POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Universidad Politécnica de Madrid

ETSI SISTEMAS INFORMÁTICOS

Fundamentos de Programación

https://short.upm.es/xlh3k

Tema 1:

Introducción a la Programación

Índice

Introducción a la Programación.

1. Conceptos generales de la programación.
2. Estructura de un programa en Java.
3. Elementos de un programa Java.
4. Entrada y salida estándar.

1. Conceptos generales de la programación

1.1. Conceptos básicos.

1.2. Tipos de lenguajes de programación.

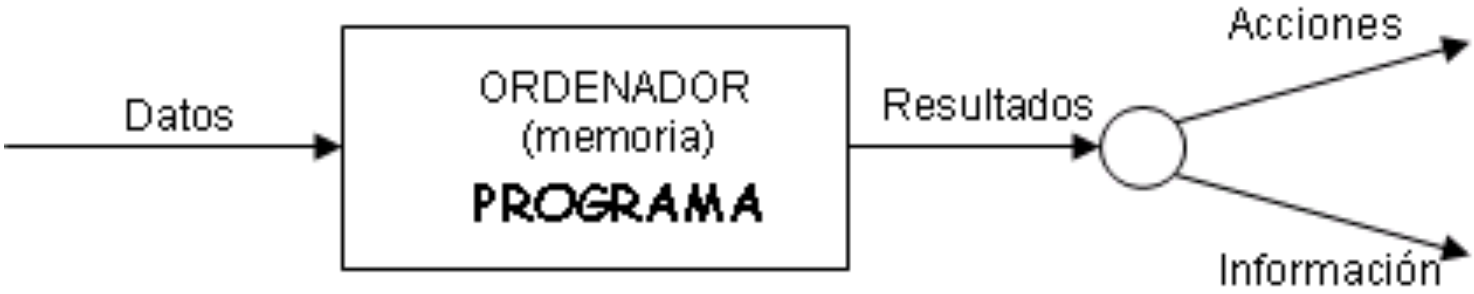
1.1. Conceptos básicos

Algoritmo:

- Conjunto de **instrucciones** o reglas bien definidas que permiten realizar una actividad mediante pasos sucesivos.

Programa:

- Conjunto de **instrucciones** que operan sobre **datos** para producir **resultados**.
- "*Conjunto de algoritmo más estructuras de datos*". Wirth (1985)



1.1. Conceptos básicos

Programación:

- Proceso de diseñar, implementar, depurar y mantener el código fuente de programas.
- El código fuente se escribe en un lenguaje de programación.

Lenguaje de programación:

- Conjunto de símbolos y reglas gramaticales para dar instrucciones a un sistema informático para que ejecute tareas específicas.

1.1. Conceptos básicos

Metodología de programación:

- Analizar el problema.
- Diseñar los algoritmos.
  - Diseño descendente y modular.
  - Programar cada módulo lo más independiente posible.
- Codificar el programa.
- Pruebas y puesta en marcha.

1.2. Tipos de lenguajes de programación

- Lenguajes de programación:
  - Lenguaje máquina.
  - Lenguaje ensamblador.
  - Lenguajes de alto nivel.

1.2. Tipos de lenguajes de programación

Lenguaje máquina:

Alfabeto {0, 1}

- Complicado: lento de redacción y programas largos.
- Elevada posibilidad de cometer errores.
- Orientado a la máquina (específico de cada tipo de máquina).
- Necesita personal especializado.

0001000000000101

cod. operación operando

BIT = Binary Digit.
Es la unidad más pequeña de representación de la información.

1.2. Tipos de lenguajes de programación

Lenguaje ensamblador:

Alfabeto: {a, b, c ... 0, 1, 2 ...), * , + ...}

- Complicado: lento de redacción y programas largos.
- Menor posibilidad de cometer errores que en el lenguaje máquina.
- Programas más legibles y más cómodos de revisar.
- Orientado a la máquina (es específico para cada tipo de máquina).
- Necesita personal especializado.
- Necesita un traductor (aumenta el trabajo del ordenador).

LOAD A
cod. op. operando

1.2. Tipos de lenguajes de programación

Lenguaje de alto nivel:

Alfabeto: {a, b, c ... 0, 1, 2, ...), * , +, ...}

- Universales --- independientes de la máquina.
- Orientados a Problemas.
- Programas más cortos.
- Necesitan un traductor.

1.2. Tipos de lenguajes de programación

Tipos de traductores:

- Compiladores.
  - Generan un programa ejecutable a partir del código fuente (Pascal, C, Fortran).
- Intérpretes.
  - Analizan, traducen y ejecutan las instrucciones una a una (basic, Ruby, Python, JavaScript).

1.2. Tipos de lenguajes de programación

- **Clasificación de los lenguajes de programación de alto nivel:**
  - Lenguajes imperativos:
    - Programación estructurada (C, Pascal, Fortran).
    - Programación orientada a objetos (Smalltalk, C++, Java, C#).
  - Lenguajes declarativos (funcionales y lógicos):
    - Lisp, Prolog.

1.2. Tipos de lenguajes de programación

Programación estructurada:

- **Todo proceso** se compone de una serie de acciones o sentencias que se ejecutan en secuencia (una a continuación de otra, desde la primera hasta la última sentencia).
- Las **acciones** pueden ser **simples** (también llamadas básicas o primitivas) o **compuestas** (estructuras de control).
- Las estructuras de control son: secuencial , alternativa o iterativa.
- Toda estructura de control tiene un solo punto de entrada y un solo punto de salida.

1.2. Tipos de lenguajes de programación

Programación orientada a objetos:

- Todos los datos (excepto los tipos primitivos) son **objetos.**
- Los **objetos** se comunican entre sí pasándose **mensajes**.
- Cada **objeto** tiene un estado (Valores de los datos).
- Un **objeto** es un caso particular (instancia) de una **clase.**
- Las **clases** definen el comportamiento de un conjunto de **objetos**.
- Un programa orientado a objetos es un conjunto de objetos que interactúan.

1.2. Tipos de lenguajes de programación

Programación orientada a objetos:

Clase: descripción de los datos y las operaciones que describen el comportamiento de una entidad(conjunto de elementos homogéneos).

Ej. Clase *Rectangulo*:

- Datos: alto, ancho y color.
- Operaciones: área, girar, perímetro, etc.

Ej. Clase *Coche*:

- Datos: marca, modelo, color, número de matrícula, año de matrícula, propietario, última itv.
- Operaciones: cambiar propietario, pasar itv, repostar, etc.

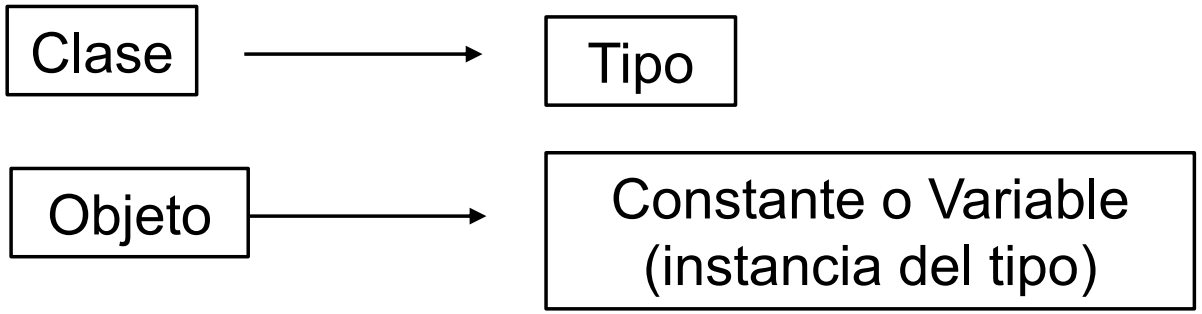
1.2. Tipos de lenguajes de programación

Programación orientada a objetos:

