Uno dei problemi più comuni che si possono incontrare quando si lavora su applicazioni web o client-server di portata internazionale, che hanno a che fare con valori numerici, è la gestione dei formati tipici della lingua a cui appartiene l'utente che sta utilizzando l'applicazione.

Per intenderci vediamo un esempio.

Supponiamo che la X-Bank sia una banca presente in diversi stati e che, tra le sue filiali, ci siano anche una filiale di Roma ed una filiale di New York.

Per chi non lo sapesse l'Italia e gli Stati Uniti, pur adottando fondamentalmente la medesima notazione numerica araba, utilizzano una rappresentazione differente (praticamente invertita) dei separatori decimali e di quelli delle migliaia.

Ad esempio la rappresentazione del numero
“mille-duecento-cinquanta-quattro-virgola-trenta-cinque” che in Italia (IT) risulta essere (con o senza separatore delle migliaia):

1.254,35 o 1254,35

negli Stati Uniti (US) diventa “mille-duecento-cinquanta-quattro-punto-trenta-cinque” ossia (con

o senza separatore delle migliaia):

1,254.35 o 1254.35

Ora vi chiederete, ma questo cosa significa? Quali conseguenze comporta ?
Ebbene le risposte a queste domande sono tanto veloci quanto evidenti.

1. Il medesimo programma dovrà ricevere gli importi in modo (formato) differente a seconda che l'operatore sia in Italia o negli Stati Uniti;
2. Ciascun importo dovrà essere indipendente dal formato utilizzato dall'operatore e, in ogni caso, dovrà denotare lo stesso valore numerico sia che sia stato inserito a New York sia che sia stato inserito a Roma.

In poche parole stiamo separando la rappresentazione del numero dal suo valore.

Un'errata gestione della localizzazione avrebbe delle conseguenze piuttosto serie per la nostra banca e per i suoi clienti. Vediamo un esempio:

Un certo cliente newyorkese X-Customer possiede un conto corrente nella nostra banca

X-Bank con un saldo del conto pari a: 2,000$ (duemila dollari). Per affari X-Customer viene in Italia e una mattina decide di versare sul proprio conto 1,000$ (mille dollari), frutto di un anticipo ricevuto per un contratto concluso. Per fare il versamento si reca alla filiale di Roma della propria banca. Lo sportellista (italiano) della filiale digita l'importo versato da X-Customer come 1.000 (mille dollari in notazione italiana).

Se il software non gestisse correttamente la localizzazione ma utilizzasse sempre la notazione americana accadrebbe che il correntista X-Customer troverebbe sul proprio conto un saldo finale di:

2,000$ + 1.000$ = 2,001.00$ (duemila-uno-punto-zero dollari)

che è ben diverso dai 3,000$ (tremila dollari) attesi.
Il problema sarebbe del tutto inverso in caso di operazione di prelievo.
Capirete bene che la X-Bank non durerebbe un solo giorno con un software del genere.

Vediamo, quindi, una possibile soluzione al problema, tenendo presente che, anche se non è l'unica possibile, almeno è piuttosto semplice da implementare.

Per la nostra soluzione adotteremo le classi DecimalFormat e Number. La classe DecimalFormat estende la classe astratta NumberFormat che “formatta” i numeri decimali.

Essa appartiene al package “java.text” e possiede una vasta gamma di funzionalità progettate per consentire l'analisi e la formattazione di numeri in qualsiasi formato linguistico locale inclusi i Paesi Occidentali, i paesi Arabi e l'India.

Inoltre supporta diversi tipi numerici compresi i numeri interi (154), i numeri in virgola fissa (154,3), i numeri in notazione scientifica (1,543E2), le percentuali (14%) e gli importi in valuta (154,3€).

In questo articolo non analizzeremo in modo approfondito le funzionalità della classe DecimalFormat ma ci limiteremo ad utilizzarla per gestire i valori numerici mediante l'uso della localizzazione.

La classe Number, invece, è una classe astratta appartenente al package “java.lang”. Essa è superclasse delle più note classi Long, BigDecimal, BigInteger, Double, Integer, Short, Float e, nel nostro esempio, viene usata come “contenitore” delle elaborazioni della classe DecimalFormat.

