Measure for Measure!

JSR-275 Public Review

Werner Keil

Santa Clara, California

January 12th, 2010

CATMedia

Our Goal

AVOIDING INTERFACE AND ARITHMETIC ERRORS

Emphasis

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

Overview

- 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** (SI) system for a key spacecraft operation
  - This also underlines the added risk when 3rd party contractors are involved or projects are developed Offshore

Unit Tests wouldn‘t find these…

- 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
```

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)

JSR-275

Base Classes

Namespace: javax.measure.*

Core parts or the API are one interface and an abstract base class

- Measurable<Q extends Quantity> (interface)
- Measure<V, Q extends Quantity> (abstract class)

JSR-275

Packages

- Unit

  holds the SI, NonSI and UCUM units

- Quantity

  holds dimensions like mass or length)

- Converter

  holds Unit Converters

- Format

  holds common formatters including UCUM)

JSR-275

Measurable (1)

Let's take the following example

```
abstract class Person
{
    void setWeight(double weight);
}
```

Should the weight be in Pound, Stone, Kilogram, or what else ???

JSR-275

Measurable (2)

Using Measurable there is no room for error

```
abstract class Person
{
    void setWeight(Measurable<Mass> weight);
}
```

Not only the interface is cleaner (the weight must a be generic mass type), but also there is no confusion on the measurement unit

JSR-275

Measurable (3)

So while either of these calls are legitimate:

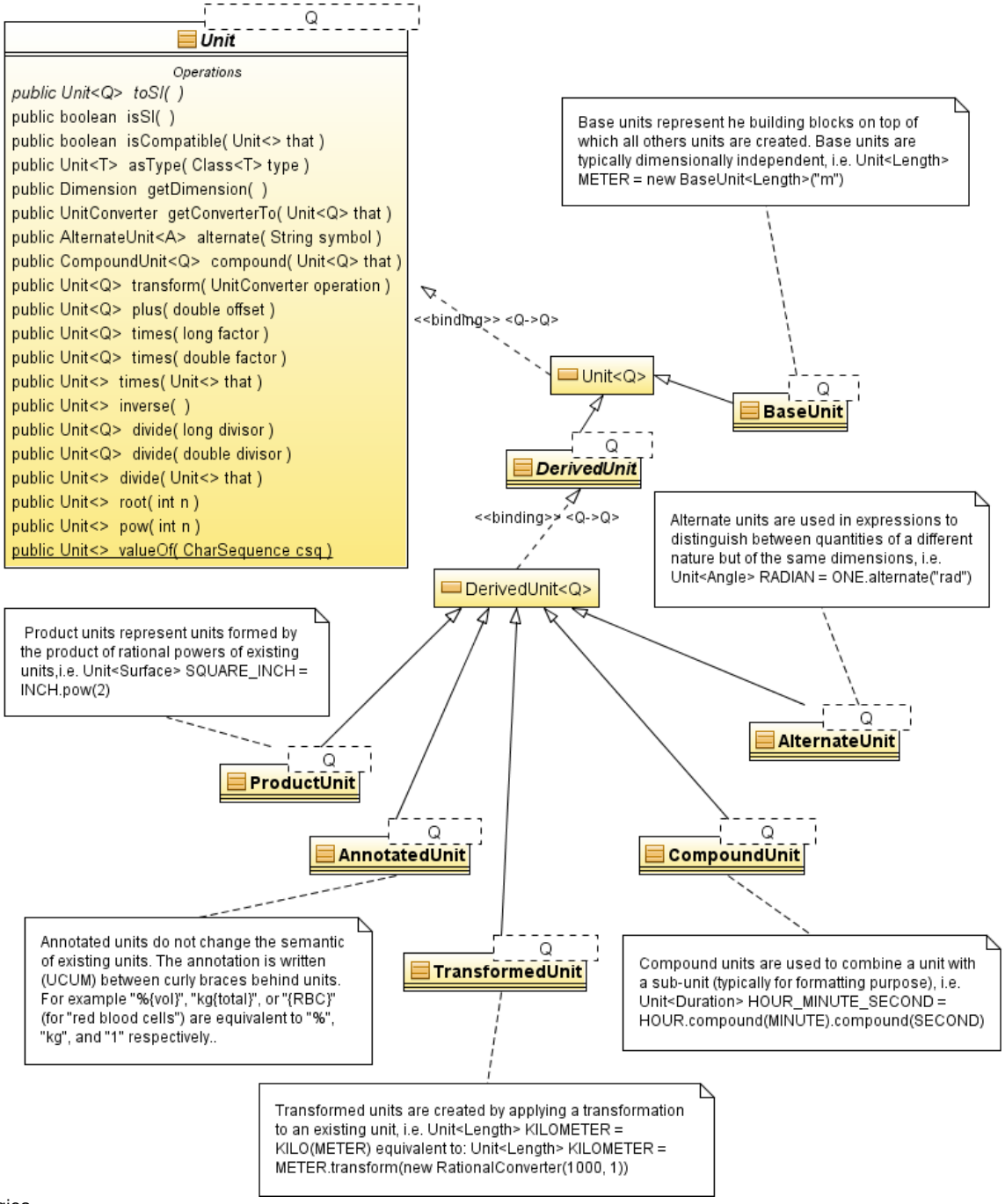
```
double weightInKg = weight.doubleValue(KILOGRAM);
double weightInLb = weight.doubleValue(POUND);
```

This one isn't:

```
double weightInLiter = weight.doubleValue(LITER);
// Compile Error
```

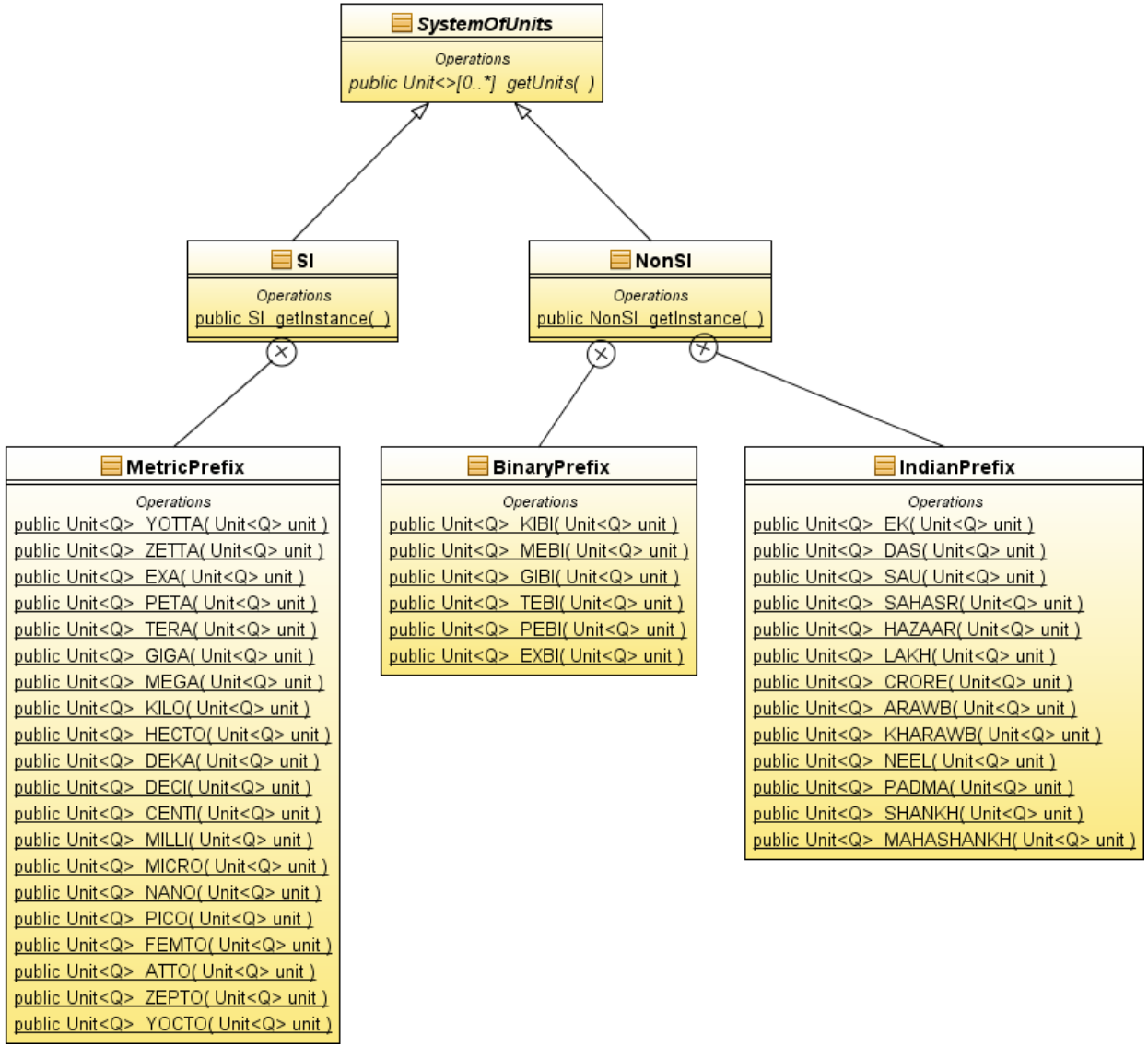
JSR-275

Units



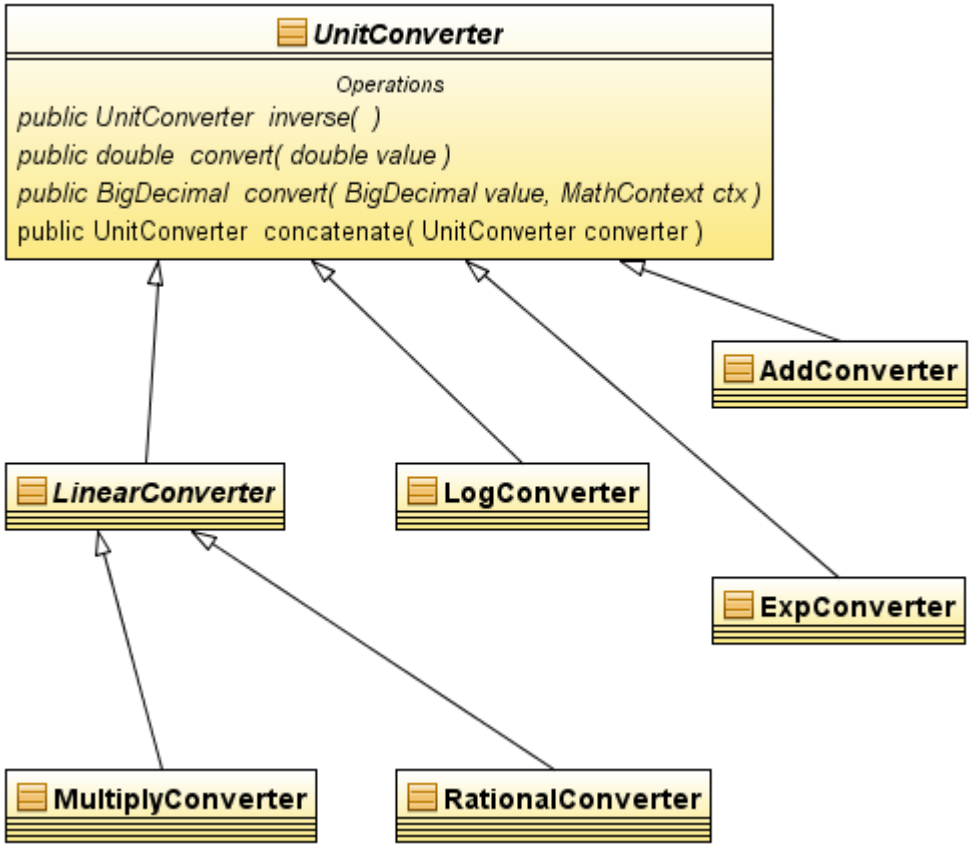
JSR-275

Systems of Units



JSR-275

Unit Converters



JSR-275

Unit Operations

Result with same dimension	*Result with different dimension*
Binary operations: •**plus** (double) *or* (long) •**times** (double) *or* (long) •**divide** (double) *or* (long) •**compound** (Unit)	*Binary operations:* •**root** (int) •**power** (int) •**times** (Unit) •**divide** (Unit) *Unary operations:* •**inverse**()

JSR-275

Measure or Measurable

Answer:

Measurable is an interface allowing all kinds of implementations.

It is matching equivalent to e.g. java.lang.Number and provides similarly named methods for conversion to primitive types such as intValue(Unit) or doubleValue(Unit).

