



Alma Mater Studiorum-Università di Bologna Scuola di Ingegneria

Supporto all'internazionalizzazione Culture locali e formattatori

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Proff. ENRICO DENTI & GABRIELE ZANNONI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)



CONVENZIONI DI FORMATTAZIONE E CULTURE LOCALI

- Le convenzioni per **date, orari, numeri, valute** non sono universali: *ogni paese, ogni cultura ha le proprie*
 - nomi dei giorni della settimana, nomi dei mesi
 - ordine in cui compaiono gli elementi (gg/mm/aa, mm/gg/aa...)
 - orari su 12 o 24 ore, mezzogiorno/mezzanotte
 - separatori ammessi per le migliaia e le parti decimali
 - simboli di valuta, posizione del simbolo (prima/dopo il valore)
 - ... e molto altro
- Appositi **formattatori** (o metodi di formattazione) provvedono a operare in base alla **cultura locale** richiesta
 - insiemi di regole per **stampa** (output) e **parsing** (input) di stringhe

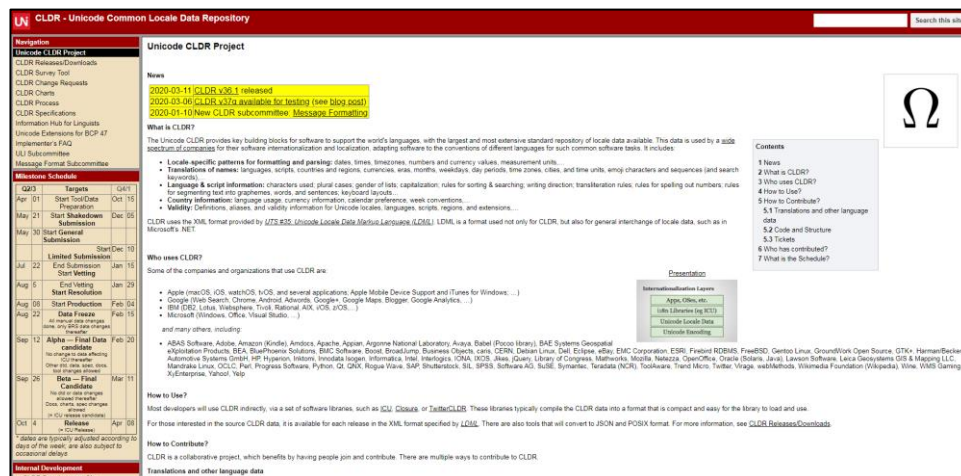
DATABASE DI CULTURE E FORMATI

- Appositi *database* raccolgono tali convenzioni

Java

Java adotta, da Java 9, il *database internazionale Unicode CLDR (Common Locale Data Repository)*

- fino a Java 8 si usava un *proprio database interno*, chiamato «JRE», simile ma *non identico* a CLDR



The screenshot shows the Unicode CLDR website. On the left, there's a sidebar with navigation links like 'Unicode CLDR Project', 'CLDR Releases/Downloads', 'CLDR Survey Tool', etc. The main content area has a 'News' section with a highlighted entry: '2020-03-11 CLDR v39.1 released'. Below this, there's a 'What is CLDR?' section explaining the project's purpose. On the right, there's a 'Contents' table of contents and a 'Presentation' diagram showing the internal structure of the CLDR data.

- in particolare, occhio alla posizione del simbolo € nei prezzi italiani..!

C#

C# adotta invece il *Microsoft Language Code ID (MS-LCID)*

Scala

Scala e Kotlin si appoggiano alla soluzione Java sottostante

Kotlin

- almeno per ora.. (Kotlin ha in sviluppo una sua libreria..)

Infrastruttura Java

(Scala, Kotlin)

LA CULTURA LOCALE IN Java

- In Java, la classe `java.util.Locale` rappresenta appunto il concetto di *cultura locale*

Java

 - insiemi di regole per la *generazione di stringhe* (per la stampa)...
 - ...e anche, a rovescio, per il *parsing* di numeri, valute, date, orari
- Una *cultura locale* è composta da *lingua* e *paese*
 - MOTIVO: una lingua è tipicamente parlata in più paesi e regioni, che spesso adottano *convenzioni diverse*
 - la *lingua* è espressa da una sigla *minuscola di due lettere*
 - il *paese* è espresso da una sigla *MAIUSCOLA di due lettere*
 - ad es. l'italiano non è parlato solo in Italia, ma anche in Svizzera: la valuta e le regole di formattazione dei numeri *non sono le stesse*
→ la cultura locale «Italia» è `it_IT`, la «Svizzera italiana» è `it_CH`
 - idem per l'inglese: `en_GB`, `en_US`, `en_CANADA`,...

LA CULTURA LOCALE IN Java

- Alcune costanti "scorciatoia" abbreviano i casi più frequenti:

Modifier and Type	Field	Description
static <code>Locale</code>	<code>CANADA</code>	Useful constant for country.
static <code>Locale</code>	<code>CANADA_FRENCH</code>	Useful constant for country.
static <code>Locale</code>	<code>CHINA</code>	Useful constant for country.
static <code>Locale</code>	<code>CHINESE</code>	Useful constant for language.
static <code>Locale</code>	<code>ENGLISH</code>	Useful constant for language.
static <code>Locale</code>	<code>FRANCE</code>	Useful constant for country.
static <code>Locale</code>	<code>FRENCH</code>	Useful constant for language.
static <code>Locale</code>	<code>GERMAN</code>	Useful constant for language.
static <code>Locale</code>	<code>GERMANY</code>	Useful constant for country.
static <code>Locale</code>	<code>ITALIAN</code>	Useful constant for language.
static <code>Locale</code>	<code>ITALY</code>	Useful constant for country.
static <code>Locale</code>	<code>JAPAN</code>	Useful constant for country.
static <code>Locale</code>	<code>JAPANESE</code>	Useful constant for language.
static <code>Locale</code>	<code>KOREA</code>	Useful constant for country.
static <code>Locale</code>	<code>KOREAN</code>	Useful constant for language.
static <code>Locale</code>	<code>PRC</code>	Useful constant for country.
static <code>char</code>	<code>PRIVATE_USE_EXTENSION</code>	The key for the private use extension ('x').
static <code>Locale</code>	<code>ROOT</code>	Useful constant for the root locale.
static <code>Locale</code>	<code>SIMPLIFIED_CHINESE</code>	Useful constant for language.
static <code>Locale</code>	<code>TAIWAN</code>	Useful constant for country.
static <code>Locale</code>	<code>TRADITIONAL_CHINESE</code>	Useful constant for language.
static <code>Locale</code>	<code>UK</code>	Useful constant for country.
static <code>char</code>	<code>UNICODE_LOCALE_EXTENSION</code>	The key for Unicode locale extension ('u').
static <code>Locale</code>	<code>US</code>	Useful constant for country.

LA CULTURA LOCALE IN Java

- Per tutti gli altri casi, i costruttori di uso generale di **Locale** consentono di configurare la localizzazione richiesta

Constructors	
Constructor	Description
<code>Locale(String language)</code>	Construct a locale from a language code.
<code>Locale(String language, String country)</code>	Construct a locale from language and country.
<code>Locale(String language, String country, String variant)</code>	Construct a locale from language, country and variant.

