0.01 + 0.09 != 0.10

A corner case cheat sheet
for Java and JVM languages

by Geoffrey De Smet

Who am I?

- Java developer (graduated in 2003)
- Founder/lead of **OptaPlanner**
  - Leading Open Source Constraint Solver in Java
- Contributor to 23+ Open Source projects

Definition

A ***corner case*** *is a bug*
that only manifests itself
with a specific combination of input values.

A **corner case cheat sheet** is a list of input values
that tend to trigger corner cases,
useful for automated and manual testing.

Numbers

Average of 2 numbers

```
public int average(int a, int b) {
    return (a + b) / 2;
}
```

Input

```
average(1000, 2000)
1500

average(1000000, 2000000)
1500000

average(1000000000, 2000000000) // Corner case
-647483648 // Overflow on a + b
```

Ariane 5 (1996)

Lift off

Overflow

Integer overflow

Problem

```
public int average(int a, int b) {
    return (a + b) / 2;
}

public int average(int a, int b) {
    double c = (a + b) / 2.0;
    return (int) c;
}
```

Solution

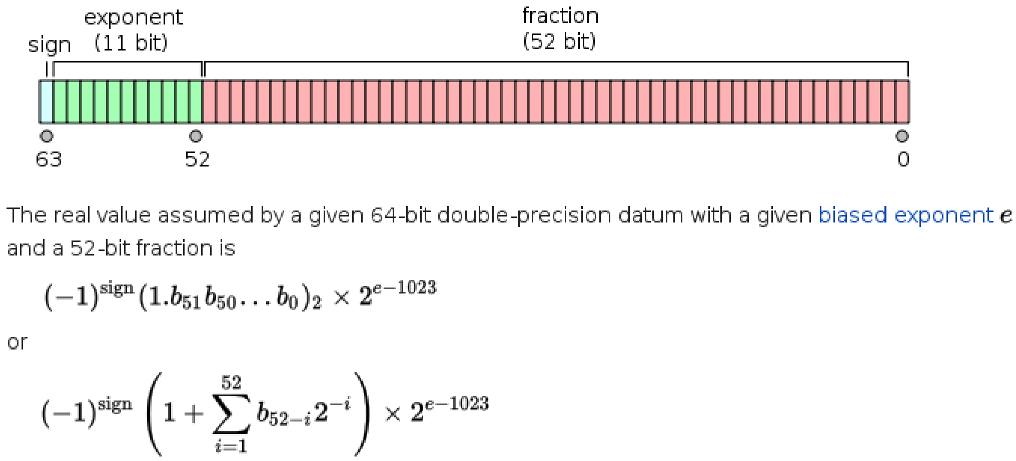
```
public int average(int a, int b) {
    long c = (((long) a) + b) / 2L;
    return (int) c;
}

average(1000000000, 2000000000)
1500000000
```

Sum of floating point numbers

Input

```
1.0 + 9.0
10.0

0.1 + 0.9
1.0

0.01 + 0.09 // Corner case
0.09999999999999999 // Compound rounding error
```

Double precision floating point (Wikipedia)



The real value assumed by a given 64-bit double-precision datum with a given biased exponent e and a 52-bit fraction is

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\ldots b_0)_2 \times 2^{e-1023}$$

or

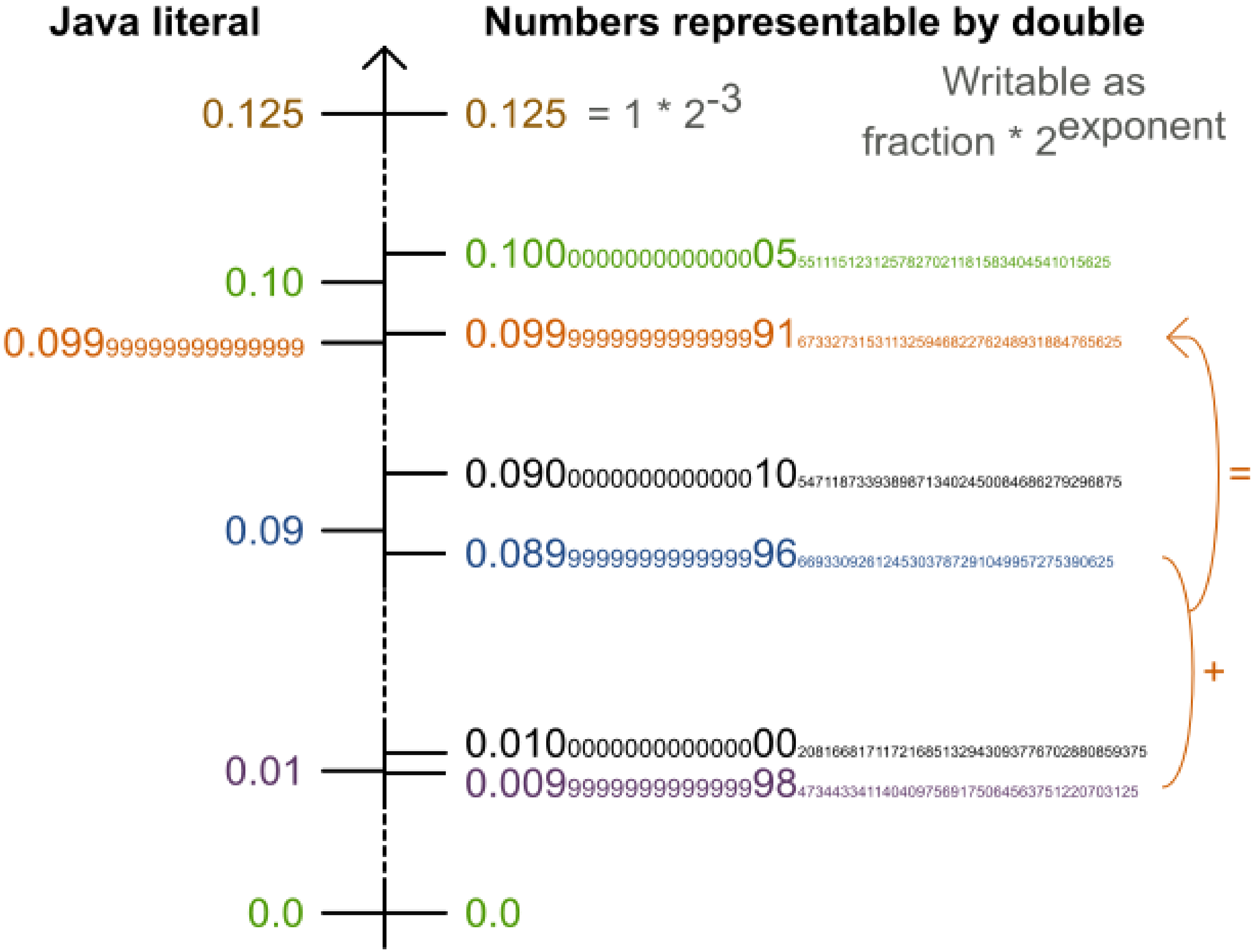
$$(-1)^{\text{sign}}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023}$$

Source: https://en.wikipedia.org/wiki/Double-precision_floating-point_format

Translation: every double value is
an integer divided by a multiplication of 2

Double arithmetic compound rounding error

What really happens when we sum 0.01 and 0.09 with doubles in Java.



