Eclipse UOMo

Click to edit Master subtitle style

Embedded Measurement

emergn

Success is a planned event

Our Goal

emergn

Avoiding Interface

and

Arithmetic Errors

Emphasis

Most of today's technologies including the current Java Language Releases lack support for common non-trivial Arithmetic problems like Unit Conversions.

Summary

- **Present Situation**
  - Historic IT Errors and Bugs
  - Cause of Conversion Errors
- **Proposed Changes**
  - Unit and Measure Support
  - Type Safety
- **Case Studies**
- **Demo**
- **Q&A**

What do these disasters have in common?

- **Patriot Missile**
  The cause was an inaccurate calculation of the time since boot due to a computer arithmetic error.
- **Ariane 5 Explosion**
  The floating point number which a value was converted from had a value greater than what would be represented by a 16 bit signed integer.

What do these disasters have in common?

- **Mars Orbiter**

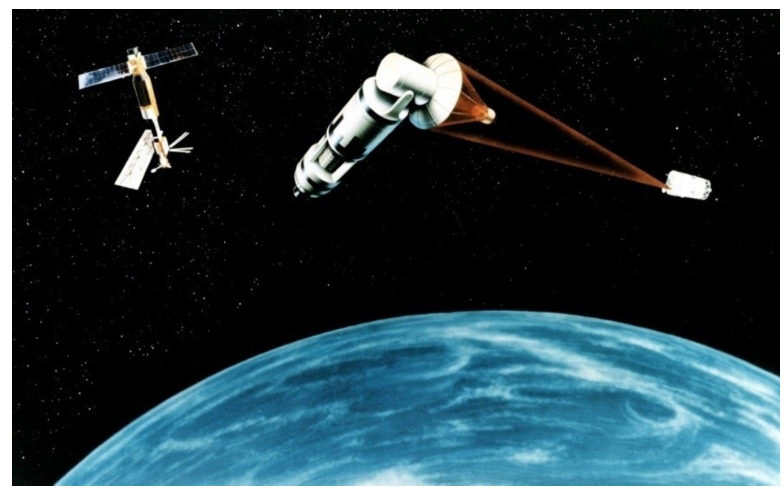
  Preliminary findings indicate that one team used English units (e.g. inches, feet and pounds) while the other used metric units for a key spacecraft operation.

  - NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation
  - This also underlines the added risk when 3rd party contractors are involved or projects are developed Offshore

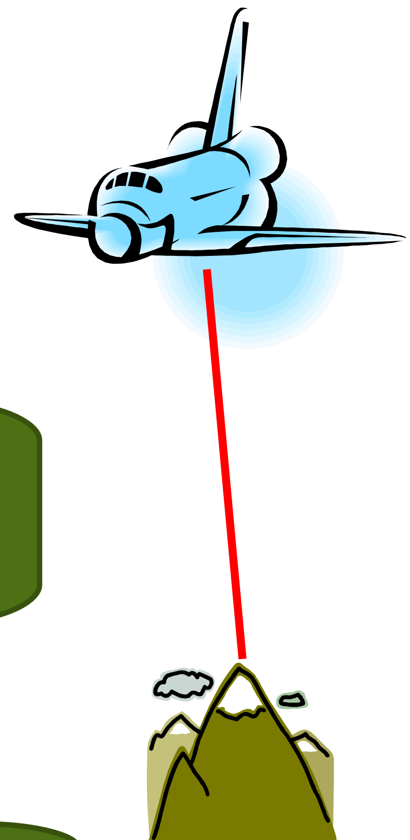
NASA "Star Wars" Experiment, 1983

23rd March 1983. Ronald Reagan announces SDI (or "Star Wars"): ground-based and space-based systems to protect the US from attack by strategic nuclear ballistic missiles.

