
# Introducción a Java

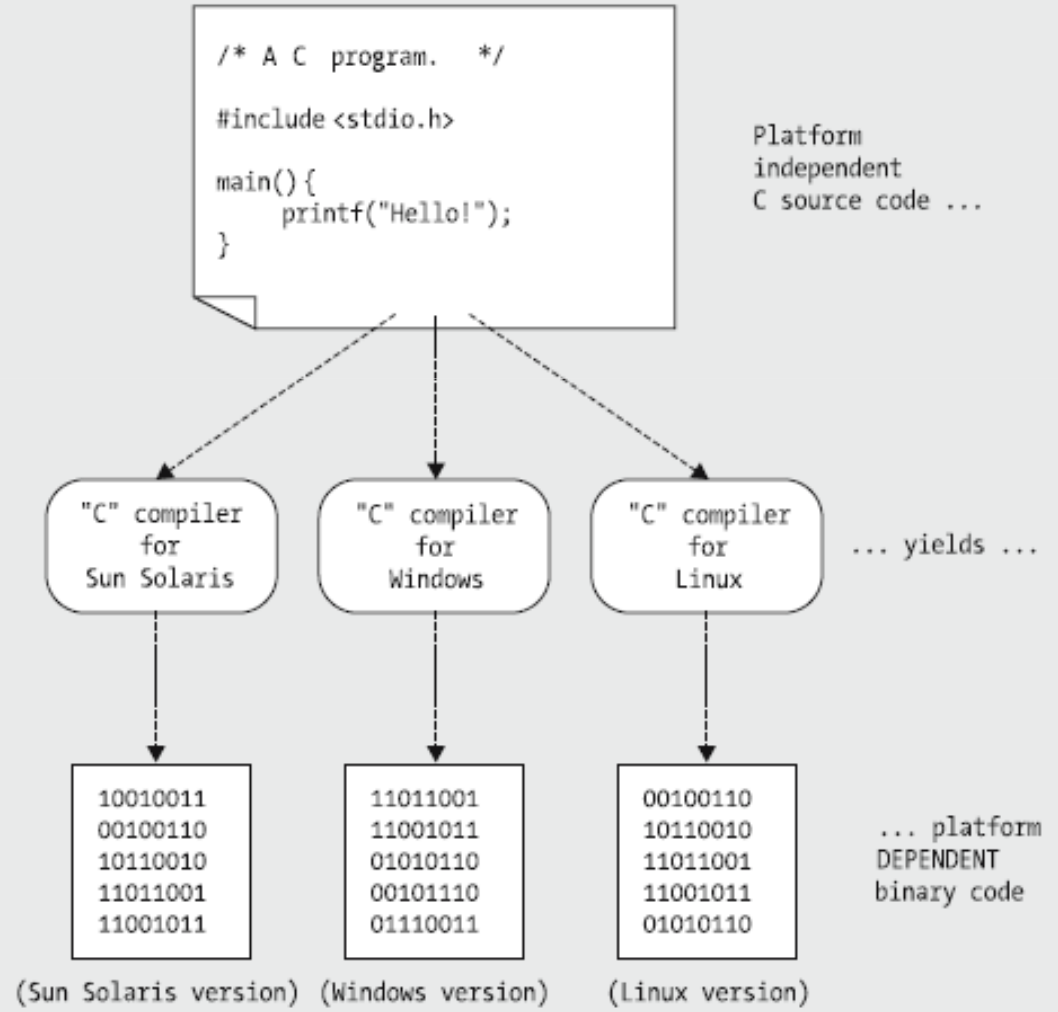
Dr. (c) Noé Alejandro Castro Sánchez

# Programas Java

- Applets
  - Pueden correr en navegadores Web
  - Agregan funcionalidad a páginas Web
  - Se llega a restringir su funcionalidad (e. g., no pueden:
    - Leer/escribir archivos en la máquina local
    - Abrir sockets para comunicarse con otros equipos).
- Aplicaciones
  - Corren independientemente de la plataforma
  - No dependen de navegadores (aunque requieren la máquina virtual JVM instalada)
  - No tienen limitaciones.