- Se non si specifica nulla, i formattatori adottano il *default*, ottenibile dalla funzione statica:
 - **Locale.getDefault**
- Per conoscere tutte le culture locali disponibili basta invocare la funzione statica :
 - **Locale.getAvailableLocales**



ESEMPIO

Java

```
System.out.println(  
    "Cultura locale predefinita: " + Locale.getDefault());
```

Output:

```
Cultura locale predefinita: it_IT
```

sigla cultura locale in uso



FAMIGLIE DI FORMATTATORI

- Ci sono due grandi famiglie di formattatori:
 - formattatori *numerici*
 - formattatori per *date e orari*
- Formattatori numerici (`java.text.NumberFormat`)
 - formattazione di numeri
 - formattazione di percentuali
 - formattazione di prezzi e valori espressi in valuta
- Formattatori per date e orari (`java.time.format.DateTimeFormatter`)
 - formattazione di una data (in 4 formati: breve, medio, lungo, full)
 - formattazione di un orario (in 2 formati: breve, medio)
 - formattazione di una data con orario (in 4x2 formati possibili)

FAMIGLIE DI FORMATTATORI

Java

- Elementi comuni...
 - entrambe le famiglie adottano il *pattern factory*:
 - **NumberFormat** espone metodi **get** (...)** distinti per *numeri*, *percentuali* e *valute*
 - **DateTimeFormatter** espone metodi **of** (...)**
- .. ed elementi non del tutto comuni
 - entrambe le famiglie espongono un metodo **format** *da usare però in modo opposto*
 - nei formattatori numerici, **format** è dichiarato in **NumberFormat**
→ il pattern d'uso è `formatter.format(valore)`
 - nei formattatori per date e orari, **format** è invece dichiarato da **LocalDate/-Time**
→ il pattern d'uso è `valore.format(formatter)`

FORMATTATORI NUMERICI

- I formattatori numerici derivano tutti dalla classe **java.text.NumberFormat**, *che funge anche da factory*
 - espone tre metodi **get** (...)** distinti per *numeri*, *percentuali* e *valute*

Java

```
double x = 43.12345678, y = 0.7, z = 13456.78;
```

```
NumberFormat fN = NumberFormat.getNumberInstance();
```

Formattatore numeri

```
NumberFormat fP = NumberFormat.getPercentInstance();
```

Formattatore percentuali

```
NumberFormat fV = NumberFormat.getCurrencyInstance();
```

Formattatore valute



ESEMPIO: FORMATTAZIONE DI NUMERI

```
double x = 43.12345678, y = 0.7, z = 13456.78;  
NumberFormat fN = NumberFormat.getNumberInstance();  
fN.setMaximumFractionDigits(2);  
System.out.println( fN.format(x) );  
System.out.println( fN.format(y) );  
System.out.println( fN.format(z) );
```

Java

Configurazione

Output:

```
43,12  
0,7  
13.456,78
```

MAX due cifre decimali,
come richiesto

Il default è `Locale.ITALY`:
usa il punto come
separatore delle migliaia
e
la virgola come separatore
decimale

ESEMPIO: FORMATTAZIONE DI NUMERI

```
double x = 43.12345678, y = 0.7, z = 13456.78;  
NumberFormat fN = NumberFormat.getNumberInstance(  
    Locale.CANADA);  
fN.setMaximumFractionDigits(2);  
System.out.println( fN.format(x) );  
System.out.println( fN.format(y) );  
System.out.println( fN.format(z) );
```

Java

Cambiando
cultura locale..

Output:

```
43.12  
0.7  
13,456.78
```

.. cambia il risultato: il Canada
ha convenzioni opposte!



ESEMPIO: FORMATTAZIONE DI NUMERI

```
double x = 43.12345678, y = 0.7, z = 13456.78;  
NumberFormat fN = NumberFormat.getNumberInstance(  
    Locale.CANADA_FRENCH);  
fN.setMaximumFractionDigits(2);  
System.out.println( fN.format(x) );  
System.out.println( fN.format(y) );  
System.out.println( fN.format(z) );
```

Java

.. e cambiando
ancora, si replica!

Output:

```
43,12  
0,7  
13 456,78
```

Però, la parte francofona adotta
convenzioni ancora diverse!

*Notare lo spazio (hard) come
separatore delle migliaia!*

*Non è lo spazio standard:
è un carattere diverso*

FORMATTAZIONE DI PERCENTUALI

```
double p = 0.4312, q = 0.7, r = 1.2345678;
```

Java

```
NumberFormat fP = NumberFormat.getPercentInstance(...);
```

```
fP.setMaximumFractionDigits(2);
```

```
System.out.println( fP.format(p) );
```

```
System.out.println( fP.format(q) );
```

```
System.out.println( fP.format(r) );
```

Formattatore percentuali
configurato su due cifre
decimali

Default (Italia):

43,12%

70%

123,46%

Canada French:

43,12 %

70 %

123,46 %

UK:

43.12%

70%

123.46%


*Notare lo spazio (hard) prima
del simbolo di percentuale !*

FORMATTAZIONE DI VALUTE

- La formattazione di *valori valuta* è analoga...
- .. ma nel caso ITALIA costituisce un punto delicato, perché *il passaggio al database CLDR ha cambiato la convenzione sulla posizione del simbolo di valuta, €*
 - fino a Java 8, il simbolo Euro era posizionato *prima* dell'importo
 - da Java 9, il simbolo Euro viene posizionato *dopo* l'importo, perché CLDR adotta una (strana) diversa convenzione al riguardo
- Rompendo la retrocompatibilità, ciò ha causato problemi
 - formattazioni diverse eseguendo in Java9+ codice scritto per Java8
 - peggio: *il parsing di stringhe col simbolo € davanti all'importo fallisce in Java9+ usando il formattatore standard italiano → ATTENZIONE*

CLDR (cldr.unicode.org)

https://www.unicode.org/cldr/charts/latest/by_type/numbers.number_formatting_patterns.html



CLDR Charts

Home | Site Map | Search

CLDR v36

By-Type Chart: Numbers: Number Formatting Patterns

2019-10-02

Index

Core Data:	Alphabetic Information Main Exemplars Punctuation Exemplars Index Exemplars Numbering Systems
Locale Display Names:	Locale Name Patterns Languages (A-D) Languages (E-J) Languages (K-N) Languages (O-S) Languages (T-Z) Scripts Geographic Regions Territories (North America) Territories (South America) Territories (Africa) Territories (Europe) Territories (Asia) Territories (Oceania) Locale Variants Keys
Date & Time:	Fields Gregorian Generic Buddhist Chinese Coptic Dangi Ethiopic Ethiopic-Amete-Alem Hebrew Indian Islamic Japanese Persian Minguo
Timezones:	Timezone Display Patterns North America South America Africa Europe Russia Western Asia Central Asia Eastern Asia Southern Asia Southeast Asia Australasia Antarctica Oceania Unknown Region Overrides
Numbers:	Symbols Minimal Pairs Number Formatting Patterns Compact Decimal Formatting
Currencies:	North America (C) South America (C) Northern/Western Europe Southern/Eastern Europe Northern Africa Western Africa Middle Africa Eastern Africa Southern Africa Western Asia (C) Central Asia (C) Eastern Asia (C) Southern Asia (C) Southeast Asia (C) Oceania (C) Unknown Region (C)
Units:	Measurement Systems Duration Graphics Length Area Volume Speed and Acceleration Mass and Weight Energy and Power Electrical and Frequency Weather Digital Coordinates Other Units Compound Units
Characters:	Category Smileys & Emotion People & Body Animals & Nature Food & Drink Travel & Places Activities Objects Symbols2 Flags Component Typography
Miscellaneous:	Displaying Lists

Standard Patterns

standard-decimal	English: <#,###>
standard-currency	English: <¤#,###0.00>

#,###0.00¤

#,###0.00¤

·bn· ·ccp·

·asa· ·ast· ·az· ·az_Cyrl· ·bas· ·be· ·br· ·bs· ·bs_Cyrl· ·ca· ·ce· ·ewo· ·ff· ·fi· ·fo· ·fr· ·gl· ·gsw· ·hr· ·hsb· ·hu· ·hy· ·is· **·it·** ·ka· ·prg· ·ps· ·pt_PT· ·rm· ·ro· ·ru· ·sah· ·se· ·sk· ·sl· ·smn· ·sq· ·s

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

17

FORMATTAZIONE DI VALUTE

```
double x = 1243.5678;
```

```
NumberFormat fV = NumberFormat.getCurrencyInstance(...);
```

```
System.out.println( fV.format(x) );
```

Java

Formattatore valute

Default (Italia):

1.243,57 €

Canada:

\$1,243.57

UK:

£1,243.57

Italia Java 8:

€ 1.243,57

Canada French:

1 243,57 \$

Notare lo spazio (hard)

NB: nelle valute non serve specificare il numero di cifre decimali: fanno già parte della convenzione, con relativi arrotondamenti