STRATEGIC DEFENSE INITIATIVE
DEPARTMENT OF DEFENSE



1985

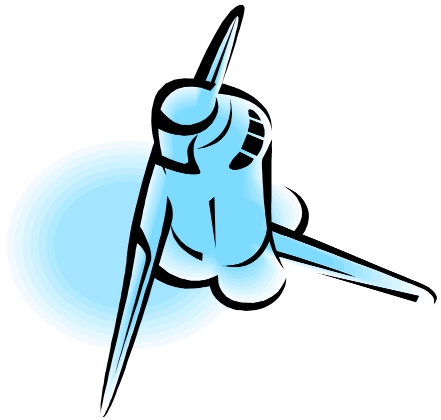


Mirror on underside of shuttle

Big mountain in Hawaii

SDI Experiment:
The Plan

1985



SDI Experiment:
What really
happened

1985: What happened?

ACM SIGSOFT SOFTWARE ENGINEERING NOTES vol 10 no 3 Jul 1985 page 10

Attention All Units, Especially Miles and Feet!

Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. The astronauts reported seeing the bright-blue low-power laser beam emanating from the top of Mona Kea, but the experiment failed because the shuttle's reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot *10,023 feet* above sea level on Mona Kea; that number was supplied to the crew in units of feet, and was correctly fed into the onboard guidance system -- which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot *10,023 nautical miles* above sea level. The San Francisco Chronicle article noted that "the laser experiment was designed to see if a low-energy laser could be used to track a high-speed target about 200 miles above the earth. By its failure yesterday, NASA unwittingly proved what the Air Force already knew -- that the laser would work only on a 'cooperative target' -- and is not likely to be useful as a tracking device for enemy missiles." [This statement appeared in the S.F. Chronicle on 20 June, excerpted from the L.A. Times; the NY Times article on that date provided some controversy on the interpretation of the significance of the problem.] The experiment was then repeated successfully on 21 June (using nautical miles). The important point is not whether this experiment proves or disproves the viability of Star Wars, but rather that here is just one more example of an unanticipated problem in a human-computer interface that had not been detected prior to its first attempted actual use.

emergn

NASA Mars Climate Orbiter, 1999

CNN - Metric mishap caused loss of NASA orbiter - September 30, 1999 - Windows Internet Explorer

http://www.cnn.com/TECH/space/9909/30/r

sci-tech> space> story page

exploringmars

Metric mishap caused loss of NASA orbiter

September 30, 1999
Web posted at: 4:21 p.m. EDT (2021 GMT)

In this story:

Metric system used by NASA for many years

Error points to nation's conversion lag

RELATED STORIES, SITES

NASA's Climate Orbiter was lost September 23, 1999

By Robin Lloyd
CNN Interactive Senior Writer

(CNN) -- NASA lost a $125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.

Lockheed Martin helped build, develop and operate the spacecraft for NASA. Its engineers provided navigation commands for Climate Orbiter's thrusters in English units although NASA has been using the metric system predominantly since at least 1990.

No one is pointing fingers at Lockheed Martin, said Tom Gavin, the JPL administrator to whom all project managers report.

Unit Tests did not find these Errors

- **All previous example illustrate three categories of errors difficult to find through Unit Testing:**
  - Interface Errors (e.g. millisecond/second, radian/degree, meters/feet).
  - Arithmetic Errors (e.g. overflow).
  - Conversion Errors.

Causes of Conversion Errors

- **Ambiguity on the unit**
  - Gallon Dry / Gallon Liquid
  - Gallon US / Gallon UK
  - Day Sidereal / Day Calendar
  - ...

- **Wrong conversion factors:**

```
static final double PIXEL_TO_INCH = 1 / 72;
double pixels = inches * PIXEL_TO_INCH
```

emergn

Present Situation

- Java does not have strongly typed primitive types
  (like e.g. Ada language).
- For performance reasons most developer prefer primitive types over objects in their interface.
- Primitives type arguments often lead to name clashes (methods with the same signature)

UCUM

Unified Code for Units of Measure

- The Unified Code for Units of Measure is a code system intended to include all units of measures being contemporarily used in international science, engineering, and business. The purpose is to facilitate unambiguous electronic communication of quantities together with their units. The focus is on electronic communication, as opposed to communication between humans. A typical application of The Unified Code for Units of Measure are electronic data interchange (EDI) protocols, but there is nothing that prevents it from

