Programación de Sistemas

Repaso

Julio Villena Román
<jvillena@it.uc3m.es>

MATERIALES BASADOS EN EL TRABAJO DE DIVERSOS AUTORES:
M.Carmen Fernández Panadero, Natividad Martínez Madrid

Programación de Sistemas

Primeros pasos en Java

Julio Villena Román
<jvillena@it.uc3m.es>

MATERIALES BASADOS EN EL TRABAJO DE DIFERENTES AUTORES:
M.Carmen Fernández Panadero, Natividad Martínez Madrid

Escenario I: Instalar y configurar el entorno

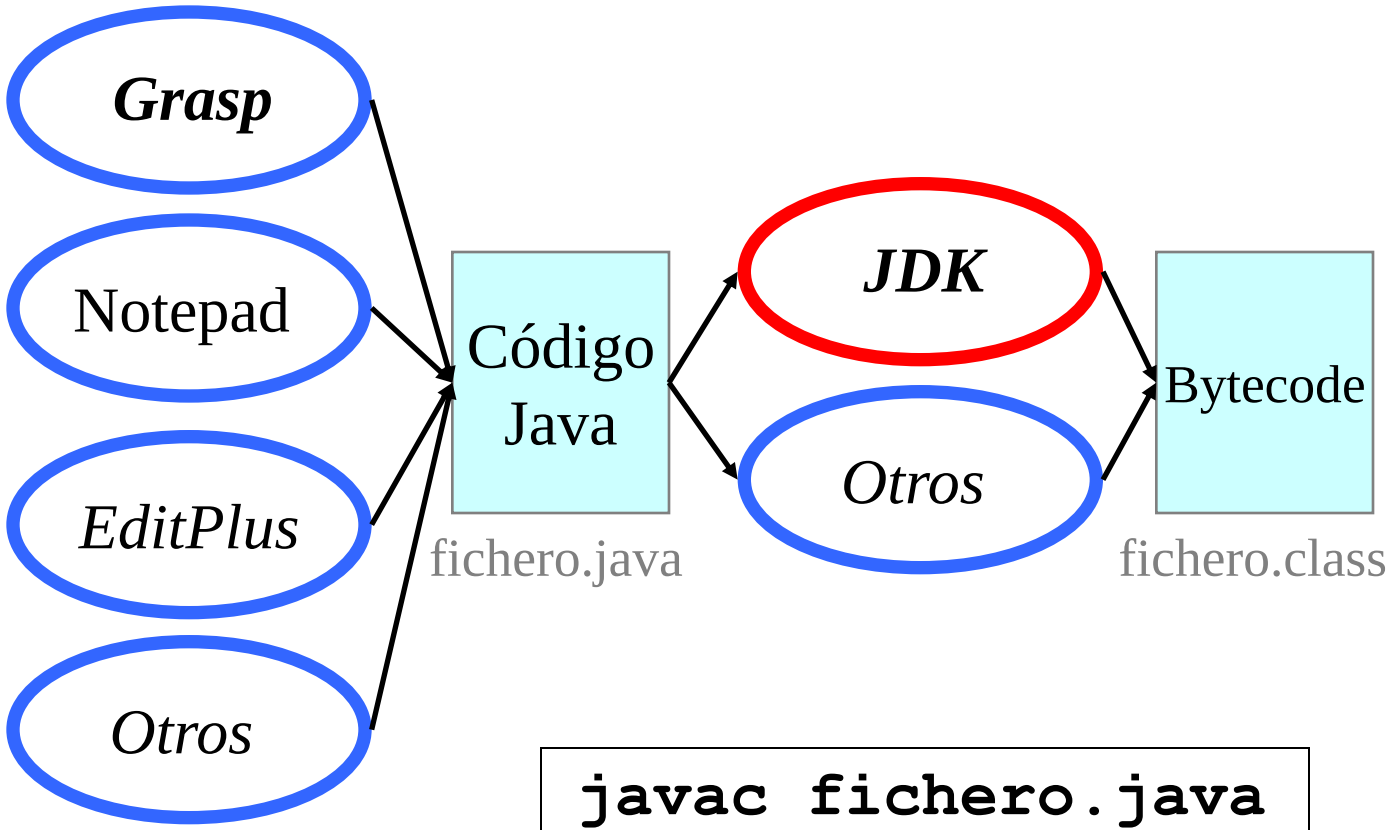
- Es tu primer día de trabajo en el departamento de programación de la empresa PROTEL donde tienen una aplicación antigua a la que hay que añadir nueva funcionalidad
- Tu jefe te proporciona un portátil y una dirección desde la que puedes descargar el código desarrollado hasta la fecha
- Objetivo: Ser capaz de ***editar, compilar, ejecutar y depurar*** un programa ya existente
- Plan de trabajo: Descargar, instalar y configurar el sw necesario para poder probar (editar, compilar, ejecutar y depurar) el código que hemos descargado.

Arquitectura de desarrollo

Fase I: Editar
Fase II: Compilar

IDEs

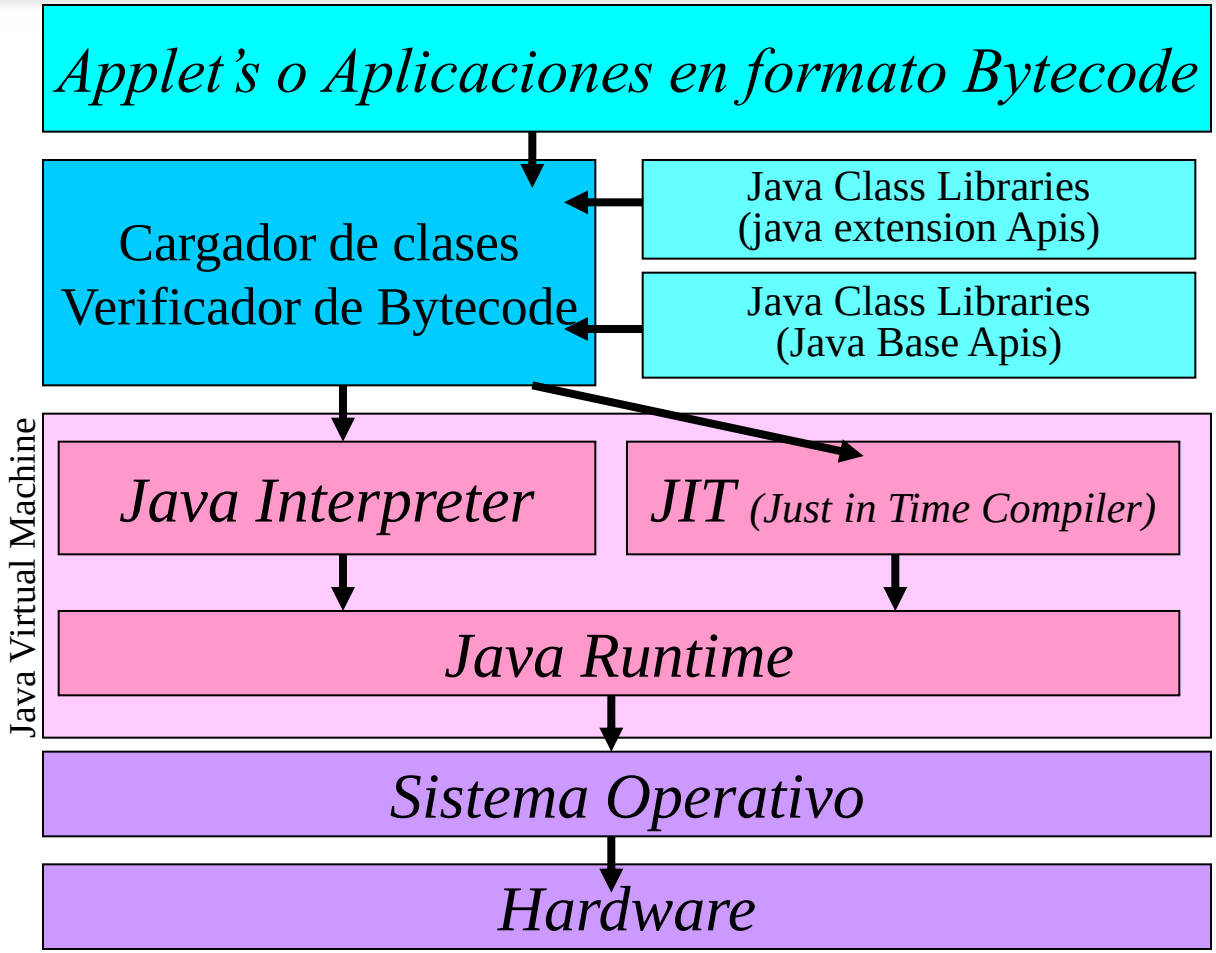
- **Eclipse**
- Netbeans
- J Builder
- Visual Café
- Java Workshop
- Visual Age
- J++




```
javac fichero.java
```

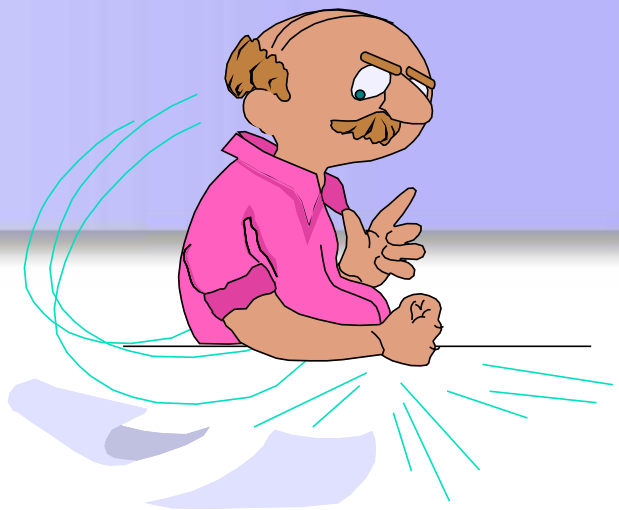
Arquitectura de ejecución

Fase III: Cargar
Fase IV: Verificar
Fase V: Ejecutar




```
java fichero
```

¿Qué puede fallar?



