Aspectos Léxicos

- Comentarios

// comentario `hasta final de línea`

/* comentario */ `más de una línea`

```
/**
  Comentario útil para herramienta javadoc
*/
```

- Espacios en blanco, por claridad y como separadores
- Declaraciones e instrucciones separadas por ;
- Sensible a mayúsculas y minúsculas en identificadores:
  - un letra seguida de letras o dígitos, incluyendo raramente $ ó _
  - Convenio de nomenclatura: unaVariable, UnaClase, UNACONSTANTE, unMetodo(estoEsOtraVariable)

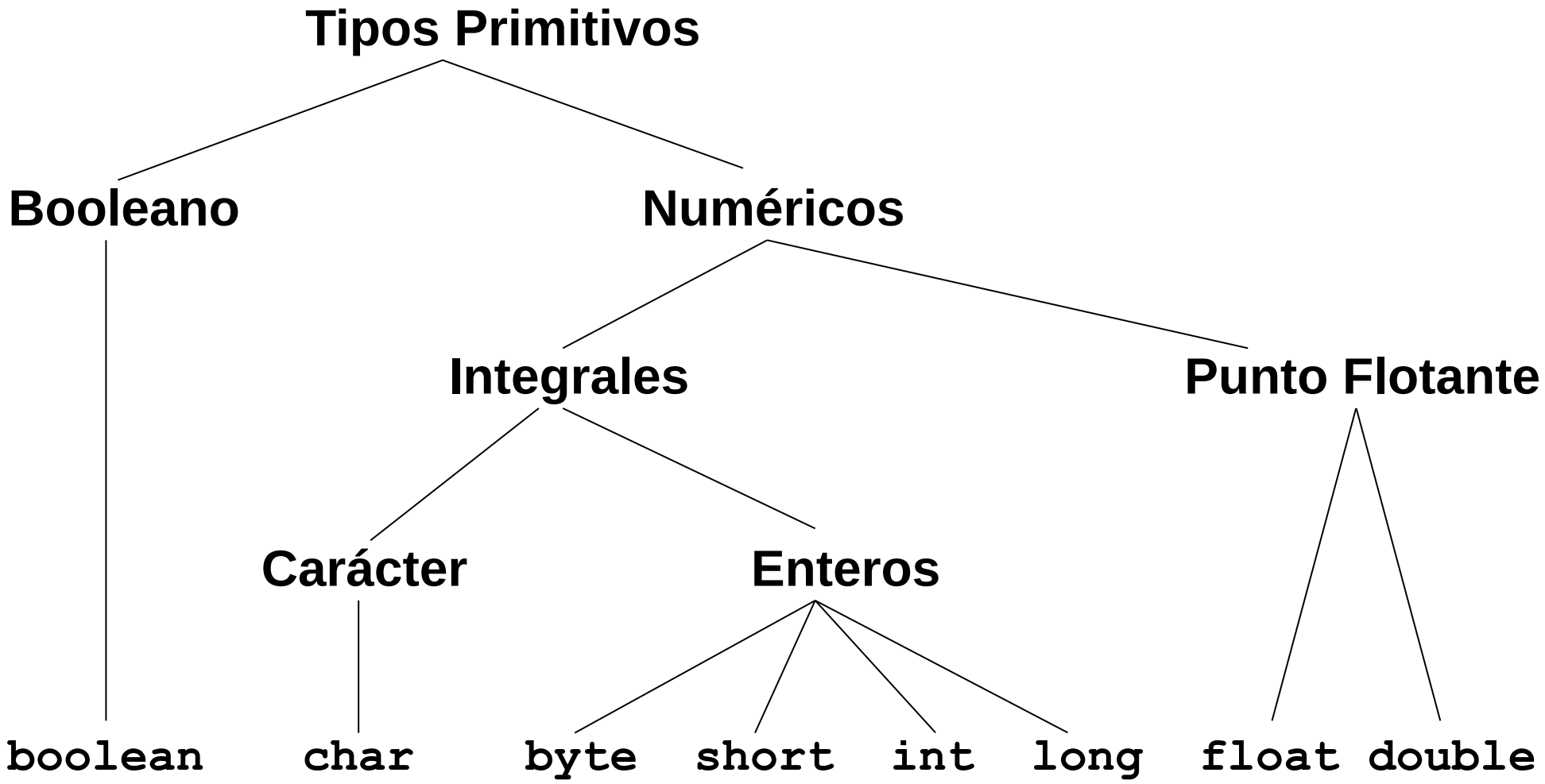
3.1. Introducción a Java (Apéndices)

- Presentación, orígenes, entorno
- Introducción mediante ejemplos
- **Elementos básicos del Lenguaje**
  - **Tipos de datos primitivos**
  - **Tipos de datos no primitivos**
  - **Instrucciones de control**
  - **Entrada/Salida**
  - **Aplicaciones ejecutables vía Web**
- Ejercicios

3.1. Introducción a Java (Apéndices)

- Presentación, orígenes, entorno
- Introducción mediante ejemplos
- **Elementos básicos del Lenguaje**
  - **Tipos de datos primitivos**
  - **Tipos de datos no primitivos**
  - **Instrucciones de control**
  - **Entrada/Salida**
  - **Aplicaciones ejecutables vía Web**
- Ejercicios

Tipos de Datos Primitivos



Tipos de Datos Primitivos

Tipos básicos o primitivos (tamaños fijos, portabilidad**)**

`byte`	1 byte	valores entre -128 y 127
`char`	2 bytes	(sin signo, caracteres Unicode, incluyen ASCII y más …)
`short`	2 bytes	valores entre -32768 y 32767
`int`	4 bytes	valores entre -2^{31} y $2^{31}-1$
`long`	8 bytes	valores entre -2^{63} y $2^{63}-1$
`float`	4 bytes	racionales con unas 6 cifras decimales significativas
`double`	8 bytes	con 15 cifras decimales
`boolean`		solo dos valores `true` y `false` (¡ojo! no numéricos)

Literales

`-81` `12345678901`**`L`** `0xBEBA` `010` (¡Ojo! Ese `010` vale 8)

