Programación 2

Introducción al Lenguaje Java

Profesores:
Ismael Figueroa - ifigueroap@gmail.com
Eduardo Godoy - eduardo.gl@gmail.com

Inicio de Java

- **Desarrollado por:** Sun Microsystems en 1991.
- **Propietario actual:** Oracle Corporation desde 2010.
- **Objetivo inicial y actual:**
  - Desarrollar un lenguaje de programación para crear software pequeños, rápidos, eficientes y portátiles para diversos dispositivos de hardware (teléfonos celulares, radiolocalizadores y asistentes digitales personales).
  - Ser el nexo universal que conecte a los usuarios con la información que esté situada en el computador local, en un servidor Web o en una base de datos.
- **Principio:** *Write Once, Run Everywhere*

Independencia de la Plataforma

- **Tanto a nivel del código fuente como del binario.**
- **Independencia en código fuente**: los tipos primitivos de datos de Java tienen tamaño consistentes en todas las plataformas de desarrollo. Las bibliotecas de Java facilitan la escritura del código, que puede desplazar se plataforma a plataforma.
- **Independencia en binario**: los archivos binarios (bytecodes) pueden ejecutarse en distintas plataformas sin necesidad de volver a compilar la fuente.

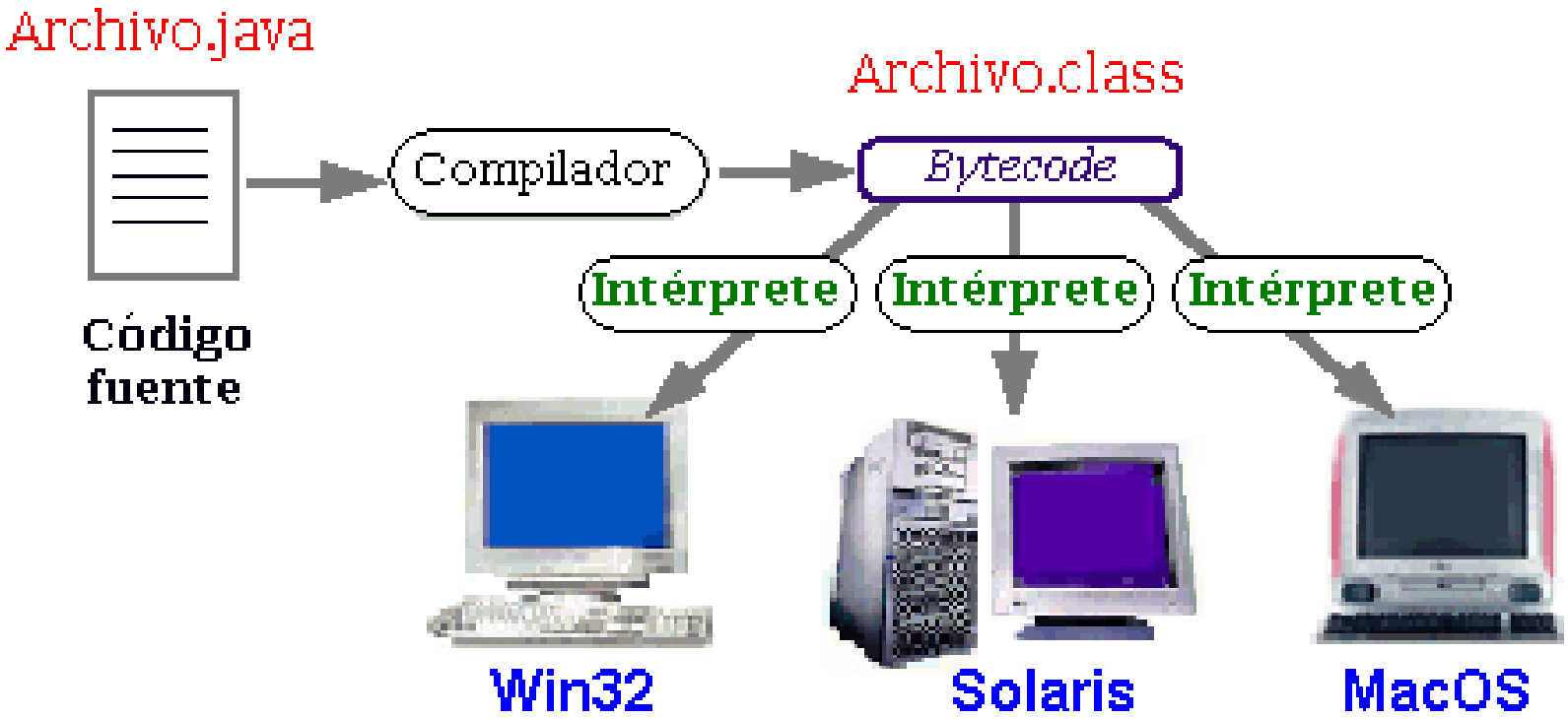
Ventajas

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Elimina redundancia a través de la herencia y polimorfismo
- Agiliza el desarrollo de software.

Más Ventajas

- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software.
- Recolección de basura.
- En Java no hay punteros (simplicidad).
- Las cadenas y los arreglos son objetos reales
- La administración de la memoria es automática

Proceso de Compilación, Interpretación y Ejecución



El Compilador `javac`

Ambiente de desarrollo Java

- Toma como entrada los archivos `*.java` y configuraciones sobre el *classpath*
- Genera archivos compilados en *bytecode* en archivos `*.class`
- El *classpath* indica dónde buscar otras clases externas a nuestro programa, tales como librerías

La Máquina Virtual de Java — JVM

Los archivos de bytecode son ejecutados por la JVM.

- La JVM carga el archivo de bytecode, y verifica qué este correctamente formado
- **Intérpretación inicial:** inicialmente el programa es ejecutado por el *intérprete JVM*, que es una forma lenta de ejecutar, pero que puede partir de inmediato
- **Compilación Just-in-Time**: durante la ejecución, la JVM va progresivamente compilando al lenguaje máquina de la plataforma, para mejorar la velocidad de ejecución

El Kit de Desarrollo — JDK

Contiene las herramientas, programas, bibliotecas, y todos los elementos necesarios para el desarrollo y depuración de aplicaciones Java

JDK / Compilación y Ejecución