PARSING DI STRINGHE NUMERICHE

- Sebbene nascano per «formattare» (in output), i formattatori supportano anche *l'operazione duale*, detta *parsing* (in input)
- Il metodo **parse** converte una **stringa numerica**
 - *correttamente formattata* secondo le regole di una *cultura locale*
 - rispettivamente come valore numerico, percentuale o valutain un **valore numerico** di classe **Number**
 - facilmente estraibile come **double**, **float** o **int** a piacere
- MA il suo uso richiede la gestione degli errori (eccezioni), poiché se la stringa è scorretta il formattatore "si arrabbia"
 - possiamo però fare qualche esperimento interattivo con **jshell**, in quanto la console interattiva *gestisce da sola gli errori* 😊

PARSING DI STRINGHE NUMERICHE: ESEMPI

```
NumberFormat fV =  
    NumberFormat.getCurrencyInstance(Locale.US) ;  
System.out.println(fV.parse("$123.56")) ;
```

Java

Formattatore USA

```
jshell> System.out.println(fv.parse("$123.56"))  
123.56
```

Il metodo **parse** restituisce un oggetto **Number**, che si può poi convertire in **double**, **float** o **int** con gli opportune arrotondamenti:

```
jshell> Number n = fv.parse("$123.456789987654321")  
n ==> 123.45678998765432  
  
jshell> double d = n.doubleValue()  
d ==> 123.45678998765432  
  
jshell> float f = n.floatValue()  
f ==> 123.45679
```

PARSING DI STRINGHE NUMERICHE: ESEMPI

```
NumberFormat fP =  
    NumberFormat.getPercentInstance(Locale.US);  
fP.setMinimumFractionDigits(2);  
System.out.println(fP.parse("72.35%"));
```

Java

Formattatore USA

```
jshell> fP.parse("72.35%")  
$12 ==> 0.7234999999999999
```

Da notare che se *invece del locale USA si usasse il locale ITALY*,
il comportamento sarebbe diverso e *rischioso*:

```
jshell> fP.parse("72.35%")  
$14 ==> 72.35
```

Formattatore IT
OCCHIO!

```
jshell> fP.parse("72.35%")  
$15 ==> 0.7234999999999999
```

Formattatore IT
ok

PERICOLO! La prima stringa col "." viene presa *solo parzialmente*, di fatto *ignorando il simbolo di % (ORRORE!)*, restituendo un risultato assurdo!

PARSING DI STRINGHE NUMERICHE: ESEMPI

Ora proviamo con il locale Canada francese:

Java

(notare la virgola e l'uso dello spazio prima del simbolo di percentuale)

```
jshell> NumberFormat fP = NumberFormat.getPercentInstance(Locale.CANADA_FRENCH)
fP.setMinimumFractionDigits(2)

jshell> fP.format(0.7235)
$12 ==> "72,35 %"
```

Facendo il parsing senza spazio, BOOM!

No, non è uno spazio normale!

```
jshell> fP.parse("72,35 %")
java.text.ParseException thrown: Unparseable number: "72,35 %"
    at NumberFormat.parse (NumberFormat.java:431)
    at (#13:1)

jshell> fP.parse("72,35%")
java.text.ParseException thrown: Unparseable number: "72,35%"
    at NumberFormat.parse (NumberFormat.java:431)
    at (#14:1)

jshell> fP.parse("72,35\u00A0%")
$15 ==> 0.7234999999999999
```

OK solo con l' hard space

Perché funzioni, nella stringa servono la virgola e il *non breakable space*

CI PENSEREMO...

Java

- La gestione efficace dei vari metodi **parse**, con annessi e connessi, sarà affrontata a tempo debito
 - ora come ora, tentando di usare **parse** in un vostro programma **otterrete errore di compilazione** perché in Java la gestione di quegli errori è obbligatoria e voi ancora non sapete farla!
 - ergo, se volete fare (piccoli) esperimenti, usate **jshell** ...
 - .. ma evitando il simbolo € perché non è ASCII (scegliete \$ o £)
 - ...o, meglio, abbiate pazienza e aspettate qualche settimana ☺
- Per ora, ricordate che:
 - formattare valori numerici in stringa è *facile e sicuro*
 - **l'inverso invece non è immediato e può dar luogo a errori**

FORMATTATORI PER DATE E ORARI

Java

- Finora abbiamo visto i formattatori numerici
 - classe `java.text.NumberFormat`
 - metodi factory `get** (...)` distinti per *numeri*, *percentuali* e *valute*
 - metodo `format` dichiarato in `NumberFormat`
→ pattern d'uso: `formatter.format(valore)`
- I formattatori per date e orari
 - sono definiti in `java.time.format.DateTimeFormatter`
 - espongono metodi factory di nome `of** (...)`
 - **invertono il pattern d'uso: `dataora.format(formatter)`**
perché il metodo `format` è ora dichiarato in `LocalDate/-Time`
e quindi il formatter diventa *l'argomento* del metodo (anziché il target)

FORMATTATORI PER DATE

Java

- I formattatori per date derivano dalla classe **`DateTimeFormatter`**, *che funge anche da factory*
 - metodi **`of** (...)`** distinti per `LocalDate`, `LocalDateTime`
 - ulteriormente **configurabili secondo diversi formati di uscita** tramite la classe ausiliaria **`java.time.format.FormatStyle`**
- **Formati di stile:**
 - **SHORT** *gg/mm/aa* 18/03/15
 - **MEDIUM** *gg siglamese aaaa* 18 mar 2015
 - **LONG** *gg mese aaaa* 18 marzo 2015
 - **FULL** *giorno gg mese aaaa* mercoledì 18 marzo 2015
 - tutti naturalmente *tarati su una specifica cultura locale* per quanto attiene all'ordine, al nome, al formato, ai separatori delle singole voci



ESEMPIO: FORMATTAZIONE DI `LocalDate`

Sperimentiamo i formattatori nei quattro diversi stili:

Java

```
DateTimeFormatter formatterShort =  
    DateTimeFormatter.ofLocalizedDate (FormatStyle.SHORT) ;  
DateTimeFormatter formatterMedium =  
    DateTimeFormatter.ofLocalizedDate (FormatStyle.MEDIUM) ;  
DateTimeFormatter formatterLong =  
    DateTimeFormatter.ofLocalizedDate (FormatStyle.LONG) ;  
DateTimeFormatter formatterFull =  
    DateTimeFormatter.ofLocalizedDate (FormatStyle.FULL) ;  
  
LocalDate d = LocalDate.now() ;  
  
System.out.println(d.format(formatterShort)) ;  
System.out.println(d.format(formatterMedium)) ;  
System.out.println(d.format(formatterLong)) ;  
System.out.println(d.format(formatterFull)) ;
```

ESEMPIO: FORMATTAZIONE DI `LocalDate`

Output con cultura locale italiana (default):

04/03/20	Short	Java
4 mar 2020	Medium	
4 marzo 2020	Long	
mercoledì 4 marzo 2020	Full	

Osserva: le convenzioni italiane

- nello short, formattano l'anno su due cifre
- nel full, non inseriscono virgole o altri separatori

Non è così in altre culture!



FORMATTAZIONE DI `LocalDate` CON CULTURE LOCALI DIVERSE

- A differenza dei formattatori numerici, in cui si passa il `Locale` desiderato come argomento al metodo factory:

Java

```
NumberFormat.getNumberInstance(Locale.CANADA)
```

- nei formattatori per date e orari si cambia approccio:
 - il metodo factory in sé non ha argomenti (imposta il default)
 - ma il risultato può essere **specializzato con il metodo `withLocale`**

```
DateTimeFormatter.ofLocalizedDate(...)  
    .withLocale(Locale.CANADA)
```

È un (altro) esempio di *fluent interface*

ESEMPIO: FORMATTAZIONE DI `LocalDate`

Output con cultura locale italiana (default):

04/03/20	Short	Java
4 mar 2020	Medium	
4 marzo 2020	Long	
mercoledì 4 marzo 2020	Full	

Output con cultura locale inglese UK e USA:

04/03/2020	Short	Java
4 Mar 2020	Medium	
4 March 2020	Long	
Wednesday, 4 March 2020	Full	

3/4/20	Short	Java
Mar 4, 2020	Medium	
March 4, 2020	Long	
Wednesday, March 4, 2020	Full	



ESEMPIO: FORMATTAZIONE DI `LocalDate`

Output con cultura locale italiana (default):

```
04/03/20  
4 mar 2020  
4 marzo 2020  
mercoledì 4 marzo 2020
```

Java

Output con cultura locale francese e franco-canadese

```
04/03/2020  
4 mars 2020  
4 mars 2020  
mercredi 4 mars 2020
```

Java

```
20-03-04  
4 mars 2020  
4 mars 2020  
mercredi 4 mars 2020
```

In febbraio e altri mesi dal nome più lungo:

Medium → *févr.*

Long → *février*



- # Java

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

FORMATTORI PER ORARI LOCALI

Esempi: cultura locale italiana, UK, USA

Java

18:37
18:37:41

Short

Medium

Italia

UK

6:37 PM
6:37:41 PM

Short

Medium

USA

6:37 p.m.
6:37:41 p.m.

