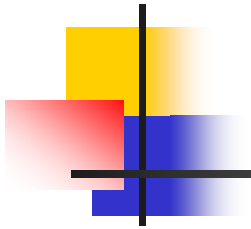
Programación Orientada a Objetos con Java: Anexos

Ingeniería del Software II
Curso 2008/2009

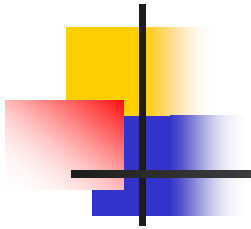
Sergio Ilarri Artigas
silarri@unizar.es

Parte del material está extraído de JavaSun y Java Tutorial:
Copyright 1994-2007 Sun Microsystems, Inc. All Rights Reserved.



Índice Detallado (I)

- Anexo I: Algunos Paquetes y Clases de Uso Común
  - Clase *Object*
  - Paquetes: *java.lang*, *java.io*, *java.util*, *java.net*
  - Manejo de Cadenas
  - Clases Numéricas
  - Arrays

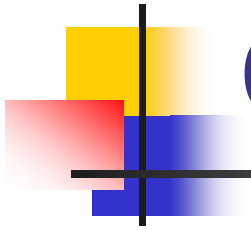


Índice Detallado (II)

- Anexo II: Flujos de E/S
  - Conceptos básicos
  - Clases básicas (paquete java.io)
  - Ejemplo de flujo de bytes/caracteres
  - Flujos con buffer
  - E/S por línea de comandos
  - Flujos de objetos
  - Objetos *File*

Anexo I:

Algunos Paquetes y Clases de Uso Común



Clase *Object* (I)

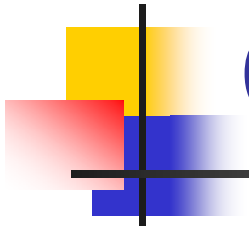
- *protected Object clone()*
  - Interface *Cloneable*
    (no implementado por Object)
  - *CloneNotSupportedException*
  - Comportamiento por defecto en Object:
    - Copia las variables miembro

Shallow copy vs. *deep copy*

```
aCloneableObject.clone();
```

Clase *Object* (II)

```
public class Stack implements Cloneable {
   private Vector items;
   …
   protected Object clone() {
       try {
           Stack s = (Stack) super.clone();
           s.items = (Vector) items.clone();
           return s;
      }
      catch (CloneNotSupportedException e) {
          throw new InternalError();
      }
   }
}
```



Clase *Object* (III)

- *Equals*

```
Integer one = new Integer(1),
    anotherOne = new Integer(1);
if (one.equals(anotherOne))
  System.out.println("objects are equal");
```

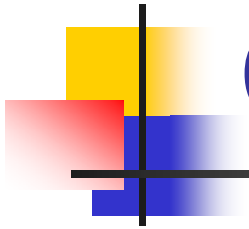
-Compara las identidades de los objetos (implementación por defecto)
-A veces, compara los contenidos
-¿Qué hacer si no nos vale el *equals* por defecto?

Clase *Object* (IV)

- *hashCode* (cods. no necesariamente únicos)
- *finalize*
- *toString*

```
System.out.println(Thread.currentThread().toString());
```

Thread[main,5,main]



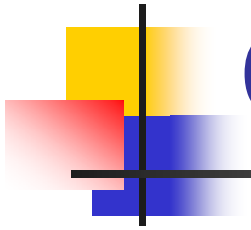
Clase *Object* (V)

- *getClass*

```
String.class or Class.forName("String")
```

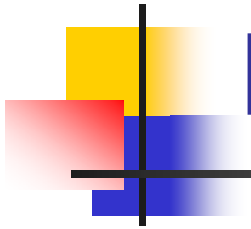
```
void PrintClassName(Object obj) {
   System.out.println("The Object's class is " +
      obj.getClass().getName());
}
```

```
Object createNewInstanceOf(Object obj) {
   return obj.getClass().newInstance();
}
```



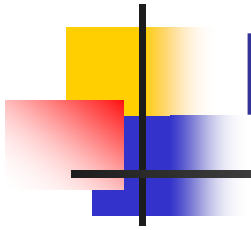
Clase *Object* (VI)

- *notify*, *notifyAll*, *wait*



Paquete *java.lang*

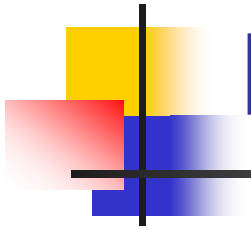
- *Class* (metaclase)
- *Object* (raíz de la jerarquía)
- *Runtime* (entorno de ejecución)
- *SecurityManager* (políticas seguridad)
- *String*
- *System*: *in*, *out*, *getProperties()*, *gc()*
- *Thread*
- *Throwable* (construcción de excepciones)
- ...



Paquete *java.io*

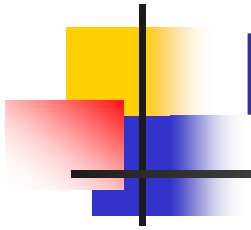
- Tipos de *streams*: bytes, caracteres, con/sin buffer, acceso aleatorio, etc.