Java literal

Numbers representable by double

Writable as
fraction $* 2^{exponent}$

0.125

0.125 $= 1 * 2^{-3}$

0.1000000000000000055511151231257827021181583404541015625

0.10

0.099999999999999999

0.09999999999999999167332731531132594682276248931884765625

=

0.09000000000000001054711873393898713402450084686279296875

0.09

0.0899999999999999966693309261245303787291049957275390625

+

0.01000000000000000020816681711721685132943093776702880859375

0.01

0.0099999999999999984734433411404097569175064563751220703125

0.0

0.0

Failure rate

Sum of 2 numbers between 0.00 and 1.00

```
  0.01 + 0.05 != 0.06
  0.01 + 0.06 != 0.07
  0.01 + 0.09 != 0.10
  0.01 + 0.14 != 0.15
  0.01 + 0.17 != 0.18
  0.01 + 0.20 != 0.21
  0.01 + 0.23 != 0.24
  0.01 + 0.28 != 0.29
  ...
  0.99 + 0.87 != 1.86
  0.99 + 0.90 != 1.89
  0.99 + 0.92 != 1.91

2106 failures (21%) out of 10000 sums
```

Failure rate: 21%

Compound rounding error

Problem

```
public double sum(double a, double b) {
    return a + b;
}
```

Solution

```
public BigDecimal sum(BigDecimal a, BigDecimal b) {
    return a.add(b);
}

sum(new BigDecimal("0.01"), new BigDecimal("0.09"))
0.10
```

or

```
public long sum(long aMillis, long bMillis) {
    return a + b; // Faster than BigDecimal.add()
}

sum(10, 90) // 10 millis is 0.010 and 90 millis is 0.090
100 // 100 millis is 0.100
```

Side effect

Floating point arithmetic is not associative

```
double a = 0.0;
for (int i = 0; i < 1000000; i++) {
    a += 0.03 + 0.02 + 0.01;
    System.out.println(a);
    a -= 0.01 + 0.02 + 0.03;
}

0.060000000000000005
0.06000000000000001
0.06000000000000002
0.060000000000000026
0.06000000000000003
0.06000000000000004
...
0.06000000000069386
0.060000000000693866
0.06000000000069387
0.06000000000069388
0.06000000000069389
```

Patriot Missile Failure (1991)

The small chopping error, when multiplied by the large number giving the time in tenths of a second, led to a significant error.

The Patriot missile battery had been in operation for 100 hours, by which time the system's internal clock had drifted by one-third of a second. Due to the missile's speed this was equivalent to a miss distance of 600 meters.

Long and double are 64-bit

Input

```
double a = 9000L;
9000.0

double a = 9000000000L;
9000000000.0

double a = 9007199254740993L; // Corner case, no casting needed
9007199254740992.0 // Rounding error

double a = 9007199254740992.0;
a == a + 1.0 // Corner case
true // Wrong
```

Arithmetic issues with Java doubles

Applies to almost all other programming languages too.

```
    9007199254740992.0
+                  1.0
----------------------
    9007199254740992.0
```

```
double a = 9007199254740992.0;
// Prints true
System.out.println(a == (a + 1.0));
```

```
  0.01
+ 0.09
---------------------
  0.09999999999999999
```

```
// Prints true
System.out.println(
    0.01 + 0.09 != 0.10);
```

```
  0.01
  0.02
+ 0.03
------
  0.06
```

```
// Prints true
System.out.println(
    (0.01 + 0.02) + 0.03
    != 0.01 + (0.02 + 0.03));
```

```
    9007199254740992.0
+                  3.0
----------------------
    9007199254740996.0
```

```
double a = 9007199254740992.0;
// Prints true
System.out.println(
    a + 3.0 - a == 4.0);
```

```
  0.01
+ 0.05
----------------------
  0.060000000000000005
```

```
// Prints true
System.out.println(
    0.01 + 0.05 != 0.06);
```

```
  0.03
  0.02
+ 0.01
----------------------
  0.060000000000000005
```

```
// Prints true
System.out.println(
    0.01 + 0.02 + 0.03
    != 0.03 + 0.02 + 0.01);
```

Cheat sheet numbers

Expression	**Actual result**
1000000000 + 2000000000	-1294967296
0.01 + 0.09	0.09999999999999999
0.01 + 0.05	0.060000000000000005
0.01 + 0.02 + 0.03	0.06
0.03 + 0.02 + 0.01	0.060000000000000005
(double) 9007199254740993L	9007199254740992.0
9007199254740992.0 + 1.0	9007199254740992.0
9007199254740992.0 + 3.0	9007199254740996.0

Text

Valid name

```
public boolean isValidFirstName(String firstName) {
    return firstName.matches("\w+");
}
```

Input

```
isValidFirstName("Alexander")
true

isValidFirstName("4l3x4nd3r")
false

isValidFirstName("Chloé")) // French name
false // Wrong

isValidFirstName("りく")) // Riku (Japanese name)
false // Wrong
```

Regular expressions for non-english

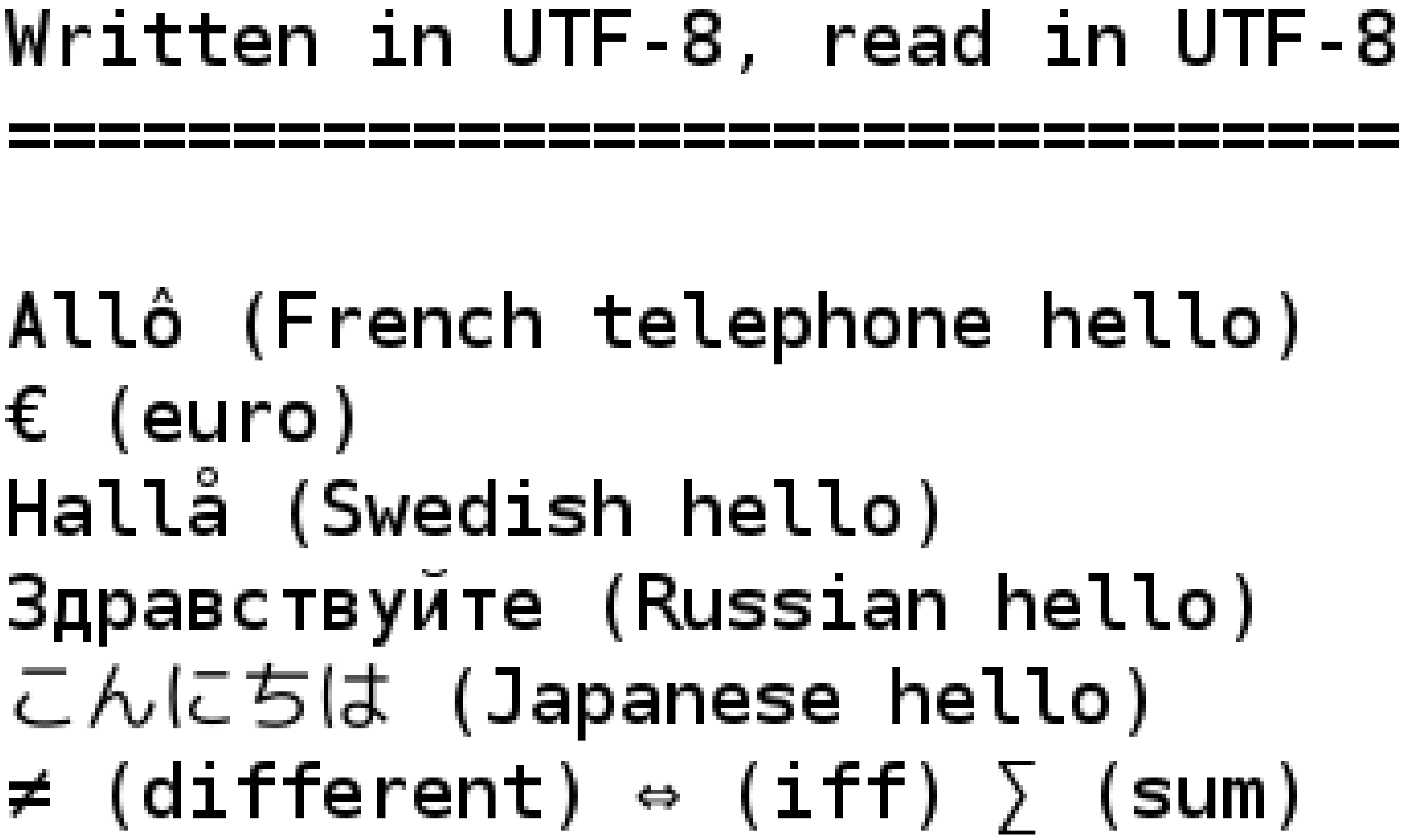
Problem

```
public boolean isValidFirstName(String firstName) {
    return firstName.matches("\w+");
}
```

