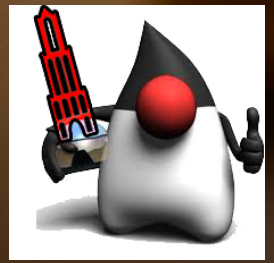
Introducing JSR 354
The Money & Currency API

Jeroen Burggraaf & Wim van Haaren

27-10-2017



blue4IT

About us

Wim van Haaren
Freelance Software Engineer
co-founder of Tritales

wim@tritales.nl

LeasePlan EQUENS DE PERSGROEP NEDERLAND BMW POLITIE
ING Sanoma Angarde amc

About us

Jeroen Burggraaf

Freelance Software Engineer

co-founder of Tritales

jeroen@tritales.nl

Rabobank

EQUENS

Belastingdienst

euronext

ProRail

Agenda

- Motivation
- Core API
- Advanced topics:
  - Conversion
  - Formatting
  - Operators & Queries
  - Streams

Motivation

Money & Java before JSR 354 came along

Why the need for a Money Library?

Martin Fowler:

"A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming language. The lack of a type causes problems, the most obvious surrounding currencies. …"

http://martinfowler.com/eaaCatalog/money.html

www.tritales.nl

Eric Evans:

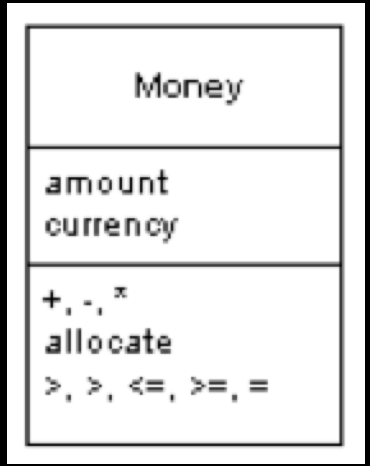
“On project after project, software developers have to reinvent the wheel, creating objects for simple recurring concepts such as "money" and "currency”…”

http://timeandmoney.sourceforge.net/

www.tritales.nl

The concept of Money

- Numerical value
- Currency
- operators



Primitive floating types

```
@Data
public class Product {
    private String name;
    private double amount;
}
```

www.tritales.nl

```
@Test
 public void subtract() throws Exception {
    Product a = new Product(1.03);
    Product b = new Product(.42);

    double result = a.getAmount() - b.getAmount();
    assertSame(0.61, result);
 }
```

www.tritales.nl

```
java.lang.AssertionError:
Expected :0.61
Actual   :0.6100000000000001
```

IEEE-754

Why computers calculate wrong:
https://youtube.com/UPWydP2rqJY

www.tritales.nl

Primitive Integer Types

```
@Data
public class Product {
    private String name;
    private long money;
}
```

Convert to cents

www.tritales.nl

```
Product a = new Product("A", 5_00);
Product b = new Product("B", 4_00);

System.out.println(
    "€" + a.getAmount() + b.getAmount();
);
> €900
```

900 euros?

www.tritales.nl

BigDecimal

CURRENCY

```
    b1.getMoney().add(b2.getMoney())
);
> 9.0
```

Euros? Dollars? Yen?

www.tritales.nl

Currency - String

```
public class Product {
    private String name;
    private String currency;
    private BigDecimal money;
}
```

Type-safety
Validation

www.tritales.nl

Currency - Enum

```
public class Product {
    private String name;
    private Currency currency;
    private BigDecimal money;
}

enum Currency {
    EURO, POUND, DOLLAR;
}
```

