Programmazione Java
Corso Pratico
07 - Accesso al filesystem

Maurizio Franco

Introduzione alla classe java.io.File

La classe java.io.File è una classe rappresentativa di un file fisico, e del percorso(path) assoluto che porta ad esso.

Si potrebbe dire che la classe File è una rappresentazione astratta del file stesso.

Accesso al Filesystem

Java mette a disposizione diversi strumenti per poter accedere in lettura e scrittura su di un file.

Qualsiasi modo di accesso ad un file, in java, avviene attraverso uno stream.

Lo stream è un flusso di dati.

Se abbiamo necessità di accedere ad una sorgente, che può essere un file, uno allocazione di memoria, un socket, apriremo, verso di esso, uno stream, e provvederemo a leggere le informazioni in maniera sequenziale.

Analogamente, se abbiamo necessità di scrivere, verso le stesse tipologie di destinazione, apriremo uno stream, e provvederemo a scrivere in maniera sequenziale.

STREAM

Per leggere	Per scrivere
open uno stream	**open** a stream
while ci sono dati	**while** ci sono dati
read dati	**write** dati
close lo stream	**close** lo stream

Stream a 8 e a 16 bit

A seconda che dobbiamo operare su caratteri o altro tipo di dati(suoni, immagini, ecc.) andremo ad utilizzare du categorie di oggetti rappresentativi degli stream.

Per i caratteri, quindi standard Unicode, 16 bit, andremo ad utilizzare gli oggetti java.io.**Reader** e java.io.**Writer.** Essi sono oggetti astratti; esistono tutta una serie di implementazioni le cui utilità variano a seconda dell'uso.

Per tutto ciò che non riguarda i caratteri, utilizzeremo oggetti stream limitati ai bytes da 8-bit dell'ISO-Latin-1.
Per queste casistiche faremo riferimento alle classi astratte java.io.**InputStream** e java.io.**OutputStream** ed a tutte le loro dirette implementazioni.

LETTURA

Reader e InputStream definiscono metodi simili ma per diversi tipi di dato.

Per esempio, la classe Reader contiene i seguenti metodi per leggere caratteri e array di caratteri:

```
int read()

int read(char cbuf[])

int read(char cbuf[], int offset, int length)
```

LETTURA pt.2

La classe InputStream definisce gli stessi metodi ma per leggere bytes e array di bytes:

int read()

int read(byte cbuf[])

int read(byte cbuf[], int offset, int length)

SCRITTURA

La classe Writer definisce I seguenti metodi per scrivere caratteri e array di caratteri:

int write (int c)

int write (char cbuf[])

int write (char cbuf[], int offset, int length)

SCRITTURA pt.2

La classe OutputStream definisce gli stessi metodi ma per I bytes:

int write (int c)

int write (byte cbuf[])

int write (byte cbuf[], int offset, int length)

Esempio di lettura file.

```
public class ReadingFromAFileDemo {
  public static void main(String[] args) throws IOException    {
        // Legge file per linee
        BufferedReader in = new BufferedReader(
          new FileReader("c:/prova.txt"));
        String s, s2 = new String();
        while((s = in.readLine())!= null) {
            s2 += s + "\n";
        }
        in.close();
        System.out.println(s2);
    }
}
```

Esempio di lettura dallo standard input.

```
public class ReadingFromKeyboard {
  public static void main(String[] args) throws IOException {
  //Legge da standard input
    BufferedReader stdin = new BufferedReader(
          new InputStreamReader(System.in));
    System.out.print("Enter a line:");
    System.out.println(stdin.readLine());
  }
}
```

Esempio di scrittura di un file

```
public class WritingToAFileDemo {
   public static void main(String[] args) throws Exception {
    String fileToWrite = "C:/corsojava/provaScrittura.txt" ;
    File f= new File(fileToWrite);
    //creiamo un outputstream con il file.
    FileOutputStream fos= new FileOutputStream(f);
    //creiamo un nuovo printwriter per scrivere
    //sull'outputstream del file passandogli come
    //parametro l'output stream del file.
    PrintWriter pw=new PrintWriter(fos);
    //scriviamo la stringa ciao sul file.
    pw.println("ciao");
    pw.close();
   }
}
```

NIO
NEW INPUT/OUTPUT

Accesso al Filesystem

Introdotte con la jdk 1.4, il package java.nio fornisce una serie di risorse per implementare lettura e scittura di dati ad alta efficienza/velocità.