`2.5` `1.72F` `11.03125D` `'A'` `'\t'` `'\u005B'`

Variables

```
char unaLetra;        // declaración
// Una variable local NO se incializa con valor por defecto,
// System.out.println(unaLetra) da error

unaLetra = 'a';       // asignación

short x, y, z;        // declaración múltiple

double miSueldoMensual;        // vamos a calcularlo
double miSueldoAnual = 15000; // inicializa la variable
final int mesesPorAño = 12;    // ¡¡es una constante!!
// asignación de variable ya declarada

miSueldoMensual = miSueldoAnual / mesesPorAño;
boolean soyMilEurista;   // declaración entre instrucciones
soyMilEurista = miSueldoMensual < 1000;  // asignación
```

Compatibilidad entre tipos numéricos

```
byte b = -15;
//byte b = -152; // Error: valores entre -128 y 127
char c = 'a';  // también válido: c = 97;  pero menos claro
short s = 1024;
int i = 50000;
long l = 120000;   // ¡¡ojo con eso!! Es una ele minúscula, no 1
float f = 5.67f;   // la f es necesaria
double d = .1234;  // igual que 0.1234
double resultado = (f*b) + (i/s) - (d*s);
System.out.println ((f*b) + " + " + (i/s) + " - " + (d*s));
System.out.println ("resultado = " + resultado);
```

- Conversión automática

```
i = s;
d = b;
d = f;
d = l; //¡Pero ojo!
```

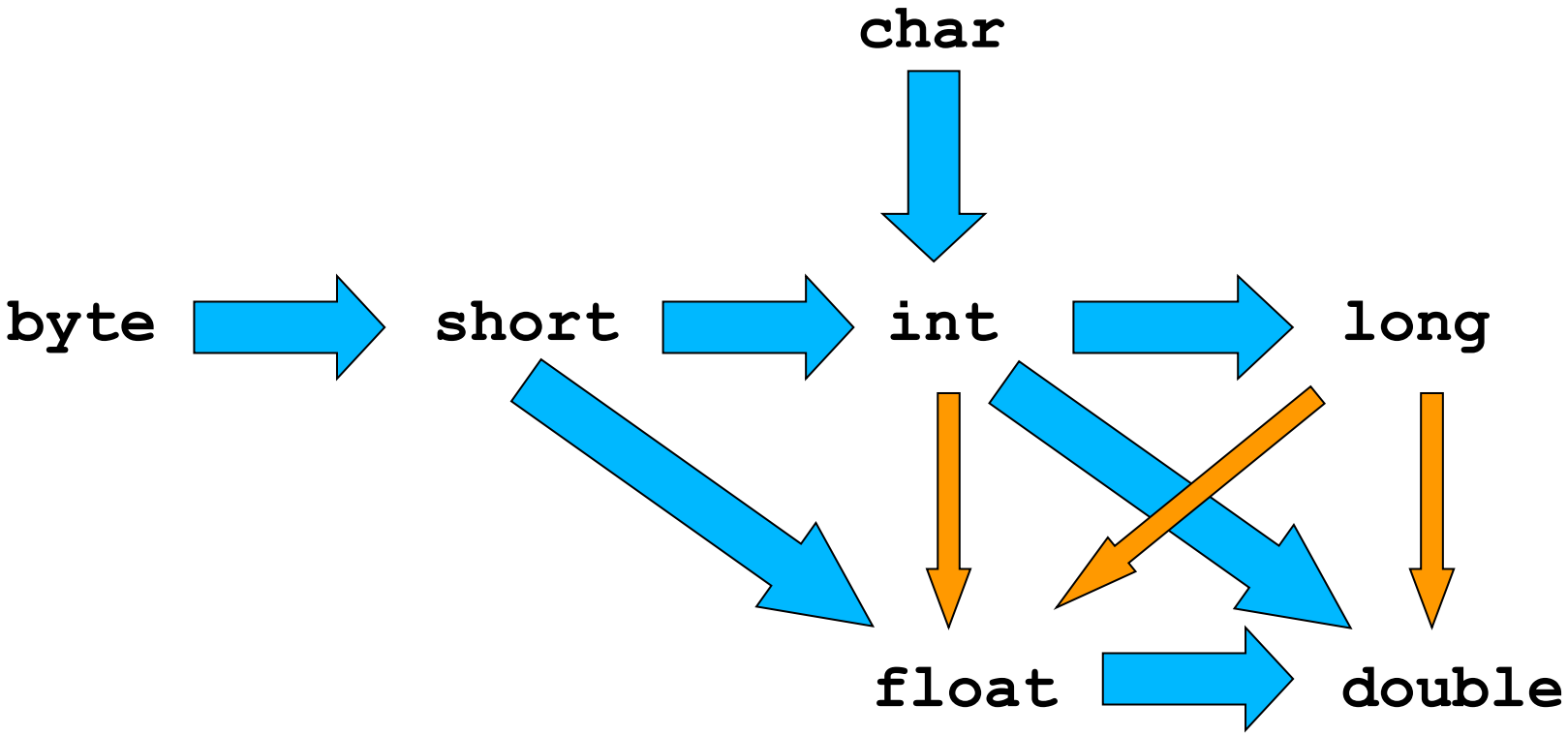
- Conversión *cast* explícito

```
s = (short) i;
f = (float) d;
i = (int) d;
b = (byte) f;
```

- char: *cast* explícito

```
s = (short) c;
c = (char) s;
c = (char) b;
```

Asignaciones con pérdida de precisión



Sin pérdida de información

Posible pérdida de información

Operadores

- 46 operadores
- Numéricos

  `+  -  *  /  %`

  `+=  -=  *=  /=  %=  --  ++`
- Lógicos

  `&  |  ^  !  &&  ||`
- Operadores de bits

  `&  |  ^  ~  <<  >>  >>>`
- Relacionales
  - Cualquier tipo: `==  !=`
  - Tipos numéricos: `>  <  >=  <=`
- Expresión condicional

