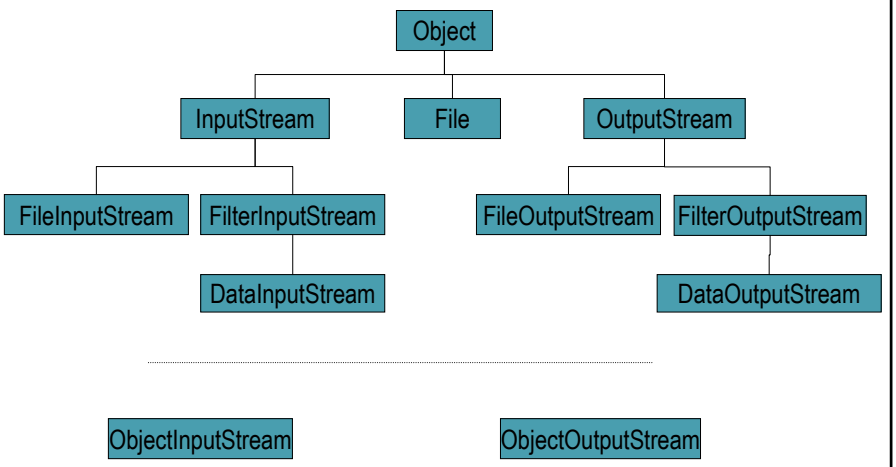
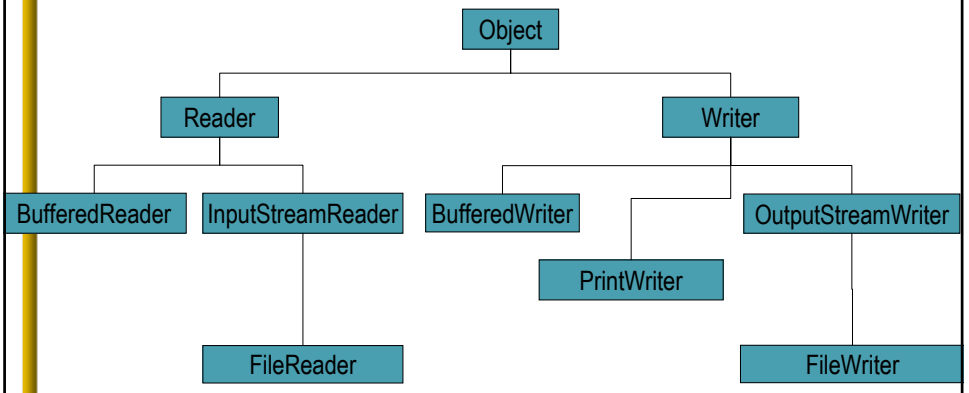
Tema 8. Entrada/Salida