i18n

ISO-4217

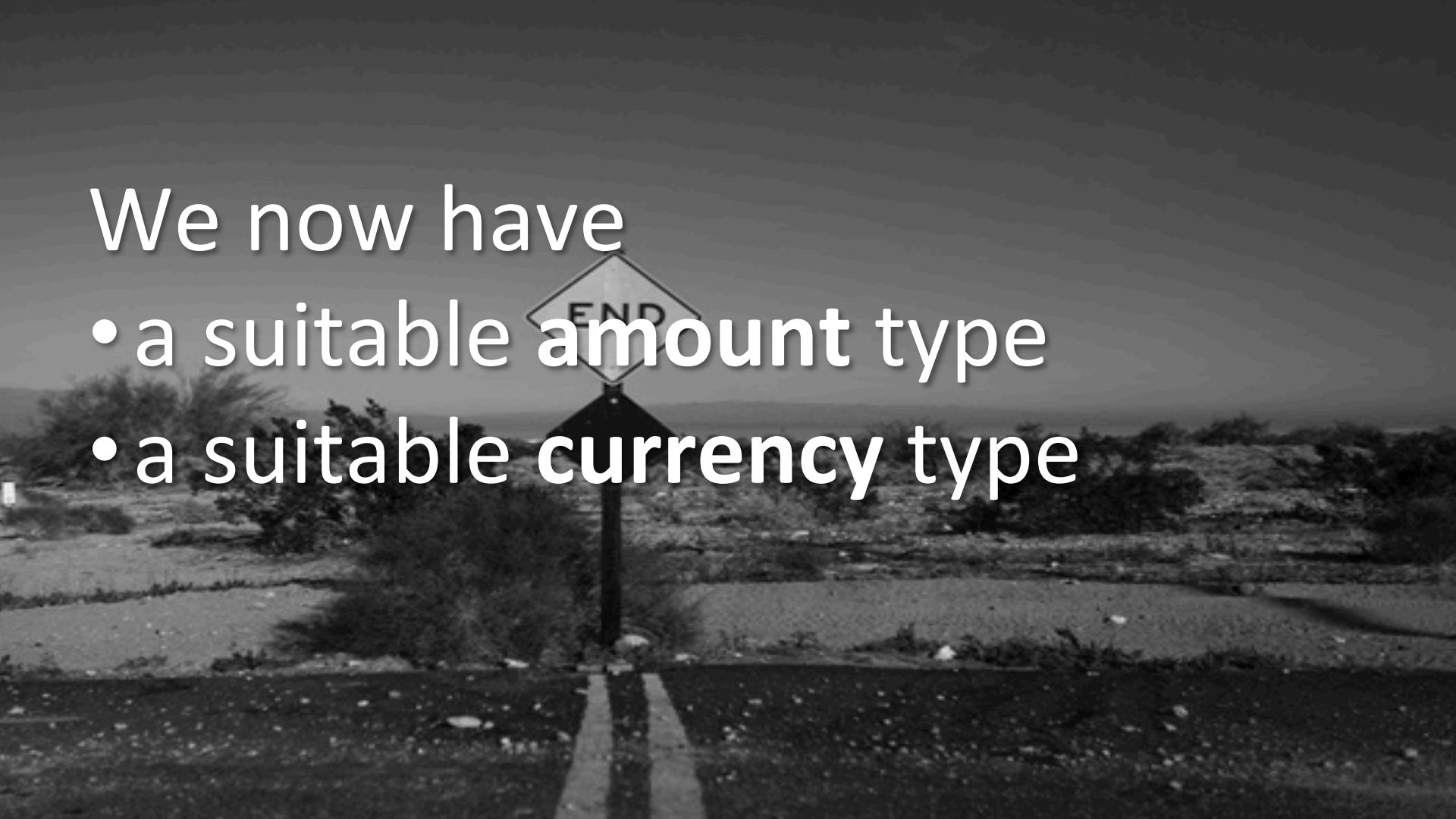
www.tritales.nl

java.util.Currency

- Simple
- Supports ISO-4217

```
public class Product {
    private String name;
    private Currency currency;
    private BigDecimal money;
}
```

www.tritales.nl



We now have

- a suitable **amount** type
- a suitable **currency** type

```
Product a = new Product("A",
                        Currency.getInstance("EUR"),
                        BigDecimal.valueOf(5));
Product b = new Product("B",
                        Currency.getInstance("USD"),
                        BigDecimal.valueOf(4));

if (a.getCurrency().equals(b.getCurrency())) {
    // ...
}
```

www.tritales.nl

Utility class

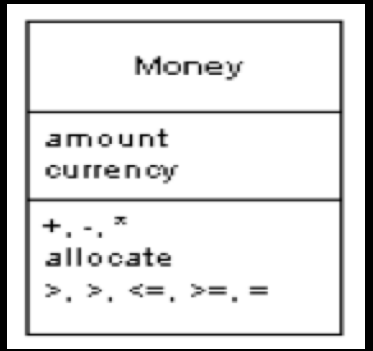
```
public static class ProductUtils {
  public static BigDecimal sum(Product a,Product b) {
    if (a.getCurrency().equals(b.getCurrency())) {
      return a.g
    }
    throw new
  }
}
```

