

# TALLER DE LENGUAJE DE PROGRAMACIÓN I

## Entorno de CONSOLA y GRÁFICOS

### UNIDAD 1

#### Escritura y Lectura de Archivos Planos Secuenciales

JAVA

# OBJETIVO

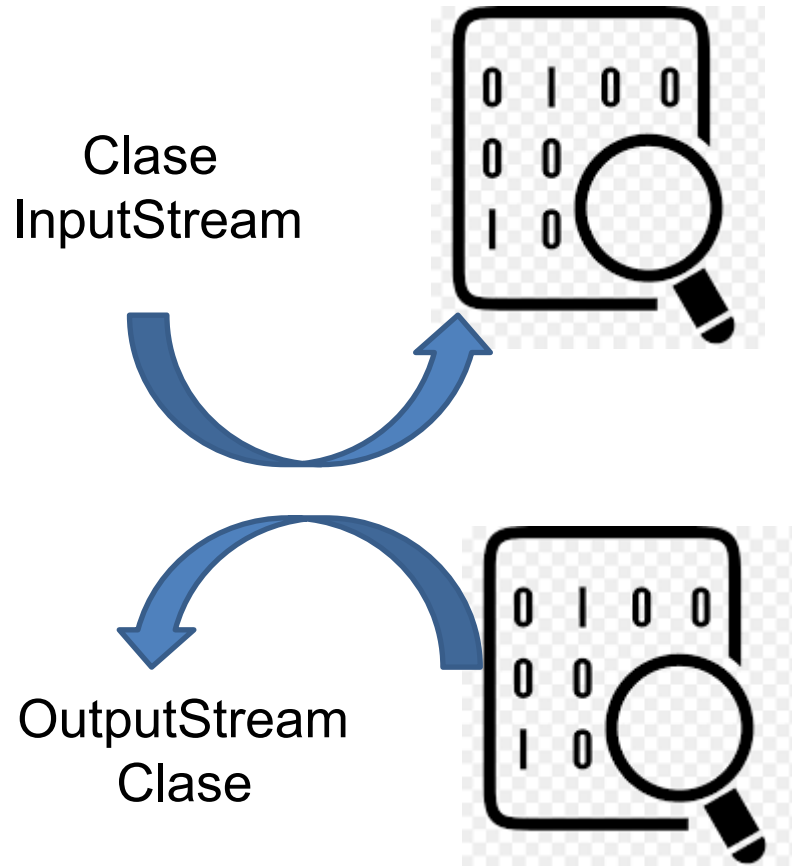
- Lectura y Escritura de Archivos Planos Secuenciales
  - Secuencias (STREAMS)
    - Flujo de byte
    - Flujo de caracteres
  - Manejo de Errores
  - Buffers

