



Berner Fachhochschule  
Haute école spécialisée bernoise  
Bern University of Applied Sciences

# Object-Oriented Programming 2

Internationalization – I18n

Prof. Dr. Annett Laube

Adapted by Andres Scheidegger

FS2018

# Outline – I18n

- ▶ What is I18n?
- ▶ `java.util.Locale`
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ Sorting

# Outline – I18n

- ▶ What is I18n?
- ▶ `java.util.Locale`
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ Sorting

# Let's look at an example:

The screenshot shows a GUI window titled "Ein Beispiel eines GUI" with a blue title bar. The main content area has a grey header with the text "Gemeinde Oberkrättligen" and "Einwohnerregister". Below the header is a form with various input fields and labels. Blue arrows point from text labels to specific elements in the form:

- Pictures**: Points to a yellow and black shield icon.
- Texts**: Points to the "Name:" label.
- Text values**: Points to the "Geschlecht:" label and the "Geburtsdatum:" label.
- Times**: Points to the "Uhrzeit:" label.
- Dates**: Points to the "Geburtsdatum:" label.
- Special formats**: Points to the "AHV-Nr.:" label.
- Phone numbers**: Points to the "Tel-Nr.:" label.
- ZIP numbers**: Points to the "PLZ:" label.
- Currency representation**: Points to the "Steuereinkommen:" label.
- Titles / honorifics**: Points to the "Titel:" label.
- Messages**: Points to the "Kein Name eingegeben" message at the bottom left.

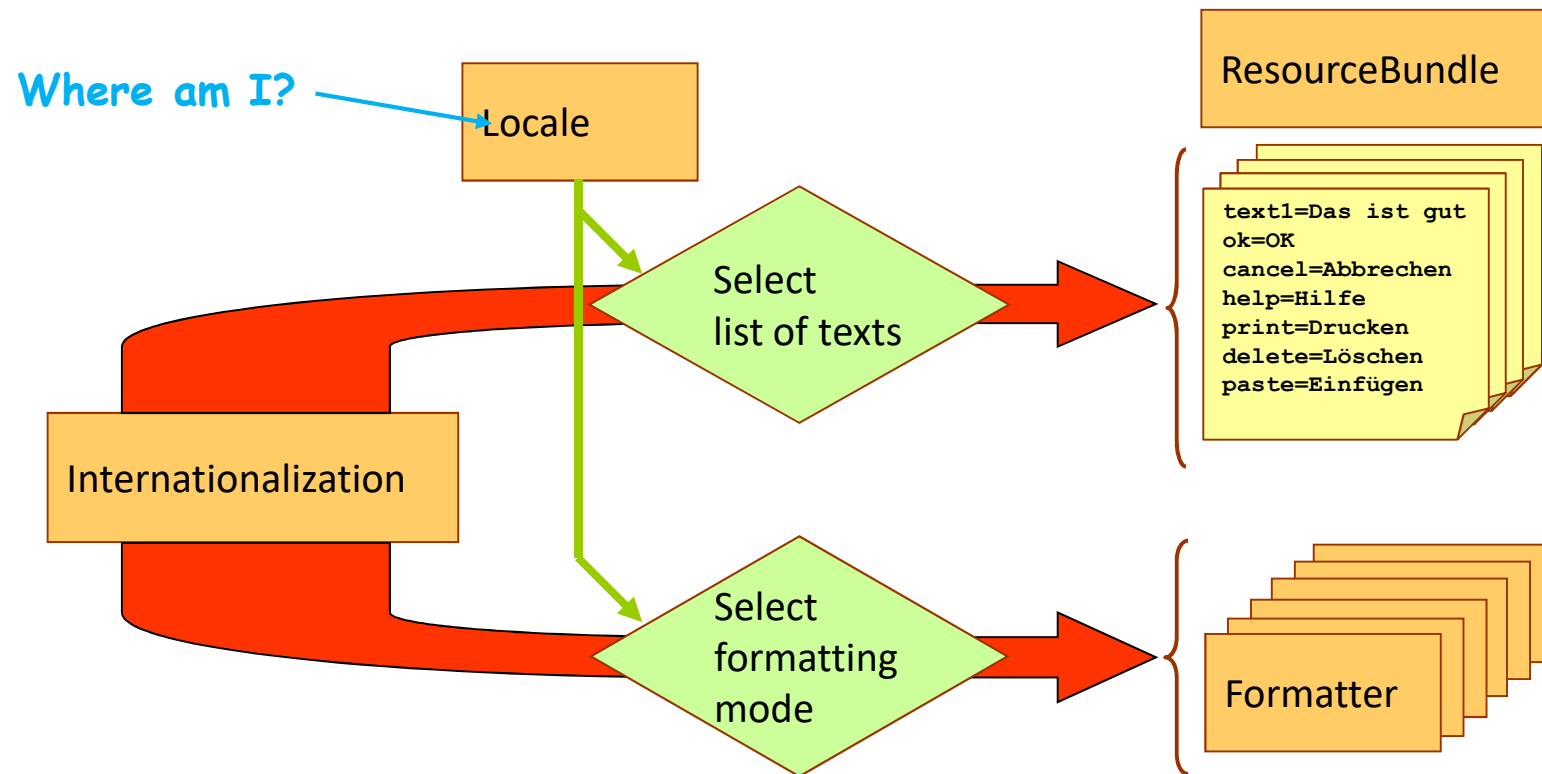
The form fields include:

- Name: (text input)
- Vorname: (text input)
- Geschlecht: (dropdown menu with "m" selected)
- Vorname 2: (text input)
- Vorname 3: (text input)
- Geburtsdatum: (text input)
- Uhrzeit: (text input)
- Zivilstand: (dropdown menu with "ledig" selected)
- Kinder: (text input)
- AHV-Nr.: (text input)
- Wohnadresse: (text input)
- Strasse: (text input)
- Nr.: (text input)
- Ort: (text input)
- PLZ: (text input)
- Tel-Nr.: (text input)
- Beruf: (text input)
- Titel: (text input)
- Steuereinkommen: (text input)

Buttons at the bottom right: "Hilfe", "Speichern", "Abbrechen".

# Internationalization

## The Principles of Internationalization



# Internationalization

- ▶ Where can I get the information about in which country I am, which language I have to use etc ?
  - ▶ from `java.util.Locale`
- ▶ Do I have to write all specific adaptations for text, pictures etc. myself ?
  - ▶ No, there are plenty of ready-made classes, mostly in `java.util.*` and `java.text.*`, such as
    - ▶ `ResourceBundle`, `PropertyResourceBundle`, `ListResourceBundle`
    - ▶ `Format`, `DateFormat`, `MessageFormat`, `NumberFormat`
    - ▶ `Calendar`, `GregorianCalendar`
    - ▶ `Collator` (for sorting/searching)
    - ▶ etc.

# Outline

- ▶ What is I18n?
- ▶ **`java.util.Locale`**
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ Sorting

# What is a Locale ?

Where do I get the Locale from ?

- ▶ A Locale object contains the **country** and the **language** of the host machine. It may specify other characteristics, too, such as a vendor or browser indication.
- ▶ The JVM sets up a Locale containing these data which is then called Default Locale.
- ▶ You may change the contents of the Locale or preset a new default Locale (for your JVM only!)

```
Locale locale = Locale.getDefault() ;
```



# Internationalization

- ▶ How do I use a Locale object?
- ▶ By asking it explicitly its country or language

```
Locale locale = Locale.getDefault();  
String country = locale.getCountry();  
String language = locale.getLanguage();  
String variant = locale.getVariant();
```

```
// or lists of all known countries or languages  
String[] countries = locale.getISOCountries();  
String[] languages = locale.getISOLanguages();
```

- ▶ By passing the Locale to locale-sensitive objects, e.g. to objects of ResourceBundle, Format, Collator etc.

## Exercise: My Locale

Write a small program that displays the Locale in your environment. Find out what is the default Locale in your system.