- Clase *File*, operaciones sobre directorios



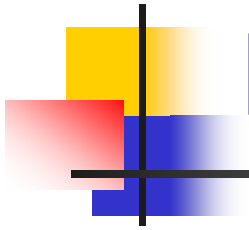
Paquete *java.util*

- *Dictionary, Hashtable, Stack, Vector*
- *Observer/Observable*
- *Random*
- *StringTokenizer*
- *Date*



Paquete *java.net*

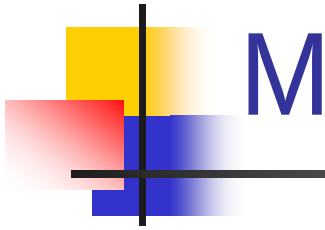
- *Sockets*
- *InetAddress*



Manejo de Cadenas (I)

- Clase *String*: cadenas estáticas (*compartibles*)
- Clase *StringBuffer*: cadenas dinámicas

```
public class StringsDemo {
   public static void main(String[] args) {
      String palindrome = "Dot saw I was Tod";
      int len = palindrome.length();
      StringBuffer dest = new StringBuffer(len);
      for (int i = (len - 1); i >= 0; i--) {
         dest.append(palindrome.charAt(i));
      }
      System.out.println(dest.toString());
   }
}
```



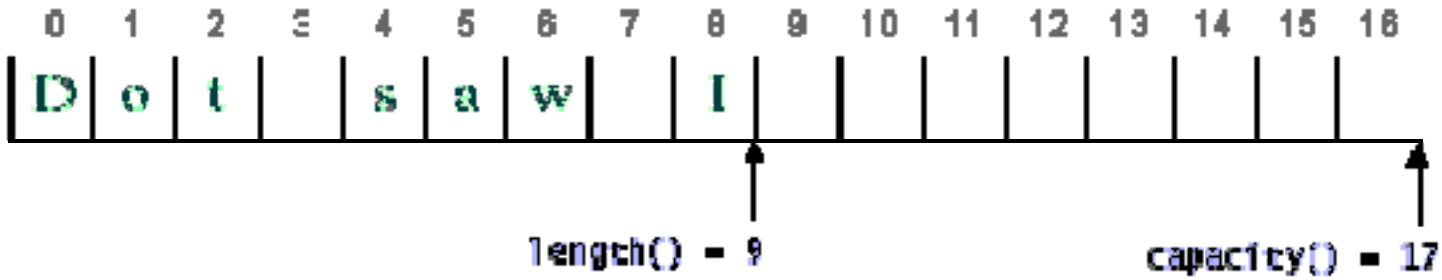
Manejo de Cadenas (II)

```
String palindrome = "Dot saw I was Tod";
```

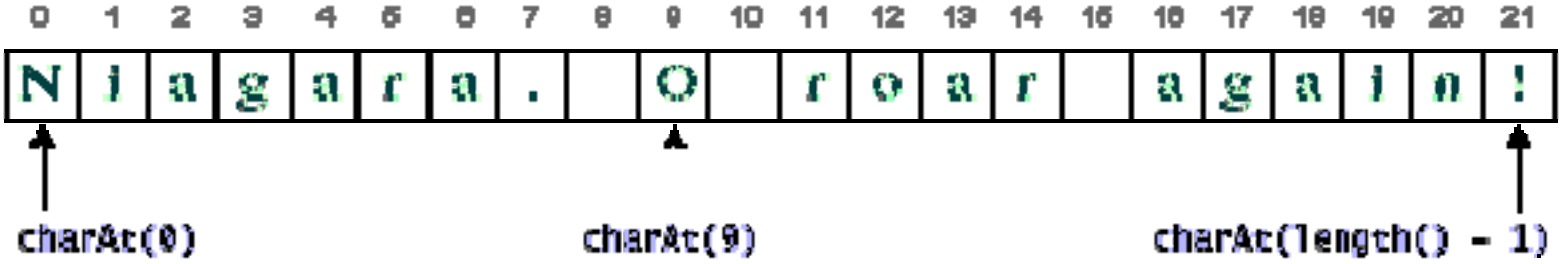
```
char[] helloArray = { 'h', 'e', 'l', 'l', 'o'};
String helloString = new String(helloArray);
System.out.println(helloString);
```

```
String palindrome = "Dot saw I was Tod";
int len = palindrome.length();
StringBuffer dest = new StringBuffer(len);
```

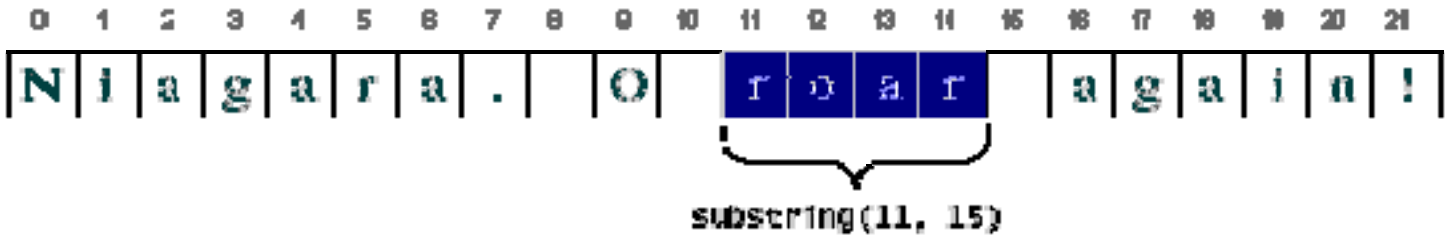
Manejo de Cadenas (III)



(StringBuffer)



Manejo de Cadenas (IV)

