till 08/05/2023

Scala in Practice

lab 07

Acceptance criteria:

Create DSL in Scala for non-programmers - financiers & accountants.
Your language-extensions should model *cash operations* between currencies: dollar (USD, $), euro (EUR, €) & złoty (PLN, zl):

- Addition between different currencies:

```
val sum1: Money = 100.01(USD) + 200(EUR) //result in dollars (most left type)

val sum2: Money = 100.01(zl) + 200($)  //result in złoty (most left type)

val sum3: Money = 5(zl) + 3(PLN) + 20.5(USD) //result in złoty (most left type)
```

- Subtraction between different currencies

```
val sub: Money = 300.01(USD) - 200(EUR) //result in dollars (most left type)
```

- Multiplication

```
val mult1: Money = 30(zl) * 20 //result in złoty
val mult2: Money = 20($) * 11 //result in dollars
```

- Conversion

```
val conv1: Money = 150.01(USD) as PLN // converts to złoty
val conv2: Money = 120.01(USD) as € // converts to euro
```

- Comparison between currencies

```
val compare1: Bool = 300.30(USD) > 200(€)
val compare2: Bool = 300.30($) < 200(EUR)
```

- Create package *money* with above logic:

```
trait Currency = ???

...

val conversion: Map[(Currency, Currency), BigDecimal] = ??? //map with constants (EUR => PLN, PLN => USD, USD => EUR) representing conversion rates between currencies. Put any values or use real ones from the past[1].
```

1 https://www.xe.com/currencyconverter/convert/?Amount=1&From=EUR&To=USD

till 08/05/2023

Scala in Practice

lab 07

```
case class CurrencyConverter(
   conversion: Map[(Currency, Currency), BigDecimal])  {
      def convert(from: Currency, to: Currency): BigDecimal = ???
}

case class Money(amount: BigDecimal, currency: Currency)(implicit
currencyConverter: CurrencyConverter)

...
```

- Create *application entry-point* object with some example tests for the above implementation

Note: Dont use any **nulls** & **vars**

Michał Kowalczykiewicz