## Trabajo Independiente

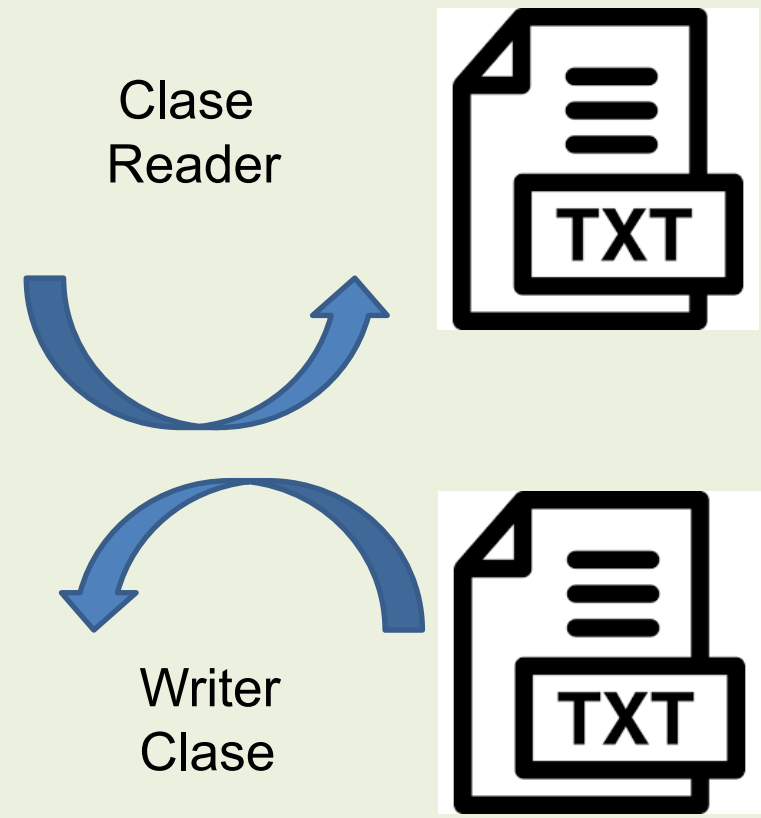
- Ejercicio

# Secuencias (STREAMS)

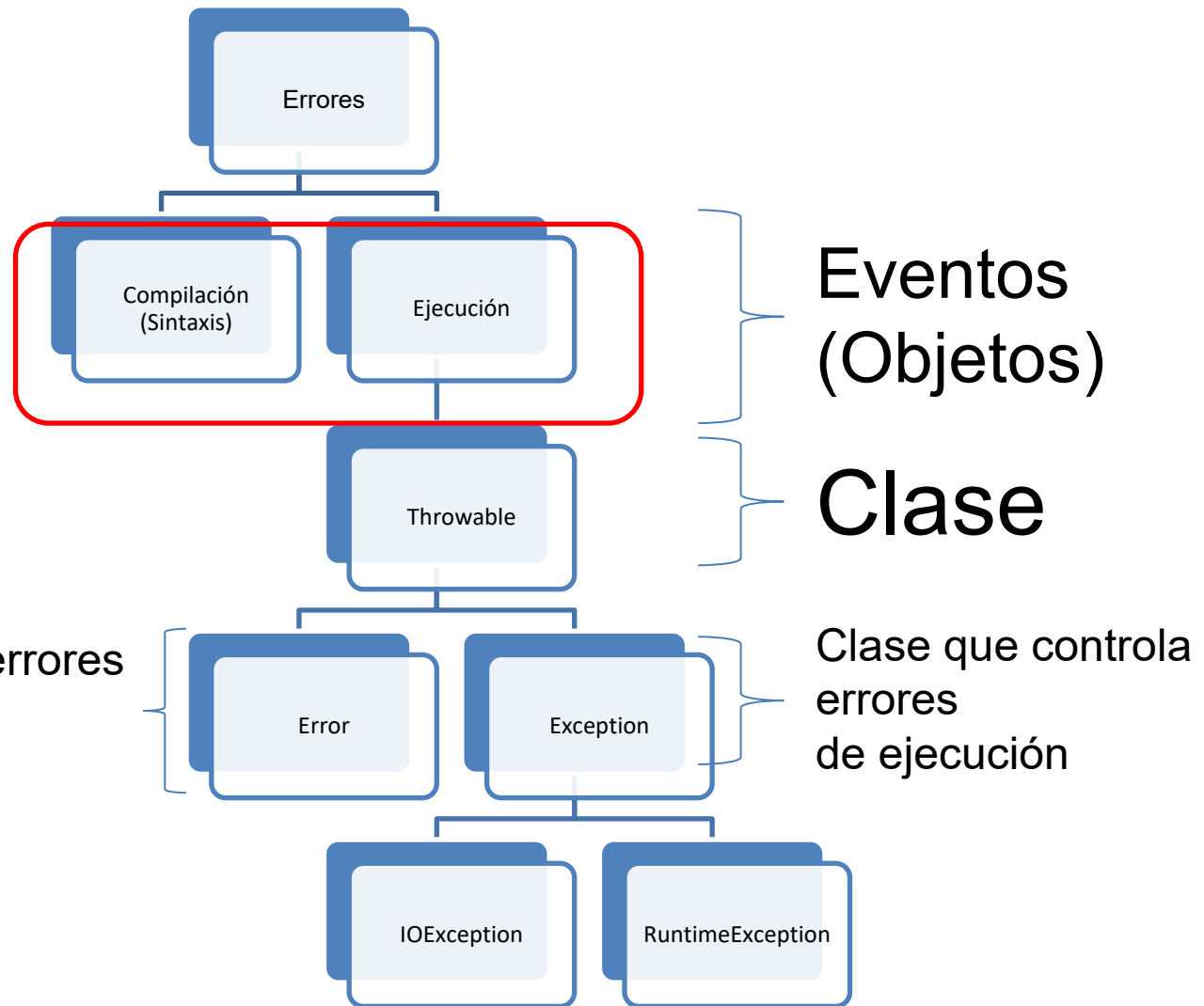
## Flujo de byte



## Flujo de caracteres



# Manejo de errores



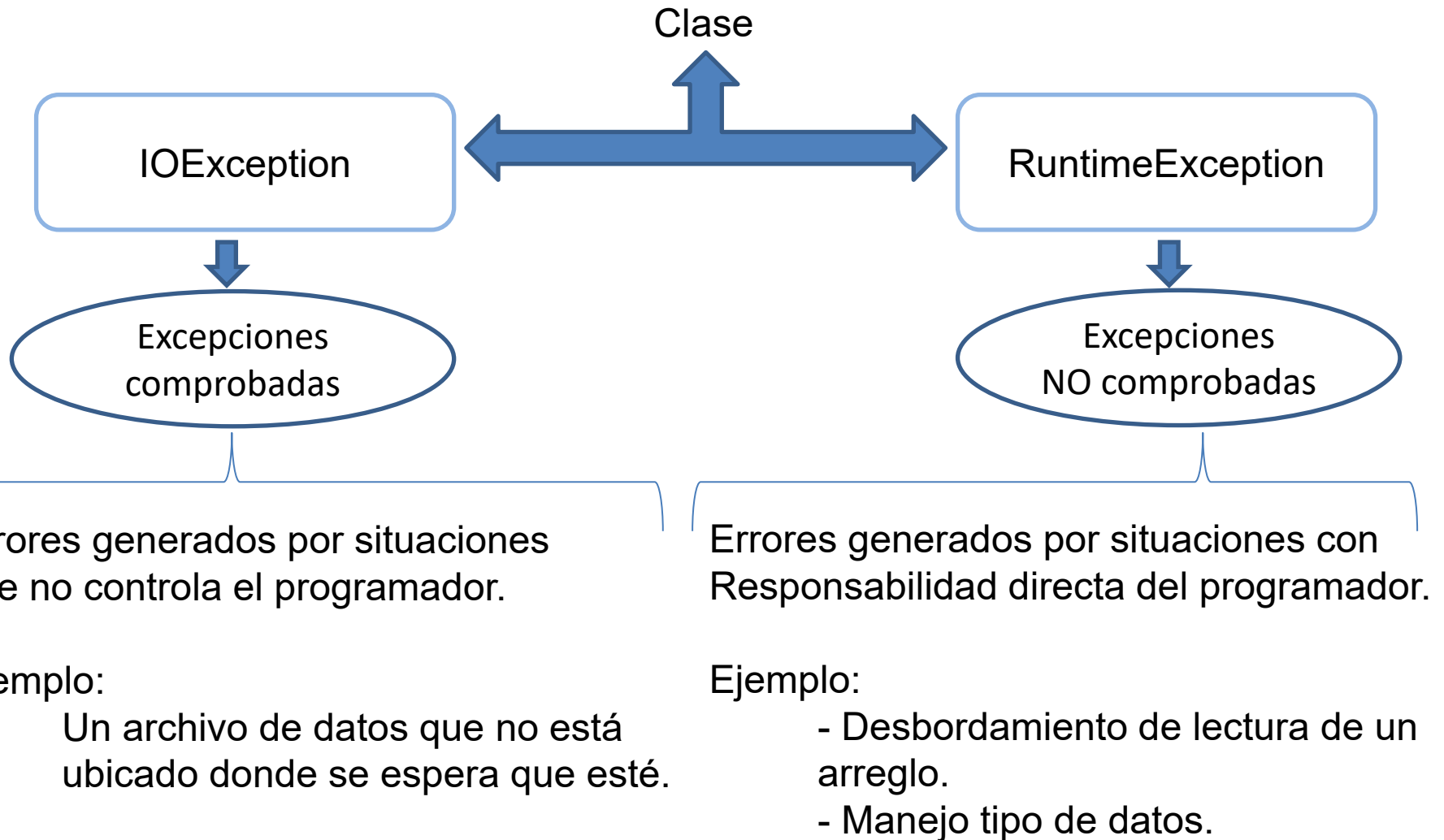
Clase que controla errores de maquina

Eventos (Objetos)

Clase