8. Entrada/Salida – Paquete java.io



8. Entrada/Salida – Paquete java.io



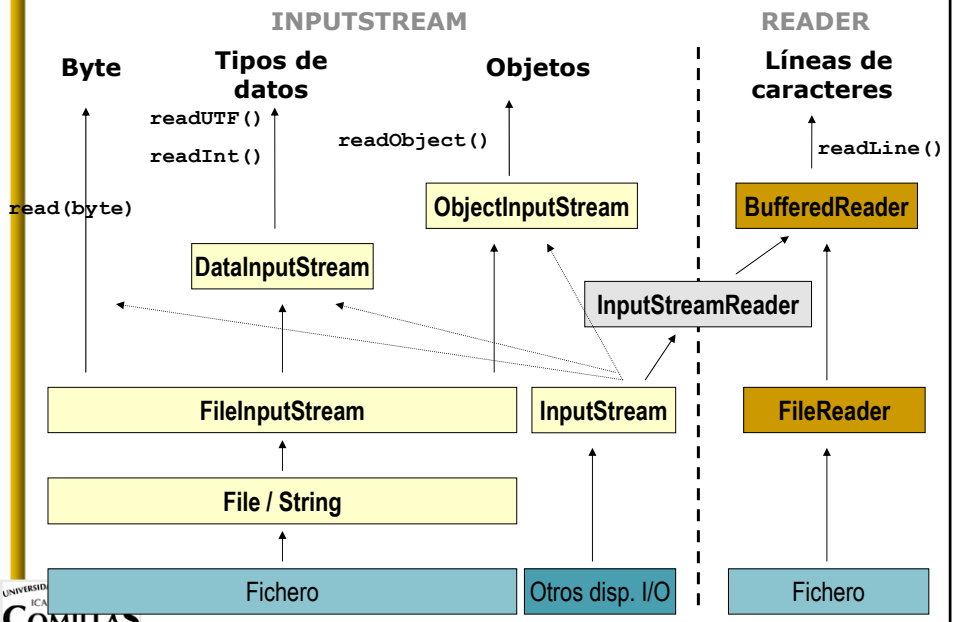
8. Entrada/Salida – Paquete java.io

- El paquete *java.io* proporciona un marco de trabajo para realizar operaciones de entrada y salida entre nuestro programa java y cualquier dispositivo externo al sistema (teclado, consola, ficheros, maquinas remotas,...)
- Se puede tener cierta independencia entre el programa Java y el dispositivo gracias a la utilización de streams.
- Un stream es un elemento software que permite el intercambio de un flujo de bits entre dos entes en un solo sentido.
- Para una operación de lectura y escritura contra un fichero, por ejemplo, necesitaríamos dos streams, uno de entrada y otro de salida.
- Las clases InputStream y OutputStream implementan en Java esta funcionalidad.

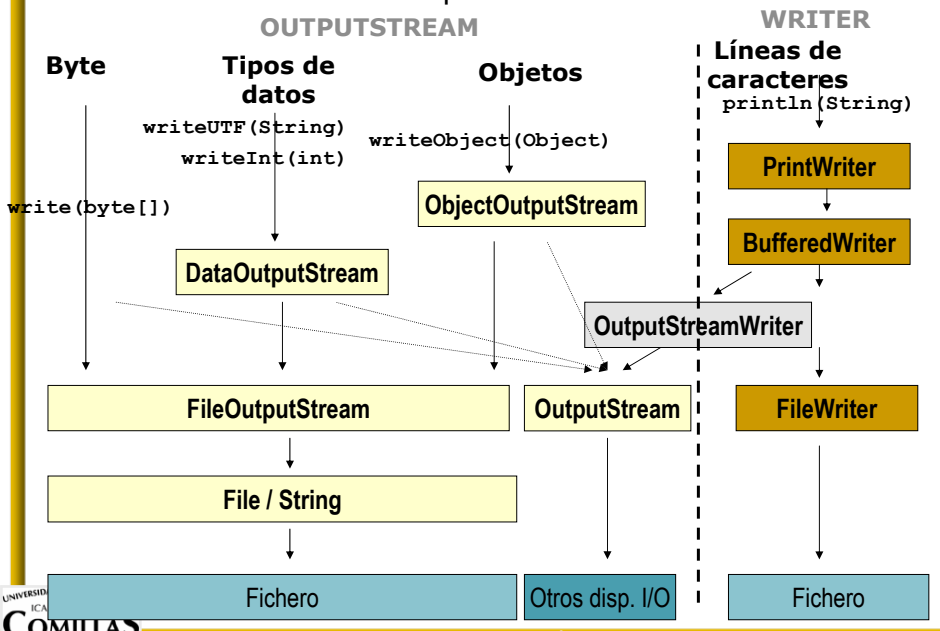
8. Entrada/Salida – Paquete java.io

- La API de Java diferencia entre los streams de caracteres y streams de bytes.
- Streams de bytes (InputStream y OutputStream)
  - Diseñados para operar a nivel de byte.
  - Hay ciertos dispositivos que sólo pueden trabajar a este nivel.
- Streams de caracteres (Reader y Writer):
  - Diseñados para optimizar las operaciones con cadenas de caracteres.
  - Posteriores a las anteriores.
  - Soportan codificaciones Unicode-16.
  - Cualquier fichero de texto puede ser accedido directamente a través de este tipo de stream.
  - Existen dispositivos, que aunque están preparados para enviar/recibir información de esta naturaleza, no pueden tratatrse directamente con este stream. En estos casos, se deberá envolver el stream de bytes con un stream de caracteres: (ver diap. sig.)
    - InputStreamReader
    - OutputStreamReader

