UNIVERSIDAD DE ALMERÍA

Grado en Ingeniería Informática
Metodología de la programación
2013

Persistencia. Archivos de texto

- Librería java.io del JDK
- La clase File
- Reader y Writer

Archivos

Un archivo o fichero es una colección de datos homogéneos almacenados en un soporte físico del computador que puede ser permanente.

Datos homogéneos: Almacena colecciones de datos del mismo tipo (igual que arrays / vectores)

Cada elemento almacenado en un fichero se denomina registro, que se compone de campos.

Puede ser almacenado en diversos soportes (Disco duro, disquete, …)

Tipos de operaciones

- Operación de **Creación**
- Operación de **Apertura**. Varios modos:
  - Sólo lectura
  - Sólo escritura
  - Lectura y Escritura
- Operaciones de **lectura / escritura**
- Operaciones de **inserción / borrado**
- Operaciones de **renombrado / eliminación**
- Operación de **desplazamiento** dentro de un fichero
- Operación de **cierre**

Operaciones para el manejo habitual de un fichero:

1.- **Crearlo** (sólo si no existía previamente)
2.- **Abrirlo**
3.- **Operar** sobre él (lectura/escritura, inserción, borrado, etc.)
4.- **Cerrarlo**

Clasificación de los ficheros según la organización de los registros en memoria:

- **Organización Secuencial**: Registros almacenados consecutivamente en memoria
según el orden lógico en que se han ido insertando.

- **Organización Directa o Aleatoria**: El orden físico de almacenamiento en memoria puede no coincidir con el orden en que han sido insertados.

- **Organización Indexada.**
    - Dos ficheros:
    - Fichero de datos: Información
    - Fichero de índice: Contiene la posición de cada uno de los registros en el fichero de datos

Clasificación de los ficheros según el acceso a la información almacenada:

- **Acceso secuencial:** Para acceder a un registro es necesario pasar por todos los anteriores. Ej: Cinta de Casete
- **Acceso directo o aleatorio:** Se puede acceder a un registro sin pasar por todos los anteriores. Ej: Disco Duro.

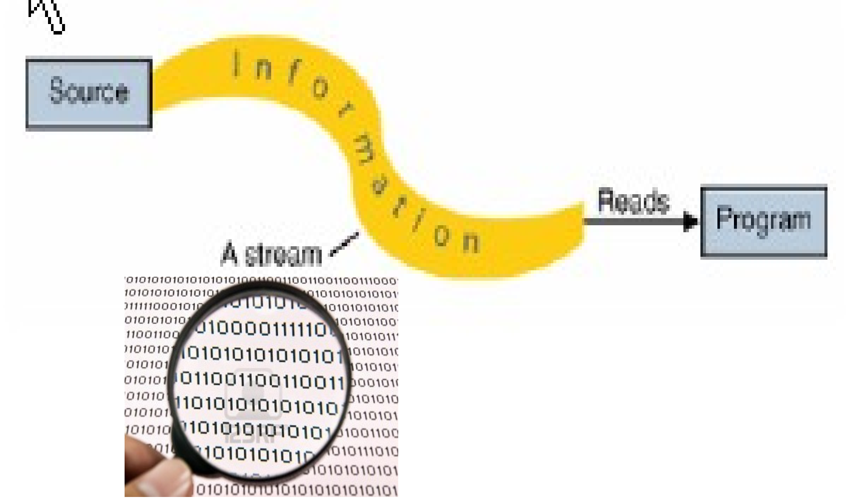
Clasificación de los ficheros según el tipo de la información almacenada:

- **Ficheros Binarios**: Almacenan secuencias de dígitos binarios (ej: ficheros que almacenan enteros, floats,…)
- **Ficheros de Texto**: Almacenan caracteres alfanuméricos en un formato estándar (ASCII, Unicode, UTF8, UTF16, etc.). Pueden ser leídos y/o modificados por aplicaciones denominadas editores de texto (Ej: Notepad, etc.).

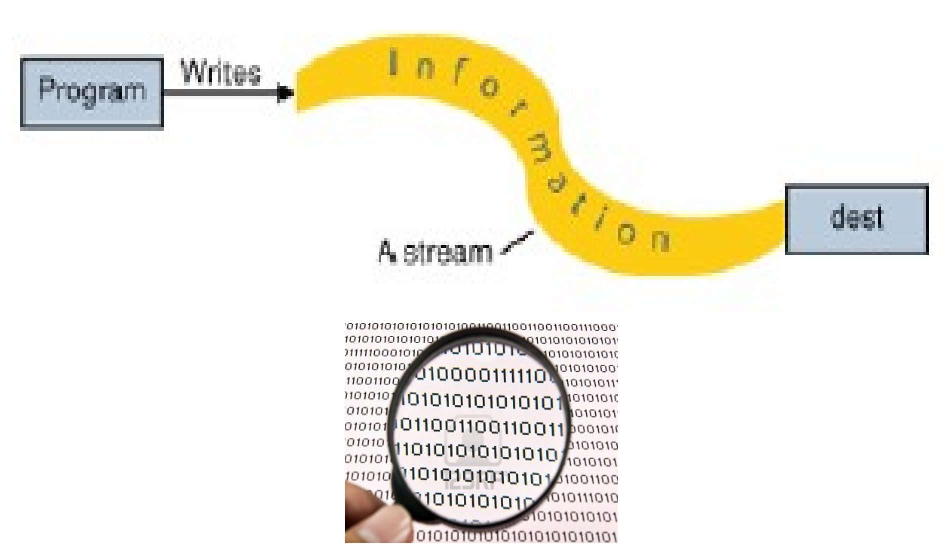
ENTRADA/SALIDA DE DATOS EN JAVA.

Los programas necesitan comunicarse con su entorno, tanto para recoger datos e información que deben procesar, como para devolver los resultados obtenidos. La manera de representar estas entradas y salidas en ***Java*** es a base de ***streams*** (flujos de datos). Un ***stream*** es una conexión entre el programa y la fuente o destino de los datos. La información se traslada ***en serie*** (un carácter a continuación de otro) a través de esta conexión. Esto da lugar a una forma general de representar muchos tipos de comunicaciones.

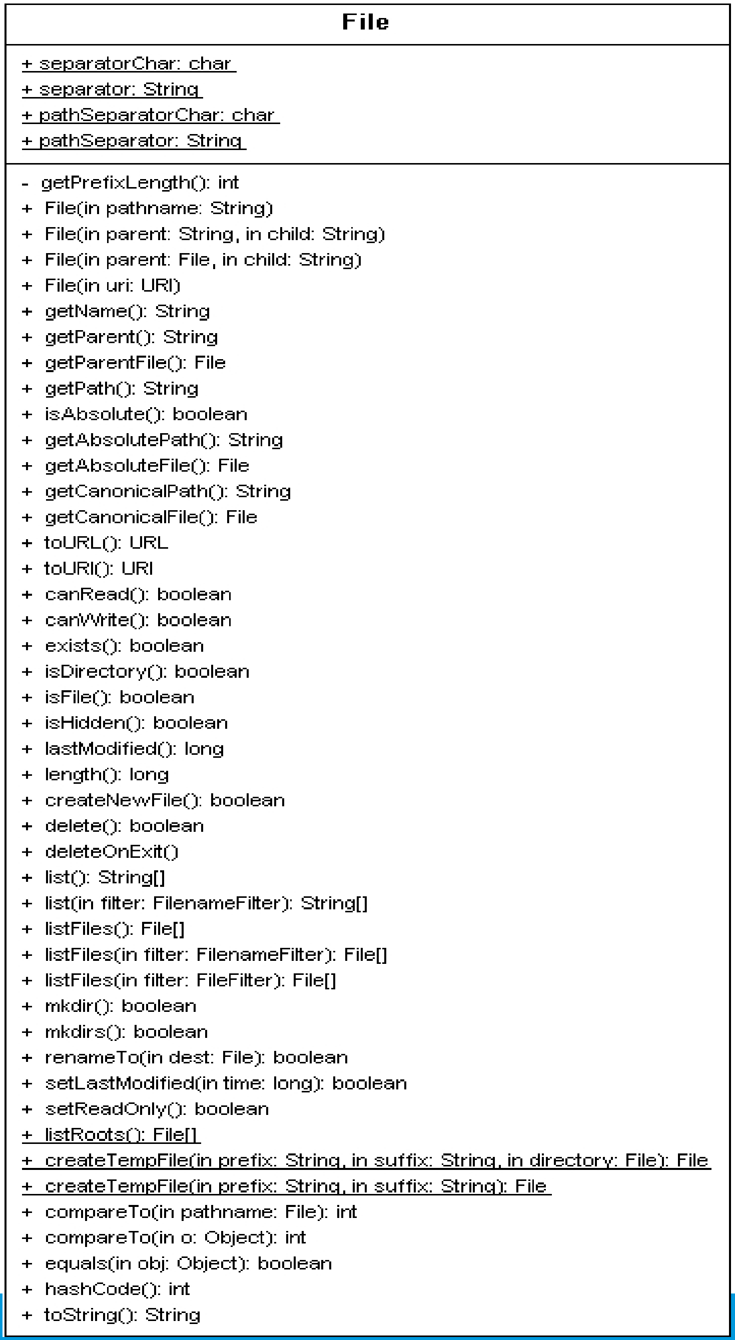
Por ejemplo, cuando se quiere imprimir algo en pantalla, se hace a través de un stream que conecta el monitor al programa. Se da a ese stream la orden de escribir algo y éste lo traslada a la pantalla. Este concepto es suficientemente general para representar la lectura/escritura de archivos, la comunicación a través de Internet o la lectura de la información de un sensor a través del puerto en serie.



