

Lenguaje Java

Entrada/Salida

Fernando Pérez Costoya
fperez@fi.upm.es

Índice

- Conceptos básicos
 - *Streams*
 - Ficheros de texto versus binarios
 - Codificación de caracteres
 - Requisitos del sistema de E/S
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

2

Fernando Pérez Costoya (2011)

Introducción

- Programa requiere datos de entrada y genera datos de salida
- Pueden corresponder a distintas fuentes y destinos de datos:
 - Teclado/ratón + pantalla
 - GUI
 - System.in, System.out, System.err
 - System.console (Java 6)
 - Ficheros
 - Red
 - Bases de datos

Entrada/salida en Java

3

Fernando Pérez Costoya (2011)

Streams

- E/S en Java: Gestión uniforme de fuentes y destinos de datos
- *Stream* → flujo (secuencia) de datos
 - *Stream* de entrada: de fuente a programa
 - *Stream* de salida: de programa a destino
- Modelo de cualquier fuente/destino
 - excepto GUI y bases de datos

Entrada/salida en Java

4

Fernando Pérez Costoya (2011)

Ficheros de texto versus binarios

- Ficheros binarios
 - No legibles directamente por usuario
 - Números almacenados en binario
 - Ejemplo: Programa.class
- Ficheros de texto
 - Legibles por usuarios
 - Visibles con cualquier editor
 - Números almacenados como secuencia de caracteres
 - Ocupan más espacio que ficheros de texto
 - Normalmente organizados en líneas
 - Ejemplo: Programa.java

Entrada/salida en Java

5

Fernando Pérez Costoya (2011)

Codificación de caracteres

- Juegos de caracteres (*Character Sets*)
 - Especifican el valor asignado a cada carácter
- Múltiples propuestas estándar
 - ASCII: 7 bits/carácter
 - Obsoleta: Soporte sólo para inglés (p.e. ç á)
 - Latin-1 (ISO 8859-1): 8 bits/carácter
 - Soporte sólo para lenguajes occidentales
 - [0-127] → ASCII = Latin-1
 - “Viejos tiempos”:
 - Leer/escribir 1 carácter implica leer/escribir 1 byte
 - Las cosas han cambiado bastante...

Entrada/salida en Java

6

Fernando Pérez Costoya (2011)

Unicode

- Más de un millón de caracteres
- Asigna valor a cada carácter pero no define codificación
 - UCS-4: Unicode con 4 bytes/carácter
 - UTF-8: Unicode con tamaño variable (de 1 a 4 bytes)
 - [0-127] → 1 byte
 - [128-2047] → 2 bytes
 - [2047-65535] → 3 bytes
 - [65536-....] → 4 bytes
 - UTF-16: Unicode con tamaño variable (2 y 4 bytes)
 - [0-65535] → 2 bytes
 - [65536-....] → 4 bytes
 - Internamente Java usa UTF-16 (char de Java ocupa 2 bytes)
- Leer/escribir 1 carácter ya no implica siempre leer/escribir 1 byte
 - Depende de juego de caracteres usado, pero **Java se encarga de ello**

Entrada/salida en Java

7

Fernando Pérez Costoya (2011)

Entrada/salida en Java

- Paquete java.io
 - Funcionalidad principal “clásica” de la E/S de Java
- Paquete java.nio
 - Introducido en Java 1.4
 - Soporte para entrada/salida de altas prestaciones
 - *Buffers*, canales y selectores
 - Ficheros proyectados en memoria
- Novedades en java.nio (NIO.2) en próxima versión de Java (Java 7)
 - Mejoras en operaciones sobre el sistema de ficheros

Entrada/salida en Java

8

Fernando Pérez Costoya (2011)

Requisitos de un sistema de entrada/salida

- Independiente del dispositivo fuente/destino
- Eficiente (p. ej. minimiza llamadas al sistema operativo)
- Gestión de múltiples formatos de datos
- Gestión transparente de filtros de datos (compresión, cifrado, ...)
- Buena capacidad de formateo de datos
- Soporte de operaciones del sistema de ficheros
- Independiente del sistema operativo

Entrada/salida en Java

9

Fernando Pérez Costoya (2011)