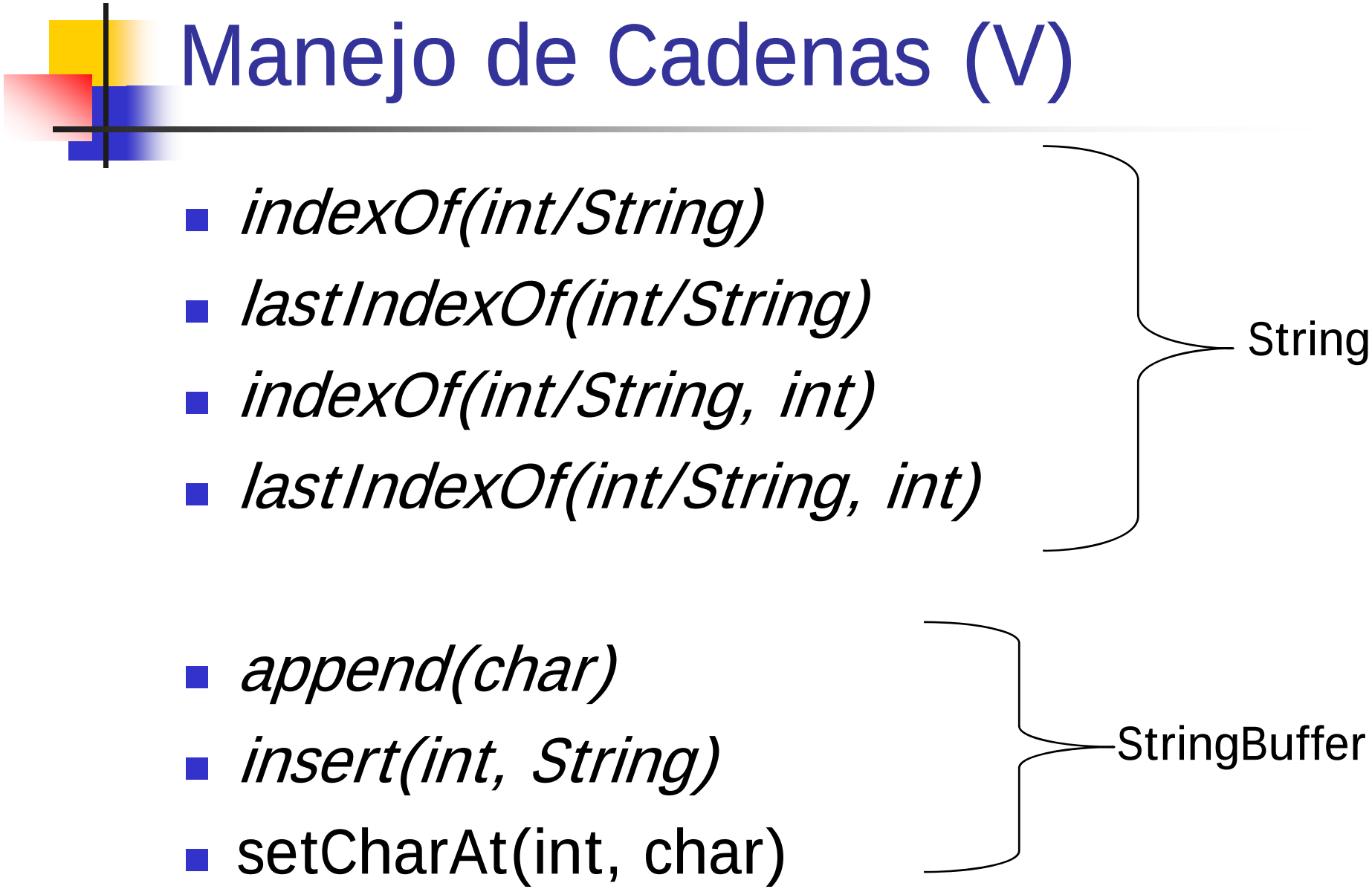


Si no se especifica,
hasta el final de la cadena

==, != compara referencias

```
str1.equals(str2)
```

Manejo de Cadenas (V)



Manejo de Cadenas (VI)

- *String toString()*
  - Object, Boolean, Integer, etc.
- *String.valueOf(...)*

```
System.out.println(String.valueOf(Math.PI));
```

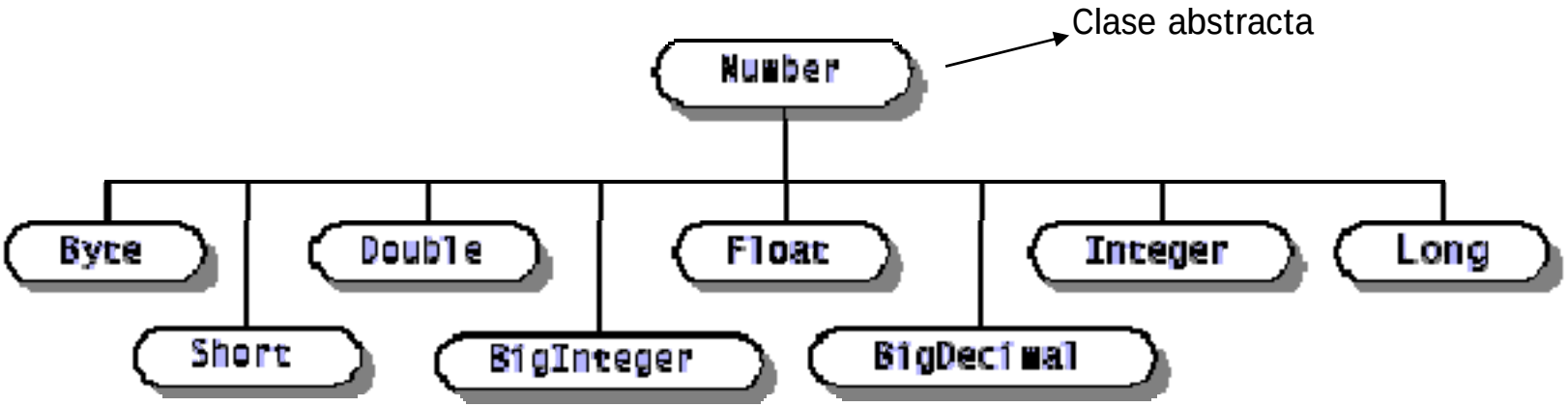
- *<aClass>.valueOf(...)*

```
Float pi = Float.valueOf(piStr);
```