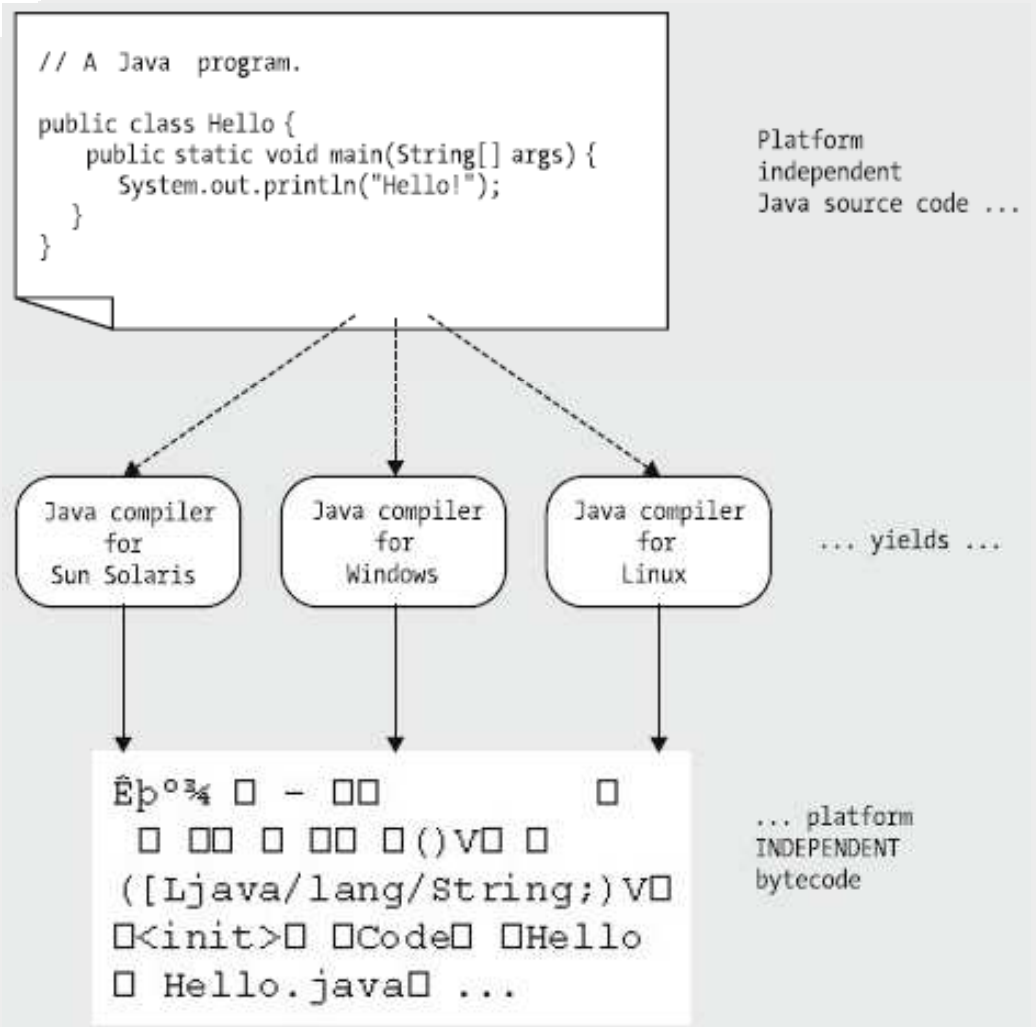
# Características (independiente de la arq.)

- Otros lenguajes producen una versión ejecutable dependiente de la arquitectura donde se obtuvo su código binario.



# Características (II)

- El código fuente Java es transformado a un código intermedio conocido como *bytecode*, neutral a la plataforma.
- El *bytecode* puede ejecutarse en toda máquina que cuente con una JVM.



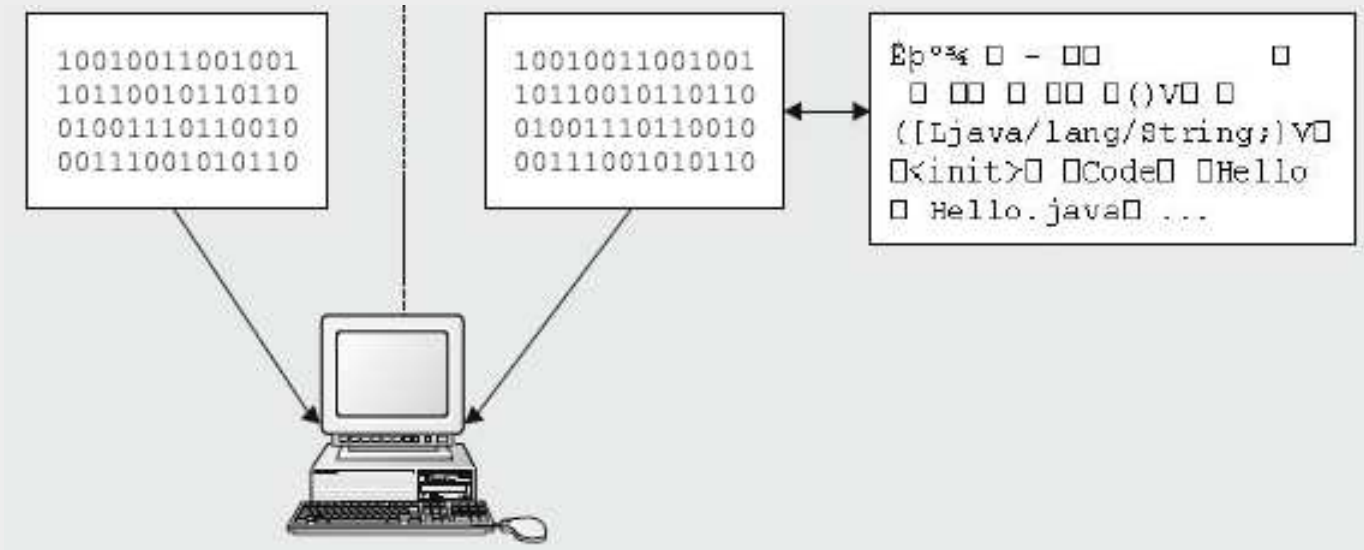
# Java Virtual Machine (JVM)

