



# Organización

- 1 Organización
- 2 Tipos
- 3 Orientación a objetos en Java
- 4 Bibliografía

# Temas

- 1 Organización
  - Componentes de un programa en Java
  - Organización por convención

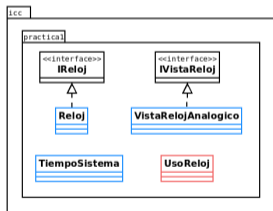
# Java

- Java es un lenguaje de programación orientado a objetos, aunque aún contiene elementos fuera de ese paradigma.
- Fue diseñado para implementar grandes sistemas de software que funcionaran a través de la red.

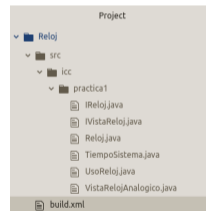
# Paquetes y clases

La organización de un proyecto en Java se refleja en dos niveles:

En su arquitectura lógica, en forma de *paquetes* y *clases*.



En la organización del código, acomodado en **directorios** y **archivos**.



- A partir de la versión 9, también se incluyen *módulos*<sup>[1]</sup>.

<sup>[1]</sup>Información en: <http://openjdk.java.net/projects/jigsaw/quick-start>

# Temas

- 1 Organización
  - Componentes de un programa en Java
  - Organización por convención

# Clases de uso

- Para poder reutilizar código entre diferentes proyectos, las funcionalidades provistas por el sistema se implementan dentro de *clases*, plantillas a partir de las cuales se general objetos con datos y sus comportamientos.
- Por otro lado, un programa concreto comenzará su ejecución a partir de una *clase de uso*, que manda llamar, conforme se requiera, las funcionalidades provistas por objetos de clases independientes.
- A semejanza del lenguaje estructurado C, las clases de uso se caracterizan por poseer un método `main` a partir del cual **inicia la ejecución del programa**.

# Ejemplo: clase de uso

```
1 package mate.números;
2
3 /** Programa para sumar dos números complejos. */
4 public class UsoComplejos {
5
6     /** Punto de entrada. */
7     public static void main(String[] args) {
8         Complejo c1 = new Complejo(4, 5);
9         Complejo c2 = new Complejo(-4, 3);
10        Complejo r = c1.suma(c2);
11        System.out.println("Suma_□" + c1 + "+" + c2 + "=" + r);
12    }
13
14 }
```

# Ejemplo

- Las clases de uso también permiten la interacción con el usuario.
- Su nombre no inicia necesariamente con la palabra `Uso`, pero puede servir como indicación del rol de la clase.

```
1 package mate.números;
2
3 /** Programa para sumar dos números enteros. */
4 public class Suma {
5     /** @param args argumentos con que fue invocado el programa. */
6     public static void main(String[] args) {
7         int n1 = Integer.parseInt(args[0]);
8         int n2 = Integer.parseInt(args[1]);
9         int r = n1 + n2;
10        System.out.println(args[0] + " + " + args[1] + " = " + r);
11    }
12 }
```

Esto se ejecuta así:

---

```
1 $ java mate.números.Suma 4 3
2 4 + 3 = 7
3 $ java mate.números.Suma 353 -23
4 353 + -23 = 330
```

---

# Tipos

- 1 Organización
- 2 Tipos**
- 3 Orientación a objetos en Java
- 4 Bibliografía

# Temas

## 2 Tipos

- Tipos de datos
- Variables
- Tipos primitivos
- Tipos derivados

# Tipo de datos

- Un *tipo de datos* especifica la interpretación semántica de la información introducida en la computadora, es decir, su significado y, por ende, cómo operar con ella.

# Tipos abstracto de datos (TAD)

- Un *tipo abstracto de datos* es una especificación formal algebraica de:
  - Un conjunto de datos
  - las operaciones que pueden realizarse con ellos con:
    - 1 Las *precondiciones* que se deben cumplir para poder realizar la operación.
    - 2 Las *postcondiciones* o relaciones que se satisfarán tras haber ejecutado la operación.
- No indica cómo serán representados estos datos en ningún sistema en particular, ni los detalles de cómo se llevarán a cabo las operaciones. Por ello son **abstractos**.