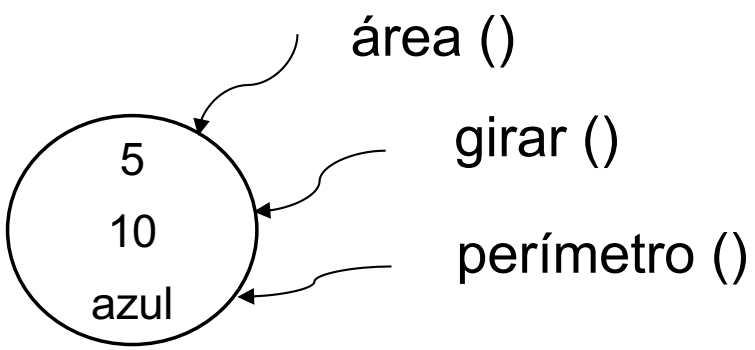
- **Objeto:** ejemplar concreto (instancia) de una clase, que responde al comportamiento definido por las operaciones de la clase a la que pertenece, adecuándose al estado de sus datos particulares.
- **Atributo**: cada uno de los datos de una clase, y por tanto, presente en todos los objetos de esa clase.
- **Estado:** conjunto de los valores de los atributos que tiene un objeto, por pertenecer a una clase, en un instante dado.
- **Método**: definición de una operación de una clase. Actúan sobre el estado del objeto (atributos del objeto).
- **Mensaje:** invocación de una operación (método) sobre un objeto.

1.2. Tipos de lenguajes de programación

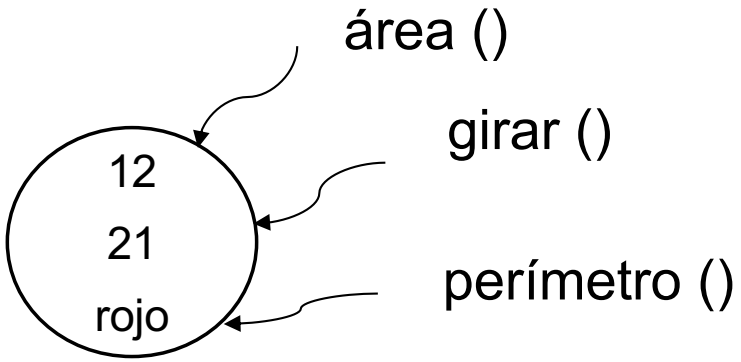
Programación orientada a objetos:



`rectangulo1`



`rectángulo2`

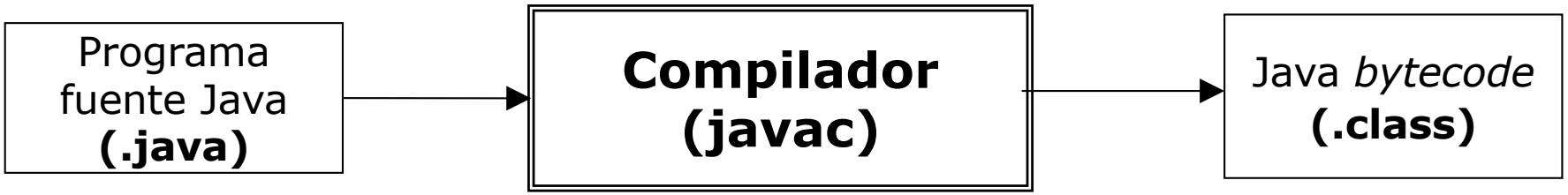


1.3. Características del lenguaje Java

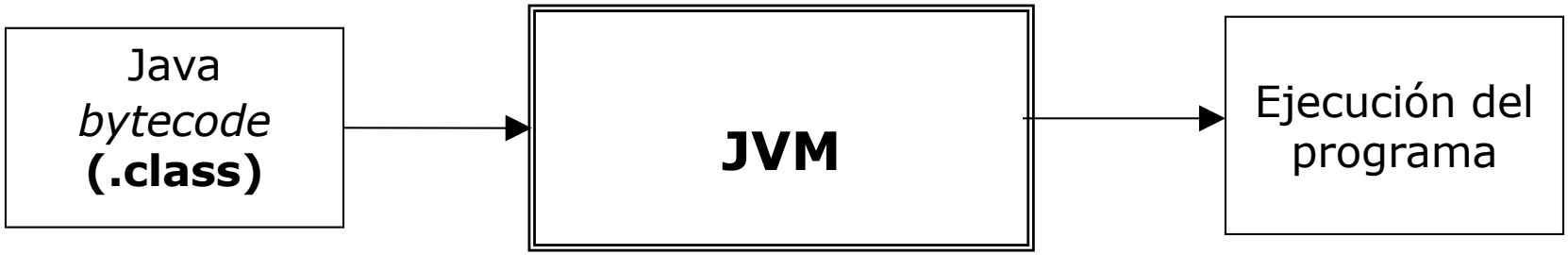
- **Orientado a Objetos**: excepto en los tipos primitivos.
- **De propósito general:** no está orientado a un tipo de aplicaciones particular.
- **Fuertemente tipado:** toda variable debe declararse con un tipo y éste no se puede cambiar durante la ejecución.
- **Robusto:** se realizan chequeos en tiempo de compilación y en tiempo de ejecución. Por ejemplo, no es posible salirse de los límites de un *array*, ni utilizar un puntero con valor *null*.
- **Gestión automática de memoria**: no es necesaria la liberación explícita de memoria. La JVM tiene un *garbage collector* que elimina periódicamente los objetos que no están referenciados.
- **Uso implícito de punteros**: no existe aritmética de punteros
- **Portable**: una aplicación se compila una vez y se puede ejecutar en diferentes plataformas.
- **Concurrente.**

1.3. Características del lenguaje Java

La plataforma de desarrollo Java:



- El *bytecode* es independiente de la plataforma.
- Cada plataforma (arquitectura + sistema operativo) tiene una Máquina Virtual Java (**JVM**).



- Los *bytecodes* son interpretados (ejecutados) por la JVM.

