

GESTIONE DEL TESTO

- ❑ Linguaggio naturale
- ❑ Il package `java.text`
- ❑ Analisi di stringhe
- ❑ Formatter per numeri, stringhe, date/time

Linguaggio naturale

Interfacce utente

- ❑ Un programma comunica con i suoi utilizzatori
 - Attraverso immagini, suoni e testi
- ❑ I messaggi non sono sempre completamente predefiniti
 - Spesso il loro contenuto è parametrico
- ❑ Anche l'utente comunica
 - Utilizza “segni convenzionali”:
 - movimenti del mouse, sequenze di caratteri
 - Occorre riconoscerne il significato

Comunicazione testuale

- ❑ Sfrutta il linguaggio naturale dell'utilizzatore
 - È legata alla sua “cultura”
 - Lingua adottata, convenzioni di rappresentazione
- ❑ Impone pesanti vincoli quando occorre realizzare programmi “multiculturali”

Sintesi di messaggio

- ❑ Un messaggio testuale è composto da una sequenza di caratteri
 - In parte prefissati
 - In parte determinati in fase di esecuzione
- ❑ Occorre tenere presenti le regole della sintassi
 - Concordanze (genere, numero, persona, ...)

//Restituisce il messaggio adattato

```
String getMessage(int i) {  
    String s;  
    if (i!=1)  
        s="Sono stati letti "+i+" elementi.";  
    else  
        s="È stato letto un elemento.";  
    return s;  
}
```

Rappresentare i numeri

- ❑ Le classi “wrapper” offrono il metodo statico `toString(...)`
 - Restituisce la stringa corrispondente al parametro
 - Non si può controllare la formattazione
 - Numero di cifre, separatore decimale, notazione, ...

Analisi di testi

- ❑ Estremamente più complessa
 - Occorre suddividere, a priori, il testo in unità elementari
- ❑ In alcuni casi occorre trattare informazioni strutturate
 - Data, ora, valuta, ...
- ❑ La disposizione ed il contenuto dipendono dalla cultura dell'interlocutore

Scandire una stringa 1

- ❑ Si può usare la classe `java.util.StringTokenizer`
 - Suddivisione basata sugli spazi
 - Pochi controlli disponibili

```
StringTokenizer st =  
    new StringTokenizer(  
        "Questa è una prova");  
  
while (st.hasMoreTokens())  
    System.out.println(st.nextToken());
```

**Questa
è
una
prova**

Scandire una stringa 2

- ❑ Si può usare la classe `java.util.Scanner`
 - Suddivisione basata sul carattere fornito (default space)
 - Input da file, tastiera, string, stream in generale
 - Controlli di 'tipo' prima della lettura

```
Scanner sc =  
    new Scanner(  
        "Questa è la prova numero 8");  
while (sc.hasNext()) {  
    if (sc.hasNextInt()) {  
        System.out.println(sc.next () + ` Int`);  
    } else {  
        System.out.println(sc.next ());  
    }  
}
```

**Questa
è
la
Prova
Numero
8 Int**

Analizzare un numero

- ❑ Le classi “wrapper” offrono metodi per convertire stringhe in numeri
 - `Integer.parseInt(String val)`
 - `Integer.valueOf(String val)`
 - `Double.parseDouble(String v)`
 - `Double.valueOf(String v)`
 - Possono lanciare **`NumberFormatException`**
- ❑ Basati sulla rappresentazione anglosassone

```
String s="123,456";  
double d;  
try {  
    d=Double.parseDouble(s);  
} catch (NumberFormatException nfe) {  
    d=0.0;  
}
```

Confronto

- ❑ La classe String offre il metodo **compareTo**
 - Basato sulla rappresentazione UNICODE dei caratteri
 - Non fornisce i risultati attesi con le lettere **accentate**

Internazionalizzazione

❑ Fattori da gestire quando si realizzano versioni per più culture

- Messaggi
- Componenti grafici
- Suggerimenti
- Suoni
- Colori
- Icone
- Numeri (valute/misure)
- Data/ora
- Indirizzi
- Numeri telefonici
- Impaginazione