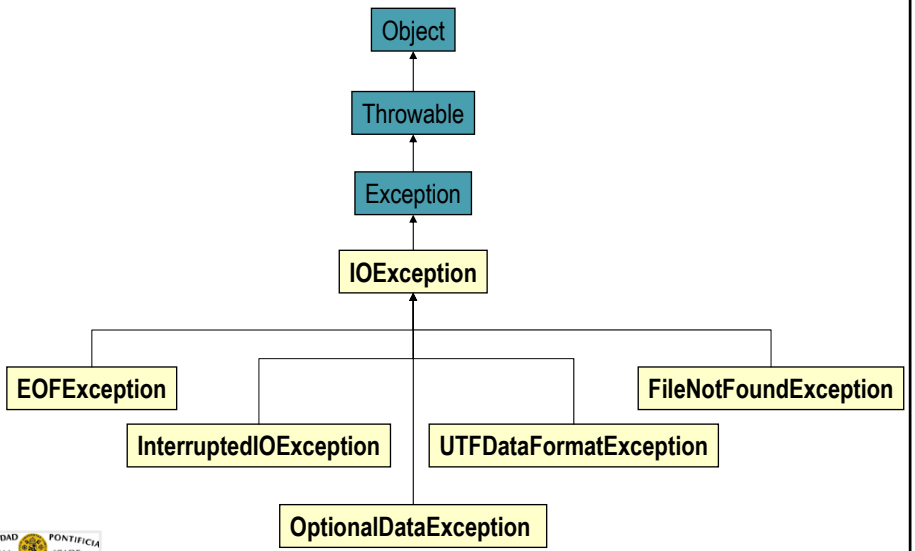
8. Entrada – Modo de Operación



8. Salida – Modo de Operación



8. Entrada/Salida – Excepciones



8. Clase File

- Implementa un fichero del sistema de archivos.
- Se emplea para obtener información del fichero sobre el cual vamos a trabajar.
- **Métodos:**
  - `String getName()`
  - `String getPath()`
  - `boolean renameTo(File)`
  - `boolean exists()`
  - `boolean delete()`
  - `boolean mkdir()`
  - `boolean isDirectory()`
  - `...`

8. Clase File - Ejemplo

```
import java.io.*;
public class FileInfo
{
   public static void main(String[] args)
   {
      File path;
      path =new File(args[0]);
      String exists =path.exists()? "Si": "No";
      String canRead =path.canRead()? "Si": "No";
      String canWrite = path.canWrite()? "Si": "No";
      String isFile = path.isFile()? "Si": "No";
      String isDir = path.isDirectory()? "Si": "No";
      System.out.println("Existe  :"+exists);
        if (path.exists()){
          System.out.println("De lectura  :"+canRead);
          System.out.println("De Escritura   :"+canWrite);
          System.out.println("Es un directorio   :"+isDir);
          System.out.println("Es un fichero    :"+isFile);
```

8. Clase FileOutputStream

- Permite abrir un fichero de texto para ser tratado en modo escritura.
- Sobre este objeto se pueden realizar operaciones byte a byte directamente.
- **Métodos**:
  - `write()`
- **Ejemplo**:

```
byte b = b[1024];
fos= new FileOutputStream (fichero);
bos = new BufferedOutputStream(fos);
…
for(i=0;b[i] != 0; i++)
          bos.write(b[i]);
…
fis.close();
```


8. Clase FileInputStream

- Permite abrir un fichero de texto para ser tratado en modo lectura secuencialmente byte a byte.
- **Métodos:**
  - `read()`
- **Ejemplo**:

```
byte b = b[1024];
fis= new FileInputStream(fichero);
…
int i = fis.read(b);
String s = new String(b);
…
fis.close();
```

8. Clase DataOutputStream

- La escritura se realiza considerando la información como tipos de datos.
- **Métodos**:
  - `writeByte(byte);` `writeInt(int);`
  - `writeUTF(String);` `writeChars(String);`
  - `writeChar(int);` `writeBytes(String);`
- **Ejemplo**:

```
DataOutputStream dos = new DataOutputStream( new
  FileOutputStream("personas.dat"));
dos.writeDouble(dni)
dos.writeChar('\t');
dos.writeInt(edad);
dos.writeChar('\t');
dos.writeUTF(nombre);
dos.writeChar('\n');
```


