Universidad Nebrija

Escuela Politécnica Superior de Ingeniería
Departamento de Ingeniería Industrial

Fundamentos de la informática

3. El lenguaje de programación Java

Contenido

Introducción

Historia de Java

- **Java fue diseñado en 1990 por James Gosling, de Sun Microsystems**
- **Inicialmente, Java se utilizaba para programar dispositivos electrónicos de consumo como calculadoras, microondas y la televisión interactiva**
- **Java no fue diseñado para Internet porque nació antes de la era World Wide Web**

Introducción

Características de Java

- **Java es un lenguaje compilado que se interpreta con una máquina virtual (Java Virtual Machine)**
- **La máquina virtual interpreta el código Java (Bytecode) y hace que los programas Java se ejecuten en cualquier dispositivo**
- **Java surgió como lenguaje en 1995 y Netscape incluyó un interprete Java en su navegador**

Introducción

Compilación vs. interpretación

- En muchos lenguajes como C o C++ el programador desarrolla un programa que, una vez compilado es ejecutado por el sistema operativo
- En Java el código fuente se escribe en archivos con extensión .java. El compilador de Java traduce el código fuente a un archivo .class (Bytecode) que no es ejecutado por el sistema operativo. El archivo .class es interpretado por la máquina virtual Java (JVM)
- Existen compiladores de Java que traducen a código ejecutable .exe a partir del Bytecode. Esto incrementa la eficiencia pero el código no es portable

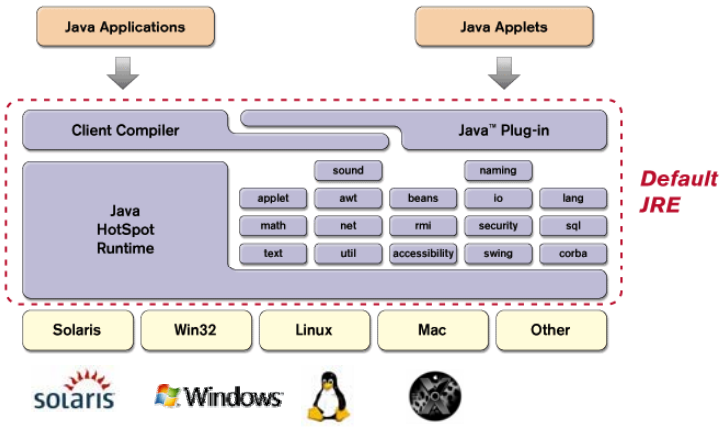
Introducción

Desarrollo de aplicaciones en Java

- El Java Development Kit (JDK) es el conjunto de herramientas que permiten desarrollar programas Java
  - Applet. Componente de una aplicación que se ejecuta en un navegador web
  - Servlet. Componente de una aplicación que se ejecuta en un servidor web (servidor de aplicaciones)
- Existen diferentes JDKs para distintos entornos
  - Java Standard Edition (SE) Estándar
  - Java Enterprise Edition (EE) Empresarial
  - Java Micro Edition (ME) Dispositivos móviles

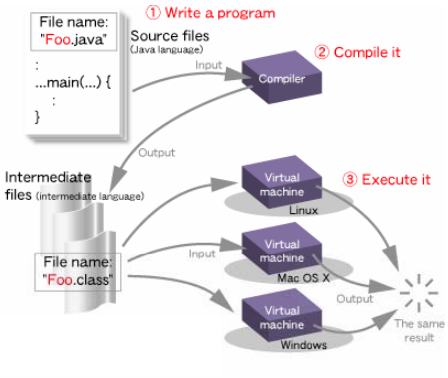
Introducción

Los programas Java son independientes de la plataforma



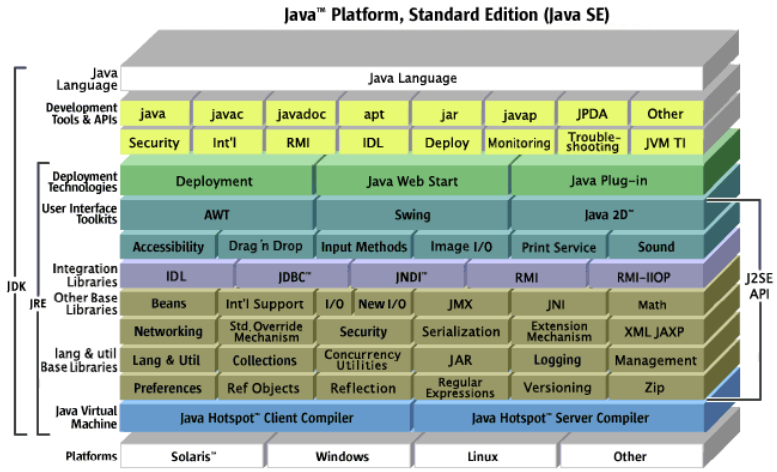
Introducción

Los programas Java producen el mismo resultado



Introducción

La plataforma Java SE



El lenguaje Java

Componentes lingüísticos de Java

- Desde un punto de vista lingüístico, un programa Java se compone de los siguientes elementos
  - Palabras reservadas
  - Identificadores
  - Signos de puntuación y símbolos
  - Operadores
  - Reglas sintácticas
- El compilador Java comprueba que se cumplen las restricciones léxicas y sintácticas del lenguaje

