Input e Output in Java

Stream

Redirecting

Scomposizione dell'input

Uso di file

Set di caratteri

Inserimento dati e test

- Riconsideriamo la versione iniziale della classe **`DataSet`** usata per illustrare le interfacce
- Scrivere un **`main`** di test che legge una serie di dati dallo standard input e li inserisce in un oggetto della classe **`DataSet`**
- Testare i metodi per la media e il massimo

Inserimento dati: quando i dati sono tanti

- È molto noioso e ripetitivo inserire dati in maniera interattiva se i dati sono molti
- Sarebbe utile poter inserire tutti i dati che servono in un file di testo e poi fare in modo che il programma li prenda da lì
- Un modo molto semplice per fare questo è quello di utilizzare la lettura di dati da console e poi, in fase di esecuzione del programma, **reindirizzare l'input da un file**
- Nel main bisogna prendere l'input da **`System.in`**

Lettura da console

```
BufferedReader console = new BufferedReader(new
        InputStreamReader(System.in));
    // Creo un DataSet
    DataSet data = new DataSet();
    boolean finito = false;
    while ( !finito ) {
      System.out.println(
        "Inserisci un valore numerico. Chiudi l'input per
  terminare");
      String input = console.readLine();
      // Continua...
```

Lettura da console

```
      // Se l'input è stato chiuso input==null
      if ( input == null )
        finito = true;
      else {
        /* è stato inserito un nuovo dato: lo aggiungo al
DataSet creato */
        double x = Double.parseDouble(input);
        data.add(x);
      }
    }
// Stampa la media e il massimo
```

Chiusura dell'input

- Per poter utilizzare questo programma (**`InputTestLoopRedirect`**) occorre lanciarlo da console (prompt dei comandi Dos o shell di Linux)

```
prompt#> java InputTestLoopRedirect
Inserisci un valore numerico. Chiudi l'input per terminare
16
Inserisci un valore numerico. Chiudi l'input per terminare
32
Inserisci un valore numerico. Chiudi l'input per terminare
^Z
Media dei dati = 24.0
Valore massimo = 32.0
```

Chiusura dell'input

- In Dos/Windows per chiudere l'input da tastiera basta premere Ctrl-z
- In Linux/Unix invece si usa Ctrl-d
- Un'alternativa è scrivere un file di testo in cui in ogni riga mettiamo un valore numerico
- Poi redirigiamo l'input del programma su questo file, invece che sulla tastiera:
- Ad esempio supponiamo di avere scritto i dati in un file "datiInputTestLoopRedirect.txt"

Redirecting dell'input

```
#> java InputTestLoopRedirect < datiInputTestLoopRedirect.txt
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Media dei dati = 24.0
Valore massimo = 32.0
```

- Il file di testo deve contenere esclusivamente le seguenti due righe:

```
16
32
```

- La chiusura dell'input viene segnalata quando viene incontrato il carattere di EOF (fine file)

Scomposizione di stringhe

- E se volessimo, per comodità, inserire i valori nel file di testo anche su una stessa riga?
- Ad esempio:

```
13  343.54  100.09  25
1.8  12  3
33
```

- Il metodo **`readLine()`** della classe **`BufferedReader`** legge una intera riga per volta
- Se proviamo a fare il parsing di una riga con diversi numeri separati da spazi bianchi il metodo **`parseDouble`** solleva un'eccezione

Scomposizione di stringhe

- Ci viene in aiuto la classe **`java.util.StringTokenizer`**
- Un oggetto di questa classe va costruito passandogli una certa stringa
- Dopodiché è possibile utilizzare la coppia di metodi
  - **`hasMoreToken()`**
  - **`nextToken()`**
- per prendere un token, cioè un insieme di caratteri contigui (non separati da spazi tab o newline), per volta fino all'esaurimento della stringa stessa

Scomposizione

- Se si prova a chiamare **`nextToken()`** senza che ci siano token viene sollevata un'eccezione: si deve sempre controllare prima con **`hasMoreToken()`**:

```
...
StringTokenizer tokenizer = new
  StringTokenizer(input);
while (tokenizer.hasMoreTokens()) {
  String token = tokenizer.nextToken();
  double x = Double.parseDouble(token);
  data.add(x);
}
...
```