# Estructuras de datos

*“Las estructuras de datos son las formas de representación interna de datos de la computadora, mediante las que se representa cualquier situación en la computadora, es decir, son los tipos de datos que maneja la máquina.*

*Por ejemplo, podemos representar a un trabajador mediante los datos nombre del empleado, número de horas trabajadas, cuota por hora, etcétera.”*

*López Román 2011*

# Temas

## 2 Tipos

- Tipos de datos
- **Variables**
- Tipos primitivos
- Tipos derivados

# Variables

- Siendo un lenguaje **imperativo**, Java hace uso de **variables** y **asignación de valores**.
- Las *variables* son localidades de memoria donde se pueden almacenar datos.
- Estos datos también pueden ser modificados a lo largo de la ejecución del programa.

# Variables

- Es posible nombrar y referirse a una variable por medio de una cadena de caracteres.

```
1 num = 2.19;
```

- Algunos lenguajes de programación, como Java, son muy estrictos con respecto al tipo de datos que pueden ser almacenados en una variable, pues esto se utilizará para verificar la semántica de la operaciones sobre ellos.

```
1 float num = 2.19f;
```

Otros lenguajes sólo requieren el nombre.

# Declaración y definición

Para lenguajes con manejo explícito de tipos, podemos distinguir dos pasos para la creación de variables:

**Declaración** Es el momento en el cual se aparta la localidad de memoria, se indica el tipo de dato que será almacenado ahí y se le asigna un nombre.

## Sintaxis (Variable)

*<declaración de variable>* ::= <tipo> <identificador>, ..., <identificador>;

*<tipo>* ::= <tipo primitivo> | <identificador de clase>

*<identificador>* ::= (<letra> | ) (<letra> | <dígito> | )<sup>\*</sup>

```
1 short edad;  
2 String nombre, dirección;  
3 Color _colorOjos;  
4 double _num;
```

**Definición** Es el momento en el cual se asigna valor a la variable por primera vez.

## Sintaxis (Asignación)

*<asignación>* ::= <identificador> = <valor>;

```
1 edad = 10;  
2 nombre = "Mónica";
```

**Declaración y definición** Es posible abreviar realizando ambas acciones en un sólo enunciado.

```
1 short edad = 10;  
2 String nombre = "Mónica";  
3 String nombre = "Mónica", dirección = "Alameda_25";
```

# Temas

## 2 Tipos

- Tipos de datos
- Variables
- **Tipos primitivos**
- Tipos derivados

# Tipos primitivos

- Una de las características *no orientadas a objetos* de Java es la presencia de tipos primitivos.
- Los tipos primitivos se encuentran cercanamente relacionados con la representación binaria de la máquina, lo cual permite operar con ellos eficientemente.
- Usualmente sus operaciones tienen asociados operadores fáciles de identificar.

Por ejemplo números:  $5 + 5$ ,  $6.78 - 8.9$ ,  $10000/5$ ,  $2^3$ , etc.

# Tipos primitivos en Java

Tabla: Tipos primitivos

Identificador	Capacidad	Representación en memoria
boolean	2 bytes	<b>true</b> o <b>false</b>
char	2 bytes	16 bits, Unicode 2.0
byte	1 byte	8 bits con signo en complemento a 2
short	2 bytes	16 bits con signo en complemento a 2
int	4 bytes	32 bits con signo en complemento a 2
long	8 bytes	64 bits con signo en complemento a 2
float	4 bytes	32 bits de acuerdo al estándar IEEE 754-1985
double	8 bytes	64 bits de acuerdo al estándar IEEE 754-1985

# Operadores y precedencia I

*Operadores*. La primer columna indica la precedencia, entre mayor es el número, primero se realiza esa operación.

	Operandos	Operador	Tipo	Asociatividad
15	postfijo unario	() [] .	Paréntesis Índice de arreglo Selector de miembro	izq a der
14	postfijo unario	++ --	Postincremento Postdecremento	der a izq

# Operadores y precedencia II

13	prefijo unario	<div>++</div> <div>--</div> <div>+</div> <div>-</div> <div>!</div> <div>~</div> <div>(tipo)</div>	<div>Preincremento</div> <div>Predecremento</div> <div>Más</div> <div>Menos</div> <div>Negación lógica</div> <div>Complemento en bits</div> <div>Conversión de tipo</div>	der a izqu
12	binario infijo	<div>*</div> <div>/</div> <div>%</div>	<div>Multiplicación</div> <div>División</div> <div>Módulo</div>	izq a der
11	binario infijo	<div>+</div> <div>-</div>	<div>Suma</div> <div>Resta</div>	izq a der

