Programación 2

Introducción a la Orientación a Objetos

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.

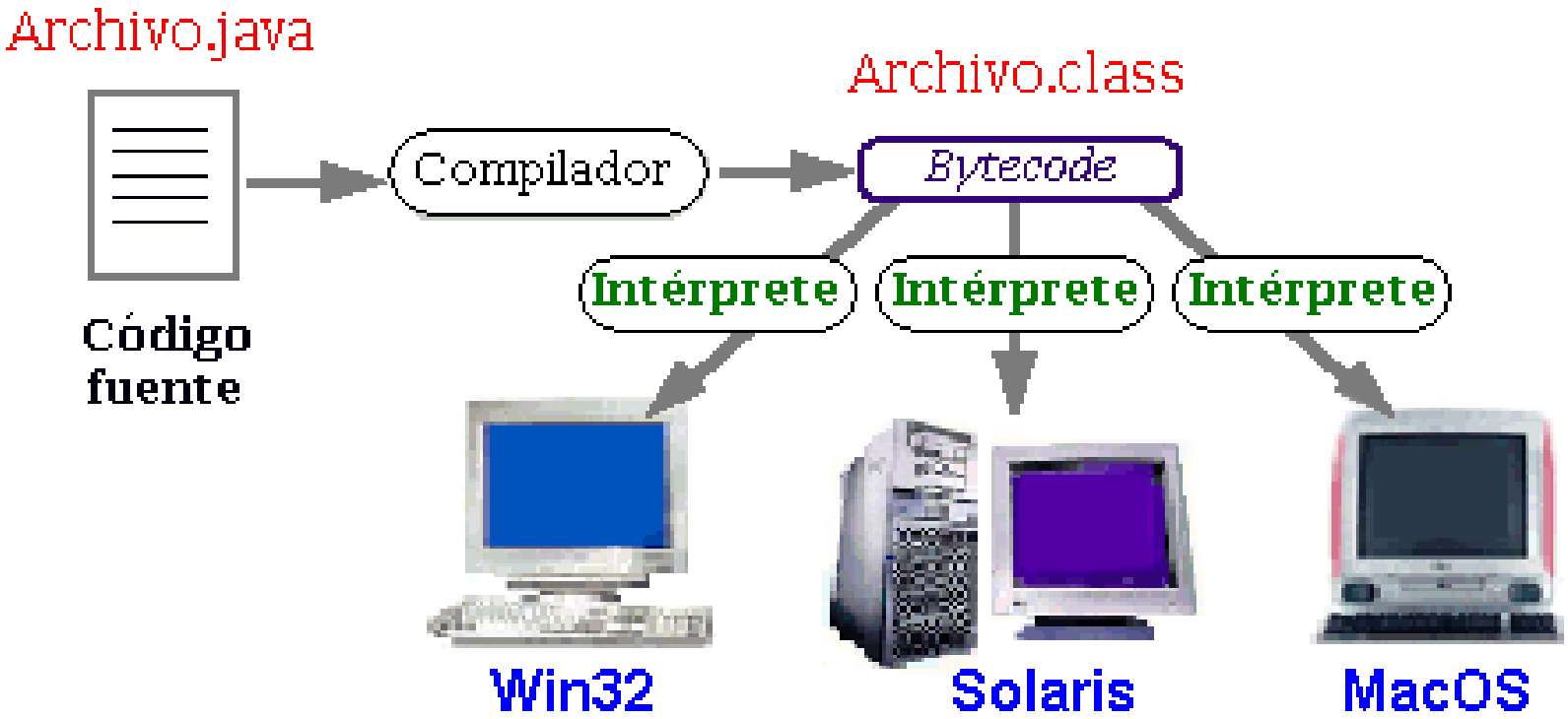
Ventajas

- 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`

```
1    > javac HolaMundo.java
2    > java HolaMundo
3    Hola Mundo!
4
```

Lenguajes de programación

Variables

Definición

- Una **variable** es un **nombre** que contiene un valor que puede cambiar a lo largo del programa.

Tipos principales de variables

- De acuerdo con el tipo de información que contienen, en JAVA hay dos tipos principales de variables:
  - Variables de **tipos primitivos**. Están definidas mediante un único valor que puede ser **entero**, de **punto flotante**, **carácter** o **booleano**. JAVA permite distinta **precición** y distintos rangos de valores para estos tipos de variables **char, byte, short, int, long, float, double, boolean**.
  - Variables **referencia**. Las variables referencia son referencias o nombres de una información más compleja: **arrays** u **objetos** de una determinada clase.