Escenarios típicos de manipulación de datos

- Manejo directo de bytes del fichero sin importar que representan
 - Ejemplo: programa que copia un fichero (sea binario o de texto)
- Manejo de fichero de texto carácter a carácter
 - Ejemplo: programa que cambia minúsculas a mayúsculas en fichero
- Manejo de fichero de texto línea a línea
 - Ejemplo: programa que copia N últimas líneas de fichero
- Manejo de fichero de texto con datos de distintos tipos
 - Ejemplo: programa que suma números presentes en fichero de texto
 - Ejemplo: programa que gestiona datos de empleados en fichero texto
- Manejo de fichero binario con datos de tipos primitivos
 - Ejemplo: programa que suma números presentes en fichero binario
- Manejo de fichero binario con datos de distintas clases
 - Ejemplo: programa que gestiona datos de empleados
- Manejo de fichero binario de forma no secuencial (*aleatoria*)
 - Ejemplo: fichero cuentas bancarias con cambios *in situ*

Entrada/salida en Java

10

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
 - Clases abstractas `InputStream`, `OutputStream`
 - Clases `FileInputStream`, `FileOutputStream`
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

11

Fernando Pérez Costoya (2011)

Clases `InputStream` y `OutputStream`

- Acceso directo a bytes de un *stream* de entrada o de salida
- Clases abstractas `java.io.InputStream` y `java.io.OutputStream`
 - Independientes del dispositivo fuente/destino
 - Base de toda la entrada/salida binaria
 - Clases derivadas para ficheros, filtros, ...
 - Programador puede crear nuevas clases derivadas
 - Uso de excepciones *checked* derivadas de `IOException`
 - Múltiples operaciones de lectura/escritura:
 - 1 byte, vector de bytes completo o parcial
 - Siempre se deben cerrar los *streams* cuando no se van a usar más

Entrada/salida en Java

12

Fernando Pérez Costoya (2011)

java.io.InputStream

```
public abstract int read() throws IOException // byte leído o -1 si EOF
public int read(byte[] data) throws IOException // n° bytes leídos o -1 si EOF
public int read(byte[] data, int offset, int length) throws IOException
public int available() throws IOException
public long skip(long bytesToSkip) throws IOException
public boolean markSupported()
public void mark(int readLimit)
public void reset() throws IOException
public void close() throws IOException
```

Entrada/salida en Java

13

Fernando Pérez Costoya (2011)

java.io.OutputStream

```
public abstract void write(int b) throws IOException
public void write(byte[] data) throws IOException
public void write(byte[] data, int offset, int length) throws IOException
public void flush() throws IOException
public void close() throws IOException
```

Entrada/salida en Java

14

Fernando Pérez Costoya (2011)

Clases FileInputStream y FileOutputStream

- Derivadas de InputStream y OutputStream para manejo de ficheros
- Diversos constructores: uno de ellos con nombre de fichero
 - public **FileInputStream**(String fileName) throws IOException
 - Abre el fichero; si no existe, excepción
 - public **FileOutputStream**(String filename) throws IOException
 - Crea el fichero si no existe; si existe lo trunca
 - public **FileOutputStream**(String name, boolean append) throws IOException
 - Crea el fichero si no existe; si existe lo trunca o se pone al final

Entrada/salida en Java

15

Fernando Pérez Costoya (2011)

Streams de red

- Aunque presentación se centra en ficheros y no lo trata
 - *Streams* pueden usarse también para transferir datos por la red
- **Socket**: punto de comunicación en Internet (paquete java.net)
 - public **Socket**(String host, int port) throws UnknownHostException, IOException
 - public InputStream **getInputStream**() throws IOException
 - public OutputStream **getOutputStream**() throws IOException
 - public **ServerSocket**(int port) throws IOException
- **URL**: Identificador de recurso web (paquete java.net)
 - public **URL**(String url) throws MalformedURLException
 - public InputStream **openStream**() throws IOException

Entrada/salida en Java

16

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
 - FilterInputStream y FilterOutputStream
 - BufferedInputStream y BufferedOutputStream
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

17

Fernando Pérez Costoya (2011)