  `(`*condición*`)? `*acción1*` : `*acción2*

Precedencia de operadores

Nivel de precedencia	Asociatividad
`[]` `new` `.` (*parametros*)	IZQDA a DCHA
`!` `~` `++` `--` `+`*expr* `-`*expr* `+` (*unario*) `-` (*unario*) (*tipo_o_clase*) *expr*	**DCHA a IZQDA**
`*` `/` `%`	IZQDA a DCHA
`+` `-`	IZQDA a DCHA
`<<` `>>` `>>>`	IZQDA a DCHA
`<` `<=` `>` `>=` `instanceof`	IZQDA a DCHA
`==` `!=`	IZQDA a DCHA
`&`	IZQDA a DCHA
`^`	IZQDA a DCHA
`|`	IZQDA a DCHA
`&&`	IZQDA a DCHA
`||`	IZQDA a DCHA
condición `?` *expr1* `:` *expr2*	**DCHA a IZQDA**
`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`	**DCHA a IZQDA**

Tipos No Primitivos (Reference Types)

- Se definen mediantes ***clases***

  Clases del propio lenguaje Java

  `String, Array, Enum, Thread, Exception,…`

  Clases de las librerías estándar

  `BigInteger, BigDecimal, File, List, Hashtable,…`

  Clases de librerías adicionales

  `JMenu, JWindow, SQLData, ImageIO, KeyGenerator,…`

  Clases definidas en la propia aplicación

  `Cuenta, CuentaCorriente, CuentaPlazoFijo, CuentaDeCredito, Tarjeta, TarjetaDeCredito, TarjetaDeDebito, TarjetaMonedero, Cliente,…`

- http://docs.oracle.com/javase/8/docs/api/index.html

3.1. Introducción a Java (Apéndices)

- Presentación, orígenes, entorno
- **Elementos básicos del Lenguaje**
  - **Tipos de datos primitivos**
  - **Tipos de datos no primitivos**
  - **Instrucciones de control**
  - **Entrada/Salida**
  - **Aplicaciones ejecutables vía Web**
- Ejercicios

Strings

- Las variables y literales ***Strings*** son Objetos de la clase java.lang.String (instancias de la clase).
- NO es un tipo primitivo, ni están terminados por \0, ni son un array de char.
- Al ser objetos, además de la cadena de caracteres, tienen métodos asociados:

  length() charAt(int) concat(String) compareTo(String) …

Strings

- Declaración del objeto de tipo String

```
String titulo;                    // inutilizables, falta crearlos
String nombre, apellido1, apellido2;
```

- Creación, reserva de memoria, inicialización y asignación

```
String autor = "Saramago";
titulo = "";                      // está creado, aunque con cadena vacia
apellido1 = autor;                // ¡Ojo! No se copia el contenido
```

- Métodos de acceso

```
char inicial = autor.charAt(0);          // inicial vale 'S'
int t = titulo.length();                 // t vale 0
```

- Errores

```
int e = nombre.length();      // Error: string no creado
autor[0]                      // Error: los strings no son arrays
```

Arrays

- Son *Objetos*.
  - Además del contenido del array, tienen un componente más `length`
- Colección indexada de elementos homogéneos:
  - Tipos primitivos o referencias.
  - Primer índice de un array `A` es 0, el último es `A.length-1`
- Arrays multidimensionales

- Declaración del objeto de tipo Array

```
int[] a;        // inutilizable, falta crearlo;
int otro[];     // esta sintaxis también es válida.
```

- Creación, reserva de memoria, inicialización

```
int[] b = new int[7];       // creado, todo con 7 ceros
char[] c = {'U', 'S', 'A'}; // creado e inicializado, {} ≈ new
byte[][] x = {{1,2},{},{3},{4,4,4,4}};  // array bidimensional
```

- Acceso al componente length:

```
int k = c.length;        // k vale 3
```

Arrays

- Acceso al contenido:

```
char n = c[0];          // n vale 'U'
int m = x[2][0];        // m vale 3, no se acepta[2,0]
```

- Errores:

```
int e1 = a.length;      // Error: array no creado
byte e2 = x[1][0];      // ArrayIndexOutOfBoundsException
```

- Asignación de arrays

```
int[] potencias;
int[] pares = {2,4,6,8};
potencias = pares;      //  No se copia el contenido
potencias[2] = 1;       // También cambia el array pares
```

- Pero se pueden copiar

```
int[] copia = new int[4];
System.arraycopy(pares,0,copia,0,pares.length);
int otro[] = pares.clone();
```

- No son strings pero se pueden convertir

```
char[] c = {‘J', ‘a', ‘v‘, ‘a’};
String lenguaje = new String(c);     // lenguaje es “Java"
```

Más utilidades en java.util.Arrays

3.1. Introducción a Java (Apéndices)

- Presentación, orígenes, entorno
- **Elementos básicos del Lenguaje**
  - **Tipos de datos primitivos**
  - **Tipos de datos no primitivos**
  - **Instrucciones de control**
  - **Entrada/Salida**
  - **Aplicaciones ejecutables vía Web**
- Ejercicios.

Instrucciones básicas en Java

- Muy similares a las de C … más alguna nueva
- **Bloques**: { … }
  - anidados, con variables locales, ámbito estático, y también con etiquetas
- **Condicionales**: if/else switch/case/break
- **Bucles**: while do/while for y un for mejorado
- **Saltos "estructurados"**: continue, break, ambos etiquetados
- Terminación y valor de retorno de método: **return**

Condicional: if

if (*condición*) *acción*$_1$ [else *acción*$_2$]

```
if ( a>b )
  if ( a>c ) { maximo = a; }  // llaves { } opcionales
  else { maximo = c; }
else
  if ( b>c ) { maximo = b; }
  else { maximo = c; }
```

```
if ( n == 0 ) {              // llaves { } imprescindibles
  if ( m == 0 ) System.out.println("Indeterminación");
}
else
  System.out.println("Resultado = "  + m/n);
```

Condicional: swith/case/break

```
switch ( expresión ) {  // tipo byte, short, char, int, enum
    case ec1: [ case ec2: … case eci: ] {
            instrucciones
            break;
    }
    …
    case ecj: … {
            instrucciones
            break;
    }
    default:
            instrucciones
            break;
}
```

Nota: ***Desde Java 7, se aceptan también objetos de tipo `String` como resultado de la expresión de control del `switch`.***