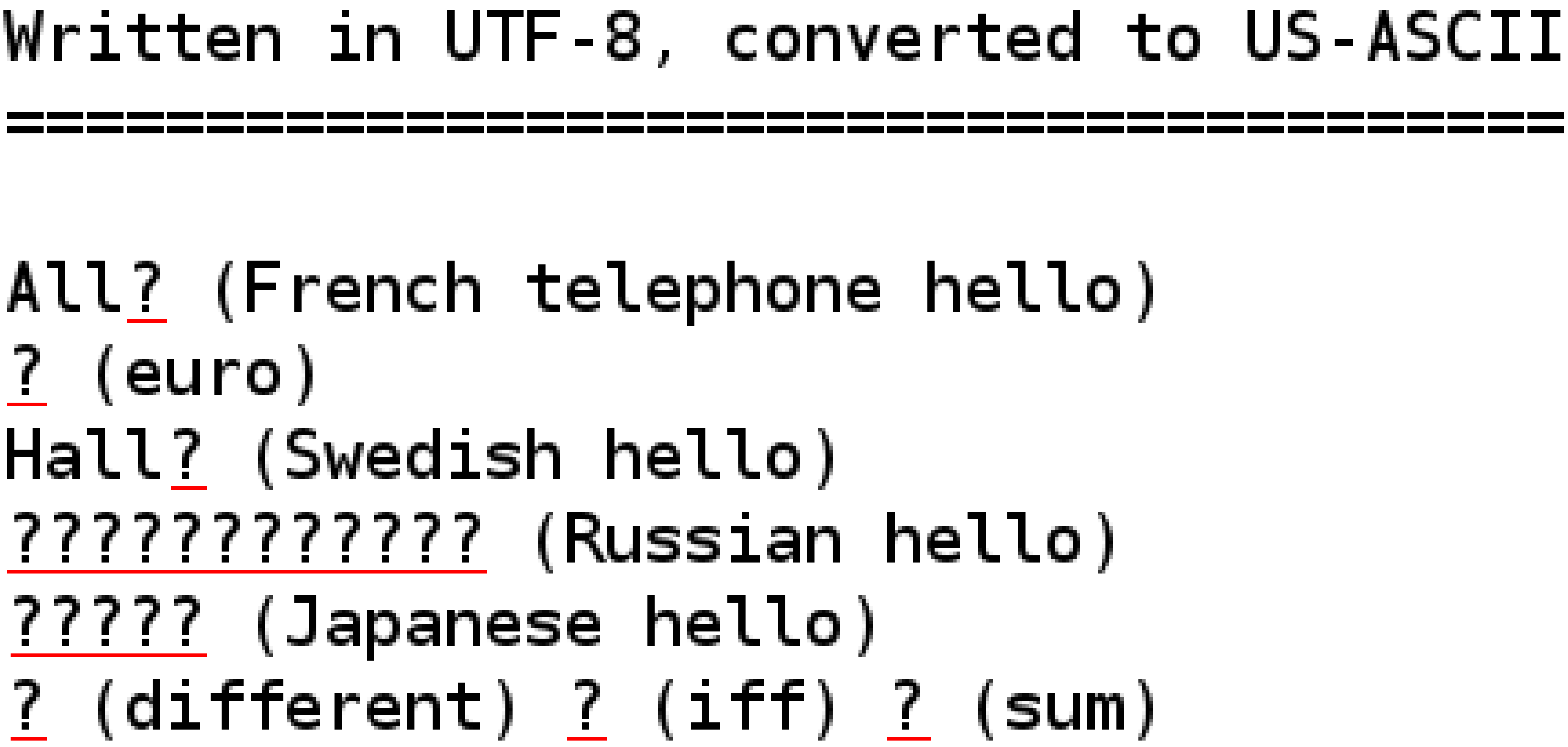
Solution

```
public boolean isValidFirstName(String firstName) {
    return firstName.matches("(?U)\w+");
}

isValidFirstName("Chloé")) // French name Chloe
true

isValidFirstName("りく")) // Japanese name Riku
true
```