Filtro

- *Stream* interpuesto delante de otro para filtrado de datos
- Filtro de entrada:
 - Clase derivada de FilterInputStream (subclase de InputStream)
 - Recibe en su constructor otro InputStream
 - Lectura de filtro lee de ese otro *stream* y puede filtrar datos leídos
- Filtro de salida:
 - Clase derivada de FilterOutputStream (subclase de OutputStream)
 - Recibe en su constructor otro OutputStream
 - Escritura en filtro escribe en ese *stream*, pero antes puede filtrar datos
- Filtros de pueden encadenar:


```
InputStream ent = new ComprimidoInputStream(new CifradoInputStream(
    new FileInputStream(args[0]));
OutputStream sal = new ComprimidoOutputStream(new CifradoOutputStream(
    new FileOutputStream(args[1]));
```

Entrada/salida en Java

18

Fernando Pérez Costoya (2011)

Clases FilterInputStream y FilterOutputStream

- No instanciables; sólo para crear subclases
- En su constructor reciben el *stream* al que se interponen
 - Lo guardan en campo interno in o out, respectivamente
- Mismos métodos que InputStream y OutputStream, respectivamente
 - Sus métodos actúan sólo de pasarela:


```
public int read() throws IOException {return in.read();}
public void write(int b) throws IOException {out.write(b);}
```
- Filtro de entrada: Subclase de FilterInputStream
 - Sobrescribe algunos de sus métodos (al menos los de lectura)
- Filtro de salida: Subclase de FilterOutputStream
 - Sobrescribe algunos de sus métodos (al menos los de escritura)
- Constructor de subclase debe invocar el de la superclase (super)

Entrada/salida en Java

19

Fernando Pérez Costoya (2011)

BufferedInputStream y BufferedOutputStream

- Lecturas y escrituras byte a byte ineficientes
 - Cada operación activa el sistema operativo
- Lecturas y escrituras multi-bytes: ¿qué tamaño usar?
 - Muy grande puede ser contraproducente
 - Óptimo depende del dispositivo
- ¿Operaciones byte a byte eficientes?: Clases filtro con *buffers*
 - BufferedInputStream: lectura lee del *buffer* interno
 - Si vacío, rellena *buffer* leyendo de *stream* subyacente
 - BufferedOutputStream: escritura escribe en el *buffer* interno
 - Si lleno o flush, escribe *buffer* en *stream* subyacente
- Constructores pueden recibir argumento con tamaño del *buffer*

Entrada/salida en Java

20

Fernando Pérez Costoya (2011)

Recapitulando sobre la E/S directa

- Clases InputStream/OutputStream son el sustrato de toda la E/S
 - Se usan indirectamente en toda operación de E/S
- Pero sólo se usan directamente en aplicaciones independientes del:
 - Formato (texto versus binario) de los ficheros utilizados
 - Contenido (enteros, flotantes, *strings*, ...) de los ficheros utilizados
- Si importa rendimiento, BufferedInputStream/BufferedOutputStream

Entrada/salida en Java

21

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
 - Reader y Writer
 - InputStreamReader y OutputStreamReader
 - Entrada de texto
 - Scanner
 - Salida de texto
 - Entrada/salida de consola
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

22

Fernando Pérez Costoya (2011)

E/S de texto

- Ficheros de texto versus binarios:
 - Menor eficiencia en espacio y tiempo pero mayor flexibilidad
- La entrada/salida de texto debe manejar distintas:
 - Codificaciones (juegos) de caracteres
 - El sistema está configurado con una por defecto
System.getProperty("file.encoding")
 - Pero hay que poder procesar datos con otras codificaciones
 - Configuraciones locales (*locales*)
 - Aspectos como carácter separador de parte entera y decimal
 - El sistema está configurado con una por defecto
Locale.getDefault()
 - Pero hay que poder manejar datos con otras configuraciones
- E/S de texto en Java, crítica personal:
 - Algún grado de redundancia: muchas formas de hacer una cosa
 - Cierta asimetría en soporte de entrada versus salida

Entrada/salida en Java

23

Fernando Pérez Costoya (2011)

