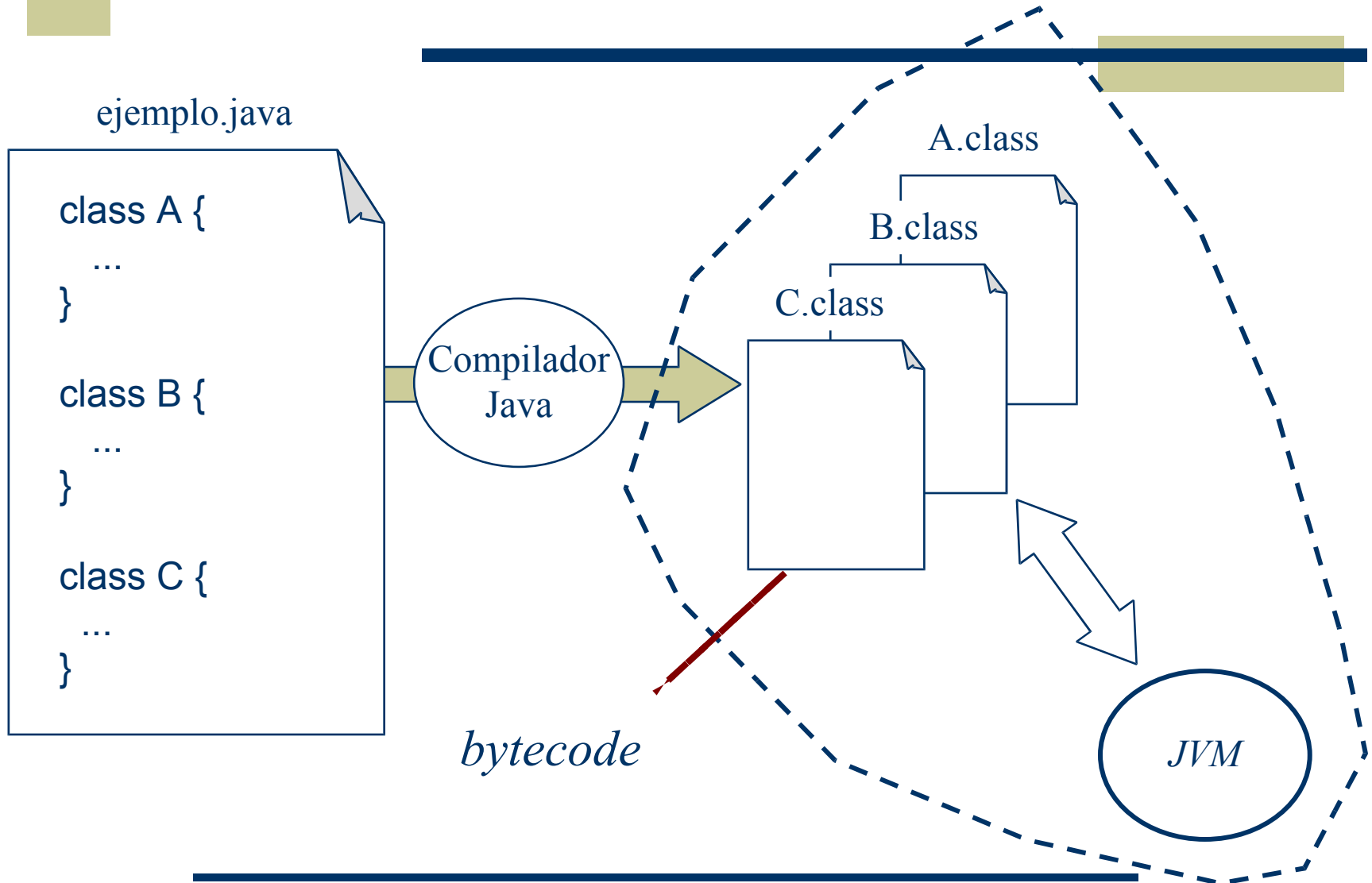




El lenguaje Java

Apéndice A Elementos del lenguaje

Compilación



Proceso de desarrollo

- ◆ Un fichero (.java) por cada clase pública
 - El nombre del fichero es el de la clase
- ◆ Un fichero con una clase con el método *main*
`public static void main(String [] args)`
- ◆ Compilar cada fichero .java
`javac NombreClase.java`
 - produce fichero .class (“ejecutable”)
- ◆ Ejecución con JVM
 - ejecutar clase con método *main* asociado
`java NombreClase`