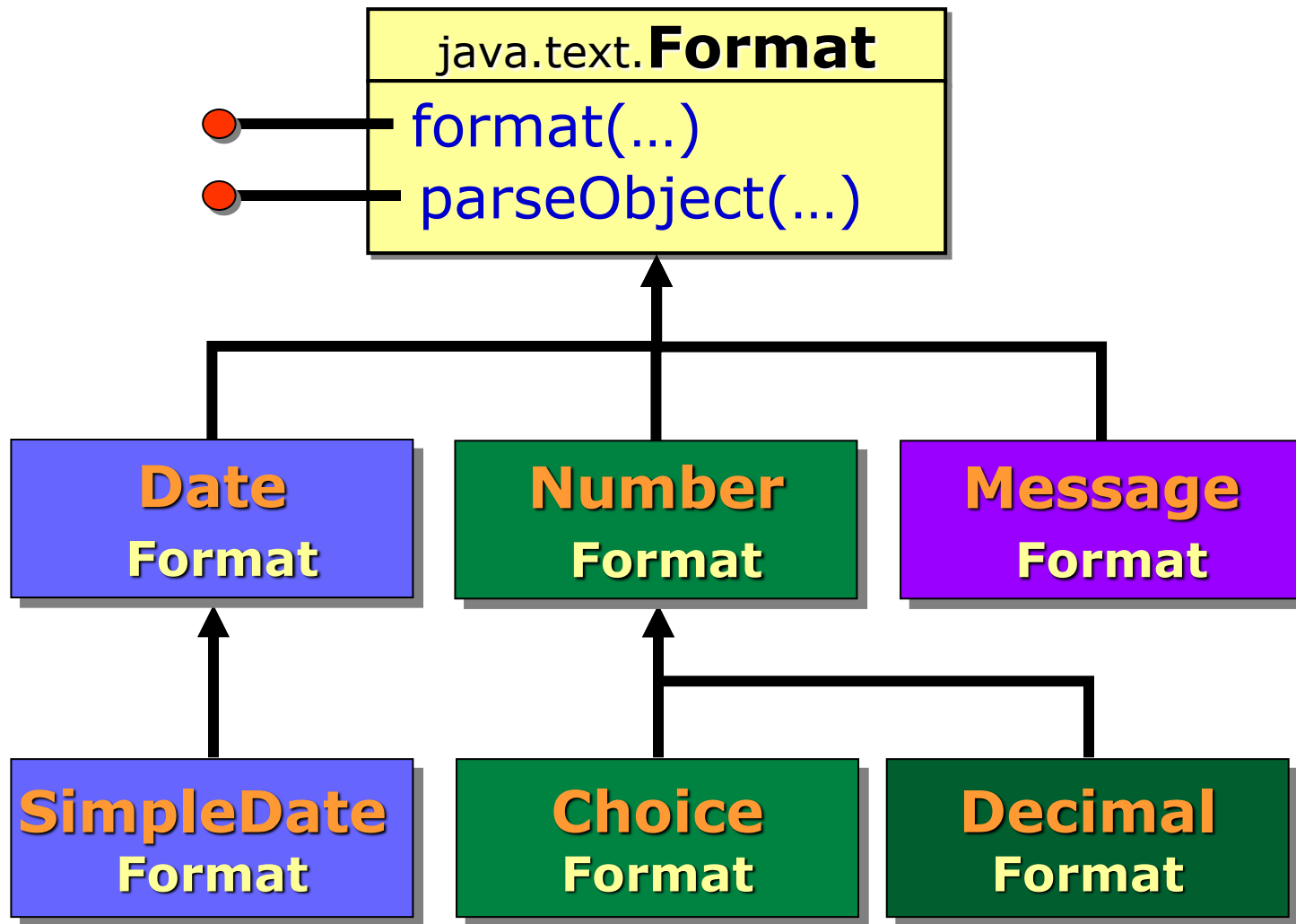
Culture

- ❑ La classe `java.util.Locale` modella un ambito **culturale specifico**
 - In termini di **lingua** e **nazione**
- ❑ La conoscenza del locale permette di adattare le operazioni di analisi e sintesi

```
String getMessage() {  
    Locale loc=Locale.getDefault();  
  
    //formato ISO 639: codice a due lettere  
    String lang=loc.getLanguage();  
  
    String msg;  
    if ( "it".equals(lang) )  
        msg="Premere <invio>";  
    else  
        msg="Press <enter>";  
    return msg;  
}
```

Il package `java.text`

- Insieme di classi specializzate nella gestione del testo
 - Analisi/sintesi/confronto
 - Numeri, date, messaggi testuali
 - In dipendenza di una cultura specifica (Locale)



NumberFormat

- ❑ Fornisce metodi per l'analisi e la sintesi di valori numerici:
 - Numeri frazionari
 - Percentuali
 - Valuta
- ❑ Fornisce risultati differenti in funzione della cultura indicata

```
String getValue(double d) {  
    Locale loc=Locale.ITALIAN;  
  
    NumberFormat nf=  
        NumberFormat.getInstance(loc);  
  
    String s=nf.format(d);  
  
    return s;  
}
```

getValue(99.95)

99,95

```
double parse(String s)
    throws ParseException {

    Locale loc=Locale.ITALIAN;

    NumberFormat nf=
        NumberFormat.getNumberInstance(loc);

    Number n=nf.parse(s);

    return n.doubleValue();
}
```