Measure is the combination of a numeric value and a unit. Measurable is more flexible, but if you need to retrieve the original numeric value stated in its original unit and precision (no conversion), then Measure or subclasses are your choice.

JSR-275

Kenai.com

As the first official JSR our EG decided to migrate to **Kenai.com**, Sun‘s Developer Cloud for Java, JavaFX, MySQL, Glassfish and other Open Source Activities

Project Kenai

http://www.kenai.com

Search for JSR-275

JSR-275

References

- GeoAPI
- Thales Group
- Orbitz/Ebookers.com
- IEM (Emergency Management)
- UCUM
- OpenEHR
- Project Noodles

JSR-275

Languages and Platforms

- Java
- Groovy/Grails
- Scala
- Android
- plus any other JVM-based language

JSR-275

Search Results

- Google: 270.000
  - Once you enter at least “JSR-2”
  - That is 3rd largest for any single JSR (only 168 and 256 have more)
- Bing: 694.000
- Yahoo: 412.000

Let’s have a look at some…

DEMOS

JSR-275

Case Study: Monetary System

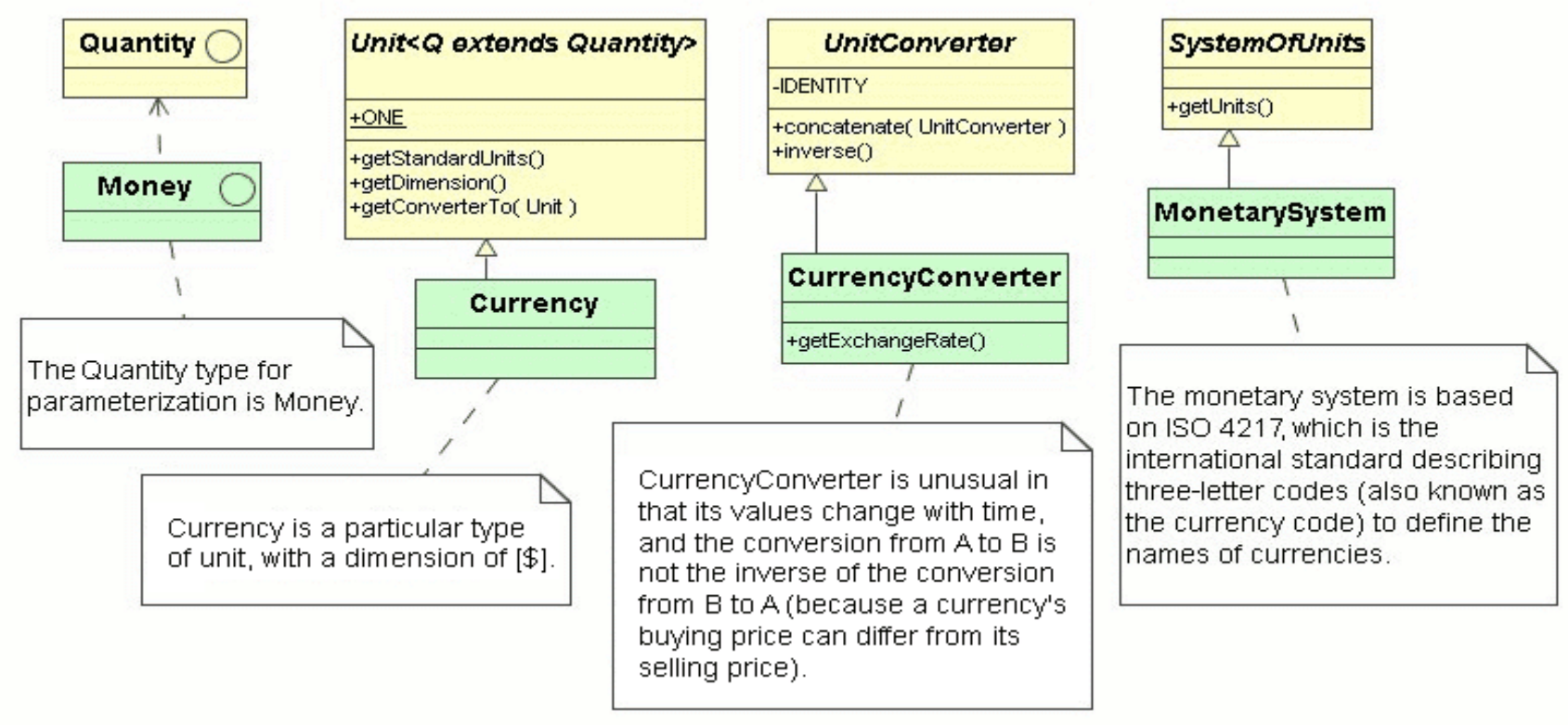
Monetary systems are not subject to JSR-275, but this illustrates, how easily the framework can be extended to non physical or scientific quantities.

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

and applying its usefulness beyond what java.util.Currency now provides

JSR-275

Currency Conversion Classes



Let’s have a look at some…

DEMOS

JSR-275

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");
```

JSR-275

Money Demo(2)

Next set the Exchange Rate for Rupees

```
((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.

JSR-275

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 this additional output:

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

Let‘s talk

Q & A