

Java Inicial

(20 horas)





Temario

1. Programación Orientada a Objetos
2. Introducción y Sintaxis Java
3. Sentencias Control Flujo
4. POO en Java
5. Relaciones entre Objetos
6. Polimorfismo, abstracción e interfaces
7. Excepciones
8. **Conceptos avanzados**



Tema 8

Conceptos Avanzados



Objetivos

1. Programación Orientada a Objetos
2. Introducción y Sintaxis Java
3. Sentencias Control Flujo
4. POO en Java
5. Relaciones entre Objetos
6. Polimorfismo, abstracción e interfaces
7. Excepciones
8. **Conceptos avanzados**

■ Entrada / Salida

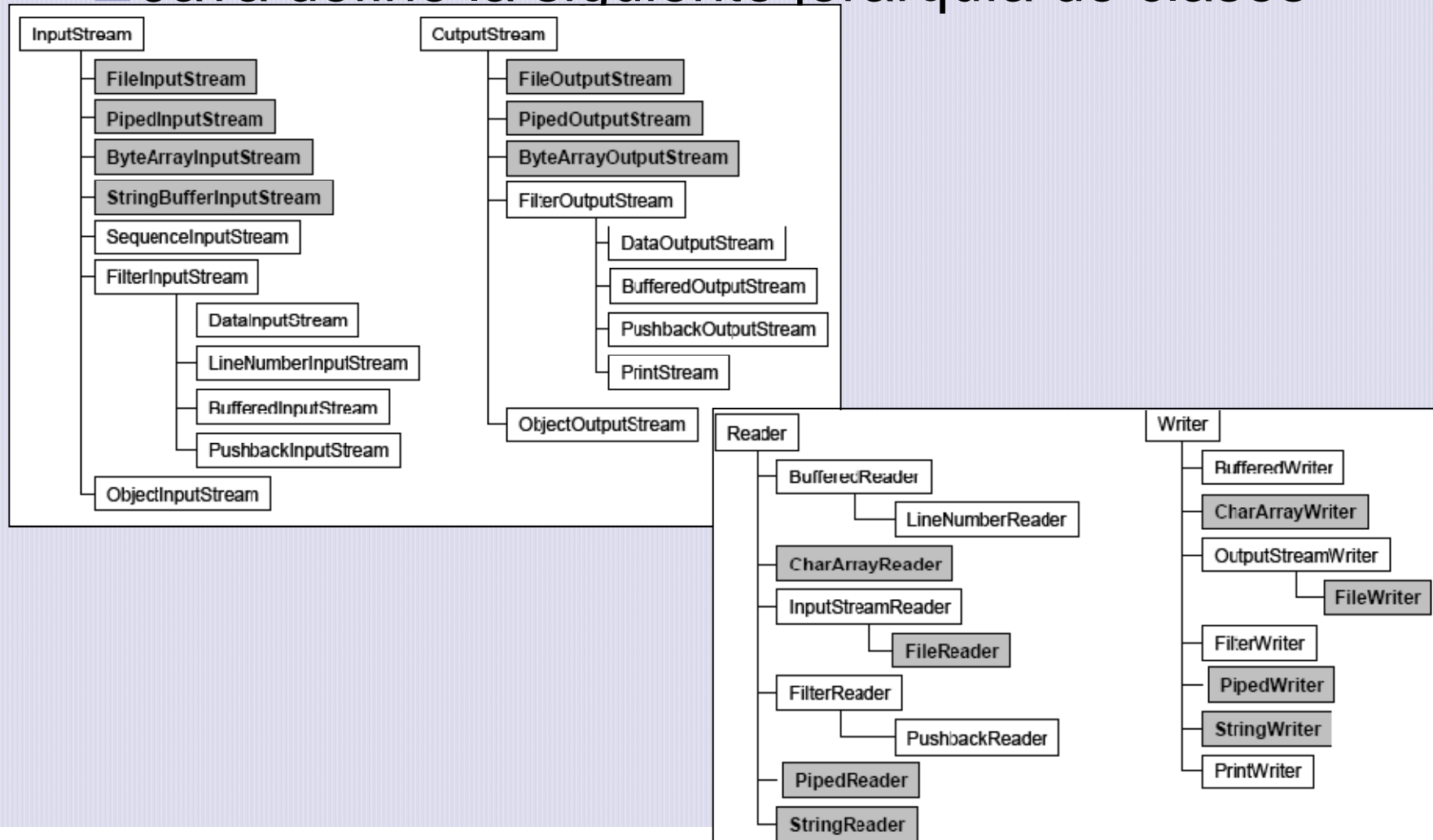
- Paquete I/O
- Entrada Estándar
- Salida Estándar
- Fichero
 - File[I/O]Stream
 - Data[I/O]Stream
 - File
- Serialización