Los valores de casos ec_i son expresiones constantes sin duplicados

Iteraciones

```
while (condición) {
        ...
}
```

```
do {
   ...
} while (condición)
```

```
for (inicialización; condición; iteración) {
        ...
}
```

```
for (tipo variable : colección/array) {
  ...
}
```

(una colección también se puede iterar con .forEach(<l-exp>), que veremos más adelante.

```
String[] palabras = {"hi","hello","hola","eh!"};
for (String elemento : palabras) {
  System.out.println(elemento);
}
```

break con etiquetas

```
boolean cond = true;
a: {
b:   {
c:     {
          System.out.println(”Antes de break”);
          if (cond) break c; else break b;
          // System.out.println(”No se ejecutaria nunca”);
       }
       System.out.println(”Esto se ejecuta si cond true”);
     }
     System.out.println(”Después de b, se ejecuta siempre”);
   }
```

- Suele servir para manejar situaciones de error
- Pero es mucho mejor aprender a utilizar excepciones

continue con etiquetas

```
for (int i = 0; i<10; i++) {              // 0 1
        System.out.print( i + " ");       // 2 3
        if (i % 2 == 0) continue;         // 4 5
        System.out.println();             // 6 7
}                                         // 8 9
```

```
externo: for (int i = 0; i<10; i++) {          // 0
             for (int j = 0; j<10; j++) {      // 0 1
               if (i < j) {                    // 0 2 4
                  System.out.println();        // 0 3 6 9
                  continue externo;            // 0 4 8 12 16
               }                               // …
               System.out.print(" " + (i * j));
             }
}
```

Terminacion y valor de retorno: return

La novedad es que se usa en ***métodos*** porque ya no hay procedimientos ni funciones (pronto veremos la diferencia)

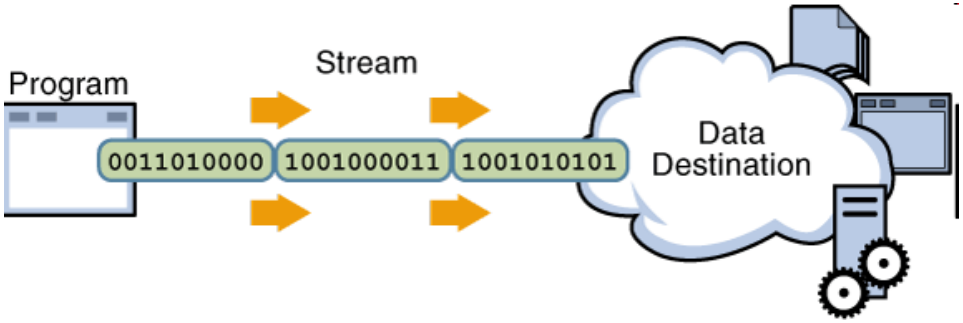
```
public class ClasePrincipal {
    public static void main(String[] args) {
        System.out.println("3! = " + factorial(3));
        return;
    }

    static long factorial(long n) {
        if (n == 0) { return 1; }
        else { return n * factorial(n - 1); }
    }
}
```

3.1. Introducción a Java (Apéndices)

- Presentación, orígenes, entorno
- **Elementos básicos del Lenguaje**
  - **Tipos de datos primitivos**
  - **Tipos de datos no primitivos**
  - **Instrucciones de control**
  - **Entrada/Salida**
  - **Aplicaciones ejecutables vía Web**
- Ejercicios

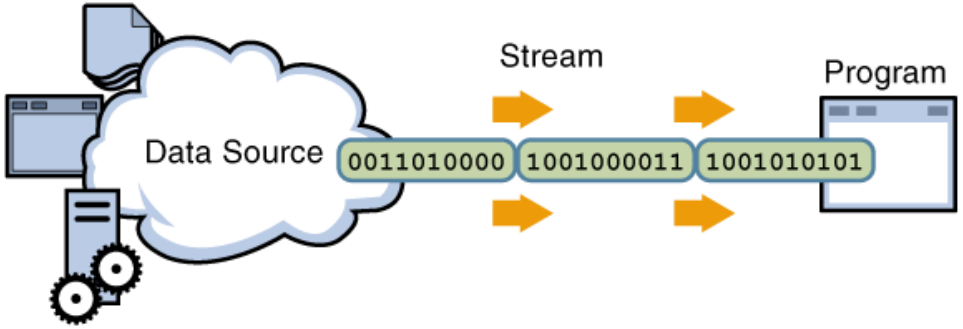
Streams de entrada/salida

- Los streams representan fuentes o destintos de datos.



Output Stream

- Encapsulan el tipo de fuente: fichero en disco, otra aplicación, un dispositivo, un array en memoria, un puerto de comunicación, etc.



Input Stream

Ejemplo básico de lectura (con un error)

```
import java.io.*;
public class DemoInput {
 public static void main(String[] args) {
  FileInputStream stream = new FileInputStream("input.txt");
  InputStreamReader reader = new InputStreamReader(stream);
  BufferedReader buffer = new BufferedReader(reader);
   String linea;
   while ((linea = buffer.readLine())!=null) {
    System.out.println("Linea leida " + linea);
   }
  stream.close();
 }
}
```

Errores de compilación: unreported *exception* **`java.io.IOException`**; must be *caught* or declared to be *thrown*