El lenguaje Java

Componentes lingüísticos de Java

```
public class PayrollApp {

    public static void main(String[] args) {
        int hours = 40;
        double payRate = 25.0, grossPay;

        grossPay = hours * payRate;

        System.out.print("Total nómina: ");
        System.out.println(grossPay + " euros");
    }
}
```


El lenguaje Java

Palabras reservadas de Java

- Las palabras reservadas (keywords) tienen un significado especial en el lenguaje.
- No pueden ser utilizadas para ninguna otra cosa que no sea para lo que han sido definidas
- En Java las palabras reservadas se escriben usando sólo letras minúsculas
- Ejemplos de palabras reservadas: public, class, static, void, int o double

El lenguaje Java

Identificadores

- **Los identificadores son palabras que utiliza el programador para dar nombre a programas, clases, variables o métodos**
- **En el programa de ejemplo se utilizan los siguentes identificadores**

PayRollApp, String y System para el programa y otras clases
args, hours, payRate, grossPay y out para las variables
main y println para los métodos

El lenguaje Java

Identificadores

- **En Java los identificadores que corresponden a las clases comienzan con letra mayúscula**
- **Los identificadores que corresponden a variables y a métodos se escriben comenzando con letra minúscula y usando una letra mayúscula al comienzo de cada nueva palabra**

```
int edad;
int totalAlumnos;
```

El lenguaje Java

Identificadores

- **Los siguientes son nombres válidos para variables**

payRate
Payrate
pay_rate
pay_1

- **Los siguientes nombres no son válidos**

pay rate (no se admite el espacio)
test#1 (el carácter # no es válido)
1stTest (comienza con un dígito)

El lenguaje Java

Identificadores

- **Java es un lenguaje que distingue entre mayúsculas y minúsculas. Los lenguajes que consideran distintas las letras mayúsculas de las minúsculas se denominan "case sensitive"**

- **Esto significa que los identificadores grossPay y GrossPay son distintos**

El lenguaje Java

Signos de puntuación y símbolos

- **Los signos de puntuación se utilizan para indicar el final de una instrucción o indicar el comienzo y el fin de un conjunto de instrucciones**
- **En Java las instrucciones terminan con punto y coma ‘;’**
- **En Java las cadenas de caracteres se delimitan utilizando comillas dobles: ”Hola Mundo”**
- **Además, se utilizan símbolos como las llaves ‘{’ y ‘}’, los paréntesis ‘(’ y ‘)’, los corchetes ‘[’ y ‘]’ y el punto ‘.’**

El lenguaje Java

Tipos de datos

- **Enteros:**
  - **Short**
  - **Byte**
  - **Int**
  - **Long**
- **Reales:**
  - **Double**
  - **Float**
- **Caractéres:**
  - **Char (Unicode)**
- **Booleanos:**
  - **Boolean**
- **Strings de caractéres:**
  - **String**

El lenguaje Java

Secuencias de escape en cadenas de caracteres

Secuencia	Nombre	Significado
\n	newline	Avanza el cursor a la siguiente línea
\t	tab	Avanza el cursor al siguiente tabulador
\b	backspace	Retrocede el cursor una posición
\r	carriage return	Mueve el cursor al inicio de la linea actual
\\	backslash	Imprime el carácter \
\'	single quote	Imprime ‘
\"	double quote	Imprime “

El lenguaje Java

Operadores aritméticos

- **Simples**

+ suma
- resta
*** producto**
/ división
% módulo o residuo

- **Aritméticos compuestos**

Operador	Operación	Equivale a
+=	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 lógicos

- **Simples**

la disyunción OR (||)
la conjunción AND (&&)
la negación NOT (!)

El lenguaje Java

Reglas sintácticas

- **Las reglas sintácticas indican cómo escribir correctamente los programas**
- **Las reglas definen el orden de los componentes léxicos en un cada línea de código del programa**
- **Un programa con errores de sintaxis no puede compilarse ni ejecutarse**

El lenguaje Java

Estructura de un programa

- **Un programa en Java debe tener por lo menos una clase**
- **Una clase almacena funciones o subrutinas, denominadas métodos. En un archivo fuente se puede tener más de una clase pero sólo una puede tener el atributo public**
- **Cuando un archivo fuente en Java contiene una clase pública, el nombre de la clase pública debe ser igual que el nombre del archivo**