Clases Reader y Writer

- Clases abstractas soporte de la E/S de texto
 - Como InputStream/OutputStream pero con caracteres en vez de bytes
 - Algunos métodos de Reader y Writer


```
public int read() throws IOException
public int read(char[] text) throws IOException
public abstract int read(char[] text, int offset, int length) throws IOException
public void write(int c) throws IOException
public void write(char[] text) throws IOException
public abstract void write(char[] text, int offset, int length) throws IOException
```
 - Dispone también de filtros con *buffering*:
 - BufferedReader y BufferedWriter
 - BufferedReader dispone de un método readLine
 - Subclases más usadas:
 - InputStreamReader y OutputStreamWriter

Entrada/salida en Java

24

Fernando Pérez Costoya (2011)

InputStreamReader y OutputStreamWriter

- Establecen puente entre E/S binaria y de texto
 - Conectan un Reader/Writer a un InputStream/OutputStream
- Constructores (si no *codificación*, usa la por defecto):


```
public InputStreamReader(InputStream in)
public InputStreamReader(InputStream in, String codificación)
public OutputStreamWriter(OutputStream out)
public OutputStreamWriter(OutputStream out, String codificación)
```
- Clases de conveniencia derivadas: FileReader y FileWriter


```
FileReader("f") ↔ new InputStreamReader(new FileInputStream("f"))
FileWriter("f") ↔ new OutputStreamWriter(new FileOutputStream("f"))
```
- Posibles encadenamientos complejos:


```
Reader r = new BufferedReader(new InputStreamReader(
    new BufferedInputStream(new FileInputStream("f")), "UTF-8"));
```

Entrada/salida en Java

25

Fernando Pérez Costoya (2011)

Entrada de texto

- Aplicación requiere entrada carácter a carácter
 - P. ej. Copiar fichero cambiando minúsculas a mayúsculas
 - Técnica habitual: Reader
 - Sería el único caso donde es razonable usarlo directamente
 - Exceptuando el **BufferedReader** del punto siguiente
 - Aunque se usa indirectamente en toda entrada de texto
- Aplicación requiere entrada línea a línea
 - P. ej. Contar nº líneas de un fichero
 - Técnicas habituales: **BufferedReader** (**readline**) o **Scanner**
- Aplicación requiere entrada con datos de distintos tipos
 - P. ej. Programa que suma números presentes en fichero de texto
 - Técnica habitual: **Scanner**

Entrada/salida en Java

26

Fernando Pérez Costoya (2011)

Clase java.util.Scanner

- Extrae tipos primitivos y *strings* de un conjunto de datos
- Constructor con distintas fuentes de datos (opcional codificación)
 - String
 - Reader
 - InputStream (genera internamente un Reader)
 - File (genera internamente un Reader)
- Entrada: elementos separados por delimitadores
 - Por defecto, delimitador: caracteres considerados espacios en blanco
 - Puede cambiarse (también el *locale*)
- Si fuente genera IOException, Scanner la captura.

Entrada/salida en Java

27

Fernando Pérez Costoya (2011)

Métodos principales de java.util.Scanner

- Preguntar por el próximo elemento:
 - hasNext: ¿Hay un elemento más en la fuente de datos?
 - Admite expresión regular
 - hasNextInt, hasNextDouble, ...: ¿Próximo elemento es int/double, ...?
 - hasNextLine: ¿Hay una línea más en la fuente de datos?
- Extraer el próximo elemento:
 - next: lee el próximo elemento y lo devuelve como un String
 - nextInt, nextDouble, ...: lee próximo elemento y convierte en int/double...
 - Si no lo es, InputMismatchException (*unchecked*)
 - nextLine: ¿Hay una línea más en la fuente de datos?
- Close

Entrada/salida en Java

28

Fernando Pérez Costoya (2011)

Clase java.io.PrintWriter

- Derivada de Writer
- Constructores: con fichero, con OutputStream, con File y con Writer
 - Opcional codificación
- Además de métodos write de Writer, ofrece métodos print y println
 - Para tipos primitivos, String y Object (imprime valor de toString)


```
public void print(boolean b)
public void print(char c)
.....
public void print(Object o)
```
- Captura IOException: uso de método checkError
- Para imprimir especificando formato: format (o printf)


```
public PrintWriter format(String format, Object ... args)
public PrintWriter format(Locale l, String format, Object ... args)
sal.format("Resultado: %d %f %s %s %n", entero, flotante, cadena, objeto);
```