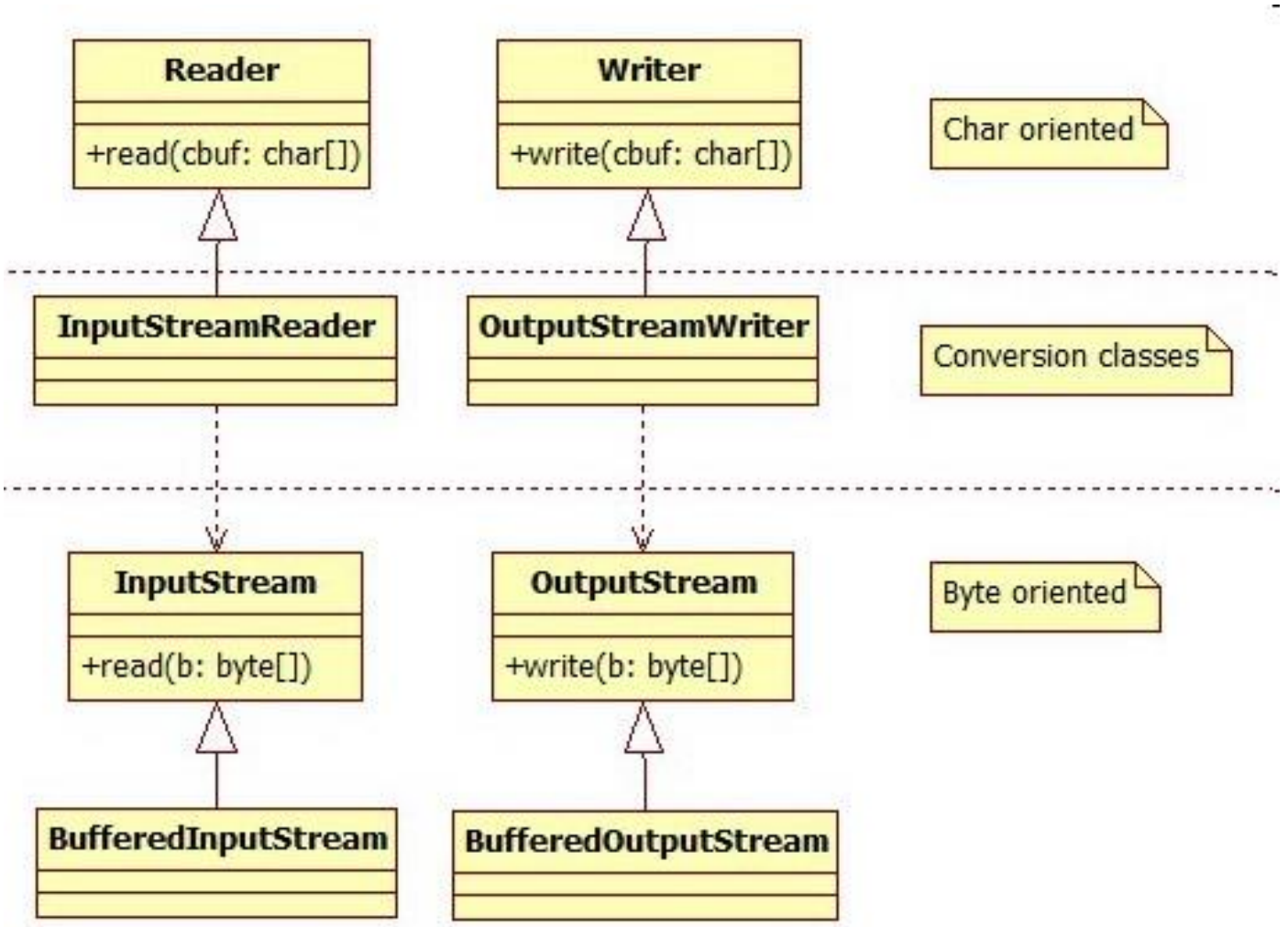
Ejemplo básico de lectura

```
import java.io.*;
public class DemoInput {
 public static void main(String[] args) throws IOException {
  BufferedReader buffer =
          new BufferedReader(
               new InputStreamReader(
                    new FileInputStream("input.txt")
               )
          );
  String linea;
  while ((linea = buffer.readLine())!=null) {
   System.out.println("Linea leida " + linea);
  }
  buffer.close();
 }
}
```

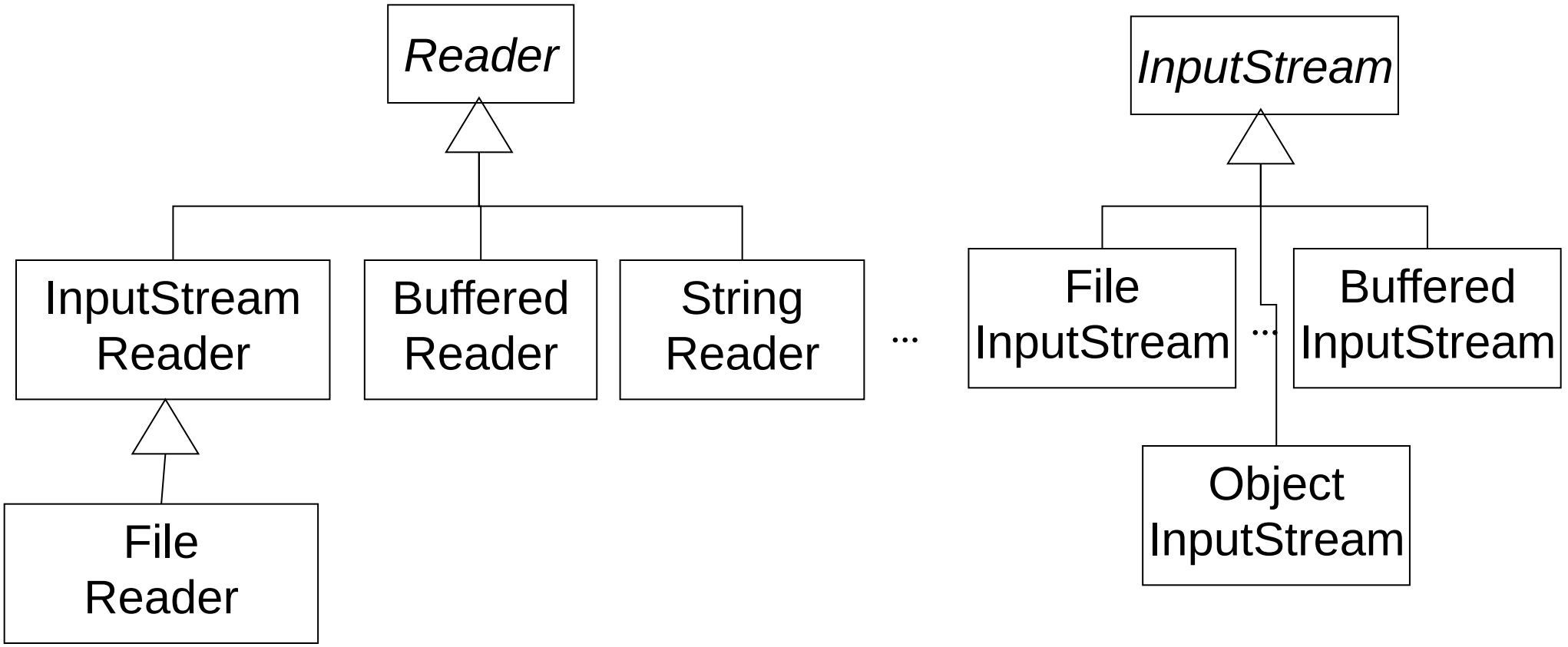
Ejemplo básico de lectura desde teclado

```
import java.io.*;
public class DemoInput {
 public static void main(String[] args)throws IOException {
  BufferedReader buffer =
          new BufferedReader(
               new InputStreamReader( System.in )
          );
  String linea;
  while ((linea = buffer.readLine())!=null) {
   System.out.println("Linea leida " + linea);
  } // Termina al introducir Ctrl+Z
  buffer.close();
 }
}  // o bien, leemos byte a byte ...  int i = System.in.read();
```