El lenguaje Java

Estructura de un programa

```
public class PayrollApp {

    public static void main(String[] args) {
        int hours = 40;
        double payRate = 25.0, grossPay;

        grossPay = hours * payRate;

        System.out.print("Total nómina: ");
        System.out.println(grossPay + " euros");
    }
}
```

En este ejemplo la clase pública se llama PayrollApp, el archivo fuente se debe llamar PayrollApp.java

El lenguaje Java

Estructura de un programa

- La definición de una clase comienza con un encabezado que contiene la palabra reservada class
- Las declaraciones, las instrucciones y los métodos que pertenecen a una clase se delimitan entre llaves { } y forman el cuerpo de la clase (class body)
- Un método se compone de un conjunto de instrucciones que tienen una finalidad determinada

El lenguaje Java

Estructura de un programa

- Una aplicación Java tiene un método main que indica el punto de inicio del programa
- El método main siempre lleva el mismo encabezado

```
public class PayrollApp {

    public static void main(String[] args) {

    }
}
```


El lenguaje Java

Estructura de un programa

- Los métodos se componen de enunciados (statements) que representan las acciones a realizar
- Existen dos tipos principales de enunciados:
  - Enunciados de declaración
  - Enunciados ejecutables

El lenguaje Java

Estructura de un programa

- **Los enunciados de declaración (declaration statements) permiten indicar el tipo de datos, nombre y el valor inicial de una variable**
- **Los enunciados ejecutables (executable statements) permiten realizar acciones como:**
  - **Obtener un valor del usuario (input)**
  - **Asignar a una variable una expresión (process)**
  - **Mostrar un valor en la pantalla (output)**

El lenguaje Java

Application Programming Interface

- **Un Application Programming Interface (API) es una librería (library) que contiene clases para llevar a cabo determinadas funciones**
- **Las clases y los métodos del API de Java están disponibles para todos los programas en Java**
- **Por ejemplo, la clase System y los métodos print y println pertenecen al API de Java**

El lenguaje Java

Organización de un programa Java en paquetes

- **El código Java se organiza en paquetes. Cada paquete representa un bloque de una aplicación y tiene una función bien definida**
- **Un paquete agrupa clases Java y facilita su clasificación**
- **Existen jerarquías entre paquetes, para diferenciar el nivel se utiliza un punto**

```
package sistema.consola;
```


El lenguaje Java

Normas de codificación

- **Paquetes. La primera letra se escribe en minúsculas**

```
package consola;
```

- **Atributos y variables. La primera letra se escribe en minúsculas. Si el identificador está formado por varias palabras, la primera letra de la segunda palabra se escribe en mayúsculas**

```
int edad;
int totalAlumnos;
```

El lenguaje Java

Normas de codificación

- **Clases. La primera letra se escribe en mayúsculas**

public class HolaMundo

- **Métodos de clases. La primera letra se escribe en minúsculas. Si el identificador está formado por varias palabras, la primera letra de la segunda palabra se escribe en mayúsculas**

public static void main (String[] args)
public leerNotaAlumno()

El lenguaje Java

Documentación interna del programa

- **El código fuente de todo programa debe estar documentado con comentarios descriptivos y explicativos**
- **Los comentarios se usan para explicar lo que hace un programa. Son anotaciones que hace el programador para explicar el código fuente**
- **Los comentarios van dirigidos a los programadores que trabajan en el desarrollo de la aplicación Java**
- **El compilador Java ignora los comentarios de un programa**

El lenguaje Java

Documentación interna del programa

- **Existen tres tipos de comentarios en Java**
  - **Comentarios de una sola línea**
  - **Comentarios de más de una línea**
  - **Comentarios de documentación**
- **Los comentarios de documentación incluyen etiquetas que son procesadas por javadoc, se utilizan para describir las clases y los métodos**

El lenguaje Java

Documentación interna del programa

- **Los comentarios de una sola línea utilizan el símbolo //**

```
// Este es un comentario de una sola linea
```

- **Los comentarios de varias líneas se delimitan por /* y */**

```
/*
  * Este es un ejemplo de un comentario de varias líneas
  * en un programa Java
*/
```

El lenguaje Java

Documentación externa del programa

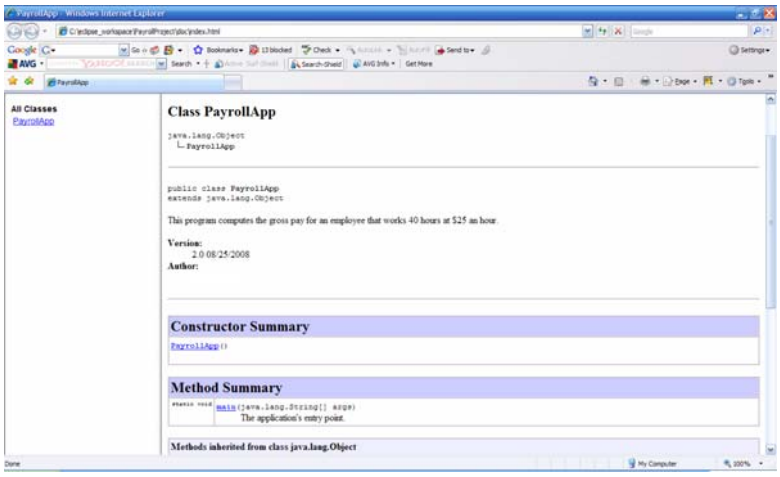
- Los comentarios de documentación inician con /**

```
/**
  * Este programa calcula el sueldo bruto mensual
  * @author Nebrija
  * @version 1.0, marzo de 2009
*/
```