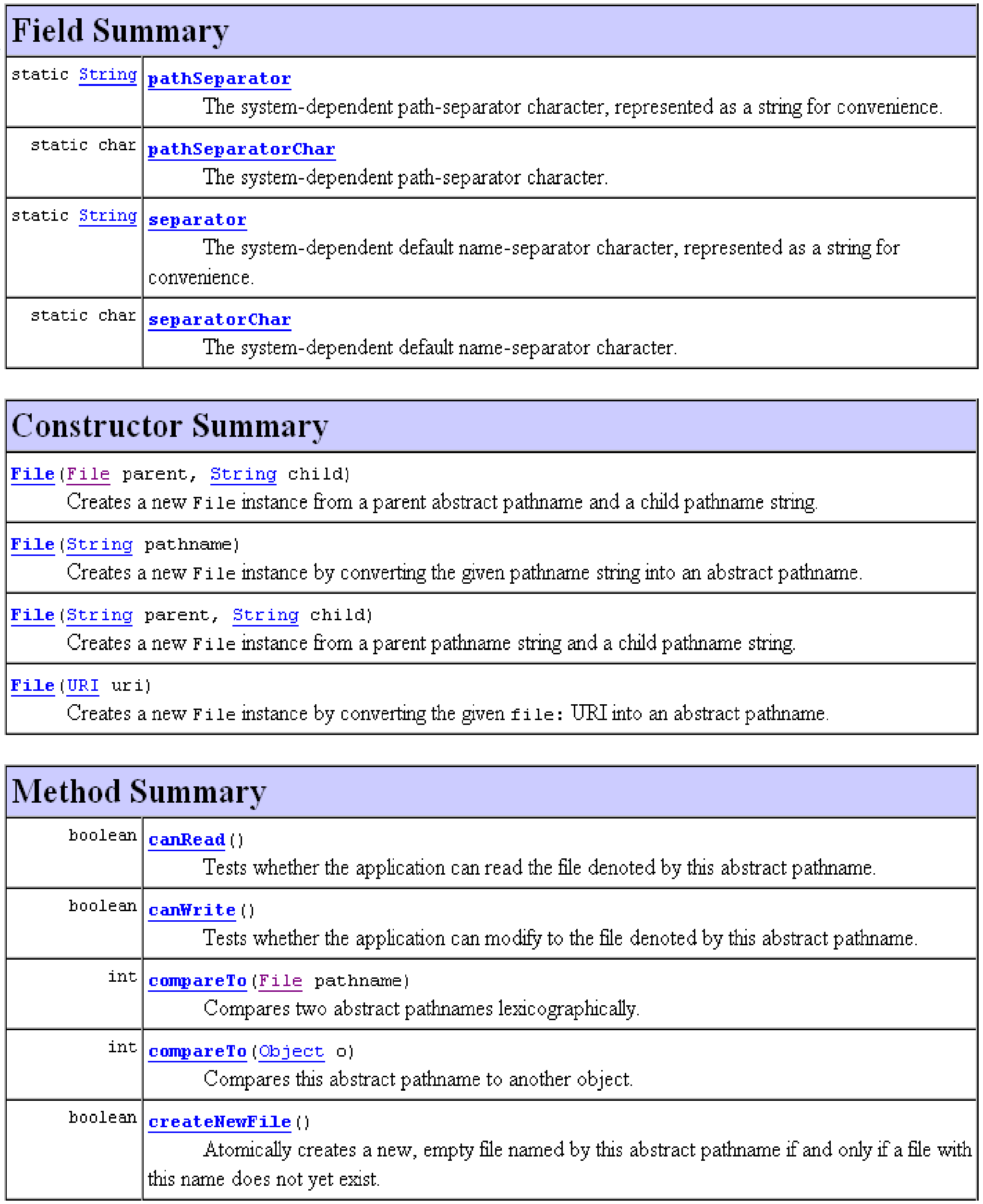
Source
Information
Reads
Program
A stream



Program
Writes
Information
dest
A stream



```
File
---------------------------------------------
+ separatorChar: char
+ separator: String
+ pathSeparatorChar: char
+ pathSeparator: String
---------------------------------------------
-  getPrefixLength(): int
+  File(in pathname: String)
+  File(in parent: String, in child: String)
+  File(in parent: File, in child: String)
+  File(in uri: URI)
+  getName(): String
+  getParent(): String
+  getParentFile(): File
+  getPath(): String
+  isAbsolute(): boolean
+  getAbsolutePath(): String
+  getAbsoluteFile(): File
+  getCanonicalPath(): String
+  getCanonicalFile(): File
+  toURL(): URL
+  toURI(): URI
+  canRead(): boolean
+  canWrite(): boolean
+  exists(): boolean
+  isDirectory(): boolean
+  isFile(): boolean
+  isHidden(): boolean
+  lastModified(): long
+  length(): long
+  createNewFile(): boolean
+  delete(): boolean
+  deleteOnExit()
+  list(): String[]
+  list(in filter: FilenameFilter): String[]
+  listFiles(): File[]
+  listFiles(in filter: FilenameFilter): File[]
+  listFiles(in filter: FileFilter): File[]
+  mkdir(): boolean
+  mkdirs(): boolean
+  renameTo(in dest: File): boolean
+  setLastModified(in time: long): boolean
+  setReadOnly(): boolean
+  listRoots(): File[]
+  createTempFile(in prefix: String, in suffix: String, in directory: File): File
+  createTempFile(in prefix: String, in suffix: String): File
+  compareTo(in pathname: File): int
+  compareTo(in o: Object): int
+  equals(in obj: Object): boolean
+  hashCode(): int
+  toString(): String
```



Field Summary

static String	**pathSeparator** The system-dependent path-separator character, represented as a string for convenience.
static char	**pathSeparatorChar** The system-dependent path-separator character.
static String	**separator** The system-dependent default name-separator character, represented as a string for convenience.
static char	**separatorChar** The system-dependent default name-separator character.

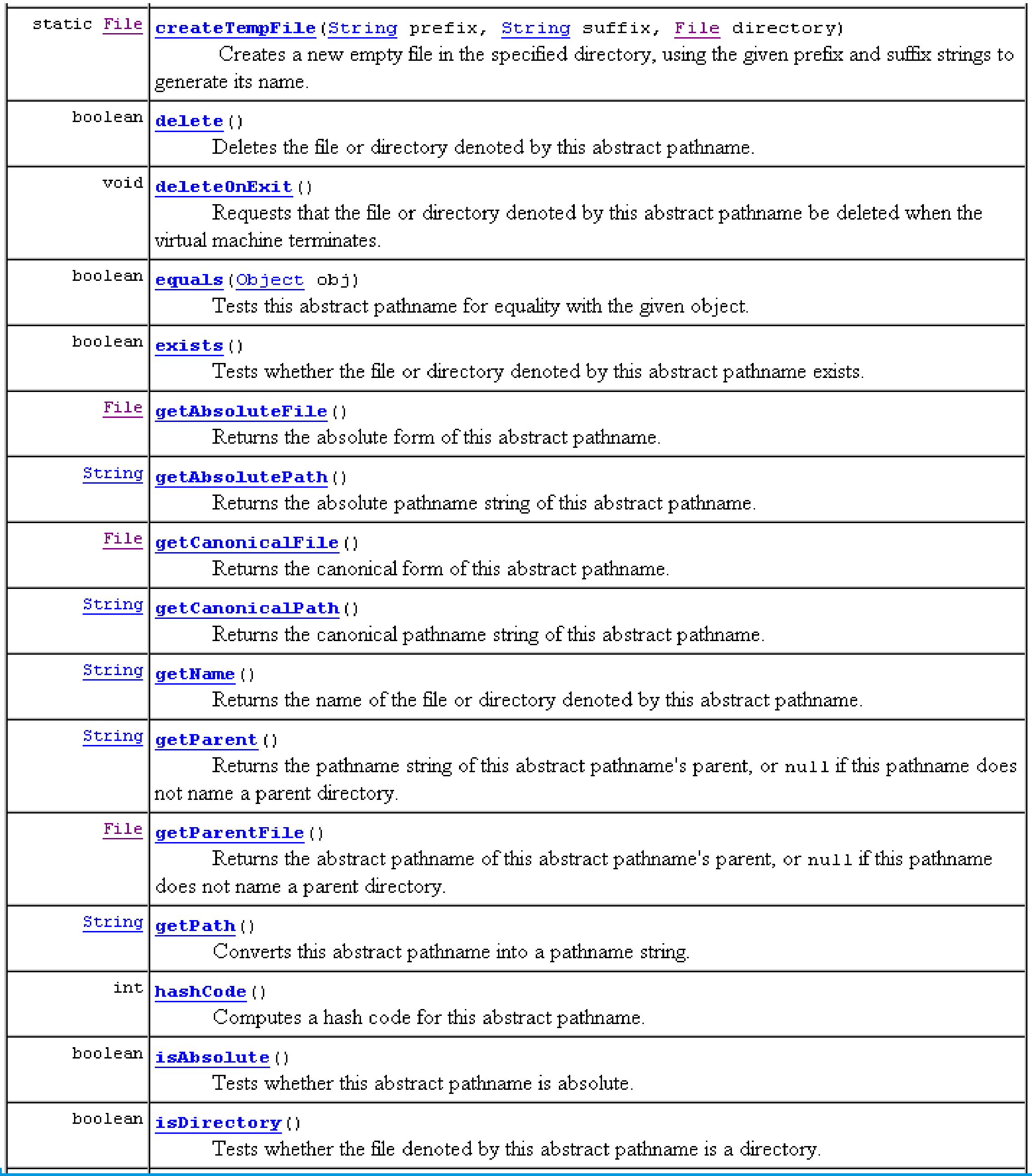
Constructor Summary

File(File parent, String child) Creates a new `File` instance from a parent abstract pathname and a child pathname string.
File(String pathname) Creates a new `File` instance by converting the given pathname string into an abstract pathname.
File(String parent, String child) Creates a new `File` instance from a parent pathname string and a child pathname string.
File(URI uri) Creates a new `File` instance by converting the given `file:` URI into an abstract pathname.