- Compilar → "Syntax Error"
- Cargar → "Class not found Exception"
- Verificar → "Security Exception"
- Ejecutar → "Null Pointer Exception"

Por dónde empezar

- Entorno de desarrollo: JDK

  `http://www.oracle.com/technetwork/Java/index.html`

- Editor: Eclipse

  `http://www.eclipse.org`

- Documentación: Java API

  `http://docs.oracle.com/Javase/7/docs/api/`

Programación de Sistemas

Estructura del lenguaje Java

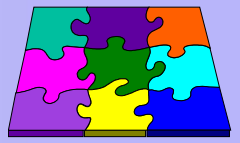
Julio Villena Román
<jvillena@it.uc3m.es>

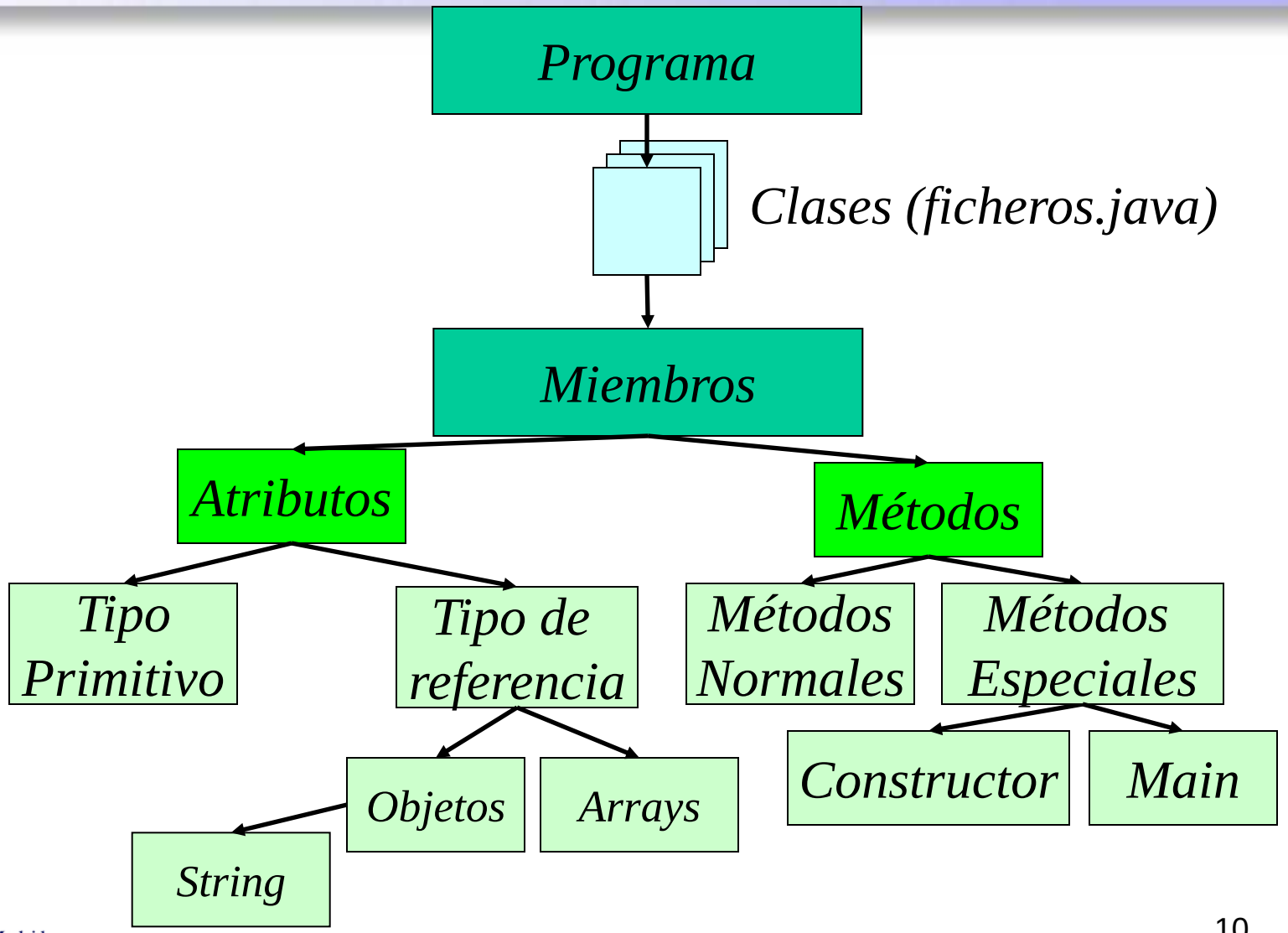
MATERIALES BASADOS EN EL TRABAJO DE DIFERENTES AUTORES:
M.Carmen Fernández Panadero, Natividad Martínez Madrid

Escenario II: Comprender el código

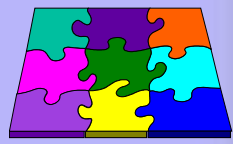
- Te informan de que en tan solo 1h 30 min es la primera reunión de programadores y para entonces tienes que haber revisado el código y tener una idea de cómo funciona la aplicación
- Objetivo: Coger soltura leyendo las estructuras del lenguaje relacionadas con clases, atributos y métodos para con un vistazo rápido comprender el funcionamiento de un programa complejo con varios ficheros
- Plan de trabajo:
  - **Repasar** brevemente la **sintaxis** del lenguaje (identificadores, palabras reservadas, etc.) para ser capaz de distinguir las palabras propias del lenguaje java de la nomenclatura del programa
  - **Reconocer** las construcciones propias del lenguaje relacionadas con **la declaración de clases, atributos** (tipos básicos y de referencia) **y métodos**
  - **Extraer un diagrama de clases** a la vista del código para conocer tipos de objetos desarrolladas, sus características (atributos) y comportamientos (métodos)
  - **Interpretar el método main** (si existe) para ver el orden en el que se desarrolla la aplicación, qué objetos se crean, a cuáles de sus métodos se invoca y en qué orden

Estructura del código





Cómo representamos en Java las clases y los objetos



OO

- Declaración de una clase
- Declaración de un atributo (constantes o variables)
- Declaración de un método
- Creación de un objeto

Java

- Identificadores
- Palabras reservadas
- Tipos primitivos y de referencia en Java

Identificadores

- Sirven para nombrar variables, métodos, clases, objetos y todo lo que el programador necesite identificar
- Comienzan con una letra, un subrayado o un símbolo $
- Distinguen mayúsculas y minúsculas y no tienen longitud máxima
- **Por convenio**:
  - Los nombres de variables, métodos y objetos comienzan por minúscula
  - Los nombres de las clases comienzan por mayúscula
  - Si contienen varias palabras se unen utilizando el convenio *Camel Case* deEsteModo separando palabras con mayúsculas (evitando espacios, subrayados o guiones)

Los identificadores no pueden ser palabras reservadas

Palabras reservadas

Reservadas:

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

Reservadas aunque no se utilizan:

cast	future	generic	inner
operator	outer	rest	var

Comentarios

- Son de 3 tipos:

```
// Comentario de una línea
```

```
/* Comienza comentario de varias líneas
   continúa
   termina */
```

```
/** Comentario para generar documentación javadoc
 @see referencia a la doc de otra clase o método
 @version datos identificativos del nº de versión
 @author Nombre del autor
 @since Fecha desde la que está disponible
 @param Parámetros que recibe el método
 @return Información sobre datos que devuelve
 @throws Excepciones que lanza
 @deprecated Indica que el método es obsoleto
 */
```

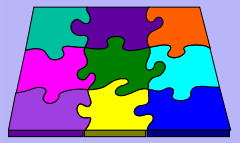
Para clases y métodos

Para clases

Para métodos

opcionales

Declaración de una clase



```
public class Coche{
   //Declaración de atributos
   // (color, velocidad, etc)
   //Declaración de métodos
   // (arrancar, parar, etc.)
}
```

Coche.java

Sintaxis

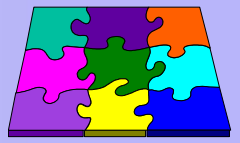
(modificadores) class NombreClase{
//código de la clase
}

El incumplimiento de esta regla se considera en muchos compiladores un error de sintaxis no sólo de estilo

Estilo

- Nombre fichero = nombre clase
- 1ª letra mayúscula
- Sin espacios en blanco ni guiones
- Las palabras se separan con mayúsculas (CamelCase) ej: MiPrimeraClase
- Indentación

Declaración de variables



```
public class Coche{
   //Declaración de atributos
    String color;
     int velocidad;
   //Declaración de métodos
   // (arrancar, parar, etc.)
}
```

Coche.java

Sintaxis

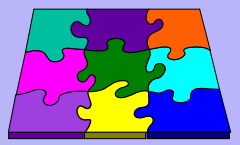
tipo nombre;
tipo nombre1, nombre2, nombre3;
tipo nombre = valor;

Inicialización de la variable a un valor

Estilo

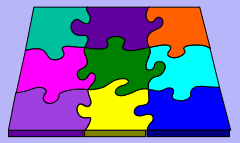
- Nombres intuitivos
- 1ª letra minúscula
- Sin espacios en blanco ni guiones
- Las palabras se separan con mayúsculas: miVariable
- Indentación

Variables



- Son las entidades sobre las que actúan los programas
- ***Declarar*** una variable es decir su nombre y tipo
- Podemos encontrar variables en:
  - como ***miembros*** de la clase (dentro de una clase)
  - como ***variables locales*** (dentro del código de un método)
  - como ***parámetro*** de un método

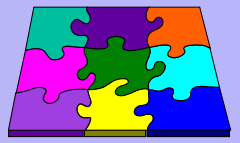
Variables



- Hay que distinguir:
  - variables de ***instancia***
  - variables de ***clase***
  - variables ***locales***
- Las variables
  - pueden inicializarse en la declaración
  - pueden declararse sin inicializar
  - Sin inicialización tienen un ***valor por defecto*** salvo las automáticas
- ***Constantes*** (variables que no se pueden modificar):
  - usar la palabra clave ***final***
  - inicializarla obligatoriamente en la declaración