Clase que controla errores de ejecución

# Manejo de errores, continuación



# Manejo de errores, continuación

- **Ejemplo de Errores NO comprobados**

package excepciones;

```
public class Excepciones1 {
```

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

```
    {  
        // TODO Auto-generated method stub
```

```
        int[] arreglo = new int[3];
```

```
        arreglo[0] = 1;
```

```
        arreglo[1] = 2;
```

```
        arreglo[2] = 3;
```

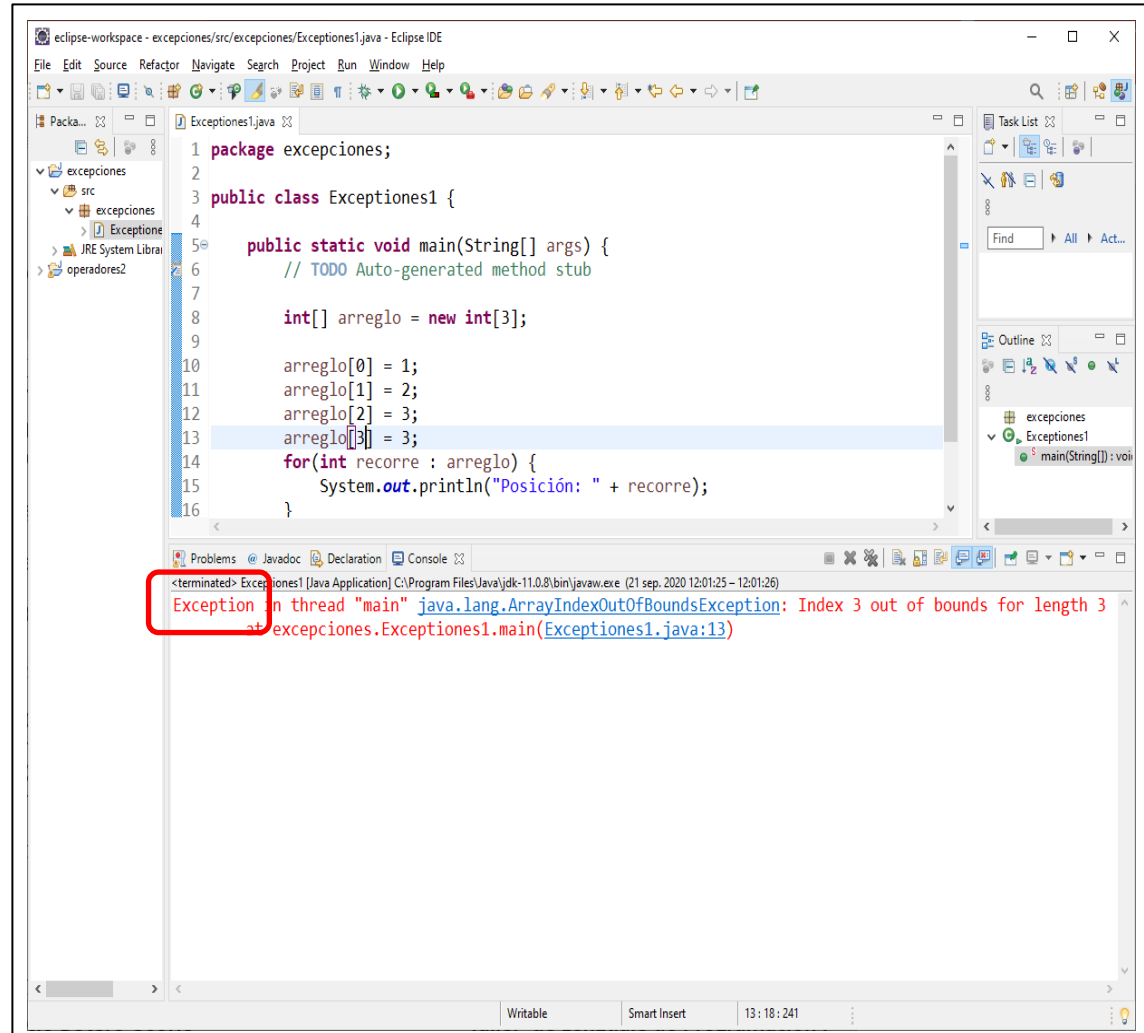
```
        for(int recorre : arreglo)
```

```
        {  
            System.out.println("Posición: " + recorre);
```

```
        }
```

```
    }
```

```
}
```