Lenguajes de programación

Variables

Rol de las variables

- Desde el punto de vista del rol o misión que cumplan en el programa, las variables pueden ser:
  - Variables **miembros** de una clase. Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias. Comúnmente son llamados **atributos** de la clase.
  - Variables **locales**. Se definen dentro de un método o más en general dentro de cualquier bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

Lenguajes de programación

Variables - Nombres de variables

Características de las 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.

Lenguajes de programación

Variables - Palabras reservadas

Palabras reservadas

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

(*) son palabras reservadas, pero no se utilizan en la actual implementación del lenguaje JAVA

Lenguajes de programación

Variables -Tipos de datos primitivos

Características de las variables

Se llaman **tipos primitivos** de variables de JAVA a aquellas variables sencillas que contienen los tipos de información más habituales: valores boolean, caracteres y valores numéricos enteros o de punto flotante.

Características de las variables

- Java dispone de **ocho** tipos primitivos de variables:
  - Un tipo para almacenar valores **true** y **false (boolean)**.
  - Un tipo para almacenar **caracteres (char)**.
  - Seis tipos para guardar valores **numéricos**.
    - Cuatro tipos para **enteros (byte, short, int y long)**
    - Dos para valores **reales de punto flotante (float y**

Lenguajes de programación

Variables -Tipos de datos primitivos

Tipo de variable	Descripción
boolean	1 byte. Valores true y false.
char	2 bytes. Unicode. Comprende el código ASCII.
byte	1 byte. Valor entero entre -128 y 127.
Short	2 bytes. Valor entero entre -32768 y 32767.
int	4 bytes. Valor 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 -1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308.

Lenguajes de programación

Variables -Tipos de datos 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 que aparece como condición en un bucle o en una bifurcación debe ser boolean.
- 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.

Lenguajes de programación

Variables -Tipos de datos primitivos

Los tipos primitivos de JAVA tienen algunas características importantes:

- 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 (significativos), respectivamente.
- Se utiliza la palabra **void** para indicar la ausencia de un tipo de variable determinado.

Lenguajes de programación

Variables -Tipos de datos primitivos

Los tipos primitivos de JAVA tienen algunas características importantes:

- A diferencia de otros lenguajes, los tipos de variables en JAVA están perfectamente definidos en todas y cada una de las posibles plataformas. Por ejemplo, un int ocupa siempre la misma memoria y tiene el mismo rango de valores, en cualquier tipo de ordenador.

- Existen extensiones de JAVA para aprovechar la arquitectura de los procesadores Intel, que permiten realizar operaciones de punto flotente con una precisión extendida de 80 bits (10 bytes).

Lenguajes de programación

Variables - Definición de variables

Definición de variables en JAVA.

- 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.
- Si no se especifica un valor en su declaración, las variables primitivas deberán ser inicializadas antes de su uso, sino, JAVA no reconocerá la variable en el contexto actual.

Lenguajes de programación

Variables - Definición de variables

Definición de variables en JAVA.

- Es importante distinguir entre la **referencia** a un objeto y el **objeto** mismo.

- Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador.
  - A diferencia de otros lenguajes, JAVA no permite acceder al valor de la dirección, pues en este lenguaje se han eliminado los **punteros**.

- Al declarar una referencia, ésta aún no se encuentra "apuntando" a ningún objeto en particular y por eso se le asigna el valor **null**.

- Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador **new**. Este operador reserva en la

Lenguajes de programación

Variables - Visibilidad y vida de las variables

Definición

- Se entiende por **visibilidad, ámbito, alcance o scope** de una variable, el fragmento o parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en una expresión.
- En JAVA todas las variables deben estar incluidas en una clase.
- En general las variables declaradas dentro de unas llaves {}, es decir dentro de un bloque, son visibles y existen dentro de estas llaves.

Lenguajes de programación

Operadores - Aritméticos

Operadores aritméticos

→ Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: **suma** (+), **resta** (-), **multiplicación** (*), **división** (/) y **resto de la división** (%).

Lenguajes de programación

Operadores - Asignación

Operadores de asignación

- Los operadores de asignación permiten asignar un valor a una variable.
- El operador de asignación por excelencia es el operador **igual** (=).
- La forma general de las sentencias de asignación con este operador es:
  - **variable = expresion**;
- JAVA dispone de otros operadores de asignación. Se trata de las versiones abreviadas del operador (=) que realizan operaciones "acumulativas" sobre una variable.