- javadoc lee el código fuente y genera documentos HTML que documentan el código

El lenguaje Java

Documentación externa del programa



Datos primitivos en Java

Tipos de datos y capacidad de almacenamiento

- El tipo de dato indica qué tipo de valores puede almacenar una variable
- Los principales tipos de datos en los lenguajes de programación son:
  - Numérico: un número que se puede usar para cálculos matemáticos (p.e. -5.1, 0, 2.67, 9)
  - Alfanumérico: uno o más símbolos, incluyendo letras y números que no se usan en cálculos matemáticos (p.e. 'a', 'A', '€', "hola")
  - Lógico o booleano: valores true y false

Datos primitivos en Java

Tipos de datos numéricos

- Los datos numéricos se pueden clasificar en enteros y números reales
- En Java se utilizan los siguientes tipos de datos para representar números enteros: byte, short, int y long
- En Java se utilizan los siguientes tipos de datos para representar números reales: float y double

Datos primitivos en Java

Tipos de datos numéricos enteros y de coma flotante

- **Cada tipo de dato tiene una capacidad de almacenamiento diferente. La elección del tipo de dato depende de los datos que necesitemos almacenar**

Tipo	Bytes	Dominio
byte	1 (8 bits)	-128 a 127
short	2 (16 bits)	-32.768 a 32.767
int	4 (32 bits)	-2.147.483.648 a 2.147.483.647
long	8 (64 bits)	+/- $9x10^{18}$
float	4 (32 bits)	3,4 e-38 hasta 3,4e+38
double	8 (64 bits)	1,7 e-308 hasta 1,7e+308

Datos primitivos en Java

Tipos de datos numéricos enteros y de coma flotante

- **Ejemplos de declaraciones de variables de tipo numérico**

```
int edad, cursoAcademico;
int totalLibros = 0;
double sueldo = 1500.75;
double totalFactura = 0.0, precioUnitario;
```

- **Java permite declarar varias variables en la misma sentencia. Además, se puede indicar el valor inicial de una variable**

Datos primitivos en Java

Tipo de dato lógico

- El tipo de dato lógico sólo puede almacenar los valores falso o verdadero
- El tipo boolean de Java almacena valores false y true

Datos primitivos en Java

Tipo de dato carácter

- En Java los caracteres se almacenan utilizando 16 bits
- Este formato es un estándar internacional denominado Unicode y tiene capacidad para almacenar 65.537 caracteres
- El formato Unicode permite almacenar caracteres latinos y arábigos o cirílicos

Variables y constantes

Uso de variables

- **Una variable almacena valores de distinto tipo en la memoria que se utiliza durante la ejecución de un programa**
- **Por ejemplo, un programa Java que calcula el área de una parcela debe multiplicar el largo por el ancho. Si el largo de la parcela es 60 metros y su ancho es 70 metros, entonces el área sería 4.200 m^2**
- **El uso de variables es necesario para almacenar temporalmente los datos de cualquier programa. Una variable debe declararse antes de utilizarla**

Variables y constantes

Variables e identificadores

- **El nombre de una variable identifica una dirección de la memoria principal (RAM) donde se almacena el valor de la variable**
- **Se recomienda utilizar nombres significativos para las variables para saber para qué sirve una variable**
- **Las constantes representan valores que no cambian durante la ejecución de un programa**

  (p.e. final double PI = 3.14159265)

Variables y constantes

Declaración de variables

- **Una variable se identifica por un carácter seguido de cero o más caracteres o números**
- **El identificador es el nombre de una variable, se utiliza para hacer referencia al valor almacenado en la variable durante el programa**

nombre
apellidos
domicilio
codigoPostal
ciudad

Variables y constantes

El ámbito de una variable

- **El ámbito de una variable u objeto es el espacio del programa en el que esa variable existe. Se denomina ámbito de vida o alcance**
- **De forma general, el ámbito de vida de una variable comienza con su declaración y termina en el bloque en el que fue declarada, delimitado por {}**

Operadores

Operadores

- **Los operadores son símbolos que representan operaciones aritméticas o lógicas**