1.3. Características del lenguaje Java

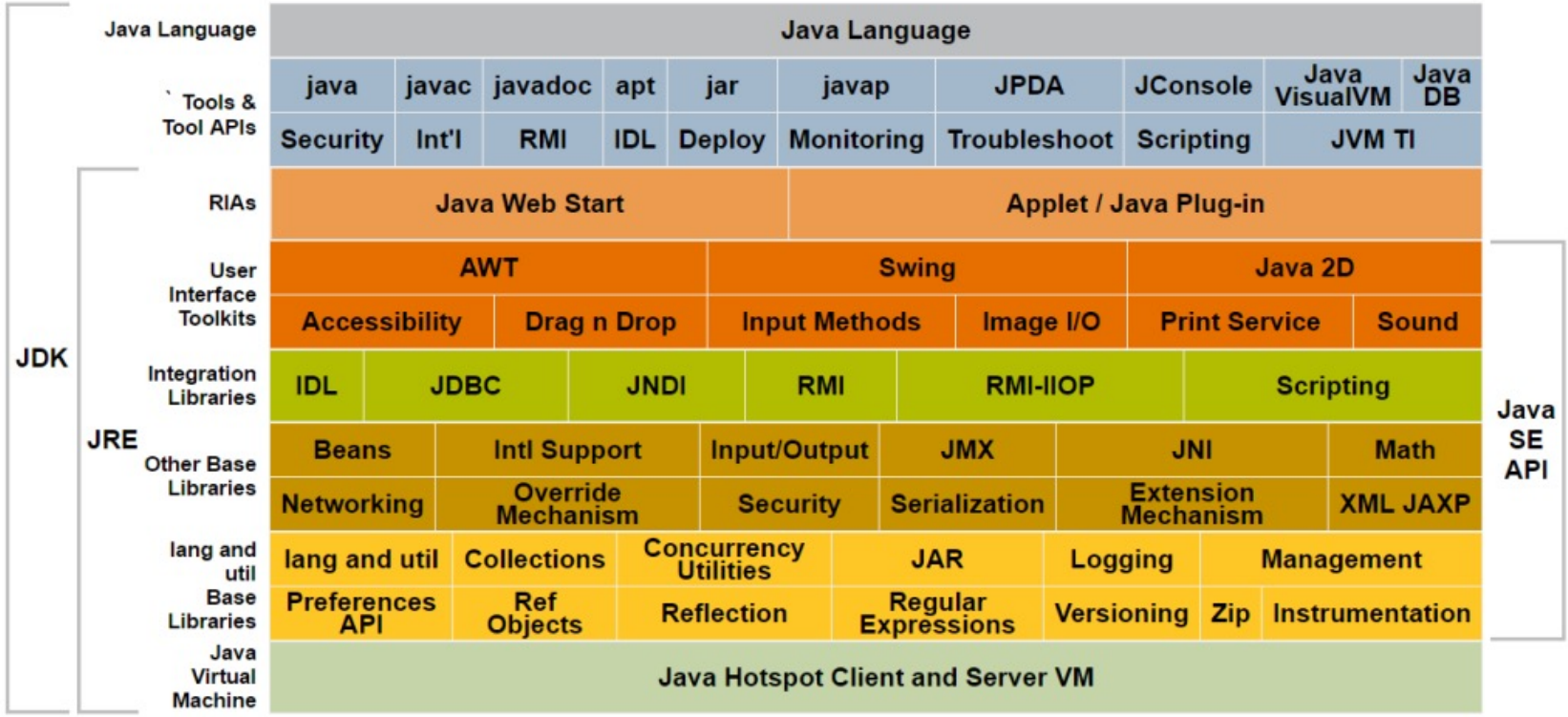
La biblioteca Java.

La biblioteca estándar de Java incorpora una serie de módulos (clases) que pueden ser reutilizados en cualquier aplicación:

- Clases básicas del lenguaje: *java.lang*
- Utilidades y estructuras de datos: *java.util*
- Acceso a ficheros: *java.io*
- Interfaz gráfica de usuario: *javax.swing*
- Acceso a redes: *java.net*
- Acceso a base de datos: *java.sql*
- …

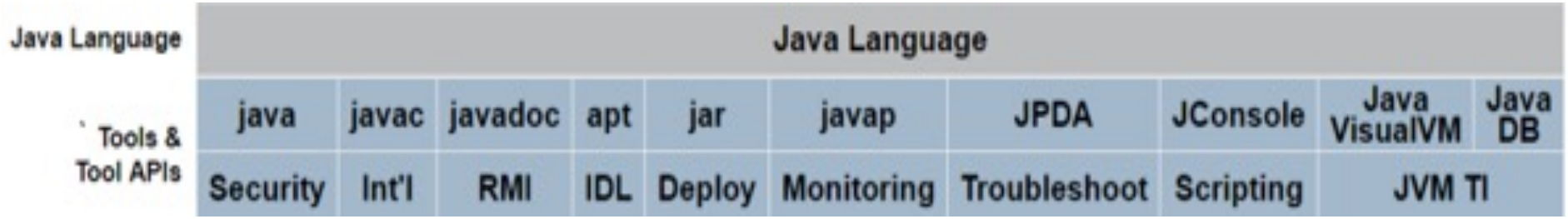
1.3. Características del lenguaje Java

- **Java SE Runtime Environment (JRE)**: incluye la máquina virtual y la biblioteca, todo lo necesario para ejecutar aplicaciones.
- **Java SE Development Kit (JDK)**: incluye al JRE y las herramientas para desarrollar aplicaciones.



1.3. Características del lenguaje Java

Herramientas de desarrollo Java (JDK)



- **java**: ejecuta los ficheros de *bytecode* (.class).
- **javac**: compila programas java (.java) obteniendo ficheros en formato de *bytecodes* (.class).
- **javadoc**: genera automáticamente documentación (.html).
- **jar**: creación de archivos de distribución (.jar).
- …

1.3. Características del lenguaje Java

Entorno de desarrollo integrado (IDE).

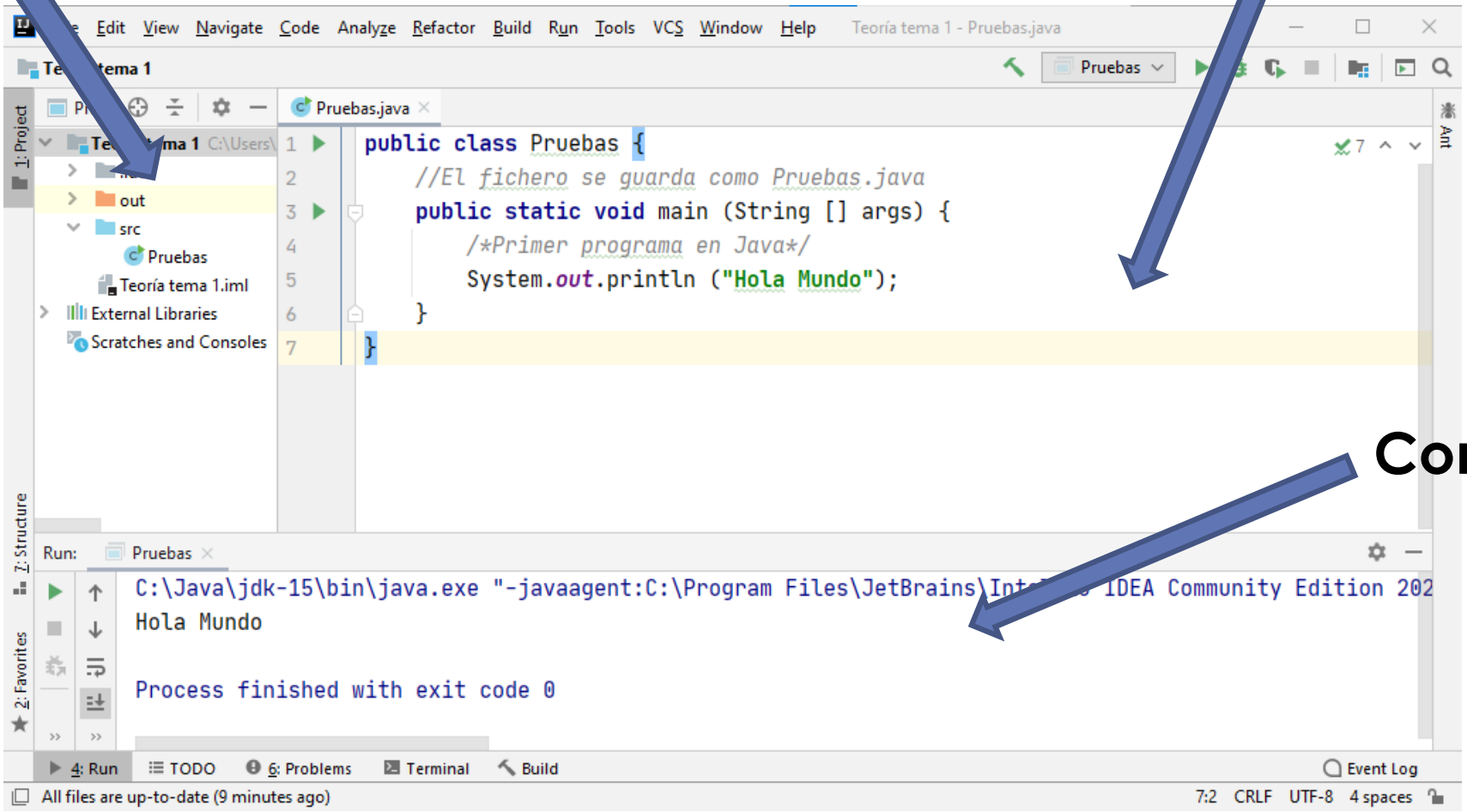
- IDE: *Integrated Development Environment.*
- Aplicación que integra varios servicios para facilitar el desarrollo de software:
  - Editor de código fuente.
  - Compilador/intérprete.
  - Depurador.
- Utilizaremos IntelliJ de idea: **Ver. IntelliJ IDEA Community Edition**
  - Disponible en el escritorio virtual de la universidad (https://escritorio.upm.es).
  - Descarga: https://www.jetbrains.com/es-es/idea/