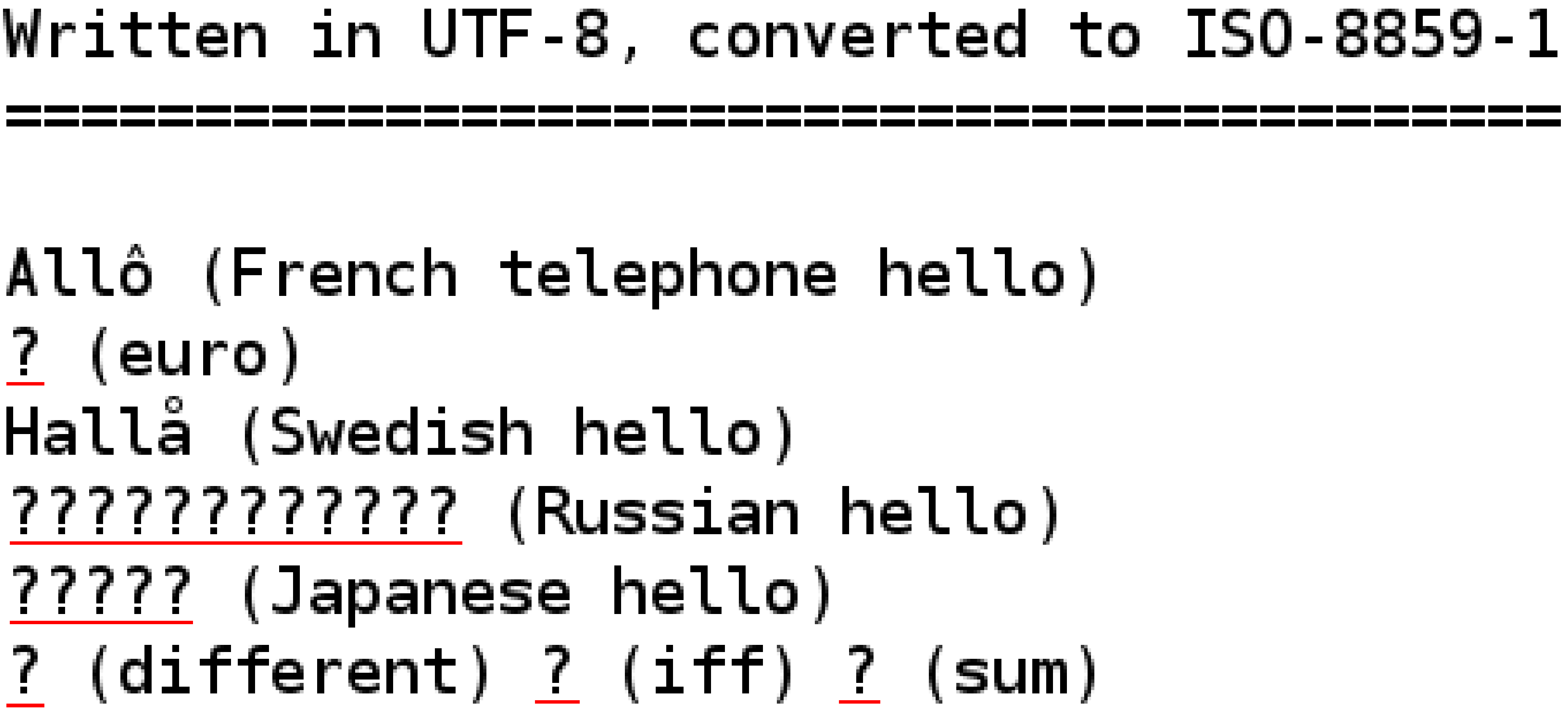
Typical encoding issues



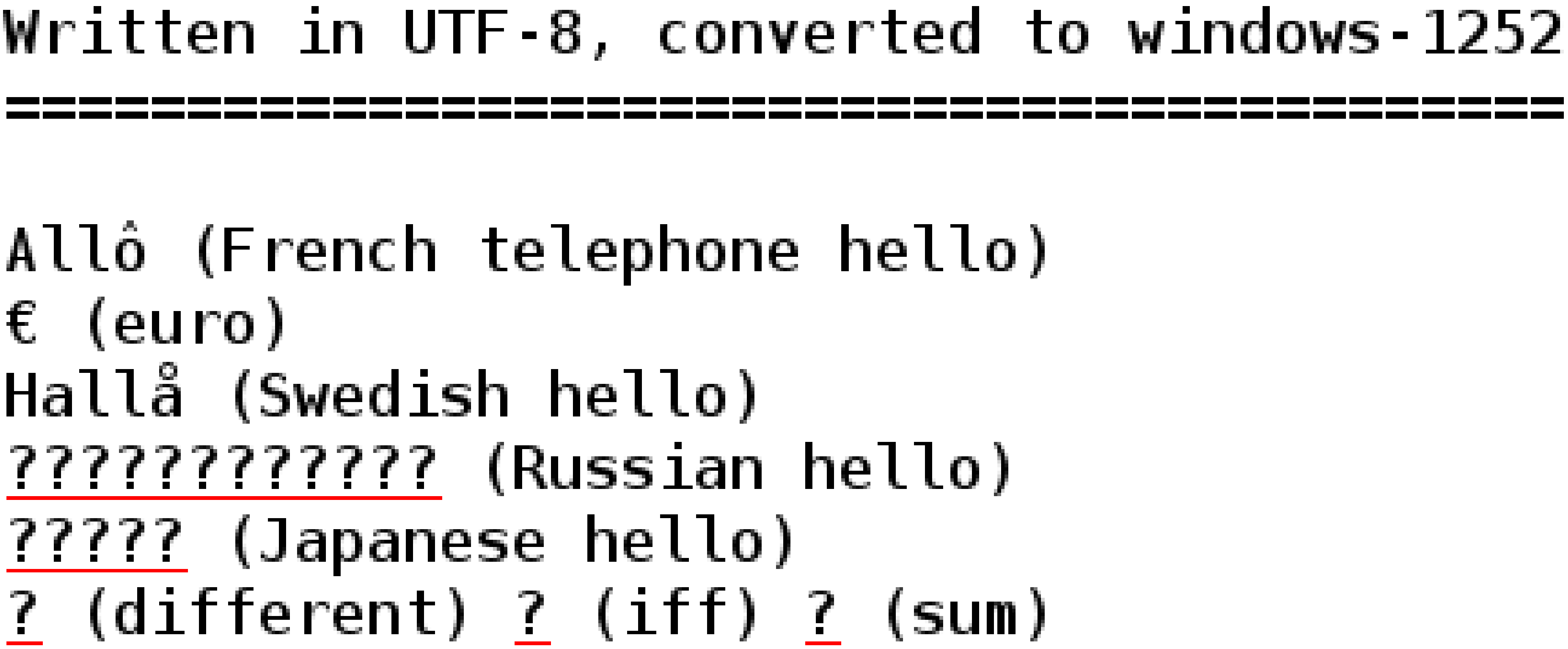
```
Written in UTF-8, read in UTF-8
===============================


Allô (French telephone hello)
€ (euro)
Hallå (Swedish hello)
Здравствуйте (Russian hello)
こんにちは (Japanese hello)
≠ (different) ⇔ (iff) ∑ (sum)
```



```
Written in UTF-8, converted to US-ASCII
=======================================

All? (French telephone hello)
? (euro)
Hall? (Swedish hello)
???????????? (Russian hello)
????? (Japanese hello)
? (different) ? (iff) ? (sum)
```



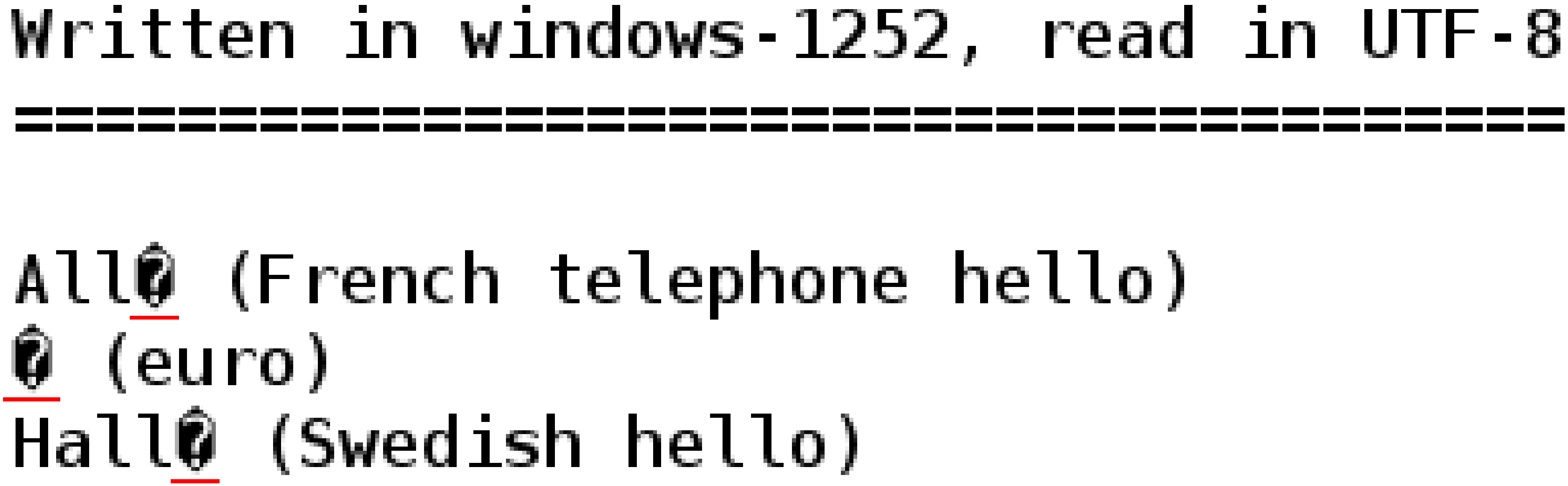
```
Written in UTF-8, converted to ISO-8859-1
=========================================

Allô (French telephone hello)
? (euro)
Hallå (Swedish hello)
???????????? (Russian hello)
????? (Japanese hello)
? (different) ? (iff) ? (sum)
```