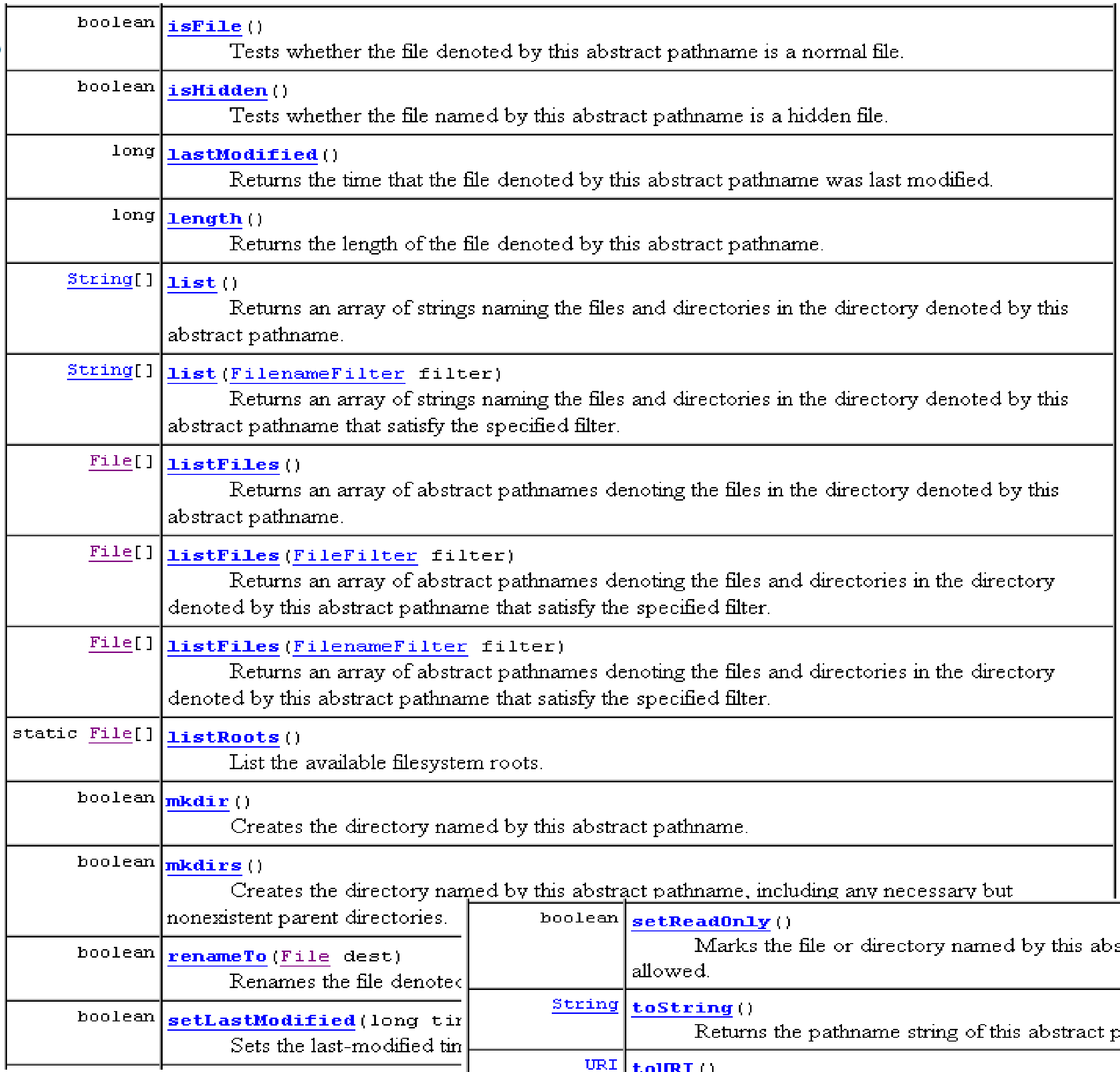
Method Summary

boolean	**canRead**() Tests whether the application can read the file denoted by this abstract pathname.
boolean	**canWrite**() Tests whether the application can modify to the file denoted by this abstract pathname.
int	**compareTo**(File pathname) Compares two abstract pathnames lexicographically.
int	**compareTo**(Object o) Compares this abstract pathname to another object.
boolean	**createNewFile**() Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

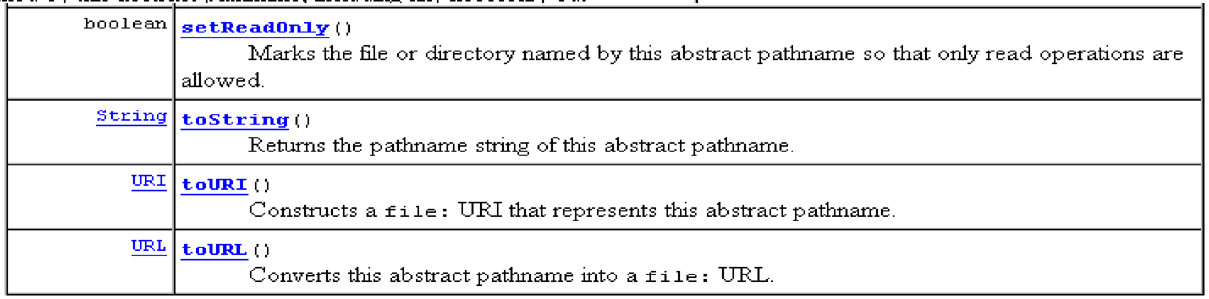
UNIVERSIDAD DE ALMERÍA



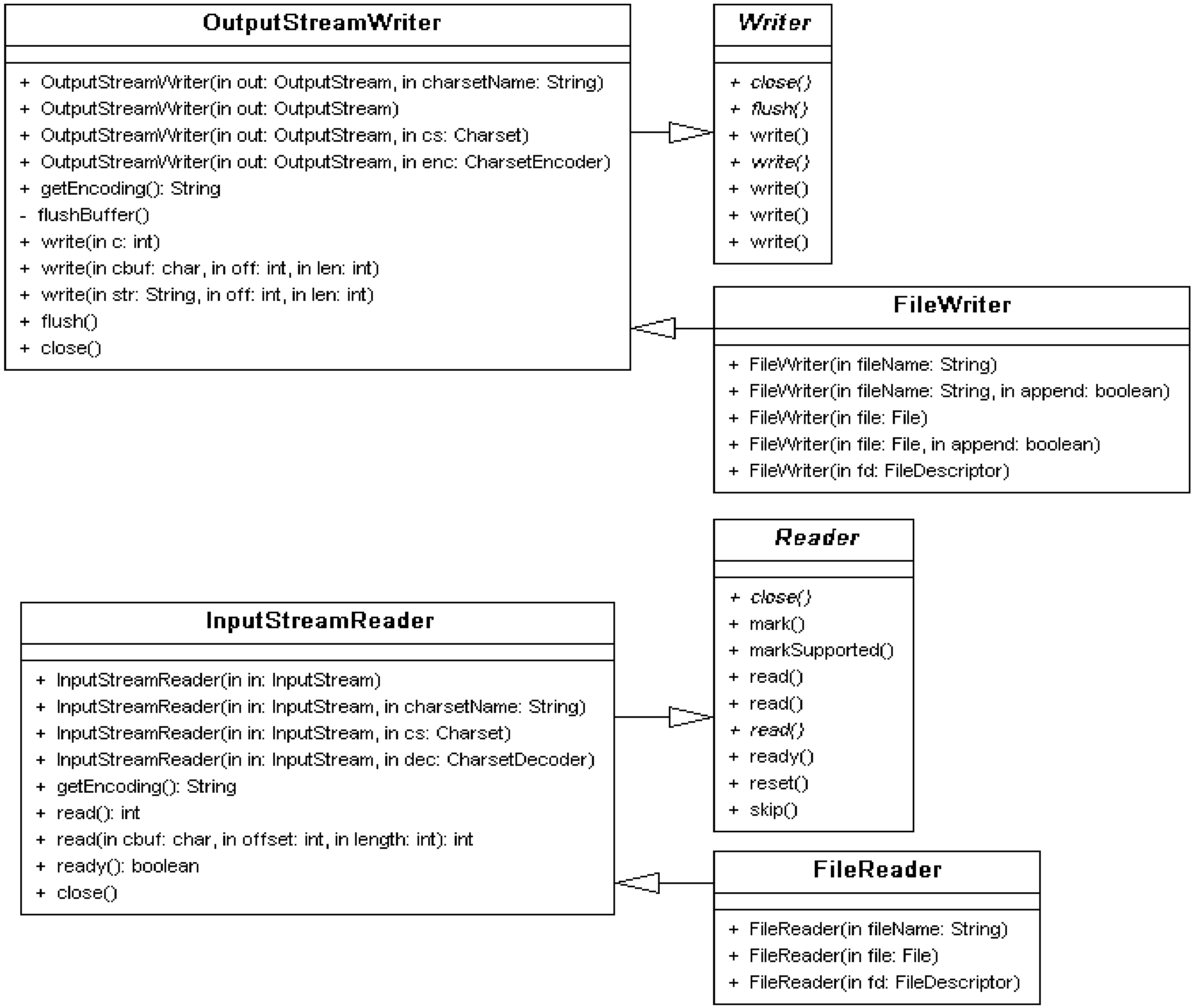
static File	**createTempFile**(String prefix, String suffix, File directory) Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
boolean	**delete**() Deletes the file or directory denoted by this abstract pathname.
void	**deleteOnExit**() Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.
boolean	**equals**(Object obj) Tests this abstract pathname for equality with the given object.
boolean	**exists**() Tests whether the file or directory denoted by this abstract pathname exists.
File	**getAbsoluteFile**() Returns the absolute form of this abstract pathname.
String	**getAbsolutePath**() Returns the absolute pathname string of this abstract pathname.
File	**getCanonicalFile**() Returns the canonical form of this abstract pathname.
String	**getCanonicalPath**() Returns the canonical pathname string of this abstract pathname.
String	**getName**() Returns the name of the file or directory denoted by this abstract pathname.
String	**getParent**() Returns the pathname string of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
File	**getParentFile**() Returns the abstract pathname of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
String	**getPath**() Converts this abstract pathname into a pathname string.
int	**hashCode**() Computes a hash code for this abstract pathname.
boolean	**isAbsolute**() Tests whether this abstract pathname is absolute.
boolean	**isDirectory**() Tests whether the file denoted by this abstract pathname is a directory.