Tutti gli input utente verranno presi da console in formato testo.

Ciascun input corrisponderà ad una stringa numerica e non potranno essere utilizzate più stringhe numeriche sulla stessa riga di input.

Per ciascuna stringa numerica inserita il programma effettuerà tre conversioni in double secondo i seguenti criteri:

1. La prima conversione utilizzerà la localizzazione (il Locale) di default dell'istanza di Java Virtual Machine (JVM) su cui il programma viene eseguito. La Java Virtual Machine stabilisce questo Locale di default al suo avvio in base alla configurazione dell'ambiente host su cui viene eseguita;
2. La seconda conversione utilizzerà il Locale statunitense (Locale.US);
3. La terza conversione utilizzerà il Locale italiano (Locale.ITALY).

Al termine delle conversioni verrà restituito in output un prospetto in cui, per ciascuna delle tre localizzazioni, verrà mostrato il valore double e la sua rappresentazione stringa, entrambi ottenuti mediante l'utilizzo dei metodi della classe DecimalFormat.

A questo punto, diamo una rapida occhiata al sorgente Java del nostro esempio:

```
package                    it.battuello.esercizi

import                     java.io.BufferedR

import                     java.io.IOExcepti

import                     java.io.InputStrea

import                     java.text.Decimal

import                     java.text.ParseEx

import                     java.util.Locale;
```

```
/**
 * @author
 *
 */
public class
    public
        double

    DecimalFormat decimalFormat = new
    DecimalFormat decimalFormatLocalUS = (DecimalFormat) DecimalFormat. getInstance
    DecimalFormat decimalFormatLocalIT = (DecimalFormat) DecimalFormat. getInstance
```

System. *out*

try

InputStreamReader inputStreamReader = **new**

BufferedReader bufferedReader = **new**

String inputString = bufferedReader.readLine();

Number number = decimalFormat.parse(inputString);

numberFromConsole = number.doubleValue();

```
                                                                    /* localizzazione in

Number numberLocalUS = decimalFormatLocalUS.parse(inputString);

numberFromConsoleLocalUS = numberLocalUS.doubleValue();

                                                                    /* localizzazione in

Number numberLocalIT = decimalFormatLocalIT.parse(inputString);

numberFromConsoleLocalIT = numberLocalIT.doubleValue();

System.                                                             out

System.                                                             out

System.                                                             out

System.                                                             out
```

```
            System.                                                    out

            System.                                                    out

            System.                                                    out

            System.                                                    out

        }                                                              catch

            System.                                                    err

        }                                                              catch

            System.                                                    err

        }

    }

}
```

Il codice è molto semplice ed intuitivo.

Tutto avviene nell'unico metodo “main” della classe “LocalizzazioneNumeri”.

La prima operazione che viene eseguita dal codice è l'inizializzazione delle variabili necessarie al nostro programma:

```
    double numberFromConsole = 0, numberFromConsoleLocalUS = 0,
numberFromConsoleLocalIT = 0;

    DecimalFormat decimalFormat = new DecimalFormat(); // Locale preso dal sistema

    DecimalFormat decimalFormatLocalUS = (DecimalFormat) DecimalFormat.getInstance(Loc
ale.
US
);

    DecimalFormat decimalFormatLocalIT = (DecimalFormat) DecimalFormat.getInstance(Loca
le.                                                                    ITA
LY
);
```

E' importante notare le dichiarazioni delle 3 istanze della classe DecimalFormat che, nell'ordine, vengono impostate secondo:

1.

Il Locale di default della macchina:

```
DecimalFormat decimalFormat = new DecimalFormat(); // Locale preso dal sistema
```

2.

il Locale statunitense:

```
DecimalFormat decimalFormatLocalUS = (DecimalFormat) DecimalFormat.getInstance(Locale.
US
);
```

3.

il Locale italiano:

```
DecimalFormat decimalFormatLocalIT = (DecimalFormat) DecimalFormat.getInstance(Locale.I
TALY
);
```

Quindi sulla console Java viene presentato all'utente la richiesta di inserimento dell'input numerico (con o senza separatori delle migliaia e dei decimali):

```
System.out.println("Inerisci un numero.");
```

e, successivamente, viene attesa la digitazione dell'input numerico da parte dell'utente seguita dalla pressione del tasto “Invio”:

```
InputStreamReader inputStreamReader = new InputStreamReader(System.in);

BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String inputString = bufferedReader.readLine();
```