***Valores por defecto*:**
números = 0
booleanos = false
referencias = null

Ámbito



- Es el bloque de código dentro del cual una variable es accesible
- ***Variable de instancia o de clase*** es accesible dentro de las {} de la clase que la contiene, y para el resto de las clases según los permisos establecidos por los modificadores:
  - private
  - protected
  - public
  - friendly
- ***Local***: su ámbito queda fijado por {} del método en que se encuentra
- ***Parámetro***: su ámbito queda fijado por {} del método en que se encuentra

Tipos básicos en Java

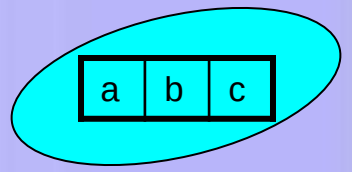
- Todas las variables de Java son de un ***tipo*** de datos
- El tipo de la variable determina:
  - Los **valores** que puede tomar
  - Las **operaciones** que se pueden realizar
- Vamos a estudiar
  - ***Tipos primitivos***
  - ***Tipos de referencia (objetos y arrays)***

Tipos primitivos

4 tipos primitivos básicos

type	literal	num of bits	`double`	`float`	`long`	`int`	`short`	`byte`	`char`
Real	`double`	64-bits	X						
	`float`	32-bits	X	X					
Entero	`long`	64-bits	X	X	X				
	`int`	32 bits	X	X	X	X			
	`short`	16 bits	X	X	X	X	X		
	`byte`	8 bits	X	X	X	X	X	X	
Caracter	`char`	Unicode (16 bits)	X	X	X	X			X
Booleano	`boolean`	1 bit							

Strings (Cadenas)
Declaración, concatenación



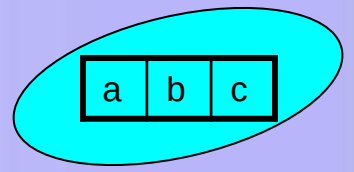
- Son secuencias de caracteres implementadas en la clase **`String`** (dentro del paquete **`java.lang`**)
- Creación de Strings

```
String vacio = new String();
String vacio = "";
String mensaje = "hola"
String repeticion = mensaje;
```

- Concatenación de cadenas
  - La concatenación de cadenas se hace utilizando el operador sobrecargado **+**

```
"este" + "aquel"          // Genera "esteaquel"
"abc"+ 5                  // Genera "abc5"
"a" + "b" + "c"           // Genera "abc"
"a" + 1 + 2               // Genera "a12"
1 + 2 + "a"               // Genera "3a"
1 + (2 + "a")             // Genera "12a"
```

Strings
Comparación



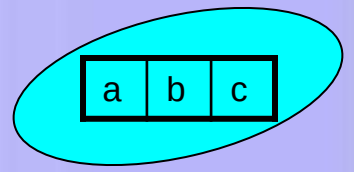
- Sobre Strings **no funcionan** los operadores relacionales (<, >, <=, <=) y de igualdad (==, !=)
  - **Estos operadores comparan objetos y no el contenido**
- En su sustitución, existen **métodos de comparación** en la clase **`String`**
  - Método **`equals`**

```
ladoIzdo.equals(ladoDcho)
    • true, si ladoIzdo y ladoDcho son iguales
```

  - Método **`compareTo`**

```
ladoIzdo.compareTo(ladoDcho)
    • entero negativo, si ladoIzdo es menor que ladoDcho
    • 0, si ladoIzdo es igual que ladoDcho
    • entero positvo, si ladoIzdo es mayor que ladoDcho
```

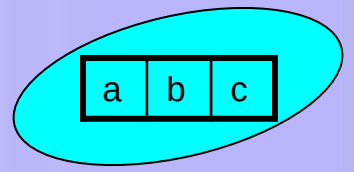
Strings
Métodos útiles del tipo `String`



- Longitud de un objeto **`String`**
  - Usar método **`length()`**
  - Como **`length()`** es un método, es necesario usar paréntesis
- Acceso a los caracteres individuales de un **`String`**
  - Usar método **`charAt()`**, con la posición
    - La primera posición es la posición **`0`**
- Subcadenas
  - Usar método **`substring()`**
    - Devuelve la referencia a un nuevo String.
    - La llamada se realiza especificando la posición de inicio y la primera posición no incluida

```
String saludo= "hola";
int lon= saludo.length();              // lon es 4
char ch = saludo.charAt(1);            // ch es 'o'
String sub= saludo.substring(2,4);     // sub es "la"
```

Strings
Conversión entre cadenas y tipos primitivos



- Para ello se utilizan las llamadas clases envoltorio (wrapper) que se encuentran en **`java.lang`**
  - Se llaman así porque "envuelven" a los tipos primitivos
  - **`Integer, Double, Float, Double, Character, …`**
  - Conversión a **`String`**
    - Métodos **`toString()`**, **`doubleValue()`**,... sin argumentos
  - Conversión de **`String`** al valor del tipo primitivo
    - Método **`parseInt()`**, **`parseFloat()`**,...
  - Conversión de **`String`** un objeto de la clase envoltorio
    - **`valueOf()`**,… con argumento
  - Conversión de un objeto de la clase envoltorio al valor del tipo primitivo
    - **`doubleValue()`**, **`intValue()`** ,... sin argumento

```
System.out.println(Integer.toString(55,2));
int x = Integer.parseInt("75");
Double y= Double.valueOf("3.14").doubleValue();
```

Constantes definidas por el usuario

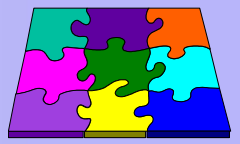
- Son valores invariantes de los tipos básicos (primitivos más el **`String`**)
- Se definen con los modificadores **`final`** (y a veces **`static`**)
  - **`static`**: variable global o de clase. Indica que se almacena sólo una vez y los objetos pueden acceder a ella usando NombreClase.nombreVariable.
  - **`final`**: su valor no se altera
  - Pueden ser **`public`**, **`private`** o **`protected`**
    - Dependiendo de la accesibilidad que se desee
  - ***Estilo:*** Suelen aparecer con todos los caracteres en **`MAYÚSCULAS`**

```
class Circunferencia {
  private static final float PI= 3.14159;
  private float radio;
  private float area;
  public Circunferencia (float radio) {
        area= 2 * PI * radio;
  }//constructor
} //class Circunferencia
```

Tipos de referencia

- Su valor es una ***referencia*** (puntero) al valor representado por la variable
- Son tipos de referencia:
  - Las matrices (arrays)
  - Las clases
  - Los interfaces

Los objetos como atributos
Declaración de objetos



```
public class Coche{
   //Declaración de atributos
    String color;
    int velocidad;
    Equipamiento equipamientoSerie;
   //Declaración de métodos
   // (arrancar, parar, etc.)
}
```

Coche.java

Sintaxis

NombreClase nombre;
NombreClase nombre1, nombre2;
NombreClase nombre = new Equipamiento();

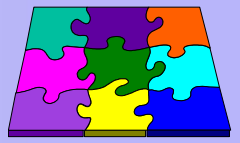
Estilo

•Recordad que las clases (tipo) se nombran con mayúscula y los identificadores (nombreObjeto) con minúscula

Declaración de un objeto
similar a variables, donde antes poníamos el tipo ahora ponemos el nombre de la clase

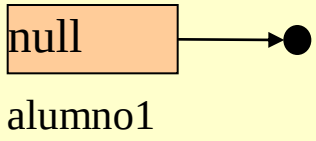
Creación de un objeto
¡¡¡Las variables se inicializan
Los objetos se crean!!!

Objetos
Declaración, creación, inicialización

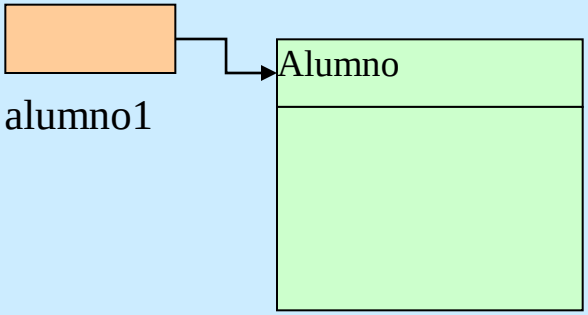


- El objeto se crea con la palabra reservada **`new`** y una llamada al constructor
- Una vez está creado el objeto, se asigna la dirección de memoria en la que se encuentra a la referencia que se había declarado anteriormente

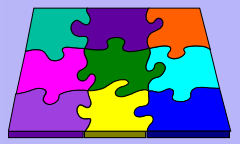
```
Alumno alumno1;
```




```
alumno1 = new Alumno();
```

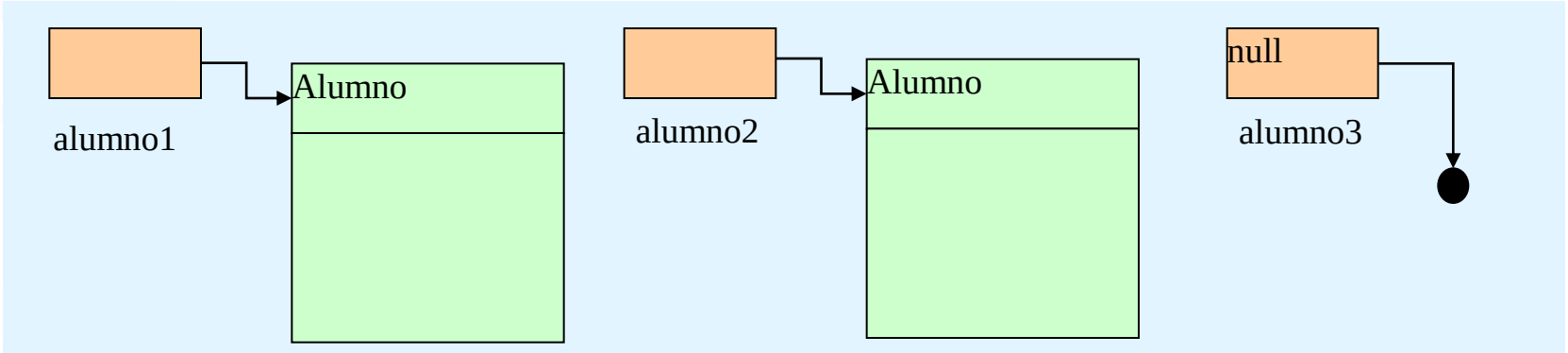


Objetos
Referencia null



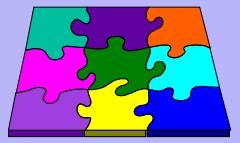
- Una referencia a un objeto puede no tener asignada ninguna instancia
  - Tiene asociado el valor especial **`null`**
- Ejemplo:

```
Alumno alumno1;                  //valen null por defecto
Alumno alumno2;
Alumno alumno3;

alumno1 = new Alumno();          // vale /= null
alumno2 = new Alumno();          // vale /= null
alumno3 = null;                  // vale null por asignación
```