1.3. Características del lenguaje Java

Proyecto

Entorno de desarrollo (IDE):

Editor



Consola

2. Estructura de un programa en Java.

2.1. Estructura básica.

2.2. Componentes de un programa en java.

2.1. Estructura básica

```
/* Este programa escribe el texto "Hola Mundo"
* en la pantalla utilizando el método
* System.out.println ()
*/
public class HolaMundo {
    //El fichero se guarda como HolaMundo.java
    public static void main (String [] args) {
        System.out.println ("Hola Mundo");
    }
}
```

Comentario

Clase

Comentario

Método

Sentencia

2.2. Componentes de un programa en java

- Clase
  - El programador decide el nombre de la clase.
  - Para indicar que es una clase, se incluye la palabra reservada ***class.***

```
public class HolaMundo{}
```

  - Incluye todas las líneas comprendidas entre { y }.
  - Se guarda en un fichero con el mismo nombre y extensión .java (fichero HolaMundo.java).

2.2. Componentes de un programa en java

- Método:
  - Todos los programas deben incluir un método *main*.

```
public static void main (String [] args){}
```

  - Consiste en una secuencia de sentencias incluidas entre { y }.
- Sentencia:
  - Indicamos que queremos escribir por pantalla el texto que aparece entre comillas.

```
System.out.println ("Hola Mundo");
```

3. Elementos de un programa Java

3.1. Comentarios.

3.2. Identificadores.

3.3. Palabras reservadas.

3.4. Tipos de datos.

3.5. Operadores.

3.6. Expresiones.

3.7. Asignaciones.

3. Elementos de un programa Java

- Un programa escrito en java es una secuencia de caracteres del **alfabeto** sobre el que se define el lenguaje.
- Estos caracteres se agrupan formando símbolos que componen cadenas sintácticas básicas o el vocabulario del lenguaje (elementos léxicos):
  - Comentarios.
  - Identificadores.
  - Palabras reservadas.
  - Tipos de datos.
  - Operadores.
  - Reglas de Sintaxis (expresiones y asignaciones).

3.1. Comentarios

- Comentarios de varias líneas:
  - Se delimitan por /* y */.
  - No se permiten anidamientos.

```
/* Este programa escribe el texto "Hola Mundo"
* en la pantalla utilizando el método
* System.out.println ()
*/
```

- Comentario en una sola línea:
  - Se indica con //.

```
// El fichero se guarda como HolaMundo.java
```

3.1. Comentarios

- Comentarios de documentación:
  - Se delimitan por /** y */

```
/**
 * Programa HolaMundo
 * @author Fundamentos de Programación
 * @versión 1.1
 */
```

  - Se puede usar javadoc para documentar automáticamente los programas.
- El compilador ignora los comentarios, se usan para que el programa sea más legible.

3.2. Identificadores

- Nombres para hacer referencia a procesos(métodos) u objetos.
- Secuencia de caracteres (letras, dígitos, y otros caracteres):
  - No pueden empezar por un número.
  - No pueden contener caracteres especiales:

    + - * / = % & # ! ? ^ “ ‘ ~ \ | < > ( ) [ ] { } : ; . ,
- Java es sensible a mayúsculas: cuadrado ≠ Cuadrado.
- Normas de estilo:
  - Los identificadores siempre comienzan con una letra.
  - Si hay varias palabras se separan con una mayúscula: holaMundo.
  - Los nombres de métodos empiezan con minúscula.
  - Los nombres de clase empiezan por mayúscula: HolaMundo.

3.2. Identificadores

Indicar cuáles de los siguientes identificadores son correctos:

DATO V	dato V	_numero V
LeerNombre V	-numero X	número V
num1 V	Num_1 V	1#_Num X
Apellido1* X	Datos V	cOlOr V

+ - * / = % & # ! ? ^ “ ‘ ~ \ | < > ( ) [ ] { } : ; . ,

3.3. Palabras reservadas

- Palabras con un significado especial que no se pueden utilizar como identificadores.
- En java tenemos las siguientes:

abstract	assert	boolean	break	byte
case	catch	char	class	const *
continue	default	do	double	else
enum	extends	final	finally	float
for	goto *	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while
false	*null*	*true*		

3.4. Tipos de datos

- Los datos en Java pueden ser de dos tipos:
  - Tipos primitivos:
    - Representan valores básicos:
      - Datos numéricos: enteros y reales.
      - Caracteres.
      - Valores lógicos.
  - Referencia a un objeto:
    - Cadenas de caracteres.
    - Otros objetos de clases predefinidas de Java.
    - Tipos recubrimiento o *wrapper*.
    - Objetos de clases definidas por el usuario.

3.4. Tipos de datos

- Clasificación de tipos primitivos:
  - *Enteros*: representan números enteros positivos y negativos con distintos rangos de valores.

    byte short int long
  - *Reales: r*epresentan números reales positivos y negativos con distintos grados de precisión.

    float double
  - *Lógicos:* representa un valor lógico (booleano).

    boolean
  - *Caracteres*: representan un único carácter Unicode (extensión de ASCII).

    char

3.4. Tipos de datos

- **Valores de los tipos primitivos:**

Tipo	Descripción	Bytes	Valores mínimo y máximo
byte	Entero con signo	1	-128 a 127
short	Entero con signo	2	-32768 a 32767
int	Entero con signo	4	-2147483648 a 2147483647
long	Entero con signo	8	-92211703685477580 8 a 92211703685477580 7
float	Real de simple precisión	4	+-3.40282347e+38 a +-1.40239846e-45
double	Real de doble precisión	8	+-1.79769313486231570e+308 a +-4.94065645841246544e-324
char	Carácter Unicode	2	\u0000 a \uFFFF
boolean	Verdadero o falso	1	true o false

3.4. Tipos de datos. Literales

Literales enteros: se escriben en base decimal, binario (base 2), octal (base 8), o hexadecimal (base 16). Son de tipo **int**:

```
Ej.: -21 (decimal)      0b00100001 (binario)
       025 (octal)      0x15 (hexadecimal)
```

- *Literales reales*: se escriben con un punto decimal o con notación científica. Son de tipo **double**:

```
Ej.: 5.0   .76   -3.14   56.34E-45
```

- *Literales lógicos*: se escriben mediante dos palabras reservadas. Son de tipo **boolean**.

```
Ej.: true false
```

3.4. Tipos de datos. Literales

- *Literales caracteres*: se escriben directamente entre comillas simples o mediante su representación Unicode en hexadecimal.

```
Ej.:        'f' 'P' '4' '*' '_' ';'
            '\u0041'(código Unicode para A)
```

o mediante la notación de la tabla para los caracteres de escape:

Carácter	Significado	Carácter	Significado
'\t'	Tabulador	'\b'	Retroceso
'\n'	Salto de línea	'\"'	Carácter comillas dobles
'\r'	Retorno de carro	'\''	Carácter comilla simple
'\f'	Salto de página	'\\'	Carácter barra hacia atrás

```
Ej.: '\n'
```