UCUM

Unified Code for Units of Measure

The Unified Code for Units of Measure is inspired by and heavily based on

- **ISO 2955-1983**
- **ANSI X3.50-1986**
- **HL7's extensions called ISO+**

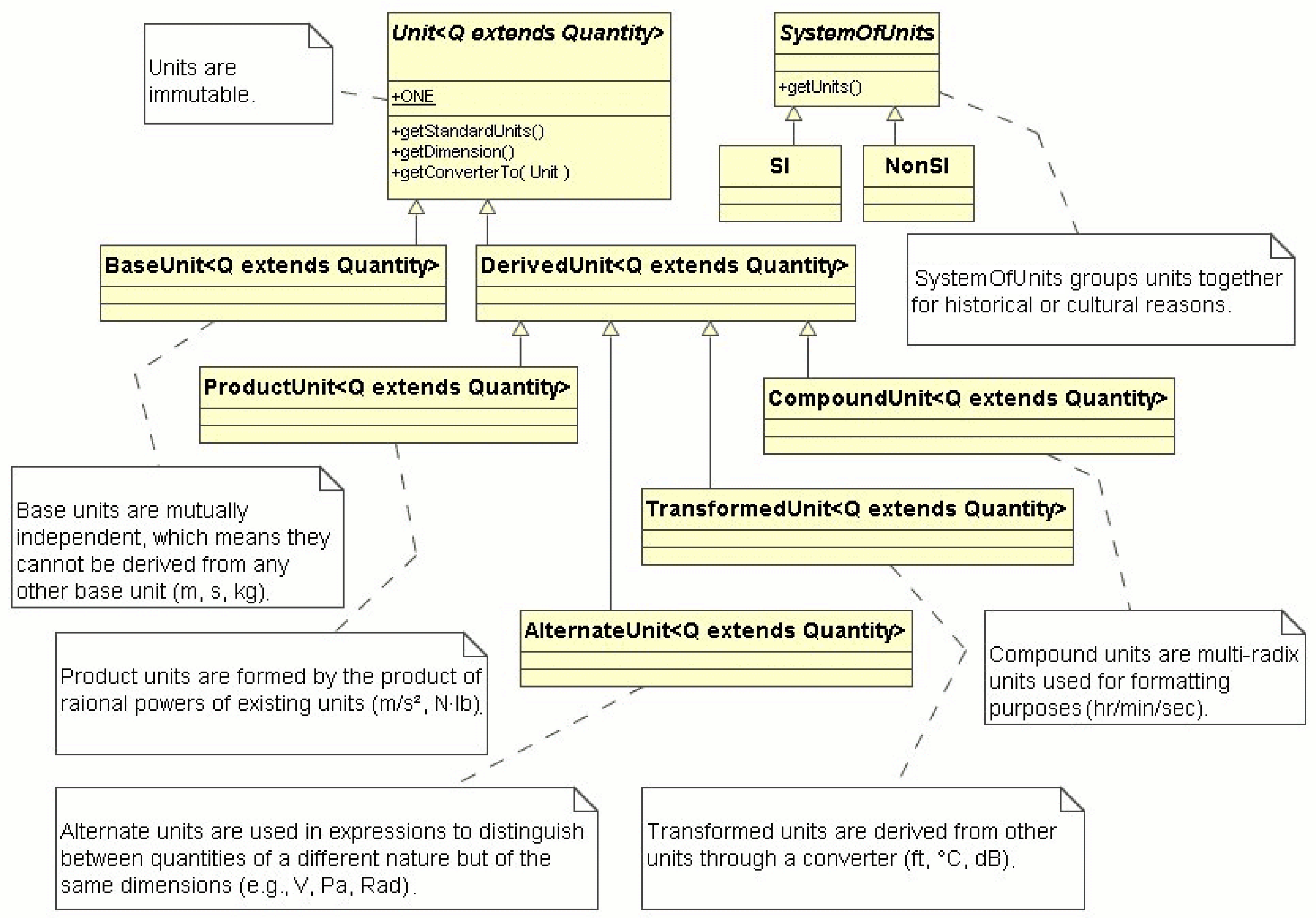
emergn

Base Classes and Packages

- **Namespace: javax.measure.***
- **Only one interface and one abstract class**
  - Measurable<Q extends Quantity> (interface)
  - Measure<V, Q extends Quantity> (abstract class)
- **Three sub-packages**
  - Unit (holds the SI and NonSI units)
  - Quantity (holds dimensions mass, length)
  - Converter (holds unit converters)