`parse("99,95")`

`99.95`

```
String getPrice(double d) {  
    Locale loc=Locale.ITALIAN;  
  
    NumberFormat nf=  
        NumberFormat.getCurrencyInstance(loc);  
  
    String s=nf.format(d);  
  
    return s;  
}
```

getPrice(99.95)

€ 99.95

DateFormat

- Analisi e rappresentazione di data e/o ora
 - Diversi formati (breve, medio, lungo, completo)
 - Permette di impostare il fuso orario (time zone)
 - Supporta calendari arbitrari (classe `java.util.Calendar`)

```
String getTime(Date d) {  
    Locale loc=Locale.ITALIAN;  
  
    DateFormat df=  
        DateFormat.getInstance(  
            DateFormat.SHORT, loc);  
  
    String s=df.format(d);  
  
    return s;  
}
```

getTime(
 new Date())

17.33

MessageFormat

- ❑ Classe che offre funzionalità sofisticate per costruire ed analizzare messaggi
 - Utilizza una stringa di formato ed un array di parametri
 - Simile alla funzione `sprintf()` del linguaggio C


```
String getMsg(String s, int i) {  
  
    String pattern=  
        "La cartella {0} contiene {1} file";  
  
    MessageFormat mf=  
        new MessageFormat(pattern);  
  
    Object[] args= new Object[]  
        {s, new Integer(i)};  
  
    return mf.format(args);  
}
```

getMsg("c:\",3)

**La cartella c:\
contiene 3 file**

Collator

- ❑ Classe specializzata nel confrontare stringhe in un dato linguaggio
 - Riordina le lettere accentate, in modo da garantirne l'opportuna sequenza
 - Offre vari livelli di precisione nel confronto

```
String[] parole= {"Squadra","sì", "si"};
```

```
Locale loc= Locale.ITALIAN;
```

```
Collator c= Collator.getInstance(loc);  
c.setStrength(Collator.SECONDARY);
```

```
Arrays.sort(parole,c);
```

```
for (int i=0; i<parole.length; i++)  
    System.out.println(parole[i]);
```

si
sì
Squadra

Formattazione numeri, stringhe, date/time

String.format

- ❑ Metodo di String per formattare stringhe, numeri di ogni tipo e precisione, date e time
 - Utilizza il tipo formato **%x** (x=a,b,c,...s,t,...)
 - I parametri richiesti sono una stringa con il pattern e i successivi parametri con i valori
 - Si richiama esplicitamente da un oggetto String
 - Si utilizza implicitamente in `System.out.printf(...)`

String.format

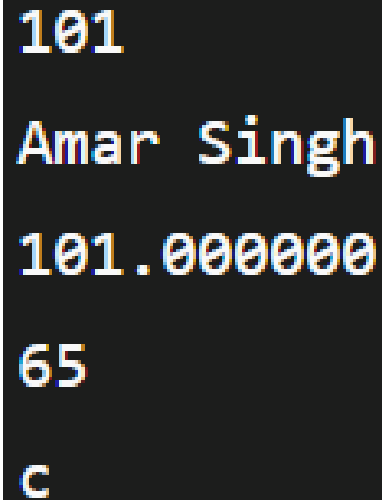
```
1. public class FormatExample{  
2. public static void main(String args[]){  
3. String name="sonoo";  
4. String sf1=String.format("name is %s",name);  
5. String sf2=String.format("value is %f",32.33434);  
6. String sf3=String.format("value is %32.12f",32.33434); //returns 12 char fra  
   ctional part filling with 0  
7.  
8. System.out.println(sf1);  
9. System.out.println(sf2);  
10. System.out.println(sf3);  
11. }}
```

Format Specifier	Data Type	Output
%a	floating point (except <i>BigDecimal</i>)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation

%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value

%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.
%x	integer (incl. byte, short, int, long, bigint)	Hex string.

```
1. public class FormatExample2 {  
2.     public static void main(String[] args) {  
3.         String str1 = String.format("%d", 101);        // Integer value  
4.         String str2 = String.format("%s", "Amar Singh"); // String value  
5.         String str3 = String.format("%f", 101.00);     // Float value  
6.         String str4 = String.format("%x", 101);        // Hexadecimal value  
7.         String str5 = String.format("%c", 'c');        // Char value  
8.         System.out.println(str1);  
9.         System.out.println(str2);  
10.        System.out.println(str3);  
11.        System.out.println(str4);  
12.        System.out.println(str5);  
13.    }  
14.  
15.}
```

A screenshot of a terminal window with a dark background, showing the output of the Java program. The output consists of five lines: '101', 'Amar Singh', '101.000000', '65', and 'c'.

```
101  
Amar Singh  
101.000000  
65  
c
```

```

1. public class FormatExample3 {
2.     public static void main(String[] args) {
3.         String str1 = String.format("%d", 101);
4.         String str2 = String.format("|%10d|", 101); // Specifying length of integer
5.         String str3 = String.format("|%-10d|", 101); // Left-justifying within the specified width
6.         String str4 = String.format("|% d|", 101);
7.         String str5 = String.format("|%010d|", 101); // Filling with zeroes
8.         System.out.println(str1);
9.         System.out.println(str2);
10.        System.out.println(str3);
11.        System.out.println(str4);
12.        System.out.println(str5);
13.    }
14.}