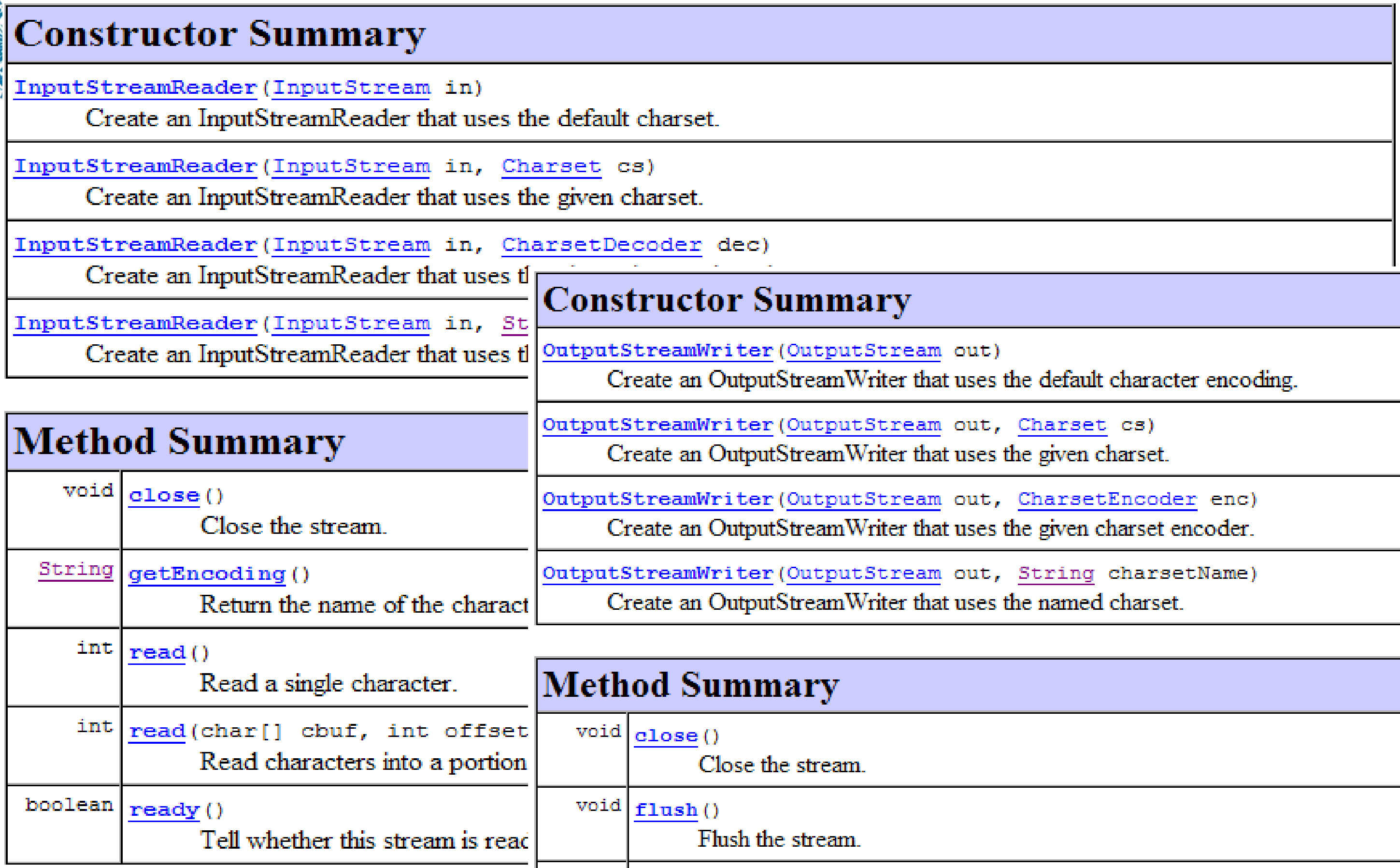


boolean	**isFile**() Tests whether the file denoted by this abstract pathname is a normal file.
boolean	**isHidden**() Tests whether the file named by this abstract pathname is a hidden file.
long	**lastModified**() Returns the time that the file denoted by this abstract pathname was last modified.
long	**length**() Returns the length of the file denoted by this abstract pathname.
String[]	**list**() Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
String[]	**list**(FilenameFilter filter) Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[]	**listFiles**() Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
File[]	**listFiles**(FileFilter filter) Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[]	**listFiles**(FilenameFilter filter) Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
static File[]	**listRoots**() List the available filesystem roots.
boolean	**mkdir**() Creates the directory named by this abstract pathname.
boolean	**mkdirs**() Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories.
boolean	**renameTo**(File dest) Renames the file denote
boolean	**setLastModified**(long ti Sets the last-modified ti



boolean	**setReadOnly**() Marks the file or directory named by this abstract pathname so that only read operations are allowed.
String	**toString**() Returns the pathname string of this abstract pathname.
URI	**toURI**() Constructs a `file:` URI that represents this abstract pathname.
URL	**toURL**() Converts this abstract pathname into a `file:` URL.


OutputStreamWriter
+ OutputStreamWriter(in out: OutputStream, in charsetName: String)
+ OutputStreamWriter(in out: OutputStream)
+ OutputStreamWriter(in out: OutputStream, in cs: Charset)
+ OutputStreamWriter(in out: OutputStream, in enc: CharsetEncoder)
+ getEncoding(): String
- flushBuffer()
+ write(in c: int)
+ write(in cbuf: char, in off: int, in len: int)
+ write(in str: String, in off: int, in len: int)
+ flush()
+ close()
Writer
+ close()
+ flush()
+ write()
+ write()
+ write()
+ write()
+ write()
FileWriter
+ FileWriter(in fileName: String)
+ FileWriter(in fileName: String, in append: boolean)
+ FileWriter(in file: File)
+ FileWriter(in file: File, in append: boolean)
+ FileWriter(in fd: FileDescriptor)
Reader
+ close()
+ mark()
+ markSupported()
+ read()
+ read()
+ read()
+ ready()
+ reset()
+ skip()
InputStreamReader
+ InputStreamReader(in in: InputStream)
+ InputStreamReader(in in: InputStream, in charsetName: String)
+ InputStreamReader(in in: InputStream, in cs: Charset)
+ InputStreamReader(in in: InputStream, in dec: CharsetDecoder)
+ getEncoding(): String
+ read(): int
+ read(in cbuf: char, in offset: int, in length: int): int
+ ready(): boolean
+ close()
FileReader
+ FileReader(in fileName: String)
+ FileReader(in file: File)
+ FileReader(in fd: FileDescriptor)

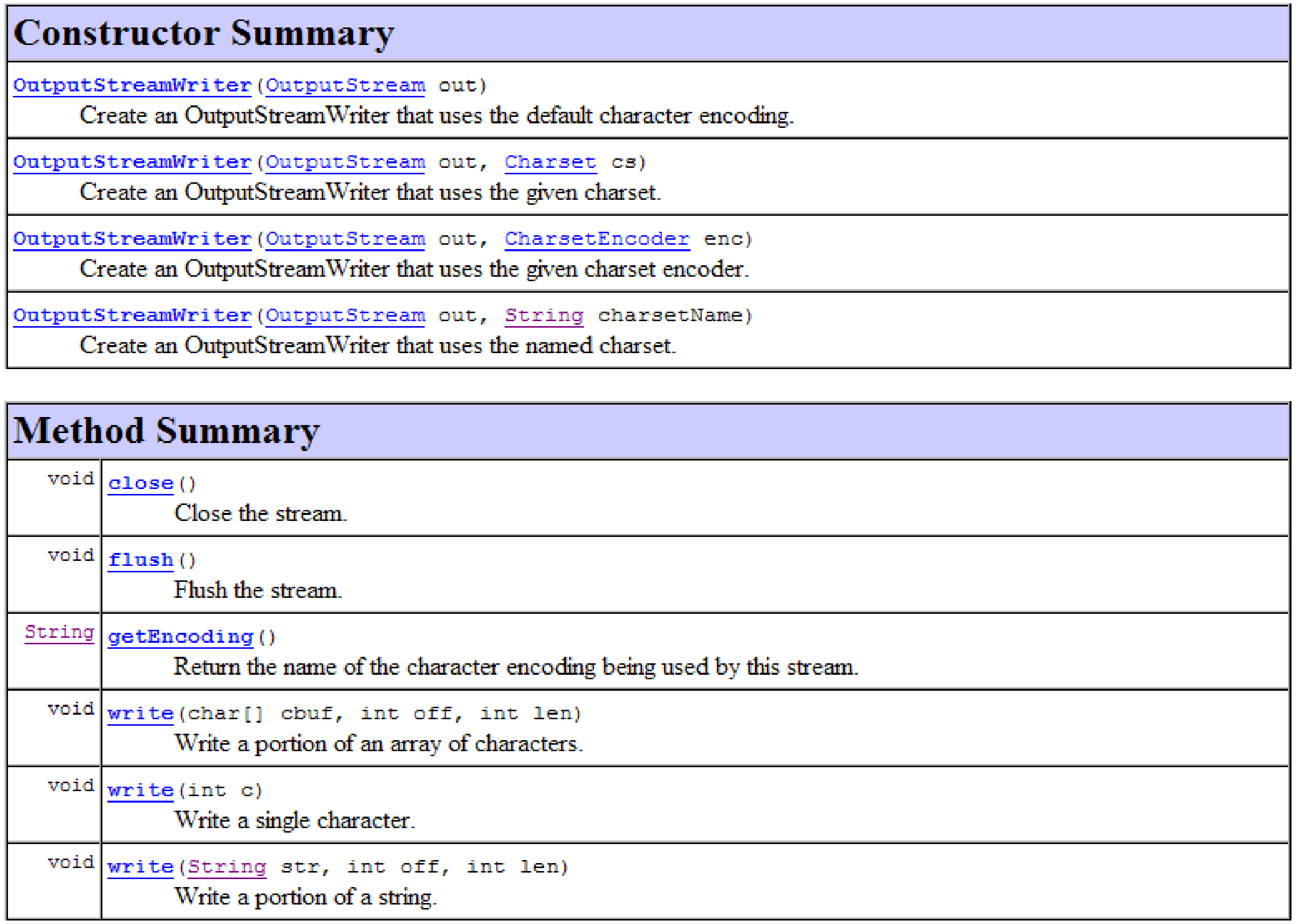


Constructor Summary

InputStreamReader(InputStream in) Create an InputStreamReader that uses the default charset.
InputStreamReader(InputStream in, Charset cs) Create an InputStreamReader that uses the given charset.
InputStreamReader(InputStream in, CharsetDecoder dec) Create an InputStreamReader that uses tl
InputStreamReader(InputStream in, St Create an InputStreamReader that uses tl

Method Summary

void	close() Close the stream.
String	getEncoding() Return the name of the charact
int	read() Read a single character.
int	read(char[] cbuf, int offset Read characters into a portion
boolean	ready() Tell whether this stream is reac