+ suma
- resta
*** producto**
/ división
% módulo o residuo

- **El operador + se utiliza para sumar dos números y para concatenar dos cadenas de caracteres, es decir, es un operador sobrecargado**

Operadores

Operadores

- **Existe un operador muy importante que se utiliza para asignar valores a las variables**

- **El operador = requiere una variable a la izquierda y una expresión a la derecha**

```
grossPay = hours * payRate;
```

- **Este operador, llamado operador de asignación, evalúa la expresión de la derecha y asigna el resultado a la variable de la izquierda**

Operadores

Operadores compuestos

- **Los operadores compuestos combinan la operación de asignación con operación aritmética**
- **Se utilizan para abreviar las operaciones aritméticas y hacer más claro el código Java**

Operador	Operación	Equivale a
+=	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

Operadores compuestos

- **Ejemplos de uso de los operadores compuestos y su significado**

Operación	Equivale a	Significado
x+=5;	x = x + 5;	suma 5 al valor de x
y -=2;	y = y - 2;	resta 2 al valor de y
z*=10;	z = z * 10;	multiplica por 10 el valor de z
a/=b;	a = a / b;	divide a por b
c%=3;	c = c % b;	calcula el módulo de c / 3

Operadores

Operadores lógicos

- **Java utiliza tres operadores lógicos: la disyunción (OR), la conjunción (AND) y la negación (NOT!)**
- **Los operadores lógicos OR (||) y AND (&&) se utilizan para evaluar expresiones lógicas**
- **El operador NOT (!) es un operador unario que devuelve la negación de una expresión lógica**

Operadores

Operadores lógicos y orden de precedencia

Operador	Símbolo Java	Descripción
NOT	!	Negación. Operador unario que convierte un valor verdadero en falso y un valor falso en verdadero
AND	&&	Conjunción. Operador n-ario que devuelve verdadero si todos los operandos son verdaderos y falso en cualquier otro caso
OR	||	Disyunción. Operador n-ario que devuelve falso si todos los operandos son falsos y verdadero en cualquier otro caso

Operadores

Operadores lógicos OR y AND

a	b	OR (a || b)	AND (a && b)
Falso	Falso	Falso	Falso
Falso	Verdadero	Verdadero	Falso
Verdadero	Falso	Verdadero	Falso
Verdadero	Verdadero	Verdadero	Verdadero

Operadores

Operadores de comparación

Operador	Descripción	Ejemplo
=	Igual	nota = 10
<	Menor que	nota < 5
>	Mayor que	nota > 9
<=	Menor o igual	nota <= 7
>=	Mayor o igual	nota >= 5
<>	Distinto de	nota <> 0

Operadores

Orden de precedencia de los operadores

Operador	Descripción
!	Negación
* / %	Producto, división, módulo
+ -	Suma, resta
< > <= >=	Menor, mayor, menor o igual, mayor igual
== !=	Igual, diferente
&&	AND
||	OR
= += -= *= /= %=	Asignación y operadores combinados de asignación

Conversión de Tipos

Conversión de tipos de datos

- **Para que se almacene un valor en una variable es necesario que el tipo del valor sea compatible con el declarado para la variable**
- **Java hace algunas conversiones automáticamente de forma que el valor y la variable donde se almacena sean compatibles excepto si esto puede ocasionar perdida de datos. Por ejemplo, los siguientes enunciados no causan problemas**

```
int number = 60;
double value = number
```

Conversión de Tipos

Conversión de tipos de datos

- **A continuación se muestran dos enunciados que sí causan problemas**

```
double number = 6.579;
int value = number;
```

- **En este caso no se puede asignar el valor de number a value a menos que:**

Se redondee el valor 6.579 a 7

Se descarte la parte fraccionaria de 6.579 para convertirlo a 6

Conversión de Tipos

Conversión de tipos de datos

- **En Java los tipos de datos guardan una jerarquía (orden de precedencia)**

- **La conversión de un valor de menor jerarquía a uno de mayor jerarquía (widening) se realiza automáticamente**

```
double   Mayor jerarquía
float
long
int
short
byte  Menor jerarquía
```

Conversión de Tipos

Conversión de tipos de datos

- **La conversión de un valor de mayor jerarquía a uno de menor jerarquía (narrowing) causa un error de sintaxis a menos que el programador fuerce la conversión haciendo "type casting"**

- **En los siguientes enunciados se utiliza el type casting para convertir el valor 6.579 a entero truncando y almacenando el nuevo valor en value**

```
double number = 6.579;
int value = (int)number;
```


Conversión de Tipos

Conversión de tipos de datos

- **¿Qué valor se almacena en la variable average?**

```
int exam1 = 97, exam2 = 88, exam3 = 93;
double average = (exam1 + exam2 + exam3) / 3;
```

- **El promedio es 92.66667, pero se almacena 92.0 porque la división entre valores enteros descarta la parte fraccionaria del resultado. Para resolver este problema se puede usar type casting**

```
int exam1 = 97, exam2 = 88, exam3 = 93;
double average = (double)(exam1 + exam2 + exam3) / 3;
```

