**Week 4 – DQ1: Introduction**

Author: Daniel Schieberle  
Last Update: July 27th 2013  
Java Version: 1.6

**Files:**  
  
**CurrencyConverter.java**    
Class to store Currency object pairs and store their conversion rates. Provides a “convert” method to convert between two currencies.  
  
  
**JCurrencyLabel.java**    
A custom JLabel extension that displays and formats a currency value (Locale specific formatting such as $1,000,000.00 or 1.000.000,00€ is optional)

**JCurrencyTextField.java**    
Class extends JTextField and provides the ability to listen to changes immediately (values are commited as the user types). Userinput is limited to valid key strokes (0…9, backspace, del key, decimal separator). Copy+Paste is supported as well.  
  
  
**SchieberleWeek4DQ1.java**   
Class for DQ 1 “business logic”. Creates the custom GUI and provides handler methods to “react” to user button clicks, etc.

**SchieberleWeek4DQ1Driver.java**   
Wrapper Class. Main application entry point.

**Approach:**

For this week’s DQ’s I continued to find a solution for having JTextFields commit value changes immediately and notify listeners about any (valid) user input. JFormattedTextField in combination with formatter object is convenient but in most cases the value is only “written” once the textfield loses its focus. Using MaskFormatter is one option but in such case the input is restricted to a predefined length (such as $10.00).

CurrencyConverter.java

It is not recommended to use primitive floating point types for storing monetary values due to rounding inaccuracies (Goetz 2003). As a consequence the CurrencyConverter class relies on Java’s BigDecimal class (available in the Math package) to store conversion rates.

BegDecimal takes arithmetic operations very “seriously” (for instance 1/3 raises an exception as the result is infinite once a proper precision is not defined)

I opted to store conversion rates for a Currency object pair as a “hashes of hashes” to allow any object instantiated from the class to store more than just one conversion option:  
[Currency from => [ Currency to => Conversion Rate]]  
Example:  
USD => [[GBP => 1.91],[EUR => 0.8]]  
As a consequence a conversion rate is found by accessing:  
Hashmap.get(fromCurrency).get(toCurrency)  
  
Once a conversion rate is set (either by the constructor or the appropriate method), the class automatically stores the vice-versa conversion setting the conversion rate as (1 / conversion rate). Additionally the (useless) 1 to 1 conversion is stored such as USD to USD to avoid any exception from such a conversion request.  
This becomes quite convenient when using combo-boxes as for a short moment a one to one conversation is shown once the user “flips” the currencies:  
USD => GBP  
GBP => GBP  
GBP => USD

JCurrencyTextField.java

The extended JTextField implementation in the class watches its own “document” for changes and parses input on the fly. In case input is given by “typing” (not copy+paste), key strokes are limited to valid input otherwise a “beep” notifies the user about the key stroke not being accepted.  
Any change to the value (valid or not) is propagated through a custom action-listener implementation.  
The class also limits the textfield to 2 fractional digits (using regular expression) and allows the valid input to be returned as a Double.

SchieberleWeek4DQ1.java

Due to the “upgraded” JTextField conversion is now performed as the user types. For convenience a “Reset” button is still offered.  
Since entering a double is part of the requirements, the conversion is performed by converting the Double received from JCurrencyTextField to BigDecimal using the String-based constructor to avoid any rounding “flaws” to be introduced.

References:  
Goetz B. (2003) ‘Java theory and practice: Where's your point?’. IBM DeveloperWorks. [Online]. Available from <http://www.ibm.com/developerworks/java/library/j-jtp0114/> [Accessed July 28th 2013]