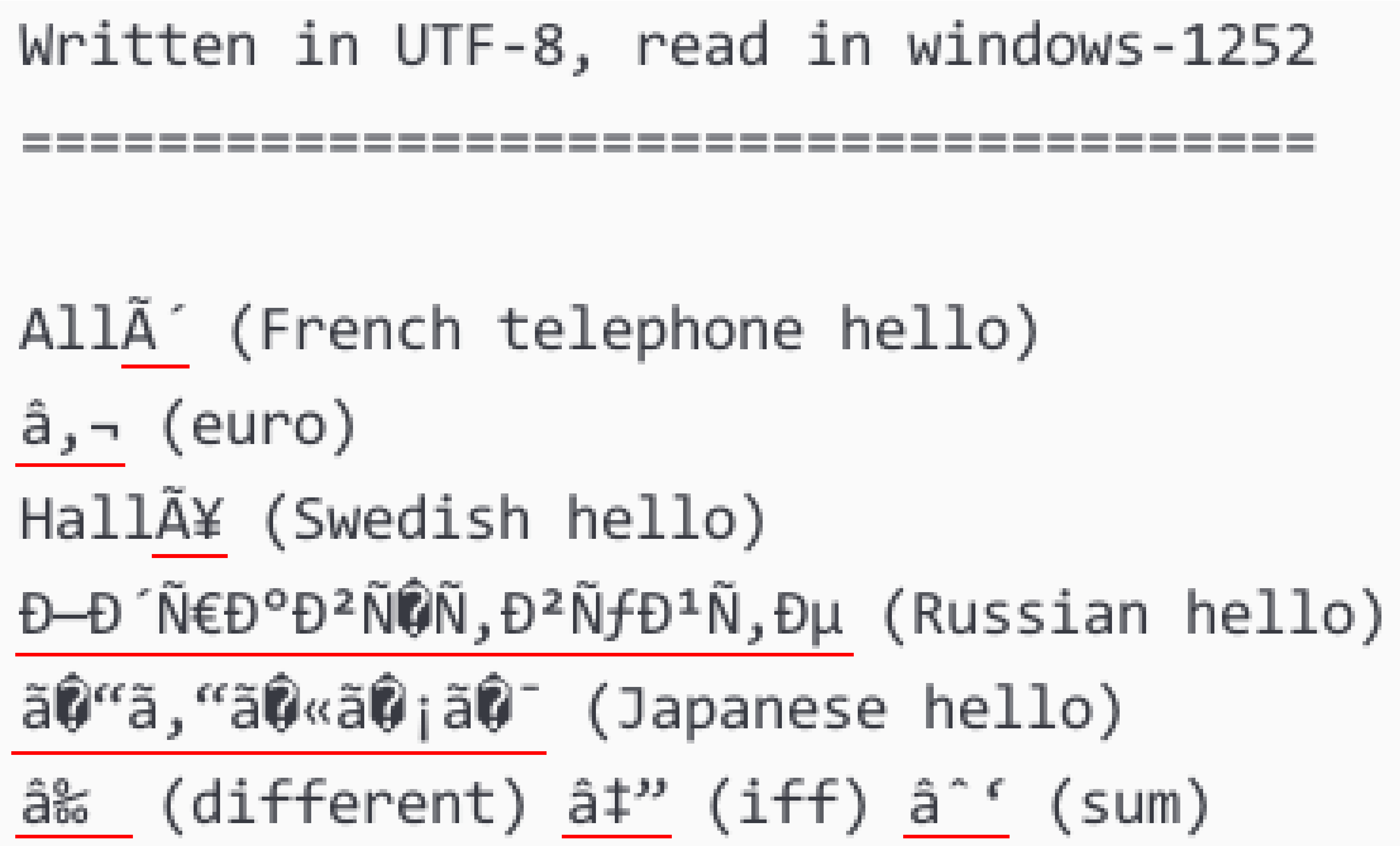
```
Written in UTF-8, converted to windows-1252
===========================================

Allô (French telephone hello)
€ (euro)
Hallå (Swedish hello)
???????????? (Russian hello)
????? (Japanese hello)
? (different) ? (iff) ? (sum)
```

Default encoding

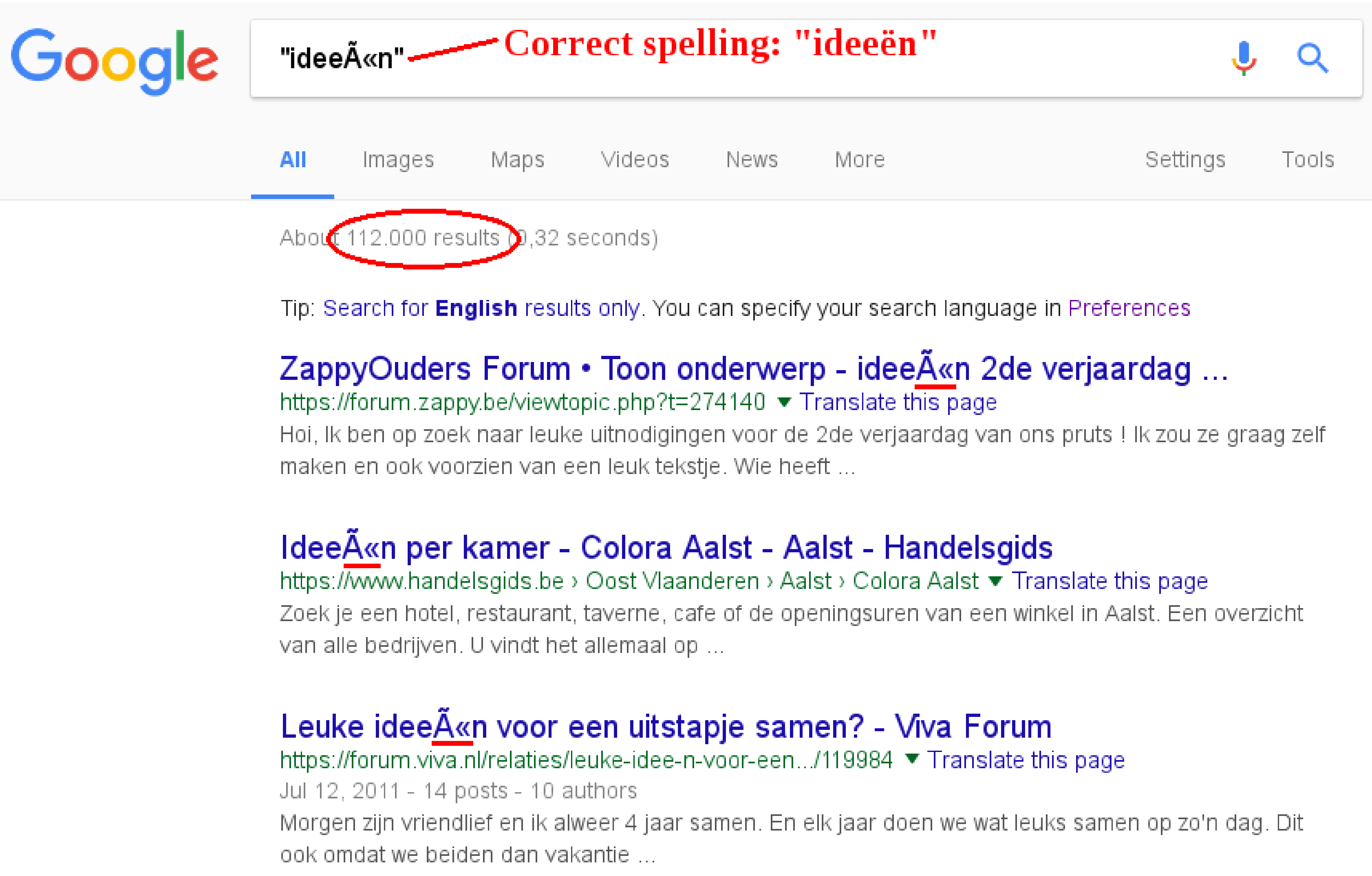
- Linux: UTF-8
- Windows (Western Europe): windows-1252



```
Written in windows-1252, read in UTF-8
======================================

All� (French telephone hello)
� (euro)
Hall� (Swedish hello)
```



```
Written in UTF-8, read in windows-1252
======================================


AllÃ´ (French telephone hello)
â‚¬ (euro)
HallÃ¥ (Swedish hello)
Ð—Ð´Ñ€Ð°Ð²Ñ�Ñ‚Ð²ÑƒÐ¹Ñ‚Ðµ (Russian hello)
ã�“ã‚“ã�«ã�¡ã�¯ (Japanese hello)
â‰  (different) â‡” (iff) âˆ‘ (sum)
```



"ideeÃ«n" Correct spelling: "ideeën"