Redirecting dell'output

- Supponiamo di scrivere il file di testo di prima:
- Supponiamo di chiamarlo datiInputTestLoopToken.txt
- Lanciando il programma con il redirecting dell'input si ottengono comunque quattro stringhe in output che corrispondono alle richieste dei dati (utili solo nel caso che non si usi il redirect)

Redirecting dell'output

```
#> java InputTestLoopToken < datiInputTestLoopToken.txt
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Media dei dati = 66.42875000000001
Valore massimo = 343.54
```

- Tutto questo output può essere anch'esso reindirizzato su un file di testo, ad esempio output.txt:

```
#> java InputTestLoopToken < datiInputTestLoopToken.txt > output.txt
```

Lettura e scrittura di file

- Naturalmente possiamo anche inserire esplicitamente nei nostri programmi la lettura di input da un file e la scrittura di output su un file
- Il tutto senza utilizzare il redirecting dello standard input o output
- Vediamo come è stato modellato il concetto di stream in Java e le varie classi per leggere/scrivere stream

Stream

- Uno stream è un flusso di entità
- Uno stream è di input se le entità fluiscono dall'esterno verso la nostra applicazione
- Uno stream è di output se le entità fluiscono dalla nostra applicazione verso l'esterno
- Generalmente le entità che scorrono in uno stream possono essere viste in due modi:
  - caratteri
  - byte