Canada

18 h 37
18 h 37 min 29 s

Canada
francese

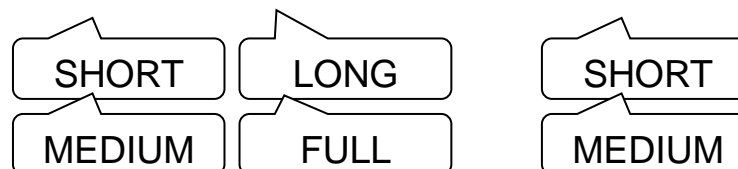
Cultura canadese fino a Java 8

FORMATTATORI PER DATE/ORARI LOCALI

- I formattatori omnnicomprensivi per date/orari sono analoghi
 - metodi **ofLocalizedDateTime**
 - doppia specifica di stili: una per le data, una per l'orario
 - data: ammessi tutti i quattro formati di stile
 - orario: possibili solo i due formati SHORT e MEDIUM
- Totale: 8 possibili combinazioni

Java

```
DateTimeFormatter.ofLocalizedDateTime (  
    dateStyle, timeStyle) ;
```



FORMATTORI PER DATE/ORARI LOCALI

Output con cultura locale italiana (default):

Java

altri esempi con altri mesi & anni

14/02/19, 19:27

14 feb 2019, 19:27

14 febbraio 2019, 19:27

giovedì 14 febbraio 2019, 19:27

14/02/19, 19:27:13

14 feb 2019, 19:27:13

14 febbraio 2019, 19:27:13

giovedì 14 febbraio 2019, 19:27:13

Tutte le 8
combinazioni

Come sarà negli altri paesi..? ☺



FORMATTORI PER DATE/ORARI ASSOLUTI

- I formattatori per date/orari gestiscono anche *orari assoluti*
 - RICORDA: un orario assoluto è tale *perché contiene una specifica* che lo àncora a un *ben preciso punto* sul globo terrestre
- Un orario assoluto può essere espresso: Java
 - specificando un fuso orario: istanze di **ZonedDateTime**
 - specificando un offset da GMT: istanze di **OffsetDateTime**
 - NB: non esiste un'analogia classe "**ZonedTime**" senza data
- **Il formattatore si adegua includendo la specifica del luogo**
 - specificando il fuso orario: tutti e quattro i possibili formati
 - specificando l'offset da GMT: solo i due formati SHORT/MEDIUM



FORMATTORI PER DATE/ORARI ASSOLUTI

- Lo schema generale resta identico:

Java

```
DateTimeFormatter.ofLocalizedTime(  
    stampa comunque solo l' orario    dateStyle, timeStyle);
```

```
DateTimeFormatter.ofLocalizedDateTime(  
    stampa data e orario    dateStyle, timeStyle);
```

- ma

- nel caso di **ZonedDateTime** ci sono 16 possibili combinazioni

tutti e quattro

tutti e quattro

- nel caso di **OffsetDateTime** ci sono solo 8 combinazioni

tutti e quattro

solo MEDIUM
e SHORT



FORMATTORI PER DATE/ORARI ASSOLUTI

```
ZonedDateTime zdt = ZonedDateTime.now();
```

Java

```
19:27  
19:27:13  
19:27:13 CET  
19:27:13 Ora standard dell'Europa centrale
```

con ofLocalizedTime
(solo orario)

Italia

```
19:27  
19:27:13  
19:27:13 CET  
19:27:13 Central European Standard Time
```

UK

```
19 h 27  
19 h 27 min 13 s  
19 h 27 min 13 s CET  
19 h 27 min 13 s heure normale d'Europe centrale
```

Canada
francese

```
19:27  
19:27:13  
19:27:13 CET  
19:27:13 heure normale d'Europe centrale
```

Francia

FORMATTATORI ISO

- Per leggere/scrivere date e orari in *formato interoperabile*
 - quindi, per interazioni machine-to-machine, *non verso umani*
- sono forniti un insieme di ***formatter standard (ISO)***
 - ce ne sono più di una dozzina!

Java

Predefined Formatters		
Formatter	Description	Example
ofLocalizedDate(dateStyle)	Formatter with date style from the locale	'2011-12-03'
ofLocalizedTime(timeStyle)	Formatter with time style from the locale	'10:15:30'
ofLocalizedDateTime(dateTimeStyle)	Formatter with a style for date and time from the locale	'3 Jun 2008 11:05:30'
ofLocalizedDateTime(dateStyle,timeStyle)	Formatter with date and time styles from the locale	'3 Jun 2008 11:05'
BASIC_ISO_DATE	Basic ISO date	'20111203'
ISO_LOCAL_DATE	ISO Local Date	'2011-12-03'
ISO_OFFSET_DATE	ISO Date with offset	'2011-12-03+01:00'
ISO_DATE	ISO Date with or without offset	'2011-12-03+01:00'; '2011-12-03'
ISO_LOCAL_TIME	Time without offset	'10:15:30'
ISO_OFFSET_TIME	Time with offset	'10:15:30+01:00'
ISO_TIME	Time with or without offset	'10:15:30+01:00'; '10:15:30'
ISO_LOCAL_DATE_TIME	ISO Local Date and Time	'2011-12-03T10:15:30'
ISO_OFFSET_DATE_TIME	Date Time with Offset	'2011-12-03T10:15:30+01:00'
ISO_ZONED_DATE_TIME	Zoned Date Time	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_DATE_TIME	Date and time with ZoneId	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_ORDINAL_DATE	Year and day of year	'2012-337'
ISO_WEEK_DATE	Year and Week	'2012-W48-6'
ISO_INSTANT	Date and Time of an Instant	'2011-12-03T10:15:30Z'
RFC_1123_DATE_TIME	RFC 1123 / RFC 822	'Tue, 3 Jun 2008 11:05:30 GMT'



FORMATTATORI ISO

```
OffsetDateTime odt = OffsetDateTime.now();
```

Java

```
ZonedDateTime zdt = ZonedDateTime.now();
```

17:04:44.9189596+01:00

con
ISO OFFSET TIME

2019-02-15T17:10:34.0637975

con
ISO LOCAL DATE TIME

```
LocalDate d = LocalDate.now();
```

20190215

con BASIC_ISO_DATE

FORMATTATORI PERSONALIZZATI

Java

- Non vi piacciono/bastano i formattatori standard?
 - vorreste il simbolo di € davanti invece che in fondo nelle valute?
 - vorreste magari poter fare il parsing del solo nome del mese (es. "Maggio") nella lingua della cultura locale prescelta, ottenendone il corrispondente indice (es. 5)?
- Potete farvi il vostro *formattatore personalizzato*
 - per numeri e valute, tramite la classe **DecimalFormat**, il cui costruttore accetta una *stringa di formato* che specifica il pattern richiesto
DecimalFormat formatter = new **DecimalFormat**(pattern) ;
 - per date/orari, tramite il metodo factory generale
DateTimeFormatter.ofPattern(pattern) ;
 - ovviamente, a patto di studiarsi i (molti) simboli del *pattern*... ☺

FORMATTORI PERSONALIZZATI

Esempio: valute con € davanti

- Da Java 9, l'adozione del database CLDR ha spostato il simbolo dell'Euro *dopo* il valore nel `Locale.ITALY`
 - ciò può essere sgradevole esteticamente, ma soprattutto creare *problemi in fase di parsing* di prezzi col simbolo € davanti
- Si può ovviare costruendosi un **formattatore personalizzato**
 - si sfrutta la classe specifica `java.text.DecimalFormat`
`DecimalFormat formatter = new DecimalFormat(pattern) ;`
 - si specifica come pattern di formato "`¤ ###0.##`"
 - il simbolo di valuta è rappresentato dal carattere jolly "`¤`",
la generica cifra numerica dal carattere jolly "`#`"
 - questo pattern specifica uno spazio dopo il simbolo di valuta,
separatori delle migliaia ogni tre cifre, uno zero davanti per i numeri con
parte intera nulla e l'uso di due cifre decimali