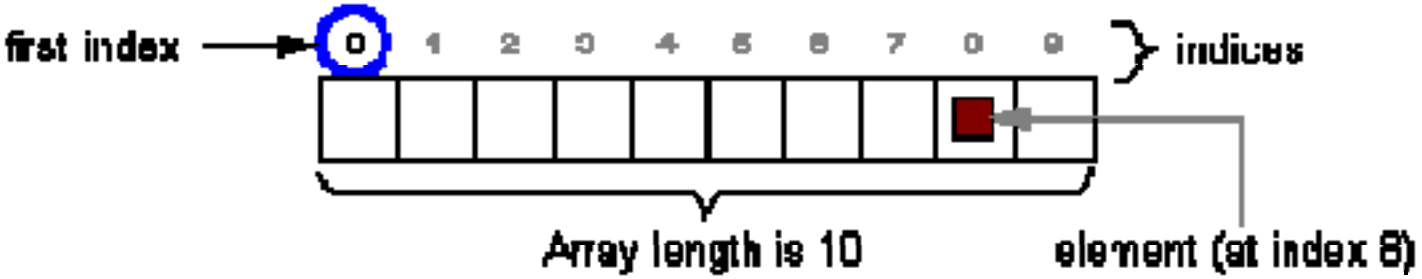
- + (sobrecargado) concatena cadenas:

```
System.out.println("con" + cat + "enation");
```

Clases Numéricas



Arrays (I)


first index
0
1
2
3
4
5
6
7
8
9
indices
Array length is 10
element (at index 8)

Arrays (II)

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] anArray;
        anArray = new int[10];
        for (int i = 0; i < anArray.length; i++) {
            anArray[i] = i;
            System.out.print(anArray[i] + " "); }
            System.out.println();
        }
    }
}
```

Reserva de espacio

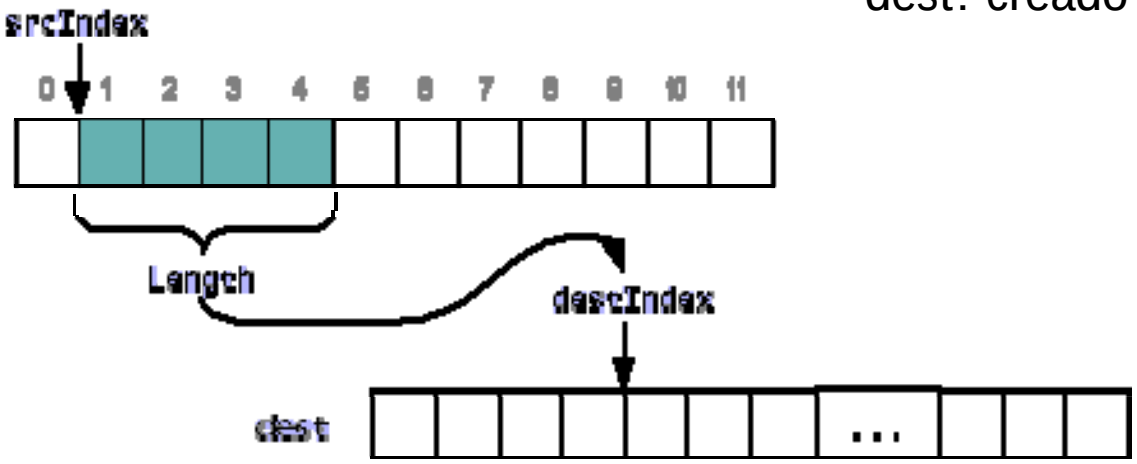
Arrays (III)

```
boolean[] answers =
    { true, false, true, true, false };
```

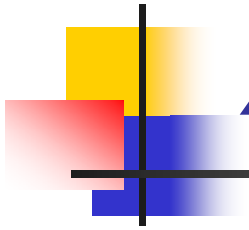
```
String[][] cartoons = {
    {"Flintstones", "Fred", "Wilma", "Pebbles"},
    {"Rubbles", "Barney", "Betty", "Bam Bam"},
    {"Scooby Doo Gang", "Scooby Doo", "Shaggy}
};
```

Arrays (IV)

dest: creado previamente