La principale differenza tra le librerie io e quelle nio sono relativa a come I dati sono pacchettizzati e trasmessi.

Le io trasmettono tramite gli streams.
Le nio trasmettono tramite blocchi.

Accesso al Filesystem

Il package NIO introduce l'interfaccia Channel.
Può essere paragonata al “vecchio” stream.
Un'istanza del Channel può essere utilizzata per leggere e scrivere i dati.

I Channel diversamente dagli stream sono bi-direzionali.

Esempio di lettera di un file...

```
public static void main(String[] args) throws
  Exception {
   String filePathToRead = "C:/corsojava/Prova.java" ;
   File fileToRead = new File (filePathToRead) ;
   FileInputStream fis = new
  FileInputStream(fileToRead) ;
   FileChannel inChannel = fis.getChannel();
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    while(inChannel.read(buffer) > 0) {
        buffer.flip();
        for (int i = 0; i < buffer.limit(); i++) {
            System.out.print((char) buffer.get());
        }
        buffer.clear();
    }
    inChannel.close();
    fis.close();
}
```

Esempio di scrittura di un file...

```
public static void main(String[] args) throws
  Exception {
   String filePathToWrite =
  "C:/corsojava/ProvaOutput.txt" ;
   FileOutputStream fos = new
  FileOutputStream(filePathToWrite) ;
   FileChannel outChannel = fos.getChannel();
        ByteBuffer buffer = ByteBuffer.allocate(1024);
   String textToWrite = "ciao" ;
   buffer.put(textToWrite.getBytes());
   buffer.flip();
   outChannel.write(buffer);
//  buffer.clear();
        outChannel.close();
        fos.close();
}
```

Esempio di copia di un file...

```
public static void main(String[] args) throws
  Exception {
   File sourceFile = new
  File("C:/corsojava/Prova.java") ;
   File destFile = new
  File("C:/corsojava/ProvaOutputJavaFile.java") ;

   FileChannel source = null;
   FileChannel destination = null;

   source = new
  FileInputStream(sourceFile).getChannel();
   destination = new
  FileOutputStream(destFile).getChannel();
   destination.transferFrom(source, 0, source.size());

   source.close();
   destination.close();
}
```

LA SERIALIZZAZIONE

La serializzazione è quel processo che permette di trasformare un'oggetto java(la cui classe implementa l'interfaccia Serializable) in una sequenza di bytes.

La serializzazione crea un'immagine dell'oggetto.

E' molto utile/usata per lo scambio di oggetti sulla rete.

La deserializzazione è quel processo che permette di trasformare una sequenza di bytes in un'oggetto java.

Non sempre si vuole serializzare tutto di un oggetto.

Ad esempio se un oggetto contiene dati sensibili o password.

Per evitare la serializzazione di alcuni attributi è sufficiente dichiararli come transient.

Come si rende un oggetto serializzabile?

implementando l'interfaccia java.io.Serializable

Come si customizza la serializzazione di un oggetto?

Facendo l'override(con relativa implementazione) dei metodi
writeObject
e
readObject

Esempio di serializzazione di un oggetto...

```
public void serializeTest () throws Exception {
        Messaggio mex = new Messaggio () ;
        [...]
        FileOutputStream fos = null ;
        fos = new FileOutputStream ("c:/store.ser");
        ObjectOutputStream oos ;
        oos = new ObjectOutputStream(fos);
        oos.writeObject(mex);
        oos.close();
}
```

Esempio di deserializzazione di un oggetto...

```
public void deserializeTest () throws Exception {
        Messaggio mex = null ;
        FileInputStream fis = null ;
        fis = new FileInputStream ("c:/store.ser");
        ObjectInputStream ois ;
        ois = new ObjectInputStream(fis);
        Messaggio mex = (Messaggio)ois.readObject();
        ois.close();
}
```

PROPERTIES