Programación en Java

Tema 2. El lenguaje de programación Java (Parte 1)

Luis Rodríguez Baena

Universidad Pontificia de Salamanca (campus Madrid)
Facultad de Informática

Elementos del lenguaje (I)

- El juego de caracteres.
  - No utiliza ASCII, sino Unicode de 16 bits.
  - Cada carácter se puede representar por `\uxxxx`.
- Comentarios.
  - `//comentario`. Ignora el texto hasta final de línea.
  - `/*comentario*/`. Multilínea
  - `/**comentario*/`. Multilínea. Se utiliza para la documentación en línea mediante javadoc.
- Identificadores.
  - Utilizan caracteres Unicode.
  - Sensibles a mayúsculas.

Elementos del lenguaje (II)

- Palabras reservadas.

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

  - `const` y `goto` no se pueden utilizar

Tipos de datos

- Datos de referencia y tipos de datos primitivos.
- Datos primitivos

Tipo	Tamaño / formato	Descripción
`boolean`	`true` o `false`	Dato lógico
`char`	Un carácter Unicode de 16 bits	Carácter
`byte`	Entero de 8 bits complemento a 2	Entero corto
`short`	Entero de 16 bits complemento a 2	Entero
`int`	Entero de 32 bits complemento a 2	Entero
`long`	Entero de 64 bits	Entero largo
`float`	Número en coma flotante de 32 bits	Real de simple precisión
`double`	Número en coma flotante de 64 bits	Read doble precisión

Literales (I)

- Literales de referencia (`null`).
- Literales booleanos (`true` y `false`).
- Literales de carácter.
  - Carácter Unicode entre comillas simples.
  - Representación hexadecimal `\uxxxx` donde `xxxx` es el valor hexadecimal del carácter.
  - Secuencias de escape

`\b`	Retroceso	`\t`	Tabulador	`\n`	Nueva línea
`\f`	Salto de página	`\'`	Comillas dobles	`\"`	Comillas simples
`\r`	Retorno de carro	`\\`	Slash invertido	`\ddd`	Carácter en octal

Literales (II)

- Literales enteros.
  - Prefijo `0` (cero) para octal y `0x` para hexadecimal.
  - Sufijo `L` o `l` para enteros largos.
- Literales de coma flotante.
  - `F` o `f` para simple precisión, `D` o `d` para doble precisión.
  - Las constantes serán `float` a no ser que se indique lo contrario.
  - Una constante `double` no se puede asignar directamente a una variable `float` aunque esté dentro de su rango.
- Literales de cadena.
  - Serie de caracteres Unicode separados por comillas dobles.

Variables (I)

- Según su tipo de información:
  - Variables de tipos primitivos.
  - Variables de referencia.
  - El tipo `void`.
- Según su papel en el programa:
  - Variables de clase. Modificador `static`.
  - Variables de instancia.
  - Componentes de un array.
  - Argumentos de métodos.
  - Argumentos de contructuctores.
  - Argumentos de controladores de excepciones.
  - Variables locales.

Variables (II)

- Declaración de variables.
  - *modificador* [static] [final]*tipodato identificador*
  - El *identificador*, por convención, comienza en minúsculas.
- Inicialización de variables.
  - Es necesario inicializar las variables locales.
  - En el resto toman sus valores por omisión.
  - Inicialización en la declaración:

    tipodato identificador expresiónInicialización
- Variables `final`.
  - No cambian su valor (constantes).
  - `final` en tipos primitivos y en tipos de referencia.

Variables (III)

- Visibilidad
  - Variables locales: el módulo donde han sido declaradas.
  - Miembros de una clase: depende del modificador.
    - ✓ Por omisión acceso "de paquete" (amistoso o friendly).
    - ✓ Modificador `public`. Acceso a todas las clases.
    - ✓ Modificador `private`. Acceso a los miembros de la clase.
    - ✓ Modificador `protected`. Accesible por las clases hijas (herencia) y por los miembros de la clase.

Modificador	Clase	Subclase	Paquete	Mundo
`private`	X			
`protected`	X	X	X	
`public`	X	X	X	X
"paquete"	X	X		

Conversiones de tipos

- Conversión *implícita* de un tipo de menos a más precisión.
  - Por ejemplo, de `char` a `int`, de `int` a `float`.
  - Perdida de precisión en algunos casos.
    - ✓ Por ejemplo, de `long` a `double` ya que `double` tiene un rango entero menor.
- Conversión explícita: *cast*.
  - Pérdida de precisión cuando el destino tiene menos dígitos significativos.
- Conversión a cadenas.
  - Siempre es posible la conversión utilizando el operador de concatenación.
  - Representación en cadena del dato u objeto: el método `toString()`.