```
public static void arraycopy(Object source,
    int srcIndex, Object dest, int destIndex,
    int length)
```



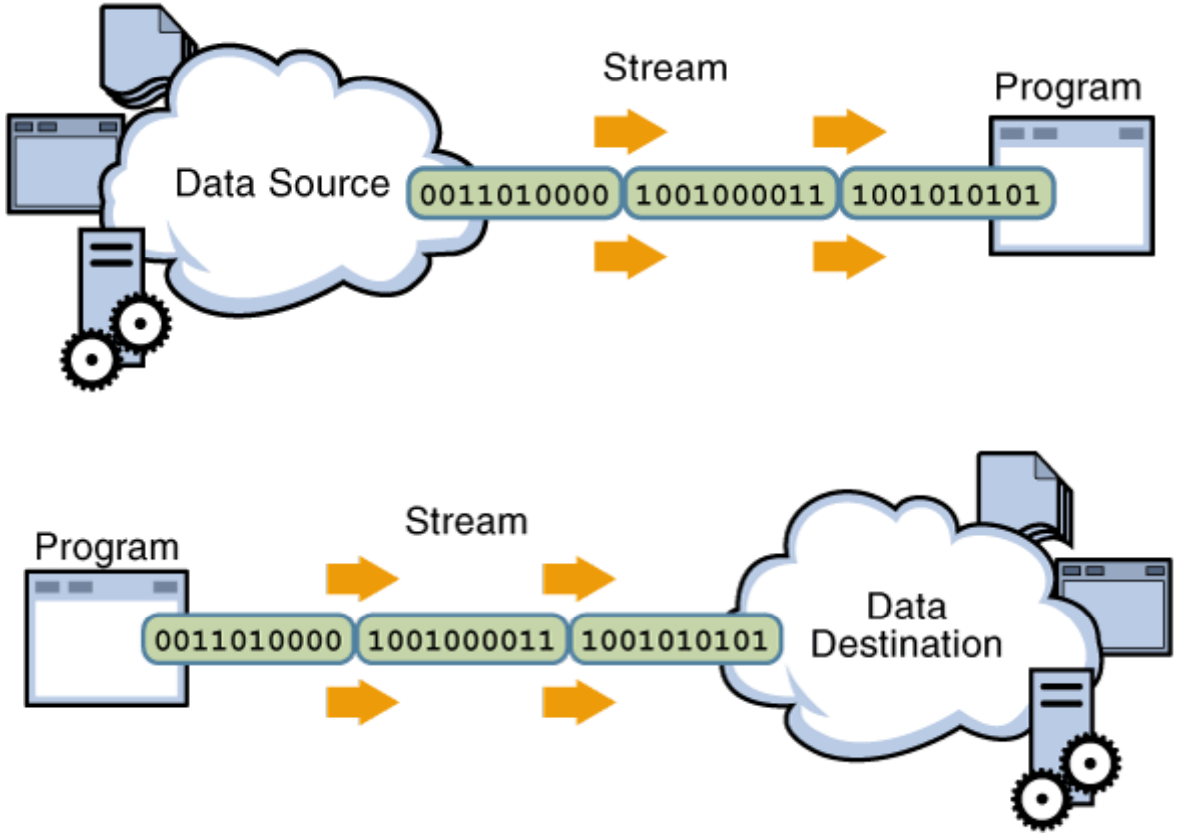
Arrays (V)

- Si necesitamos más espacio

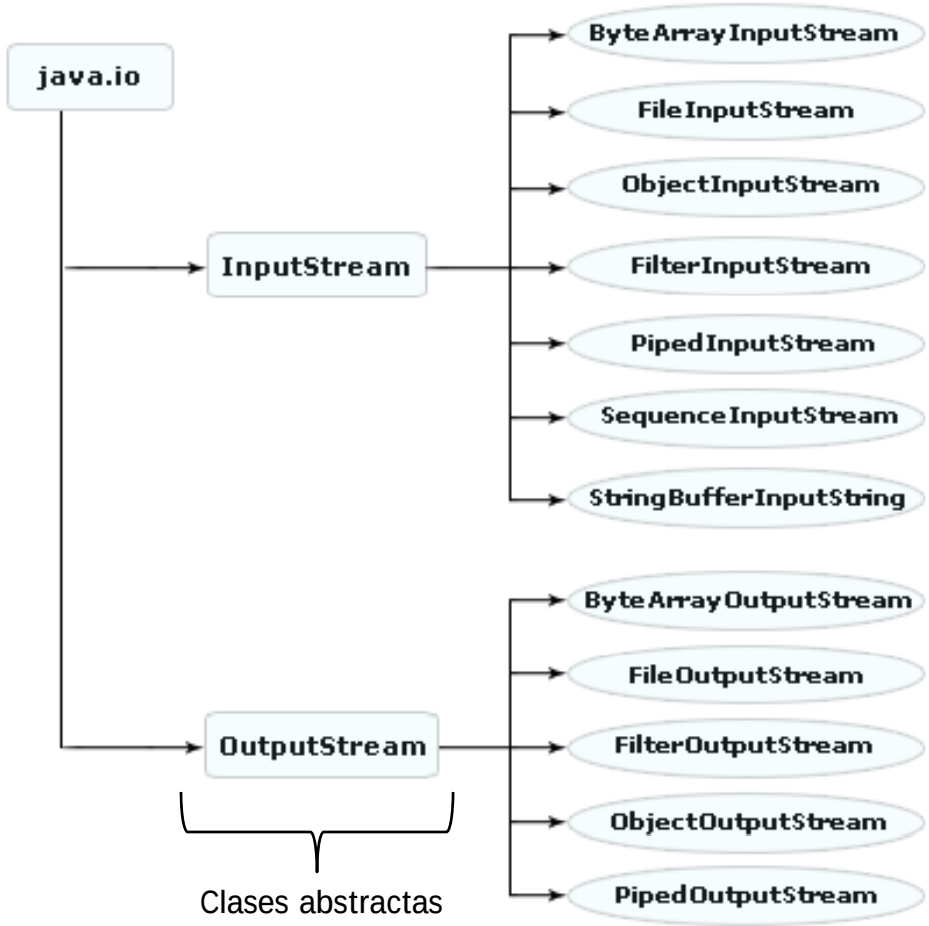
```
int[] v = new int[5];
…
int[] v1 = v;
v = new int[20];
for(int i = 0; i < 5; i++)
  v[i] = v1[i];
```

Anexo II: Flujos de E/S

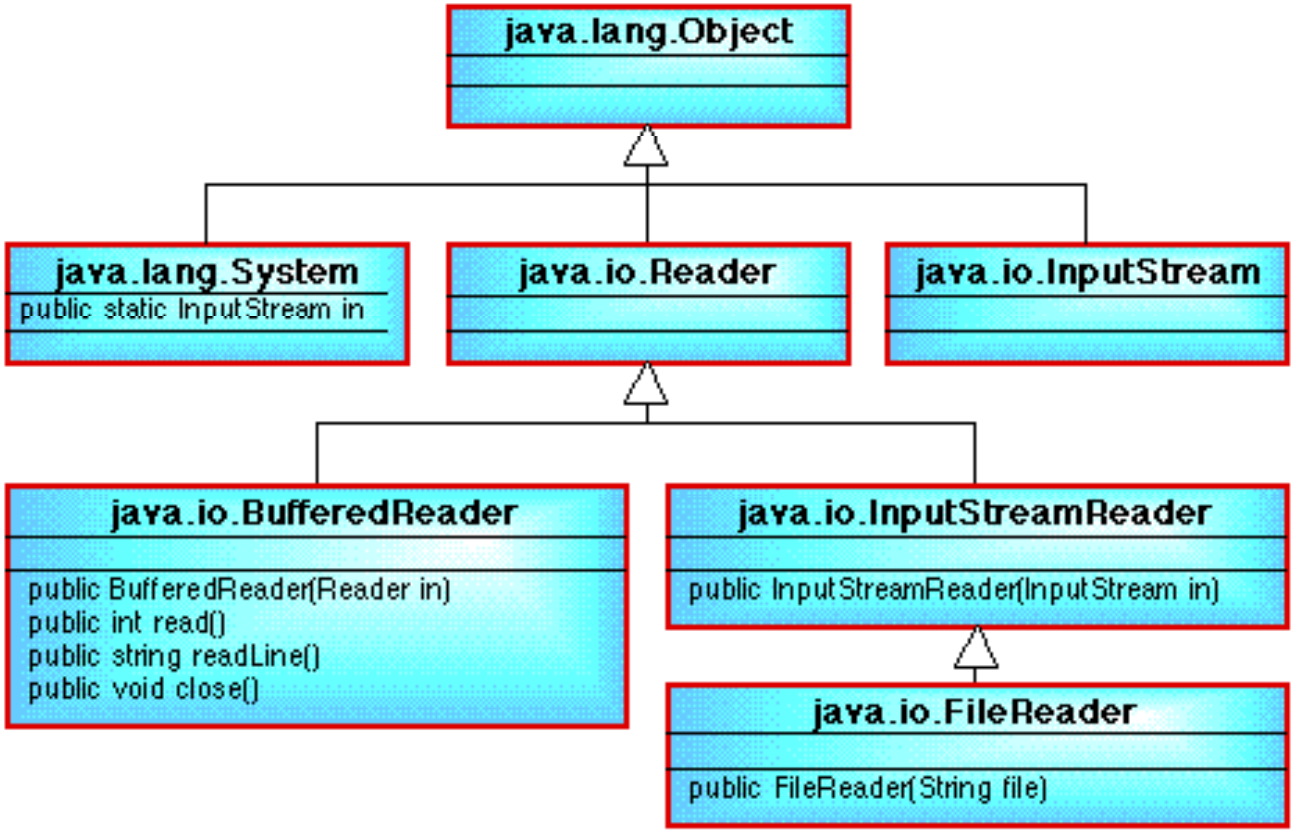
Flujos de Entrada y de Salida



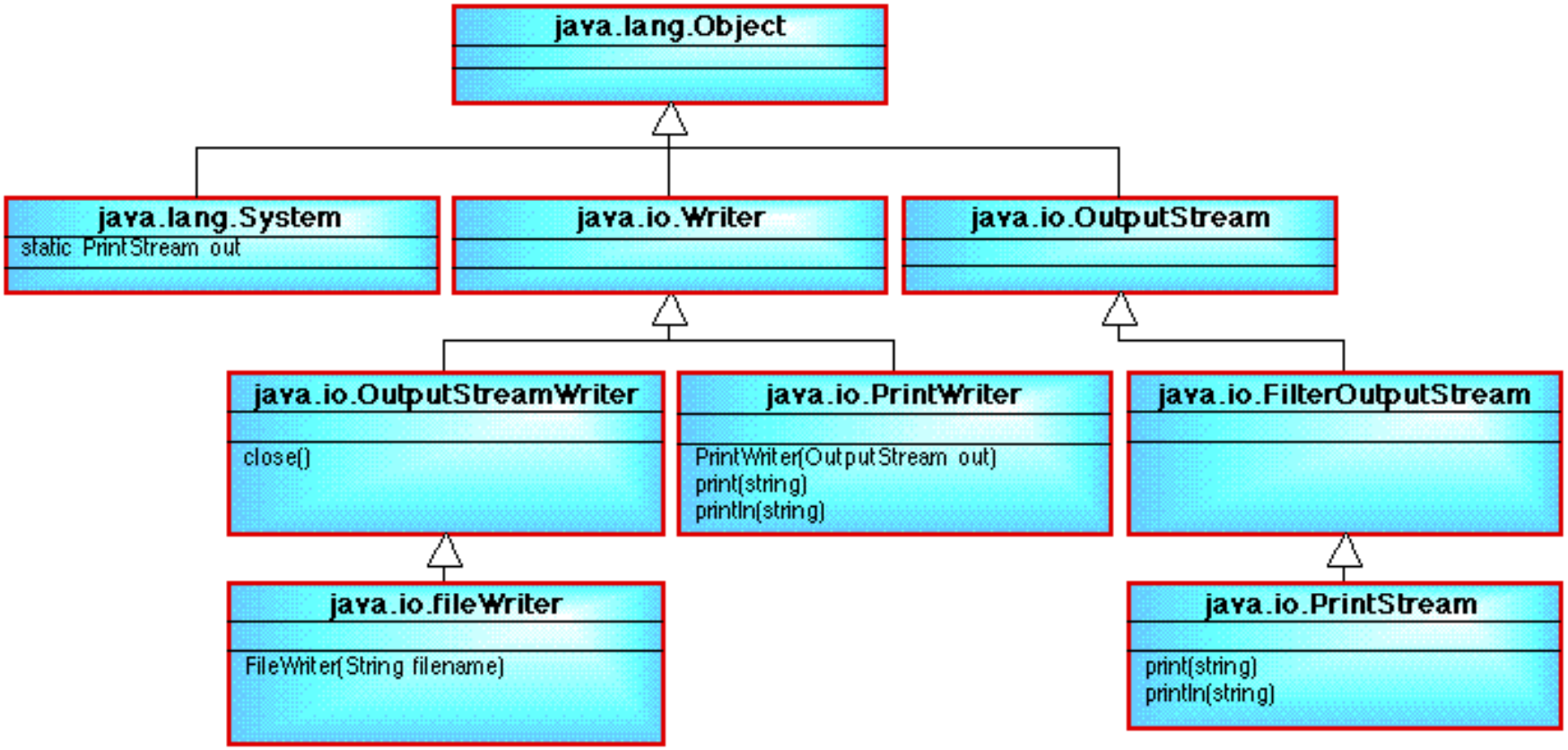
Clases Básicas en java.io



Lectura


java.lang.Object
java.lang.System
public static InputStream in
java.io.Reader
java.io.InputStream
java.io.BufferedReader
public BufferedReader(Reader in)
public int read()
public string readLine()
public void close()
java.io.InputStreamReader
public InputStreamReader(InputStream in)
java.io.FileReader
public FileReader(String file)

Escritura


java.lang.Object
java.lang.System
static PrintStream out
java.io.Writer
java.io.OutputStream
java.io.OutputStreamWriter
close()
java.io.PrintWriter
PrintWriter(OutputStream out)
print(string)
println(string)
java.io.FilterOutputStream
java.io.fileWriter
FileWriter(String filename)
java.io.PrintStream
print(string)
println(string)

Ejemplo de Flujos de Bytes