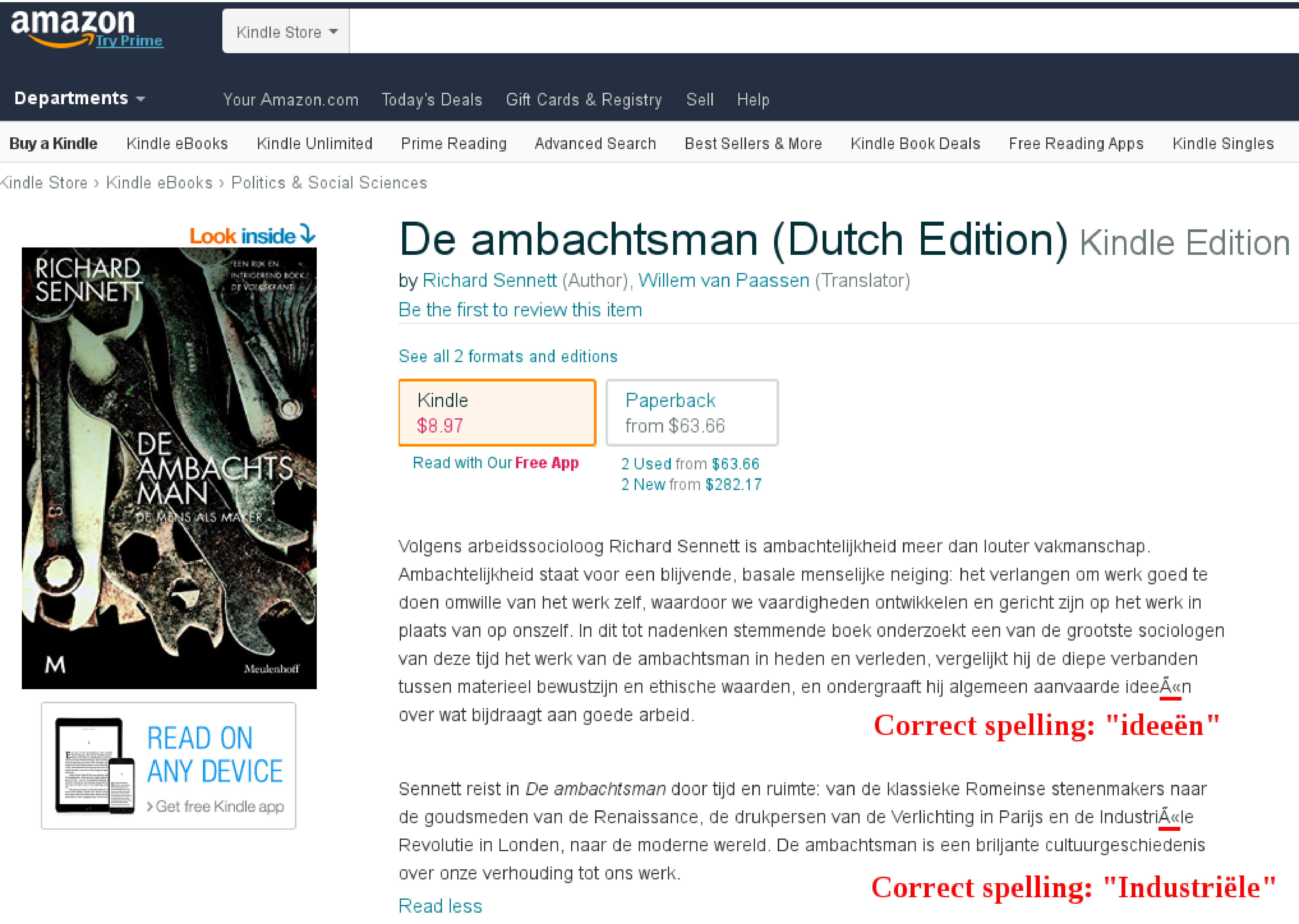
About 112.000 results (0,32 seconds)

Tip: Search for **English** results only. You can specify your search language in Preferences

ZappyOuders Forum • Toon onderwerp - ideeÃ«n 2de verjaardag ...
https://forum.zappy.be/viewtopic.php?t=274140 ▾ Translate this page
Hoi, Ik ben op zoek naar leuke uitnodigingen voor de 2de verjaardag van ons pruts ! Ik zou ze graag zelf maken en ook voorzien van een leuk tekstje. Wie heeft ...

IdeeÃ«n per kamer - Colora Aalst - Aalst - Handelsgids
https://www.handelsgids.be › Oost Vlaanderen › Aalst › Colora Aalst ▾ Translate this page
Zoek je een hotel, restaurant, taverne, cafe of de openingsuren van een winkel in Aalst. Een overzicht van alle bedrijven. U vindt het allemaal op ...

Leuke ideeÃ«n voor een uitstapje samen? - Viva Forum
https://forum.viva.nl/relaties/leuke-idee-n-voor-een.../119984 ▾ Translate this page
Jul 12, 2011 - 14 posts - 10 authors
Morgen zijn vriendlief en ik alweer 4 jaar samen. En elk jaar doen we wat leuks samen op zo'n dag. Dit ook omdat we beiden dan vakantie ...



De ambachtsman (Dutch Edition) Kindle Edition

by Richard Sennett (Author), Willem van Paassen (Translator)

Be the first to review this item

Volgens arbeidssocioloog Richard Sennett is ambachtelijkheid meer dan louter vakmanschap. Ambachtelijkheid staat voor een blijvende, basale menselijke neiging: het verlangen om werk goed te doen omwille van het werk zelf, waardoor we vaardigheden ontwikkelen en gericht zijn op het werk in plaats van op onszelf. In dit tot nadenken stemmende boek onderzoekt een van de grootste sociologen van deze tijd het werk van de ambachtsman in heden en verleden, vergelijkt hij de diepe verbanden tussen materieel bewustzijn en ethische waarden, en ondergraaft hij algemeen aanvaarde ideeÃ«n over wat bijdraagt aan goede arbeid.

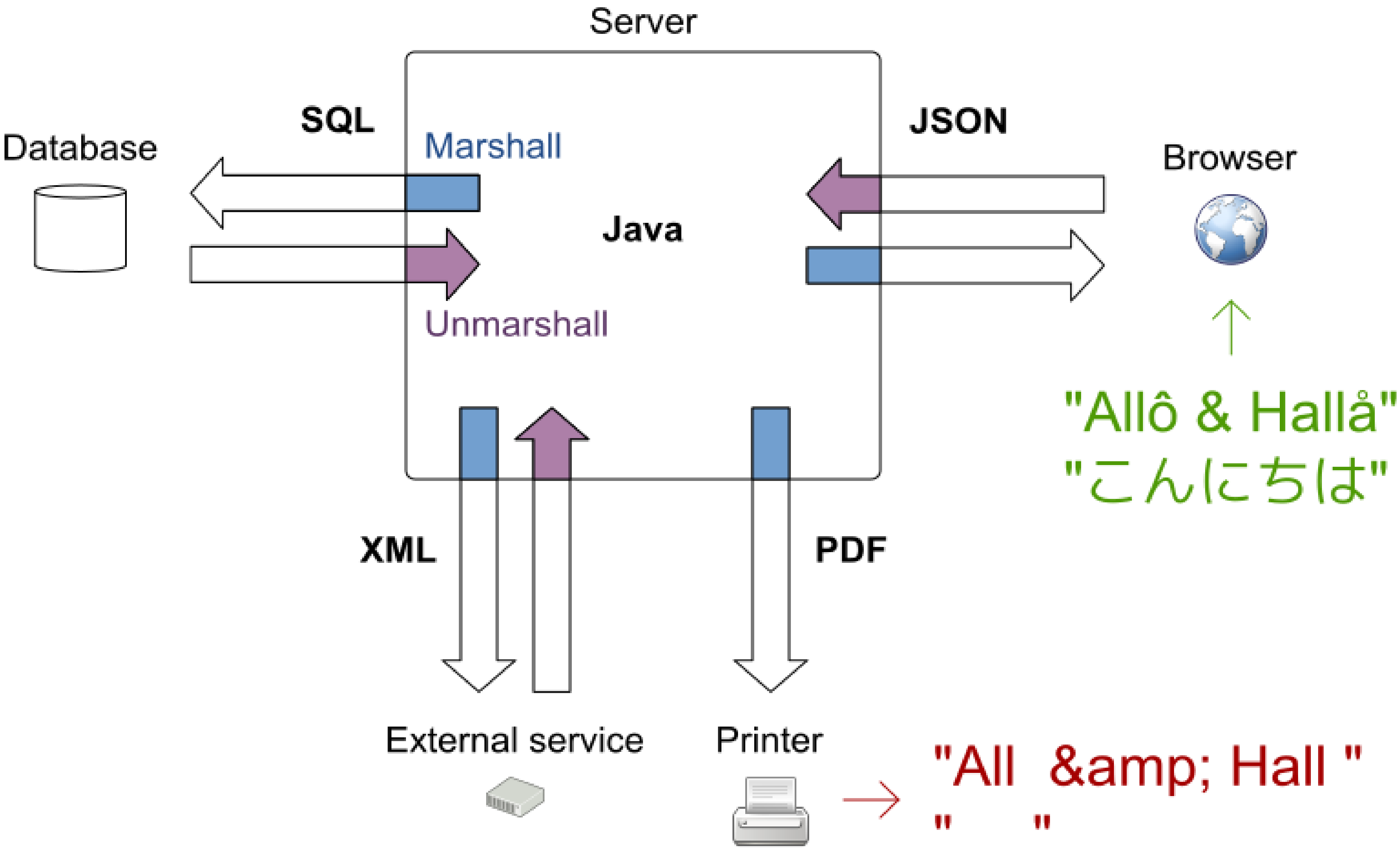
Correct spelling: "ideeën"