- `javac`: es el compilador de Java, incluido solamente en el JDK. Se ejecuta desde línea de comandos: `javac HolaMundo.java`
- `java`: es el ejecutor de programas Java, se incluye tanto en el JDK como en el JRE. También se usa desde la línea de comandos: `javac HolaMundo.class`
- `javadoc`: Crea documentación en formato HTML a partir de el código fuente y los comentarios. Se usa desde la línea de comandos: `javadoc HolaMundo.java`

JDK / Utilitarios

- `jar`: crea librerías en formato JAR (Java ARChive), que permite empaquetar clases y otros recursos en un único paquete portable

- `jdb`: depurador de Java, con soporte para breakpoints y avance paso a paso. Se usa sobre archivos bytecode: `jdb HolaMundo 10` (10 es la linea del breakpoint). *En general no se usa manualmente sino como parte de un editor integrado*

Hola Mundo en Java

HolaMundo.java

```
1   public class HolaMundo {
2     public static void main(String[] args) {
3       System.out.println("Hola Mundo!");
4     }
5   }
6
```

Ejecutando Hola Mundo

Compilación y ejecución usando `javac` y `java`

```
> javac HolaMundo.java
> java HolaMundo
Hola Mundo!

```

La Clase Principal

- Todo método debe necesariamente pertenecer a una clase.
- Todo programa ejecutable debe tener una *clase principal*, que en general se llama `Main`.
- La clase principal **debe** contener el método `main`

```
public class Main {
  public static void main(String[] args) {
    /* codigo metodo principal */
  }
}
```

Tipos de Datos Primitivos

Se llaman **tipos primitivos** a aquellos que tienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante. Son definidos por defecto en la JVM, y están soportados en todas las plataformas donde la JVM funcione.

Java dispone de **ocho** tipos primitivos de variables:

- `boolean`: permite valores `true` y `false`
- `char`: representa characteres
- `byte`, `short`, `int`, `long`: valores enteros, de distintos tamaños máximos y mínimos
- `float`, `double`: valores reales de punto flotante, con distinta precisión

Precisión de los Tipos Primitivos

Tipo de variable	Descripción
`boolean`	1 byte. Valores `true` y `false`
`char`	2 bytes. Unicode. Comprende el código ASCII
`byte`	1 byte. Entero entre -128 y 127
`short`	2 bytes. Entero entre -32768 y 32767
`int`	4 bytes. Entero entre -2.147.483.648 y 2.147.483.647.
`long`	8 bytes. Valor entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.
`float`	4 bytes (entre 6 y 7 cifras decimales equivalentes). De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38.
`double`	8 bytes (unas 15 cifras decimales equivalentes). De -1.7976931348623E308 a -4.9406564584124E-324 y de 4.9406564584124E-324 a 1.7976931348623E308.