- Software que sabe cómo interpretar y ejecutar los *bytede* de Java.

Código binario listo para correr en una arq. determinada

JVM dependiente de la plataforma corriendo en una arq. específica

*Bytecode* corre bajo el control de una JVM dependiente de la plataforma



# Java Virtual Machine (JVM): Ejecución

- Programas Java **compilados** a *bytecode*
  - *Bytecode* es similar a código máquina, pero no puede correr directamente en ninguna máquina física.
  - JVM es el único que puede leerlo, simulando una máquina real.
- JVM es un **intérprete** que traduce el *bytecode* a código máquina.

# Características generales de Java

- Lenguaje simple: similar a C/C++
- Es seguro: la JVM hace comprobaciones de seguridad, además carece de características inseguras como los punteros
- Orientado a objetos (todo es un objeto, salvo algunos tipos primitivos)
- Amplia cantidad de librerías: gestión de red, creación de interfaces gráficas, acceso a datos, etc.

# Aplicación en Java

```
1. //Ejemplo de aplicación “Hola Mundo”
2. //archivo: Saludo.java
3. public class Saludo
4. {
5.    public void saludar()
6.    {
7.          System.out.println(“Hola");
8.    }
9. }
```

# Aplicación en Java (II)

```
1. //Ejemplo de aplicación “Hola Mundo”
2. //archivo: Saludo.java
3. public class Saludo
4. {
5.    public void saludar()
6.    {
7.          System.out.println(“Hola");
8.    }
9. }
```

Comentarios

Declaración de la clase *Saludo*

Declaración de un método

Método para escribir en el flujo de salida estándar

# Aplicación en Java (III)

```
1. //Ejemplo de aplicación "Hola Mundo“
2. //archivo: PruebaSaludo.java
3. public class PruebaSaludo
4. {
5.    public static void main(String[] args)
6.    {
7.          Saludo hello = new Saludo();
8.          hello.saludar();
9.    }
10. }
```

# Aplicación en Java (IV)

```
1. //Ejemplo de aplicación "Hola Mundo“
2. //archivo: PruebaSaludo.java
3. public class PruebaSaludo
4. {
5.    public static void main(String[] args)
6.    {
7.          Saludo hello = new Saludo();
8.          hello.saludar();
9.    }
10. }
```

