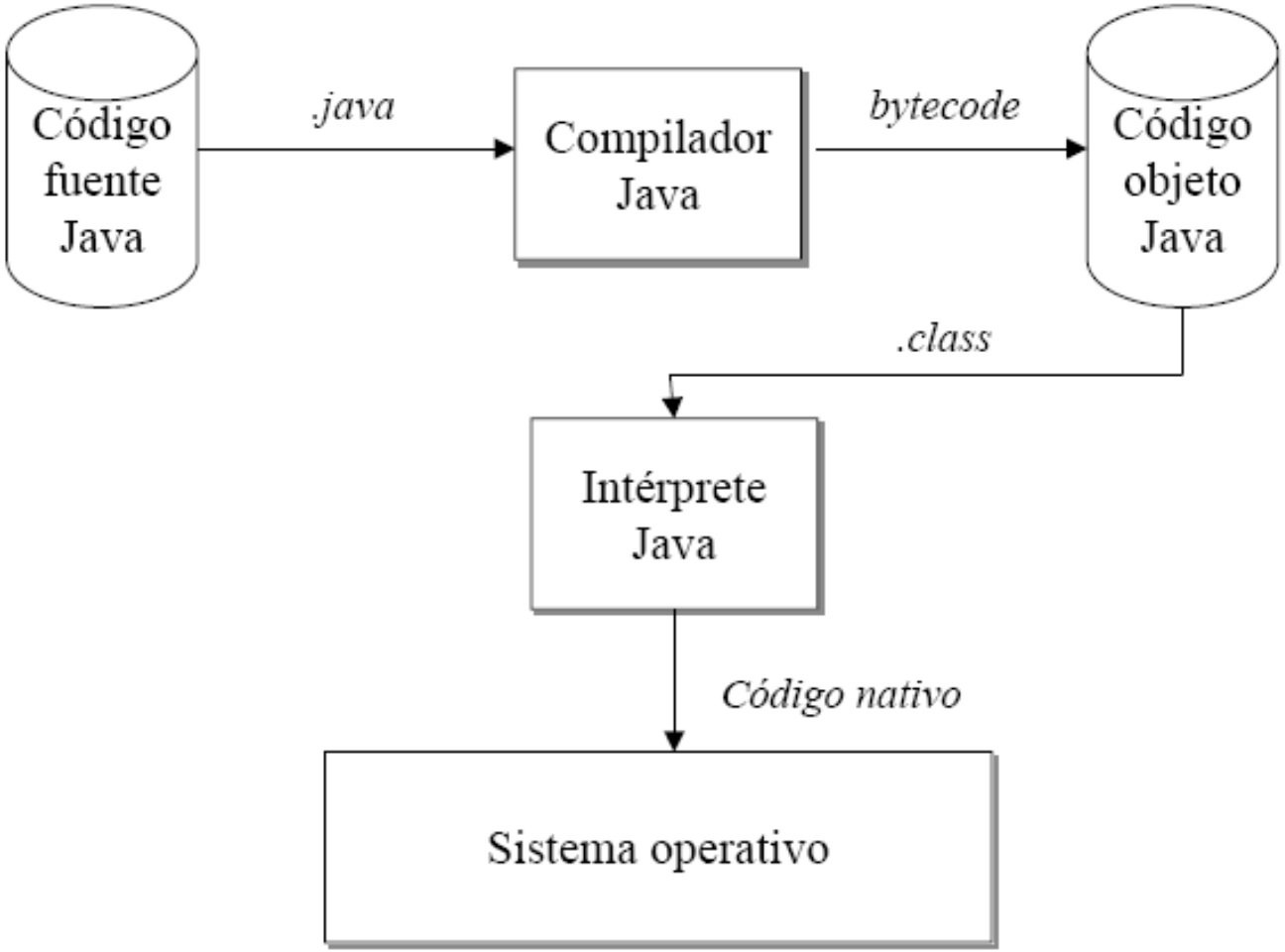
Ambiente de desarrollo Java

A partir de un código fuente Java (archivo .java), se genera a través de un compilador Java (javac), código objeto Java (bytecodes) en un archivo .class.