■ Entrada/Salida

- Necesitamos datos en las aplicaciones
 - Interactuando con el usuario: el teclado
 - Accediendo al disco: ficheros
 - Mediante una conexión TCP/IP
- Podemos mostrar nuestro resultado en:
 - Pantalla
 - Fichero
 - Enviándolo en una conexión TCP/IP
- **Stream**: Un flujo de datos, una conexión entre nuestra aplicación y un origen/destino de los datos.

■ Jerarquía Entrada/Salida

□ Java define la siguiente jerarquía de clases



- **Entrada/Salida (*paquete I/O*)**
 - Necesitamos datos en las aplicaciones
 - Interactuando con el usuario: el teclado
 - Accediendo al disco: ficheros
 - Mediante una conexión TCP/IP
 - Podemos mostrar nuestro resultado en:
 - Pantalla
 - Fichero
 - Enviándolo en una conexión TCP/IP
 - **Stream**: Un flujo de datos, una conexión entre nuestra aplicación y un origen/destino de los datos.

- Clases que indican origen/destino de los datos.
 - Clases de E/S en disco:
 - *FileReader - FileWriter*
 - *FileInputStream - FileOutputStream.*
 - Clases de E/S en memoria:
 - *StringReader StringWriter*
 - *CharArrayReader CharArrayWriter*
 - *ByteArrayInputStream ByteArrayOutputStream*
 - *StringBufferInputStream.*
 - Conexión bilateral para transmisión de datos (tuberías):
 - *PipedReader PipedWriter*
 - *PipedInputStream PipedOutputStream*

- Clases que modifican comportamiento.
 - Utilización de un buffer minimizando el acceso al dispositivo:
 - *BufferedReader* *BufferedWriter*
 - *BufferedInputStream* *BufferedOutputStream*
 - Conversión de un flujo de byte en uno de caracteres. Es la conexión con la jerarquía
 - *InputStream/OutputStream*
 - *InputStreamReader* *OutputStreamWriter*
 - Serialización de objetos
 - *ObjectInputStream* y *ObjectOutputStream*
 - Filtros o procesos sobre el stream
 - *FilterReader* *FilterWriter* *FilterInputStream* *FilterOutputStream*
 - Manejar datos en formato de Java (independencia de la plataforma)
 - *DataInputStream* *DataOutputStream*
 - Con métodos adaptados para imprimir tipos de Java
 - *PrintWriter* o *PrintStream*

■ Entrada / Salida estándar

- La clase `System` (*java.lang*) define los siguientes objetos estáticos:
- Permiten la lectura/escritura de caracteres.
 - **System.in**: `InputStream` utilizado para la recepción de datos de la entrada estándar (normalmente el teclado)
 - **System.out**: `PrintStream` utilizado para el envío de datos a la salida estándar (normalmente la pantalla)
 - **System.err**: `PrintStream` utilizado para el envío de mensaje de error a la salida estándar normalmente asociado con la pantalla)