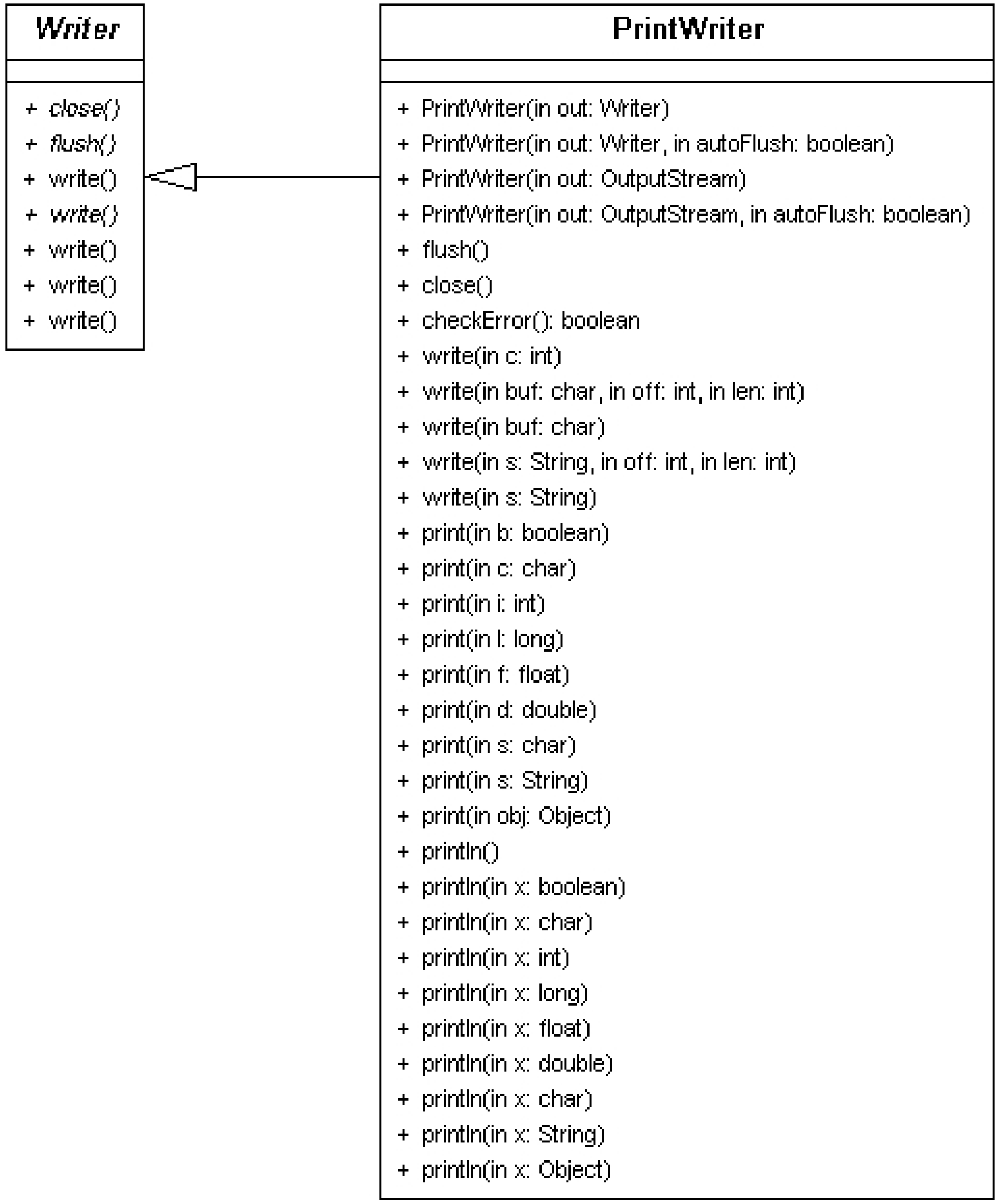


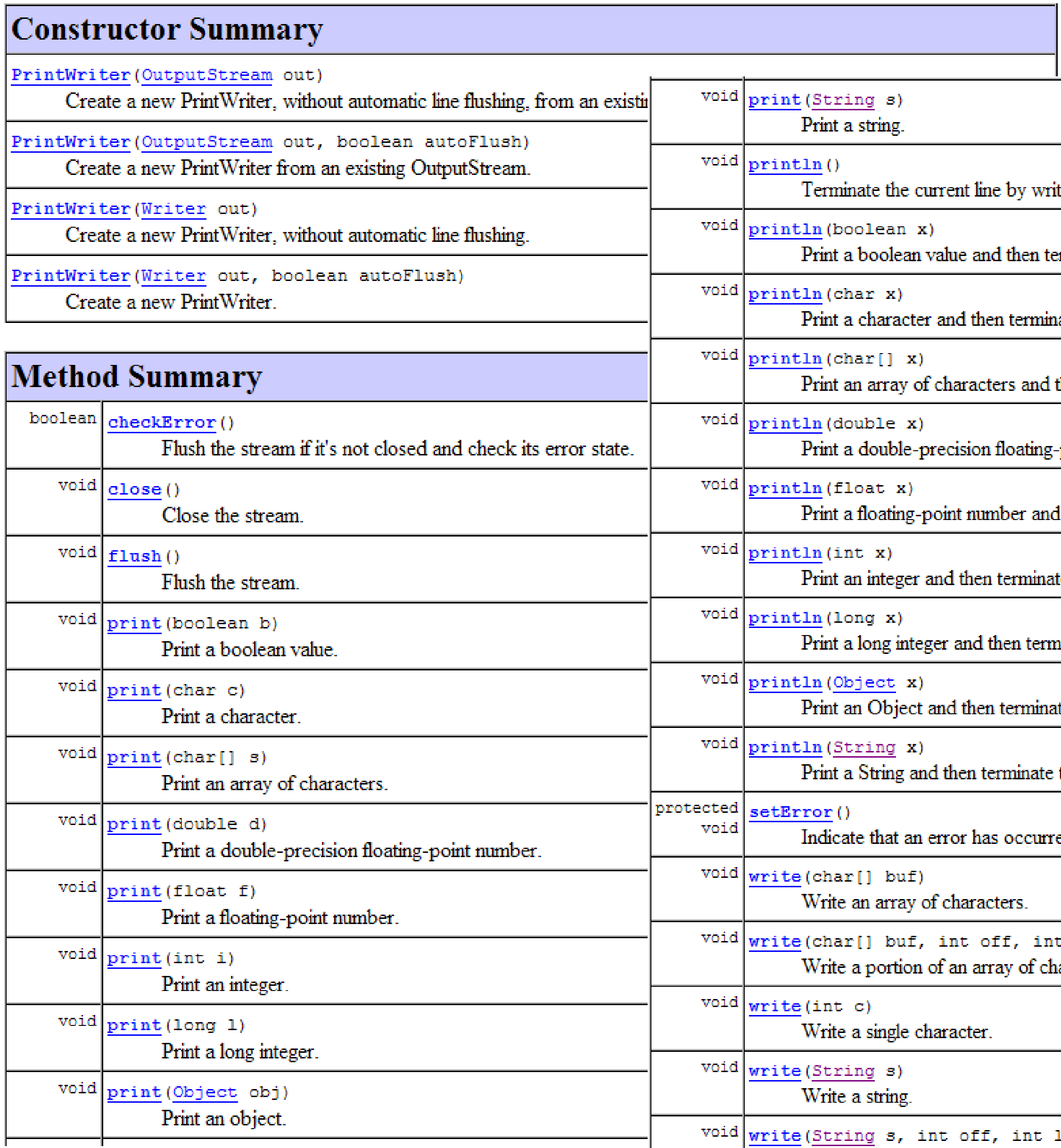
Constructor Summary

OutputStreamWriter(OutputStream out) Create an OutputStreamWriter that uses the default character encoding.
OutputStreamWriter(OutputStream out, Charset cs) Create an OutputStreamWriter that uses the given charset.
OutputStreamWriter(OutputStream out, CharsetEncoder enc) Create an OutputStreamWriter that uses the given charset encoder.
OutputStreamWriter(OutputStream out, String charsetName) Create an OutputStreamWriter that uses the named charset.

Method Summary

void	close() Close the stream.
void	flush() Flush the stream.
String	getEncoding() Return the name of the character encoding being used by this stream.
void	write(char[] cbuf, int off, int len) Write a portion of an array of characters.
void	write(int c) Write a single character.
void	write(String str, int off, int len) Write a portion of a string.


Writer
+ close()
+ flush()
+ write()
+ write()
+ write()
+ write()
+ write()
PrintWriter
+ PrintWriter(in out: Writer)
+ PrintWriter(in out: Writer, in autoFlush: boolean)
+ PrintWriter(in out: OutputStream)
+ PrintWriter(in out: OutputStream, in autoFlush: boolean)
+ flush()
+ close()
+ checkError(): boolean
+ write(in c: int)
+ write(in buf: char, in off: int, in len: int)
+ write(in buf: char)
+ write(in s: String, in off: int, in len: int)
+ write(in s: String)
+ print(in b: boolean)
+ print(in c: char)
+ print(in i: int)
+ print(in l: long)
+ print(in f: float)
+ print(in d: double)
+ print(in s: char)
+ print(in s: String)
+ print(in obj: Object)
+ println()
+ println(in x: boolean)
+ println(in x: char)
+ println(in x: int)
+ println(in x: long)
+ println(in x: float)
+ println(in x: double)
+ println(in x: char)
+ println(in x: String)
+ println(in x: Object)