# Outline

- ▶ What is l18n?
- ▶ `java.util.Locale`
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ Sorting

# Internationalization for Texts/Messages

- ▶ **ResourceBundle** is a specialized Collection to store locale-dependent information
- ▶ It offers two relevant features for internationalization:
  - ▶ **ResourceBundle** consists of a list of {key, value} pairs of items (values may be texts or any kind of objects)
  - ▶ **ResourceBundle** allows retrieval of a Locale-specific list; if there is no list exactly matching the Locale, it provides the best-possible fit
  - ▶ May be backed by property files for different locales

```
# This is the UITexts_fr properties file
computer = ordinateur
disk = disque dur
monitor = écran
keyboard = clavier
```

# Internationalization for Texts/Messages

**ResourceBundle** = set of classes sharing the same base name:

```
UITexts  
UITexts_de  
UITexts_en_GB  
UITexts_fr_CA_UNIX
```

**However** you don't need to implement classes. You just provide property files with the same name as the class and the file extension **.property**

**Selecting** a ResourceBundle is done as follows:

```
Locale current = new Locale("fr", "CA", "UNIX");  
ResourceBundle introLabels =  
    ResourceBundle.getBundle("UITexts", current);
```

# Exercise: ResourceBundle

Run the **ResourceBundleDemo.java** on your machine.

Insert another language in the supported locals and provide a resource bundle for it.

Look up the correct notation on <http://www.localeplanet.com/java/>.



# Internationalization for Texts/Messages

- ▶ Texts and objects are requested from a ResourceBundle using `getString()`
- ▶ Thus, an application using localized items looks as follows:

```
Locale current = Locale.getDefault();
ResourceBundle uiTexts = ResourceBundle.getBundle("UITexts", current);
//...

// get a text from the resource bundle
statusLine.setText(uiTexts.getString("fileNotFound"));
```



**Attention**  
MissingResourceException  
could be thrown

# Internationalization for Texts/Messages

- ▶ To avoid **MissingResourceException** use a private method, e.g.

```
private String getI18nString(String key) {  
    try {  
        return bundle.getString(key);  
    } catch (MissingResourceException e) {  
        System.err.println("Missing key " + key + " in " +  
            bundle.getBaseBundleName());  
        return "! " + key + " !";  
    }  
}
```



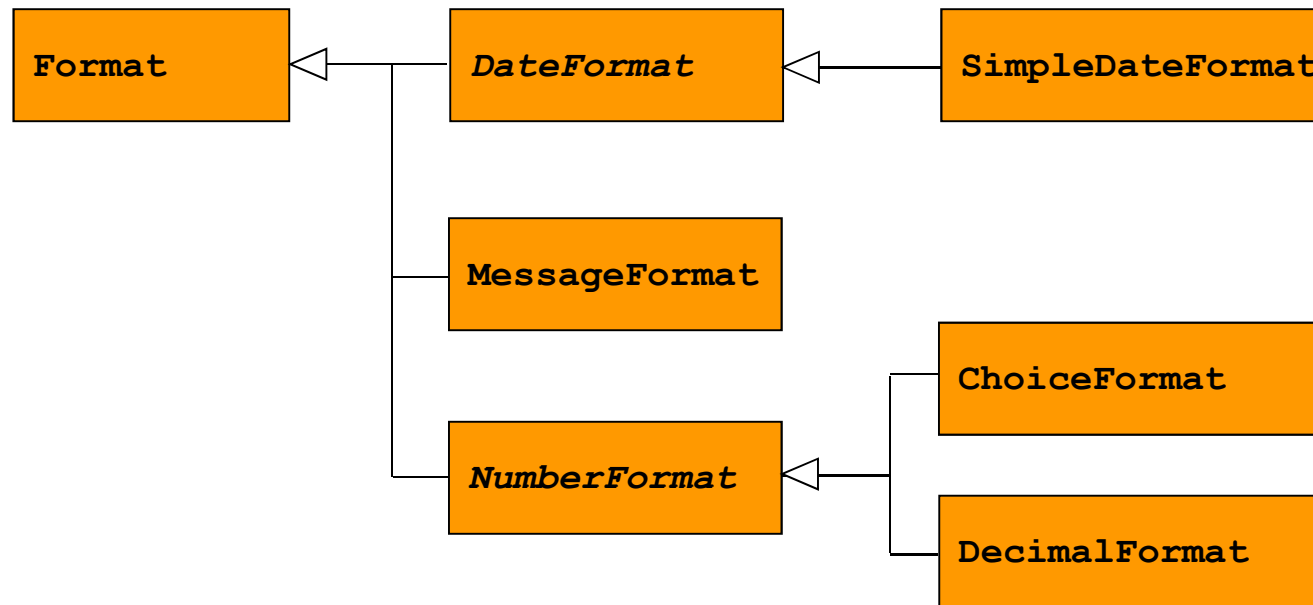
# Outline

- ▶ What is l18n?
- ▶ `java.util.Locale`
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ Sorting

# Internationalized Formatting

The **Format** class and its descendants is used to do locale-dependent formatting of texts, dates/times and numbers/currencies

**Format** offers the following subclasses for internationalization:



# Internationalized Formatting

- ▶ Use the static `getXXXInstance()` method of the `Format` subclasses to create an instance that can format objects in your program
  - ▶ E.g to create a formatter for time.