3.4. Tipos de datos. Literales

Indicar los literales enteros incorrectos	Indicar los literales reales incorrectos	Indicar los literales carácter incorrectos
1.000.000 X	1.000.000 X	"a" X
546	-5.56	'a'
9234567890 X	-5,56 X	'aaa' X
145	6.56	'0'
-14534	.67	'9'
+32.7 X	8E20	'\n'
+53		
898+ X		
25,56 X		

Tipo	Valores mínimo y máximo
byte	-128 a 127
short	-32768 a 32767
int	-2147483648 a 2147483647
long	-9221170368547758O8 a 9221170368547758O7

Tipo
float
double

Carácter		Carácter	
'\b'	Retroceso	'\t'	Tabulador
'\"'	Carácter comilla simple	'\r'	Retorno de carro
'\\'	Carácter barra hacia atrás	'\f'	Salto de página

3.4. Tipos de datos. Variables y Constantes

Los datos de un programa pueden ser:

- **Variables.**
  - Su valor puede variar durante la ejecución de un programa.
  - Referencia simbólica a una posición de memoria, donde se almacena un valor de un cierto tipo de datos.
  - Ejemplo: base, altura, perímetro y área de un círculo.
- **Constantes.**
  - No se deben modificar durante la ejecución de un programa.
  - Ejemplo: PI, TAMANIO_MAXIMO(TAMANIO_MAXIMO)

Declaración de variables

```
<tipo> <variable> [ = <expresión[1]> ];
```

- Regla de estilo:
  - El nombre de la variable debe ser significativo, sin abreviaturas.
  - Debe ser en minúsculas salvo cuando se produce un cambio de palabra, donde la primera letra de la nueva palabra será en mayúscula.
- Ejemplos:

```
int contador;
boolean aprobadoParcial;
char símboloNota;  // mejor no
```

[1] *las expresiones se verán más adelante.*

Declaración de variables

- Si se quiere dar un valor inicial a la variable:

  <tipoDato> <identificadorVariable> = <valor>;

- Ejemplos:

```
int contador = 0;
boolean aprobadoParcial = false;
char simboloNota = 'N';
```

Declaración de variables

- Declarar las siguientes variables:
  - Variables enteras: i, j, valor.
  - Variable entera larga: factorial.
  - Variable entera corta: indicador.
  - Variables double: x, y.
  - Variables float: nota, precio.
  - Variables carácter: a, b, primerCaracter.
  - Variables lógicas: primero, último.

```
int i, j, valor;
long factorial;
short indicador;
double x, y;
float nota, precio;
char a, b, primerCaracter;
boolean primero, ultimo;
```

Declaración de variables

- Declarar las siguientes variables, y asignarles valor en la declaración:
  - Variables enteras: i, j.
  - Variable entera larga: factorial.
  - Variable entera corta: indicador.
  - Variables double: x, y.
  - Variables float: nota, precio.
  - Variables carácter: a, primerCaracter.
  - Variable lógica: primero.

```
int i=0, j=0;
long factorial=1;
short indicador=0;
double x=0.8869, y=1.765E-1;
float nota=6.3f, precio=34.99f;
char a='A', primerCaracter='a';
boolean primero=true;
```

Declaración de variables

- Escribir la declaración más apropiada para las siguientes variables:
  - edad. `int edad;`
  - altura. `double altura;`
  - peso. `double peso;`
  - alumnoRepetidor. `boolean alumnoRepetidor;`
  - contador. `int contador;`
  - inicial. `char inicial;`

Declaración de constantes

```
final <tipo> <constante> [ = <expresión[1]> ];
```

- Regla de estilo:
  - el nombre de la constante debe ser significativo, sin abreviaturas.
  - Debe ser en mayúsculas, separándose con un subrayado cuando se produce un cambio de palabra.
- Ejemplo:

```
final int MAXIMO = 100;
final double PONDERACION = 0.8;
final char SEPARADOR_FECHAS = '/';
```

[1] *las expresiones se verán más adelante.*

Declaración de constantes

- Escribir la declaración más apropiada para las siguientes constantes:
  - Valor de Pi.
  - Carácter dos puntos.
  - Número de días de la semana.
  - Primera vocal.

```
final double PI=3.14159;
final char DOS_PUNTOS=':';
final int DIAS_SEMANA=7;
final char PRIMERA_VOCAL='a';
```

3.5. Operadores

- **Operadores Aritméticos**: operando sobre valores numéricos devuelven un valor del tipo de los operandos, excepto *byte*, *short* y *char*, que devuelven *int*.

Operador	Uso	Descripción
+	<opI> + <opD>	Devuelve la suma de <opI> y <opD>
-	<opI> – <opD>	Devuelve la resta de <opD> de <opI>
*	<opI> * <opD>	Devuelve el producto de <opI> y <opD>
/	<opI> / <opD>	Devuelve el cociente de dividir <opI> entre <opD>
%	<opI> % <opD>	Devuelve el resto de dividir <opI> entre <opD>
+	+ <op>	Devuelve el valor de <op>
-	– <op>	Devuelve el valor opuesto de <op>

3.5. Operadores aritméticos

- Ejemplos:

Operación	Tipo del resultado	Resultado
7/3	`int`	`2`
5.0/2	`double`	`2.5`
7/0	`int`	`/ by zero`
7.0/0.0	`double`	`Infinity`
0.0/0.0	`double`	`NaN (indeterminado)`
7%3	`int`	`1`

3.5. Operadores relacionales

- **Operadores Relacionales**: operan sobre dos valores del mismo tipo y devuelven un valor de tipo lógico.

Operador	Uso	Descripción
>	<opI> > <opD>	Devuelve cierto si opI es mayor que opD
>=	<op1> >= <op2>	Devuelve cierto si opI es mayor o igual que opD
<	<op1> < <op2>	Devuelve cierto si opI es menor que opD
<=	<op1> <= <op2>	Devuelve cierto si opI es menor o igual que opD
==	<op1> == <op2>	Devuelve cierto si opI es igual que opD
!=	<op1> != <op2>	Devuelve cierto si opI es distinto de opD

3.5. Operadores lógicos

- **Operadores Lógicos**: operan sobre valores lógicos y devuelven un valor de tipo lógico.

Operador	Uso	Descripción
!	! <op>	Devuelve el valor negado
&&	<opI> && <opD>	Y lógico: Devuelve verdadero si ambos operandos son verdaderos
||	<opI> || <opD>	O lógico: Devuelve verdadero si algún operando es verdadero

x	!x
false	true
true	false

a	b	a && b	a || b
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

3.6. Expresiones

- **Reglas de construcción:**
  - Los operandos serán los valores de variables, constantes, literales o el valor devuelto por otra expresión.
  - Cada operador debe acompañarse en tipo, número y posición de operandos según su clase (carácter binario o unario).
- **Reglas de evaluación:**
  - Cuando un operando está situ
    será operando de aquel operac
    precedencia y, en caso de igua
    establecida para ese nivel.
  - En el operador && se evalúa primero la parte izquierda y si resulta *false*, no se evalúa la parte derecha.
  - En el operador || se evalúa primero la parte izquierda y si resulta *true*, no se evalúa la parte derecha.

a	b	a && b	a || b
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

3.6. Expresiones

- Indicar el valor de evaluar la siguiente expresión:

$$(10 * (50 + 20) \% 7) + (4 / 3)$$

$$(10 * 70 \% 7) + (4 / 3)$$

$$(700 \% 7) + (4 / 3)$$

$$0 + (4 / 3)$$

$$0 + 1$$

$$1$$

3.6. Expresiones

Ejemplos:

- Expresión que compruebe si un número es par o no.

```
numero%2 == 0
```

- Expresión que compruebe si dos caracteres c1 y c2 son iguales.

```
c1 == c2
```

- ¿Puedes darle un significado a las expresiones siguientes? (siendo *a* una variable entera).