A lot of utility classes
And you'd better not forget to use them

www.tritales.nl

Money

By now, an abstraction for representing money should seem obvious



Money

```
public class Money {
    private final BigDecimal value;
    private final Currency currency;

    // behavior goes here ...
}
```

www.tritales.nl

Motivation for Money

- Monetary values are a key feature to many applications
- No standard value type to represent a monetary amount
- ISO-4217 ambiguous/confusing
- Historical & virtual currencies missing
- Conversions & arithmetic missing
- Formatting missing

www.tritales.nl

JSR 354 – Money and Currency API

- Started 2012 – finalized 2015
- ~~Targeted for JDK 9~~
- Standalone API:
  - Available for JDK 8 (and a backport for 7)
  - Specification (http://javamoney.github.io/api.html)
  - RI “Moneta” (http://javamoney.github.io/ri.html)

www.tritales.nl


Core API

JSR 354

- API
  - http://javamoney.github.io/api.html
- Moneta - Reference Implemantation (RI)
  - http://javamoney.github.io/ri.html

```
<dependency>
    <groupId>org.javamoney</groupId>
    <artifactId>moneta</artifactId>
    <version>1.1</version>
</dependency>
```

www.tritales.nl

CurrencyUnit

```
CurrencyUnit currency =
    Monetary.getCurrency("EUR");

CurrencyUnit currency =
    Monetary.getCurrency(
        new Locale("nl", "NL"));
```

www.tritales.nl

Custom Currency

```
Monetary.isCurrencyAvailable("BTC"); // false

CurrencyUnit bitcoin = CurrencyUnitBuilder
    .of("BTC", "BtcCurrencyProvider")
    .setDefaultFractionDigits(2)
    .build(true);

Monetary.isCurrencyAvailable("BTC"); // true
```

www.tritales.nl

MonetaryAmount

```
boolean isGreaterThan(MonetaryAmount amt)
boolean isGreaterThanOrEqualTo(MonetaryAmount amt)
boolean isLessThan(MonetaryAmount amt)
boolean isLessThanOrEqualTo(MonetaryAmount amt)
boolean isEqualTo(MonetaryAmount amt)
boolean isNegative()
boolean isNegativeOrZero()
boolean isPositive()
boolean isPositiveOrZero()
boolean isZero()
```

```
MonetaryAmount add(MonetaryAmount amt)
MonetaryAmount subtract(MonetaryAmount amt)
MonetaryAmount multiply(Number multiplicand)
MonetaryAmount divide(Number divisor)
MonetaryAmount remainder(Number divisor)
MonetaryAmount negate()
```

www.tritales.nl

```
CurrencyUnit getCurrency()

MonetaryAmount with(MonetaryOperator operator)

<R> R query(MonetaryQuery<R> query)
```

Creating Money

```
MonetaryAmount money =
   MonetaryAmounts
      .getDefaultAmountFactory()
          .setCurrency("EUR")
          .setNumber(200.5)
          .create();
```

www.tritales.nl

Creating Money with Moneta

```
CurrencyUnit cur =
    Monetary.getCurrency(Locale.US);

MonetaryAmount m1 = Money.of(10,"EUR");
MonetaryAmount m2 =
    FastMoney.of(20, cur);
MonetaryAmount m3 =
    RoundedMoney.of(30, cur,
        MonetaryOperators.rounding());
```

www.tritales.nl

NumberValue

```
MonetaryAmount m1 =
    Money.of(10.21, "EUR");
NumberValue numberValue =
    m1.getNumber();
```

NumberValue extends java.lang.Number

→ convert to primitive types (long, double, etc.)