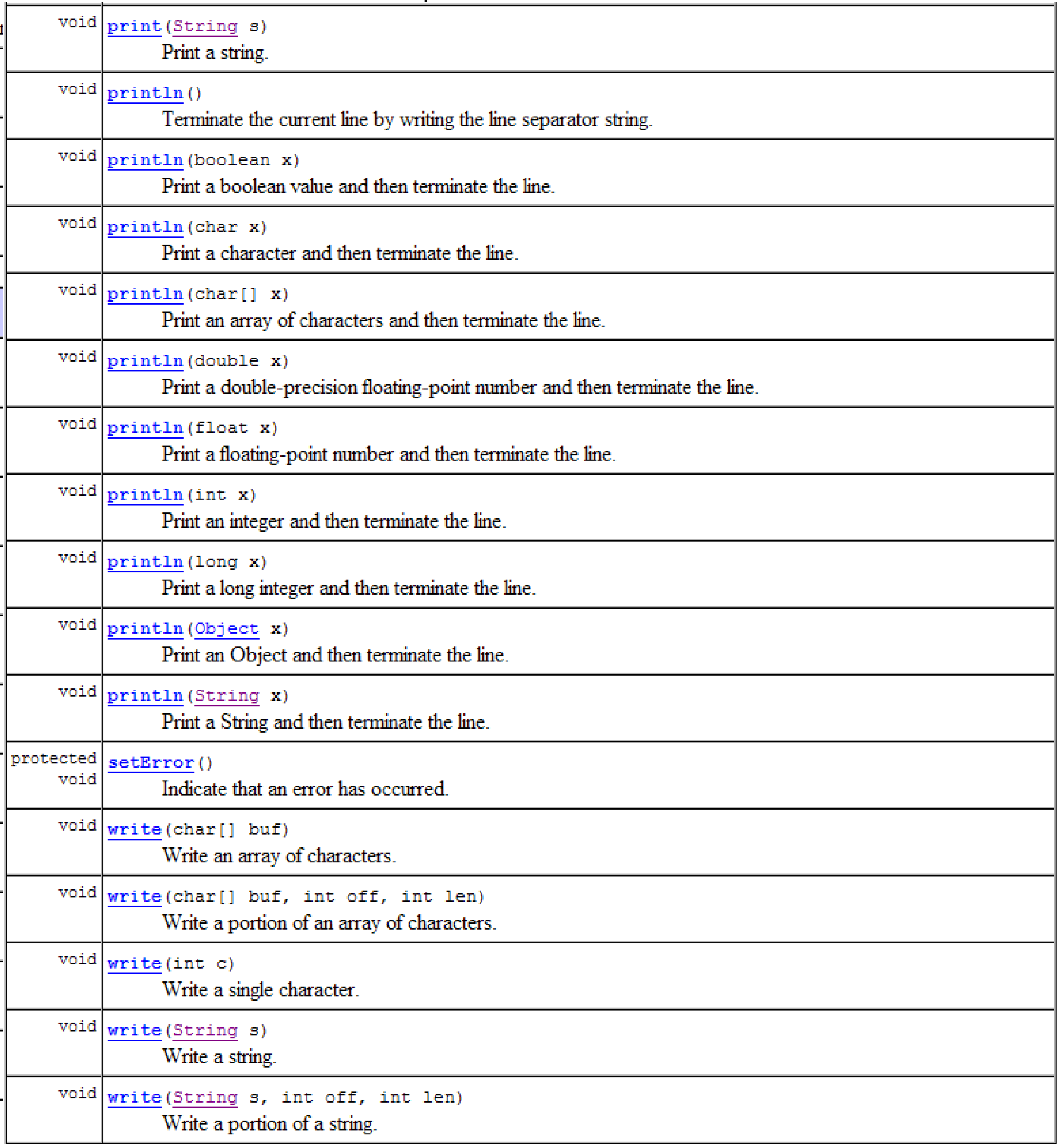


Constructor Summary

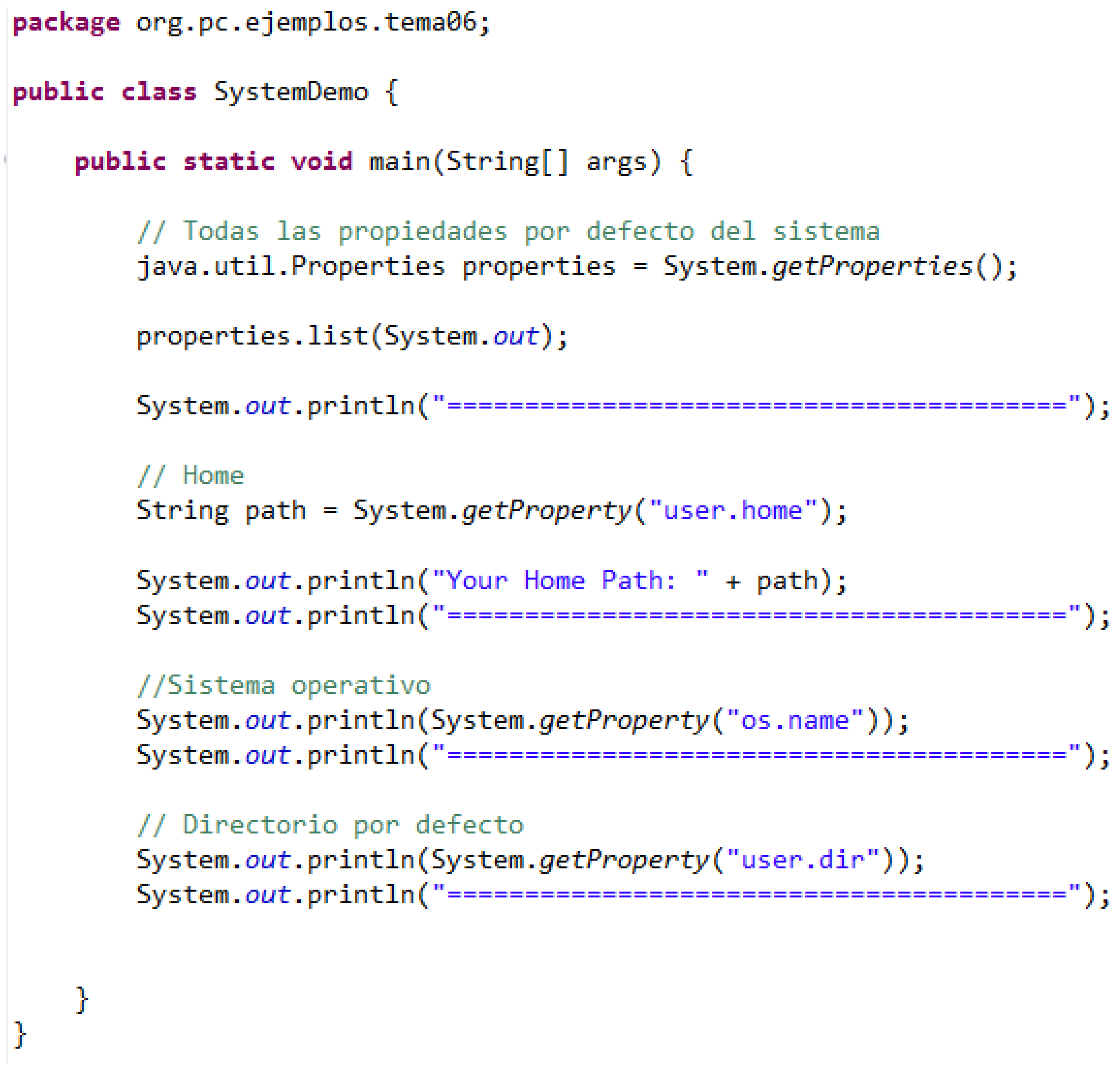
`PrintWriter(OutputStream out)` Create a new PrintWriter, without automatic line flushing, from an existi
`PrintWriter(OutputStream out, boolean autoFlush)` Create a new PrintWriter from an existing OutputStream.
`PrintWriter(Writer out)` Create a new PrintWriter, without automatic line flushing.
`PrintWriter(Writer out, boolean autoFlush)` Create a new PrintWriter.

Method Summary

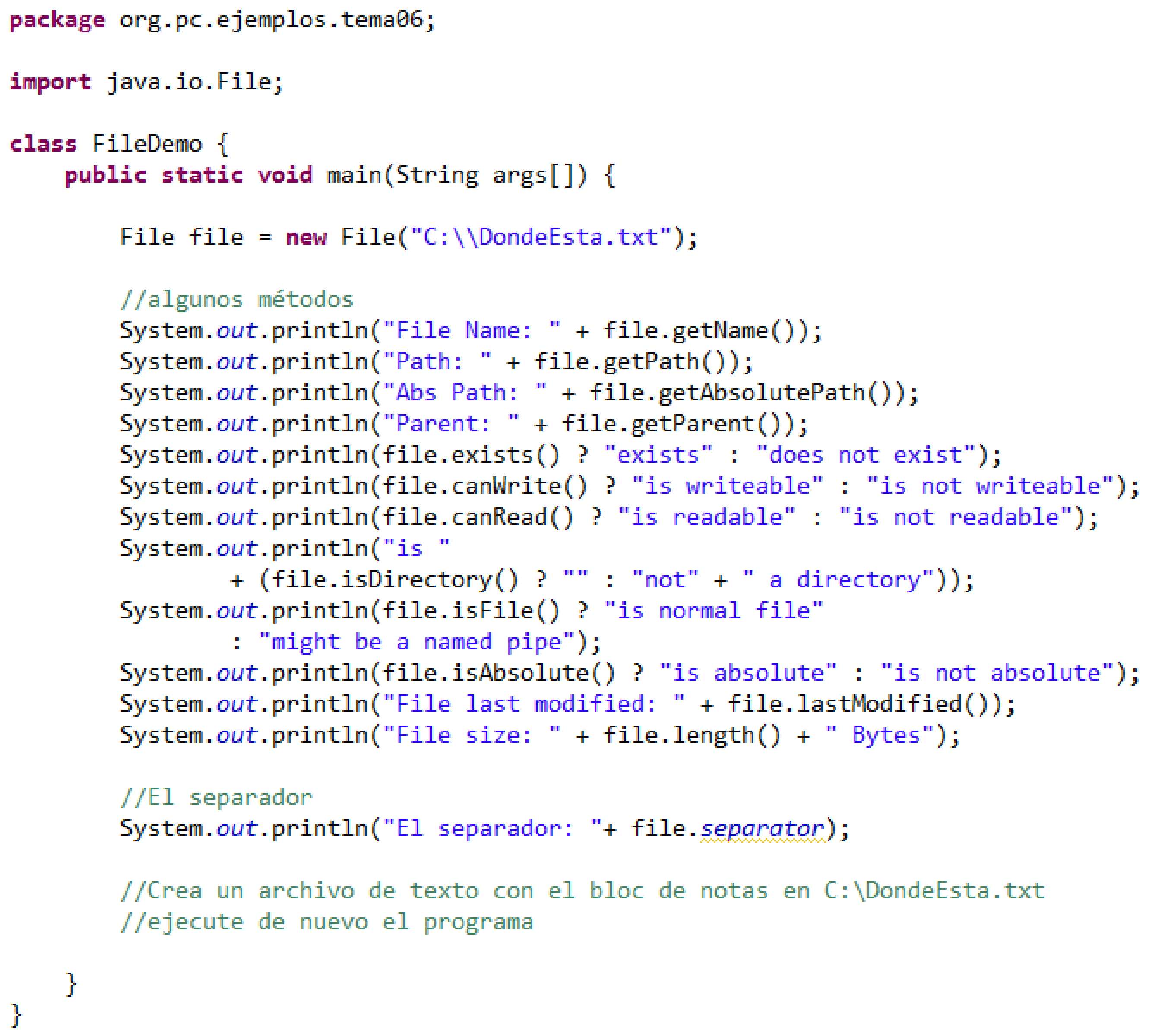
`boolean`	`checkError()` Flush the stream if it's not closed and check its error state.
`void`	`close()` Close the stream.
`void`	`flush()` Flush the stream.
`void`	`print(boolean b)` Print a boolean value.
`void`	`print(char c)` Print a character.
`void`	`print(char[] s)` Print an array of characters.
`void`	`print(double d)` Print a double-precision floating-point number.
`void`	`print(float f)` Print a floating-point number.
`void`	`print(int i)` Print an integer.
`void`	`print(long l)` Print a long integer.
`void`	`print(Object obj)` Print an object.