Objetos

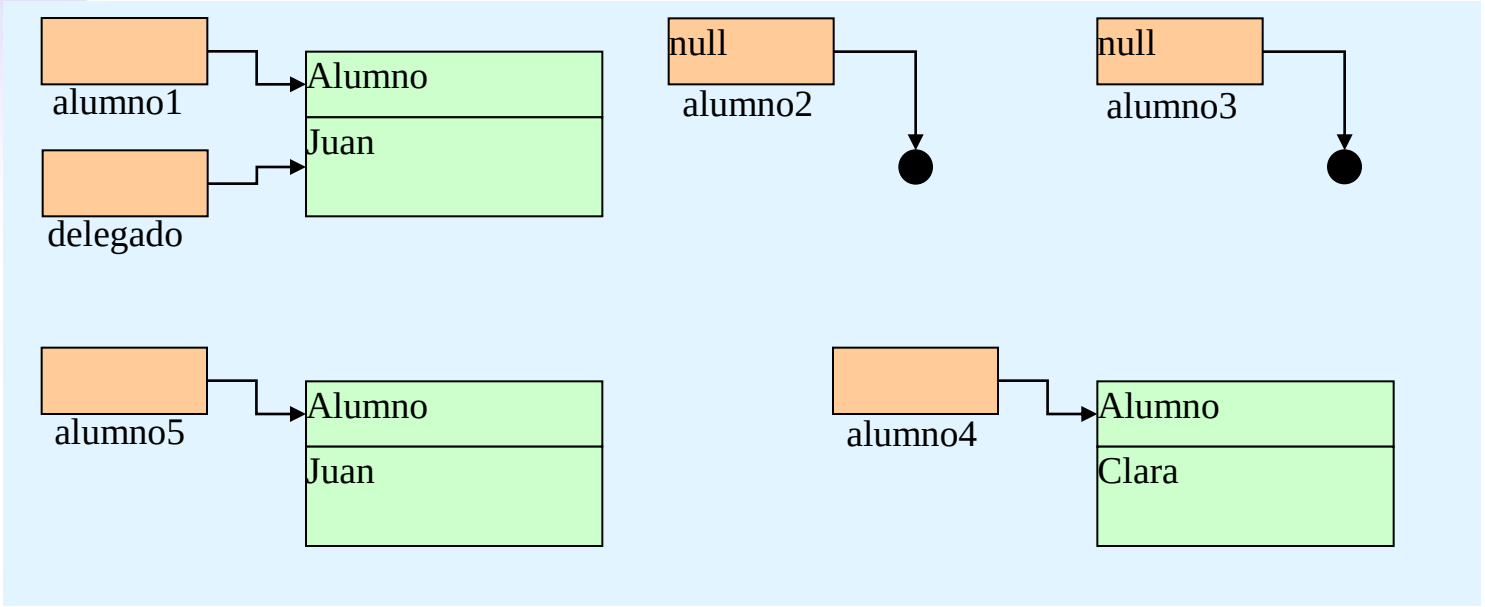
Alias



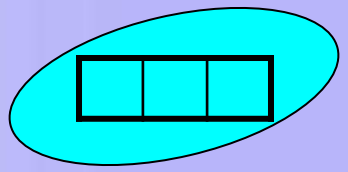
- Un objeto puede tener varias referencias, que se conocen como alias

```
Alumno delegado;
delegado = alumno1;
```

- ¿Qué resultados darían las comparaciones de las distintas referencias en la figura?

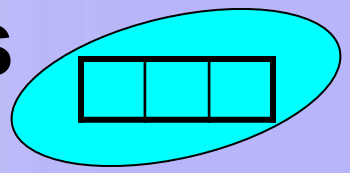


Arrays (Matrices)
¿Qué son?



- Colección de entidades del **mismo tipo** almacenadas en una unidad
- El operador **indexación [ ]** permite el acceso a cada elemento del array
- El atributo **length** nos permite saber el nº de elementos que contiene el array (no confundir con el método `length()` de la clase String)
- Rango del Array
  - De `0` hasta `length - 1`
  - **Cuidado** no sobrepasar la longitud máxima
    - **Surge la exepción** `IndexOutOfBoundsException`

Los Arrays como atributos
Declaración de Arrays



```
public class Coche{
   // Declaración de atributos
   String equipamiento[] = new String [10];
   // ...
}
```

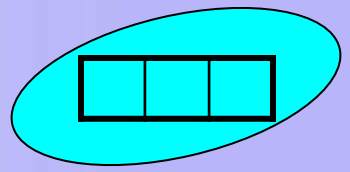
Dos formas de declarar un array

tipo nombreArray[];
tipo[] nombreArray;
tipo nombreArray[] = *new tipo*[numPosiciones];

Creación de un Array
¡¡¡Al crear un Array hay que especificar su capacidad!!!

Creación de un Array
¡¡¡Las variables se inicializan Los Arrays (al igual que los objetos) se crean!!!

Arrays
Declaración, Creación, Inicialización



- ***Declaración***: Consiste en asignar un **identificador** al array y decir de qué **tipo** son los elementos que va a almacenar.

```
tipo nombreArray[];
tipo []nombreArray;
```

  - Se puede hacer de 2 formas
  - Después de la declaración aún no se ha asignado memoria para almacenar el array no podemos acceder a su contenido

Valores por defecto:
int, short, long = 0
float, double = 0.0
booleanos = false
String = null
Object = null

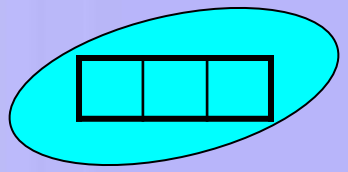
- ***Creación***: Consiste en reservar espacio en memoria para el array
  - Es necesario utilizar **new** y especificar **tamaño** del array

```
nombreArray[] = new tipo[numPosiciones];
```

  - Una vez creado el array sus elementos tienen los valores por defecto hasta que el array sea inicializado

Arrays

Declaración, Creación, **Inicialización**



- ***Inicialización***: Consiste en **dar valores** a los distintos elementos del array podemos hacerlo de varias formas:
  - Elemento a elemento

```
nombreArray[0] = elemento0;
nombreArray[1] = elemento1;
...
```

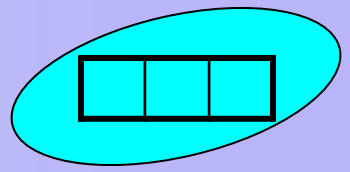
  - Mediante un bucle

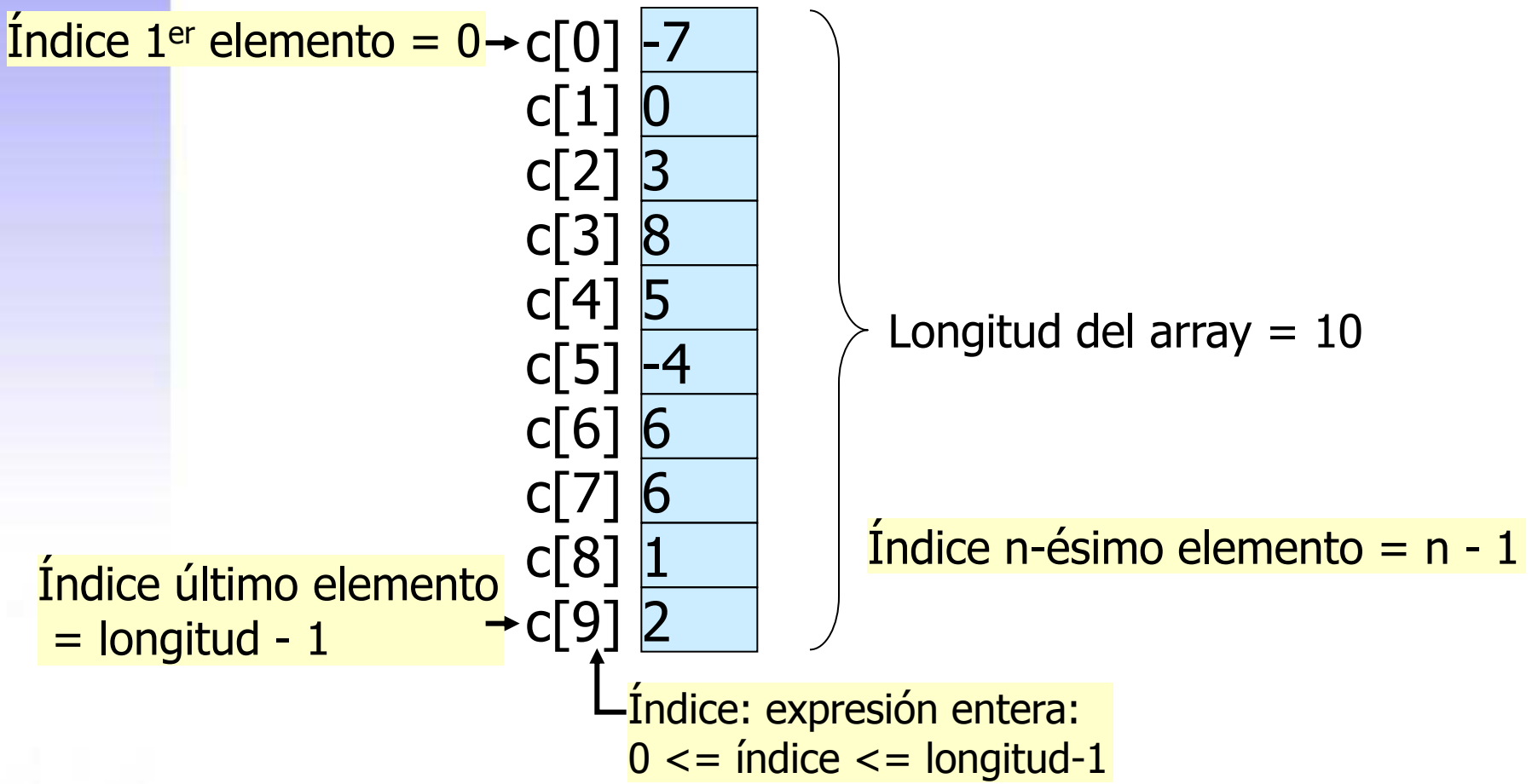
```
for(int i = 0; i < nombreArray.length; i++) {
   nombreArray[i] = elemento-i;
}
```