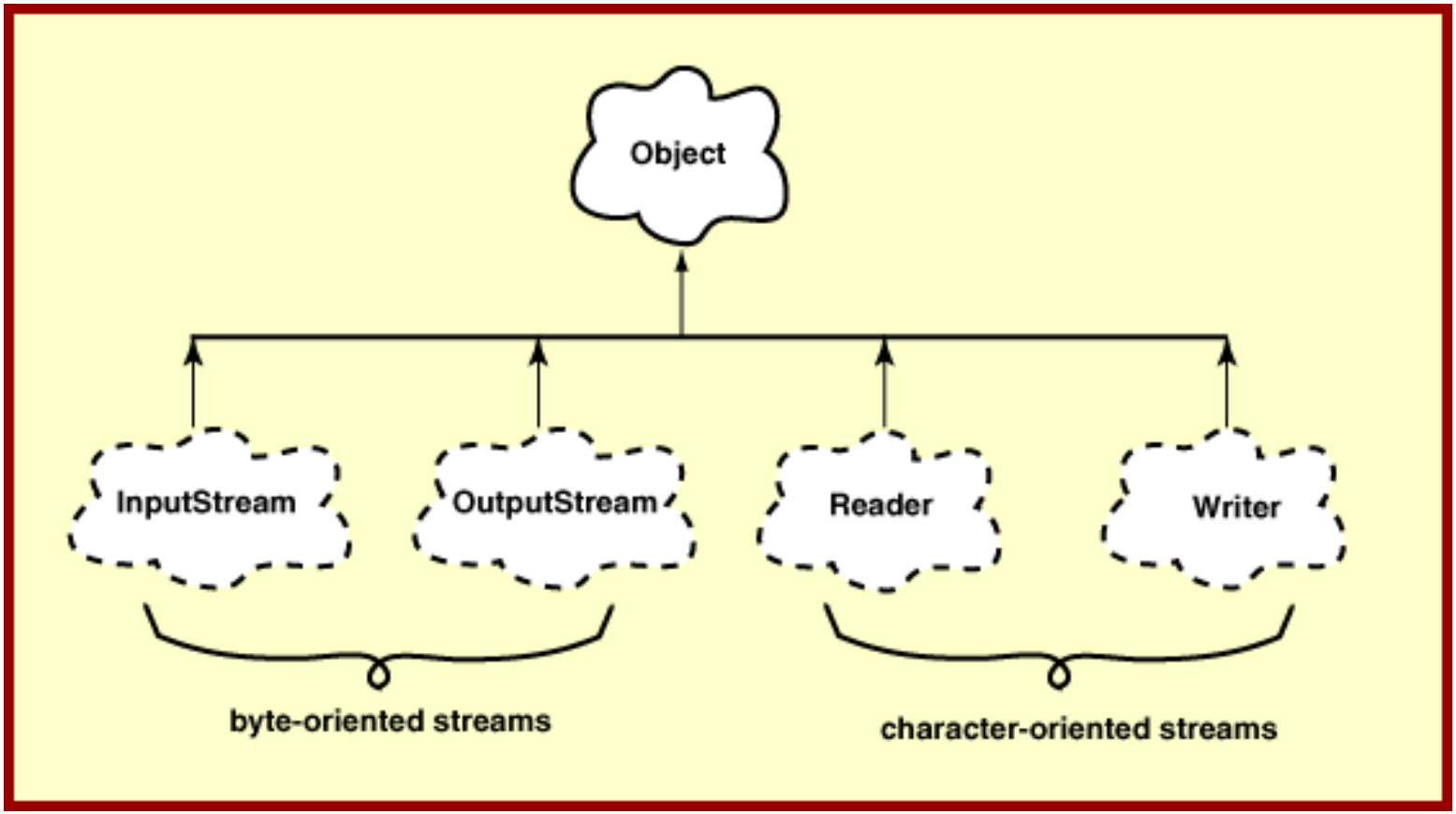
Stream

- Gli stream servono per memorizzare dati o per leggere dati precedentemente memorizzati
- Se i dati sono in formato testo allora sono rappresentati da caratteri, altrimenti da byte
- ‘1’ ‘2’ ‘3’ ‘4’ ‘5’ rappresenta il numero 12345 in un **file di testo**
- 0 0 48 57 è una sequenza di 4 byte che rappresenta il numero 12345 (12345 = 48 * 256 + 57) in un **file binario**

Input e output

- Le librerie standard usano il concetto di *flusso* (*stream*) che rappresenta un flusso di entità:
  - in input se esse fluiscono dall'esterno verso l'applicazione
  - in output se esse fluiscono dall'applicazione verso l'esterno
- Generalmente le entità che scorrono in uno stream possono essere viste in due modi:
  - caratteri
  - byte
- Secondo queste due filosofie possiamo individuare due *gerarchie* di classi per l'I/O, una basa sulle classi **`InputStream`** e **`OutputStream`** e l'altra basata su **`Reader`** per l'input e **`Writer`** per l'output

Confronto tra classi base



Le classi da usare in java.io

- Gli oggetti delle classi **`FileReader`** e **`FileWriter`** rappresentano **file di testo** in input o in output
- Gli oggetti delle classi **`FileInputStream`** e **`FileOutputString`** rappresentano **file binari** in input o in output
- Alla creazione di uno qualsiasi di questi oggetti va passata al costruttore una stringa contentente il path+nome del file da cercare

Apertura di file

```
FileReader reader = new FileReader("input.txt");
```

- Apre un file di testo in lettura

```
FileWriter writer = new FileWriter("output.txt");
```

- Apre un file di testo in scrittura

```
FileInputStream inputStream = new
  FileInputStream("input.dat");
```

- Apre un file binario in lettura

```
FileOutputStream outputStream = new
  FileOutputStream("output.dat");
```

- Apre un file binario in scrittura

Percorsi relativi

- Scriviamo in “Dir/prova.tx t” o “Dir\prova.tx t” :

```
FileWriter fw = new FileWriter("Dir" + File.separator
+ "prova.txt");
```