Aspectos léxicos

- ◆ Espacios en blanco
- ◆ Sentencias separadas por ';'
- ◆ Comentarios
 - // hasta final de línea
 - /* */ más de una línea
 - /** */ *cabeceras javadoc*
- ◆ Sensible a mayúsculas/minúsculas
- ◆ **Convenios de nomenclatura**

Valores y variables

Tipos básicos

byte	1 byte
char	2 bytes (sin signo, caracteres Unicode, incluyen los ASCII)
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
boolean	1 bit (true o false, no compatible con tipos numéricos)

Variables y literales

Declaración, utilización, ámbito léxico, etc. similar a C

Operadores

- ◆ 46 operadores
- ◆ Numéricos
 - +, -, *, /, %, +=, -=, *=, /=, %=, --, ++
- ◆ Lógicos
 - &, |, ^, !, &&, ||
- ◆ Operadores de bits
 - &, |, ^, ~, >>, <<
- ◆ Relacionales
 - Cualquier tipo: ==, !=
 - Tipos numéricos: >, <, >=, <=
- ◆ Condicional
 - If-then-else:(condición)? acción1 : acción2

Precedencia de operadores

() [] .
++ -- ~ !
new
* / %
+ -
>> >>> <<
> >= < <= instanceof
== !=
&
&
&
&&
||
?:
= += *= -= /=

Arrays

- ◆ Declaración

```
int a[ ];
```

- ◆ Reserva de memoria

```
a = new int [3];  
int b[ ] = {34, 23, 9, 1, 18};
```

- ◆ Se puede saber su tamaño

```
a.length; // = 3 (pero no cambiarlo: sólo lectura)
```

- ◆ Asignación de valores

```
a[1] = 419;  
b = a;           // Válido: los arrays no son constantes
```

Arrays (2)

- ◆ Errores

`a[6] = 7;` *// Fuera de rango*

`a = {51, 29, 7};` *// Sólo válido en inicialización*

`int c[5];` *// La dimensión sólo al hacer new*

`char str[] = "hola";` *// Los strings no son arrays*

Arrays multidimensionales

```
float a[ ][ ] = new float [4][3];
float m[ ][ ] = new float [4][ ];
m[0] = new float [2];
m[1] = new float [5];
m[2] = new float [m[1].length];
float x[ ] = {4.5, 8/3, m[2][1]};
m[3] = x; // O cualquier expresión que devuelva un float[ ]
a[0] = m[1]; // Las variables de array no son constantes
a[2, 1] = 7.3 // Error de sintaxis: a[2][1]
```

Condicionales

if (*condición*) *acción*₁ [else *acción*₂]

switch (*expresión*) {

case *valor*₁:

...

break;

...

case *valor*_n:

...

break;

default:

...

}

byte, char, short ó int

tienen que
ser literales

Iteraciones

```
while (condición) {
```

```
    ...
```

```
}
```

```
do {
```

```
    ...
```

```
} while (condición)
```

```
for (inicialización; condición; iteración) {
```

```
    ...
```

```
}
```

Break

```
boolean t = true;
a: {
b:  {
c:   {
        System.out.println ("Antes de break");
        if (t) break b;
        System.out.println ("Esto no se ejecuta");
    }
    System.out.println ("Esto no se ejecuta");
}
System.out.println ("Después de b");
}
```

Continue

```
for (int i = 0; i < 10; i++) {  
    System.out.print( i + " ");  
    if (i % 2 == 0) continue;  
    System.out.println ();  
}  
  
outer: for (int i = 0; i < 10; i++) {  
    for (int j = 0; j < 10; j++) {  
        if (j > i) {  
            System.out.println ();  
            continue outer;  
        }  
        System.out.print ( " " + (i * j));  
    }  
}  
}
```



Bibliotecas de clases fundamentales

Cadenas de caracteres

La clase java.lang.String

- ◆ *String* encapsula cadenas de caracteres y su manipulación
- ◆ *String* ≠ char[]
- ◆ Los caracteres de un *String* no se pueden modificar
- ◆ Constructores
 - String (char[])
 - String (String)
- ◆ Creación de *strings*
 - char[] chars = {'a','b','c'};
 - String s = new String (chars);
 - String s1 = "Hello";
 - String s2 = new String (s1);
- ◆ Literales: Java crea objetos de tipo *String* para los literales

Métodos de la clase *String* (I)

- ◆ Acceso (la notación `str[n]` no existe para *Strings*)
 - `String str = "abcdabc";`
 - `str.length (); // → 7`
 - `str.charAt (4); // → 'a'`
 - `str.substring (3, 5); // → "da"`
 - `str.indexOf ("bc"); // → 1`
 - `str.lastIndexOf ("bc"); // → 5`