Características de los Tipos Primitivos

Los tipos primitivos de Java tienen algunas características importantes:

- El tipo `boolean` no es un valor numérico.
  - Sólo admite los valores `true` o `false`.
  - El tipo boolean no se identifica con el igual o distinto de cero, como en otros lenguajes.
  - El resultado de la expresión lógica para un `if` o similares **debe** ser de tipo booleano
- El tipo `char` contiene caracteres en código Unicode (que incluye el código ASCII), y ocupan 16 bits (2 bytes) por carácter. Comprende los caracteres de prácticamente todos los idiomas.

Características de los Tipos Primitivos

- Los tipos `byte`, `short`, `int` y `long` son números enteros que pueden ser positivos o negativos, con distintos valores máximos y mínimos. A diferencia de otros lenguajes, en Java no hay enteros `unsigned`.
- Los tipos `float` y `double` son valores de punto flotante, números reales, con 6-7 y 15 cifras decimales exactas—significativas—respectivamente.
- Se utiliza la palabra `void` para indicar la ausencia de un tipo de variable determinado.

Portabilidad de los Tipos Primitivos

A diferencia de otros lenguajes, los tipos de variables en Java están perfectamente definidos en todas y cada una de las posibles plataformas soportadas por la JVM. Por ejemplo, un `int` ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier plataforma soportada.

Variables

Una variable es un nombre asociado a un valor en memoria. Las variables se llaman así porque el valor asociado puede cambiar durante la ejecución del programa.

En Java todas las variables están asociadas a un tipo de dato, sea este primitivo o de referencia.

Definición de Variables

- Una variable se define especificando el tipo y el nombre de dicha variable
- Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase
- Las variables deben ser inicializadas antes de su uso, en otro caso tendrán valores nulos, y no podrán ser usadas

```
public class Auto {
  // Atributos marca y agno
  String marca;
  Integer agno;

  // Variable parametro km
  void acelerar(Integer km) {
    // Variable local aux
    int aux = 15;
    /* ... */
  }
}
```

Variables y Tipos de Referencia

Hay dos tipos de variables, según su tipo:

- De **tipos primitivos**: entero, float, etc.
- De **referencia** u **objeto**: funcionan como referencias a objetos de cualquier clase.

Esto cobra relevancia cuando se pasan como argumento a los métodos

```
// Tipos primitivos
int valor = 0;
double = 3.14;

// De referencia
String nombre = "Juan";
ArrayList a = new ArrayList();
```

Variables Locales y Como Atributos de Clase

- **Atributos:** son variables que se definen en la clase, fuera de los métodos, y que definen las características propias de cada instancia.
- **Parámetros de métodos:** son aquellas que se reciben como parámetro de un método. *Java siempre usa paso por valor!*
- **Variables locales de método:** son aquellas que se definen solo dentro de un método, para su adecuado funcionamiento

```
public class Auto {
  // Atributos marca y agno
  String marca;
  Integer agno;

  // Variable parametro km
  void acelerar(Integer km) {
    // Variable local aux
    int aux = 15;
    /* ... */
  }
}
```

Nombres de Variables

- Los nombres de variables en Java se pueden crear con mucha libertad.
- Pueden ser cualquier conjunto de caracteres numéricos y alfanuméricos, sin algunos caracteres especiales utilizados por Java como operadores o separadores (,.+-*/ etc).
- Existe una serie de palabras reservadas las cuales tienen un significado especial para Java y por lo tanto no se pueden utilizar como nombres de variables.

Palabras Reservadas para Nombres

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

Variables de Referencia

- Las variables con tipo de referencia indican dónde esta guardado el objeto en la memoria.
- A diferencia de C, en Java no hay punteros, y en realidad no podemos saber la dirección de memoria del objeto.
- Las referencias no inicializadas tienen el valor especial `null`. Llamar métodos o atributos sobre `null` siempre va a fallar
- Las nuevas referencias/objetos se crean utilizando el operador `new`, que veremos próximamente.