```
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }

        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Devuelve un entero, no un byte:
-1 indica “fin de flujo”

Cerrar los flujos para evitar *“memory leaks”*

Ejemplo de Flujos de Caracteres

Carácter a carácter

```
public static void main(String[] args) throws IOException {
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Ejemplo de Flujos de Caracteres

Línea a línea

```
public class CopyLines {
   public static void main(String[] args) throws IOException {
      BufferedReader inputStream = null;
      PrintWriter outputStream = null;
      try {
         inputStream =
            new BufferedReader(new FileReader("xanadu.txt"));
         outputStream =
            new PrintWriter(new FileWriter("characteroutput.txt"));

         String l;
         while ((l = inputStream.readLine()) != null) {
            outputStream.println(l);
         }
      } finally {
         if (inputStream != null) {
            inputStream.close();
         }
         if (outputStream != null) {
            outputStream.close();
         }
      }
   }
}
```

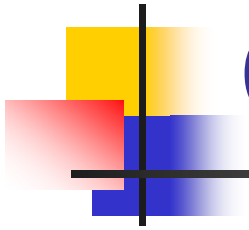
Flujos con Buffer

```
inputStream =
        new BufferedReader(new FileReader("xanadu.txt"));
    outputStream =
        new BufferedWriter(new FileWriter("characteroutput.txt"));
```

- **BufferedInputStream**, **BufferedOutputStream**:
  - flujos de bytes con buffer
- **BufferedReader**, **BufferedWriter**:
  - flujos de caracteres con buffer

Método ***flush***, posibilidad de *autoflush*

Entrada/Salida por Línea de Comandos



- Flujos estándar:
  - System.in
  - System.out
  - System.err

  System.in → InputStream; System.out, System.err → PrintStream
- Aunque uno esperaría que fueran flujos de caracteres, son flujos de bytes (por razones históricas)
- Para usar la entrada estándar como flujo de caracteres:
  - InputStreamReader cin = new InputStreamReader(System.in);

Ejemplo de Flujos de Datos (I)

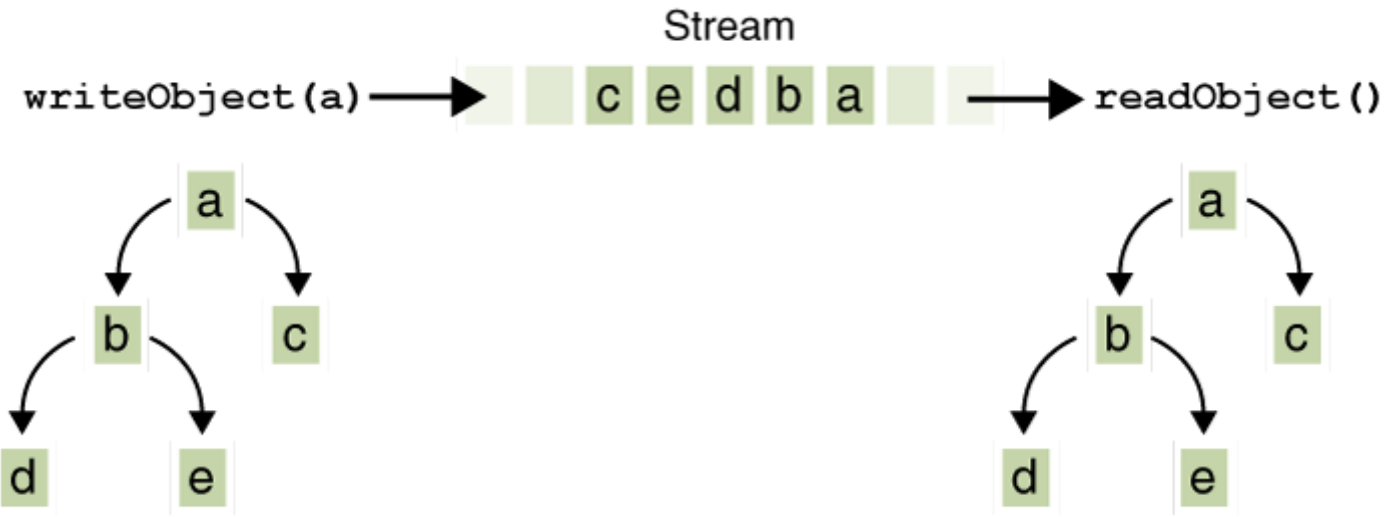
```
static final String dataFile = "invoicedata";