Entrada/salida en Java

29

Fernando Pérez Costoya (2011)

java.io.PrintStream y java.util.Formatter

- PrintStream: muy similar a PrintWriter pero no es un Writer
 - Es un FilterOutputStream con un OutputStreamWriter interno
 - Sólo se mantiene porque System.out y System.err son de este tipo
- Formatter: complementario de Scanner
 - No es un Writer ni un OutputStream
 - Constructores: con fichero, con OutputStream, con File y con Writer
 - Codificación y locale opcionales
 - No ofrece print, sólo format
 - Un poco lio:
 - Si utilizas directamente PrintWriter/PrintStream sobre un fichero
 - Internamente, crean un Formatter para su método format (printf)
 - Si utilizas directamente Formatter sobre un fichero
 - Internamente crean un Writer para escribir

Entrada/salida en Java

30

Fernando Pérez Costoya (2011)

Especificación del formato (1/2)

- %[argument_index\$][flags][width][.precision]conversion
- Conversion:
 - Entero: d, o, x, X
 - Flotante: f, e, g, G, a, A
 - Carácter: c, C; Booleano: b, B;
 - Generales: h, H (NULL/null o hash code), s, S (toString)
 - Fecha/hora: aplicables a Date, Calendar y long (ms. desde 1/1/1970)
 - argument_index: n° argumento asociado a esta especificación
 - Por defecto, en orden; se numeran desde 1; si < se refiere al previo
 - width: mínimo n° de caracteres que ocupa el dato en la salida
 - Si dato más pequeño, se rellena
 - precision: para flotantes, n° dígitos decimales
 - Para no numéricos, máximo n° caracteres que ocupa dato en salida

Entrada/salida en Java

31

Fernando Pérez Costoya (2011)

Especificación del formato (2/2)

- flags: diversas opciones no aplicables a todas las conversiones
 - - justificado izquierda
 - 0 rellenar con 0
 - + positivos con signo +
 - ' ' usarlo en vez de +
 - (negativos entre paréntesis
 - , usar separadores para miles según locale
 - # presentación alternativa:
 - Para conversión o, comenzar número con 0
 - Para conversión x|X, comenzar número con 0x|0X
 - Para flotantes, separador decimal siempre presente

Entrada/salida en Java

32

Fernando Pérez Costoya (2011)

E/S de consola

- System.in es un InputStream → lee bytes
 - Uso de Scanner(System.in)
 - Si se requiere acceder a todos los caracteres:
 - InputStreamReader(System.in)
- System.out y System.err son PrintStream
- En Java 6, clase System.Console

Entrada/salida en Java

33

Fernando Pérez Costoya (2011)

Recapitulación sobre la E/S de texto

- Uso de Scanner para entrada de texto
 - También para System.in:
- Y de PrintWriter o Formatter para la salida de texto
 - Uso directo de System.out y System.err ya que son PrintStream
- Excepto si se requiere acceder a todos los caracteres:
 - Incluidos los delimitadores
 - InputStreamReader y OutputStreamWriter y
 - O FileReader y FileWriter si vale codificación por defecto

Entrada/salida en Java

34

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
 - Clases DataInputStream y DataOutputStream
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

35

Fernando Pérez Costoya (2011)

DataInputStream y DataOutputStream

- Lectura/Escritura en binario de tipos primitivos y String
- Subclases de FilterInputStream y FilterOutputStream
 - InputStream e = new DataInputStream(new FileInputStream("F"));
 - OutputStream s = new DataOutputStream(new FileOutputStream("F"));
- Además de read/write bytes, métodos para tipos primitivos:
 - public boolean readBoolean() throws IOException
 - public double readDouble() throws IOException
 - public void writeBoolean(boolean v) throws IOException
 - public void writeDouble(double d) throws IOException
- Y String (escribe/lee longitud en 2 bytes y caracteres en UTF-8):
 - public String readUTF() throws IOException
 - public void writeUTF(String s) throws IOException.
- Para detectar EOF debe capturarse EOFException