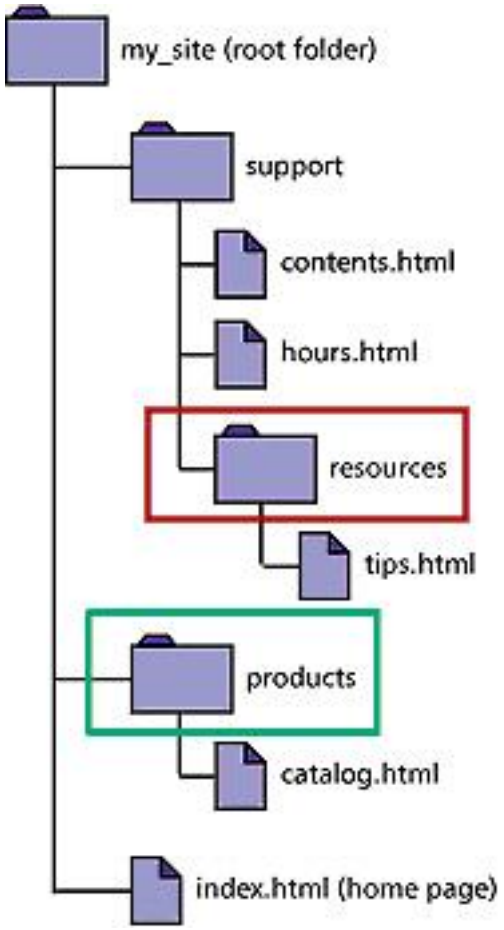
  generiamo un file di testo nella *sottocartella* “Dir” della cartella corrente

- Possiamo anche *risalire* la gerarchia usando l’operatore “..”:

```
FileWriter fw = new FileWriter(".." + File.separator
+ "prova.txt");
```

  generiamo un file prova nella directory *padre* della directory corrente!

Percorsi complessi



- Supponiamo che la directory corrente sia “resources” e che vogliamo creare un file “testo.txt” in “products”; qual è il percorso da definire e il codice necessario??

```
String path = ".." +
   File.separator + ".." +
   File.separator + "products" +
   File.separator + "testo.txt";
File mioFile = new File(path);
mioFile.createNewFile();
```

Lettura

- Sia **`FileInputStream`** che **`FileReader`** hanno un metodo **`read()`** che serve per leggere un byte o un carattere (rispettivamente) alla volta
- In ogni caso entrambi i metodi restituiscono un int:
  - se il valore restituito è –1 allora è stata raggiunta la fine del file
  - Se il valore restituito è non negativo allora si può fare il casting a **`byte`** o a **`char`** (rispettivamente) per ottenere il valore letto

Lettura

- File di testo:

```
int next = reader.read();
char c;
if (next != -1)
  c = (char) next; // c è il carattere letto
else fine file
```

- File binario:

```
int next = inputStream.read();
byte b;
if (next != -1)
  b = (byte) next; // b è il byte letto
else fine file
```

Scrittura

- File di testo

```
char c = ...;
writer.write(c);
```

- File binario:

```
byte b = ...;
outputStream.write(b);
```

Chiusura

- Ogni file aperto in qualsiasi modalità va chiuso quando il programma ha finito di operare su di esso:

```
riferimentoAlFile.close();
```

Agevolazioni per i file di testo

- Leggere o scrivere un carattere per volta nei file di testo può risultare scomodo
- Possiamo incapsularli in oggetti più sofisticati che realizzano una interfaccia a linee
- È quello che facciamo sempre, ad esempio, quando leggiamo linee di testo dallo standard input
- La classe da usare per i file di testo in lettura la conosciamo già: è **`BufferedReader`**

Agevolazioni per i file di testo

```
FileReader file = new
  FileReader("input.txt");
BufferedReader in = new
  BufferedReader(file);
String inputLine = in.readLine();
```

- Già lo conosciamo: otteniamo una linea di testo con il metodo **`readLine()`**

Agevolazioni per i file di testo

```
FileWriter file = new
  FileWriter("output.txt");
PrintWriter out = new PrintWriter(file);
out.println(29.25);
out.println(new Rectangle(5,10,20,30));
out.println("Hello, World!");
```

- La classe **`PrintWriter`** è molto simile alla classe **`PrintStream`** che già conosciamo (è la classe a cui appartiene **`System.out`**)
- Il metodo **`println`** si usa nel modo che conosciamo

Esempio

- Creiamo un logger con input da tastiera e copia su file!
- Facciamo un redirect dell'input!