FORMATTORI PERSONALIZZATI

Esempio: valute con € davanti

- Formattatore personalizzato:

Java

```
DecimalFormat f = new DecimalFormat("¤ ###0.##");
```

- Esempi di formattazioni:

```
System.out.println(f.format(1234.567));
```

```
System.out.println(f.format(-1234.567));
```

```
System.out.println(f.format(0.567));
```

```
System.out.println(f.format(12345678.91234));
```

€ 1.234,57

-€ 1.234,57

€ 0,57

€ 12.345.678,91

Arrotondamento automatico alla seconda cifra decimale 😊

Però, i valori negativi con "-" davanti non sono splendidi... 😞

FORMATTORI PERSONALIZZATI

Esempio: valute con € davanti

- Si può perfezionare dando una doppia specifica:

Java

- la prima valida per valori positivi
- la seconda valida per valori negativi

```
var f = new DecimalFormat("#,##0.##;# -#,##0.##");
```

Specifica per valori negativi

- Esempi di formattazioni:

```
System.out.println(f.format(1234.567));
```

```
System.out.println(f.format(-1234.567));
```

```
System.out.println(f.format(0.567));
```

```
System.out.println(f.format(12345678.91234));
```

Due spazi per i positivi,
così da garantire
incolonnamento corretto

€ 1.234,57

€ -1.234,57

€ 0,57

€ 12.345.678,91

Ora è tutto molto
più carino 😊



FORMATTORI PERSONALIZZATI PER DATE/ORARI

Java

Pattern Letters and Symbols			
Symbol	Meaning	Presentation	Examples
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
g	modified-julian-day	number	2451334
Q/q	quarter-of-year	number/text	3; 03; Q3; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	day-of-week-in-month	number	3
a	am-pm-of-day	text	PM
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-day (1-24)	number	24
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30
s	second-of-minute	number	55
S	fraction-of-second	fraction	978
A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
v	generic time-zone name	zone-name	Pacific Time; PT

v	generic time-zone name	zone-name	Pacific Time; PT
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-O	GMT+8; GMT+08:00; UTC-08:00
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15
Z	zone-offset	offset-Z	+0000; -0800; -08:00
p	pad next	pad modifier	1
'	escape for text	delimiter	
"	single quote	literal	'
[optional section start		
]	optional section end		
#	reserved for future use		
{	reserved for future use		
}	reserved for future use		

FORMATTORI PERSONALIZZATI

Esempio: parsing del nome del mese

- Problema: come risalire alla *costante enumerativa mese* partendo dal *nome* del mese stesso?
 - ad esempio, come ottenere `Month.MAY` partendo da `"May"` ?
- Se il **nome del mese** è **in inglese**, basta **`Month.valueOf`**
 - purché la stringa sia scritta ESATTAMENTE come si aspetta `Month`, ossia *tutta in maiuscolo*

```
jshell1> Month.valueOf("May")
Exception java.lang.IllegalArgumentException: No enum constant
h.May
    at Enum.valueOf (Enum.java:240)
    at Month.valueOf (Month.java:106)
    at (#87:1)

jshell1> Month.valueOf("MAY")
$88 ==> MAY

jshell1> Month.valueOf("may")
Exception java.lang.IllegalArgumentException: No enum constant
h.may
    at Enum.valueOf (Enum.java:240)
    at Month.valueOf (Month.java:106)
    at (#89:1)
```

Java



FORMATTORI PERSONALIZZATI

Esempio: parsing del nome del mese

- Ma se il nome del mese è in italiano??
 - il metodo `Month.valueOf` non funziona più!
- Definire un nostro enumerativo **Mese** per i mesi in italiano *non avrebbe alcun senso*, sarebbe una *follia*
 - sarebbe un doppione
 - ma soprattutto *non sarebbe integrato nella libreria `java.time`* che, quindi, non riuscirebbe a usarlo
 - ergo, *farebbe impazzire* perché bisognerebbe *continuamente convertire valori dell'uno in valori dell'altro*: DELIRIO!
- Molto meglio definire un *formattatore personalizzato*, tramite il metodo `DateTimeFormatter.ofPattern`

Java



FORMATTATORI PERSONALIZZATI

Esempio: parsing del nome del mese

Patterns for Formatting and Parsing

Patterns are based on a simple sequence of letters and symbols. A pattern is used to create a Formatter u
'3 Dec 2011'. A formatter created from a pattern can be used as many times as necessary, it is immutable

For example:

```
LocalDate date = LocalDate.now();  
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");  
String text = date.format(formatter);  
LocalDate parsedDate = LocalDate.parse(text, formatter);
```

All letters 'A' to 'Z' and 'a' to 'z' are reserved as pattern letters. The following pattern letters are

Symbol	Meaning	Presentation	Examples
-----	-----	-----	-----
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
Q/q	quarter-of-year	number/text	3; 03; 03; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	week-of-month	number	3
a	am-pm-of-day	text	PM
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-am-pm (1-24)	number	0
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30
s	second-of-minute	number	55
S	fraction-of-second	fraction	978
A	milli-of-day	number	1234

A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-0	GMT+8; GMT+08:00; UTC-08:00;
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15;
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15;
Z	zone-offset	offset-Z	+0000; -0800; -08:00;
p	pad next	pad modifier	1
'	escape for text	delimiter	
''	single quote	literal	'
[optional section start		
]	optional section end		
#	reserved for future use		
{	reserved for future use		
}	reserved for future use		

*Less than 4 pattern letters will use the short form.
Exactly 4 pattern letters will use the full form.*

D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10

Java



FORMATTORI PERSONALIZZATI

Esempio: parsing del nome del mese

- Poiché vogliamo riconoscere il nome completo del mese
 - ad esempio "maggio", *non "mag" o altre forme abbreviate*
- ci serve un formattatore in formato FULL → pattern **"MMMM"**

```
public class MyDateUtils {  
    public static int getMonth(String s, Locale l){  
        DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM", l);  
        java.time.temporal.TemporalAccessor t =  
            f.parse(s.toLowerCase());  
  
        return t.get(  
            java.time.temporal.ChronoField.MONTH_OF_YEAR);  
    }  
}
```

Java

Converte prima la stringa in minuscolo, perché viene riconosciuto solo "maggio"

Il metodo `parse` restituisce un `TemporalAccessor`, da cui occorre estrarre lo specifico campo `ChronoField.MONTH_OF_YEAR`

FORMATTORI PERSONALIZZATI

Esempio: parsing del nome del mese

- Finalmente, possiamo collaudarlo:

Java

```
System.out.println(  
    MyDateUtils.getMonth("Maggio", Locale.ITALY));
```

Risultato: **5**

Accettata solo perché il nostro metodo converte prima la stringa in minuscolo

- Volete provare altri paesi e culture?
- OCCHIO, *non sempre i mesi vanno scritti minuscoli!*
 - forse, dopo tutto, incapsulare dentro al metodo una conversione *valida solo per l'italiano* non è stata una grande idea! ☹
 - meglio toglierla e lasciare al chiamante tale onore (onere..)

```
java.time.temporal.TemporalAccessor t =  
f.parse(s.toLowerCase());
```



FORMATTORI PERSONALIZZATI

Esempio: parsing del nome del mese

- Refactoring (senza conversioni della stringa)

Java

```
public class MyDateUtils {  
    public static int getMonth(String s, Locale l){  
        DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM", l);  
        java.time.temporal.TemporalAccessor t = f.parse(s);  
        return t.get(java.time.temporal.ChronoField.MONTH_OF_YEAR);  
    }  
}
```

- Nuovi test:

```
System.out.println(  
    MyDateUtils.getMonth("mei", new Locale("nl", "NL")));  
System.out.println(  
    MyDateUtils.getMonth("mai", Locale.FRENCH));  
System.out.println(  
    MyDateUtils.getMonth("May", Locale.UK));
```

