Fluxos d'informació en Java

Fitxers en Java: la classe File

La classe **File** representa objectes de tipus arxiu, que encapsulen el nom de l'arxiu i les seves propietats: dimensió, data de la darrera actualització, permisos de lectura i/o escriptura, etc. Permet fer la representació abstracta d'arxius, crear-los, esborrar-los, canviar les seves propietats, però no llegir ni escriure en ells.

Podem invocar de diverses maneres al constructor de la classe amb un arxiu determinat, donada la seva ruta completa. Per exemple, si es tracta de l'arxiu "\docs\carta.doc":

```
File unArxiu = new File("\docs\carta.doc");
File unArxiu = new File("\docs", "carta.doc");
File unDir = new File("\docs");
File unArxiu = new File(unDir, "carta.doc");
```

La taula següent reuneix els principals mètodes de la classe File:

Noms dels arxius	
String toString()	Retorna la conversió a String
String getName()	Retorna el nom
String getPath()	Retorna la ruta
String getAbsolutePath()	Retorna la ruta absoluta
String getParent()	Retorna la ruta pare
boolean renameTo(File nom)	Canvia el nom
Verificacions	
boolean exists()	Existència
boolean canWrite()	Permís d'escriptura
boolean canRead()	Permís de lectura
boolean isFile()	És un arxiu normal
boolean isDirectory()	Correspon a un directori
boolean isAbsolute()	És una ruta absoluta
boolean isHidden()	És ocult
Propietats	
long lastModified()	Data I hora de la darrera modificació
boolean setLastModified(long time)	Canvia la data I hora de la darrera modificació
long length()	Tamany de l'arxiu
boolean setReadOnly()	Activa atribut de només lectura
Directori i altres	
boolean mkdir()	Crea el directori
boolean mkdirs()	Crea el directori i els directoris pare necessaris
String[] list()	Array de noms dels arxius que conté el directori
File[] listFiles()	Array de File dels arxius que conté el directori
boolean delete()	Esborra l'arxiu o directori

El següent exemple ListDir.java il·lustra algunes de les característiques bàsiques mitjançant una classe per llistar els noms dels arxius d'un directori.

```
/**
 * ListDir.java
 * Llista el contingut d'un directori entrat per la línia d'ordres
 * @author Jose Moreno
 * @version
 */
import java.io.*;
public class ListDir {
	public static void main(String [] args) {
		File dir;		//objecte File per al directori
		String [] llistaArxius;  //vector de noms d'arxius del directori
		String sortida="";
		//comprovar paràmetres d'us (s'ha especificat la ruta al directori)
		if (args.length>0){
			dir = new File(args[0]);
			//comprovar que dir existeix i és un directori
			if (dir.exists() && dir.isDirectory()) {
				//obtenir llista d'arxius
				sortida+="\nRuta absoluta: "+dir.getAbsolutePath();
				sortida+="\nRuta relativa: "+dir.getPath();
				sortida+="\nNom: "+dir.getName();
				sortida+="\nContingut: ";
				sortida+="\n----------";
				llistaArxius = dir.list();
				for (int i=0; i<llistaArxius.length; i++) {
					sortida+="\n"; //canvi de línia
					sortida+=llistaArxius[i];
				}
			}
			else {
				sortida="Directori inexistent o no és un directori";
			}
		}
		else {
			sortida="Us: ListDir ruta_dir";
		}
		//escriure sortida
		System.out.println(sortida);
	}	//fi main()
}	//fi class ListDir
```

En el següent exemple, definim una classe ArxiuAtributs per mostrar les propietat d'un arxiu entrat per línia d'ordres.

```
/**
 * ArxiuAtributs.java
 * Mostra les propietats d'un fitxer entrat per la línia d'ordres
 * @author Jose Moreno
 * @version
 */
import java.io.*;
public class ArxiuAtributs {
	public static void main(String [] args) throws IOException
	{
		String sortida=""; //sortida
		//comprovar paràmetres d'us (s'ha especificat la ruta a l'arxiu)
		if (args.length==1){
			File f = new File(args[0]);
			sortida += "\nRuta: "+f.getPath();
			sortida += "\tNom: "+f.getName();
			sortida += "\nRuta absoluta: "+f.getAbsolutePath();
			//comprovar que l'arxiu existeix
			if (f.exists()) {
				sortida += "\nAtributs: ";	//atributs
				sortida += f.isDirectory()?"+d":"-d";
```

```
                        sortida += f.isFile()?"+f":"-f";
                        sortida += f.isHidden()?"+h":"-h";
                        sortida += f.canRead()?"+r":"-r";
                        sortida += f.canWrite()?"+w":"-w";
                        sortida += "\nTamany: "+f.length();
                        java.util.Date data = new java.util.Date(f.lastModified());
                        sortida += "\nDarrera modificacio: "+data.toString();
                }
                else {
                        sortida="L'arxiu o directori no existeix";
                }
        }
        else {
                sortida="Us: ArxiuAtributs ruta_dir";
        }
        //escriure sortida
        System.out.println(sortida);
    }    //fi main()
}    //fi class ArxiuAtributs
```

Fluxos (streams) en Java

La comunicació en Java entre un programa i un arxiu de dades es realitza a través d'una abstracció anomenada flux (**stream**). El paquet java.io de la biblioteca estàndard de Java conté una col·lecció de classes amb els algorismes de lectura i escriptura sobre arxius.