www.tritales.nl


X45448914449
EURO
EURO 10
TEN
10
Conversion

Conversion

“Changing currency by applying exchange rate”

```
M
M
MonetaryAmount result = m1.add(m2);
```

javax.money.MonetaryException:
Currency mismatch: USD/EU

www.tritales.nl

ExchangeRateProvider

```
MonetaryAmount money = Money.of(10, "EUR");

ExchangeRateProvider provider = MonetaryConversions.
                getExchangeRateProvider(ExchangeRateType.ECB);

CurrencyConversion currencyConversion =
                provider.getCurrencyConversion("USD");

MonetaryAmount result = currencyConversion.apply(money);
// USD 11.785
```

www.tritales.nl

Exchange rate on a specific date

```
MonetaryAmount money = Money.of(10,"EUR");
LocalDate localDate = Year.of(2016).atMonth(Month.MAY).atDay(10);

ExchangeRateProvider provider =
        MonetaryConversions.getExchangeRateProvider(ExchangeRateType.IMF_HIST);

ConversionQuery query =
        ConversionQueryBuilder.of().setTermCurrency(“USD”).set(localDate).build();

CurrencyConversion currencyConversion = provider.getCurrencyConversion(query);
MonetaryAmount result = currencyConversion.apply(money);
// USD 11.37200
```

www.tritales.nl


Formatting

MonetaryAmountFormat

```
public interface MonetaryAmountFormat
                 extends MonetaryQuery<String> {
  AmountFormatContext getContext();

  default String format(MonetaryAmount amount) {}

  void print(Appendable appendable,
        MonetaryAmount amount) throws IOException;

  MonetaryAmount parse(CharSequence text)
        throws MonetaryParseException;
}
```

www.tritales.nl

format()

```
MonetaryAmountFormat format =
    MonetaryFormats.getAmountFormat(Locale.US);


String s = format.format(Money.of(12, “USD”));
// USD12.00
```

www.tritales.nl

Custom formats

```
AmountFormatQuery formatQuery =
        AmountFormatQueryBuilder.of(new Locale("nl", "NL"))
        .set(CurrencyStyle.SYMBOL)
        .set("pattern", "¤ ###,###.00")
        .build();

MonetaryAmountFormat format =
           MonetaryFormats.getAmountFormat(formatQuery);

String result = format.format(Money.of(12, “USD”));
// € 10,99
```

www.tritales.nl


Operator
& Query

MonetaryOperator & MonetaryQuery

Functional interfaces

```
public interface MonetaryOperator {
    MonetaryAmount apply(MonetaryAmount amount);
}

public interface MonetaryQuery<R>{
    R queryFrom(MonetaryAmount amount);
}
```

www.tritales.nl

MonetaryOperator

```
MonetaryAmount money = Money.of(10, “EUR”);

MonetaryOperator operator = m -> m.multiply(2);

operator.apply(money);
// EUR 20.00000

money.with(operator);
// EUR 20.00000
```

www.tritales.nl

MonetaryQuery

```
MonetaryAmount money =
    Money.of(10, "EUR");

MonetaryQuery<String> query =
      m -> m.getCurrency().getCurrencyCode();

query.queryFrom(money);
//EUR

money.query(query);
//EUR
```

www.tritales.nl

Streams

Sorting alphabetically

```
MonetaryAmount[] m =
  { Money.of(9,"EUR"), Money.of(10,"USD"),
    Money.of(11,"GBP"), Money.of(26,"GBP") };

Stream.of(m)
    .sorted(MonetaryFunctions.sortCurrencyUnit())
    .collect(Collectors.toList());
// [EUR 9, GBP 11, GBP 26, USD 10]
```

www.tritales.nl

Sorting by value

```
MonetaryAmount[] m =
   { Money.of(9,"EUR"), Money.of(10,"USD"),
     M                                    )};

Strea
    .sorted(MonetaryFunctions.sortNumber())
    .collect(Collectors.toList());
// [EUR 9, USD 10, GBP 11, GBP 26]
```