8. Clase DataInputStream

- La lectura se realiza considerando la información como tipos de datos
- **Métodos**:
  - `byte readByte();` `int readInt();`
  - `String readUTF();`
  - `char readChar();` `String readLine();`
- **Ejemplo**:

```
DataInputStream dis = new DataInputStream(
    new FileInputStream("personas.dat"));
dni = dis.readDouble();
dis.readChar(); // elimina el tabulador
edad = dis.readInt();
dis.readChar(); //  elimina el tabulador
nombre = dis.readUTF();
```

8. Clase ObjectOutputStream()

- Se exporta el estado de un objeto a una cadena de bytes.
- Los objetos deben ser **serializables**.
- **Métodos:**
  - `writeObject();`
- **Ejemplo:**

```
FileOutputStream fos = new
  FileOutputStream("Fechas.txt");
ObjectOutputStream s = new ObjectOutputStream(fos);
s.writeObject("Hoy");
s.writeObject(new Date());
s.flush();
s.close();
```


8. Clase ObjectInputStream()

- Se importa el estado de un objeto de una cadena de bytes.
- Los objetos deben ser **serializables**.
- **Métodos**:
  - `readObject();`
- **Ejemplo**:

```
FileInputStream fis = new FileInputStream("Fechas.txt");
ObjectInputStream s = new ObjectInputStream(fis);
String hoy = (String)s.readObject();
Date fecha = (Date)s.readObject();
s.close();
```

8. Clase PrintWriter

- Escritura optimizada de un fichero de texto a nivel de líneas.

```
import java.io.*;
class EscribeLinea
{
   public static void main(String args[])
   {
        try
        {
                PrintWriter pw = new PrintWriter(new
   BufferedWriter(new FileWriter(args[0])), true);
                pw.println("Primera linea del fichero");
                pw.println("Segunda linea del fichero");
                pw.println("Tercera linea del fichero");
                pw.close();
        }
        catch(IOException e)
        {
                System.out.println("Se produjo un error de E/S");
        }
   }
```


8. Clase BufferedReader (con FileReader)

- Las lecturas se realizan a nivel de líneas de caracteres.
- Se apoya en la clase FileReader.
- El método más importante que posee es `readLine()`.

```
import java.io.*;
class LeeLinea
{
   public static void main(String args[])
   {
        try
        {       String s="";
                BufferedReader br=new BufferedReader(new
                        FileReader(args[0]));
                while ((s = br.readLine()) != null )
                        System.out.println(s);
                br.close();
        }
        catch(IOException e)
        {
                System.out.println("Se produjo un error de E/S");
        }
```

8. Clase BufferedReader (con FileInputStream)

- Existen casos en los tendremos que leer líneas de caracteres sobre dispositivos que trabajan con streams (p.e. el Teclado).
- Entones utilizaremos las clases InputStreamReader o OutputStreamWriter.

```
import java.io.*;
class LeeLinea
{
   public static void main(String args[])
   {
        try
        {       String s="";
                BufferedReader br=new BufferedReader(new
   InputStreamReader(new FileInputStream(args[0])));
                while ((s = br.readLine()) != null )
                        System.out.println(s);
                br.close();
        } catch(IOException e) {
                System.out.println("Excepción E/S: "+e);
        }
   }
```


8. EOFException - Ejemplo

```
import java.io.*;
class EOF
{
   public static void main(String args[])
   {
      byte ch;
      try
      {
        DataInputStream d = new DataInputStream(new
   FileInputStream("EOF.java"));
        while (true){
              ch = d.readByte();
              System.out.print((char)ch);
       }
      catch(EOFException eof)
       {
         System.out.println(">>Terminacion normal del programa");
       }
```

8. EOFException - Ejemplo

```
      catch(FileNotFoundException noFile)
      {
         System.out.println(">>El Fichero no se encuentra"+noFile);
      }
      catch(IOException io)
      {
         System.out.println(">>Error de IO"+io);
       }
      catch(Exception e)
      {
         System.out.println(">>Salida anormal del programa"+e);
       }
   }
}
```