# Operadores y precedencia III

10	binario infijo	<<  >>  >>>	corrimiento de bits a la izquierda  corrimiento de bits a la derecha con extensión de signo  corrimiento de bits a la derecha con llenando con ceros	izq a der
9	binario infijo	<  <=  >  >=  instanceof	relacional menor que  relacional menor o igual  relacional mayor que  relacional mayor o igual  comparación de tipos (sólo objetos)	izq a der

# Operadores y precedencia IV

8	binario infijo	== !=	relacional igual a relacional distinto de	izq a der
7	binario infijo	&	AND de bits	izq a der
6	binario infijo	^	OR exclusivo de bits	izq a der
5	binario infijo		OR inclusivo de bits	izq a der
4	binario infijo	&&	AND lógico	izq a der
3	binario infijo		OR lógico	izq a der
2	ternario infijo	? :	Condicional ternario	izq a der

# Operadores y precedencia V

1	binario infijo	=	Asignación	der a izq
		+=	Autosuma y asignación	
		-=	Autorresta y asignación	
		*=	Automultiplicación y asignación	
		/=	Autodivisión y asignación	
		%=	Automódulo y asignación	
		&=		
		^=		
		=		
		<<=		
		>>=		

# Operadores y precedencia VI

		>>>=		
--	--	------	--	--

Fuente: [http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op\\_precedence.html](http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html) y Valdés y Gurovich 2008.

# Temas

## 2 Tipos

- Tipos de datos
- Variables
- Tipos primitivos
- Tipos derivados

# Clases e Interfaces

- Diferentes lenguajes proveen al programador de mecanismos para definir sus propios tipos.
- Las interfaces de programación para aplicaciones (APIs) ya incluyen varios tipos definidos de esta manera.
- En Java hay dos formas de definir tipos:
  - 1 Clases
  - 2 Interfaces

En este caso, las operaciones correspondientes toman la forma de *métodos*, en lugar de los operadores, que usan los tipos primitivos.

```
1 String hola = "Hola_a_todos";  
2 String todos = hola.substring(7);
```

# Orientación a objetos en Java

- 1 Organización
- 2 Tipos
- 3 Orientación a objetos en Java**
- 4 Bibliografía

# Temas

## 3 Orientación a objetos en Java

- Clases
- Atributos
- Métodos

# Declaración de clases

## Sintaxis (Firma)

```
<declaración de clase> ::= <acceso> class <identificador> {  
    <declaraciones>  
    <método main>  
}  
<acceso> ::= public | private | protected | ∅ (i.e. package)  
<identificador> ::= (<letra>|_)(<letra>|<dígito>|_)*
```

Por convención, los nombres de clases inician con mayúscula.

```
1  public class Persona{  
2  
3  }
```

# Temas

## 3 Orientación a objetos en Java

- Clases
- Atributos
- Métodos

# Atributos

## Sintaxis (Atributos)

```
<declaración de atributo> ::= <acceso> <modificador> <tipo>  
    <identificador>, ..., <identificador>;  
<modificador> ::= final | static | ∅  
<tipo> ::= <tipo primitivo> | <identificador de clase>
```

```
1  public class Persona{  
2      public static final String SLOGAN = ";Que_viva_la_gente!";  
3      private String nombre;  
4      private short  edad;  
5      private Color  colorOjos;  
6      private Color  colorCabello;  
7      private Sexo   sexo;  
8  }
```

# Temas

## 3 Orientación a objetos en Java

- Clases
- Atributos
- Métodos

# Métodos

- Funcionalmente los métodos de una clase pueden ser clasificados en:
  - Constructores:** Crean a los objetos e inicializan sus atributos.
  - De acceso y modificación:** Devuelven el valor de un atributo o permiten asignarle un valor, según las restricciones impuestas por la clase, respectivamente.
  - De actualización y manipulación:** Cambian el estado del objeto<sup>[2]</sup>.
  - De implementación:** Ofrecen los servicios especiales para los que fue diseñado el objeto.
  - Auxiliares:** Permiten al objeto organizar mejor sus tareas y no proveen servicios al exterior, es decir, se declaran como privados (o protegidos).
- Método **main**. Permite ejecutar a la clase como un programa, es estático, por lo que no aplica a ningún objeto en particular.

---

<sup>[2]</sup> Los valores de sus atributos

# Constructor

- El constructor permite crear a los objetos que las clases describen.
- Java asigna valores por defecto a los atributos del objeto.
  - Para los atributos de tipo numérico, es **0**.
  - Para los booleanos es **false**.
  - Para objetos el valor es **null**, es decir “no hay objeto”.

Aunque las versiones de Java más recientes marcan error si el programador no asigna un valor explícitamente.