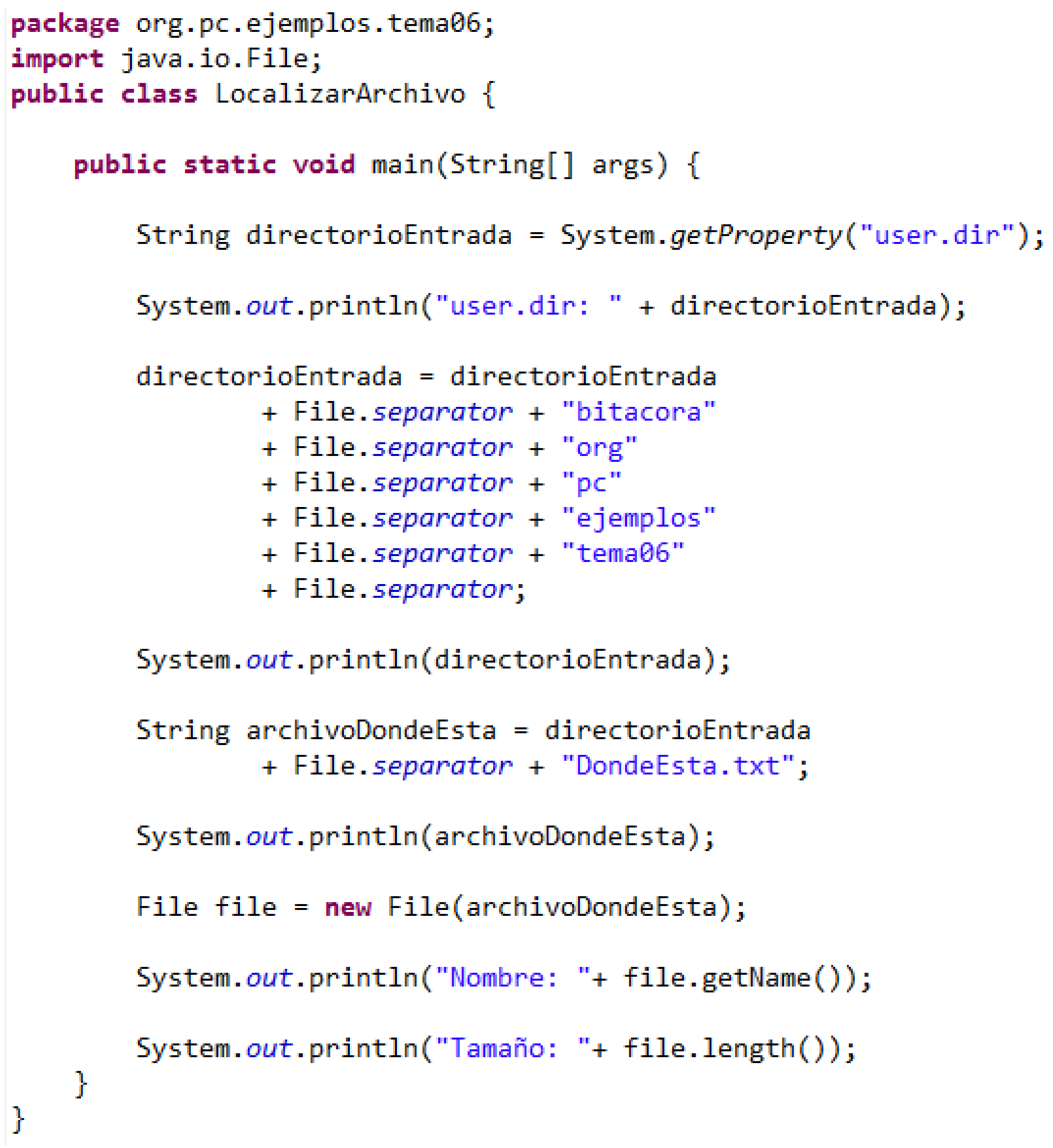
`void`	`print(String s)` Print a string.
`void`	`println()` Terminate the current line by writing the line separator string.
`void`	`println(boolean x)` Print a boolean value and then terminate the line.
`void`	`println(char x)` Print a character and then terminate the line.
`void`	`println(char[] x)` Print an array of characters and then terminate the line.
`void`	`println(double x)` Print a double-precision floating-point number and then terminate the line.
`void`	`println(float x)` Print a floating-point number and then terminate the line.
`void`	`println(int x)` Print an integer and then terminate the line.
`void`	`println(long x)` Print a long integer and then terminate the line.
`void`	`println(Object x)` Print an Object and then terminate the line.
`void`	`println(String x)` Print a String and then terminate the line.
`protected void`	`setError()` Indicate that an error has occurred.
`void`	`write(char[] buf)` Write an array of characters.
`void`	`write(char[] buf, int off, int len)` Write a portion of an array of characters.
`void`	`write(int c)` Write a single character.
`void`	`write(String s)` Write a string.
`void`	`write(String s, int off, int len)` Write a portion of a string.



```java
package org.pc.ejemplos.tema06;

public class SystemDemo {

    public static void main(String[] args) {

        // Todas las propiedades por defecto del sistema
        java.util.Properties properties = System.getProperties();

        properties.list(System.out);

        System.out.println("========================================");

        // Home
        String path = System.getProperty("user.home");

        System.out.println("Your Home Path: " + path);
        System.out.println("========================================");

        //Sistema operativo
        System.out.println(System.getProperty("os.name"));
        System.out.println("========================================");

        // Directorio por defecto
        System.out.println(System.getProperty("user.dir"));
        System.out.println("========================================");


    }
}
```



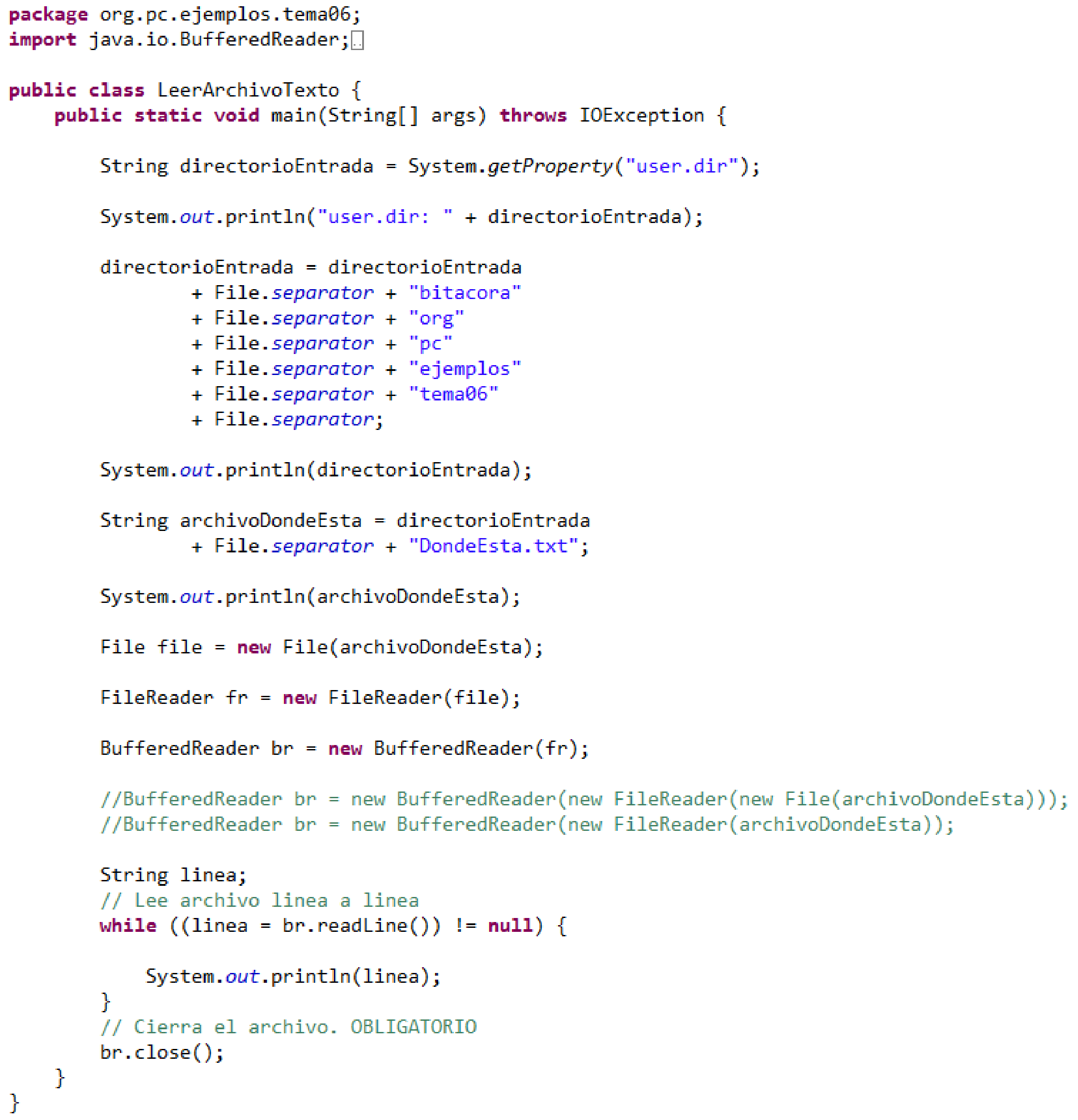
```
package org.pc.ejemplos.tema06;

import java.io.File;

class FileDemo {
    public static void main(String args[]) {

        File file = new File("C:\\DondeEsta.txt");

        //algunos métodos
        System.out.println("File Name: " + file.getName());
        System.out.println("Path: " + file.getPath());
        System.out.println("Abs Path: " + file.getAbsolutePath());
        System.out.println("Parent: " + file.getParent());
        System.out.println(file.exists() ? "exists" : "does not exist");
        System.out.println(file.canWrite() ? "is writeable" : "is not writeable");
        System.out.println(file.canRead() ? "is readable" : "is not readable");
        System.out.println("is "
                + (file.isDirectory() ? "" : "not" + " a directory"));
        System.out.println(file.isFile() ? "is normal file"
                : "might be a named pipe");
        System.out.println(file.isAbsolute() ? "is absolute" : "is not absolute");
        System.out.println("File last modified: " + file.lastModified());
        System.out.println("File size: " + file.length() + " Bytes");

        //El separador
        System.out.println("El separador: "+ file.separator);

        //Crea un archivo de texto con el bloc de notas en C:\DondeEsta.txt
        //ejecute de nuevo el programa

    }
}
```