Tipos de datos

La clase String

- **Una variable que almacene una cadena de caracteres debe ser declarada de tipo String. Una cadena de caracteres puede ser un nombre, una o varias letras, una frase, una oración o cualquier combinación de caracteres entre comillas dobles**
- **String es una clase, no es un tipo primitivo de datos. String es un tipos de dato compuesto a partir de tipos de datos primitivos (char)**
- **Los tipos primitivos de datos de Java son: short, byte, int, long, float, double, char y boolean**

Tipos de datos

La clase String

- **La declaración de una variable de tipo String**

```
String holaMundo = "Hola mundo";
String holaMundo = new String("Hola mundo");
```

- **Concatenación de cadenas de caracteres**

```
String hola = "Hola";
String mundo = "mundo";
String holaMundo = hola + " " + mundo;
```

Tipos de datos

La clase String

- **El total de caracteres de una variable de tipo String**

```
String holaMundo = "Hola mundo";
System.out.println("Caracteres: " + holamundo.Length());
```

- **Los caracteres de una cadena**

```
String holaMundo = "Hola mundo";
char carácter = holaMundo.charAt(1);
```


Tipos de datos

Comparación de cadenas de caracteres (objetos String)

- **Los operadores relacionales no son válidos para comparar objetos de tipo String, es necesario usar los métodos compareTo o equals de la clase String**
- **Los métodos equals y compareTo distinguen ente letras mayúsculas y minúsculas, son métodos "case sensitive"**
- **Para comparar cadenas de caracteres sin distinguir mayúsculas y minúsculas se usan los métodos equalsIgnoreCase y compareToIgnoreCase**

Tipos de datos

Ejemplo de comparación de cadenas de caracteres

```
public class ComparacionStrings {

   public static void main(String [] args){

      String nombre1 = "Juan", nombre2 = "Luis";

      // Comparación con el método equals

      if (nombre1.equals(nombre2))
         System.out.println(nombre1 +
            " es igual a" + nombre2);
      else
         System.out.println("Los nombres son diferentes");

    }
}
```


Tipos de datos

Arrays o matrices

- **Un array es un conjunto ordenado de variables u objetos que tienen el mismo nombre y tipo de dato**
- **Declaración de variables de tipo array**

// declaración utilizando new

int notas[] = new int[10];

//declaración con valores iniciales

int notas[] = {5, 7, 8, 7, 9}

Tipos de datos

Arrays o matrices

- **El número de elementos de un array se obtiene con el método length()**

System.out.println(notas.length());

- **Si se accede a un elemento fuera del rango del array se produce la excepción ArrayIndexOutOfBoundsException**

Control del flujo de un programa

Estructura condicional if else

- **Una instrucción if que contiene, en la parte verdadera o en la falsa otra estructura condicional se conoce como if anidado**

- **La sintaxis de un if anidado es:**

```
if (expresion)
   instruccion o bloque de instrucciones
else
   if (expression)
      instruccion o bloque de instrucciones
```

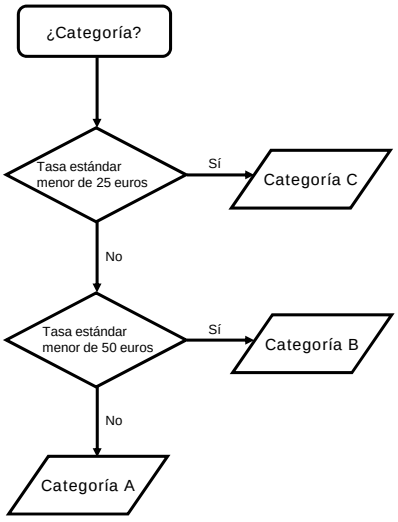
Control del flujo de un programa

Estructura condicional if else

- **Hay que tener cuidado al construir un if anidado. El compilador empareja cada else con el if anterior más próximo**
- **Si el código de un if es un bloque de instrucciones es necesario utilizar las llaves para delimitan el inicio y el fin del bloque. Si el código del if es una sola instrucción, no es necesario utilizar llaves para delimitar el bloque de instrucciones**
- **Es recomendable alinear la instrucción else para identificar fácilmente el if al que corresponde**

Control del flujo de un programa

Diagrama de flujo



Control del flujo de un programa

Código Java

```
public class Categorias {

   public static void main(String[] args) {
      int tasaEstandar = 20;

      if (tasaEstandar < 25)
         System.out.println("Categoría C");
      else
         if (tasaEstandar < 50)
            System.out.println("Categoría B");
         else
            System.out.println("Categoría A");
   }
}
```