■ Entrada estándar - Teclado

□ Clases para leer datos de la entrada estándar

■ **InputStream**: Lectura de datos carácter a carácter

```
class TestTeclado02
{
    public static void main(String args[])
    {
        byte datos[] = new byte[20];
        int numDatos;
        try
        {
            numDatos = System.in.read(datos);
            System.out.println("Caracteres:" + numDatos);
            for (int i = 0; i < numDatos - 2; i++)
                //numDatos-2 por RETORNO DE CARRO y SALTO DE LINEA
            {
                System.out.print((char) datos[i]);
            }
            System.out.println ();
        }
        catch (java.io.IOException e){}
    }
}
```

■ Entrada estándar - Teclado

```
import java.io.*;

public class TestTeclado03
{
    public static void main(String args[])
    {
        BufferedReader teclado =
            new BufferedReader(
                new InputStreamReader(System.in));
        String cadena="";
        do
        {
            try{
                cadena = teclado.readLine();
                System.out.println("Leido: " + cadena);
            }
            catch(IOException e){}
        }while(!cadena.equalsIgnoreCase("FIN"));
    }
}
```

■ Entrada estándar - Teclado

```
1 Esta es la primera linea
2 Leido 1: Esta es la primera linea
3 Esta es la segunda linea
4 Leido 2: Esta es la segunda linea
5 RESET
6 Leido 3: RESET
7 Esta es una nueva primera linea
8 Leido 1: Esta es una nueva primera linea
9 Esta es una nueva segunda linea
10 Leido 2: Esta es una nueva segunda linea
11 fin
12 Leido 3: fin
```

```
        cadena = teclado.readLine();
        System.out.println ("Leido " +
                             teclado.getLineNumber() +
                             ": "+cadena);
        if(cadena.equalsIgnoreCase("RESET"))
            teclado.setLineNumber(0);
    }while(!cadena.equalsIgnoreCase("FIN"));
}
catch(IOException e){
}
}
```

■ Salida estándar – Pantalla

- El objeto *out* definido en la clase `System` nos permite realizar salida por consola
- El objeto Pertenece a la clase *PrintStream*
- Permite imprimir datos del tipo que sea

Método	Imprime un...
<code>void print(boolean b)</code>	boolean
<code>void print(char c)</code>	carácter
<code>void print(char [] s)</code>	array de caracteres
<code>void print(double d)</code>	double
<code>void print(long l)</code>	long
<code>void print(Object o)</code>	Object
<code>void print(String s)</code>	String

■ Entrada / Salida sobre Ficheros

□ File, FileDescriptor, FileInputStream y FileOutputStream:

- Nos permiten definir streams tomando como origen/destino un archivo.

□ RandomAccessFile:

- Nos permite trabajar con el acceso directo sobre ficheros.

□ FileReader y FileWriter:

- Nos permite el acceso a ficheros trabajando con caracteres.

■ E /S de Bytes sobre Ficheros

□ FileInputStream y FileOutputStream:

- A partir de un objeto de la clase File

```
FileInputStream(File f)  
FileOutputStream(File f)
```

- A partir de un objeto de la clase FileDescriptor

```
FileInputStream(FileDescriptor fd)  
FileOutputStream(FileDescriptor fd)
```

- A partir de un nombre de archivo (String)

```
FileInputStream(String nombre)  
FileOutputStream(String nombre)
```

■ E /S de Bytes sobre Ficheros

□ FileInputStream y FileOutputStream:

- A partir de un objeto de la clase File

```
FileInputStream(File f)  
FileOutputStream(File f)
```

- A partir de un objeto de la clase FileDescriptor

```
FileInputStream(FileDescriptor fd)  
FileOutputStream(FileDescriptor fd)
```

- A partir de un nombre de archivo (String)

```
FileInputStream(String nombre)  
FileOutputStream(String nombre)
```