  10 < a < 100

  10 < a && a < 100

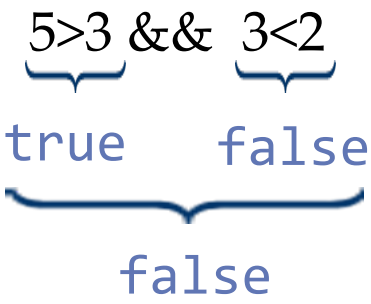
¿Son equivalentes?
Evaluarla para a = 4 y a = -4.

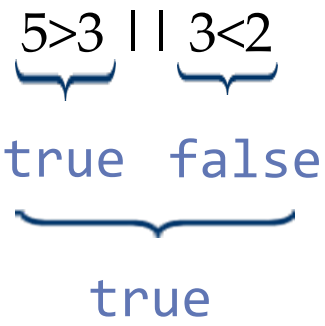
```
a=4, false   a=-4, false
```

3.6. Expresiones

Ejemplos:

- ¿Cuál es el resultado de las siguientes expresiones?:





3.6 Expresiones

- **Tabla de precedencia y asociatividad:**

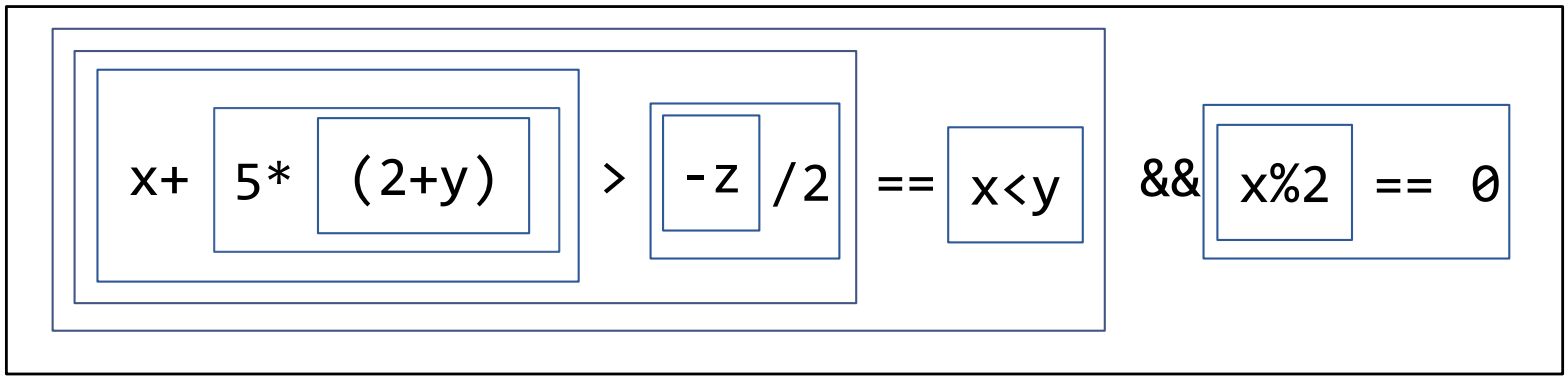
Nivel	Tipos de Operadores	Operadores	Asociatividad
max	Unarios	+ - !	Derecha > Izquierda
	Multiplicativos	* / %	Izquierda > Derecha
	Aditivos	+ -	Izquierda > Derecha
	Relacionales	< > <= >=	No asociativo
	Igualdad	== !=	Izquierda > Derecha
	Y Lógico	&&	Izquierda > Derecha
min	O Lógico	||	Izquierda > Derecha

Los paréntesis rompen las reglas de precedencia y asociatividad.

Nivel	Tipos de Operadores	Operadores	Asociatividad
max	Unarios	+ - !	Derecha > Izquierda
	Multiplicativos	* / %	Izquierda > Derecha
	Aditivos	+ -	Izquierda > Derecha
	Relacionales	< > <= >=	No asociativo
	Igualdad	== !=	Izquierda > Derecha
	Y Lógico	&&	Izquierda > Derecha
min	O Lógico	||	Izquierda > Derecha

```
Ej.: int x = 5;
     int y = 5;
     int z = 5;
     boolean expresion = x+5*(2+y)>-z/2==x<y&&x%2==0;
```

Aplicación de las reglas de precedencia y asociatividad:



3.6. Expresiones

Ejemplo. Convertir las siguientes expresiones algorítmicas en expresiones algebraicas :

$$\frac{1}{(x-5)} - \frac{3xy}{4}$$

- a) 1/(x-5)-3*x*y/4

$$m + \frac{n}{p-q}$$

- b) m+n/(p-q)

$$\frac{a^2}{b-c} + \frac{d-e}{f-\frac{gh}{j}}$$

- c) a*a/(b-c)+(d-e)/(f-g*h/j)

3.6. Expresiones

Conversión de tipos:

- *Promoción:* transforman un dato de un tipo a otro con el mismo o mayor espacio en memoria para almacenar información. No hay pérdida de información.
- *Contracción:* transforman un dato de un tipo a otro con menor espacio en memoria para almacenar información con la consecuente posible pérdida de información.

3.6. Expresiones

Conversión de tipos:

De forma implícita:

- Si se combinan operandos de distinto tipo, se convierten al de mayor precisión (promoción).
- Si se asigna un valor de precisión menor a una variable de mayor precisión (promoción).

```
Ej.: double d = 5.5 + 3;
     double f = 5;
```

3.6. Expresiones

Conversión de tipos:

De forma explícita:

- A través del operador de conversión de tipos (*cast*), cuyo nivel de precedencia es el de los operadores unarios. Pueden realizarse conversiones por promoción o por contracción.

```
(<tipo>) <operando>
```

```
Ejemplo: float f = (float) 5.5;
         int i = (int) f;
         byte b = (byte) (f / 2);
```

3.6. Expresiones

Ejemplo. Determina cuál es el tipo de dato de las siguientes expresiones aritméticas, si tenemos definidas las siguientes variables:

```
byte b;
short s;
long ln;
int i, j;
float f;
double d;
char c;
```

i + c	b + c	f - c	s + j
5 / j	c + c	d / i	'a' +'b'
5.5 / j	i * f * 2.5	i * f * 2.5F	2 / i
2.0 / j	2 / i + 2.0 / j	c + d	j – 5L

3.7. Asignaciones

- Permite almacenar un valor de un tipo sobre una variable de su mismo tipo.

```
<variable> = <expresión>;
```

```
Ej.: int x = 5;
     int y = 8;
     int intercambio = x;
     x = y;
     y = intercambio;
```

- También se pueden hacer asignaciones múltiples:

```
Ej.: int i, j, k;
      i = j = k = 0;
```

Asignamos el valor cero a las tres variables.

3.7. Asignaciones

- Java es un lenguaje fuertemente tipado, por lo que los siguientes ejemplos producirían un error:

```
Ej. error: float f =  5.5;
           int i =  f;
           byte b = f / 2;
```

3.7. Asignaciones

Operadores de Incremento/Decremento:

Operan sobre una variable de tipo numérico y devuelven un valor del tipo de la variable.

Op.	Uso	Descripción
++	<op> ++	Incrementa en 1 el valor de <op> Devuelve el valor de <op> antes de incrementar
++	++ <op>	Incrementa en 1 el valor de <op> Devuelve el valor de <op> después de incrementar
--	<op> --	Decrementa en 1 el valor de <op> Devuelve el valor de <op> antes de decrementar
--	-- <op>	Decrementa en 1 el valor de <op> Devuelve el valor de <op> después de decrementar

3.7. Asignaciones

Operadores de Incremento/Decremento:

Ejemplo:

Declaramos las siguientes variables:

```
int  i = 15,  j, k = 25,  m;
```

Se ejecutan las siguientes instrucciones:

```
j = i++;
//Equivale a j = i; i = i+1;
```

```
Resultado:
  j = 15
  i = 16
```

```
m = ++k;
//Equivale a k = k+1; m = k;
```

```
Resultado:
  k = 26
  m = 26
```

3.7. Asignaciones

Operadores de Acumulación:

Operan sobre una variable de tipo numérico y una expresión del mismo tipo. Devuelven un valor del tipo de la variable.