emergn

Units and System of Units



Unit Operations

Results with

Same Dimension	Different Dimension
Binary Operations	Binary Operations
plus (double) or (long)	root(int)
times(double) or (long)	power(int)
divide(double) or (long)	times(Unit)
compound(Unit)	divide(Unit)

Unary Operations

inverse()

The King is Dead…

Units of Measure API

- **Namespace: org.unitsofmeasure.***
- **Only interfaces and one abstract class**
  - public interface Quantity<Q extends Quantity<Q>>
  - public interface Unit<Q extends Quantity<Q>>
- **Two sub-packages**
  - quantity (holds dimensions mass, length)
  - util (misc items and helpers, **optional**)

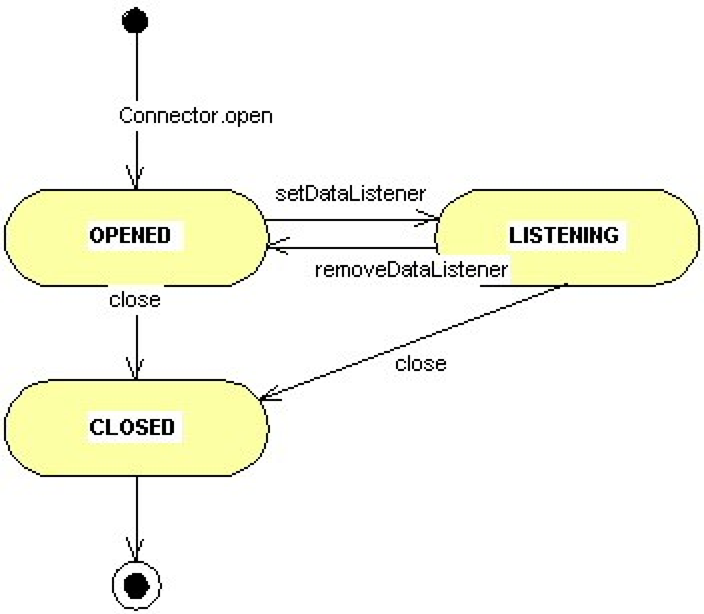
emergn

Mobile Sensor API

- **Namespace: javax.microediton.sensor***
- **Focusing on Sensors, but it got a minimalistic Unit API "in the closet"**
  - Unit
    Essentially an SI singleton holding relevant unit constants, too.
  - ChannelInfo
    Holding name, accuracy, data type,measurement ranges, scale and unit
  - MeasurementRange
    Range of possible values from minimum to maximum

JSR-256

Sensor States



JSR-256

Sensor Groups

- **Context types categorize sensors into three groups:**
  1. ambient, sensors measuring some ambient property of the environment
  2. device, sensors measuring properties related to the device
  3. user, sensors measuring some function of the user

Quantity

- **The quantity provides a more precise qualifier. The unit and the quantity has a close relation. Some quantities are listed in tables of Unit class. When the quantity and context type is known, it is often easy to guess the full purpose of the sensor. Some examples are given here:**

```
Quantity: electric_current + context type:
ambient = sensor measuring electric current,
amperemeter
```

```
Quantity: catalytic_activity + contex type:
ambient = sensor measuring catalytic
activity
```

Measurement Package

- **Namespace: org.osgi.util.measurement**
- **SI only Unit API "in the closet"**
  - Unit
    Essentially an SI singleton holding relevant unit constants, too.
  - Measurement
    Represents a value with an error, a unit and a time-stamp.
  - State
    Groups a state name, value and timestamp.