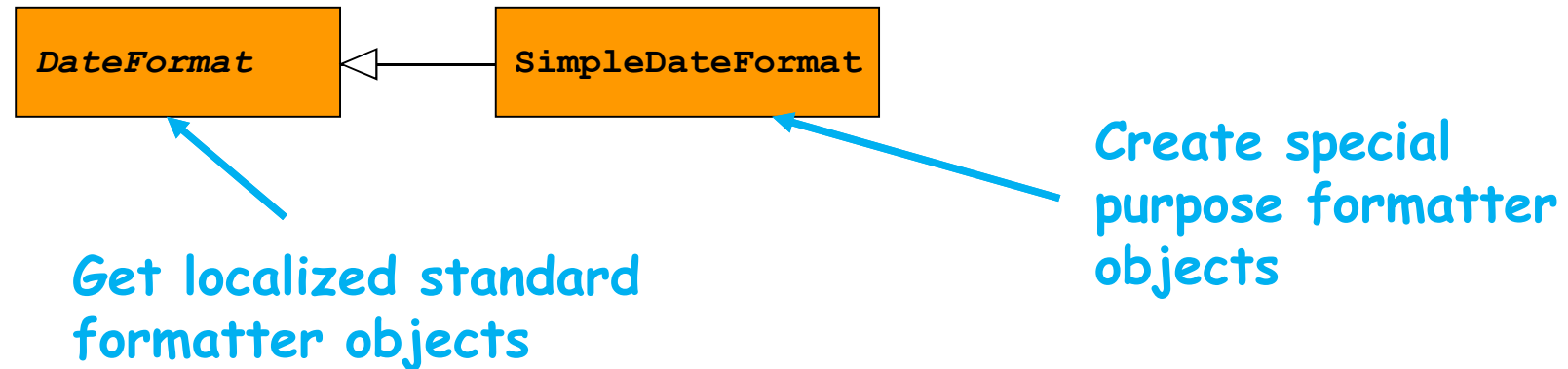
```
DateFormat timeFormatter =  
    DateFormat.getTimeInstance(DateFormat.MEDIUM, Locale.GERMAN);
```

- ▶ Almost all `Format` subclasses offer a

```
Locale[] getAvailableLocales()
```

method returning an array of those Locales which are supported by this class.

# Internationalized Date/Time Formatting



- ▶ Although `DateFormat` is an abstract class, it has static factory methods to create formatter objects for date, time, date&time and time zone
- ▶ The `SimpleDateFormat` class is used when you have to create a specific format which doesn't correspond to any of the standard representations

# Date/Time Formatting Using Standard Formatters

A **standard formatter** is created by:

```
DateFormat timeFormatter = DateFormat.getInstance(int style, Locale aLocale);
```

The specific item to be formatted;  
may be time, date or both



The formatting style to be used  
→ next slide

The Locale to be used

► ... and used:

```
DateFormat timeFormatter =
```

```
    DateFormat.getInstance(DateFormat.MEDIUM, Locale.GERMAN);
```

```
String myText = timeFormatter.format(myDate); // or
```

```
StringBuffer myTextBuffer = timeFormatter.format(myDate, buffer,  
fieldPosition);
```

# Date/Time Formatting Using Standard Formatters

## Examples (DateTimeFormatter):

<u>Style</u>	<u>Locale</u>	<u>Format</u>
<b>SHORT</b>	France	18/10/04 13:30
	Germany	18.10.04 13:30
	England	18/10/04 13:30
<b>MEDIUM</b>	France	18 oct. 2004 13:30:00
	Germany	18.01.2004 13:30:00
	England	18-Oct-2004 13:30:00
<b>LONG</b>	France	18 octobre 2004 13:30:00 CET
	Germany	18. Oktober 2004 13:30:00 MEZ
	England	18 October 2004 13:30:00 CET
<b>FULL</b>	France	lundi 18 octobre 2004 13 h 30 CET
	Germany	Montag, 18. Oktober 2004 13.30 Uhr MEZ
	England	Monday, 18 October 2004 13:30:00 o'clock CET

# Exercise: Date and Time Formatter

Write a program that creates the different date and time formats (SHORT, MEDIUM, LONG, FULL) for at least 3 languages.