# Lectura y Escritura de Archivos Planos Secuenciales

## Clase Reader (Lectura de Archivos Planos Secuenciales)

Curso Java. Streams I. Accediend... x Reader (Java Platform SE 7) x +

https://docs.oracle.com/javase/7/docs/api/

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.io

**Class Reader**

java.lang.Object  
java.io.Reader

**All Implemented Interfaces:**

Closeable, AutoCloseable, Readable

**Direct Known Subclasses:**

BufferedReader, CharArrayReader, FilterReader, **InputStreamReader**, PipedReader, StringReader

public abstract class Reader  
extends Object  
implements Readable, Closeable

Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

**Since:**

JDK1.1

**See Also:**

BufferedReader, LineNumberReader, CharArrayReader, InputStreamReader, FileReader, FilterReader, PushbackReader, PipedReader, StringReader, Writer

**Field Summary**

**Fields**

Modifier and Type	Field and Description
protected Object	lock The object used to synchronize operations on this stream.

**Constructor Summary**

**Constructors**

Modifier	Constructor and Description
protected	Reader() Creates a new character-stream reader whose critical sections will synchronize on the reader itself.
protected	Reader(Object lock) Creates a new character-stream reader whose critical sections will synchronize on the given object.

# Lectura y Escritura de Archivos Planos Secuenciales

## Clase InputStreamReades

The screenshot shows the Oracle Java API documentation for the `java.io.InputStreamReader` class. The browser address bar shows the URL `https://docs.oracle.com/javase/7/docs/api/`. The left sidebar contains a list of Java packages and classes, with `java.io` selected. The main content area displays the class details for `InputStreamReader`.

**Class Overview:**

- Package: `java.io`
- Superclasses: `java.lang.Object`, `java.io.Reader`, `java.io.InputStreamReader`
- All Implemented Interfaces: `Closeable`, `AutoCloseable`, `Readable`
- Direct Known Subclasses: `FileReader`

**Class Definition:**

```
public class InputStreamReader
    extends Reader
```

**Description:**

An `InputStreamReader` is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Each invocation of one of an `InputStreamReader`'s `read()` methods may cause one or more bytes to be read from the underlying byte-input stream. To enable the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

For top efficiency, consider wrapping an `InputStreamReader` within a `BufferedReader`. For example:

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

**Since:** JDK1.1

**See Also:** `BufferedReader`, `InputStream`, `Charset`

**Field Summary:**

Fields inherited from class `java.io.Reader`

Field
<code>lock</code>

**Constructor Summary:**

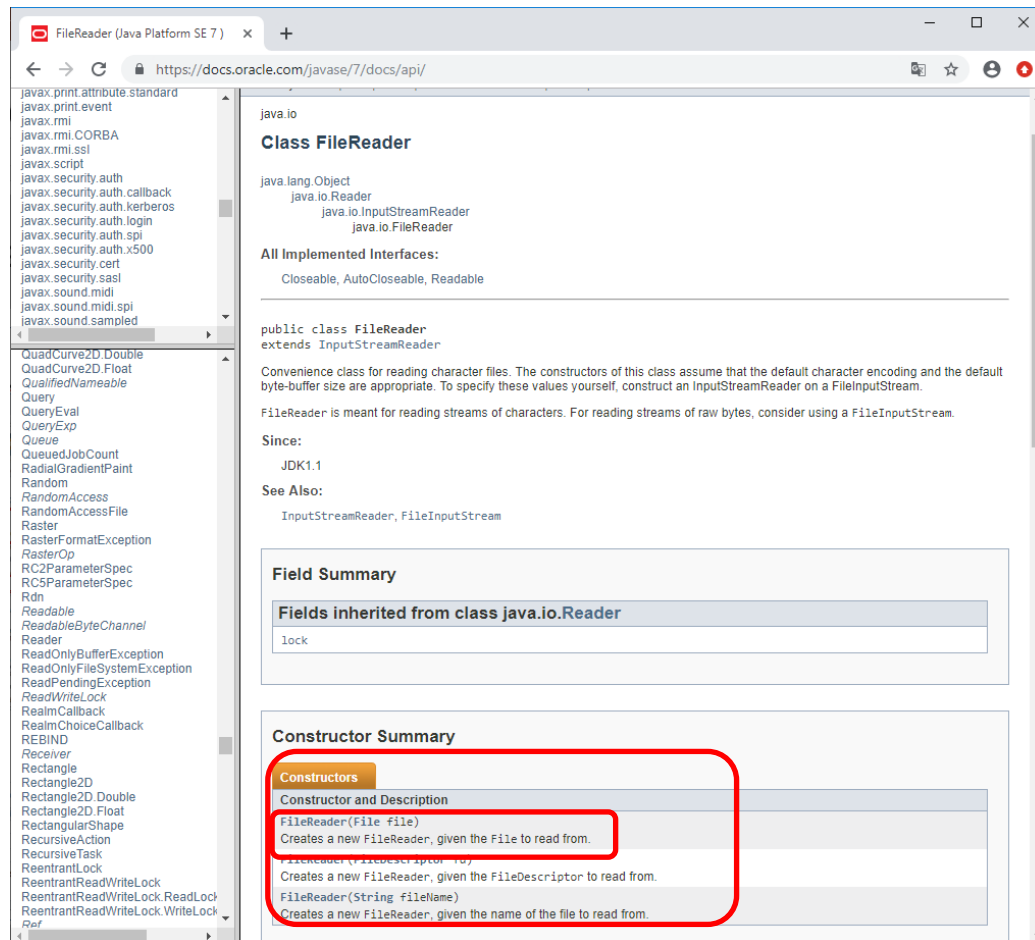
Constructors

Constructor	Description
-------------	-------------



# Lectura y Escritura de Archivos Planos Secuenciales

## Clase – Constructor **FileReader**



The screenshot shows the Java Platform SE 7 API documentation for the `FileReader` class. The browser address bar shows `https://docs.oracle.com/javase/7/docs/api/`. The left sidebar lists various Java packages, with `java.io` selected. The main content area displays the `FileReader` class details.

**Class FileReader**

java.lang.Object  
java.io.Reader  
java.io.InputStreamReader  
java.io.FileReader

**All Implemented Interfaces:**

Closeable, AutoCloseable, Readable

**public class FileReader**  
extends `InputStreamReader`

Convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an `InputStreamReader` on a `FileInputStream`.

`FileReader` is meant for reading streams of characters. For reading streams of raw bytes, consider using a `FileInputStream`.

**Since:**  
JDK1.1

**See Also:**  
`InputStreamReader`, `FileInputStream`

**Field Summary**

**Fields inherited from class java.io.Reader**

`lock`

**Constructor Summary**

**Constructors**

Constructor and Description
<code>FileReader(File file)</code> Creates a new <code>FileReader</code> , given the <code>File</code> to read from.
<code>FileReader(FileDescriptor fdObj)</code> Creates a new <code>FileReader</code> , given the <code>FileDescriptor</code> to read from.
<code>FileReader(String fileName)</code> Creates a new <code>FileReader</code> , given the name of the file to read from.

# Lectura y Escritura de Archivos Planos Secuenciales

## Constructor FileReader

The screenshot shows the Java API documentation for the `FileReader` class. The left sidebar lists various Java packages, and the main content area displays the class details. Three constructors are listed, with the second one highlighted by a red rectangle.

**Constructor Detail**

**FileReader**

```
public FileReader(String fileName)
    throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

**Parameters:**

- `fileName` - the name of the file to read from

**Throws:**

- `FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

**FileReader**

```
public FileReader(File file)
    throws FileNotFoundException
```

Creates a new `FileReader`, given the `File` to read from.

**Parameters:**

- `file` - the `File` to read from

**Throws:**

- `FileNotFoundException` - if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

**FileReader**