Control del flujo de un programa

Operadores lógicos

- **Java utiliza tres operadores lógicos: la disyunción (OR), la conjunción (AND) y la negación (NOT!)**
- **Los operadores lógicos OR (||) y AND (&&) se utilizan para evaluar expresiones lógicas**
- **El operador NOT (!) es un operador unario que devuelve la negación de una expresión lógica**

Control del flujo de un programa

Operadores de comparación

Operador	Descripción	Ejemplo
=	Igual	nota = 10
<	Menor que	nota < 5
>	Mayor que	nota > 9
<=	Menor o igual	nota <= 7
>=	Mayor o igual	nota >= 5
<>	Distinto de	nota <> 0

Control del flujo de un programa

Evaluación "Short circuit" de expresiones lógicas

- **La evaluación "Short Circuit" calcula el valor lógico de una expresión realizando el número mínimo posible de cálculos**
- **Si un operando de una expresión lógica construida con AND (&&) es falso, entonces el resultado es falso**
- **Si un operando de una expresión lógica construida con OR (||) es verdadero, entonces el resultado es verdadero**

Control del flujo de un programa

El operador condicional ?

- **El operador condicional(?) es un operador ternario porque utiliza tres operandos**
- **Permite escribir un enunciado simple de tipo if else**

```
condicion ? parte-verdadera : parte-falsa
```

```
num = x > 100 ? 20 :50;
```

```
System.out.print("Nota: " +
   (nota>=5 ? "Aprobado" : "suspenso"));
```


Control del flujo de un programa

Estructura condicional switch

- **Un switch permite usar una variable o expresión de tipo ordinal para determinar el bloque de instrucciones a ejecutar**
- **Un switch puede evaluar una variable de tipo carácter o entero y tomar decisiones en base a su valor o contenido**

Control del flujo de un programa

La sintaxis del switch

```
switch (expresion) {
  case valor1:
    instruccion-o-bloque-de-instrucciones
    break;
  case valor2:
    instruccion-o-bloque-de-instrucciones
    break;
  case valor-n:
    instruccion-o-bloque-de-instrucciones
    break;
  default:
    instruccion-o-bloque-de-instrucciones
}
```


Control del flujo de un programa

Ejemplo del switch

```
public class Tasas {

   public static void main(String[] args) {
      char categoria = 'A';

      switch (categoria) {
         case 'A': System.out.println("Tasa >= 50");
                   break;
         case 'B': System.out.println("Tasa < 50");
                   break;
         case 'C': System.out.println("Tasa < 25");
                   break;
         default: System.out.println("error");
      }
   }
}
```

Control del flujo de un programa

Estructura condicional switch

- Cada case debe contener una expresión o valor único
- Si el contenido de la variable o el valor de la expresión cumple un case, la ejecución del programa se transfiere al case se ejecutan las instrucciones que vienen a continuación hasta que se encuentra un break
- Si un case no tiene break, entonces la ejecución continúa en el siguiente case
- La sección default es opcional y se ejecutada si no se cumple algún case

Estructuras iterativas

Estructuras iterativas o bucles

- Una estructura iterativa o bucle es una estructura de control que repite una o más instrucciones
- El cuerpo del bucle es el grupo de instrucciones que se repiten, la ejecución de este grupo de instrucciones se controla con una expresión lógica
- La evaluación de la expresión lógica puede realizarse antes o después del cuerpo del bucle. Si la expresión se evalúan antes el bucle se denomina "pre-test loop", en caso contrario es un "post test loop"

Estructuras iterativas

Estructuras iterativas y variables de control

- **Normalmente la condición de control del bucle depende del valor de una variable, denominada variable de control**
- **El valor de la variable de control determina el número de veces que se ejecuta el cuerpo del bucle, por lo que es necesario comprobar que:**
  - **La variable de control se inicializa correctamente antes de iniciar el bucle**
  - **La expresión lógica es adecuada para verificar el valor de la variable de control**
  - **Se actualiza la variable de control dentro del bucle**

Estructuras iterativas

Estructuras iterativas y variables de control

- **Un contador es una variable de control de un bucle. El contador debe inicializarse con el valor apropiado e incrementarse en el cuerpo del bucle. Si dentro del bucle no se modifica valor del contador, entonces el bucle puede ser infinito**
- **Para incrementar el contador se utiliza el operador de asignación y la variable a ambos lados de la expresión: contador = contador + 1**
- **En Java se puede utilizar los operadores ++ y += para acumular: contador+=1 o contador++**

Estructuras iterativas

Estructuras iterativas de Java

- **En Java existen tres estructuras iterativas:**

for
while
do while

- **El for controla la ejecución del cuerpo del bucle con una variable interna (contador), el while y el do-while utilizan una variable externa**

Estructuras iterativas

Estructura iterativa for

- **El for se repite un número determinado de veces, según del valor de la variable interna de control**