  - Por asignación directa

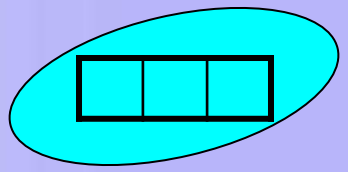
```
nombreArray = {elem1, elem2, elem3, ...};
```

Arrays




Índice 1er elemento = 0
c[0] -7
c[1] 0
c[2] 3
c[3] 8
c[4] 5
c[5] -4
c[6] 6
c[7] 6
c[8] 1
c[9] 2
Longitud del array = 10
Índice n-ésimo elemento = n - 1
Índice último elemento = longitud - 1
Índice: expresión entera:
0 <= índice <= longitud-1

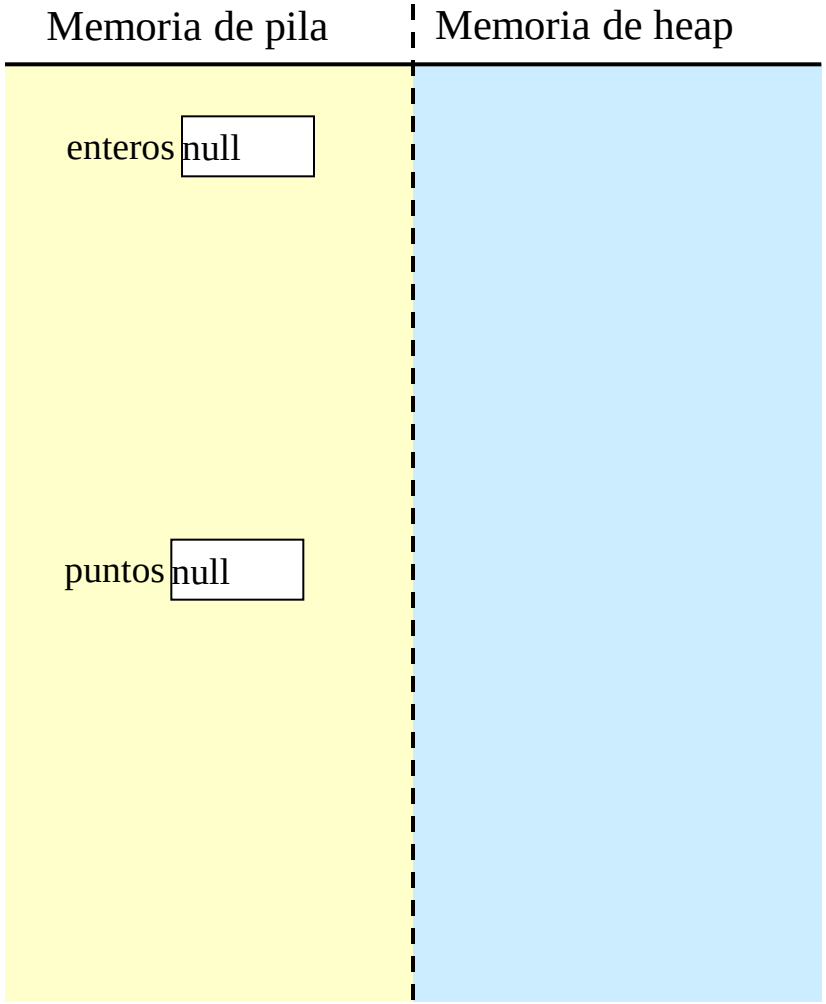
Arrays
Uso de la memoria en la declaración de arrays



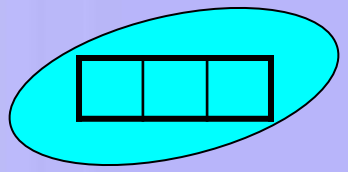
```
int[] enteros;
```

```
Punto[] puntos;
```

```
class Punto {
  int x;
  int y;
  Punto (int x, int y){
    this.x = x;
    this.y = y;
  }
}
```



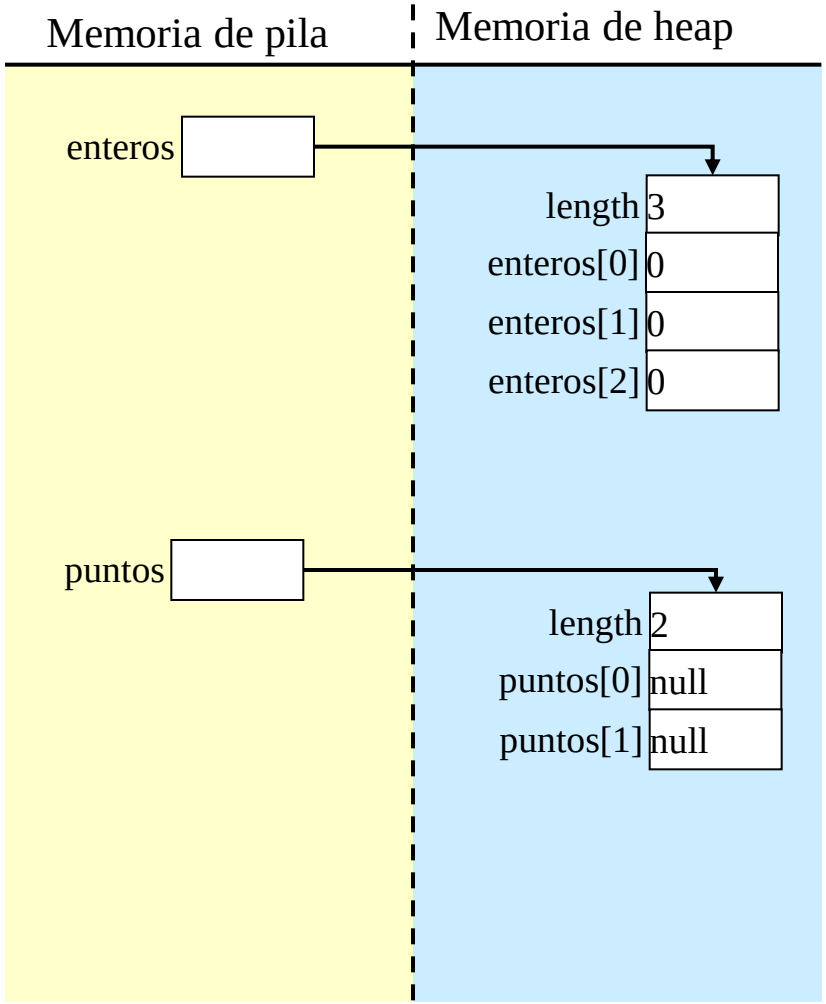
Arrays
Uso de la memoria en la creación de arrays



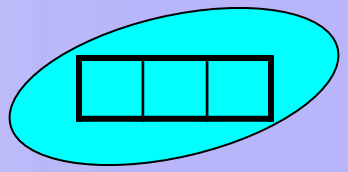
```
enteros = new int[3];
```

```
puntos = new Punto[2];
```

¡Cuidado! No es una llamada al constructor

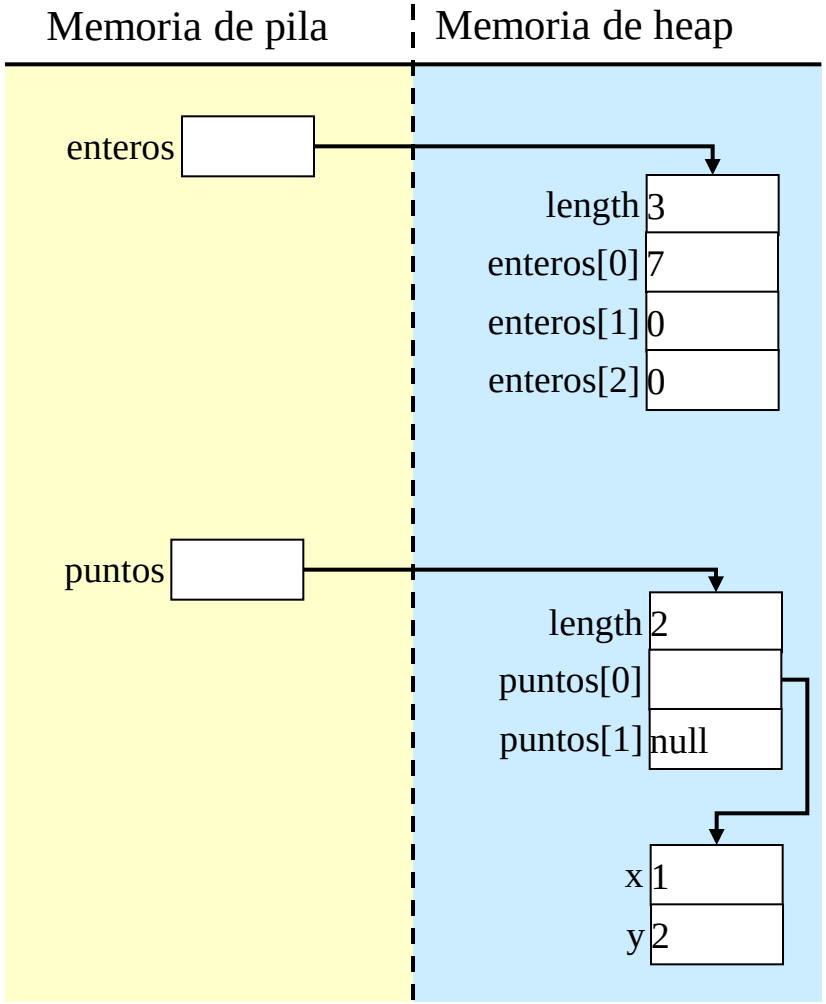


Arrays
Uso de la memoria en la inicialización de arrays

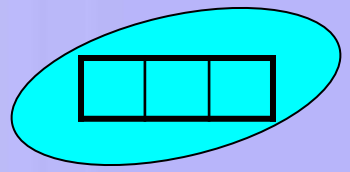


```
enteros[0] = 7;
```

```
puntos[0] = new Punto(1,2);
```