Nombre de clase

Punto de inicio del programa

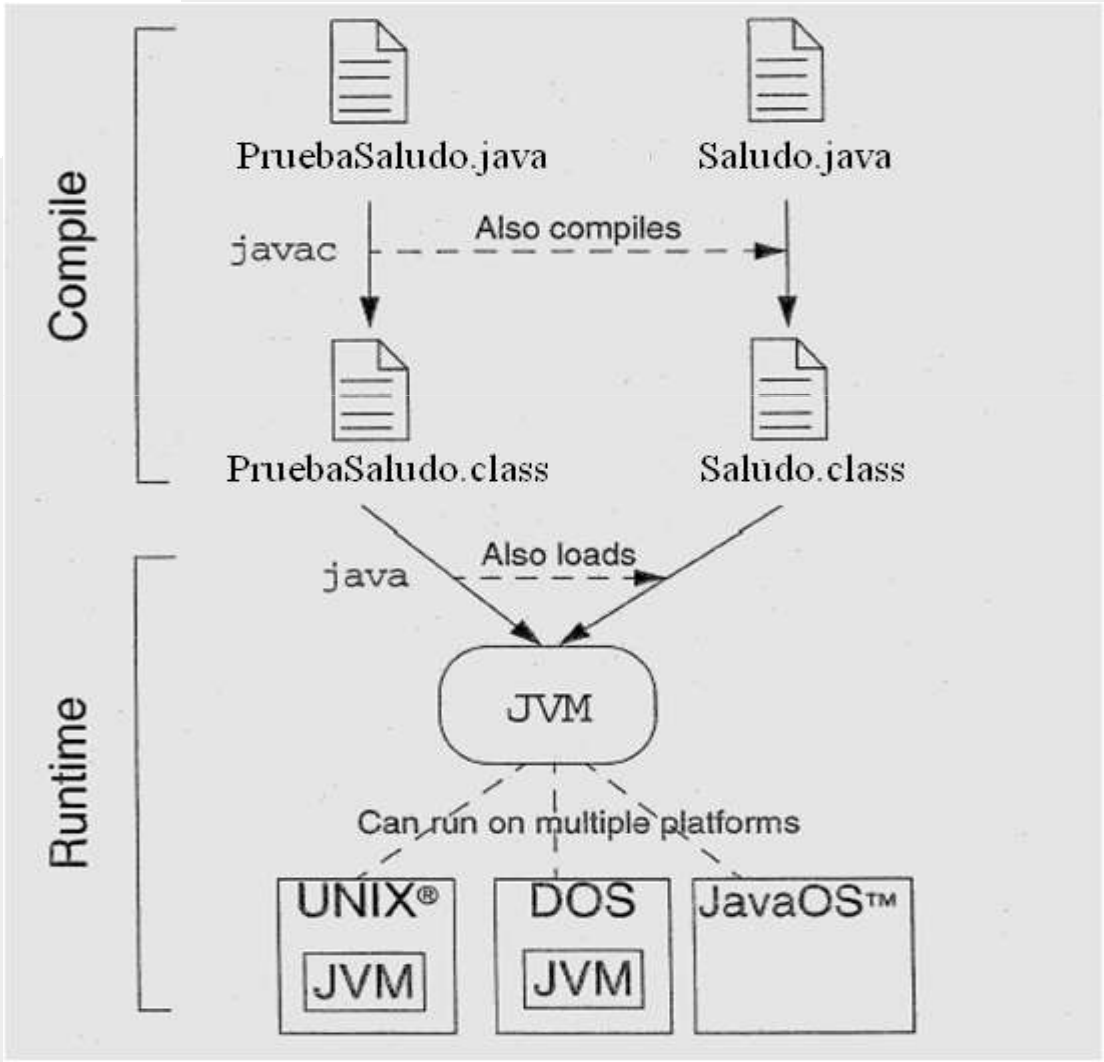
Creación de objeto

Ejecución del método del objeto

# Aplicación en Java: descripción

- Dos programas con extensión *.java*
- Cada uno declara una clase (*Saludo* y *PruebaSaludo*)
- El nombre del archivo se corresponde con el nombre de la clase que declara
- Uno de los archivos (PruebaSaludo) declara el método *main*
- Cuando se indica a Java que ejecute un archivo (clase), busca el método *main* en dicha clase y lo ejecuta.

# Ejecución de aplicación



# Ejecución de aplicación

- Compilación: se utiliza el programa *javac*
  - Se ubica en directorio *bin* donde esté instalado java
- Importante que *bin* esté en el *path* del sistema
  - Comprobación: ejecutar en consola:
    **`C:\>`** `javac`
  - ¿Mensaje "javac no se reconoce como comando interno o externo"? Ejecutar en consola:
    **`C:\>`** `set PATH=%PATH%;`*`directorio_java`*`\bin`
  - La carpeta `bin` se encuentra dentro de la carpeta que inicia con nombre `jdk`
  - Ejemplo: `set PATH=%PATH%;C:\Archivos de programa\Java\jdk1.6.0_25\bin`
- Ubicarse en el directorio del archivo fuente (.java)

# Ejecución de aplicación (II)

- Compilación de archivos: ejecutar en consola:
  - C:\> **javac PruebaSaludo.java**
  - Creación de archivos *PruebaSaludo.class* y *Saludo.class*
- Ejecución de aplicación: ejecutar en consola:
  - C:\> **java PruebaSaludo**
    Hola
- El comando java admite como parámetro **el nombre de la clase, no del archivo:**
  - *PruebaSaludo en* la ejecución en consola no lleva ninguna extensión.