Logger 1

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

public class E05_FileWriterEcho {
        public static void main(String[] args) throws IOException {
            BufferedReader console = new BufferedReader(
            new InputStreamReader(System.in));
            FileWriter file = new FileWriter("logger.txt");
            PrintWriter out = new PrintWriter(file);

            int i = 4;
            System.out.printf("Inserisci %d righe:\n", i);
            while(i-- > 0)
               out.println(console.readLine());
            out.close();

            System.out.println("Logger end!");

        }

}
```

```
Prompt> java E05_FileWriterEcho
```

Logger 2

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

public class E05_FileWriterEcho {
        public static void main(String[] args) throws IOException {
            BufferedReader console = new BufferedReader(
            new InputStreamReader(System.in));
            FileWriter file = new FileWriter("logger.txt");
            PrintWriter out = new PrintWriter(file);

            int i = 4;
            System.out.printf("Inserisci %d righe:\n", i);
            while(i-- > 0)
               out.println(console.readLine());
            out.close();

            System.out.println("Logger end!");

        }
}
```

```
Prompt> java E05_FileWriterEcho < input.txt
```

Esercizio

- Modificare Logger 2 in modo da chiedere in input quante righe copiare.

- All'avvio usare **`JOptionPane.showInputDialog`** per chiedere il n° di input.

Logger 3

```
/* imprt ugualia prima*/

public class E05_FileWriterEcho_2 {

      public static void main(String[] args) throws IOException {
            BufferedReader console = new BufferedReader(
            new InputStreamReader(System.in));
            FileWriter file = new FileWriter("logger.txt");
            PrintWriter out = new PrintWriter(file);

            String input = "";
            System.out.printf("Inserisci una righa:\n");

            do{
               input = console.readLine();
               if(input == null)
                 break;
               out.println(input);

               System.out.printf("Inserisci una nuova righa:\n");
            }while(true);

            out.close();

            System.out.println("Logger end!");

      }
}
```

```
Prompt> java E05_FileWriterEcho < input.txt
```

Ricerca di un file nelle cartelle

- Può essere utile, quando si vuole aprire un file in input o in output, presentare all'utente la classica finestra di selezione di un file che gli permette di navigare tra le sue cartelle
- Ci viene in aiuto la classe **`javax.swing.JFileChooser`**

Ricerca di un file nelle cartelle

```
JFileChooser chooser = new JFileChooser();
FileReader in = null;
if (chooser.showOpenDialog(null) ==
            JFileChooser.APPROVE_OPTION) {
  File selectedFile =
                  chooser.getSelectedFile();
  in = new FileReader(selectedFile);
}
```

Logger 4

```
public class E05_FileWriterEcho_3 {
  public static void main(String[] args) throws IOException {
    // Apertura file di input
    JFileChooser chooser = new JFileChooser();
    FileReader in = null;
    if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
      File selectedFile = chooser.getSelectedFile();
      in = new FileReader(selectedFile);
    }

    BufferedReader console = new BufferedReader(in);
    FileWriter file = new FileWriter("logger.txt");
    PrintWriter out = new PrintWriter(file);

    String input = "";
    do{
      input = console.readLine();
      if(input == null)
        break;
      out.println(input);
    }while(true);

    out.close();

    JOptionPane.showMessageDialog(null, "Copia terminata");
    System.out.println("Logger end!");
    System.exit(0);
  }
}
```

```
Prompt> java E05_FileWriterEcho_3
```

Esercizio

- Scrivere un main di una classe **`MyTextCopy`** che permette di:
- selezionare un file di testo in input
- scegliere un file di output
- copiare il contenuto del file di input in quello di output (copy)
- Scrivere poi un main di una classe **`MyCopy`** che faccia la stessa cosa per i file binari

Copy

```
public class E05_MyTextCopy_Bin {
    public static void main(String[] args) throws IOException {
      // Apertura file di input
      JFileChooser chooser = new JFileChooser();
      FileInputStream in = null;
      if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
        File selectedFile = chooser.getSelectedFile();
        in = new FileInputStream(selectedFile);
      }

      // Apertura file di output
      JFileChooser chooserSave = new JFileChooser();
      FileOutputStream file = null;
      if (chooserSave.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
        File selectedFile = chooserSave.getSelectedFile();
        file = new FileOutputStream(selectedFile);
      }

      int b = -1;
      while((b = in.read()) != -1){
        file.write(b);
      }
      file.close();
      in.close();

      JOptionPane.showMessageDialog(null, "Copia terminata");
      System.out.println("Logger end!");
      System.exit(0);
    }
}
```

E' lento!!