Una vez obtenido este código compilador a un nivel independiente de la plataforma, el intérprete de Java (máquina virtual de Java, JVM) lee los bytecodes generando código nativo específico de una plataforma.

Ambiente de desarrollo Java(...)



Estructura de un programa Java

Un programa en Java se estructura con base en las siguientes reglas:

- Al menos define una sola clase, pero pueden haber varias
- Para mejor organización, cada clase debería estar en su propio archivo.
- El nombre del archivo debe ser el mismo que el nombre de la clase definida.
- Tal archivo debe tener la extensión .java

Estructura de un programa Java(...)

Típicamente, un programa Java se estructura en tres partes:

- Declaración del paquete,
- Importación de clases (o paquetes)
- Declaración de la clase

Estructura de un programa Java(...)

Ejemplo:

```
1 //Declaración de paquete
2 package examenes.Examen1;
3
4 //Declaración de importaciones
5 import javax.swing.JButton; //Importa una clase
   específica
6 import java.util.*; //Importa un paquete entero
7
8 //Definición de clase
9 public class Prueba {
10 . . .
11}
```

Variables

- Una variable es una porción de memoria referenciado con un identificador que sirven para almacenar un dato. Una variable solo puede almacenar un solo tipo de dato.
- Por lo tanto, cada dato tiene su representación en memoria y su posible rango de valores. Una variable se declara de la forma:

```
tipo identificador;
```

Variables (...)

- Un identificador es usado por el programador para referenciar una variable, un paquete, un método o una clase.
- Las reglas para los identificadores son:
  - Cadenas de texto Unicode de cualquier tamaño
  - El carácter inicial debe ser una letra, un guion bajo (_) o signo de peso ($)
  - No puede ser igual a una palabra reservada, literales booleanas (true, false) o null
  - Después, puede contener cualquier combinación de caracteres excepto los espacios y caracteres reservados
  - Es sensitivo al caso

Variables (...)

Ejemplos:

```
entero
granInterface
2_nodo3                 //ilegal
$nombre
!direccion              //ilegal
Mi credencial           //ilegal
miPesoAnterior
```

Por convención, los identificadores de variables comienzan con letra minúscula. Si fuera conformado por varias palabras, las demás palabras después de la inicial, se escriben con letra mayúscula.

Palabras reservadas

El lenguaje Java está basado en C++, por lo que su juego de palabras reservadas es similar, más algunas muy particulares del lenguaje.

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

Tipos de datos primitivos

Java ofrece sentencias de tipos para representar datos primitivos como

- Valores numéricos enteros y reales
- Caracteres
- Valores lógicos

Tipos de datos primitivos(...)

Los tipos de datos numéricos enteros son:

Nombre	Tamaño	Valores
byte	8 bits	-128 a 127
short	16 bits	-32768 a 32767
int	32 bits	-2147483648 a 2147483647
long	64 bits	-2^{63} a $2^{63}-1$

Tipos de datos primitivos(...)

Los tipos de datos numéricos decimales son:

Nombre	Tamaño	Valores
float	32 bits	-3.4E8 a 3.4E38
double	64 bits	-1.7E308 a 1.7E308

Tipos de datos primitivos(...)

- Los tipos de datos booleanos son:

Nombre	Tamaño	Valores
`boolean`	1 bit (1 byte)	`true, false`

Tipos de datos primitivos(...)

El tipo de dato caracteres:

Nombre	Tamaño	Valores
char	16 bits	Tabla Unicode

Literales

Una literales un valor especificado en el programa fuente a diferencia de aquéllos que son obtenidos en tiempo de ejecución. Las literales pueden representar variables primitivas o cadenas.

- No se puede asignar un valor a una literal.
- La literal debe aparecer al lado izquierdo de una asignación.

Literales(...)

Los únicos valores posibles para el tipo de dato booleano son false y true.

Ejemplos:

```
boolean aprobado = true;
boolean esNumeroPrimo = false;
```

Literales(...)