# Date/Time Formatting Using Standard Formatters

- ▶ A standard formatter can also be used for input parsing:

```
DateFormat dateFormatterFull =  
    DateFormat.getDateInstance(DateFormat.FULL, Locale.GERMAN);  
  
myDate = dateFormatterFull.parse("Mittwoch, 4. April 2018");
```

Can throw a ParseException

- ▶ By default, the formatter also accepts date or time values not adhering to the Locale's representation (*lenient*); this feature may be turned off if required

```
dateFormatterMed.setLenient(false);
```

- ▶ See: `ParseDemo.java`





# Date/Time Formatting Using SimpleDateFormat

The **SimpleDateFormat** class is a parameterizable formatter for special purposes. It is created by specifying a format string when calling its constructor. In the most simple form this is:

```
SimpleDateFormat simpleFormat =  
    new SimpleDateFormat(pattern);
```

where the **pattern** specifies the format to be applied, e.g.

## Date and Time Pattern

"yyyy.MM.dd G 'at' HH:mm:ss z"

"EEE, MMM d, ''yy"

"h:mm a"

"hh 'o'clock' a, zzzz"

"K:mm a, z"

## Result

2001.07.04 AD at 12:08:56 PDT

Wed, Jul 4, '01

12:08 PM

12 o'clock PM, Pacific Daylight Time

0:08 PM, PDT

# Date/Time Formatting Using SimpleDateFormat

<u>Sb</u>	<u>Meaning</u>	<u>Pres.</u>	<u>Ex.</u>	<u>Sb</u>	<u>Meaning</u>	<u>Pres.</u>	<u>Ex.</u>
G	Era designator	Text	AD	k	Hour in day (1-24)	Num.	24
y	Year	Year	1996	K	Hour in AM/PM (0-11)	Num.	0
M	Month in year	Month	July	h	Hour in AM/PM (1-12)	Num.	12
w	Week in year	Num.	27	m	Minute in hour	Num.	30
W	Week in month	Num.	2	s	Second in minute	Num.	55
D	Day in year	Num.	189	S	Millisecond	Num.	978
d	Day in month	Num.	10	z	Time zone	General time zone	GMT
F	Day of week in month	Num.	2	Z	Time zone	RFC 822 time zone	-0800
E	Day in week	Text	Friday	'	Escape for text	Delim.	'at'
a	AM/PM marker	Text	PM	"	Single quote	Literal	o'clock
H	Hour in day (0-23)	Num.	0				

# Doing Arithmetics with Date/Time

Sometimes you have to do arithmetics with dates or times.

For this you may use the Calendar classes. Java provides the `GregorianCalendar` only, but there are many others available as Open Source.

```
Calendar rightNow = Calendar.getInstance();  
rightNow.setDate(now);  
rightNow.add(Calendar.DAY_OF_MONTH, -5);
```

[Further explanations on the Java API documentation](#)

# New Date and Time API



- ▶ Human Time
  - ▶ local date/time: `LocalDate`
  - ▶ Zoned time: `ZonedDateTime`
- ▶ Package `java.time.*`
- ▶ Examples `LocalDate`

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

LocalDate einsteinBirthday = LocalDate.of(1879, Month.MARCH, 14);

LocalDate lastDayinFebruar = einsteinBirthday.minusMonths(1)
    .with(TemporalAdjusters.lastDayOfMonth());

LocalDate programmersDay = LocalDate.of(2015, 1, 1).plusDays(256);
```

# New Date and Time API



## ► Examples **LocalTime**

```
LocalTime atTen = LocalTime.of(10, 00);  
  
LocalTime tenFifteen = atTen.plusMinutes(15);  
  
LocalTime breakfastTime = tenFifteen.minusHours(2);
```

## ► **LocalDateTime** combines **LocalDate** and **LocalTime**