Operador	Uso	Equivale a
+=	<opI> += <opD>	<opI> = <opI> + <opD>
-=	<opI> -= <opD>	<opI> = <opI> – <opD>
*=	<opI> *= <opD>	<opI> = <opI> * <opD>
/=	<opI> /= <opD>	<opI> = <opI> / <opD>
%=	<opI> %= <opD>	<opI> = <opI> % <opD>

```
Ej.: int variable = 100;
     variable /= 2;
```

Equivalente a:

```
variable = variable/2
```

3.7. Asignaciones

Operadores de Acumulación:

Ejemplo:

Declaramos las siguientes variables:

```
int i = 15, j = 23;
```

Se ejecutan las siguientes instrucciones:

```
i += 34;
//Equivale a i = i+34;
```

Resultado:
i = 49

```
i /= 3;
//Equivale a i = i/3;
```

Resultado:
i = 16

```
j *=2;
//Equivale a j = j*2;
```

Resultado:
j = 46

```
j %=7;
//Equivale a j = j%7;
```

Resultado:
i = 4

4. Entrada y salida estándar.

4.1. Clase String.

4.2. Salida estándar.

4.3. Entrada estándar.

4.4. Ejemplos.

4.5. Salida con formato.

4. Entrada y salida estándar

- Permite la comunicación entre el programa y el usuario.
- La salida estándar se utiliza para mostrar el valor del resultado de una expresión, variable, constante, o literal.
- La entrada estándar se utiliza para introducir información necesaria para un programa.
- Los dispositivos estándar de entrada y salida normalmente son el teclado y la pantalla del ordenador.
- Por defecto, Java utiliza las cadenas de caracteres (objetos de la clase **String**) para manejar la entrada/salida.
- Para realizar la entrada y salida, trabajaremos sobre la clase **System**. Utilizaremos los siguientes atributos:
  - *out* hace referencia a la salida estándar (pantalla),
  - para referirnos al teclado usaremos *in*.

4.1. Clase String

- En java se usa la clase **String** para almacenar cadenas de caracteres.
- En Java, cada vez que usamos una cadena de caracteres instanciamos un objeto de la clase String.
- La clase String permite definir literales de forma similar a como hacemos con tipos de datos primitivos:

```
String saludo = "Hola ";
String mundo = "mundo";
```

- Se pueden concatenar cadenas usando el operador +:

```
String saludoCompleto = saludo+mundo;
```

4.1. Clase String

- Los tipos de datos primitivos se convierten automáticamente a String al concatenarlos:

```
String cadena = "Valor: ";
int valor = 5;
cadena = cadena + valor;
//cadena contendrá "Valor: 5 "
```

- Java realiza esta conversión automática por medio de las clases de recubrimiento:
  - Byte, Short, Integer, Long, Float, Double, Boolean.

4.2. Salida estándar

Salida

- La clase **System** utiliza entre otros los métodos **print** y **println** para poder utilizar la salida estándar.
- Para mostrar en la pantalla (**out**) el mensaje:

Programar es divertido

se puede utilizar el método **println** de la siguiente manera:

```
System.out.println ("Programar es divertido");
```

- Como el mensaje es una cadena de caracteres, éste debe colocarse entre comillas dobles ("").
- También podemos definir variables de tipo String :

```
String saludo = "Hola ", mundo = " mundo ";
System.out.println (saludo+mundo);
```

4.2. Salida estándar

Salida

- El método **`println`** muestra un valor en la pantalla y mueve el cursor al inicio de la próxima línea.
- El método **`print`** muestra un valor en la pantalla pero no mueve el cursor.
- Por ejemplo, otra forma de mostrar el mensaje **`Programar es divertido`** es:

```
System.out.print("Programar");
System.out.println(" es divertido");
```

4.2. Salida estándar

Salida

- Cuando se quieren escribir en una sola línea datos de distintos tipos, java los convierte automáticamente a String por medio de las clases de recubrimiento.
- Si tenemos definidas las variables:

```
double total = 100.5;
int descuento = 30;
```

- Podemos mostrar ambos valores junto con cadenas en una sola línea usando el operador + para concatenarlos :

```
System.out.println("Total: "+total+", descuento: "+descuento);
```

- Se escribirá por pantalla:

```
Total: 100.5, descuento: 30
```

4.2. Salida estándar

Ejemplo:

- Programa que calcula una nómina a partir de las horas trabajadas y el precio por hora, y lo muestra por pantalla.

```
public class Nomina {
    public static void main(String[] args) {
        int horas = 40;
        double precioHora = 25.0, total;
        total = horas * precioHora;
        System.out.println ("Nomina: €"+total);
    }
}
```

- Resultado:

```
Nomina: €1000.0


Process finished with exit code 0
```

4.3. Entrada estándar

Entrada

- Para leer datos utilizaremos la clase Scanner, perteneciente al paquete `java.util`.
- Hay que importar la clase con:

```
import java.util.Scanner;
```

- Cuando vayamos a leer datos de teclado, primero creamos un objeto de la clase Scanner:

```
Scanner teclado = new Scanner(System.in);
```

- System.in es la entrada estándar.

4.3. Entrada estándar

Entrada

- Algunos métodos de lectura:
  - `teclado.next ()` lee la siguiente cadena.
  - `teclado.nextLine ()` lee hasta el siguiente salto de línea.
  - `teclado.nextInt ()` lee el siguiente número entero.
  - `teclado.hasNextInt ()` comprueba si a continuación hay un número entero.
  - `teclado.nextDouble ()` lee el siguiente número real.
  - `teclado.hasNextDouble ()` comprueba si a continuación hay un número real.
- Clase *Scanner*

4.4. Ejemplos

- Programa que recoge de teclado las horas trabajadas y el precio por hora, calcula la nómina y la muestra por pantalla.

```
import java.util.Scanner;
public class Nomina2 {
    public static void main(String[] args) {
        int horas;
        double precioHora, total;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Introduzca el número de horas: ");
        horas = teclado.nextInt();
        System.out.print("Introduzca el precio por hora: ");
        precioHora = teclado.nextDouble();
        total = horas * precioHora;
        System.out.print ("Nomina: €");
        System.out.println (total);

    }
}
```

```
Introduzca el número de horas: 40
Introduzca el precio por hora: 25
Nomina: €1000.0

Process finished with exit code 0
```

4.4. Ejemplos

- Programa que recoge de teclado la base y la altura de un triángulo, calcula su área y la muestra por pantalla.

```
import java.util.Scanner;

public class ÁreaTriángulo {
    public static void main(String[] args) {
        double base, altura, area;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Introduzca la base: ");
        base = teclado.nextDouble();
        System.out.print("Introduzca la altura: ");
        altura = teclado.nextDouble();
        area = base * altura / 2;
        System.out.print("Área: " + area);
    }
}
```

```
Introduzca la base: 3,7
Introduzca la altura: 7,2
Área: 13.32
Process finished with exit code 0
```

4.4. Ejemplos

- Escribir un programa Java con las siguientes sentencias:
  - Declarar una variable entera n y leer de teclado su valor.
  - Incrementar n en 45. Escribir el resultado por pantalla.
  - Escribir en la misma línea n, preincremento de n, postincremento de n, y n otra vez.
  - Duplicar su valor. Escribir el resultado por pantalla.
  - Declarar una variable de tipo double (x). Pedir por teclado su valor.
  - Declarar una variable de tipo float (y). Guardar en la misma el valor de n dividido por x. Escribir la variable por pantalla.