Una literal carácter es expresada mediante la especificación del carácter entre comillas sencillas (‘) de la forma:

```
char letraA= ‘A’;
```

Otra forma de expresarlos es mediante la especificación en Unicode, esto es, la secuencia \u seguida de 4 números hexadecimales.

```
char letraUnicode= ‘\u4b67’;
```

Literales(...)

Java soporta secuencia de escape para algunos caracteres especiales

Secuencia de escape	Descripción
\uxxxx	Carácter UNICODE hexadecimal (xxxx)
\'	Comilla Simple
\"	Comilla doble
\\	Barra invertida
\r	Retorno de carro
\n	Nueva línea
\f	Alimentación de pagina
\t	tabulador
\b	Retroceso

Literales(...)

Las literales flotantes expresan números de punto flotante. Una literal flotante es de algún tipo de entre los siguientes:

- Con punto decimal, double PI = 3.1416;
- Notación científica, float f = 4.53e+21;
- Con el sufijo f, float real = 15.1364f;
- Con el sufijo d, double d = 16.98d;

Literales(...)


```
public class TiposSimples{
public static void main(String args[]) {
    byte b = 0x55;
    short s = 0x55ff;
    int i = 1000000;
    long l = 0xffffffffL;
    char c = 'a';
    float f = .25f;
    double d = .00001234;
    boolean bool= true;
    ...
```

Literales(...)


```
    ...
    System.out.println("byteb = " + b);
    System.out.println("short s = " + s);
    System.out.println("inti = " + i);
    System.out.println("long l = " + l);
    System.out.println("charc = " + c);
    System.out.println("floatf = " + f);
    System.out.println("doubled = " + d);
    System.out.println("boolean bool= " + bool);
    }
}
```

Constantes

Una constante es una variable cuyo contenido no puede ser modificado una vez inicializado su valor. En Java se les llama variables finales. Una constante se declara usando el modificador final.

```
final tipo = valor;
```

Ejemplos:

```
final double PI = 3.1416;
final int MAX = 6;
```

Operaciones aritméticas

Java permite el cálculo de operaciones aritméticas básicas tales como:

- Suma (+)
- Resta (–)
- Multiplicación (*)
- División (/)
- Módulo (%)

Operaciones aritméticas

Ejemplos:

```
1   int  a = 2;
2   int  b = (a*50)/(4-a);
3   long  c = 8390L;
4   a = c/b;                  //¿Cuánto vale a?
5   double  d = 5.5;
6   float  f = d*2;           //¿Cuánto vale f?
```

Impresión en consola

La forma más elemental de presentación de información en consola, es utilizando la función

```
System.out.println(...);
```

Donde ... Es una cadena de texto o una concatenación de cadenas.

Ejemplo:

```
“esta es una cadena” + “concatenada”
“el valor de i es “ + i
```

Incremento/decremento

Al igual que C++, Java ofrece operadores para incrementar una variable en una cantidad dada.

Suponga la línea siguiente:

```
int i = 0;
Incremento en 1
i++; o ++i;    //equivalente a i = i+1;
Decremento en 1
i--; o --i;    //equivalente a i = i-1;
```

Incremento/decremento(...)

Incremento en n

```
i  +=  n;  //equivalente a i = i+n;
```

Decremento en n

```
i -=  n;  //equivalente a i = i-n;
```

Ejemplos:

```
1 int a = 20;
2 int b = 30;
3 int c = a++ + --b; //¿Cuánto vale c?
```

Promoción de datos (casting)

Hay situaciones donde se requiere ejecutar operaciones con diferentes tipos de datos (ej. Sumar un double con un int). Tales casos pueden generar ambigüedad.

Considere la siguiente operación:

double res = 1.5 + 3/2; ¿Cuánto vale res?

La respuesta es 2.5.

¿Por qué? Debido a que 3 y 2 son enteros, entonces el resultado es interpretado como un entero, por lo tanto, el resultado parcial sería 1 + 1.5 = 2.5. ¿Cómo hacer para que sea interpretado como un dato double?

Promoción de datos (...)

Existen dos tipos de *casting*:

Implícito

- Si uno de los operandos es double, entonces el otro operando es interpretado como doublé
- Si uno de los operandos es float, entonces el otro es interpretado como float
- Si uno de los operandos es long, entonces el otro operando es interpretado como long

Promoción de datos (...)