Clases Básicas I/O



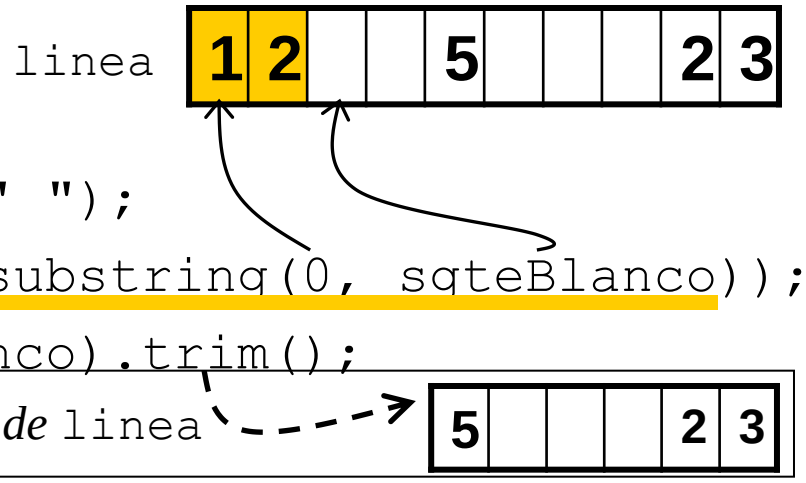
Clases Básicas I/O



Lectura de caracteres **Lectura de bytes**

Ejemplo básico de lectura desde teclado

Si *cada* **`linea`** contiene 3 números enteros separados por blancos, *pero sin blancos antes del primer número,* podemos sumar los tres números de cada línea así:




```
int sgteBlanco = linea.indexOf(" ");
int x = Integer.parseInt(linea.substring(0, sgteBlanco));
linea = linea.substring(sgteBlanco).trim();

sgteBlanco = linea.indexOf(" ");
int y = Integer.parseInt(linea.substring(0, sgteBlanco));
linea = linea.substring(sgteBlanco).trim();
int z = Integer.parseInt(linea);
System.out.println("Suma: " + (x + y + z) );
```

StreamTokenizer para extraer números

Sumar cada 3 enteros leídos (ignora blancos extra y saltos de línea)

```
import java.io.*;
public class DemoInput {
 public static void main(String[] args) throws IOException {
  FileInputStream stream = new FileInputStream("input.txt");
  InputStreamReader reader = new InputStreamReader(stream);
  StreamTokenizer tokens = new StreamTokenizer(reader);
  while (tokens.nextToken() != StreamTokenizer.TT_EOF) {
    int x = (int) tokens.nval;
    tokens.nextToken(); int y = (int) tokens.nval;
    tokens.nextToken(); int z = (int) tokens.nval;
    System.out.println("Suma: " + (x + y + z));
  }
  stream.close();
 }
} // ¿Qué pasa si en total hay  3n+1  ó  3n+2  números?
```

Tipo de token con `StreamTokenizer`

Ignoramos las palabras que preceden al primer número

```
while (tokens.nextToken() != StreamTokenizer.TT_EOF) {
 switch (tokens.ttype) {
  case StreamTokenizer.TT_WORD:
    System.out.println("ignoramos: " + tokens.sval);
    break;
  case StreamTokenizer.TT_NUMBER:
    int x = (int) tokens.nval;
    tokens.nextToken(); int y = (int) tokens.nval;
    tokens.nextToken(); int z = (int) tokens.nval;
    System.out.println("Suma: " + (x + y + z));
    break;
 }
}
```

Otra alternativa es usar la clase: **java.util.Scanner**

String Split

```
public static void main(String[] args) throws IOException {
 BufferedReader buffer = new BufferedReader(new InputStreamReader( System.in ));
 String linea;

  while ((linea = buffer.readLine())!=null) {
   String split[] = linea.split("\\s+");
   int suma = 0;
   for (String n : split)
     suma += Integer.parseInt(n);
   System.out.println("Suma="+suma);
  } // Termina al introducir Ctrl+Z
  buffer.close();
}
```

Salida de texto con formato por defecto

Si para imprimir números desde 0.15 a 0.20 hacemos esto:

```
public class DemoOutputTxt1 {
  public static void main(String[] args) {
    for (double i = 0.15; i <= 0.20; i = i + 0.01)
      System.out.println(i);
  }
}
```

Se obtiene la siguiente salida en pantalla:

```
0.15
0.16
0.17
0.18000000000000002
0.19000000000000003
```

Y no sale el 0.20 !?