```
LocalDateTime jdk8Release = LocalDateTime.of(2014, 3, 18, 8, 30);
```

# Formatting new Date and Time



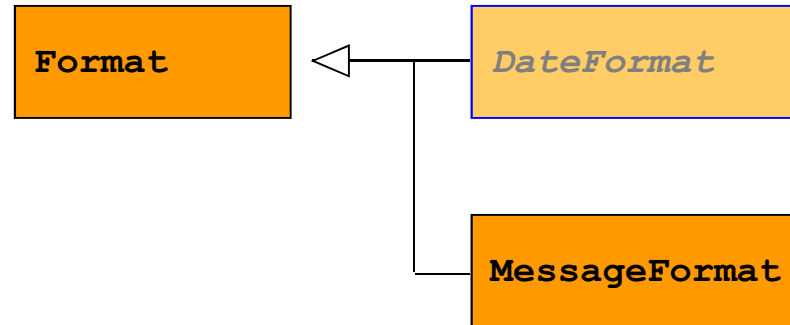
```
Locale currentLocale = new Locale("en","us");
LocalDate date = LocalDate.now();

DateTimeFormatter formatter =
    DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL).withLocale(currentLocale);
System.out.println(formatter.format(date));
// or alternatively
System.out.println(date.format(formatter));

LocalTime time = LocalTime.now();
formatter =
    DateTimeFormatter.ofLocalizedTime(FormatStyle.MEDIUM).withLocale(currentLocale);
System.out.println(formatter.format(time));

LocalDateTime dateTime = LocalDateTime.now();
formatter = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG,FormatStyle.SHORT)
    .withLocale(currentLocale);
System.out.println(formatter.format(dateTime));
```

# Message Formatting



MessageFormat helps you to format complex messages such as

*At 12:30 PM on Jul 3, 2053, there was a gravitation anomaly on planet 7.*

MessageFormat is not an abstract class, and it doesn't have static factory methods to create formatter objects. Instead, it is parameterized at creation time by the constructor.

**MessageFormat** cannot be given a Locale, i.e. you have to create a class per Locale yourself.

# Message Formatting

This message

At 12:30 PM on Jul 3, 2053, there was a gravitation anomaly on planet 7.

was obtained by

```
String pattern =  
    "At {1,time} on {1,date}, there was {2} on planet {0,number,integer}."  
  
Object[] arguments = {  
    new Integer(7), new Date(System.currentTimeMillis()), "a gravitation anomaly"  
};  
  