Els fluxos de dades de bytes es poden classificar en dos grans grups: fluxos d'entrada (**InputStream**) i fluxos de sortida (**OutputStream**), classes abstractes que deriven directament de la classe base Object.

Jerarquia de classes per a fluxos d'entrada i de sortida	
InputStream	**OutputStream**
FileInputStream	FileOutputStream
ByteArrayInputStream	ByteArrayOutputStream
PipeInputStream	PipeOutputStream
SequenceInputStream	
StringBufferInputStream	
FilterInputStream	FilterOutputStream
BufferedInputStream	BufferedOutputStream
LineNumberInputStream	
PushBackInputStream	
DataInputStream	DataOutputStream
	PrintStream

Els mètodes més importants de les classes base són:

InputStream	public int **read**() throws IOException
OutputStream	public void **write**(int b) throws IOException

FileInputStream i FileOutputStream proporcionen operacions bàsiques per llegir i escriure bytes a i des d'arxiu. Els constructors s'encarreguen d'obrir els arxius i accepten com a paràmetre un objecte de tipus String o de tipus File. Cas de no trobar l'arxiu, llancen l'excepció FileNotFoundException.

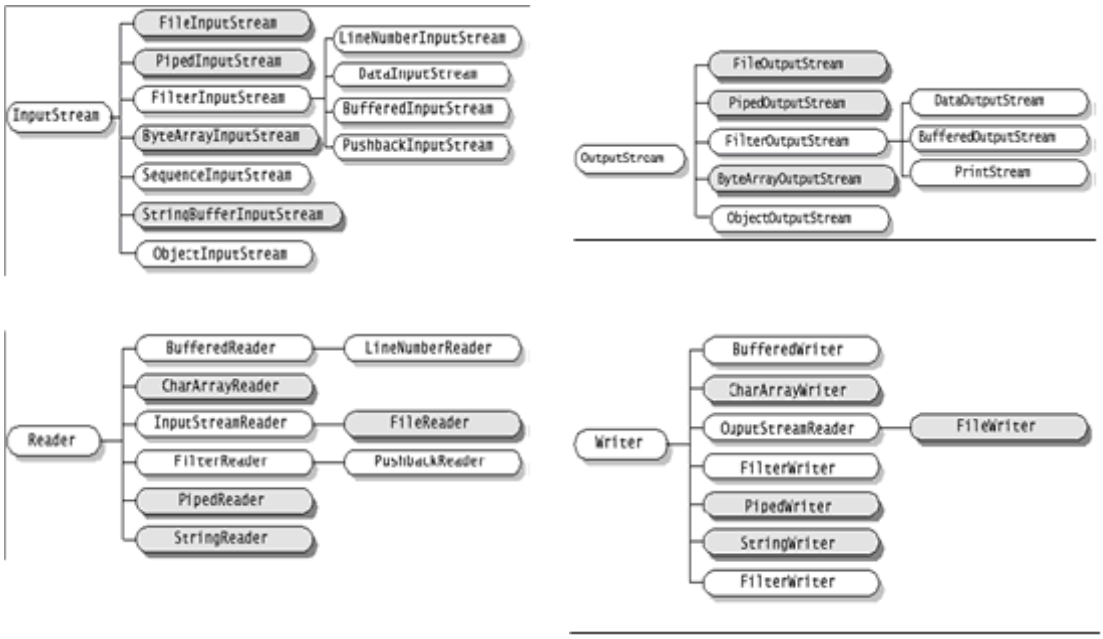
Les classes ByteArrayInputStream i ByteArrayOutputStream associen un flux amb un array de bytes en comptes de amb un arxiu. Les classes PipeInputStream i PipeOutputStream s'utilitzen per transferir dades entre tasques sincronitzades. Les classes FilterInputStream i FilterOutputStream, que deriven directament de InputStream i OutputStream respectivament, són classes abstractes orientades a *bytes*, però organitzats en seqüències per formar les dades primitives (int, long, double, ...).

Anàlogament a les classes base orientades a bytes, tenim les classes base abstractes que tracten amb caràcters: **Reader** i **Writer**. Al igual que amb les classes de bytes, també es disposa de classes que aporten entrada/sortida amb buffer: **BufferedReader** i **PrintWriter**. Els caràcters en Java es codifiquen en Unicode i ocupen 16 bits cada un.

BufferedReader té el mètode **readLine()** per llegir una línia sencera.

PrintWriter té els mètodes **print()** i **println()**.