Métodos de la clase *String* (II)

- ◆ Manipulación (se devuelve un *String*)
 - `str.concat("xyz"); // → "abcdabcxyz"`
 - `str + "xyz"; // igual (devuelve nuevo String)`
 - `str += "xyz"; // lo que se espera; str = nuevo String.`
 - `str.toUpperCase (); // → "ABCDABC"`
 - `str.replace ('b', 'x'); // → "axcdaxc"`
 - `" Hola que tal ".trim (); // → "Hola que tal"`
 - `"me gusta el futbol".split(" "); // → String[4]`
 - ...

Métodos de la clase *String* (III)

◆ Conversión a *String*:

static String valueOf (<cualquier tipo>)

String.valueOf (2.34567); // → "2.34567"

String.valueOf (34); // → "34"

String.valueOf (new Plano3D ()); // → "Plano3D@1d07be"

(excepto si Plano3D ha redefinido su método "toString()")

◆ Comparación

String s1 = "abc", s2 = new String ("abc"), s3 = "abx";

s1 == s2; // → false

s1.equals (s2); // → true

s1.compareTo (s3); // → -21 < 0

La clase `java.lang.StringBuffer`

- ◆ Los caracteres de un *StringBuffer* sí se pueden modificar.
- ◆ Los objetos de tipo *StringBuffer* gestionan automáticamente su capacidad.
 - Toman una capacidad inicial
 - La incrementan cuando es necesario

◆ Constructores

`StringBuffer()` // Capacidad inicial: 16 caracteres

`StringBuffer(int c)` // Capacidad inicial: c caracteres

`StringBuffer(String s)` // Capacidad inicial: `s.length()` + 16 caracteres

Métodos de *StringBuffer* (I)

- ◆ Acceso (igual que para *String*): `length ()`, `charAt (int)`, ...
- ◆ **Conversión a *String*: `toString ()`**
- ◆ **Modificación de la cadena**
 - `StringBuffer str = new StringBuffer ("abcdef");`
 - `str.setCharAt (2, 'q');` // `str = "abqdef"`
 - `str.append ("ghi");` // `str = "abqdefghi"`
 - `str.insert (3, "xyz");` // `str = "abqxyzdefghi"`
 - `str.insert (6, 1.23);` // `str = "abqxyz1.23defghi"`
 - `str.delete (2, 10);` // `str = "abdefghi"` (versión 1.2)
 - `str.reverse ();` // `str = "ihgfedba"`

Métodos de *StringBuffer* (II)

- ◆ Manipulación de longitud y capacidad
 - `length ()`, `capacity ()`
 - `setLength (int)`, `ensureCapacity (int)`
- ◆ Operador de concatenación: internamente se utiliza un *StringBuffer*
 - `"DNI de " + cliente.nombre + ": " + cliente.dni`
 - `new StringBuffer().append("DNI de ")`
`.append(cliente.nombre).append(": ")`
`.append(cliente.dni).toString()`

Bibliotecas de clases fundamentales

Clases útiles

Funciones matemáticas: La clase java.lang.Math

◆ Constantes:

- Math.PI
- Math.E

◆ Métodos:

sqrt (double), pow (double, double),
random (), abs (double), max (double, double),
round (double), cos (double), sin(double), tan (double),
acos (double), exp (double), log (double), etc.

- (Existen versiones float, long, int para abs, max, min, round)

Otras clases

- ◆ Package java.util:
 - Stack, LinkedList, HashMap, **ArrayList**
 - **Collections**
 - **StringTokenizer**
 - Date
- ◆ Package java.text:
 - DateFormat
 - DecimalFormat
- ◆ Package java.math:
 - BigDecimal
 - BigInteger (precisión y capacidad arbitrarias)
- ◆ La clase java.lang.Runtime:
 - `getRuntime()`
 - `exec(String)`
 - `exit(int)`

Colecciones y ArrayLists

◆ ArrayList:

```
ArrayList al = new ArrayList();  
al.add("casa");  
al.add("mesa");  
al.add(new Integer(1));  
al.size(); // = 3  
al.set(0, "chalet");  
al.get(0); // = (Object)"chalet"
```

◆ Array estático:

```
Object ao[] = new Object[10];  
ao[0] = "casa";  
ao[1] = "mesa";  
ao[2] = new Integer(1);  
ao.length; // = 10  
ao[0] = "chalet";  
ao[0]; // = (Object)"chalet"
```