Explícito

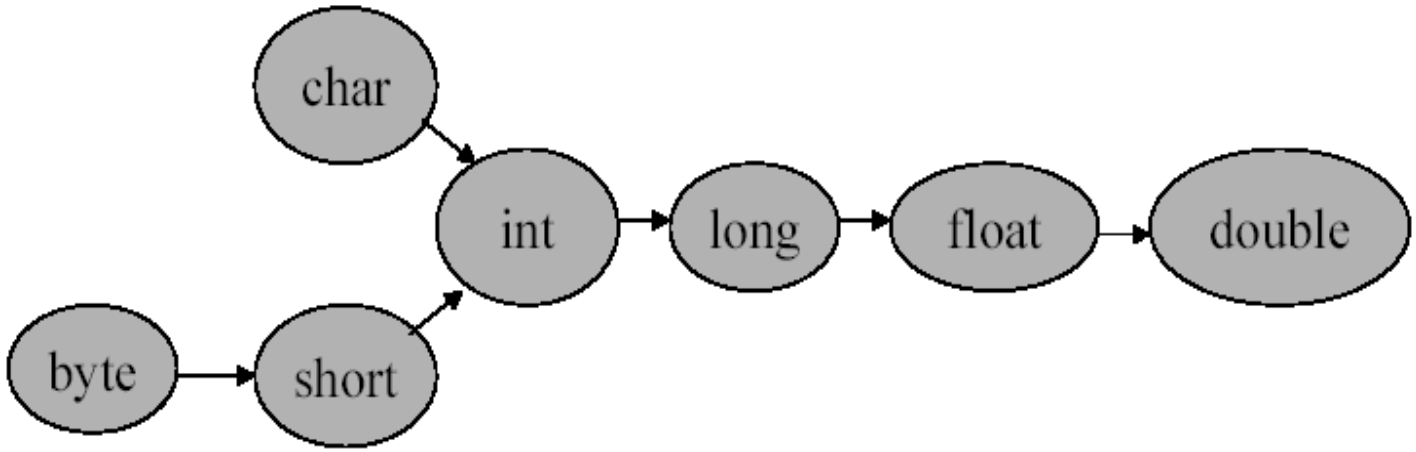
Se antecede el tipo de dato a la cual se quiere ajustar el resultado de una operación

```
res = 1.5 + ((double) 3)/2;
```

En este caso, `res` contiene el valor 3.0.

Promoción de datos (...)

El esquema de conversión se ilustra en la siguiente gráfica:



Residuo de una división

Para calcular el residuo de una división se utiliza el operador %.

```
1  Int  a  =  20 % 6;
2  int  b  =  a % 2;
3  double  res  / =  a  %  b / (a+b);
```

Operaciones a nivel de bits

A diferencia de lenguajes como Basic, Java ofrece operadores para ejecutar acciones a nivel de bits. Esto es, para modificar el contenido de una variable bit por bit.

Tales operadores son:

- &, AND lógico
- |, OR lógico
- ^, XOR lógico
- ~, negación o complemento
- >>>, corrimiento derecho con propagación de signo
- <<, corrimiento izquierda con 0s a la derecha
- >>, corrimiento derecho con 0s a la izquierda

Operaciones a nivel de bits(...)

Ejemplos:

```
byte  a  =  (byte)  0xFF;
a  =  (byte)  ~a|0x80;
a  >>>=  7;  // ¿Cuánto vale a?
```

Operaciones booleanas

Son expresiones que generan un valor booleano (true o false). Para conjuntar una expresión booleana es necesario utilizar operadores booleanos. En Java éstos son:

- – &&, AND lógica
- – || OR lógica
- – !, negación o diferencia
- – ==, igualdad
- – !=, desigualdad
- – >, mayor que
- – <, menor que
- – >=, mayor o igual
- – <=, menor o igual

Operaciones booleanas(...)