Salida de texto formateada

Podemos controlar el formato de salida usando **printf**

```
import java.util.Date;
public class DemoOutputTxt2 {
  public static void main(String[] args) {
    for (double i = 0.15; i <= 0.20; i = i + 0.01)
      System.out.printf("%5.2f\n", i);
    System.out.printf("%tc\n", new Date());
  }
}
```

Se obtiene la siguiente salida en pantalla:

```
0,15
0,16
0,17
0,18
0,19
vie dic 03 12:18:04 CET 2010
```

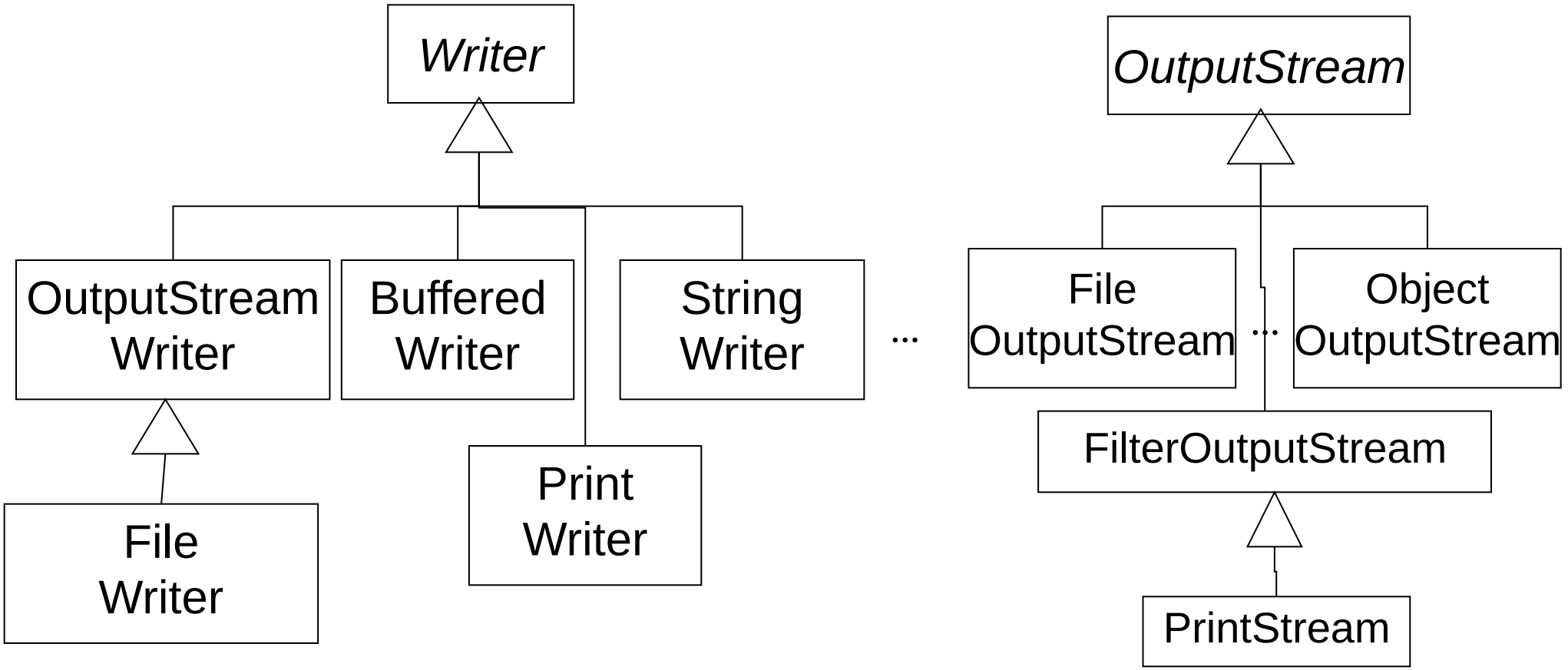
PERO sigue sin salir el 0.20 !?

Salida de texto en ficheros

Podemos controlar el formato de salida usando **printf**

```
import java.io.*;
import java.util.*;
public class DemoOutputTxt3 {
  public static void main(String[] args) throws IOException{
    FileOutputStream stream = new
                              FileOutputStream("numeros.txt");
    PrintWriter salida = new PrintWriter(stream);
    for (double i = 0.15; i <= 0.20; i = i + 0.01)
      salida.printf("%5.2f\n", i);
    salida.printf("%tc\n", new Date());
    salida.flush();
    stream.close();
  }
} // Se obtiene la misma salida de antes pero en numeros.txt
```

Clases Básicas I/O



Escritura de caracteres **Escritura de bytes**

Salida en formato interno de objetos

Escribimos todo un vector de puntos con un solo **`writeObject`**

```
import java.io.*;
import java.util.*;
public class DemoPuntos {
 public static void main(String[] args) throws IOException {
  ObjectOutputStream salidaObjetos =
        new ObjectOutputStream(
            new FileOutputStream("puntos.objectData") );
   Punto puntos[] = {new Punto(3,4),
                     new Punto(0,3), new Punto(2,6)};
  salidaObjetos.writeObject(puntos);
   salidaObjetos.close();
  }
}
```

Clases auxiliares para DemoPuntos

Para el ejemplo anterior necesitamos esta clase auxiliar

```
class Punto implements Serializable {
  private int x, y; // componentes privados

  public Punto (int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

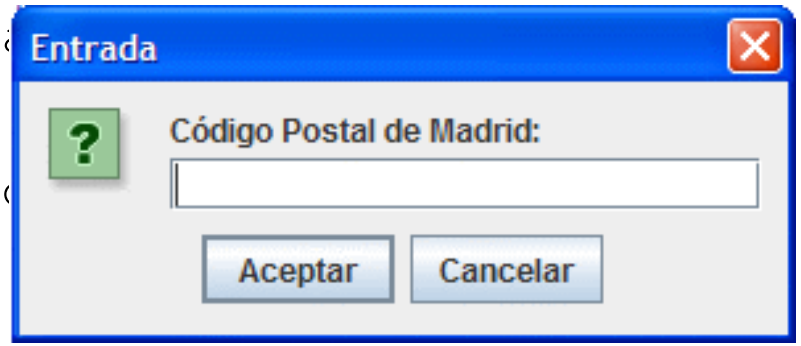
Entrada en formato interno de objetos

Leemos todo el vector de puntos con un solo **`readObject`**