static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
static final int[] units = { 12, 8, 13, 29, 50 };
static final String[] descs = { "Java T-shirt",
        "Java Mug",
        "Duke Juggling Dolls",
        "Java Pin",
        "Java Key Chain" };
 out = new DataOutputStream(new
           BufferedOutputStream(new FileOutputStream(dataFile)));

for (int i = 0; i < prices.length; i ++) {
    out.writeDouble(prices[i]);
    out.writeInt(units[i]);
    out.writeUTF(descs[i]);
}
```

Ejemplo de Flujos de Datos (II)

```
in = new DataInputStream(new
          BufferedInputStream(new FileInputStream(dataFile)));

double price;
int unit;
String desc;
double total = 0.0;

 while (true) {
    try {
        price = in.readDouble();
        unit = in.readInt();
        desc = in.readUTF();
        System.out.format("You ordered %d units of %s at $%.2f%n",
               unit, desc, price);
        total += unit * price;
    }
} catch (EOFException e) {
}
```

Forma de detectar el final de fichero con *data streams*

Flujos de Objetos (I)



•Para clases que implementan el interface *Serializable*
•*ObjectInputStream*, *ObjectOutputStream*
•Implementan los interfaces *ObjectInput* y *ObjectOutput,* subinterfaces de *DataInput* y *DataOutput*

- Por tanto, pueden contener tanto objetos como valores primitivos

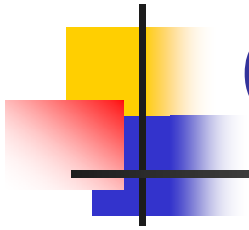
Flujos de Objetos (II)

```
Object ob = new Object();
out.writeObject(ob);
out.writeObject(ob);
```

```
Object ob1 = in.readObject();
Object ob2 = in.readObject();
```

ob1 y ob2 son referencias al mismo objeto:

- Un flujo sólo puede contener una copia del objeto, aunque puede contener cualquier número de referencias a él

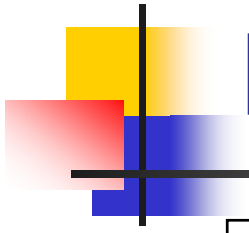


Objetos *File*

- Representan nombres de fichero
- *File a = new File("xanadu.txt");*

Method Invoked	Returns on Microsoft Windows	Returns on Solaris
a.toString()	xanadu.txt	xanadu.txt
a.getName()	xanadu.txt	xanadu.txt
b.getParent()	NULL	NULL
a.getAbsolutePath()	c:\java\examples\xanadu.txt	/home/cafe/java/examples/xanadu.txt
a.getCanonicalPath()	c:\java\examples\xanadu.txt	/home/cafe/java/examples/xanadu.txt

- *File b = new File(".." + File.separator + "examples" + File.separator + "xanadu.txt");*
- *File.compareTo()* identifica *a* y *b* como diferentes



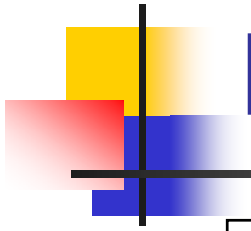
Ejemplo de Uso de *File* (I)

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

      out.print("File system roots: ");
      for (File root : File.listRoots()) {
         out.format("%s ", root);
      }
      out.println();

      for (String fileName : args) {
         out.format("%n------%nnew File(%s)%n", fileName);
         File f = new File(fileName);
         out.format("toString(): %s%n", f);
         out.format("exists(): %b%n", f.exists());
         out.format("lastModified(): %tc%n", f.lastModified());
         out.format("isFile(): %b%n", f.isFile());
         out.format("isDirectory(): %b%n", f.isDirectory());
         out.format("isHidden(): %b%n", f.isHidden());
         ...
```

Ejemplo de Uso de *File* (II)

```
...
          out.format("canRead(): %b%n", f.canRead());
           out.format("canWrite(): %b%n", f.canWrite());
           out.format("canExecute(): %b%n", f.canExecute());
           out.format("isAbsolute(): %b%n", f.isAbsolute());
           out.format("length(): %d%n", f.length());
           out.format("getName(): %s%n", f.getName());
           out.format("getPath(): %s%n", f.getPath());
           out.format("getAbsolutePath(): %s%n", f.getAbsolutePath());
           out.format("getCanonicalPath(): %s%n", f.getCanonicalPath());
           out.format("getParent(): %s%n", f.getParent());
           out.format("toURI: %s%n", f.toURI());
       }
   }
}
```

Otros métodos: *delete*, *deleteOnExit*, *setLastModified*, *renameTo*, *mkdir*, *mkdirs*, etc.
Métodos estáticos: *createTempFile*, etc.



Ejemplo: Copiar un Fichero

```
void copy(File src, File dst) throws IOException {
        InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dst);

        // Transfer bytes from in to out
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();
    }
```

Ficheros de Acceso Aleatorio

```
try {
        File f = new File("filename");
        RandomAccessFile raf = new RandomAccessFile(f, "rw");

        // Read a character
        char ch = raf.readChar();

        // Seek to end of file
        raf.seek(f.length());

        // Append to the end
        raf.writeChars("aString");
        raf.close();
    } catch (IOException e) {
}
```

Otros métodos: int *skipBytes*(int), void *seek*(long), long *getFilePointer*(), etc.

http://www.exampledepot.com/egs/java.io/ReadFromStdIn.html?l=rel