- Estos valores se pueden cambiar en el constructor.
- Si el programador no agrega explícitamente ningún constructor, Java pone un **constructor por defecto**, que no recibe parámetros y asigna los valores por defecto a todas las variables.
- En cuanto el programador agrega un constructor, desaparece el constructor por defecto.

# Sintaxis

¡El constructor siempre se llama igual que la clase y no tiene tipo de regreso!

## Sintaxis (Constructor)

```
<constructor> ::= <acceso> <identificador de Clase> (<Parámetros>){  
    <implementación>  
}  
<Parámetros> ::= <parámetro> (, <Parámetros>)* | ∅  
<parámetro> ::= <tipo> <identificador>
```

# Ejemplo

```
1  public class Persona{
2      ...
3      //Firma: Persona()
4      public Persona(){
5          ...
6      }
7      //Firma: Persona(String, short, Color, Color, Sexo)
8      public Persona(String nombre, short edad,
9                      Color colorOjos, Color colorCabello,
10                     Sexo sexo){
11          ...
12      }
13 }
```

# Métodos en general

## Sintaxis (Método)

```
<método> ::= <acceso> (<tipo>|void) <identificador>(<Parámetros>){  
    <implementación>  
}
```

- Los nombres de los métodos inician con minúsculas.

# Métodos

```
1  public class Persona{
2      ...
3      public String getNombre(){ //Acceso
4          ...
5      }
6      private void daPaso(Dirección dirección){//Auxiliar
7          ...
8      }
9      public void camina(){ //Manipulación
10         ...
11     }
12     public Reporte hazReporte(){ //Implementación
13         ..
14     }
15 }
```

# Advertencia

**OJO:** no repetir firmas de métodos dentro de una misma clase.

```
1  public class Matemático{  
2      ...  
3      public double calculaÁngulo(double cateto,  
4                                  double hipotenusa){  
5          ...  
6      } //Firma: calculaÁngulo(double, double)  
7      public double calculaÁngulo(double cOpuesto,  
8                                  double cAdyacente){  
9          ...  
10     } //Firma: calculaÁngulo(double, double)  
11 }
```

# Método main

```
1  public class Argumentos{
2      public static void main(String[] args){
3          for(int i = 0; i < args.length; i++){
4              System.out.println("Argumento_" + i + ":_")
5                  + args[i]);
6          }
7      }
8  }
```

Al llamar al programa en consola:

```
user@compu:~/$ java Argumentos
user@compu:~/$ java Argumentos algo otro tercero
Argumento 1: algo
Argumento 2: otro
Argumento 3: tercero
```

# Construcción de objetos

## Sintaxis (Constructor)

`<construcción de objeto> ::= new <invocación de método constructor>`

```
1 Persona m = new Persona("Kike", 15, colorOjos, colorCabello,
2                     Sexo.MASCULINO);
3 Persona f = new Persona("Lilí", 15, colorOjos, colorCabello,
4                     Sexo.FEMENINO);
```

- Los objetos se crean en el montículo.
- Lo que devuelven `new` y la función constructora es la dirección del objeto que fue creado.

# Accediendo a los miembros de un objeto

## Sintaxis (Derreferenciar)

`<Referencia a miembro> ::= <identificador de objeto>.<id de miembro>`

Solamente es posible acceder a los atributos para los que se tiene permiso.

```
1 // Dentro de la clase Persona
2 Persona otra = new Persona();
3 otra.nombre;
```

# this

- Todos los métodos de una clase reciben como parámetro al objeto al que se envía el mensaje. En Java se le llama `this`.

```
1 public class Persona{
2     /** Constructor. */
3     public Persona(String nombre, short edad,
4                     Color colorOjos, Color colorCabello,
5                     Sexo sexo){
6         this.nombre = nombre;
7         this.edad = edad;
8         this.colorOjos = colorOjos;
9         this.colorCabello = colorCabello;
10        this.sexo = sexo;
11    }
12 }
```

# Invocación de método

## Sintaxis (Invocar método)

```
<invocación de método> ::= this.<nombre del método> (Argumentos)
    | <nombre del método> (Argumentos)
<Argumentos> ::= <argumento>,<Argumentos> | ∅
<argumento> ::= <constante> | <variable>
```

```
1  persona.camina();
2  this.daPaso();
```

# Bibliografía

- 1 Organización
- 2 Tipos
- 3 Orientación a objetos en Java
- 4 Bibliografía



# Licencia

Creative Commons  
Atribución-No Comercial-Compartir Igual