EUR 10 == USD 10

www.tritales.nl

Sorting using exchange rate

```
MonetaryAmount[] m =
     { Money.of(9,"EUR"), Money.of(10,"USD"),
     Money.of(7.8,"GBP") };

ExchangeRateProvider provider = MonetaryConversions
     .getExchangeRateProvider(ExchangeRateType.IMF);

Stream.of(m)
     .sorted(MonetaryFunctions.sortValuable(provider))
     .collect(Collectors.toList());
// [USD 10, GBP 7.8, EUR 9]
```

www.tritales.nl

Reduction Methods

Predefined higher order functions in the RI Moneta, e.g.:

- `MonetaryFunctions.sum()`
- `MonetaryFunctions.average()`
- `MonetaryFunctions.min()`
- `MonetaryFunctions.max()`

www.tritales.nl

sum()

```
MonetaryAmount[] money = {
    Money.of(10, "USD"), Money.of(10, "USD"),
    Money.of(10, "USD"), Money.of(9, "USD"),
    Money.of(8, "USD")
};

Optional<MonetaryAmount> result =
  Stream.of(money)
    .reduce(MonetaryFunctions.sum());
// USD 47
```

www.tritales.nl

```
MonetaryAmount[] money = {
    Money.of(10, "USD"),
    Money.of(12, "EUR"),

Op
  Stream.of(money)
    .reduce(MonetaryFunctions.sum());
```

javax.money.MonetaryException:
Currency mismatch: EUR/USD

www.tritales.nl

```
ExchangeRateProvider provider =
  MonetaryConversions
      .getExchangeRateProvider(
          ExchangeRateType.IMF);

Stream.of(m1,m2,m3)
    .reduce(MonetaryFunctions.sum(
       provider, Monetary.getCurrency("EUR")));
// € 34.60
```

www.tritales.nl

Predicates

Predicates

Predicates can be used to filter:

```
Collection<T> filteredCollection =
  Stream.of(collection)
    .filter(predicate)
  .collect(Collectors.toList());
```

Moneta provides several predicates:

```
Predicate<MonetaryAmount> isGreaterThan =
 MonetaryFunctions.isGreaterThan(money);
```

www.tritales.nl

Predicates on currency

```
MonetaryAmount[] m =
     {Money.of(10, "USD"), Money.of(9, "USD"), Money.of(8, "EUR")};

Stream.of(m)
     .filter(MonetaryFunctions.isCurrency(dollar))
     .collect(Collectors.toList()); // [USD 10, USD 9]

Stream.of(m)
     .anyMatch(MonetaryFunctions.isCurrency(dollar)); // true

Stream.of(m)
     .allMatch(MonetaryFunctions.isCurrency(dollar)); // false
```

www.tritales.nl

Predicates on value

```
MonetaryAmount[] m = {Money.of(10,
  "USD"),Money.of(8, "EUR"),Money.of(9, "USD")};

Stream.of(m).filter(MonetaryFunctions
    .isGreaterThan(Money.zero(dollar)))
    .collect(Collectors.toList());
// [USD 10, EUR 8, USD 9]

Stream.of(m).anyMatch(MonetaryFunctions
    .isGreaterThan(Money.zero(dollar)));
// true
```

www.tritales.nl


Conclusion

Summarizing Money

- Currently no proper money support in the JDK
- JSR 354 fixes this in future JDK or as library in JDK 7/8:
  - MonetaryAmount class, encapsulates amount & currency
  - Exchange rate conversions
  - Formatting
  - Operators & queries
  - Sorting & reduction
  - Predicates

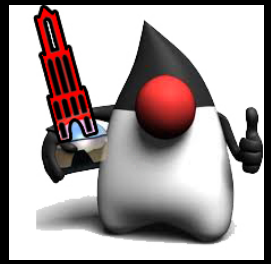
References

- https://www.gitbook.com/book/otaviojava/money-api/details
- https://jcp.org/aboutJava/communityprocess/final/jsr354/index.html
- http://javamoney.github.io/
    - http://javamoney.github.io/api.html
    - http://javamoney.github.io/ri.html
    - http://javamoney.github.io/lib.html

?

wim@tritales.nl

jeroen@tritales.nl



blue4IT

www.tritales.nl