8. Concatenación de Streams

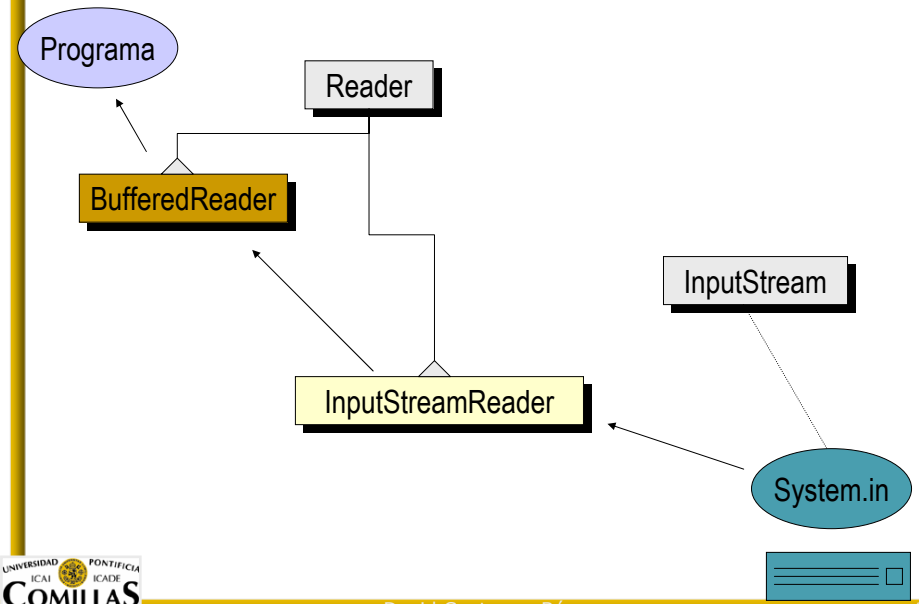
- Podemos concatenar ficheros como si se trataran de cadenas, mediante la clase **SequenceInputStream**.

```
import java.io.*;
public class SecuenciarFlujos{
   public static void main(String[] arg)   {
      try{  FileInputStream f1Entrada=new FileInputStream(arg[0]);
            FileInputStream f2Entrada=new FileInputStream(arg[1]);
            InputStream entrada=new
            SequenceInputStream(f1Entrada,f2Entrada);
           int dato;
           while((dato=entrada.read())!=-1) {
                System.out.print((char)dato);
             }
          }catch(IOException e) {
                System.out.println("Excepción E/S: "+e);
          }  }}
```

8. Dispositivos Estándar de E/S

- Siguiendo la nomenclatura UNIX, Java tiene la entrada estándar (System.in), salida estándar (System.out) y salida estándar de error (System.err).
  - System.out es un PrintStream preformateado
  - System.err un PrintStream preformateado
  - System.in es un InputStream sin formatear

8. Lectura por la entrada estándar (teclado)



8. Lectura por la entrada estándar (teclado)

```
public String leeTeclado()
{
        String s = null;
        try
        {
                BufferedReader b = new
BufferedReader(new InputStreamReader(System.in));
                s = b.readLine();
        }
        catch(IOException e)
        {
                e.printStackTrace();
        }
        return s;
}
```


8. Ejemplo de otro tipo de stream

- La ventaja de trabajar con streams:

```
void leeURL()
{
    try
    {
        URL miUrl = new URL("http://www.upco.es/")
        InputStream is = miUrl.openStream();
        BufferedReader br = new BufferedReader(new
        InputStreamReader(is));
        while((s = br.readLine()).length()!=0)
                System.out.println(s);
    }
    catch(IOException e)
    {
        System.out.println("Error de IO");
    }
}
```

8.1 Serialización

- Proporciona un mecanismo de persistencia e integridad.
- Permite convertir un objeto a un flujo de bytes para ser tratados por un stream.
- Se obtiene mediante la implementación del interface java.io.Serializable.
- Este interface no define ningún método.
- Proceso reversible:
  - serialización – deserialización
  - escritura - lectura

8.1 Serialización

- Ejemplo:

```
class Persona implements java.io.Serializable
{
   //Definición de la clase
}
```