4.4. Ejemplos

- Si en la variable n se introduce el valor 29, y en la variable x se introduce el valor 3.5, se obtendría el siguiente resultado:

```
Introduzca el valor de n (int): 29
n+45 vale: 74
n: 74, ++n: 75, n++: 75, n: 76
n duplicado: 152
Introduzca el valor de x (double): 3,5
y: 43.42857

Process finished with exit code 0
```

4.4. Ejemplos

```
import java.util.Scanner;
public class Ejemplo3 {
    public static void main (String [] args) {
        int n;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Introduzca el valor de n (int): ");
        n = teclado.nextInt();
        n+=45;
        System.out.println ("n+45 vale: "+n);
        System.out.print ("n: "+n+", ++n: "+ ++n);
        System.out.println (", n++: " + n++ + ", n: "+n);
        n*=2;
        System.out.println("n duplicado: "+n);
        double x;
        System.out.print("Introduzca el valor de x (double): ");
        x = teclado.nextDouble();
        float y = (float) (n/x);
        System.out.println("y: "+y);
    }
}
```

4.5. Salida con formato

- Al escribir por pantalla con los métodos **print** y **println** no podemos controlar el formato que tienen los elementos.
- Para escribir con formato, se usa el método **printf** de la clase **System**
  - Podremos definir las posiciones que queremos usar para escribir un número.
  - En el caso de los reales, podremos indicar cuantas cifras decimales queremos mostrar.
  - Ejemplo de printf. Si ejecutamos las siguientes instrucciones:

```
int entero = 3;
double real = 12.34798;
System.out.printf ("Número entero: %02d, real: %.2f\n",entero, real);
```

  - Se escribirá por pantalla:

```
Número entero: 03, real: 12,35
```

4.5. Salida con formato

- La sintaxis del método printf del atributo *out* de la clase System es la siguiente:

```
printf ("cadenaFormato", dato1, dato2, …);
```

- cadenaFormato contiene el texto que queremos escribir y su formato.
  - Está constituido por especificadores de formato, caracteres y secuencias de escape.
  - Se deben incluir tantos especificadores de formato como datos queremos mostrar.
- dato1, dato2, ... Son variables, constantes o expresiones.
- Los especificadores de formato se asocian uno a uno con los datos.
- Especificadores de formato:
  - %d valor entero decimal
  - %c valor carácter
  - %f valor real en forma decimal
  - %e o %E valor real en notación científica (coma/punto flotante)

```
System.out.printf ("Número entero: %02d, real: %.2f\n",entero, real);
```

4.5. Salida con formato

- Sintaxis general del formato:

```
%[flag][anchoDeCampo][.precisión]tipo
```

  - anchoDeCampo, longitud mínima que debe ocupar el valor que se quiere mostrar.
    - Si el valor es mayor que el ancho de campo indicado se ignora el formato y se emplea el espacio necesario.
  - precisión, indica el número de decimales que se deben mostrar de un valor en coma flotante:
    - Si el número de decimales del dato es menor que la precisión se completa con ceros.
    - Si el número de decimales del dato es mayor, se limita el número de decimales, redondeando el último.

4.5. Salida con formato

- Sintaxis general del formato:

```
%[flag][anchoDeCampo][.precisión]tipo
```

  - [flag] son caracteres que introducen modificaciones en el modo en que se presenta el valor:
    - '-' el valor queda justificado a la izquierda.
    - '+' el valor se escribe con signo, sea positivo o negativo.
    - ' ' si el valor es positivo se deja un espacio en blanco al comienzo, si es negativo se escribe el signo.
    - ',' se introducen puntos para separar cada tres cifras.
    - Si se quiere rellenar los huecos con ceros, se coloca un cero antes del número que indica el ancho del campo.
  - tipo indica el tipo del dato:
    - c para indicar que vamos a escribir un carácter.
    - d para los enteros en base decimal (x para hexadecimal).
    - f (con decimales) o e (notación científica) para los números reales.
    - b para escribir un dato lógico (tipo *boolean*).

4.5. Salida con formato

- Ejemplo 1 . Escribir un programa en java que:
  - Lea dos números enteros: el número que jugamos a la lotería y el número premiado en el sorteo e imprima tres líneas:
    - en la primera línea se escribirán ambos números con 5 dígitos y separando los miles.
    - en la segunda línea se escribirá nuestro número con 5 dígitos, completando con ceros por la izquierda si es necesario, y a continuación se escribirá *true* si nuestro número es el número premiado y *false* en cualquier otro caso.
    - en la tercera línea se escribirá nuestro número (con el mismo formato que en la línea anterior), así como *true* si nuestro número está premiado con el reintegro y *false* en cualquier otro caso.
  - La salida por pantalla debe ser similar a la siguiente:

```
Introduce tu número de la loteria: 4389
Introduce el número premiado:      34569
Número jugado: 4.389    Número premiado: 34.569
¿El número 04389 está premiado? false
¿El número 04389 tiene reintegro? true
Process finished with exit code 0
```

4.5. Salida con formato

- Solución del ejemplo 1.

```
import java.util.Scanner;
public class Ejemplo1SalidaConFormato {
    public static void main (String [] args) {
        Scanner teclado = new Scanner(System.in);
        int numeroJugado, numeroPremiado;
        boolean premiado, reintegro;
        System.out.print ("Introduce tu número de la loteria: ");
        numeroJugado = teclado.nextInt();
        System.out.print ("Introduce el número premiado:      ");
        numeroPremiado = teclado.nextInt();
        premiado = numeroJugado==numeroPremiado;
        System.out.printf ("Número jugado: %,5d", numeroJugado);
        System.out.printf ("\tNúmero premiado: %,5d", numeroPremiado);
        System.out.printf("\n¿El número %05d está premiado? %b",numeroJugado,premiado);
        reintegro = numeroJugado%10 == numeroPremiado%10;
        System.out.printf("\n¿El número %05d tiene reintegro? %b",numeroJugado,reintegro);
    }
}
```

```
Introduce tu número de la loteria: 4389
Introduce el número premiado:      34569
Número jugado: 4.389    Número premiado: 34.569
¿El número 04389 está premiado? false
¿El número 04389 tiene reintegro? true
Process finished with exit code 0
```

4.5. Salida con formato

- Ejemplo 2. Escribir un programa en java que:
  - lea de teclado el precio de un artículo sin iva,
  - escriba por pantalla en una línea el importe sin iva, en la siguiente el iva (correspondiente al 21%), y en la siguiente línea el importe con IVA incluido.
  - Los números se deberán escribir con dos cifras decimales, y ocupando 10 posiciones.
  - A continuación de los importes se dejará un espacio de separación y se incluirá el símbolo del euro.
  - La salida por pantalla debe ser similar a la siguiente:

```
Introduzca el importe: 29,74
Importe sin iva:     29,74 €
IVA:                  6,25 €
Importe total:       35,99 €
Process finished with exit code 0
```

4.5. Salida con formato

- Solución del ejemplo 2.

```
import java.util.Scanner;
public class Ejemplo2SalidaConFormato {
    public static void main (String [] args) {
        Scanner teclado = new Scanner(System.in);
        double importe, iva, importeTotal;
        System.out.print("Introduzca el importe: ");
        importe = teclado.nextDouble();
        iva = importe *.21;
        importeTotal= importe+iva;
        System.out.printf("Importe sin iva:%10.2f €\n",importe);
        System.out.printf("IVA:\t\t\t%10.2f €\n", iva);
        System.out.printf("Importe total:\t%10.2f €",importeTotal);
    }
}
```

```
Introduzca el importe: 29,74
Importe sin iva:     29,74 €
IVA:                  6,25 €
Importe total:       35,99 €
Process finished with exit code 0
```