Arrays (Ejemplos)
Declaración, Creación, Inicialización



Arrays de tipos primitivos

```
int a[];              //Declara
a = new int[3]  //Crea
a[0]=1;             //Inicializa
a[1]=2;
a[2]=3;
```

```
int a[] = new int[3]  //Declara y Crea
a[0]=1;                    //Inicialización
a[1]=2;
a[2]=3;
```

```
int a[] = new int[3]  //Declara y crea
for(int i=0; i<a.length;i++){ //inicializa
    a[i]=i+1;
}
```

```
int a[] = {1, 2, 3};  //Declaración, creación Inicialización
```

Arrays de objetos (Tipos de referencia)

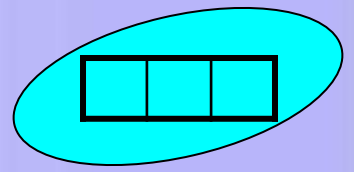
```
MiClase a[];      //Declara
a = new MiClase[3]  //Crea
a[0]=new MiClase(param1);
a[1]=new MiClase(param2);
a[2]=new MiClase(param3);
```

```
MiClase a[] = new MiClase[3]
//inicializa
a[0]=new MiClase(param1);
a[1]=new MiClase(param2);
a[2]=new MiClase(param3);
```

```
MiClase a[] = new MiClase[3]
//inicializa
for(int i=0; i<a.length;i++){
    a[i]=new MiClase(param-i);
}
```

```
MiClase[] a = {new MiClase(param1), new MiClase(param2), new MiClase(param3)};
```

Arrays (Errores frecuentes):
Declaración, creación, inicialización



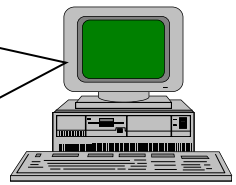
```
public class EjemplosMatrices{
    public static void main(String args[]){
       double miMatriz[];
      System.out.println(miMatriz[0]);
    }
}
```

MAL

compilar

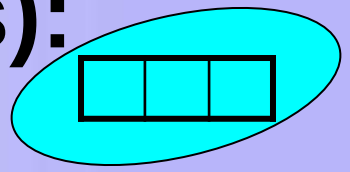
Falla la compilación

variable miMatriz may not have been initialized

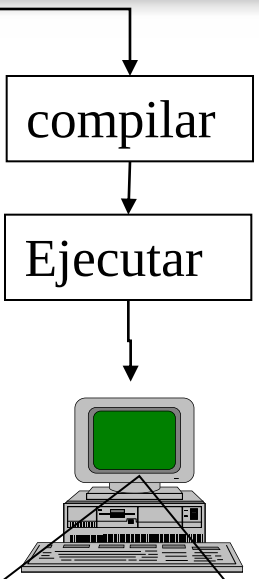


Cuando la matriz sólo ha sido ***declarada*** no podemos acceder a sus elementos. El programa no compilaría y daría un ***error***

Arrays (Errores frecuentes):
Declaración, creación, inicialización



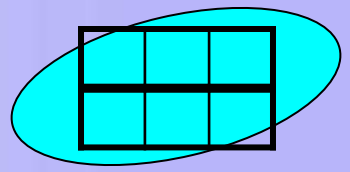
```
public class EjemplosMatrices2{
    public static void main(String args[]){
      int miMatrizDeEnteros[] = new int[10];
      float miMatrizDeReales[]= new float[10];
      boolean miMatrizDeBooleanos[] = new boolean[10];
      char miMatrizDeCaracteres[] = new char[10];
      String miMatrizDeStrings[] = new String[10];
      Object miMatrizDeObjetos[] = new Object[10];
      System.out.println("Entero por defecto: " + miMatrizDeEnteros[0]);
      System.out.println("Real por defecto: " + miMatrizDeReales[0]);
      System.out.println("Booleano por defecto: " + miMatrizDeBooleanos[0]);
      System.out.println("Carácter por defecto: " + miMatrizDeCaracteres[0]);
      System.out.println("String por defecto: " + miMatrizDeStrings[0]);
      System.out.println("Objeto por defecto: " + miMatrizDeObjetos[0]);
    }
}
```



Entero por defecto: 0
Real por defecto: 0.0
Booleano por defecto: false
Carácter por defecto:
String por defecto: null
Objeto por defecto: null

Cuando la matriz sólo ha sido ***declarada y creada*** pero ***no inicializada*** podemos acceder a sus elementos pero estos tienen su ***valor por defecto***

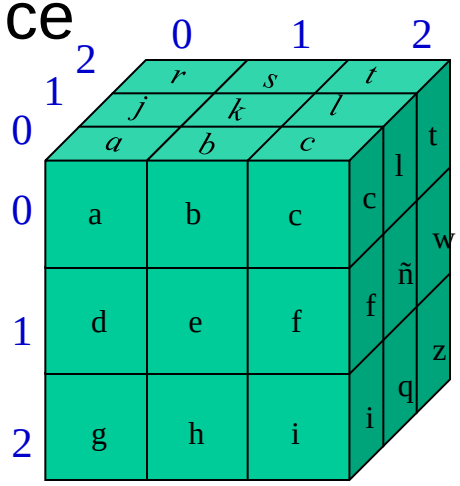
Arrays Multidimensionales



- Es un array en el que el acceso a los elementos se hace utilizando más de un índice

	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I

a[0][2]='C'

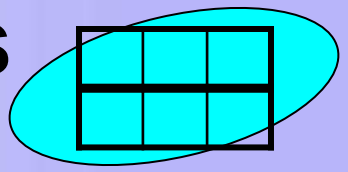


a[0][2][1]='l'

```
char a[][];                 //Declara
a = new char[3][3]          //Crea
a[0][0]='A';                //Inicializa
...
```

```
char a[][][];               //Declara
a = new char[3][3][3]       //Crea
a[0][0][0]='a'
...
```

Arrays Multidimensionales
Ejemplos



Declarar y crear directamente

```
//Declaracion y Creacion
String [][]miMatriz = new String[3][4]
```

null	null	null	null
null	null	null	null
null	null	null	null

Declarar y crear por pasos

```
int [][] miMatriz ;                    // Declarar el array
miMatriz = new int[numFilas][]; // Crear el array de referencia para las filas
for(int i=0; i<numFilas; i++){     // Reservar memoria para las filas
    miMatriz[i]= new int[numColumnas];
}
```

Otros ejemplos

```
// Matriz 3x3 inicializada a 0
int [][] a= new int[3][3];
```

0	0	0
0	0	0
0	0	0

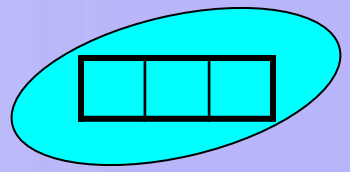
```
int [][] b= {{1, 2, 3},
             {4, 5, 6}};
```

1	2	3
4	5	6

```
int [][] c = new[3][];
c[0] = new int[5];
c[1] = new int[4];
c[2] = new int[3];
```

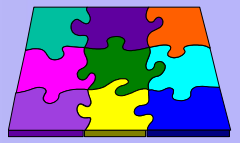
0	0	0	0	0
0	0	0	0	
0	0	0		

Arrays
Actividad para casa



- Escribe un programa que multiplique dos matrices bidimensionales

Declaración de métodos



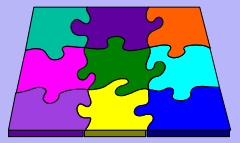
```
public class Coche{
  //Declaración de atributos
   private String color;
   private int velocidad;
  //Declaración de métodos
   public void arrancar(){
       //código para el metodo arrancar
   }
   public void avanzar(int velocidad){
       //código para el método avanzar
   }
   public String queColor(){
       //código para averiguar el color
     return color;
   }
}
```

Coche.java

Estilo

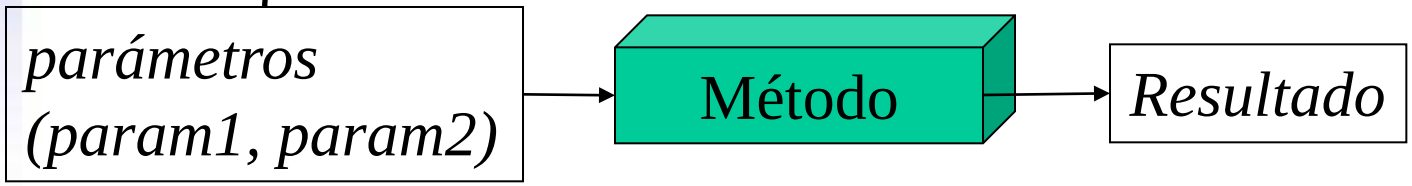
- Nombres intuitivos
- 1ª letra minúscula
- Sin espacios en blanco ni guiones
- Las palabras se separan con mayúsculas miMetodo()
- Indentación

Declaración de métodos



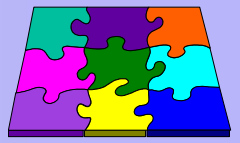
```
public class Coche{
   //...
    public void avanzar(int velocidad){
        //codigo para el método avanzar
    }
   //...
}
```