- **El for es un "pretest loop", evalúa la condición antes de ejecutar el cuerpo del bucle. El for permite inicializar la variable de control, evaluar la condición y actualizar la variable de control en una sola línea**

for (*inicializacion*; *evaluación*; *incremento*) {

instruccion o bloque de instrucciones

}

Estructuras iterativas

Estructura iterativa for

- **El for se compone de tres secciones: inicialización, evaluación e incremento de la variable de control**
- **La sección de inicialización permite establecer el valor inicial de la variable de control**
- **La sección de evaluación establece la expresión lógica que debe cumplir la variable de control para que se ejecute el cuerpo del bucle**
- **La sección de incremento permite modificar el valor de la variable de control**

Estructuras iterativas

Estructura iterativa for

- **La sección de incremento se ejecuta cada vez que finaliza la ejecución del cuerpo del bucle, antes de evaluar la variable de control**
- **Se debe evitar actualizar la variable de control dentro del cuerpo del bucle, para esto existe la sección de incremento del for**

Estructuras iterativas

Estructura iterativa for

```
public class Factorial {

   public static void main(String[] args) {
int num = 5, fac = 1;

for (int i=num; i>1; i--)
   fac = fac * i;

System.out.print("El factorial de " + num + " es " +
   fac);
   }
}
```


Estructuras iterativas

Estructuras iterativas anidadas

- Igual que es posible anidar estructuras condicionales if, también podemos anidar bucles
- Un bucle anidado contiene otro bucle, de manera que el bucle interior se ejecuta en cada iteración del bucle exterior

Estructuras iterativas

¿Cuántas veces se ejecuta System.out.print(j)?

```
for (int i = 1; i < = 5; i++)  {
   System.out.println("i: " + i);
   System.out.print("j: ");

   for (int j = 1; j < = 3 ; j++)
      System.out.print(j);

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


Estructuras iterativas

Estructura iterativa while

- **El while es una estructura iterativa "pre-test loop". La condición se evalúa antes de ejecutar el cuerpo del bucle**
- **Si la condición se cumple, se ejecuta el cuerpo del bucle, en caso contrario se ejecuta la instrucción siguiente al while**

```
while (condicion) {

   instruccion o bloque de instrucciones

}
```

Estructuras iterativas

Estructura iterativa while

```
Scanner entrada = new Scanner(System.in);

System.out.print("Introduzca un número entre 1 y 10: ");

num = entrada.nextInt();

while (num < 1 || num > 10) {
   System.out.println("Número no valido !!");

   System.out.print("Introduzca un número entre 1 y 10: ");

   num = entrada.nextInt();
}
```


Estructuras iterativas

Estructura iterativa do-while

- El do-while es una estructura iterativa "post-test loop". La condición se evalúa después de ejecutar el cuerpo del bucle
- El do-while se ejecuta al menos una vez

```
do {

   instruccion o bloque de instrucciones

} while (condicion);
```

Estructuras iterativas

Estructura iterativa do-while

```
Scanner entrada = new Scanner(System.in);

do {
   System.out.print("Introduzca un número entre 1 y 10: ");

   num = entrada.nextInt();

   if (num < 1 || num > 10)
      System.out.println("Número no valido !!");

} while (num < 1 || num > 10);
```


Estructuras iterativas

¿Qué tipo de estructura iterativa utilizar?

- El bucle no necesariamente se debe ejecutar: while
- El bucle debe ejecutarse al menos una vez: do-while
- El bucle se va a ejecutar un número determinado de veces: for

Estructuras iterativas

Las instrucciones break y continue

- **La instrucción break termina la ejecución de un bucle incondicionalmente**
- **El uso del break no se considera una forma adecuada de terminar un bucle**
- **La instrucción continue finaliza la ejecución de la iteración actual del bucle y continúa con la siguiente iteración. Igual que sucede con la instrucción break, puede hacer que el código sea mas difícil de entender y por esto se aconseja limitar su uso a situaciones muy concretas**

Input-Output

Operaciones de entrada y salida de Java

- **Cualquier programa Java que necesite realizar una operación Input-Output lo hace a través de un stream**
- **Un stream permite introducir o extraer información de dispositivos físicos como el teclado, la pantalla, una impresora, un archivo, etc.**
- **El paquete System define tres clases:**

in (entrada estándar)
out (salida estándar)
err (salida de errores estándar)

Input-Output

Entrada de teclado

- **Para una entrada de teclado es necesario crear un BufferedReader a partir de System.in**
- **Una vez creado el objeto de entrada para el teclado, se puede leer ejecutando el método readline() de la clase BufferedReader**
- **El método readLine() devuelve un String**

Input-Output

```
import java.io.*;

public class Nombre {

   public static void main(String args[]) throws
IOException {

      String nombre, continuar;

      BufferedReader entrada = new BufferedReader(new
  InputStreamReader(System.in));

      System.out.print("¿Cómo te llamas? ");
      nombre = entrada.readLine();
      System.out.printñn("Hola " + nombre");
   }
}
```