Eclipse UOMo

One Small Step…

Eclipse UOMo

One Unit Framework to Measure them All

- **Namespace: org.eclipse.uomo.***
- **Two main areas**
  - Static Type Safe Units of Measure Support
    - Based on Units of Measure API
    - On top of ICU4J, the Globalization standard at Eclipse and others (Android, GWT, Google Financial, etc.)
  - UCUM Reference Implementation
    - Successor to Eclipse OHF UCUM Bundle

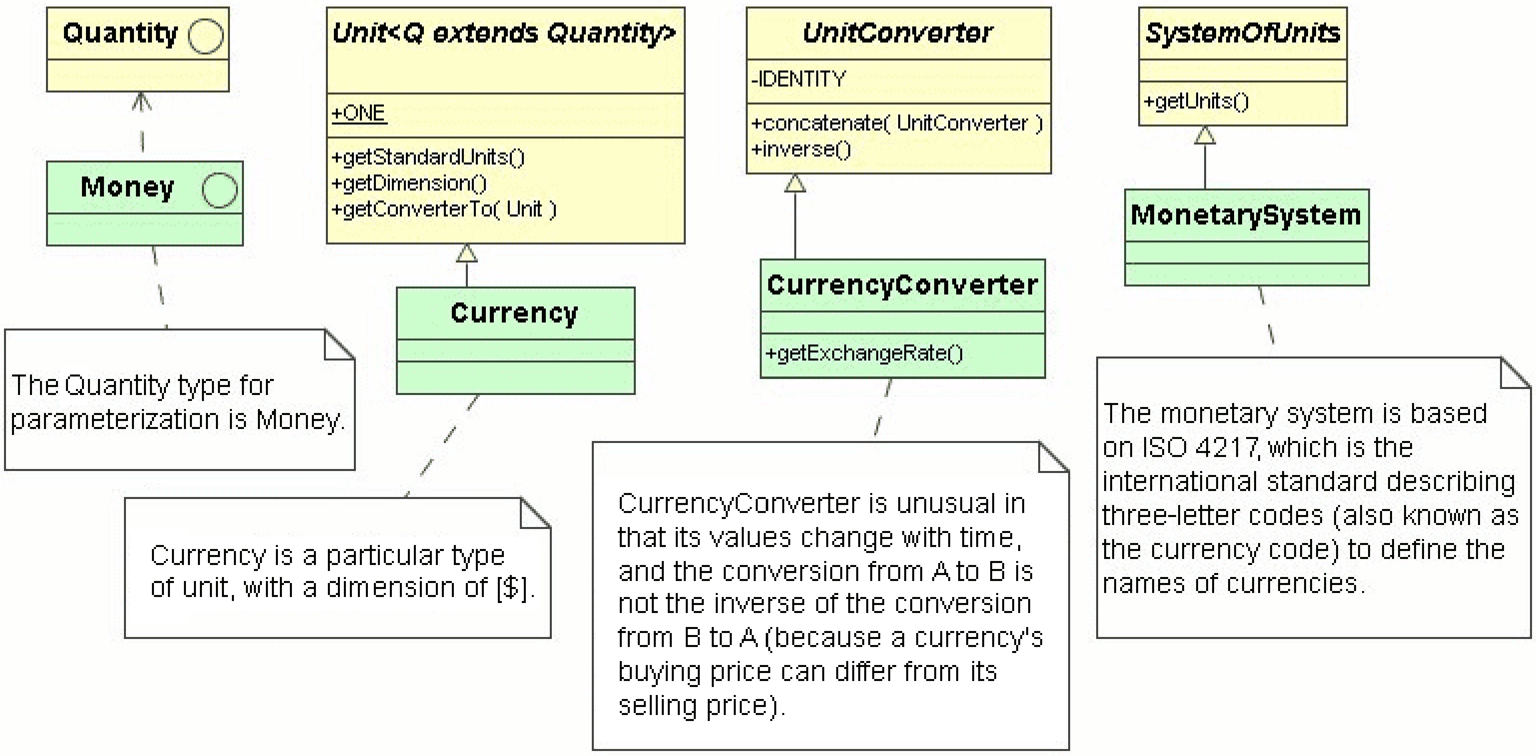
Case Study: Monetary System

Monetary systems are not currently in scope for UOMo, but this illustrates, how easily the framework can be extended to non physical or scientific quantities.