Sennett reist in *De ambachtsman* door tijd en ruimte: van de klassieke Romeinse stenenmakers naar de goudsmeden van de Renaissance, de drukpersen van de Verlichting in Parijs en de IndustriÃ«le Revolutie in Londen, naar de moderne wereld. De ambachtsman is een briljante cultuurgeschiedenis over onze verhouding tot ons werk.

Correct spelling: "Industriële"

Data conversions

In a typical webapplication, data is marshalled and unmarshalled in multiple formats.



Server

SQL

Database

Marshall

Java

Unmarshall

JSON

Browser

"Allô & Hallå"
"こんにちは"

XML

PDF

External service

Printer

"All &amp; Hall "
"   "

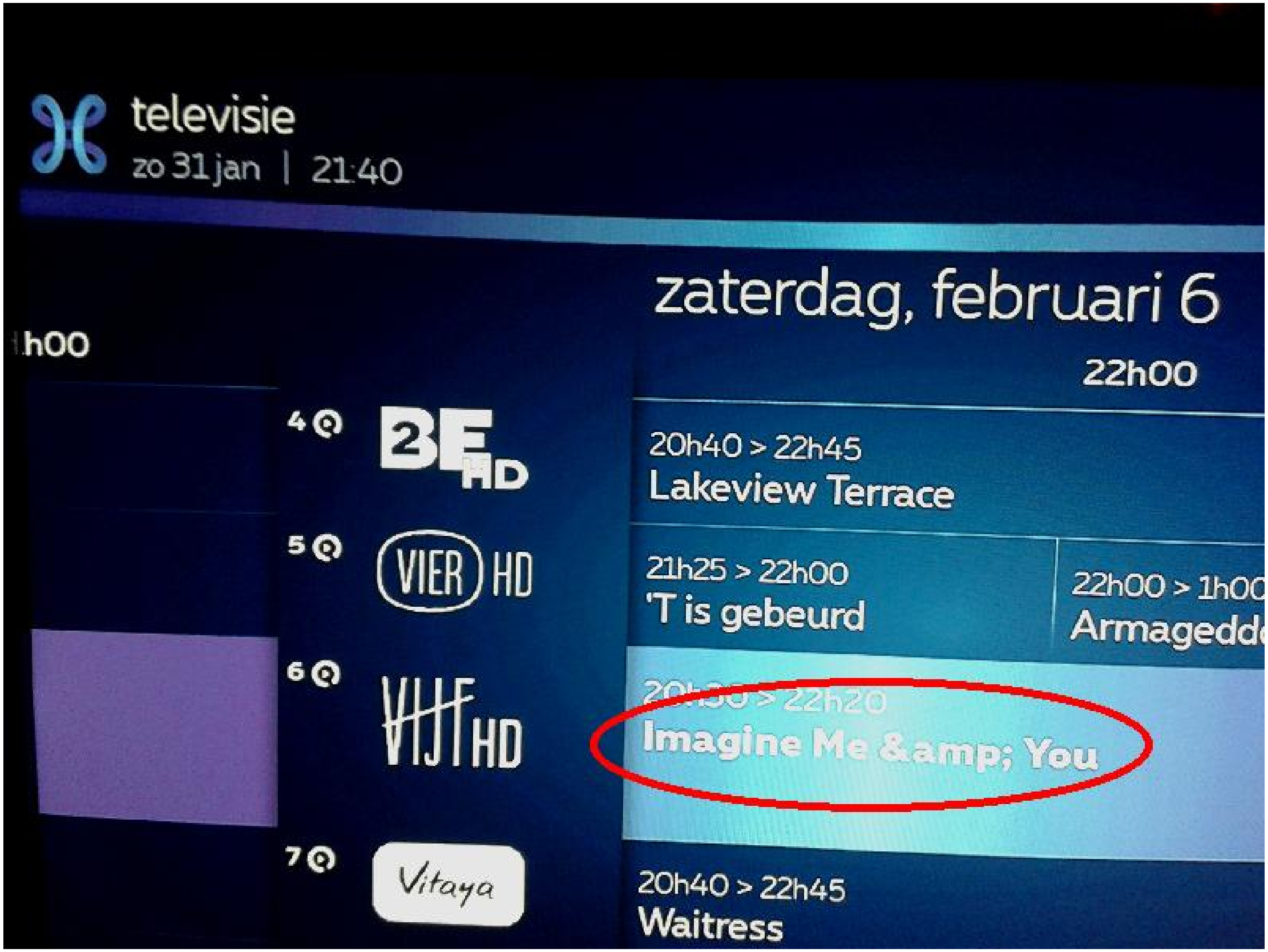
Escape characters

*An **escape character** is a character
which invokes an alternative interpretation
on subsequent characters in a character sequence.*

- Java string literal: \ (backslash)
- XML: & (ampersand)

Failure to handle escape characters correctly
often causes security issues (SQL inject, XSS, ...)

Digital TV



televisie
zo 31 jan | 21:40

zaterdag, februari 6

h00

22h00

4 2BE HD
20h40 > 22h45
Lakeview Terrace

5 VIER HD
21h25 > 22h00
'T is gebeurd
22h00 > 1h00
Armagedd

6 VIJF HD
20h30 > 22h20
Imagine Me &amp; You

7 Vitaya
20h40 > 22h45
Waitress

Imagine Me &amp; You

Cheat sheet text

String	**Why**
Allô (French telephone hello)	ISO 8859-1
€ (euro)	Since 1996, not in 8859-1
Hallå (Swedish hello)	Mostly ASCII
Здравствуйте (Russian hello)	Looks a bit like ASCII
こんにちは (Japanese hello)	No ASCII whatsoever
≠ (different) ⇔ (iff) ∑ (sum)	Math symbols
\ (backslash) " (double) ' (single)	Java/SQL/... special chars
& (ampersand) < (lower than)	XML special chars
` (slant) # (number sign) $ (dollar)	Shell special chars

Dates and time

Days between 2 dates

```
private static final long MILLISECONDS_IN_DAY = 24L * 60L * 60L * 1000L;

public long daysBetween(Date a, Date b) {
    return (b.getTime() - a.getTime()) / MILLISECONDS_IN_DAY;
}
```