Coche.java



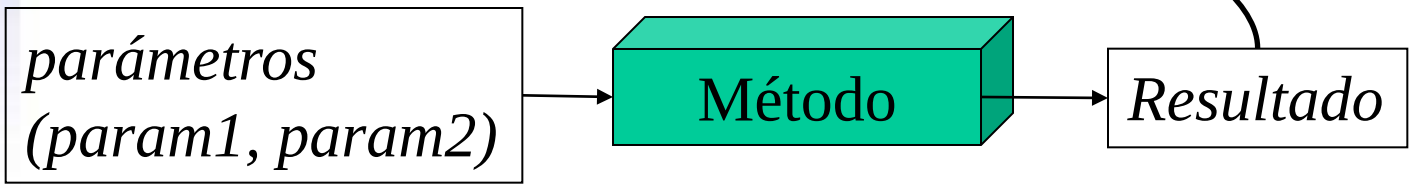
(modificadores) tipoRetorno NombreMetodo(tipo1 param1, tipo2 param2) {
//código del método
return expresion; //cuando el tipoRetorno es void no es necesario
}

Declaración de métodos



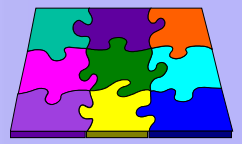
```
public class Coche{
   //...
    public String queColor(){
        //codigo para averiguar el color
      return color;
    }
   //...
}
```

Coche.java



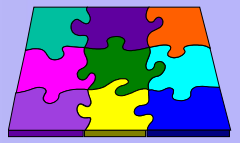

```
(modificadores) tipoRetorno NombreMetodo(tipo1 param1, tipo2 param2) {
        //código del método
        return expresion;
}
```

Declaración de métodos



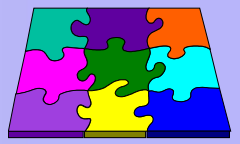
- Los métodos
  - tienen 0, 1 o más ***argumentos*** (parámetros)
  - definen el ***tipo de resultado*** en su declaración (salvo los constructores)
  - pueden tener ***variables locales***. Estas variables **no** se inician por defecto
- En el cuerpo de un método no se pueden declarar otros métodos
- Si el método devuelve un resultado la última sentencia **a ejecutar** debe ser un ***return***

Métodos constructores



- Cuando se crea un objeto sus miembros se ***inicializan*** con un método constructor
- Los constructores:
  - llevan el ***mismo nombre*** que la clase
  - ***No*** tienen ***tipo*** de resultado
- Conviene que haya al menos 1 constructor
- Pueden existir varios que se distinguirán por los parámetros que aceptan (***sobrecarga***)
- Si no existen se crea un ***constructor por defecto*** que inicializa las variables a su valor por defecto
- Si la clase tiene algún constructor el constructor por defecto deja de existir pero el programador puede crear un constructor sin parámetros con la misma función que el constructor por defecto

El método principal (main)



- Es el método que busca el interprete para ejecutar en ***primer*** lugar.
- Los parámetros del main ***(String args[ ])*** representan un array de Strings que guarda los argumentos que escribimos en la línea de comandos al ejecutar el programa.

```
java HolaMundo arg1 arg2 ...
```

- ***void*** indica que no devuelve ningún tipo de datos
- ***static*** indica que este método se refiere a toda la clase, es decir no hay un método main por cada objeto

Programación de Sistemas

Java imperativo

Julio Villena Román
<jvillena@it.uc3m.es>

MATERIALES BASADOS EN EL TRABAJO DE DIFERENTES AUTORES:
M.Carmen Fernández Panadero, Natividad Martínez Madrid

Escenario III: Implementar un método

- Una vez terminada la reunión de programadores y para probar tu pericia antes de integrarte en el equipo, tu jefe decide encargarte la implementación de varios métodos sencillos. Al ser tu primera tarea se trata de métodos que trabajan de forma independiente (no invocan a otros atributos y métodos).
- Objetivo:
  - Ser capaz de descomponer un problema hasta identificar los pasos básicos para su resolución (**diseño y representación de algoritmos**)
  - Utilizar las estructuras básicas de un lenguaje de programación, como variables, operadores y sentencias de control de flujo (bucles, condicionales) para **implementar un algoritmo**
- Plan de trabajo:
  - Entrenarme en el **diseño de algoritmos y su representación**. Descomponer pequeños problemas en pasos para su resolución **sin utilizar código**.
  - Memorizar la sintaxis de java en lo referente a (**operadores, bucles y condicionales**)
  - Entrenarme en usar **java para implementar algoritmos** previamente diseñados
  - Coger **soltura y velocidad implementando problemas típicos** (Ej en un array: recorrerlo, buscar un elemento, intercambiar elementos en dos posiciones, ordenar)

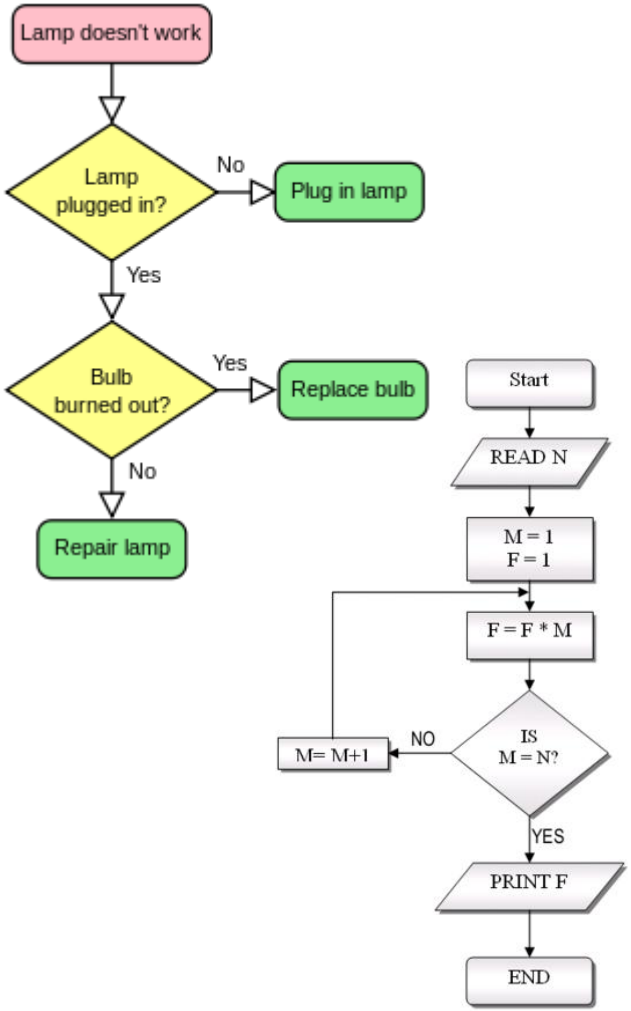
Fase I: Pensar el algoritmo

¿Qué herramientas tenemos para representar algoritmos?

- Una vez pensado el algoritmo que queremos implementar tenemos que representarlo:
  - Pseudocódigo
  - Diagramas de flujo (flowchart)
    - Las **figuras**: representan sentencias
    - Las **líneas de flujo**: representan orden en el que se ejecutan.

Fase I: Pensar el algoritmo

Flowchart vs Pseudocódigo




```
To play "One Potato, Two Potato":
Gather all players in a circle
Players put both fists in the circle
Choose a player to be the counter
The counter begins chanting
He repeats until one fist is left:
[
The counter repeats 8 times:
  [Hit one fist
   If 1-3 or 5-7 say count + "potato"
   If count is 4 say "Four!"
   If count is 8:
    [Say "More!":
     Current fist is taken out
     Restart chant on next fist]
   If count ≠ 8 add 1 to count]
if there is only one fist left:
   that player is "it"
] End
```

Fase II: Implementar el algoritmo

¿Qué código podemos utilizar dentro de un método?

- Variables
- Operadores
  - Por tipo
    - Asignación
    - Aritméticos
    - Relacionales
    - Lógicos
    - Condicional
  - Por número de operandos
    - Unarios
    - Binarios
- Operaciones con objetos (no en este escenario)
  - Creación de objetos
  - invocación de sus atributos o métodos
- Estructuras de control de flujo (pueden apilarse y anidarse)
  - Secuencia
  - Iteración (bucles)
    - For
    - While
    - Do-while
  - Selección (condicionales)
    - If
    - If-else
    - Switch
- Ruptura de flujo
    - Break
    - Continue
    - Excepciones (no en este escenario)

Operadores

- Por **número** de operandos
  - Unarios (un solo operando ej: ++, --)
  - Binarios (dos operandos ej: &&, %)
- Por **tipo** de operador
  - De asignación (=)
  - Aritméticos (+, -, *, /, %)
  - Relacionales (>, >=, <, <=, ==, !=)
  - Lógicos (&&, ll, !)
  - Operador condicional (`condicion?sentencia1:sentencia2`)

```
System.out.println( notaEstudiante >= 5 ? "Aprobado" : "Suspenso" );
```

Operadores
Notas a recordar

- Unarios
  - i++ (primero evalúa luego incrementa)
  - ++i (primero incrementa y luego evalúa)
  - Ej si i=3
    - i++ valdría 3
    - ++i valdría 4
- Binarios (se pueden abreviar)
  - x+=3 equivale a x= x+3
- Asignación vs. comparación
  - El operador = asigna
    - Ej. `var = 5`, asigna `5` a `var`
  - El operador == compara
    - Ej. `var == 5`, devuelve `true` (después de la asignación anterior)
- El operador condicional es más difícil de entender que un simple if-else conviene evitar su uso

Sentencias de selección
(Condicionales)

- If

```
if(condición) {
    sentencias1;
}
```

- If-else

```
if( condición) {
    sentencias1;
} else {
    sentencias2;
}
```

```
if( condición) {
    sentencias1;
} else if(condicion2) {
    sentencias2;
} else {
    sentencias3;
}
```

- switch

```
switch (expresion ) {
        case valor1:
          sentencias1;
          break;

        case valor2:
          sentencias2;
          break;

        default:
          sentencias3;
        }
```

Sentencias de selección

Notas a recordar para if e if-else

- Indentar bien el código contribuye a su legibilidad
- Las llaves fijan el **ámbito** de todo lo declarado entre ellas
- No poner llaves equivale a ponerlas únicamente en la primera sentencia

```
if (notaEstudiante >= 5)
    System.out.println ("Aprobado");
else
    System.out.println ("Suspenso");
```

Sentencias de selección

Notas a recordar para switch

- Tipos expresion validos: `byte`, `short`, `int`, `long` y `char`, `String`
- Ejemplos:
  - `int num=5; switch(num){}`
  - `char character='z' switch(character){}`
  - `String string="cadena" switch(cadena){}`
- Si no se ponen *breaks* se ejecuta el código de todos los bloques siguientes hasta que se encuentre un `break` o el final del `switch`.
- No hace falta colocar el código asociado a cada case entre llaves {}

Sentencias de iteración
(Bucles)

- For:

```
for( inicialización; condición; actualización) {
   sentencias;
}
```

- While:

```
while( condición) {
   sentencias1;
}
```

- Do-while:

```
do {
   sentencias1;
} while(condición)
```

Sentencias de iteración
(Ejemplos for)

- Ejemplos

```
int i=0;
for (i =0;i<10;)
{ i=i+2;}
```

```
int i=0;
for (i=13;i<10; i++)
{ i=i+2;}
```

```
int i=4;
for (;i<10;)
{ i=i+2;}
```

```
int i=0;
for ( ; ; )
{ i=i+2;}
```

```
int i, suma;
for (i =0, suma=5;i<10;suma+=i)
{ i=i+8;}
```

¿cuántas veces se ejecutan estos bucles?