La jerarquia de classes de *streams* es pot veure de forma resumida a continuació:



Lectura i escriptura de bytes a arxiu: FileInputStream i FileOutputStream

El següent exemple il·lustra l'escriptura a arxiu de dades en binari (bytes). En aquest cas, l'arxiu de sortida s'obre en mode escriptura. Si volem obrir-lo de manera que es pugui afegir informació sense destruir-ne el contingut previ, només cal cal afegir al constructor un paràmetre amb el valor true.

```
/**
 * EscriureBytes.java
 * Exemple d'ús de escriptura de bytes a arxiu
 * @author Jose Moreno
 * @version
 */
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
public class EscriureBytes {
	public static void main(String[] args) {
		byte [] llista ={10, 15, 25, 30, 45};
		if(args.length == 1) {//passat el nom de l'arxiu
			File arxiu = new File(args[0]);
			try{
				FileOutputStream fos = new FileOutputStream(arxiu);
				for(int i = 0; i < llista.length; i++){
					fos.write(llista[i]);
					fos.flush();
				}
				fos.close();
			} catch(IOException e) {
				System.out.println("Problema d'entrada o sortida relacionat amb la
següent excepció: ");
				e.printStackTrace();
			}
		}
		else {
			System.out.println("Us: EscriureBytes nom_arxiu");
		}
	}
}
```

Complementàriament, necessitem el següent exemple per llegir les dades en binari escrites amb l'anterior classe.

```
/**
 * LlegirBytes.java
 * Exemple d'ús de lectura de bytes d'arxiu
 * @author Jose Moreno
 * @version
 */
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class LlegirBytes {
	public static void main(String[] args) {
		if(args.length == 1) { //passat el nom de l'arxiu
			File arxiu = new File(args[0]);
			int x=0; //byte llegit
			try{
				FileInputStream fis = new FileInputStream(arxiu);
				while ((x = fis.read()) != -1) { //mentres no fi d'arxiu, llegir
					System.out.print(" "+(byte)x);
				}
				fis.close();
			} catch(IOException e) {
				System.out.println("Problema d'entrada o sortida relacionat amb la
següent excepció: ");
				e.printStackTrace();
			}
		}
		else {
			System.out.println("Us: LlegirBytes nom_arxiu");
		}
	}
}
```

Per llegir i escriure a fitxer els tipus primitius s'utilitzen les classes **DataInputStream** i **DataOutputStream**, que contenen mètodes per a la lectura i l'escriptura (binària) de cada un dels tipus primitius. També podem utilitzar, per a l'escriptura, la classe **PrintStream**, que sobrecarrega els mètodes print() i println() per als diferents tipus primitius.

Lectura i escriptura de caràcters a arxiu: BufferedReader, BufferedWriter i PrintWriter

Les classes **BufferedReader** i **BufferedWriter** deriven, respectivament de Reader i Writer, les quals serveixen per treballar amb caràcters, aportant a aquelles la capacitat de *buffer* per millorar el rendiment quan s'ha d'accedir a arxius en disc.

L'escriptura de caràcters es fa amb el mètode void write() i la lectura de caràcters amb el mètode int read().

Per facilitar l'escriptura de tipus primitius, es disposa de la classe **PrintWriter**, la qual a més del mètode void write() té sobrecarregats els mètodes **void print()** i **void println()** per escriure cada un dels tipus de dades primitius.

Exemple EscriureCars.java

Aquest exemple il·lustra la manera d'escriure caràcters amb la classe BufferedWriter a un fitxer.

```
/**
 * EscriureCars.java
 * Exemple d'ús de escriptura de caràcters a arxiu
 * @author Jose Moreno
 * @version
 */
import java.io.*;
public class EscriureCars {
	public static void main(String[] args) {
		char [] llista = {'a', 'e', 'i', 'o', 'u'};
		if(args.length == 1) {//passat el nom de l'arxiu
			try{
				BufferedWriter bw = new BufferedWriter(new FileWriter(args[0]));
				for(int i = 0; i < llista.length; i++){
					bw.write(llista[i]);
				}
				bw.close();
			} catch(IOException e) {
				System.out.println("Problema d'entrada o sortida relacionat amb la
següent excepció: ");
				e.printStackTrace();
			}
		}
		else {
			System.out.println("Us: EscriureCars nom_arxiu");
		}
	}
}
```

Exemple LlegirCars.java

Aquest exemple il·lustra la manera de llegir caràcters amb la classe BufferedReader d'un arxiu. És l'exemple complementari de l'anterior i serveix per llegir els arxius creats amb ell.

```
/**
 * LlegirCars.java
 * Exemple d'ús de escriptura de caràcters a arxiu
 * @author Jose Moreno
 * @version
 */
import java.io.*;
public class LlegirCars {
	public static void main(String[] args) {
		int c; //caràcter llegit
		if(args.length == 1) {//passat el nom de l'arxiu
```

```
                try{
                BufferedReader br = new BufferedReader(new FileReader(args[0]));
                while ((c=br.read()) != -1) {
                        System.out.print((char)c);
                }
                br.close();
                } catch(IOException e) {
                        System.out.println("Problema d'entrada o sortida relacionat amb la
següent excepció: ");
                        e.printStackTrace();
                }
        }
        else {
                System.out.println("Us: LlegirCars nom_arxiu");
        }
    }
}
```