Esercizio

- Unificare **MyTextCopy** e **MyTextCopyBin** che avete appena creato!

- All'avvio usare **`JOptionPane.showInputDialog`** per chiedere la modalità copia (opzioni valide):
  - Testo
  - Binario
  - Se la stringa diversa ripeto la richiesta

Append

- A volte si vogliono aggiungere dei caratteri o dei byte alla fine di un file già esistente
- L'apertura di un file con i costruttori **`FileWriter(String nomeFile)`** o **`FileOutputStream(String nomeFile)`** cancella il contentuto del file **`nomeFile`**, se questo era esistente

Append

- Per mantenere tutte le informazioni e aggiungerne di nuove si possono utilizzare i costrutturori **`FileWriter(String nomeFile, true)`** o **`FileOutputStream(String nomeFile, true)`**
- Il secondo parametro booleano con valore **`true`** indica che l'apertura deve essere fatta in append

Append

- Se si inserisce il valore **`false`** si ha lo stesso comportamento del costruttore che prende solo la stringa contenente il nome del file
- Le stesse considerazioni valgono immutate nel caso in cui il primo parametro passato ai costruttori sia un oggetto **`File`** invece che il nome di un file

Esercizio

- Modificare il logger (e.s. Logger4) in modo da prevedere l'opzione di append.

- All'avvio usare **`JOptionPane.showInputDialog`** per chiedere l'attivazione dell'append:
  - Si
  - No
  - Se la stringa è diversa ripeto la richiesta

Random Access File

- Se si vuole aprire un file (di testo o binario) sia in lettura che in scrittura bisogna utilizzare la classe **`RandomAccessFile`** del pacchetto **`java.io`**
- In questo caso il file non è più visto come uno stream
- Il file viene visto come un grande array di byte che si trova sul disco

Random Access File

- E' possibile navigare in questo array tramite un indice (position), leggere, scrivere e aggiungere posizioni in fondo
- Per altre informazioni consultare le API

Set di caratteri

- Sappiamo che la rappresentazione interna dei caratteri di Java è a 16 bit
- Il Charset corrispondente è denominato

  UTF-16

- Tale Charset è a lunghezza fissa, cioè utilizza 16 bit per mappare 2^{16} caratteri diversi e ogni carattere, anche se i bit significativi sono meno di 16, occupa sempre 16 bit
- Negli anni sono stati definiti altri standard

Set di caratteri

- Il primo set di caratteri che sia stato standardizzato è quello ASCII a 7 bit che contiene 33 simboli non stampabili e 95 simboli stampabili. Denominazione: US-ASCII
- Contiene le 26 lettere dell'alfabeto inglese maiuscole e minuscole, le cifre decimali e qualche simbolo di punteggiatura
- Non ci sono lettere accentate e non ci sono lettere di altri alfabeti o altri simboli

Set di caratteri

- A partire dall'US-ASCII sono stati definiti molti set di caratteri a 8 bit
- I 128 simboli in più ottenibili rispetto a quelli dell'US-ASCII sono stati assegnati con gruppi di caratteri affini
- Ad esempio il Charset ISO-8859-1, denominato anche ISO-LATIN-1, include, oltre all'US-ASCII, tutte le lettere accentate degli alfabeti delle lingue dell'Europa Occidentale

Set di caratteri

- Il Charset ISO-8859-15, denominato ache ISO-LATIN-9, è una modifica dell'ISO-8859-1 che ridefinisce alcuni caratteri
- Ad esempio al valore decimale 162 viene associato il simbolo dell'euro, sostituendo un simbolo meno cruciale per i Paesi che utilizzavano il set di caratteri precedente ISO-8859-1