Such extension can be valuable by leveraging the framework's capabilities (formatting, conversion,…)

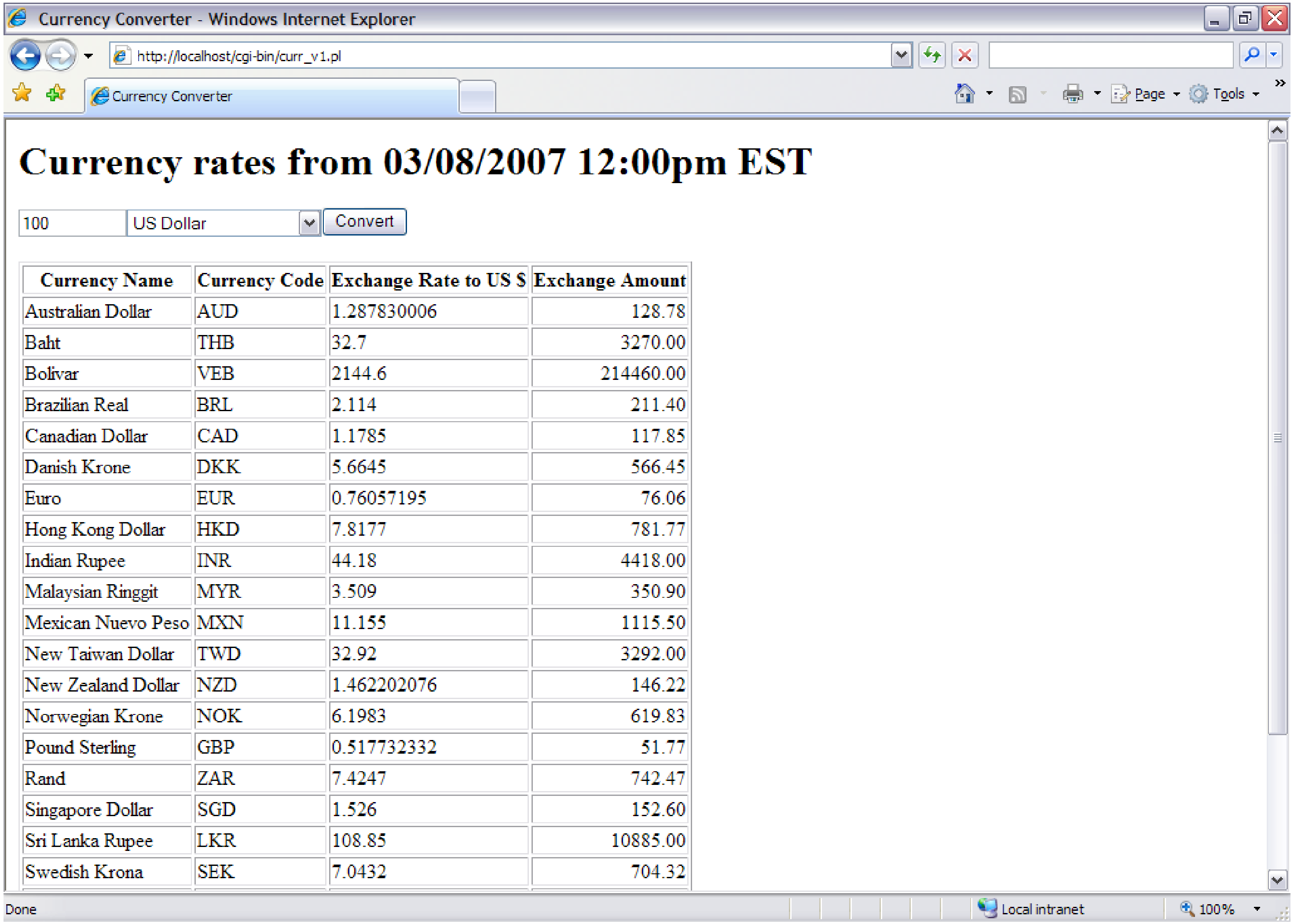
and applying its usefulness beyond what e.g. java.util.Currency now has to offer.