Entrada/salida en Java

36

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
 - *Serialización* de objetos
 - Clases `ObjectOutputStream` y `ObjectInputStream`
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

37

Fernando Pérez Costoya (2011)

Serialización de objetos

- Data Streams: leer/escribir objetos de tipos primitivos y `String`
 - ¿Puedo hacerlo con objetos de cualquier clase?
 - **Object Streams**: función lectura/escritura recupera/guarda objeto
 - *Serialización*: “Aplanar” todos los campos del objeto en un *stream*
 - Técnica requerida por primera vez en RMI
 - No trivial: guardar datos no estáticos del objeto pero también
 - De la superclase y de objetos que contiene y así sucesivamente
 - Object Streams vs. Data Streams
 - Más cómodos: 1 sola llamada guarda/recupera el objeto
 - Menos eficientes y seguros: no controla que se guarda
 - Más frágiles: cambio en clase → objeto irrecuperable
 - En versión *serializada* se guarda información para detectarlo
 - No todo objeto es *serializable* por su propia esencia
 - Por ejemplo, un *stream* de entrada/salida

Entrada/salida en Java

38

Fernando Pérez Costoya (2011)

ObjectInputStream y ObjectOutputStream

- Subclases de `InputStream` y `OutputStream`
`InputStream e = new ObjectInputStream(new FileInputStream("F"));`
`OutputStream s = new ObjectOutputStream(new FileOutputStream("F"));`
- Métodos para salvar/recuperar objeto:
`public final void writeObject(Object o) throws IOException`
`public final Object readObject() throws OptionalDataException,`
`ClassNotFoundException, IOException`
- Clase debe implementar la interfaz `Serializable` (sin métodos)
- Si algún campo no serializable, se puede marcar como `transient`
- Clase puede especificar su propio procedimiento de *serialización*

Entrada/salida en Java

39

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
 - Clase `File`
 - Información de fichero
 - Atributos
 - Operaciones sobre ficheros
 - Directorios
- Ficheros de acceso *aleatorio*

Entrada/salida en Java

40

Fernando Pérez Costoya (2011)

Problemática manejo uniforme nom. fichero

- Difícil por peculiaridades de cada sistema operativo
 - \ vs. /
 - Límite de longitud
 - Juego de caracteres usado
 - Distinción entre mayúsculas y minúsculas
 - Uso de extensiones
 - Atributos del fichero diversos
- En cuanto al uso de rutas, no deberían incluirse en el programa
 - Args
 - System.in
 - JFileChooser

Entrada/salida en Java

41

Fernando Pérez Costoya (2011)

Clase File

- Cobertura de nombre (ruta) de fichero (o directorio)
 - Aunque no exista
- Constructores:
 - File (String f)
 - File (String d, String f)
 - File (File d, String f)
- Permite evitar rutas explícitas dependientes del SO:
 - d1/d2/f o d1\d2f
 - new(new (new File ("d1", "d2"), "f")
- A continuación revisamos sus métodos principales

Entrada/salida en Java

42

Fernando Pérez Costoya (2011)

Información del fichero

- Si existe y su tipo
 - boolean exists()
 - boolean isFile()
 - boolean isDirectory()
- Operaciones sobre la ruta
 - String getName() → sólo nombre; no ruta
 - String getParent() → ruta sin nombre
 - String getPath() → tal como se especificó en constructor
 - boolean isAbsolute()
 - String getAbsolutePath() → Si absoluto en constructor, lo retorna
 - Sino, propiedad del sistema user.dir + ruta en constructor
 - String getCanonicalPath() → Ruta absoluta normalizada
 - Traduce alias, enlaces simbólicos, ...
 - Compacta: ../dir1/./dir2/f → ../dir2/f

Entrada/salida en Java

43

Fernando Pérez Costoya (2011)

Atributos del fichero

- Permisos
 - boolean canRead()
 - boolean canWrite()
 - boolean canExecute()
 - boolean setReadable(boolean)
 - boolean setWritable(boolean)
 - boolean setExecutable(boolean)
- Algunos más...
 - boolean isHidden()
 - long lastModified()
 - boolean setLastModified(long)
 - long length()