```
import java.io.*;
import java.util.*;
public class DemoLeerPuntos {
  public static void main(String[] args) throws Exception {
     ObjectInputStream entradaObjetos =
         new ObjectInputStream(
           new FileInputStream("puntos.objectData") );
     Punto p[] = (Punto[])entradaObjetos.readObject();
     entradaObjetos.close();
     System.out.println("Leidos " + p.length + " puntos");
  }
}
```

Entrada con interfaz gráfica

```
import javax.swing.*;
public class DemoMiniGUI {
 public static void main(String[] args) {
  String codigoPostal =
      JOptionPane.showInputDialog("Código Postal de Madrid:");
  int cp = Integer.parseInt(codigoPostal);

  while (cp < 28000 || 28999 < cp) {
    System.out.println("C.P."+codigoPostal+"incorrecto");
    codigoPostal =
       JOptionPane.showInputDialog("Código postal de Madrid:");
    cp = Integer.parseInt(codigoPosta
   }
   System.out.println("C.P. " + codi
  }
}
```



Aplicaciones accesibles vía Web: applet

Aplicaciones (pequeñas) para ejecutar dentro de una página web

```
<HTML>
<HEAD>   <TITLE> Ver Fecha </TITLE> </HEAD>
<BODY>
   <APPLET CODE="VerFecha.class">
   </APPLET>
</BODY>
</HTML>
```

El navegador descarga el código Java compilado **`VerFecha.class`**

Y lo ejecuta (si tiene instalado el plug-in correspondiente, una JVM)

El código se pone en el servidor, pero lo ejecuta el cliente

`http://www.eps.uam.es/~eperez/VerFecha.html`

Aplicaciones accesibles vía Web: `applet`

Es sencillo convertir un aplicación estándar en un applet:

La clase principal debe definirse como subclase de **`JApplet`**

El método principal **(**antes **`main`)** en el applet es **`init()`**

Desde él se da control a la aplicación estándar

```
import java.util.*;
import javax.swing.*;
public class VerFecha extends JApplet {
    public void init() {
        JLabel label = new JLabel( (new Date()).toString() );
        add(label); // añade elemento a la interfaz gráfica
    }
}
```

3.1. Introducción a Java

- Presentación, orígenes, entorno
- Elementos básicos del Lenguaje
- **Ejercicios**

Ejercicio 0

¿Por qué no se imprime el valor **0,20** para **i** con este programa?

```
public class Rompecabezas0 {
 public static void main(String[] args) {
   for (double i = 0.15; i <= 0.20; i = i + 0.01)
      System.out.printf("%8.5f\n", i);
 }
}
```

En cambio, este otro sí que imprime hasta desde **0.0625** hasta **0.75**

```
public class Rompecabezas00 {
 public static void main(String[] args) {
   for (double i = 0.0625; i <= 0.75; i = i + 0.03125)
      System.out.printf("%8.5f\n", i);
 }
}
```

Ejercicio 1

¿Qué el valor de **`j`** imprime este programa?

```
public class BucleSorpresa {
 public static void main(String[] args) {
  int j = 0;
  for (int i = 0; i < 10; i++) {
     j = j++;
  }
  System.out.println(j);   // ¿Se imprimirá 10?
 }
}
```

Ejercicio 2

Buscamos un entero que no es igual que redondeándolo ¿Lo encontraremos?

```
public class RedondeoSorpresa {
 public static void main(String[] args) {
  int i = 0;
  while (Math.round(i) == i) {
      i++;
  }
  System.out.println("Sorpresa! " + i);
 }
}
```

Ejercicio 3

Querríamos contar cuántos enteros positivos puede manejar **`int`**

```
public class CuentaEnterosPositivos {
 public static void main(String[] args) {
  int contador = 0;
  for (int i = 1; i <= Integer.MAX_VALUE; i++) {
      contador++;
  }
  System.out.println(contador + " enteros.");
 }
}
```

Ejercicio 4

¿Qué imprime exactamente este programa?

(a) 5

(b) 2.5

(c) Nada. Se produce una excepción.

(d) Ninguna de las anteriores.

```
import java.io.*;
public class LocuraCondicional {
 public static void main(String[] args)
                                  throws IOException {
   int i  = System.in.read();
   boolean par =  i % 2 == 0;
   System.out.println( (par || !par) ? 5 : 2.5 );
  }
}
```

Ejercicio 5

¿Qué imprime este programa con un bucle **`while`** buscador de 0?

```
public class BuscaCeros {
  public static void main(String[] args) {
    int[] valores = {-2, 3, 0, -4, 0, 5};
    boolean hayCeros = false;
    int i = 0;
    while (i < valores.length) {
       hayCeros = hayCeros || valores[i++] == 0;
    }
    if (hayCeros)
       System.out.println("Hay algun cero.");
  }
}
```

Ejercicio 6

¿Qué imprime este programa con método **`Misterio`** tan simple?

```
class  Misterio {
  private int m;
  public void Misterio() { m = 13; }
  public int valor() { return m; }
}

public class MetodoSimple {
  public static void main(String[] args) {
    Misterio misterio = new Misterio();
    System.out.println( misterio.valor() );
  }
}
```

Hay muchos más rompecabezas

Rompecabezas producidos por la potencia y complejidad de Java

Cualquier lenguaje tan potente y complejo tendrá rompecabezas

La solución última a todos está en la *especificación* del lenguaje

`http://java.sun.com/docs/books/jls/`

Un documento extenso y complejo, pero minucioso y preciso

Difícil de asimilar por completo para muchos programadores

Pero de obligada comprensión y cumplimiento para quien

- programe el compilador de java (y la maquina virtual)
- programe grandes aplicaciones complejas en Java

Elementos del lenguaje

Ha sido inevitable anticipar algunos conceptos que se detallan en los siguientes temas

- Primer ejemplo completo y compilación.

  Control de visibilidad: public, private, … hay más.

- Declaraciones, asignaciones, operadores, expresiones, …

  Sobrecarga de operadores, polimorfismo, …

- Tipos de datos no primitivos: String, Arrays, …

  Objetos, clases, subclases, herencia, …

- Entrada/salida: formato texto, formato interno, interactiva

  Interfaces, clases abstractas,

  Librerías auxiliares, interacción gráfica con el usuario,

  Tratamiento de errores mediante excepciones, …