¿cuál es el valor de i en cada caso al salir del bucle?

Sentencias de iteración
(Ejemplos for)

```
int i=0;
{
```

```
int
for (
{ i=i
```

El que usarás casi siempre
(¡**memorízalo**!)

```
for (int i =0;i<5;i++){
   //sentencias
}
```

Sentencias de iteración

Notas a recordar

- Si en un `for` aparecen varias sentencias en inicialización, condición o actualizacion, se separan con comas.

```
for(i=0, suma=0 ; i<=n; i++, suma+=n) {
   sentencias;
}
```

- Los bucles anidados:
  - Ralentizan la ejecución
  - Se usan para recorrer matrices n-dimensionales (un bucle por dimensión)
- Las sentencias dentro de un `while` pueden no ejecutarse nunca, en un `do-while` se ejecutan al menos una vez
- Evitar bucles infinitos (comprobar siempre condición de terminación)
- Un `for` siempre se puede convertir en un `while` y viceversa

Sentencias de iteración
Comparativa

- Comparativa `for` vs `while` vs `do-while`

	Inic	Act	Condición	Min Ej	Uso
`For`	Sí	Sí	Continuar	0	Alto
`While`	No	No	Continuar	0	Alto
`do while`	No	No	Continuar	1	Bajo

- Inic: Posibilidad de inicialización de variables
- Act: Posibilidad de actualización de variables
- Condición: Indica si la condición es de continuación o salida
- Min Ej: número mínimo de veces que ejecuta el bloque de código
- Uso: Indica el grado de uso de la estructura de control

Sentencias de iteración

Pautas de uso

- Cuándo usar `while` o `for`

	`for`	`while`
Sabemos el número de iteraciones (Ej array)	X	
Se desconoce el número de iteraciones		X
Incremento de variables en cada ciclo	X	
Hay que inicializar variables	X	X

Ej: lectura de fichero con while.

Ej: recorrer arrays con for.

Ruptura de flujo:
Sentencia break

break: cuando se ejecuta dentro de un `while`, `for`, `do/while` o `switch` hace que se abandone la estructura donde aparece.

```
int j=0;
while(j<10){
  j++;
  break;
  System.out.println("Esto nunca se ejecuta");
}
System.out.println("j vale: "+j);
```

El bucle sólo se ejecuta una única vez y se imprime el mensaje "j vale: 1"

Ruptura de flujo:
Sentencia continue

continue: si se ejecuta dentro de un bloque `while`, `for` o `do/while` se salta el resto de las sentencias del bucle y continua con la siguiente iteración

```
int j=0
while(j<10){
  j++;
  continue;
  System.out.println("Esto nunca se ejecuta");
}
```

Nunca se imprime el mensaje "Esto nunca se ejecuta"

Implementar un método:
Fase 1.1: pensar el algoritmo

- **Problema**: Escribir un programa que calcule si un número es primo

```
1  2  3  4  .  .  .  n/2  .  .  .  n
```

- **Fase 1**: **Pensar el algoritmo** (descomponer el problema en pasos)
  - Empezando desde 2, comprobamos si cada número es un divisor entero de n
  - Sólo hace falta repetirlo hasta n/2
  - O hasta que encontremos un divisor entero
  - Para esto utilizaremos un *centinela*
    - Variable booleana a la que asignaremos valores y que nos ayudará en el control del bucle

Implementar un método:

Fase 1.2: Representar el algoritmo

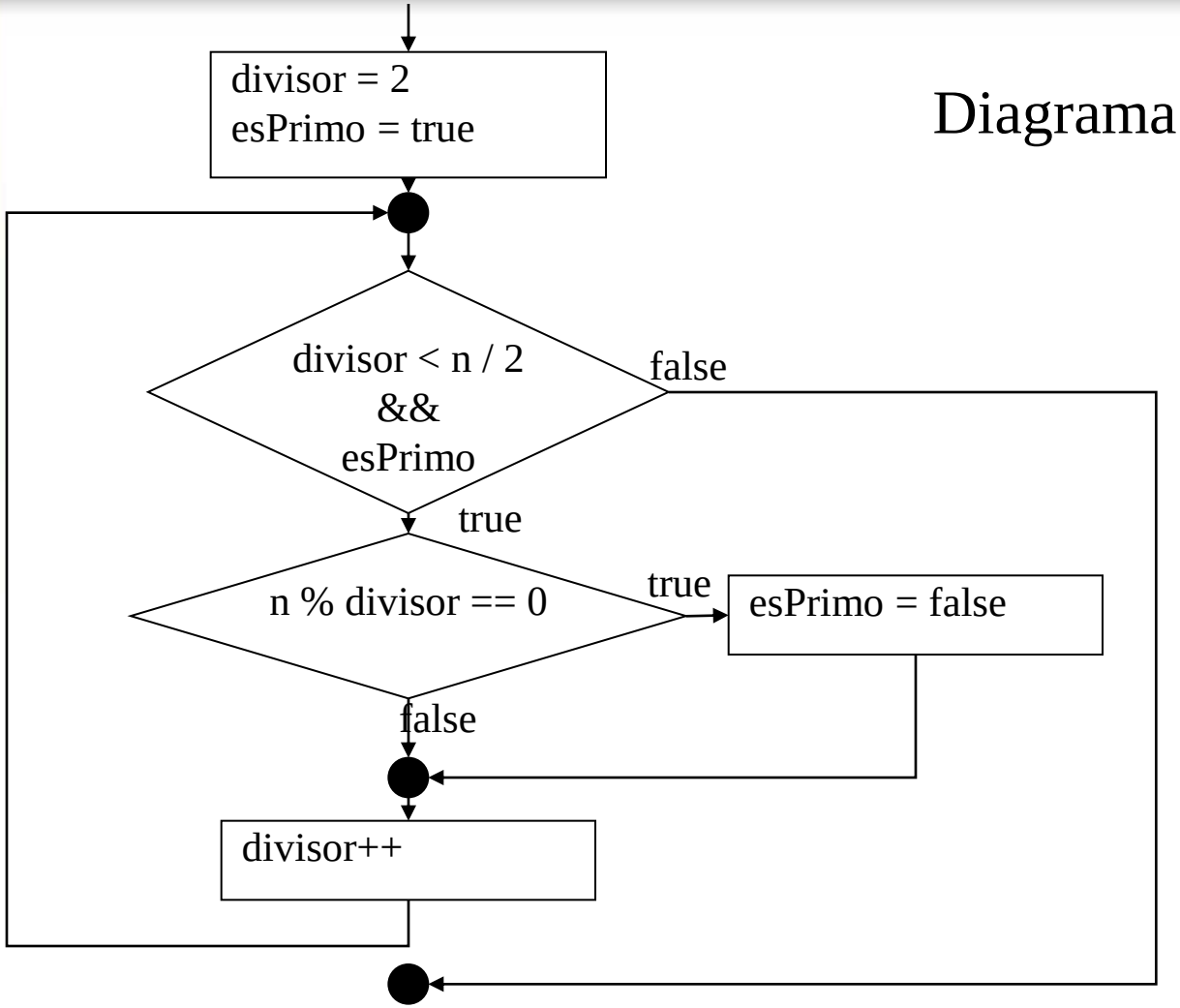


Diagrama de Flujo

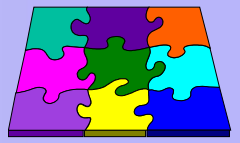
Implementar un método:
Fase 2: Escribir el código

```
public boolean calcularSiEsPrimo (int numero) {

   int divisor = 2;
   boolean esPrimo = true;

   while ((divisor < numero/2) && esPrimo){
     if (numero % divisor == 0)
       esPrimo = false;
     divisor++;
   }

   System.out.print("El numero " +numero);
   if (esPrimo)
     System.out.println(" es primo.");
   else
     System.out.println(" no es primo.");

   return esPrimo;
 }
```

Implementar un método:
Ejemplos: trabajando con arrays

- Para practicar todo lo visto hasta ahora supongamos que tenemos que implementar métodos para:
  - Imprimir un array (practicar bucles)
  - Encontrar un elemento en un array
    - Practicar condicionales y estructuras anidadas
    - Practicar diferencias al comparar
      - Tipos básicos (números, caracteres booleanos)
      - Comparar Strings
      - Comparar Objetos
  - Intercambiar dos elementos en un array (practicar uso de variables auxiliares)
  - Ordenar un array (practicar la copia de elementos entre dos arrays)

Repaso
Resultados del aprendizaje



- Después de este bloque deberías ser capaz de:
  - **Instalar y configurar** un entorno para trabajar con java
  - **Comprender un programa** con varios ficheros y ser capaz de extraer un diagrama de clases y saber desde qué clase empieza a ejecutarse
  - **Identificar estructuras básicas relacionadas con clases y objetos** como las ***declaraciones*** de:
    - Clases
    - Miembros
      - Atributos
        - » De **tipo básico** (primitivos, String)
        - » De **tipo de referencia** (objetos y arrays)
      - Métodos
        - » Método **main**
        - » Métodos **constructores**
        - » Métodos normales
  - **Diseñar e implementar un algoritmo** sencillo en el interior de un método **utilizando operadores y estructuras básicas de control** (bucles y condicionales)