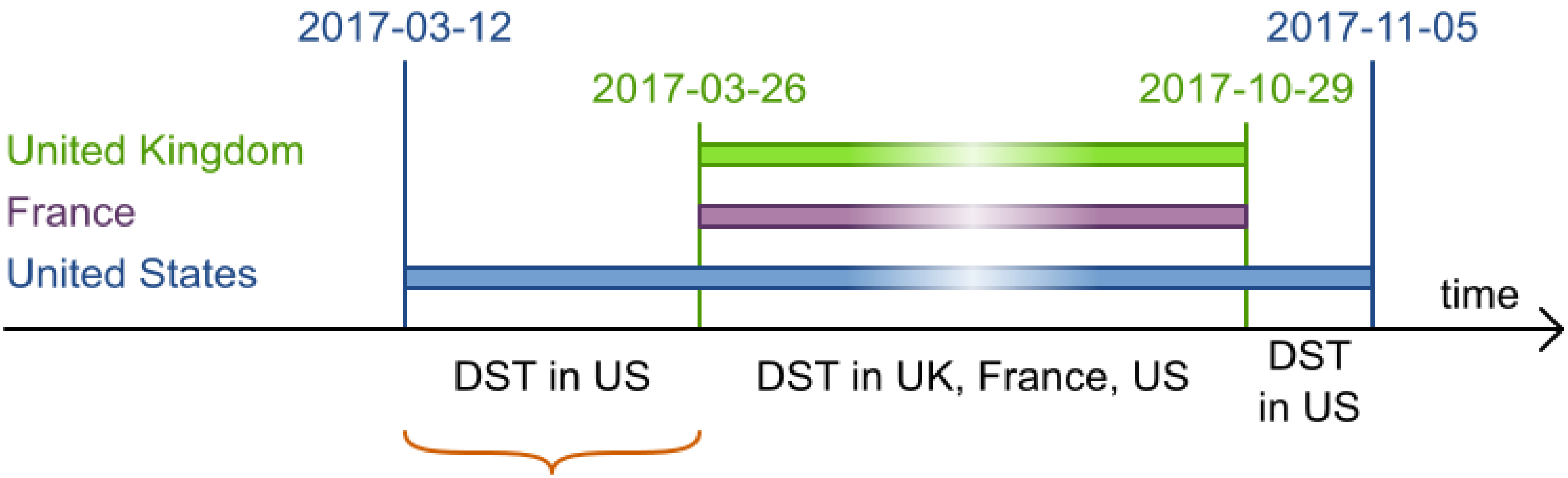
Input

```
daysBetween(parse("2017-02-01"), parse("2017-02-02"))
1

daysBetween(parse("2017-03-12"), parse("2017-03-13"))
1 // In UK and France
0 // In US, because of Daylight Saving Time

daysBetween(parse("2017-03-26"), parse("2017-03-27"))
0 // In UK and France because of Daylight Saving Time
1 // In US
```

One day is *usually* 24 hours.

Daylight Saving Time changes

The special season for software bugs.



That time of the year
when europeans are always
late for american conf calls

New York: 23 hours between 2017-03-12 00:00 and 2017-03-13 00:00.

New York: 24 hours between 2017-03-26 00:00 and 2017-03-27 00:00.

London: 24 hours between 2017-03-12 00:00 and 2017-03-13 00:00.

London: 23 hours between 2017-03-26 00:00 and 2017-03-27 00:00.

Days are not a multiple of hours

Problem

```
private static final long MILLISECONDS_IN_DAY = 24L * 60L * 60L * 1000L;

public long daysBetween(Date a, Date b) {
    return (b.getTime() - a.getTime()) / MILLISECONDS_IN_DAY;
}
```

Solution: Never use java.util.Date!

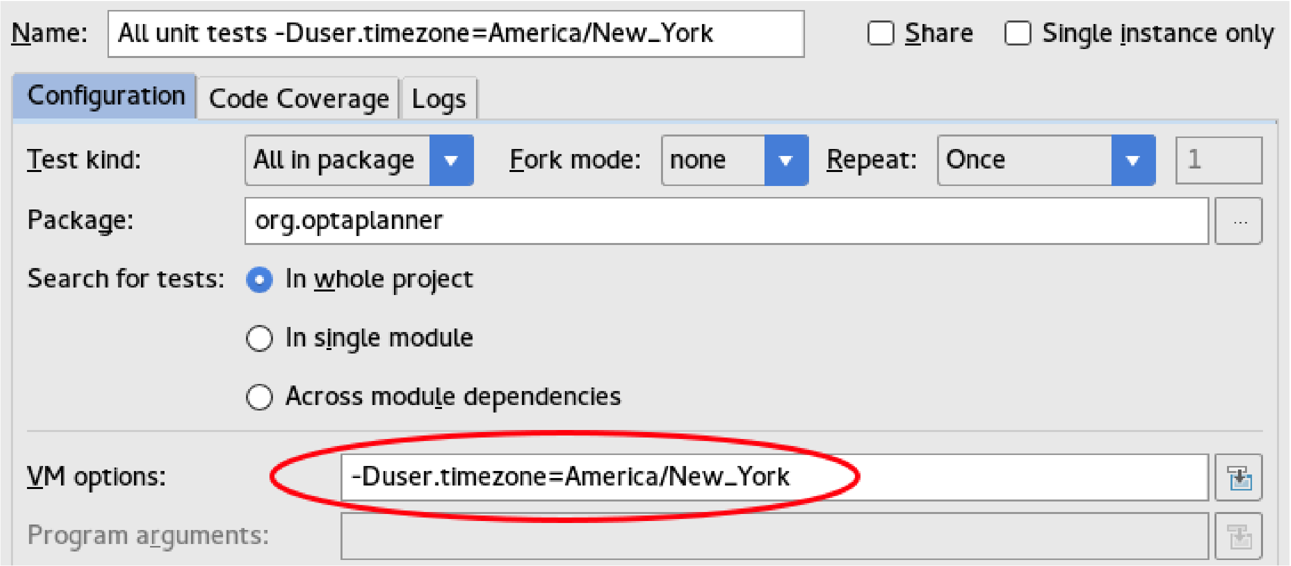
```
public long daysBetween(LocalDate a, LocalDate b) {
    return ChronoUnit.DAYS.between(a, b);
}

TimeZone.setDefault(TimeZone.getTimeZone("America/New_York"));
daysBetween(LocalDate.of(2017, 3, 12), LocalDate.of(2017, 3, 13))
1
daysBetween(LocalDate.of(2017, 3, 26), LocalDate.of(2017, 3, 27))
1
```

Always use java.time classes.

Tip

Run all your tests in a different timezone

```
java -Duser.timezone=America/New_York ...

java -Duser.timezone=Europe/Paris ...
```



Name: All unit tests -Duser.timezone=America/New_York

□ Share □ Single instance only

Test kind: All in package

Fork mode: none

Repeat: Once 1

Package: org.optaplanner

Search for tests:
● In whole project
○ In single module
○ Across module dependencies

VM options: -Duser.timezone=America/New_York

Program arguments:

Cheat sheet dates and time

Expression	**Actual result**
From 2017-03-12 00:00 to 2017-03-13 00:00	23 hours in America/New_York
From 2017-03-26 00:00 to 2017-03-27 00:00	23 hours in Europe/Paris
From 2017-10-29 00:00 to 2017-10-30 00:00	25 hours in Europe/Paris
From 2017-11-05 00:00 to 2017-11-06 00:00	25 hours in America/New_York

Q & A

Slides	ge0ffrey.github.io/ge0ffrey-presentations/ (https://ge0ffrey.github.io/ge0ffrey-presentations/)
Cheat Sheet	.../cornerCaseCheatSheet/cheatSheetJava.html (https://ge0ffrey.github.io/ge0ffrey-presentations/cornerCaseCheatSheet/cheatSheetJava.htm
Feedback	@GeoffreyDeSmet (https://twitter.com/GeoffreyDeSmet)