Ejemplos:

```
1   Boolean   b1 = (5 == 4);
2   Boolean   b2 = !b1;
3   b1   =   b1   ||   b2;
```

¿A qué es igual la expresión !(b1 || b2)?

```
!b1 && !b2
```

Sentencias de Selección

Son las que permiten redirigir el flujo del programa con base en el resultado de una evaluación de una expresión booleana (condición).

```
if (expresión-booleana) {
   bloque de código1;
} else {
   bloque de código2;
}
```

Si la expresión booleana se cumple entonces se ejecuta el bloque 1, en otro caso, el bloque 2.

Sentencias de Selección(...)


```
public class IfElse {
   public static void main(String[] args) {
      int mes = 4;
      String estacion;
      if (mes == 12 || mes == 1 || mes == 2) {
         estacion = "Invierno";
      } else if (mes == 3 || mes == 4 || mes == 5) {
         estacion = "Primavera";
      } else if (mes == 6 || mes == 7 || mes == 8) {
         estacion = "Verano";
      } else if (mes == 9 || mes == 10 || mes == 11) {
         estacion = "Otoño";
      } else {
         estacion = "Mes desconocido";
      }
      System.out.println("Abril está en " + estacion);
   }
}
```

Sentencias de selección(...)

Si se requiere efectuar una selección de entre varias opciones, y la opción es entera, entonces es conveniente usar la estructura switch.

```
switch (expresión) {
   case valor1:
      sentencias;
      break;
   case valor2:
      sentencias;
      break;
   default:
      sentencias;
      break;
}
```

Sentencias de Selección(...)

- La expresión debe ser del tipo byte, short, char o int.
- No puede ser long, ni float, ni double.
- Los argumentos de los casos (case) deben ser literales constantes.
- Si la expresión no se satisface en ninguno de los casos, entonces se ejecuta el código del segmento default.

Sentencias de selección(...)


```
public class Switch {
   public static void main(String[] args) {
      int mes = 4;
      String estacion;
      switch (mes) {
              case 12:
              case 1:
              case 2: estacion = "Invierno";
                      break;
              case 3:
              case 4:
              case 5: estacion = "Primavera";
                      break;
```

Sentencias de selección(...)


```
                case 6:
                case 7:
                case 8: estacion = "Verano";
                        break;
                case 9:
                case 10:
                case 11: estacion = "Otoño";
                         break;
                default: estacion = "Mes desconocido";
        }
        System.out.println("Abril esta en " + estacion);
    }
}
```

Sentencias de iteración

Las sentencias de iteración son mejor conocidas como bucles. En éstos se ejecuta un bloque de sentencias mientras una expresión booleana sea verdadera.

```
while (expresión-booleana) {
    sentencias;
}
```

Sentencias de iteración(...)


```
public class While {
    public static void main(String[] args) {
        int n=10;
        while (--n >= 0)
            System.out.println("No." + (n+1));
        }
    }
}
```

Sentencias de iteración(...)

Una variante de esta forma es definir hasta el final la expresión booleana.

```
do {
    sentencias;
} while (expresión-booleana);
```

El ciclo por lo menos se ejecuta UNA vez.

Sentencias de iteración(...)


```
public class DoWhile {
    public static void main(String[] args) {
        int n = 10;
        do {
            System.out.println("No." + n);
        } while (--n > 0);
    }
}
```

Sentencias de iteración(...)

Otra manera de declarar bucles es mediante la declaración de un rango de veces en el cual se efectúa un bloque de código.

```
for (inicialización;terminación;incremento) {
    sentencias;
}
```

Sentencias de iteración(...)


```
public class For {
   public static void main(String[] args) {
      for (int i=1; i<10; i++)
         System.out.println("i=" + i);
      }
   }
}
public class For2 {
   public static void main(String[] args) {
      int a,b;
      for (a=1,b=4; a<b; a++,b--)
         System.out.println("a=" + a);
         System.out.println("b=" + b);
      }
   }
}
```

Sentencias de iteración(...)

1. La parte de inicialización se ejecuta inmediatamente que se entra al ciclo.

2. La condición de paro o terminación se evalúa antes de entrar al cuerpo del ciclo.

3. La expresión de incremento se efectúa una vez que terminó de ejecutar el bloque del ciclo por primera vez.

Sentencias de paro

Un ciclo puede terminarse abruptamente o condicionarse su ejecución.