String result = MessageFormat.format(pattern, arguments);
```



Also possible to pass as  
optional parameters

Further explanations on the [Java API documentation](#)



# Message Formatting

MessageFormat uses patterns of the following form:

*MessageFormatPattern:*

*String*

*MessageFormatPattern FormatElement String*

*FormatElement:*

*{ ArgumentIndex }*

*{ ArgumentIndex , FormatType }*

*{ ArgumentIndex , FormatType , FormatStyle }*

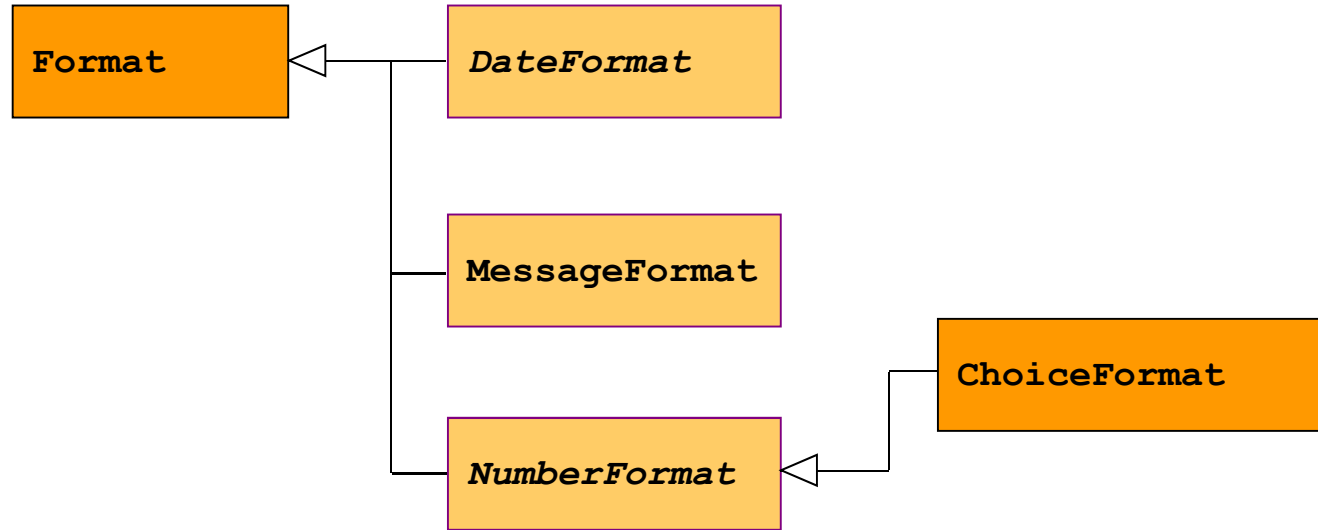
*FormatType: one of*

*number date time choice*

*FormatStyle:*

*short medium long full integer currency percent SubformatPattern*

# Advanced Message Formatting using ChoiceFormat



ChoiceFormat helps you to reflect different grammatical forms such as

**The directory ABCD contains no files.**

**The directory ABCD contains one file.**

**The directory ABCD contains 128 files.**

# Message Formatting

**MessageFormat** and **ChoiceFormat** have proven to be very „tricky“ classes!!!  
Read the API doc carefully and do in-depth unit testing!

**MessageFormat** and **ChoiceFormat** are not thread-safe!  
If you intend to use these objects from different threads, protect them by synchronization!

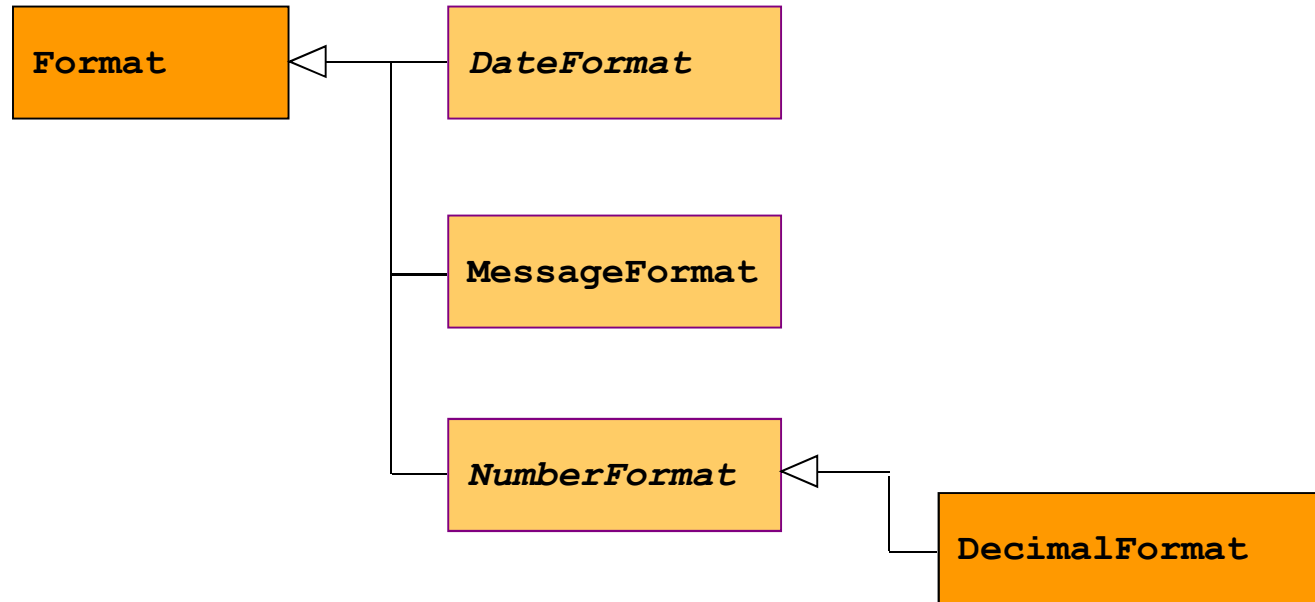
Especially, **MessageFormat** can be nicely used with Resource Bundles .

## Exercise: ChoiceFormat

Analyse the program FormatDemo.java.



# Formatting Numbers and Currencies



**DecimalFormat** provides you with conversion functions for numbers and currencies as well as a special representation for percentages.

As with **DateFormat**, actual **DecimalFormat** instances are created using the appropriate factory method of **NumberFormat**, such as **getCurrencyInstance ()** , **getNumberInstance ()** etc.

# Number/Currency Formatting Using NumberFormat


A standard localized formatter is created by:

```
NumberFormat someFormatter = NumberFormat.getNumberInstance(Locale aLocale);
```



The specific item to be formatted;  
may be

`getCurrencyInstance(...)`  
`getIntegerInstance(...)`  
`getNumberInstance(...)`  
`getPercentInstance(...)`



The Locale to be used  
(or no parameter →  
using the default Locale)

Number representation is done according to the specific rules of the applicable Locale.  
If you need a specific format, use `DecimalFormat` with an explicit setup!

# Number/Currency Formatting Using NumberFormat

A standard formatter can also be used for input parsing:

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

```
Number number = format.parse("1234.5678", new ParsePosition(2));
```

will return Number(34.5678)

By default, the formatter also accepts numbers not adhering to the Locale's representation.

Be careful with too large numbers:

- If possible these are returned as **Long** objects – if too large, as a **Double** truncated at the low end.
- If you explicitly use **setParseBigDecimal()**, values will be returned as **BigDecimal** objects.

## Exercise: NumberFormat

Analyse the program `NumberFormatDemo.java`.





# Number Formatting Using DecimalFormat

As with **SimpleDateFormat**, **DecimalFormat** is used when special formatting is required.

A formatting pattern is described by the following characters:

<u>char</u>	<u>Meaning</u>	<u>char</u>	<u>Meaning</u>
0	Digit; will pad if necessary	;	Pos./Neg. pattern separator
#	Digit; zero shows as space	%	*100, shows as percentage
.	Decimal separator	‰	*1000, shows as per mille (\u2030)
-	Negative prefix (minus)	¤	Currency sign (\u00A4 or local)
,	Grouping separator	‘	To quote special char. in
E	Mantissa-Exponent separator		prefix or suffix (see API descr.)

# Outline

- ▶ What is l18n?
- ▶ `java.util.Locale`
- ▶ Culture Dependent Content
- ▶ Formatting
- ▶ **Sorting**

# Sorting

Sorting is done using a comparator telling which of two elements is first / second in a given ordering scheme.

Sorting of text must take into account the peculiarities of the alphabetic sorting for each language.

This is done by a class named **Collator**.

A standard localized collator is created by e.g.:

```
Collator fr_FR_Collator = Collator.getInstance(new Locale("fr", "FR"));
```

```
Collator defaultCollator = Collator.getInstance(); // to use the default Locale
```

# Sorting

`abstract class java.text.Collator implements Comparator<Object>, Cloneable`

Methods:

- ▶ `static Collator getInstance()` Collator for current Locale
- ▶ `static Collator getInstance(Locale desiredLocale)` Collator for given Locale.
- ▶ `abstract int compare(String source, String target)`  
Compares 2 strings. Return value is <0, 0 or >0.
- ▶ `int compare( Object o1, Object o2 )`  
Compares 2 objects. Calls `compare( (String)o1, (String)o2 )`.

# Sorting

Sorting an array of Strings is then done as follows (using a plain BubbleSort algorithm):

```
Collator defaultCollator = Collator.getInstance();
String tmp;
for (int i = 0; i < words.length; i++) {
    for (int j = i + 1; j < words.length; j++) {
        if (defaultCollator.compare(words[i], words[j]) > 0) {
            tmp = words[i];
            words[i] = words[j];
            words[j] = tmp;
        }
    }
}
```

## Exercise: Collater

Analyse the program CollatorDemo.java.



# Sorting

Of course, you may also use the same comparator as an ordering criterion for an ordered set, such as :

```
Comparator myCollator =  
    (Comparator) Collator.getInstance();  
  
TreeSet orderedSet = new TreeSet<String>(myCollator);
```

... which allows you to insert, remove or search your Strings (words/phrases) according to the Locale's ordering.