Lenguajes de programación

Operadores - Asignación acumulativas

Operador	Uso	Expesión equivalente
$+=$	op1 $+=$ op2	op1 $=$ op1 $+$ op2
$-=$	op1 $-=$ op2	op1 $=$ op1 $-$ op2
$*=$	op1 $*=$ op2	op1 $=$ op1 $*$ op2
$/=$	op1 $/=$ op2	op1 $=$ op1 $/$ op2
$\%=$	op1 $\%=$ op2	op1 $=$ op1 $\%$ op2

Lenguajes de programación

Operadores - Unarios e Instanceof

Operadores unarios

Los operadores **más** (+) y **menos** (-) unarios sirven para mantener o cambiar el signo de una variable o expresión numérica.

Operador **instanceof**

- El operador **instanceof** permite saber si un objeto pertenece o no a una determinada clase.
- Es un operador binario cuya forma general es:
  - objectName **instanceof** ClassName
- Devuelve **true o false** según el objeto pertenezca o no a la clase.

Lenguajes de programación

Operadores - Condicional ?

Operador condicional ?

- Este operador permite realizar bifurcaciones condicionales sencillas.
- Su forma general es la siguiente:
  - expresionBooleana ? respExpresionVerdero : respExpresionFalso
- Se evalúa **expresionBooleana** y devuelve **respExpresionVerdero** si es verdadera o **respExpresionFalso** si es falsa.
- Es el único operador ternario en JAVA (requiere de tres argumentos).

Lenguajes de programación

Operadores - Incrementales

Operadores incrementales

- JAVA dispone del operador **incremento** $(++)$ y **decremento** $(--)$.
- El operador $(++)$ incrementa en una unidad la variable a la que se aplica, mientras que $(--)$ la reduce en una unidad.
- Estos operadores se pueden utilizar de dos formas:
  - **Precediendo** a la variable $(++x)$. En este caso primero se utiliza la variable en la expresión (con el valor actual) y luego se incrementa.
  - **Siguiendo** a la variable $(x++)$. En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.

Lenguajes de programación

Operadores - Relacionales

Operadores relacionales

- Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor.
- El resultado de estos operadores es siempre un valor **boolean (true o false)** según se cumpla o no la relación considerada.
- Estos operadores se utilizan con mucha frecuencia en las **bifurcaciones** y en los **bucles**.

Lenguajes de programación

Operadores - Relacionales

Operador	Uso	El resultado es true si:
$>$	op1 $>$ op2	op1 es mayor que op2
$>=$	op1 $>=$ op2	op1 es mayor o igual que op2
$<$	op1 $<$ op2	op1 es menor que op2
$<=$	op1 $<=$ op2	op1 es mayor o igual que op2
$==$	op1 $==$ op2	op1 y op2 son iguales
$!=$	op1 $!=$ op2	op1 y op2 son diferentes

Lenguajes de programación

Operadores - Lógicos

Operadores lógicos

- Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos **(true y/o false)** o los resultados de los operadores relacionales.

- Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir).

- Esto puede traer resultados no deseados y por eso se han añadido los operadores (**&**) y (|) que garantizan que los dos operandos se evalúan siempre.

Lenguajes de programación

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 verdadero ya no se evalúa op2
! =	Negación	! op	op es falso y es falso si op es verdadero ya no se evalúa op2
&	AND	op1 **&** op2	op1 y op2 son true. Siempre se evalúa op2.
|	OR	op1 | op2	op1 u op2 son true. Siempre se evalúa op2.

Lenguajes de programación

Operadores - Concatenación de caracteres

Operador de concatenación de caráteres

- El operador **más (+)** se utiliza también para concatenar cadenas de caracteres.
- Por ejemplo, para escribir una cantidad con un rótulo y unas unidades puede utilizarse la sentencia:
  - System.out.println("El total asciende a " + totalUnidades);
- En el ejemplo anterior, el operador de concatenación se utiliza para construir la cadena de caracteres que se desea imprimir por medio del mótodo **println()**.
- La variable numérica **totalUnidades** es convertida automáticamente por JAVA en cadena de caracteres para poderla concatenar. En otras ocasiones se deberá llamar explícitamente a un método para que

Lenguajes de programación

Estructuras de control

Definición

- Facilitan la realización de determinadas acciones, mientras que una condición se cumpla.
- Permiten tomar decisiones de **qué** hacer, en función de las condiciones que se den en el programa en un momento dado de su ejecución.