El uso de la sentencia break nos permite romper con la ejecución de un bucle.

```
ciclo {
    ...
    break;
    ...
}
```

Sentencias de paro(...)


```
public class Break {
   public static void main(String[] args) {
      int i;
      for (;;) {
         i++;
         if (i == 100)
            break;
      }
   }
}
```

Sentencias de paro(...)

Otra sentencia de control de ejecución dentro del bloque de un ciclo es la sentencia *continue*. Esta sentencia rompe con la ejecución del ciclo sin salirse del ciclo.

```
ciclo {
    ...
    continue;
    ...
}
```

Sentencias de paro(...)


```
public class For3 {
    public static void main(String[] args) {
        for (int i = 1000; i>0; i--) {
            if ((i%2) != 0) continue;
            System.out.println(i);
        }
    }
}
```

Arreglos

Un **arreglo** de datos primitivos es una colección ordenada de datos primitivos. Los arreglos son homogéneos, todos los datos del arreglo deben ser del mismo tipo.

Para crear un arreglo se siguen tres pasos:

- Declaración
- Construcción
- Inicialización

El alcance de una variable

Una variable tiene un período de vida que está determinado por el ámbito donde se declaró. Por lo tanto, el alcance o cobertura indica su visibilidad y usabilidad.

- Si la variable se declara en el ámbito de una clase (variable global), su cobertura es de toda la clase.
- Si la variable se declara en el ámbito de una función (variable local), su cobertura es solo la función.
- Si la variable se declara dentro de una sentencia (if, for, while, etc.), su cobertura es solo la sentencia.

El alcance de una variable(...)

Ejemplo:

```
int x;
...
if (x > 0) {
    int y = x + 2;
    ...
}
x = y + 3; //¿Cuánto vale la variable x?
```

Arreglos(...)

La *declaración* especifica al compilador cuál es el nombre del arreglo y de qué tipo es.
Ejemplos:

```
1 int[] enteros;
2 double[] dobles;
3 int[][] matriz;
4 char cadena[];
```

Como se ve, la declaración no define el tamaño del arreglo.

Arreglos(...)

La *construcción* del arreglo se hace en tiempo de ejecución y lo que hace es reservar espacio en memoria para ese conjunto de datos.

Ejemplos:

```
1 int[] enteros = new int[50]; //Arreglo de 50 enteros
2 int[][] matriz = new int[2][2];
```

Arreglos(...)

La *inicialización* es la asignación de valores a cada elemento del arreglo.

Ejemplos:

```
1 int[] enteros = {5,6,7}; //Arreglo de 50 enteros
2 int[] a = new int[2];
3 a[0] = 8;
4 a[1] = -2;
5 int[][] matriz = {{1,0},
6 {0,1}};
```

Arreglos(...)

Cuando no se inicializa el arreglo, los valores por defectos son:

- `int, short, byte en 0`
- `long en 0L`
- `float en 0.0f`
- `double en 0.0d`
- `char en '\u0000'`
- `boolean en false`

Arreglos(...)


```
public class ArreglosSimples {
    public static void main(String args[]) {
        int[][] matriz = new int[3][2];
        int[0][1] = “5”;
        int[0][2] = 6;
        int[1][3] = 0x50;
        int[1][0] = ‘a’;
        System.out.println(matriz[0]);
    }
}
```

Lo nuevo en JDK

Simplificación de ciclos en arreglos

Suponga el siguiente código:

```
int[] arr = new int[10];
for (int i=0; i<arr.length; i++) {
   arr[i] = i;
}
```

Lo nuevo en JDK(...)

El código es mejorado como sigue:

```
int[] arr = new int[10];
for (int i: arr) {
   arr[i] = i;
}
```

Lo nuevo en JDK(...)

La forma genérica es:

```
for (contador: arreglo) {
   //Código
}
```

Contador es el nombre o la declaración de la variable que funge como índice de arreglo.
Arreglo es el nombre del arreglo a recorrer.

Automáticamente el índice se incrementa en 1 y el ciclo termina hasta el último elemento del arreglo.