Entrada/salida en Java

44

Fernando Pérez Costoya (2011)

Operaciones sobre ficheros

- boolean `createNewFile()`
- boolean `renameTo(File destino)`
- boolean `delete()`
- void `deleteOnExit()`
- static File `createTempFile(String prefijo, String sufijo, File dir)`

Entrada/salida en Java

45

Fernando Pérez Costoya (2011)

Directorios

- Clase File también para directorios
 - Pero algunos métodos comportamiento diferente
 - delete sólo de directorios vacíos
- Métodos específicos para directorios
 - boolean `mkdir()`
 - boolean `makedirs()` → crea subdirectorios necesarios
 - String [] `list()` → retorna todas las entradas del directorio
 - File [] `listFiles()` → igual pero retorna objetos File
 - String [] `list(FilenameFilter filtro)` → sólo retorna las que cumplen filtro
 - File [] `listFiles(FilenameFilter filtro)` → igual pero File
 - String [] `list(FileFilter filtro)` → sólo retorna las que cumplen filtro
 - File [] `listFiles(FileFilter filtro)` → igual pero File

Entrada/salida en Java

46

Fernando Pérez Costoya (2011)

Filtros

- Filtro `FilenameFilter`: clase que implementa interfaz `FilenameFilter`
 - Debe implementar método `accept` de la interfaz:
 - boolean `accept()` (File dir, String nombre)
- Filtro `FileFilter`: clase que implementa interfaz `FileFilter`
 - Debe implementar método `accept` de la interfaz:
 - boolean `accept(File pathname)`

Entrada/salida en Java

47

Fernando Pérez Costoya (2011)

Índice

- Conceptos básicos
- Entrada/salida directa
- Filtros
- Entrada/salida de texto
- Entrada/salida binaria de tipos primitivos
- Entrada/salida binaria de clases
- Manejo de ficheros y directorios
- Ficheros de acceso *aleatorio*
 - Clase `RandomAccessFile`

Entrada/salida en Java

48

Fernando Pérez Costoya (2011)

Ficheros de acceso *aleatorio*

- Clases de entrada/salida estudiadas presentan limitaciones:
 - Acceso secuencial
 - Apertura de fichero o para lectura (R) o para escritura (W)
- Clase `RandomAccessFile` (subclase directa de `Object`) permite:
 - Apertura RW para ficheros binarios
 - Acceso *aleatorio*: lectura/escritura a partir de cualquier byte
- Constructores:
 - `public RandomAccessFile(String f, String modo) throws FileNotFoundException`
 - `public RandomAccessFile(File file, String modo) throws FileNotFoundException`
- Modo:
 - "r": abierto sólo para lectura.
 - "rw": abierto para lectura/escritura, si no existe se crea.
 - "rws": igual pero modificaciones se escriben a disco inmediatamente.
 - "rwd": igual pero sólo se escriben inmediatamente datos no metadatos.
 - "rws"/"rwd" vs. "rw": más fiabilidad pero menos eficiencia.

Entrada/salida en Java

49

Fernando Pérez Costoya (2011)

Clase `RandomAccessFile`

- Proporciona métodos de `DataInputStream` y `DataOutputStream`:
 - `read/write bytes` y `readBoolean/readDouble/writeBoolean/writeDouble,...`
 - Para detectar EOF debe capturarse `EOFException`
- Implementa concepto de puntero a fichero (*file pointer*):
 - Posición del fichero donde comenzará la próxima lectura/escritura
 - En apertura 0
 - Cada lectura/escritura avanza el puntero conforme a lo leído/escrito
 - Métodos para conocer dónde está el puntero y para reposicionarlo
 - Escritura al final del fichero lo extiende y avanza el puntero
- Métodos específicos más usados:
 - `long getFilePointer()` → obtiene posición actual del puntero
 - `void seek(long pos)` → establece nueva posición del puntero
 - `long length()` → devuelve longitud del fichero
 - `void setLength(long t)` → cambia longitud del fichero
 - Si mayor, lo extiende; si menor, lo trunca.

Entrada/salida en Java

50

Fernando Pérez Costoya (2011)