■ Ejemplo

```
import java.io.*;
public class TestFichero01
{
```

```
Indique cuantos quiere...4
Introduzca los datos
Esta es la primera cadena
Esta es la segunda cadena
Esta es la cuarta cadena
No, la cuarta y ultima es esta
El contenido del fichero es...
Esta es la primera cadena
Esta es la segunda cadena
Esta es la cuarta cadena
No, la cuarta y ultima es esta
```

```
{
    cadena = teclado.readLine();
    fEscritura.write(cadena.getBytes());
    fEscritura.write("\n".getBytes());
}
```

```
fEscritura.close();
```

```
FileInputStream fLectura =
    new FileInputStream("Entrada.dat");
System.out.println ("El contenido del fichero es...");
while(fLectura.available() !=0)
{
    System.out.print ((char)fLectura.read());
}
```

```
fLectura.close();
```

■ DataInputStream y DataOutputStream

- En el ejemplo anterior tuvimos que tratar los datos como bytes

Método	Lee un...
<code>boolean readBoolean()</code>	boolean
<code>byte readByte()</code>	byte
<code>char readChar()</code>	char
<code>double readDouble()</code>	double
<code>float readFloat()</code>	float
<code>int readInt()</code>	int
<code>long readLong()</code>	long
<code>short readShort()</code>	short
<code>String readUTF()</code>	cadena en formato UTF-8

■ Ejemplo

```
Cuantos datos desea...
10
1 3 5 7 9 11 13 15 17 19
Los dobl
2
6
10
14
18
22
26
30
34
38
```

Contenido Legible

```
fSalida.println("Los dobles son...");
DataInputStream fEntrada = new DataInputStream(
    new FileInputStream("Numeros.dat"));
System.out.println ("Los dobles son...");
while(fEntrada.available() != 0) {
    System.out.println (fEntrada.readInt()*2);
}
fEntrada.close();
}
```

```
.in));
n(
;
);
());
```

■ File

- Nos permite acceder al sistema de ficheros
- Crear un objeto a través de:
 - El nombre
 - El nombre a partir de un File padre
 - El nombre a partir de un File hijo
 - La ruta absoluta (*URI – Uniform Resource Identifier*)
- Operaciones comunes sobre archivos

Método	Detalle
<code>boolean canRead()</code> <code>boolean canWrite()</code>	Determina si permite lectura/escritura
<code>boolean exists()</code>	Determina si existe el fichero en el path
<code>boolean isFile()</code>	Determina si es un fichero
<code>boolean isDirectory()</code>	Determina si es un directorio
<code>boolean isHidden()</code>	Determina si está oculto

■ Ejemplo File

□ Listar el contenido de la carpeta raíz C:\

```
import java.io.*;
public class ListaPath
{
    public static void main(String args[])
    {
        try
        {
            File roots [] = File.listRoots();
            for (int i = 0; i<roots.length; i++){
                if(roots[i].getAbsolutePath().equalsIgnoreCase("c:\\"))
                {
                    File files[] = roots[i].listFiles();
                    for (int j = 0; j<files.length; j++){
                        if(files[j].isFile())
                            System.out.println("\t\t<FILE>" +
                                                    files[j].getName());
                        else
                            System.out.println("<DIR>" +
                                                    files[j].getName());
                    }
                }
            }
        }
        catch(Exception e){}
    }
}
```

- ¿Que ocurre cuando un aplicación acaba?
- ¿Cómo puedo recuperar el estado?
- ¿Cómo puedo guardar mis objetos?

serialización

■ Serialización

- Una de las características más potentes de Java es la posibilidad de **serializar** un objeto.
- Convertirlo en una secuencia de bytes y enviarlo a un fichero en disco.
- Por un *socket* a otro ordenador a través de la red, etc

- **Serialización: Proceso**
- Declarar la implementación de la interfaz *Serializable* en la clase que deseemos serializar. Se trata de una interfaz vacía, por lo que no hay operaciones que implementar
- Para serializar el objeto crearíamos un stream *ObjectOutputStream* y escribiríamos el objeto mediante la operación *writeObject ()*
- Para deserializar el objeto crearíamos un stream *ObjectInputStream*, leeríamos el objeto mediante *readObject ()* y realizaríamos un casting a la clase del objeto

