + y); // Suma de primitivo + envoltorio  
        y = Double.valueOf(,cadena); // Conversión de String a primitivo double  
        System.out.println("Suma de Primitivo + envoltorio double: " + y);  
        System.out.println("¿Es dígito? " + Character.isDigit(,c));  
        System.out.println("¿Es espacio en blanco? " + Character.isWhitespace(,z));  
        System.out.println("¿Es mayúscula? " + Character.isUpperCase(,c));  
        System.out.println("¿Es minúscula? " + Character.isLowerCase(,z)); } }
```



# Aplicación

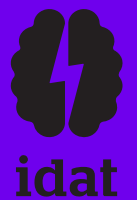
Revisar ejemplos y realizar ejercicios prácticos





# Término

Indicaciones generales y/o Resumen de Sesión



# Resumen de Sesión



- Métodos de una clase
- Métodos con/sin valor de retorno
- Modificadores de acceso y de comportamiento
- Variables globales y locales
- Parámetros y argumentos
- Métodos sobrecargados
- Métodos de clases propias de java
- Clases envoltorio – wrappers
- Ejemplos y ejercicios



GRACIAS