Set di caratteri

- Attualmente molti sistemi operativi utilizzano ISO-8859-1 (o 15) come set di caratteri di default per i file di testo se il “locale” del sistema si trova in un Paese dell'Europa occidentale
- Quindi probabilmente nel vostro PC i file di testo verranno scritti utilizzando questa codifica a 8 bit
- Cioè nel file ogni byte (8 bit) rappresenta un solo carattere: quel carattere definito dal Charset

Set di caratteri

- Nell'ultima generazione di set di caratteri la rappresentazione di un certo carattere richiede un numero diverso di bit, a seconda del carattere stesso
- I Charset di questo tipo sono detti a lunghezza variabile
- Fra i vari standard di questo tipo di certo il più usato è l'UTF-8
- Nell'UTF-8 i caratteri dell'US-ASCII richiedono 8 bit (7 bit normali e il più significativo a 0)

Set di caratteri

- Nell'UTF-8 i caratteri dei vari Charset simili a ISO-8859-1 richiedono un numero di byte che va da 1 a 2
- L'UTF-8 codifica caratteri di tutte le lingue del mondo, anche morte
- L'UTF-8 può arrivare ad usare fino a 4 byte per rappresentare un carattere
- Tuttavia vengono usati più byte solo quando è necessario

Set di caratteri

- L'UTF-8 è quindi una codifica che risparmia molto rispetto alle codifiche a lunghezza fissa
- In alcuni sistemi operativi è possibile impostare l'UTF-8 come set di caratteri di default
- Nelle pagine web è sempre più diffuso l'UTF-8 come Charset di codifica
- Si può dire che al momento attuale rappresenta il miglior candidato per uno standard mondiale

Charset e Java

- In Java è possibile gestire i vari Charset in maniera semplice e trasparente
- Si può aprire un file di testo in lettura o un file di testo in scrittura specificando quale Charset si vuole utilizzare
- In questo modo la rappresentazione interna sarà sempre UTF-16, ma l'esportazione e l'importazione verso/da altri Charset permette il funzionamento del programma in contesti in cui ci sono diverse rappresentazioni

Charset e file di input

- La classe **`InputStreamReader`** ha un costruttore che permette di specificare un Charset
- Quindi per aprire un file di testo in input che contiene testo codificato in UTF-8 possiamo scrivere:

Charset e file di input

```
FileInputStream f =
        new FileInputStream("myFile.txt");
InputStremaReader isr =
        new InputStreamReader(f, "UTF-8");
BufferedReader input =
                    new BufferedReader(isr);
```

- Possiamo leggere correttamente da **`input`** le righe del file come **`String`** che usano la rappresentazione interna di Java

Charset e file di output

- La classe **`OutputStreamWriter`** ha l'analogo costruttore di **`InputStreamReader`** che permette di specificare il Charset
- In questo caso dalla rappresentazione interna di Java nelle **`String`** i caratteri verranno scritti nello stream di output codificati nel Charset indicato
- Vediamo l'esempio di una scrittura su un file di testo con codifica ISO-8859-1

Charset e file di output

```
FileOutputStream f =
        new FileOutputStream("myFileO.txt");
OutputStremaWriter osw =
   new OutputStreamWriter(f, "ISO-8859-1");
PrintWriter output =
                  new PrintWriter(osw);
```

- Possiamo scrivere correttamente su **`output`** qualsiasi oggetto. I caratteri in output saranno in formato ISO-8859-1

Esercizio

- Scrivere una classe **`TextConversions`** che contiene diversi metodi statici per la conversione del contenuto di files di testo
- Ad esempio ci sarà un metodo

```
public static boolean
        fromUtf8toLatin1(
                        File from, File to)
```

Esercizio

- Il metodo deve convertire i caratteri del file **`from`** dal formato UTF-8 al formato ISO-8859-1 e produrre una copia del file così codificato in **`to`**
- Prevedere diversi tipi di conversione fra Charset diversi
- Prevedere anche la conversione che ha come output lo stesso file: il file viene convertito in un file temporaneo che poi va a sostituire il file originale