```
public FileReader(FileDescriptor fd)
```

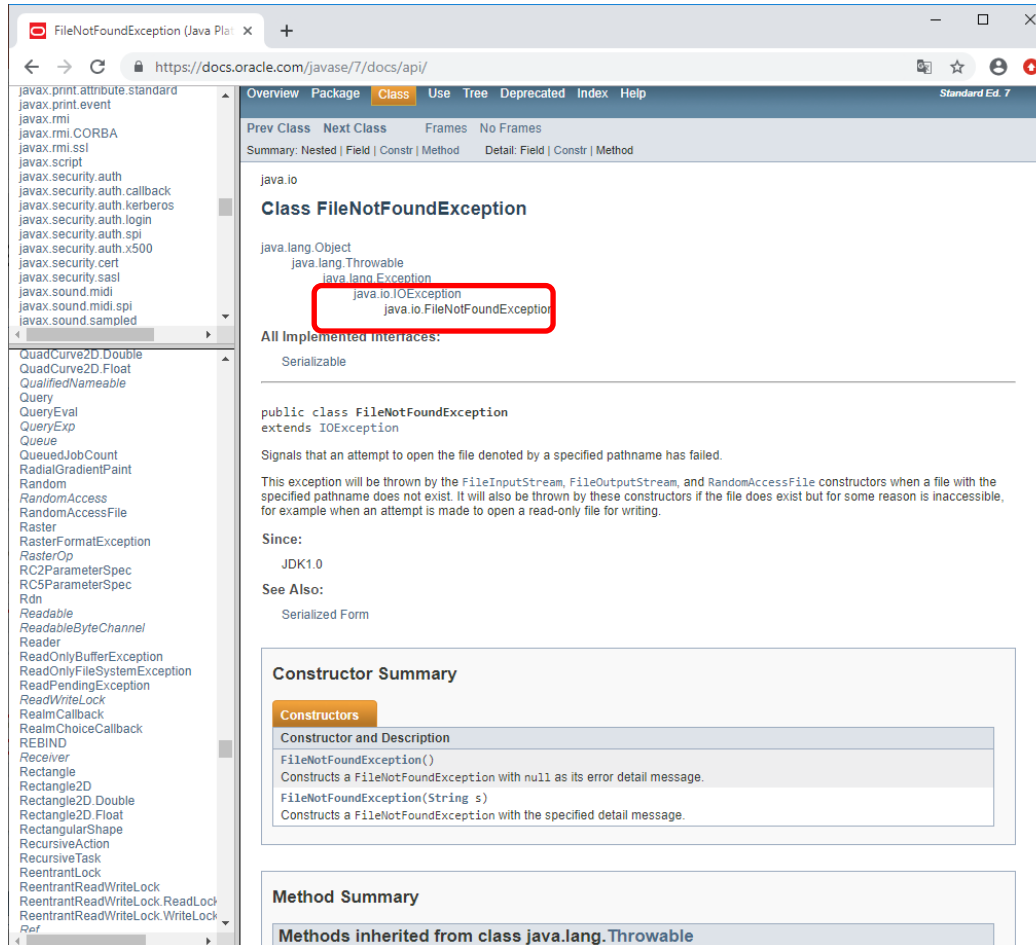
Creates a new `FileReader`, given the `FileDescriptor` to read from.

**Parameters:**

- `fd` - the `FileDescriptor` to read from

# Lectura y Escritura de Archivos Planos Secuenciales

## Lanzamiento de la Exception de `FileNotFoundException`



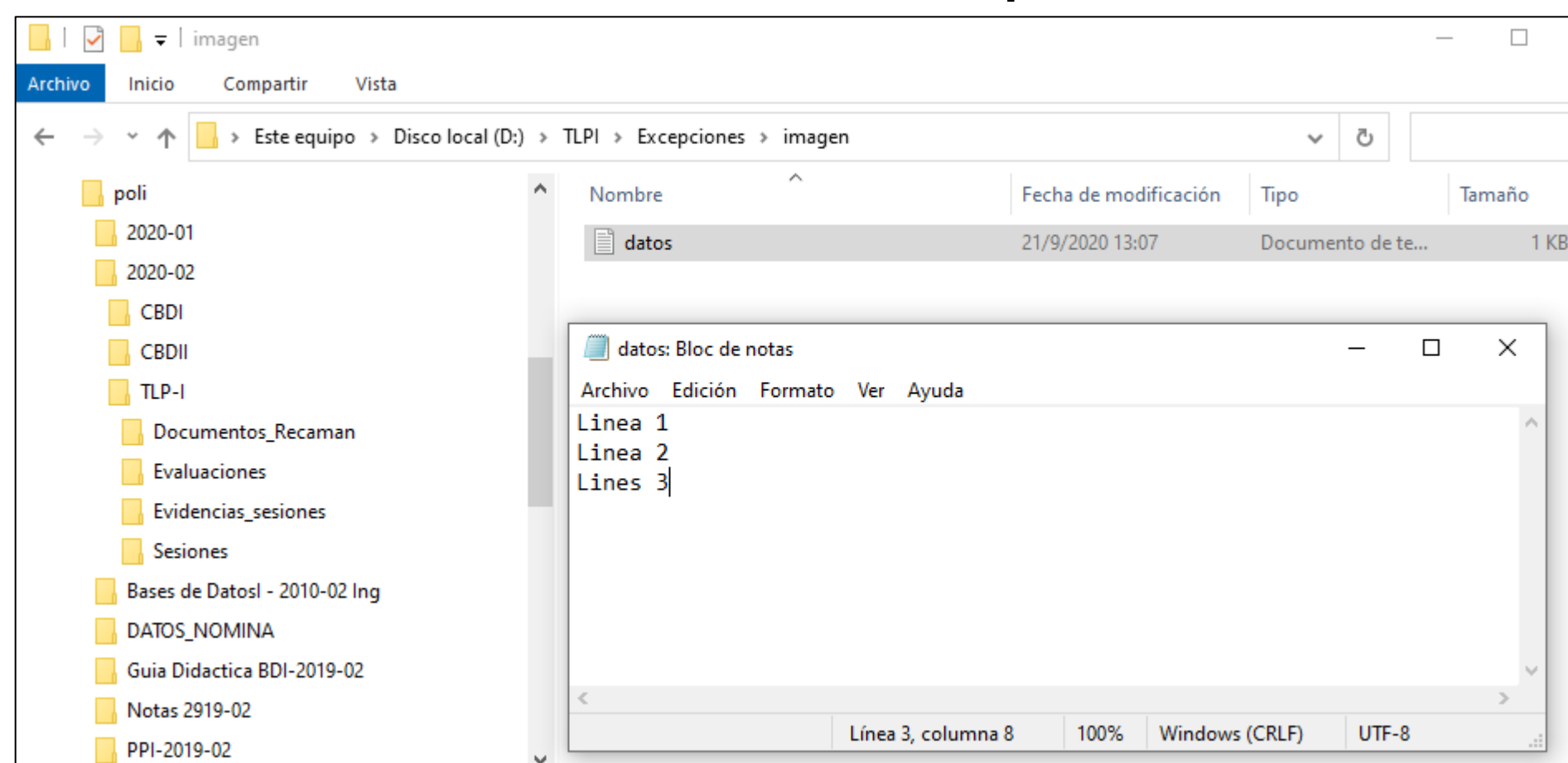
The screenshot displays the Oracle Java API documentation for the `FileNotFoundException` class. The browser address bar shows the URL `https://docs.oracle.com/javase/7/docs/api/`. The left sidebar contains a list of Java packages and classes, with `java.io.FileNotFoundException` selected. The main content area shows the class hierarchy: `java.io` → `java.lang.Object` → `java.lang.Throwable` → `java.lang.Exception` → `java.io.IOException` → `java.io.FileNotFoundException`. The `FileNotFoundException` class is highlighted with a red box. Below the hierarchy, it lists "All Implemented Interfaces: Serializable". The class definition is shown as `public class FileNotFoundException extends IOException`. A description states: "Signals that an attempt to open the file denoted by a specified pathname has failed." It also mentions that this exception is thrown by `FileInputStream`, `FileOutputStream`, and `RandomAccessFile` constructors when a file with the specified pathname does not exist or is inaccessible. The "Since:" field indicates it was introduced in JDK 1.0. The "See Also:" field points to "Serialized Form". The "Constructor Summary" section lists two constructors: `FileNotFoundException()` and `FileNotFoundException(String s)`. The "Method Summary" section indicates that methods are inherited from `java.lang.Throwable`.

## Control de la Exception