Monetary Types



emergn

Currency Conversion



Currency rates from 03/08/2007 12:00pm EST

100 | US Dollar | Convert

Currency Name	Currency Code	Exchange Rate to US $	Exchange Amount
Australian Dollar	AUD	1.287830006	128.78
Baht	THB	32.7	3270.00
Bolivar	VEB	2144.6	214460.00
Brazilian Real	BRL	2.114	211.40
Canadian Dollar	CAD	1.1785	117.85
Danish Krone	DKK	5.6645	566.45
Euro	EUR	0.76057195	76.06
Hong Kong Dollar	HKD	7.8177	781.77
Indian Rupee	INR	44.18	4418.00
Malaysian Ringgit	MYR	3.509	350.90
Mexican Nuevo Peso	MXN	11.155	1115.50
New Taiwan Dollar	TWD	32.92	3292.00
New Zealand Dollar	NZD	1.462202076	146.22
Norwegian Krone	NOK	6.1983	619.83
Pound Sterling	GBP	0.517732332	51.77
Rand	ZAR	7.4247	742.47
Singapore Dollar	SGD	1.526	152.60
Sri Lanka Rupee	LKR	108.85	10885.00
Swedish Krona	SEK	7.0432	704.32

Trading Example

What happens, if we use built in java.util.Currency and Standard JSP formats

Portfolio

Cash: 64102.56 €   Market: FRA

Symbol	Company	Price	Change	% Change	Shares	Open	Volume	Current Value *	Gain/Loss
IBM	"IBM"	¤115.43	¤0.37	-32%	50	¤115.80	2,655,471	3699.68 €	-¤15.98
JAVA	"JAVA"	¤16.56	¤0.44	273%	200	¤16.12	5,750,460	2123.08 €	¤545.90
DELL	"DELL"	¤19.52	¤0.08	41%	200	¤19.44	14,293,015	2502.56 €	¤82.30
GOOG	"GOOG"	¤426.88	¤1.62	38%	100	¤425.26	5,523,309	27363.97 €	¤38.05
MSFT	"MSFT"	¤28.58	¤0.20	71%	100	¤28.38	47,317,464	1832.15 €	¤71.00

* in local Currency

Make a trade
Log out

Money Demo (1)

We'll extend MoneyDemo to show fuel costs in Indian Rupees.

First by adding a new currency to MonetarySystem.

```
// Use currency not defined as constant (Indian Rupee).
public static final DerivedUnit<Money> INR = monetary(
        new Currency(„INR„));
```

Then add this line to MoneyDemo.
(also change static import to `MonetarySystem.*;`)

```
UnitFormat.getInstance().label(INR, „Rp");
```

Money Demo (2)

Next set the Exchange Rate for Rp.

```
((Currency) INR).setExchangeRate(0.022);   // 1.0Rp = ~0.022 $
```

Note, the explicit cast is required here, because getUnits() in SystemOfUnits currently requires a neutral <?> generic collection type.

Money Demo (3)

Then we add the following line to the "Display cost." section of MoneyDemo

```
System.out.println("Trip cost = " + tripCost + " (" +
tripCost.to(INR) + ")");
```

Resulting in the additional output:

```
Trip cost = 87.50 $ (3977.26 Rp)
```

Links

Eclipse – Project UOMo
http://www.eclipse.org/proposals/uomo/

UCUM
http://www.unitsofmeasure.org

Units of Measure API
http://www.javaforge.com/project/uom

Questions

werner@emergn.com

or

info@catmedia.us

Twitter: @wernerkeil