Non esiste una costante predefinita per l'Olanda, occorre costruirla

Infrastruttura C#

FORMATTATORI IN C#

- Come in Java, anche qui ci sono più tipologie di formattatori:
 - formattatori *numerici*
 - formattazione di numeri
 - formattazione di percentuali
 - formattazione di prezzi e valori espressi in valuta
 - formattatori per *date e orari*
 - formattazione di una data
 - formattazione di un orario
 - formattazione di una data con orario
- A differenza di Java, spesso non è necessario creare il formattatore esplicitamente
 - grazie alla *string interpolation*, basta indicare l'appropriata *specifica di formato* nel metodo che genera la stringa (es. `WriteLine`)

FORMATTATORI IN C#

- Cultura locale: **System.Globalization.CultureInfo**

- Esempio:

```
System.Globalization.CultureInfo itLocale =  
    new System.Globalization.CultureInfo("it-IT");
```

C#

- Formattazione di numeri tramite specifiche in **WriteLine**

```
Console.WriteLine("x={0:N}, y={1:c}, z={2:p}", x,y,z);
```

Numeric

Currency

Percentage

- Formattazione di date tramite appositi metodi **ToString**

- Formattazione con cultura locale di default

```
Console.WriteLine(today.ToString("F"));
```

- Formattazione con cultura locale specifica

```
Console.WriteLine(today.ToString("F",itLocale));
```

Formato Full

CULTURE LOCALI IN C#

- Le culture locali si distinguono in *neutre* («it») e *specifiche* («it_IT»)
 - neutre: solo la lingua, prescindendo da uno specifico paese
 - specifiche: coppia lingua + Paese

C#

ESEMPIO

```
Console.WriteLine("{0,-20}{1,4}", "#All cultures:",  
    CultureInfo.GetCultures(CultureTypes.AllCultures).Length);  
  
Console.WriteLine("{0,-20}{1,4}", "#Neutral cultures:",  
    CultureInfo.GetCultures(CultureTypes.NeutralCultures).Length);  
  
Console.WriteLine("{0,-20}{1,4}", "#Specific cultures:",  
    CultureInfo.GetCultures(CultureTypes.SpecificCultures).Length);
```

```
#All cultures:      354  
#Neutral cultures:  144  
#Specific cultures: 210
```

.NET Framework 4
C# 5.0

```
#All cultures:      733  
#Neutral cultures:  223  
#Specific cultures: 510
```

.NET Framework 5
C# 6+

CULTURE LOCALI IN C#

- Una tabella per elencarle tutte..?

```
Console.OutputEncoding = System.Text.Encoding.UTF8;
```

Utile per vedere anche
nomi con simboli «strani»

C#

```
Console.WriteLine("{0,-15}{0,-5}{0,-45}{0,-40}",
    "Culture", "ISO", "Display name", "English Name");

foreach (CultureInfo ci in CultureInfo.GetCultures(CultureTypes.AllCultures)) {
    Console.Write("{0,-15}", ci.Name);
    Console.Write("{0,-5}", ci.TwoLetterISOLanguageName);
    Console.Write("{0,-45}", ci.DisplayName);
    Console.WriteLine("{0,-40}", ci.EnglishName);
};
```

Culture	Culture	Culture	Culture
	iv		Invariant Language (Invariant Country)
af	af	af	Afrikaans
af-NA	af	Afrikaans (Namibië)	Afrikaans (Namibia)
af-ZA	af	af (ZA)	Afrikaans (South Africa)
...			
it	it	it	Italian
it-CH	it	it (CH)	Italian (Switzerland)
it-IT	it	it (IT)	Italian (Italy)
it-SM	it	italiano (San Marino)	Italian (San Marino)
it-VA	it	italiano (Città del Vaticano)	Italian (Vatican City)

FORMATTAZIONE DI NUMERI: ESEMPI

```
double x = 43.12345678, y = 0.7, z = 13456.78, w = 13_456.78;
double u = 0.4312, v = 0.7, r = 1.2345678;
```

C#

```
CultureInfo itIT = new CultureInfo("it-IT");
CultureInfo itCH = new CultureInfo("it-CH");
CultureInfo enGB = new CultureInfo("en-GB");
CultureInfo enCA = new CultureInfo("en-CA");
CultureInfo frCA = new CultureInfo("fr-CA");
CultureInfo deDE = new CultureInfo("de-DE");
```

```
Current culture: en-GB
x = 43.12345678, y = 0.7,
z = 13456.78, w = 13,456.780,
w = £13,456.78
u = 43.12%, v = 70.00%, r = 123.46%
```

```
CultureInfo.DefaultThreadCurrentCulture = enUK;
```

```
Console.WriteLine($"Current culture: {CultureInfo.CurrentCulture.Name}");
```

String interpolation with prefix (C#6.0)

String interpolation: argument #3 (z) as Currency

```
Console.WriteLine("x = {0}, y = {1}, z = {2}, w = {3:N}, w = {3:c}" , x,y,z,w);
```

String interpolation: argument #0 (x)

String interpolation: argument #3 (z) as Number

```
Console.WriteLine("u = {0:p2}, v = {1:p2}, r = {2:p2}" , u,v,r);
```

String interpolation: argument #0 (u) as Percentage with 2 decimal digits

FORMATTAZIONE DI NUMERI: ESEMPI

Current culture:

x = 43.12345678, y = 0.7, z = 13456.78, w = 13,456.78, w = ¤13,456.78
p = 43.12 %, q = 70.00 %, r = 123.46 %

Current culture: en-GB

x = 43.12345678, y = 0.7, z = 13456.78, w = 13,456.780, w = £13,456.78
p = 43.12%, q = 70.00%, r = 123.46%

Current culture: en-CA

x = 43.12345678, y = 0.7, z = 13456.78, w = 13,456.780, w = \$13,456.78
p = 43.12%, q = 70.00%, r = 123.46%

Current culture: fr-CA

x = 43,12345678, y = 0,7, z = 13456,78, w = 13 456,780, w = 13 456,78 \$
p = 43,12 %, q = 70,00 %, r = 123,46 %

Current culture: it-IT

x = 43,12345678, y = 0,7, z = 13456,78, w = 13.456,780, w = 13.456,78 €
p = 43,12%, q = 70,00%, r = 123,46%

Current culture: it-CH

x = 43.12345678, y = 0.7, z = 13456.78, w = 13'456.780, w = CHF 13'456.78
p = 43.12%, q = 70.00%, r = 123.46%

Current culture: de-DE

x = 43,12345678, y = 0,7, z = 13.456,780, w = 13.456,780, w = 13.456,78 €
p = 43,12 %, q = 70,00 %, r = 123,46 %

C#

Default:
indefinita

Canada
(en/fr)

Italia vs.
Svizzera it.

Germania: spazio prima
del simbolo di %

ALTRE CARATTERISTICHE CULTURALI

```
Console.WriteLine("First day of the week in {0}: {1}",  
    CultureInfo.CurrentCulture.DisplayName,  
    CultureInfo.CurrentCulture.DateTimeFormat.FirstDayOfWeek.ToString());
```

Primo giorno
della settimana

C#

```
Console.WriteLine("{0}",  
    String.Join(", ", CultureInfo.CurrentCulture.DateTimeFormat.DayNames));
```

Nomi dei giorni della settimana

```
Console.WriteLine("{0}", String.Join(", ",  
    CultureInfo.CurrentCulture.DateTimeFormat.MonthNames));
```

Nomi dei mesi

```
Current culture: en-GB  
First day of the week in en (GB): Monday  
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday  
January, February, March, April, May, June, July, August, September, October, November, December,  
Current culture: fr-CA  
First day of the week in fr (CA): Sunday  
dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi  
janvier, février, mars, avril, mai, juin, juillet, août, septembre, octobre, novembre, décembre,  
Current culture: it-IT  
First day of the week in it (IT): Monday  
domenica, lunedì, martedì, mercoledì, giovedì, venerdì, sabato  
gennaio, febbraio, marzo, aprile, maggio, giugno, luglio, agosto, settembre, ottobre, novembre, dicembre,  
Current culture: de-DE  
First day of the week in de (DE): Monday  
Sonntag, Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag  
Januar, Februar, März, April, Mai, Juni, Juli, August, September, Oktober, November, Dezember,
```