```
Try
{
    .....
}
catch(Captura el error)
{
    .....
}
catch(Captura el error)
{
    .....
}
```

# Lectura y Escritura de Archivos Planos Secuenciales

## Definir un archivo de datos para leer



# Lectura y Escritura de Archivos Planos Secuenciales

## Definir la clase que lea el archivo

```
package excepciones;

import java.io.*;

public class LeerArchivo
{
    public static void main(String[] args)
    {
        try
        {
            FileReader archivo = new FileReader("D:\\TLPI\\Excepciones\\imagen\\datos.txt");

            int c=0;
            while (c != -1) {
                c = archivo.read();
                System.out.print(((char)c));
            }
            archivo.close();
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("No encontro el archivo");
        }
    }
}
```

# Lectura y Escritura de Archivos Planos Secuenciales

## Definir la clase que cree un archivo

```
package excepciones;

import java.io.*;

import javax.swing.JOptionPane;

public class EscribirArchivo
{
    public static void main(String[] args)
    {
        try
        {
            FileWriter crear = new FileWriter("D:\\TLPI\\Excepciones\\imagen\\datosSalida.txt");

            crear.write(JOptionPane.showInputDialog(null, "Valor"));

            crear.close();
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

# Lectura y Escritura de Archivos Planos Secuenciales

## Definir la clase que cree un archivo multilinea

```
package excepciones;

import java.io.*;
import javax.swing.JOptionPane;

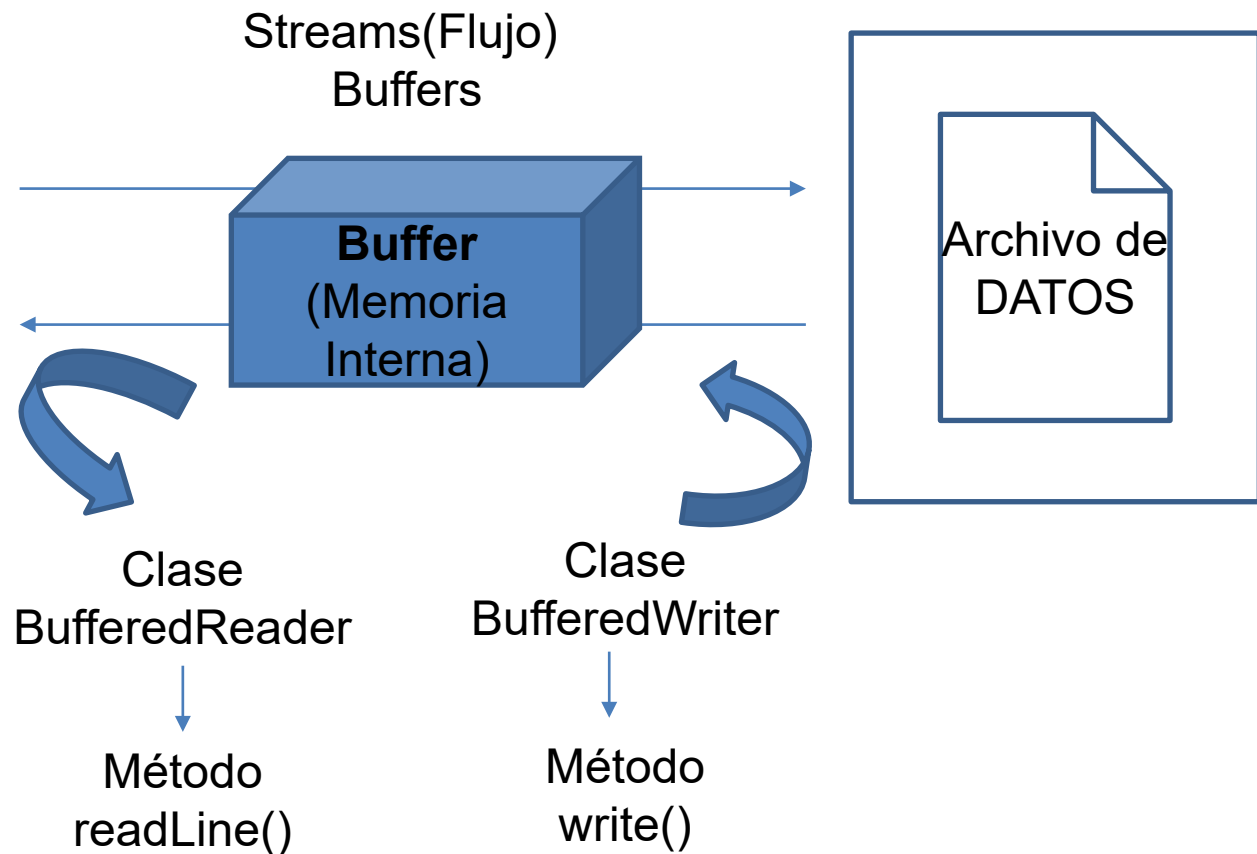
public class EscribeArchivoAdiciona
{
    public static void main(String[] args)
    {
        try
        {
            FileWriter crear = new FileWriter("D:\\TLPI\\Excepciones\\imagen\\datosSalida.txt");
            PrintWriter linea_crear = new PrintWriter(crear);
            for (int i = 0; i < 5; i++)
            {
                linea_crear.println(JOptionPane.showInputDialog(null,"Valor"));
            }
            crear.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



# Lectura y Escritura de Archivos Planos Secuenciales

## Buffers

```
// Leer_Fichero.java
class Leer_Fichero{
    public void lee(){
        try {
            entrada=new FileReader("C:/Users/Juan/Desktop/ejemplo.txt");
            int c=0;
            while(c!=-1){
                c=entrada.read();
                char letra=(char)c;
                System.out.print(letra);
            }
            //entrada.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            System.out.println("No se ha encontrado el archivo");
        }finally{
            try {
                entrada.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```



# Lectura y Escritura de Archivos Planos Secuenciales

## Buffers Escribir

```
package banco;  
import java.io.*;
```

```
public class CrearAbonadoBufferWrite  
{
```

```
    public CrearAbonadoBufferWrite() {}
```

```
    public void CrearAbonadoMultiple()  
    {
```

```
        try  
        {
```

```
            FileWriter archivo = new FileWriter("D:\\TLPI\\banco\\datos\\AbonadosMultiples.txt",true);
```

```
            String wid;
```

```
            BufferedReader datos = new BufferedReader(new InputStreamReader(System.in));
```

```
            BufferedWriter escribir = new BufferedWriter(archivo);
```

```
            PrintWriter linea = new PrintWriter(escribir);
```

```
            do
```

```
            {
```

```
                System.out.println("Ingrese la Identificación, cero(0) para terminar");
```

```
                wid = datos.readLine();
```

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

```
                if(!wid.equals("0"))
```

```
                {
```

```
                    linea.append(wid + "\n");
```

```
                }
```

```
            }while(!wid.equals("0"));
```

```
            linea.close();
```

```
        }catch(IOException e) {}
```

```
    }
```

```
}
```

Permite adicionar  
lineas al final del  
archivo

Retorna un String

# Lectura y Escritura de Archivos Planos Secuenciales

## Buffers Leer

```
package banco;

import java.io.*;
public class ListarAbonados
{
    public ListarAbonados() {}

    public void MostrarAbonados()
    {
        try
        {
            FileReader datos = new FileReader("D:\\TLPI\\banco\\datos\\AbonadosMultiples.txt");
            BufferedReader bufferdatos = new BufferedReader(datos);
            String linea = "";
            while (linea != null)
            {
                linea = bufferdatos.readLine();
                if(linea != null)
                {
                    System.out.println(linea);
                }
            }
        }catch(IOException e) {}
    }
}
```

# Trabajo Independiente

- Desarrollar un caso de verificación de movimientos de Débito y Crédito en cuenta bancaria de usuarios