```

```

101
|           101|
|101         |
| 101|
|00000000101|

```

java.util.Formatter

- ❑ Fornisce supporto per layout justification e allineamento, formati comuni per numeric, string, date/time, con gestione Locale.
- ❑ Supporta Calendar e BigDecimal
- ❑ Gestisce l'ordinamento nella formattazione
- ❑ Utilizza **StringBuilder** per gestire le stringhe da formattare

java.util.Formatter

```
import java.util.Calendar;  
import java.util.Formatter;  
public class Main {  
    public static void main(String args[]) {  
        Formatter fmt = new Formatter();  
        Calendar cal = Calendar.getInstance();  
        fmt.format("%f", 343.5);  
        System.out.println(fmt);  
        fmt.close();  
    }  
}
```

java.util.Formatter per date/time

- ❑ **%t** formatta informazioni di time e date
- ❑ **%t** richiede l'uso di un **suffisso** per descrivere con precisione il formato desiderato di time e date (i.e. **%tr**, **%tH**, ...)
- ❑ Per visualizzare minuti, usare **%tM**, e **M** indica minuti in campo char lungo 2
- ❑ L'argomento corrispondente al **%t** deve essere del tipo **Calendar**, **Date**, **Long**, o **long**

java.util.Formatter suffissi per date/time

Suffix	Replaced By
a	Abbreviated weekday name
A	Full weekday name
b	Abbreviated month name
B	Full month name
c	Standard date and time string formatted as day month date hh::mm:ss tzzone year
C	First two digits of year
d	Day of month as a decimal (01-31)
D	month/day/year
e	Day of month as a decimal (1-31)
F	year-month-day
h	Abbreviated month name
H	Hour (00 to 23)
I	Hour (01 to 12)
j	Day of year as a decimal (001 to 366)
k	Hour (0 to 23)
l	Hour (1 to 12)
L	Millisecond (000 to 999)
m	Month as decimal (01 to 13)
M	Minute as decimal (00 to 59)

java.util.Formatter suffissi per date/time

N	Nanosecond (000000000 to 999999999)
p	Locale's equivalent of AM or PM in lowercase
Q	Milliseconds from 1/1/1970
r	hh:mm:ss (12-hour format)
R	hh:mm (24-hour format)
S	Seconds (00 to 60)
s	Seconds from 1/1/1970 UTC
T	hh:mm:ss (24-hour format)
y	Year in decimal without century (00 to 99)
Y	Year in decimal including century (0001 to 9999)
z	Offset from UTC
Z	Time zone name


```
import java.util.Calendar;
import java.util.Formatter;
public class Main {
    public static void main(String args[]) {
        Formatter fmt = new Formatter();
        Calendar cal = Calendar.getInstance();
        // Display month by name and number.
        fmt = new Formatter();
        fmt.format("%tB %tb %tm", cal, cal, cal);
        System.out.println(fmt);
    }
}
```

June Jun 06

```
import java.util.Calendar;
import java.util.Formatter;
public class Main {
    public static void main(String args[]) {
        Formatter fmt = new Formatter();
        Calendar cal = Calendar.getInstance();
        // Display standard 12-hour time format.
        fmt.format("%tr", cal); System.out.println(fmt);//
    }
}
```

09:13:45 AM

```
import java.util.Calendar;
import java.util.Formatter;
public class Main {
    public static void main(String args[]) {
        Formatter fmt = new Formatter();
        Calendar cal =
            Calendar.getInstance();
        // Display complete time and date information.
        fmt = new Formatter();
        fmt.format("%tc", cal);
        System.out.println(fmt);
    }
}
```

Thu Aug 09 09:14:01 PDT 2002

```
import java.util.Calendar;  
import java.util.Formatter;
```

```
public class Main {  
    public static void main(String args[]) {  
        Formatter fmt = new Formatter();  
        Calendar cal = Calendar.getInstance();  
        // Display just hour and minute.  
        fmt = new Formatter();  
        fmt.format("%tI:%tM", cal, cal);  
        System.out.println(fmt);  
    }  
}
```

9:14

```
import java.util.Calendar;  
import java.util.Formatter;
```

```
public class Main {  
  public static void main(String args[]) {  
    Formatter fmt = new Formatter();  
    Calendar cal = Calendar.getInstance();  
    fmt.format("Today is day %te of %<tB, %<tY", cal);  
    System.out.println(fmt);  
  }  
}
```

Today is day 7 of June, 2006