■ Serialización

- Lectura y escritura de objetos en ***streams***
- Para poder serializar el objeto debe de cumplir una serie de condiciones
 - La clase debe implementar la interfaz *serializable*
 - Las clases de sus atributos también
 - Si queremos que algún atributo no se serialice
 - *transient*
- Las clases que recuperan/almacenan son:
 - ***ObjectInputStream*** [*readObject()*]
 - ***ObjectOutputStream*** [*writeObject(Object unObjeto)*]

■ Serialización: Ejemplo *Cuenta*

```
import java.io.*;
import java.util.*;

// Es necesario que la clase Movimiento implemente la interfaz Serializable para
// que los objetos puedan ser escritos en disco

public class Cuenta {
    long numero;
    Cliente titular;
    private float saldo;
    float interesAnual;

    LinkedList movimientos;

    static private class Movimiento implements Serializable {
        Date fecha;
        char tipo;
        float importe;
        float saldo;

        public Movimiento (Date aFecha, char aTipo, float aImporte, float aSaldo) {
            fecha = aFecha;
            tipo = aTipo;
            importe = aImporte;
            saldo = aSaldo;
        }
    }
}
```

■ Serialización: Ejemplo *Cuenta*

- Extraigo los movimientos de un fichero.
- Los leo en el constructor

```
Cuenta (long aNumero) throws FileNotFoundException, IOException, ClassNotFoundException {
    ObjectInputStream ois = new ObjectInputStream (new FileInputStream (aNumero + ".cnt"));
    numero = ois.readLong ();
    titular = (Cliente) ois.readObject ();
    saldo = ois.readFloat ();
    interesAnual = ois.readFloat ();
    movimientos = (LinkedList) ois.readObject ();
    ois.close ();
}
```

■ Serialización: Ejemplo *Cuenta*

- Para guardar los movimientos a disco.
 - Abro un fichero.
 - Escribo el objeto serializado

```
void salvar () throws FileNotFoundException, IOException {  
    ObjectOutputStream oos = new ObjectOutputStream (new FileOutputStream (numero + ".cnt"));  
    oos.writeLong (numero);  
    oos.writeObject (titular);  
    oos.writeFloat (saldo);  
    oos.writeFloat (interesAnual);  
    oos.writeObject (movimientos);  
    oos.close ();  
}
```

■ Serialización: Ejemplo

```
import java.io.*;
public class TestSerializacion
{
    public static void main(String args[]){
        Spool sOriginal = new Spool(10);
        Factura fOriginal = new Factura(1,"Enrique","XYZ, s.a",520.50);
        sOriginal.put(fOriginal);
        fOriginal = new Factura(2,"Enrique","AC Enterprise, s.a",1000.00);
        sOriginal.put(fOriginal);
        fOriginal = new Factura(3,"Javier Palomino","One Act, s.a",350.70);
        sOriginal.put(fOriginal);
        //IMPRESION DESDE EL SPOOL ORIGINAL
        System.out.println ("Impresion del Spool ORIGINAL-----");
        System.out.println (sOriginal.get());
        //ALMACENAMIENTO DEL ESTADO DEL SPOOL Y DE LAS FACTURAS PENDIENTES
        try{
            ObjectOutputStream fSalida = new ObjectOutputStream(
                new FileOutputStream("Spool.txt"));
            fSalida.writeObject(sOriginal);
```



Conclusiones

1. Programación Orientada a Objetos
2. Introducción y Sintaxis Java
3. Sentencias Control Flujo
4. POO en Java
5. Relaciones entre Objetos
6. Polimorfismo, abstracción e interfaces
7. Excepciones
8. **Conceptos avanzados**

■ Entrada / Salida

- Paquete I/O
- Entrada Estándar
- Salida Estándar
- Fichero
 - File[I/O]Stream
 - Data[I/O]Stream
 - File
- Serialización