Alcance de las Variables

El **alcance** de una variable es el fragmento o sección de código donde ésta es accesible, y por tanto puede ser usada

- En Java existen distintos niveles de alcance para variables de clase o locales.
- Las variables de clase son accesibles desde todos los métodos de la clase
- Los parámetros solo son accesibles en el método que los recibe
- Las variables locales solo están disponibles en el bloque—código entre { } en el que están definidas.

```
public class Auto {
  // Atributos marca y agno
  String marca;
  Integer agno;

  // Variable parametro km
  void acelerar(Integer km) {
    // Variable local aux
    int aux = 15;
    /* ... */
  }

  void frenar(Integer ms) {
    /* error de alcance
    aux es variable local
    de acelerar */
    int delta = km - aux;
    /* ... */
}
```

Operadores Aritméticos

Son los operadores binarios usuales:
+, -, *, /, %

- Suma +
- Resta -
- Multiplicación *
- División /
- Módulo o resto %

```
float calculos(int m, int n, float r)
  int a = m + 1;
  int b = a - 42*r;
  int resto = m % n;

  return resto/r;
}
```

Asignación

- Los operadores de asignación permiten asignar un valor a una variable
- El operador de asignación por excelencia es el operador =
- La forma general de las sentencias de asignación con este operador es:
  - `variable = expresion;`
- Java dispone otros operadores de asignación que son acumulativos: `+=, -=, *=, %=`

Asignación Acumulativa

Operador	Uso	Expresión equivalente
+=	`a += b`	`a = a + b`
-=	`a -= b`	`a = a - b`
*=	`a *= b`	`a = a * b`
/=	`a /= b`	`a = a / b`
%	`a%= b`	`a = a% b`

Operadores de Comparación

- Sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor
- El resultado de estos operadores es siempre un valor booleano—`true` o `false`, según se cumpla o no la comparación

Operadores de Comparación

Operador	Uso	El resultado es `true` si:
>	`a > b`	a es mayor que b
>=	`a >= b`	a es mayor o igual que b
<	`a < b`	a es menor que b
<=	`a <= b`	a es mayor o igual que b
==	`a == b`	a y b son iguales
!=	`a != b`	a y b son diferentes

Operadores Lógicos

Se utilizan para construir expresiones lógicas, combinando distintos valores de verdad, y evaluar así condiciones más complejas.

Corto-circuito

Los operadores de conjunción y disyunción tienen un *comportamiento de corto-circuito* apenas se determina si el valor es falso o verdadero

Operadores Lógicos

Operador	Nombre	Uso	Resultado es `true` si:
`&&`	AND	`op1 && op2`	`op1` y `op2` son true. Si `op1` es `false` ya no se evalúa `op2`.
`||`	OR	`op1 || op2`	`op1` y `op2` son true. Si `op1` es `true` ya no se evalúa `op2`
`!`	NOT	`! op`	`op` es falso y es falso si `op` es verdadero
`&`	AND	`op1 & op2`	`op1` y `op2` son true. Siempre se evalúa `op2`.
`|`	OR	`op1 | op2`	`op1` u `op2` son true. Siempre se evalúa `op2`.

Corto-Circuito vs No Corto-Circuito

```java
public class CortoCircuito {
  public static boolean metodo(int n) {
    System.out.println("Evaluando: " + n);
    return (n < 3);
  }

  public static void main(String[] args) {
    // Imprime "Evaluando 2" y "Evaluando 3"
    boolean b1 = metodo(2) && metodo(3);
    System.out.println(b1);

    // Solo imprime "Evaluando 3"
    // Porque metodo(3) es falso no ejecuta metodo(2)
    boolean b2 = metodo(3) && metodo(2);
    System.out.println(b2);

    // El operador & no tiene corto circuito
    // Imprime "Evaluando 3" y "Evaluando 2"
    boolean b3 = metodo(3) & metodo(2);
    System.out.println(b3);
  }
}
```

Operador de Concatenación

- El operador + sirve para concatenar strings
- Si se concatena un número con un string, el número se transforma automáticamente a string
- En general todo objeto debe implementar el método `toString()` para poder ser representado

```
public class Hola {

  public static void main(String[] args) {
    String nombre = "Pepito";
    int edad = 33;

    String saludo = "Hola! :";
    saludo += nombre + " " + edad;
    System.out.println(saludo);
  }
}
```

Preguntas

Preguntas ?