Ad inserimento avvenuto il programma converte la stringa ricevuta in input in tre valori numerici di tipo double utilizzando i tre valori Locale stabiliti in fase di inizializzazione delle istanze della

classe DecimalFormat:

```
/* localizzazione di default del sistema */

Number number = decimalFormat.parse(inputString);

numberFromConsole = number.doubleValue();



/* localizzazione input US */

Number numberLocalUS = decimalFormatLocalUS.parse(inputString);

numberFromConsoleLocalUS = numberLocalUS.doubleValue();



/* localizzazione input ITA */

Number numberLocalIT = decimalFormatLocalIT.parse(inputString);
```

```
numberFromConsoleLocalIT = numberLocalIT.doubleValue();
```

Ovviamente durante l'inserimento o il parsing della stringa di input possono verificarsi delle eccezioni. E' per questo motivo che sono state gestite le eccezioni di tipo java.io.IOException e quelle di tipo java.text.ParseException.

E' bene precisare però che il controllo degli errori di parsing dell'input utente in questo codice è abbastanza leggero in quanto gli unici casi in cui viene sollevata un'eccezione è quando l'input è nullo, l'input inizia con una stringa oppure l'input è interamente stringa.

Invece, se l'input inizia con cifre numeriche e termina con una stringa (o con altri simboli) non viene sollevata alcuna eccezione ma viene considerato come input utente l'insieme delle cifre numeriche che precedono la parte stringa.

Ad esempio, se inseriamo come input la stringa:

“gjhg”

il programma ci risponde con l'eccezione:

'Unparseable number: "gjhg"'

ma se inseriamo la stringa:

“128+50”

l'output restituito diventa:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 128.0

Stringa risultante secondo il locale di sistema: 128

Double ricevuto secondo il locale US: 128.0

Stringa risultante secondo il locale US: 128

Double ricevuto secondo il locale IT: 128.0

Stringa risultante secondo il locale IT: 128

Questo potrebbe essere un problema ma un controllo formale più corretto dell'input utente esula dagli scopi di questo articolo e comunque è abbastanza semplice da realizzare.

Detto questo, quindi, vediamo cosa succede con la localizzazione degli input numerici.

Supponiamo di inserire il valore:

1.250

(mille-duecento-cinquanta) in notazione italiana.

L'applicazione ci ritorna questo output:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 1250.0

Stringa risultante secondo il locale di sistema: 1.250

Double ricevuto secondo il locale US: 1.25

Stringa risultante secondo il locale US: 1.25

Double ricevuto secondo il locale IT: 1250.0

Stringa risultante secondo il locale IT: 1.250

Esaminiamolo in dettaglio partendo dall'alto.

La prima e la seconda riga ci dicono quali sono i separatori delle migliaia e dei decimali nelle lingue italiana e statunitense.

La terza e la quarta riga ci danno il valore double e la sua rappresentazione stringa ottenuti parsando la stringa di input con il Locale di default del sistema.

Il fatto che il valore numerico double e la sua riconversione in stringa siano rimasti inalterati, rispetto all'input inserito, dimostra che il Locale di default della mia JVM è correttamente settato su Locale.ITALY.

La quinta e la sesta riga, invece, sono più interessanti, in quanto mostrano la conversione dell'input utente secondo il Locale americano. Come si vede, sia il valore numerico double che la sua riconversione in stringa sono stati interpretati come 1.25 (ovvero 1,25 in italiano), dato che il separatore dei decimali per il Locale.US è il punto.

Infine, la settima ed ottava riga rappresentano una conversione “forzata” mediante l'uso del Locale italiano e, quindi, essendo quest'ultimo uguale a quello di default del sistema, l'output è uguale a quello della terza e quarta riga.

Ora vediamo cosa accade inserendo in input il valore:

1,250

(mille-duecento-cinquanta) in notazione americana.