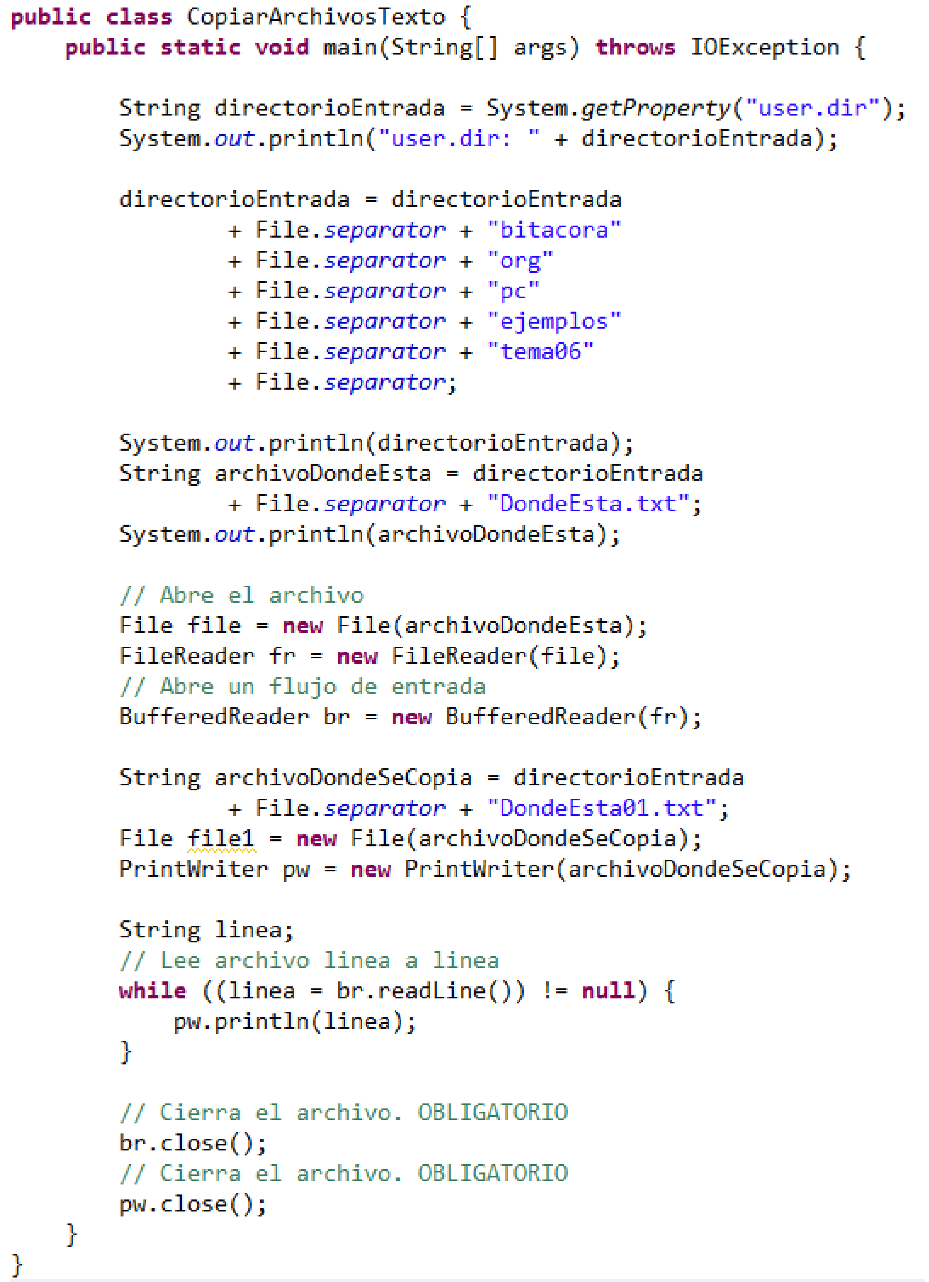
```java
package org.pc.ejemplos.tema06;
import java.io.File;
public class LocalizarArchivo {

    public static void main(String[] args) {

        String directorioEntrada = System.getProperty("user.dir");

        System.out.println("user.dir: " + directorioEntrada);

        directorioEntrada = directorioEntrada
                + File.separator + "bitacora"
                + File.separator + "org"
                + File.separator + "pc"
                + File.separator + "ejemplos"
                + File.separator + "tema06"
                + File.separator;

        System.out.println(directorioEntrada);

        String archivoDondeEsta = directorioEntrada
                + File.separator + "DondeEsta.txt";

        System.out.println(archivoDondeEsta);

        File file = new File(archivoDondeEsta);

        System.out.println("Nombre: "+ file.getName());

        System.out.println("Tamaño: "+ file.length());
    }
}
```

```java
package org.pc.ejemplos.tema06;

import java.io.File;

public class EscribirArchivoTexto {

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

        String directorioEntrada = System.getProperty("user.dir"

        System.out.println("user.dir: " + directorioEntrada);

        directorioEntrada = directorioEntrada
                + File.separator + "bitacora"
                + File.separator + "org"
                + File.separator + "pc"
                + File.separator + "ejemplos"
                + File.separator + "tema06"
                + File.separator;

        System.out.println(directorioEntrada);

        String archivoDondeEsta = directorioEntrada
                + File.separator + "DondeEsta.txt";

        System.out.println(archivoDondeEsta);

        File file = new File(archivoDondeEsta);

        //Borra lo que hubiera en el archivo existente
        FileWriter fw = new FileWriter(file);

        //Añade a lo que hubiera en el archivo existente
        //FileWriter fw = new FileWriter(file,true);

        // Escribe cadena al archivo
        for (int i = 0; i < 12; i++) {
          fw.write("Linea " + i + "\n");
        }

        // cerrar
        fw.close();
```

```java
/*
        //Lo mismo utilizando PrintWriter

        //Borra lo que hubiera en el archivo existente
        PrintWriter pw = new PrintWriter(file);

        //Tambien es valido
        //PrintWriter pw = new PrintWriter(archivoDondeEsta);


        // Añade a lo que hubiera en el archivo existente
        // es necesario un FileWriter
        //FileWriter fw1 = new FileWriter(file,true);
        //PrintWriter pw = new PrintWriter(fw1);

        //PrintWriter pw = new PrintWriter(file);

        // Escribe cadena al archivo
        for (int i = 0; i < 12; i++) {
            pw.println("Linea " + i );
        }

        // el mismo metodo para todos los tipos basicos
        int numero = 50;
        pw.println(numero);
        pw.println("Utilizando PrintWriter");

        // cerrar
        pw.close();

*/
    }
}
```



```java
package org.pc.ejemplos.tema06;
import java.io.BufferedReader;

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

        String directorioEntrada = System.getProperty("user.dir");

        System.out.println("user.dir: " + directorioEntrada);

        directorioEntrada = directorioEntrada
                + File.separator + "bitacora"
                + File.separator + "org"
                + File.separator + "pc"
                + File.separator + "ejemplos"
                + File.separator + "tema06"
                + File.separator;

        System.out.println(directorioEntrada);

        String archivoDondeEsta = directorioEntrada
                + File.separator + "DondeEsta.txt";

        System.out.println(archivoDondeEsta);

        File file = new File(archivoDondeEsta);

        FileReader fr = new FileReader(file);

        BufferedReader br = new BufferedReader(fr);

        //BufferedReader br = new BufferedReader(new FileReader(new File(archivoDondeEsta)));
        //BufferedReader br = new BufferedReader(new FileReader(archivoDondeEsta));

        String linea;
        // Lee archivo linea a linea
        while ((linea = br.readLine()) != null) {

            System.out.println(linea);
        }
        // Cierra el archivo. OBLIGATORIO
        br.close();
    }
}
```



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

        String directorioEntrada = System.getProperty("user.dir");
        System.out.println("user.dir: " + directorioEntrada);

        directorioEntrada = directorioEntrada
                + File.separator + "bitacora"
                + File.separator + "org"
                + File.separator + "pc"
                + File.separator + "ejemplos"
                + File.separator + "tema06"
                + File.separator;

        System.out.println(directorioEntrada);
        String archivoDondeEsta = directorioEntrada
                + File.separator + "DondeEsta.txt";
        System.out.println(archivoDondeEsta);

        // Abre el archivo
        File file = new File(archivoDondeEsta);
        FileReader fr = new FileReader(file);
        // Abre un flujo de entrada
        BufferedReader br = new BufferedReader(fr);

        String archivoDondeSeCopia = directorioEntrada
                + File.separator + "DondeEsta01.txt";
        File file1 = new File(archivoDondeSeCopia);
        PrintWriter pw = new PrintWriter(archivoDondeSeCopia);

        String linea;
        // Lee archivo linea a linea
        while ((linea = br.readLine()) != null) {
            pw.println(linea);
        }

        // Cierra el archivo. OBLIGATORIO
        br.close();
        // Cierra el archivo. OBLIGATORIO
        pw.close();
    }
}
```

Ejemplos de archivos de texto (javaalmanac)

www.exampledepot.com

e35. Reading Text from a File

```
try {
    BufferedReader in = new BufferedReader(new FileReader("infilename"));
    String str;
    while ((str = in.readLine()) != null) {
        process(str);
    }
    in.close();
} catch (IOException e) {
}
```

e37. Writing to a File

If the file does not already exist, it is automatically created.

```
try {
    BufferedWriter out = new BufferedWriter(new FileWriter("outfilename"));
    out.write("aString");
    out.close();
} catch (IOException e) {
}
```

e38. Appending to a File

```
try {
    BufferedWriter out = new BufferedWriter(new FileWriter("filename", true));
    out.write("aString");
    out.close();
} catch (IOException e) {
}
```

¡Muchas Gracias
Fin!