C#

FORMATTAZIONE DI DATE E ORARI: ESEMPI

```
System.Globalization.CultureInfo itLocale =  
    new System.Globalization.CultureInfo("it-IT");  
  
DateTime today = DateTime.Today; // date only (time set to midnight)  
Console.WriteLine(today.ToString("F"));  
Console.WriteLine(today.ToString("F", itLocale));  
  
DateTime now = DateTime.Now; // date and time  
Console.WriteLine(now.ToString("F"));  
Console.WriteLine(now.ToString("F", itLocale));
```

C#

F o f

```
Saturday, 20 February 2021 00:00  
sabato 20 febbraio 2021 00:00  
Saturday, 20 February 2021 00:00:00  
sabato 20 febbraio 2021 00:00:00
```

f

Today

```
Saturday, 20 February 2021 18:30  
sabato 20 febbraio 2021 18:30  
Saturday, 20 February 2021 18:30:15  
sabato 20 febbraio 2021 18:30:15
```

Now

F

d = data breve

f = data completa, ora breve

g = data generale, ora breve

t = ora breve

m = mese e giorno

y = mese e anno

D = data estesa

F = data completa, ora estesa

G = data generale, ora estesa

T = ora estesa

M = mese e giorno

Y = mese e anno

FORMATTAZIONE DI DATE E ORARI: ESEMPI

```
DateTime now = DateTime.Now; // date and time
```

```
Console.WriteLine(now.ToString("d", enGB));
```

20/02/2021

```
Console.WriteLine(now.ToString("d", itIT));
```

20/02/2021

```
Console.WriteLine(now.ToString("D", enGB));
```

Saturday, 20 February 2021

```
Console.WriteLine(now.ToString("D", itIT));
```

sabato 20 febbraio 2021

```
Console.WriteLine(now.ToString("t", enGB));
```

18:24

```
Console.WriteLine(now.ToString("t", itIT));
```

18:24

```
Console.WriteLine(now.ToString("T", enGB));
```

18:24:17

```
Console.WriteLine(now.ToString("T", itIT));
```

18:24:17

```
Console.WriteLine(now.ToString("m", enGB));
```

20 February

```
Console.WriteLine(now.ToString("m", itIT));
```

20 febbraio

```
Console.WriteLine(now.ToString("Y", enGB));
```

February 2021

```
Console.WriteLine(now.ToString("Y", itIT));
```

febbraio 2021

C#

d = data breve

D = data estesa

f = data completa, ora breve

F = data completa, ora estesa

g = data generale, ora breve

G = data generale, ora estesa

t = ora breve

T = ora estesa

m o **M** = mese e giorno

y o **Y** = mese e anno



FORMATTAZIONE DI DATE E ORARI: ESEMPI

```
DateTime xmas2020 = new DateTime(2020, 12, 25);  
DateTime xmas2020noon = new DateTime(2020, 12, 25, 12, 0, 0);  
Console.WriteLine(xmas2020.ToString("F", itLocale));  
Console.WriteLine(xmas2020noon.ToString("F", itLocale));  
  
DateTime xmas2021 = xmas2020.AddYears(1);  
DateTime xmas2021noon = xmas2021.AddHours(12);  
Console.WriteLine(xmas2021.ToString("F", itLocale));  
Console.WriteLine(xmas2021noon.ToString("F", itLocale));
```

Solo giorno

C#

Giorno
e ora

```
venerdì 25 dicembre 2020 00:00:00  
venerdì 25 dicembre 2020 12:00:00  
  
sabato 25 dicembre 2021 00:00:00  
sabato 25 dicembre 2021 12:00:00
```



FORMATTAZIONE DI DATE E ORARI: ESEMPI

```
TimeZoneInfo localZone = TimeZoneInfo.Local;  
TimeZoneInfo utcZone   = TimeZoneInfo.Utc;
```

C#

```
Console.WriteLine("Current timezone: {0}", localZone.StandardName);  
Console.WriteLine("Daylight name: {0}", localZone.DaylightName);  
Console.WriteLine("Bias: {0}", localZone.BaseUtcOffset);  
Console.WriteLine("UTC timezone: {0}", utcZone.StandardName);  
Console.WriteLine("UTC Daylight name: {0}", utcZone.DaylightName);  
Console.WriteLine("UTC Bias: {0}", utcZone.BaseUtcOffset);
```

```
Current timezone: Coordinated Universal Time  
Daylight name: Coordinated Universal Time  
Bias: 00:00:00  
UTC timezone: UTC  
UTC Daylight name: UTC  
UTC Bias: 00:00:00
```

FORMATTAZIONE DI DATE E ORARI: ESEMPI

```

DateTime whatTime = today           // 19 febbraio 2021 00:00:00
                                     // 19 febbraio 2021 12:00:00
                                     // 19 febbraio 2021 12:10:00
                                     // 18 febbraio 2021 23:10:00
                                     // 18 febbraio 2021 23:10:10
    .AddHours(12)
    .AddMinutes(10)
    .AddHours(-13)
    .AddSeconds(10);
Console.WriteLine(whatTime.ToString("F", itLocale));

DateTimeOffset whatTimeAbsolute =
    new DateTimeOffset(2021, 2, 19, 21, 4, 0, new TimeSpan(1, 0, 0));
Console.WriteLine("whatTimeAbsolute = {0}",
    whatTimeAbsolute.ToString("F", itLocale));
Console.WriteLine("whatTimeAbsolute.UTC = {0}",
    whatTimeAbsolute.UtcDateTime.ToString("F", itLocale));

```

Ore 21:04 in UTC+1

Proprietà UTC di un
orario assoluto

```

giovedì 18 febbraio 2021 23:10:10

```

```

whatTimeAbsolute = venerdì 19 febbraio 2021 21:04:00

```

```

whatTimeAbsolute.UTC = venerdì 19 febbraio 2021 20:04:00

```

Le 21:04 in UTC+1...

...sono le 20:04 UTC

FORMATTAZIONE DI DATE E ORARI: ESEMPI

```

C#
63 giorni, 0 ore e minuti
1 anno di differenza (365 giorni)
DateTime supporta operatori aritmetici!
97 giorni
1-esimo (2°) giorno della settimana
Costante enumerativa

```

`TimeSpan p1 = new TimeSpan(63,0,0,0);`
`Console.WriteLine(p1.ToString());`
`TimeSpan p2 = xmas2021 - xmas2020;`
`Console.WriteLine(p2.ToString());`
`DateTime primoAprile = xmas2020.Add(new TimeSpan(97,0,0,0));`
`Console.WriteLine(primoAprile.ToString("F", itLocale));`
`DayOfWeek d = (DayOfWeek)1;`
`Console.WriteLine(d.ToString("F")); // Monday`
`Console.WriteLine(DayOfWeek.Tuesday); // Tuesday`

```

63.00:00:00
365.00:00:00
giovedì 1 aprile 2021 00:00:00
Monday
Tuesday

```

97 giorni
dopo Natale

FORMATTAZIONE DI DATE E ORARI: ESEMPI

```
DateTime startSemester = new DateTime(2021, 2, 17);
```

C#

Formati di ToString: giorno breve / esteso

```
Console.WriteLine(startSemester.ToString("ddd"));
Console.WriteLine(startSemester.ToString("ddd", itIT));
Console.WriteLine(startSemester.ToString("dddd"));
Console.WriteLine(startSemester.ToString("dddd", itIT));
```

Wed
mer
Wednesday
mercoledì

```
Console.WriteLine("xmas2020 = {0}, xmas2021 = {1}, isBefore = {2}",
    xmas2020.ToString("F", itLocale),
    xmas2021.ToString("F", itLocale),
    xmas2020 < xmas2021);
```

Su DateTime sono
possibili anche
operatori relazionali

```
xmas2020 = venerdì 25 dicembre 2020 00:00:00,  
xmas2021 = sabato 25 dicembre 2021 00:00:00,  
isBefore = True
```