En JAVA

El lenguaje JAVA soporte cuatro tipos de estructuras de control:

- **Toma de decisión:** if / if-else / switch-case.
- **Bucle o ciclo:** for / while / do-while.
- **Saltos:** break / continue / return / goto.
- **Excepciones.**

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- En JAVA la sentencia **if** / **if-else** dota a los programas de la habilidad de ejecutar conjuntos de sentencias según la condición.

La sentencia para **if** es:

```
if (condición) {
    Bloque de sentencias si la condición es verdadera.
}
```

La sentencia para **if-else** es:

```
if (condición) {
    Bloque de sentencias si la condición es verdadera.
}
else {
    Bloque de sentencias si la condición es falsa.
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Supongamos que un programa debe realizar diferentes acciones dependiendo de si el usuario oprime el botón **aceptar** o el botón **cancelar** en una ventana de diálogo. Nuestro programa puede realizar esta bifurcación usando la sentencia **if-else**:

Bifurcación con la sentencia **if-else**.

```
if (respuesta.equals("Aceptar")) {
    System.out.println("Su petición esta siendo atendida");
}
else {
    System.out.println("Cancelando acción" );
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Se pueden anidar expresiones **if-else**, para poder implementar aquellos casos con múltiples acciones. Esto es lo que se suele denominar como sentencias **else-if**.

Ejemplo de Bifurcaciones múltiples:

- Supongamos que se desea escribir un programa que clasifique según el contenido de una variable denominada **valor**, asigne una letra a otra variable denominada **clasificacion**.
- A, para un valor entre 100 y 91 (inclusive).
- B, para un valor entre 90 y 81 (inclusive).
- C, para un valor entre 80 y 71 (inclusive).
- F, si no es ninguno de los anteriores.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de Bifurcaciones múltiples - Solución

```
int valor;
char clasificacion;

if (valor > 90   &&   valor <= 100) {
    clasificacion = 'A';
}
else if (valor > 80   &&   valor <= 90) {
    clasificacion = 'B';
}
else if (valor > 70   &&   valor <= 80) {
    clasificacion = 'C';
}
else {
    clasificacion = 'F';
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Este sistema de programación (else-if) no es recomendable, por la pérdida de claridad en la lectura del programa. Por ello el lenguaje JAVA incluye la sentencia **switch-case**, para dirigir el flujo de control de variables con múltiples valores.
- La sentencia **switch** permite seleccionar entre varias opciones, según el valor de cierta expresión.
- Cada sentencia case debe ser única y el valor que evalúa debe ser del mismo tipo que el devuelto por la expresión de la sentencia **switch**.
- Las sentencias **break** permiten salir del **switch** y continar con la siguiente instrucción. Las sentencias **break** son necesarias porque sin ellas se ejecutarían secuencialmente las sentencias **case** restantes.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

- Finalmente, se puede usar la sentencia **default** para manejar los valores que no son explícitamente contemplados por alguna de las sentencias **case**. Su uso es altamente recomendado.

La forma general de la sentencia **switch** es la siguiente:

```
switch (expresionMultivalor) {
    case valor1:
        bloqueDeSentenciasParaValor1;
        break;
    case valor2:
        bloqueDeSentenciasParaValor2;
        break;
   ...
    default:
        bloqueDeSentenciasParaValorNoDeterminado;
        break;
}
```

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de sentencia **switch**

- Supongamos un programa con una variable entera **meses** cuyo valor indica el mes actual y se desea imprimir el nombre del mes en que estemos. Se puede utilizar la sentencia **switch** para realizar esta operación.
- Si el valor a comparar no existe o no es un valor válido, el programa debe indicar el error.
- Luego de implementar la solución con la sentencia **switch**, implementarla con la sentencia **else-if** y comparar ambas soluciones.

Lenguajes de programación

Estructuras de control - Sentencias condicionales

Ejemplo de sentencia **switch** - Solución

```
int mes;

switch (mes) {
    case 1:
       System.out.println("Enero");
       break;
    case 2:
       System.out.println("Febrero");
       break;
    //... Otros meses
    default:
       System.out.println("Mes no válido");
       break;
}
```

Preguntas

Preguntas ?