# Datos primitivos

- Nombres en Java son iguales o muy similares a los definidos en lenguaje C/C++
  - En C/C++, el tamaño es definido por el compilador.
  - En Java, sus características y comportamiento es siempre el mismo, independientemente del equipo donde corra.

# Tipo Caracter

- C: representa un carácter ASCII (1 byte)
- Java: representa un carácter Unicode (2 bytes)

```
// C char examples
char letterJ = 'J';
char letterA = 'A';
char letterV = '\126';
char digit0  = '\x030k';
char digit1  = '1';
char digit2  = '2';


// Java char examples
char letterJ = 'J';
char letterA = 'A';
char letterV = '\u0056';
char digit0  = '\u0030';
char digit1  = '1';
char digit2  = '2';
```

Para almacenar códigos Unicode en archivos ASCII se usa la secuencia de escape **\u**

# Tipo Entero

- C: tamaño asignado por compilador: en gral. 2 bytes
- Java: el lenguaje define estrictamente 4 bytes

```
// C integers
long  int val_1 = 250000;
long      val_2 = 0x36B;
int       val_3 = -1800;
short int val_4 = 017;
short     val_5 = -25;


// Java integers
long  val_1 = 250000;
long  val_2 = 0176;
int   val_3 = 0x3F;
short val_4 = -93;
short val_5 = 25;
byte  val_6 = 120;
byte  val_7 = -34;
```

Valor octal
Valor hexadecimal

Almacena números usando 8 bits (-128 a 127)

# Tipo flotante

- Ambos definen *float…*
  - C: en general 4 bytes de tamaño
  - Java: siempre 4 bytes de tamao

```
// C floating-points
float  val_1 = 0.25f;
float  val_2 = 12.4901f;
float  val_3 = 25.138e-10f;
double val_4 = 0.123456789;
double val_5 = 1.9876540e5;
double val_6 = 0.000001234;


//Java floating-points
float  val_1 = 0.25f;
float  val_2 = 12.4901f;
float  val_3 = 25.138e-10f;
double val_4 = 0.123456789;
double val_5 = 1.9876540e5;
double val_6 = 0.000001234;
```

- … y double
  - C: en general 8 bytes de tamaño
  - Java: siempre 8 bytes de tamaño

# Tipo boolean

- C++: implementado como entero (*true* corresponde a 1 y *false* a 0). Se pueden usar indistintamente.
- Java: únicos valores válidos *true* y *false*.

```
//C++ boolean
bool on  = true;
bool off = false;
bool yes = 0;
bool no  = 1;


// Java boolean
boolean on  = true;
boolean off = false;
// This is illegal
boolean yes = 1;
// This is illegal
boolean no  = 0;
```

# Ejercicio I

- Implementar en un archivo *.java* la clase *Contador* del ejercicio visto en la presentación anterior.
- Implementar otro archivo *.java* la clase *PruebaContador* e implementar la siguiente funcionalidad:
  - Crear un objeto contador
  - Simular que se pulsa 15 veces el botón de incremento
  - Imprimir el valor del contador
  - Simular la pulsación del botón *resetear*
  - Imprimir el valor del contador

# Plantilla de una clase

```
public class NombreClase
{ // Variables de clase (atributos)
      DeclaraciónVariable1
      DeclaraciónVariable2
      ...
  // Métodos de la clase
      MethodDefinition1
      MethodDefinition2
      ...
} // Fin de la clase
```

# Inicialización y constructores

- La inicialización de atributos y la creación del objeto se han visto como operaciones diferentes.
- La abstracción de la POO permite combinarlas implícitamente.
- Constructor: método que se invoca automáticamente cuando el objeto es creado.
  - Permite moldear la manera en que el objeto es creado e inicializado.
- Características:
  - Mismo nombre de la clase
  - No retorna valor

# Inicialización y constructores (II)

```
class Contador
{
   int valor, limite;
   …
   public Contador()
   {
      valor = 0;
      limite = 999;
   }
   …
```

# Ocultando la implementación

- Acceder directamente a los atributos de un objeto es permisible, pero no ideal, porque une el código implementado con la representación actual del objeto

```
…
public static void main(String[] args)
{
   Contador cont1 = new Contador();
   …
   cont1.imprimir();   // 10
   cont1.memoria = 99; // No debería ser posible
   cont1.imprimir();   // 99
}
…
```

# Ocultando la implementación (II)

- Permite hacer inaccesibles las representaciones (estados) del objeto.
- Java implementa 4 tipos de accesos, los más comúnes son `private` y `public`
  - `private` permite el acceso únicamente desde la propia clase
  - `Public` permite el acceso a cualquier código cliente

```
class Counter {
   private int valor, limite;
…
```