PARSING DI NUMERI, DATE E ORARI

- Il parsing viene gestito dai metodi statici **TryParse** C#
 - analizzano la stringa e ne determinano *in modo smart* il formato
 - l'output è un argomento *passato per indirizzo*
 - il risultato restituito è un **Bool** che riporta successo/fallimento
 - ci sono anche i metodi (non statici) **Parse** e **ParseExact** che però lanciano errore se la stringa non rispetta il formato previsto
- Ci sono fondamentalmente due versioni di **TryParse**
 - quella a due argomenti, che usa la cultura predefinita
 - quella a quattro argomenti, che consente di specificare la cultura
- Ci sono metodi **TryParse**
 - nei tipi numerici (**double**, **decimal**, **int**...) per parsing di numeri
 - in **DateTime**, per date e orari

PARSING DI NUMERI, DATE E ORARI

Schema base per numeri

C#

```
double d;  
if (double.TryParse(numString, out d)) ...  
if (double.TryParse(numString, culture, style, out d)) ...
```

Passaggio per
indirizzo

Schema base per date

C#

```
DateTime dateValue;  
if (DateTime.TryParse(dateString, out dateValue)) ...  
if (DateTime.TryParse(dateString, culture, style, out dateValue)) ...
```

Passaggio per
indirizzo

PARSING DI NUMERI: ESEMPI

C#

```
using System;
using System.Globalization;

public class Program
{
    public static void Main() {
        CultureInfo[] cultures = { new CultureInfo("it-IT"), new CultureInfo("en-US") };
        string[] numStrings = { "12€", "12 €", "12,34 €", "€ 12", "€12", "€ 12,54", "€ 12.21", "13", "$ 13", "13.66$", "13,77$" };

        foreach (CultureInfo culture in cultures) {
            Console.WriteLine("Cultura: {0}", culture);
            foreach (string numString in numStrings){
                double d;
                if( double.TryParse(numString, NumberStyles.Currency, culture, out d))
                    Console.WriteLine("La stringa {0} è stata convertita in {1} secondo la cultura {2}", numString, d, culture);
                else
                    Console.WriteLine("ERRORE: la stringa {0} non è stata riconosciuta nella cultura {1}", numString, culture);
            }
        }
    }
}
```

```
Cultura: it-IT
La stringa 12€ è stata convertita in 12 secondo la cultura it-IT
La stringa 12 € è stata convertita in 12 secondo la cultura it-IT
La stringa 12,34 € è stata convertita in 12.34 secondo la cultura it-IT
La stringa € 12 è stata convertita in 12 secondo la cultura it-IT
La stringa €12 è stata convertita in 12 secondo la cultura it-IT
La stringa € 12,54 è stata convertita in 12.54 secondo la cultura it-IT
La stringa € 12.21 è stata convertita in 12.21 secondo la cultura it-IT
La stringa 13 è stata convertita in 13 secondo la cultura it-IT
ERRORE: la stringa $ 13 non è stata riconosciuta nella cultura it-IT
ERRORE: la stringa 13.66$ non è stata riconosciuta nella cultura it-IT
ERRORE: la stringa 13,77$ non è stata riconosciuta nella cultura it-IT
Cultura: en-US
ERRORE: la stringa 12€ non è stata riconosciuta nella cultura en-US
ERRORE: la stringa 12 € non è stata riconosciuta nella cultura en-US
ERRORE: la stringa 12,34 € non è stata riconosciuta nella cultura en-US
ERRORE: la stringa € 12 non è stata riconosciuta nella cultura en-US
ERRORE: la stringa €12 non è stata riconosciuta nella cultura en-US
ERRORE: la stringa € 12,54 non è stata riconosciuta nella cultura en-US
ERRORE: la stringa € 12.21 non è stata riconosciuta nella cultura en-US
La stringa 13 è stata convertita in 13 secondo la cultura en-US
La stringa $ 13 è stata convertita in 13 secondo la cultura en-US
La stringa 13.66$ è stata convertita in 13.66 secondo la cultura en-US
La stringa 13,77$ è stata convertita in 13.77 secondo la cultura en-US
```

OCCHIO a quel che succede
usando il separatore decimale
sbagliato..!

PARSING DI DATE E ORARI: ESEMPI

C#

```
public class Program
{
    public static void Main() {
        string[] dateStrings = { "05/01/2009 14:57:32.8", "2009-05-01 14:57:32.8",
                                "2009-05-01T14:57:32.8375298-04:00", "5/01/2008",
                                "5/01/2008 14:57:32.80 -07:00",
                                "1 May 2008 2:57:32.8 PM", "16-05-2009 1:00:32 PM",
                                "Fri, 15 May 2009 20:10:57 GMT" };

        DateTime dateValue;

        Console.WriteLine("Attempting to parse strings using {0} culture:", CultureInfo.CurrentCulture.Name);
        foreach (string dateString in dateStrings) {
            if (DateTime.TryParse(dateString, out dateValue))
                Console.WriteLine("  Converted '{0}' to {1} ({2}).", dateString, dateValue.ToString("F"), dateValue.Kind);
            else
                Console.WriteLine("  Unable to parse '{0}'.", dateString);
        }

        CultureInfo itIT = new CultureInfo("it-IT");
        Console.WriteLine("Tentativo di parsing delle stesse stringhe usando la cultura {0}:", itIT);
        foreach (string dateString in dateStrings) {
            DateTimeStyles style = DateTimeStyles.None; // OR: AssumeLocal AdjustToUniversal
            if (DateTime.TryParse(dateString, itIT, style, out dateValue))
                Console.WriteLine("  Converted '{0}' to {1} ({2}).", dateString, dateValue.ToString("F", itIT), dateValue.Kind);
            else
                Console.WriteLine("  Unable to parse '{0}'.", dateString);
        }
    }
}
```

PARSING DI DATE E ORARI: ESEMPI

```
string[] dateStrings = { "05/01/2009 14:57:32.8", "2009-05-01 14:57:32.8",
    "2009-05-01T14:57:32.8375298-04:00", "5/01/2008",
    "5/01/2008 14:57:32.80 -07:00",
    "1 May 2008 2:57:32.8 PM", "16-05-2009 1:00:32 PM",
    "Fri, 15 May 2009 20:10:57 GMT" };
```

C#

Cultura USA:
16-05-2009 non è interpretabile

```
Attempt to parse strings using en-US culture:
Converted '05/01/2009 14:57:32.8' to Friday, May 1, 2009 2:57:32 PM (Unspecified).
Converted '2009-05-01 14:57:32.8' to Friday, May 1, 2009 2:57:32 PM (Unspecified).
Converted '2009-05-01T14:57:32.8375298-04:00' to Friday, May 1, 2009 6:57:32 PM (Local).
Converted '5/01/2008' to Thursday, May 1, 2008 12:00:00 AM (Unspecified).
Converted '5/01/2008 14:57:32.80 -07:00' to Thursday, May 1, 2008 9:57:32 PM (Local).
Converted '1 May 2008 2:57:32.8 PM' to Thursday, May 1, 2008 2:57:32 PM (Unspecified).
Unable to parse '16-05-2009 1:00:32 PM'
Converted 'Fri, 15 May 2009 20:10:57 GMT' to Friday, May 15, 2009 8:10:57 PM (Local).

Tentativo di parsing delle stesse stringhe usando la cultura it-IT:
Converted '05/01/2009 14:57:32.8' to lunedì 5 gennaio 2009 14:57:32 (Unspecified).
Converted '2009-05-01 14:57:32.8' to venerdì 1 maggio 2009 14:57:32 (Unspecified).
Converted '2009-05-01T14:57:32.8375298-04:00' to venerdì 1 maggio 2009 18:57:32 (Local).
Converted '5/01/2008' to sabato 5 gennaio 2008 00:00:00 (Unspecified).
Converted '5/01/2008 14:57:32.80 -07:00' to sabato 5 gennaio 2008 21:57:32 (Local).
Converted '1 May 2008 2:57:32.8 PM' to giovedì 1 maggio 2008 14:57:32 (Unspecified).
Converted '16-05-2009 1:00:32 PM' to sabato 16 maggio 2009 13:00:32 (Unspecified).
Converted 'Fri, 15 May 2009 20:10:57 GMT' to venerdì 15 maggio 2009 20:10:57 (Local).
```

Cultura USA:
5/1/2009 è il
1° maggio 2009

Cultura Italia:
5/1/2009 è il
5 gennaio 2009

Cultura Italia:
16-05-2009 è correttamente interpretato come il 16 maggio 2009