L'applicazione restituisce il seguente output:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 1.25

Stringa risultante secondo il locale di sistema: 1,25

Double ricevuto secondo il locale US: 1250.0

Stringa risultante secondo il locale US: 1,250

Double ricevuto secondo il locale IT: 1.25

Stringa risultante secondo il locale IT: 1,25

Diversamente dal caso precedente, il Locale di default del sistema (terza e quarta riga di output) e quello italiano (settima ed ottava riga) interpretano il numero inserito come 1,25 (in notazione italiana) mentre solo il Locale americano lo interpreta correttamente come 1250.

Si osservi che il problema evidenziato nei due casi precedenti sussiste in quanto è stato utilizzato il separatore delle migliaia. Difatti, se l'utente avesse inserito il valore:

1250

l'output del nostro programma sarebbe stato:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 1250.0

Stringa risultante secondo il locale di sistema: 1.250

Double ricevuto secondo il locale US: 1250.0

Stringa risultante secondo il locale US: 1,250

Double ricevuto secondo il locale IT: 1250.0

Stringa risultante secondo il locale IT: 1.250

e tutti gli importi sarebbero stati corretti per i diversi Locale utilizzati.

Tuttavia, mentre in generale è “semplice“ rinunciare all'uso dei separatori delle migliaia invece, per alcuni tipi di applicazioni come quelle contabili, è praticamente impossibile fare a meno dei separatori decimali.

Vediamo, quindi, cosa accade se inseriamo
mille-duecento-cinquanta-virgola-quattrocento-ottanta-sei (in notazione italiana):

1250,486

Il nostro programma restituisce il seguente output:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 1250.486

Stringa risultante secondo il locale di sistema: 1.250,486

Double ricevuto secondo il locale US: 1250486.0

Stringa risultante secondo il locale US: 1,250,486

Double ricevuto secondo il locale IT: 1250.486

Stringa risultante secondo il locale IT: 1.250,486

Come si vede nelle righe 3 e 4 (Locale di default) e nelle righe 7 e 8 (Locale italiano) il numero viene correttamente rappresentato sia nel suo valore double che nella sua rappresentazione stringa, mentre nelle righe 5 e 6 (Locale americano) il valore double e la sua rappresentazione stringa sono diventati un-milione-duecento-cinquanta-mila-quattrocento-ottanta-sei che è ben diverso dal valore inserito.

Inoltre, se inseriamo lo stesso valore adoperando il separatore delle migliaia:

1.250,486

l'output del nostro programma risulta essere:

IT: (migliaia - decimali) = ('.' - ',')

US: (migliaia - decimali) = (',' - '.')

Double ricevuto secondo locale di sistema: 1250.486

Stringa risultante secondo il locale di sistema: 1.250,486

Double ricevuto secondo il locale US: 1.25

Stringa risultante secondo il locale US: 1.25

Double ricevuto secondo il locale IT: 1250.486

Stringa risultante secondo il locale IT: 1.250,486

che nelle righe 5 e 6, relative al Locale americano, fornisce un valore double ed una rappresentazione stringa diversa da quella dell'input precedente.

Riassumendo quando si progetta un'applicazione Java che gestisce degli input numerici bisogna tenere in considerazione il Locale dell'utente che effettuerà gli inserimenti.

In software stand-alone è relativamente semplice gestire la localizzazione. Tipicamente è sufficiente usare il Locale di default della macchina su cui gira l'applicazione oppure prevedere una sezione di configurazione in cui l'operatore imposta i dati relativi alla propria lingua o nazione.

In applicazioni client-server, come quelle bancarie, invece, il server opera secondo un proprio Locale mentre ciascun client adotta una propria configurazione non necessariamente uguale a quella del server.

La localizzazione interessa anche il simbolo di valuta, il fuso orario, il formato delle date e la codifica utilizzata per i caratteri.

Una trattazione completa di questi argomenti esula dagli scopi di questo articolo che, al contrario, vuole essere solo un'introduzione per coloro che si avvicinano a questo tipo di problematiche.

Per applicazioni web esistono diversi framework che forniscono supporto per l'internazionalizzazione/localizzazione delle applicazioni web, come JSF e Struts. Questi framework mediante le impostazioni di internazionalizzazione consentono anche di tradurre i contenuti web da una lingua ad un'altra (ovviamente a patto che ciò sia stato opportunamente previsto e configurato).