Clases de envoltura (I)

- Los tipos primitivos tienen clases de envoltura en el paquete `java.lang`.
  - `Boolean`, `Character`, `Byte`, `Short`, `Integer`, `Long`, `Float` y `Double`.
  - Proporcionan constantes y métodos adicionales a los tipos primitivos.
  - Permiten pasar por referencia tipos primitivos.
- Constructores:
  - `Integer i = new Integer(5);`
  - `Double d = new Double("123.30");`

Clases de envoltura (II)

- Algunos métodos
  - `public` *`Tipo tipo`*`Value()`. Devuelve un dato del tipo primitivo definido por *`tipo`*.
    - ✓ `System.out.println(d.intValue());` devuelve el dato primitivo `123`
  - `public static` *`Tipo`* `valueOf(String` *`cad`*`)`. Devuelve el mismo valor que el constructor `new` *`tipo`*`(`*`cad`*`)`.
    - ✓ `d1 = Double.valueOf("129.32"); //d1 = 129.32`
  - `public static toString()`. Devuelve una representación del objeto en forma de cadena.

Clases de envoltura (III)

- Algunos métodos
  - `public int compareTo(`*`Tipo otro`*`)`. Devuelve un valor igual, menor o mayor que 0, dependiendo si el objeto que lo invoca es igual mayor o menor que *`otro`*.
    - ✓ `System.out.println(d.compareTo(d1)); //Escribe -1`
  - `public int compareTo(Object obj)`. Si `obj` es de distinto tipo que el objeto actual devuelve `null`.
  - `public boolean equals(Object obj)`. Devuelve `true` si los objetos son del mismo tipo y envuelven el mismo valor.
    - ✓ Devuelven null si obj no es del mismo tipo que el objeto que lo invoca.

Operadores (I)

Aritméticos unarios	+,-
Aritméticos binarios	+,-,*,/,%
Asignación	=, +=, -=,*=,/=,%=
Concatenación	+, +=
Incrementales	++,-- (antes o después de la expresión)
Relacionales	<,<=,>,>=,==,!=
	`instanceof` (evalúa si una referencia a un objeto es una instancia de una clase o interfaz)
Lógicos	&,|,^, ! (no producen cortocircuito)
Lógicos condicionales	&&,|| (producen cortocircuito)

Operadores (II)

?:	exprLógica?valor1:valor2
De bits	&, |, ^,~
De bits (desplazamiento)	<<,>> (desplaza bits a la derecha, rellenando con el más significativo, el signo, a la izquierda), >>> (desplaza bits a la derecha rellenando con 0 a la izquierda)
De bits (asignación)	<<=,>>=,>>>=

Operadores (III)

- Prioridades

Sufijo	[],.,(argumentos),expr++,expr--
Unarios	+,-,++expr,--expr,~,!
Creación o tipo	new, (*tipo*)expr
Multiplicadores	*,/,%
Aditivos	+,-
Desplazamiento	<<,>>,>>>
Relacionales	<,>,<=,>=,==,instanceof
Igualdad	==,!=
De bits y lógico (AND)	&

Operadores (IV)

- Prioridades *(continuación)*

De bits y lógico (XOR)	^
De bits y lógico (OR)	|
AND condicional	&&
OR condicional	||
Condicional	?:
Asignación	=,+=,-=,*=,/=,%=,<<=,>>=,>>>=

Sentencias

- De expresión.
  - Terminadas en punto y coma.
  - De asignación, de incremento y decremento, llamadas a métodos y creación de objetos.
- De declaración.
  - Declaración de variables locales.
- Bloques.
  - Agrupación de sentencias delimitadas por llaves.
  - Se utilizan allí donde se puede utilizar una expresión.

Sentencias condicionales

```
if(expresión-lógica)
   sentencia1
[else
   sentencia2]

switch (expresión-ordinal) {
   case n  : sentencias [;break];
   case n1 : sentencias [;break];
   case n2 : sentencias [;break];
   ...
   [default: sentencias]
}
```

Sentencias repetitivas

```
while (expresión-lógica)
   sentencia


do
   sentencia
while (expresión-lógica)


for (expresión-inicial; expresión-lógica; incremento)
   sentencia
```

- En el `for` se puede utilizar el operador , (coma) para separar varias sentencias en la expresión inicial o en la modificación

Saltos

- `break [`*`etiqueta`*`];`
  - Para salir de cualquier bucle o bloque de sentencias.
  - *`etiqueta`*, indica una sentencia identificada por *`etiqueta`*`: sentencia.`
- `continue [`*`etiqueta`*`];`
  - Permite ir al comienzo de cualquier bucle o saltar a una etiqueta.
- `return [`*`expresión`*`];`
  - Permite salir de un método devolviendo el valor de la expresión.