The “complete” I18n4Java manual

Users, Administrators and Developers Guide to I18n4Java

Rick-Rainer Ludwig

version: 1.0.0
date: 30th July 2010

This is the “official” manual for KickStart.

Contents

List of Figures

List of Tables

Chapter 1

About I18n4Java

I18n4Java is a new approach to do internationalization (in short i18n) in Java. The standard approach by using property files to store strings in specified keys and to change the files for different language output has a serious flaws:

The original message is not shown. It's wise to see the string in the source code. With the whole string in sight, one knows the context, the size and the message format and the needed parameters and their types.

I18n4Java tries to make this right. All original string in the original development language are written within the source. The Java MessageFormat facilities are used to parameterize the string with localization already included. Translators get the whole freedom with this approach to translate all string within the right context and with the possibility to reorder the parameters to get a grammatically correct translation in any language. The MessageFormat approach could be used before with the properties file approach, too, but the string were not cleanly located within the source code and it was hard and time consuming to get all information about the string to be displayed to know the size of the output, the number and format of all parameters and also the position and the context within the application. I18n4Java solves these issues.

Chapter 2

Installation

The library `i18n4java.jar` is just copied to an appropriate position and the path is added to `CLASSPATH`. Afterwards I18n4Java can be used.

Chapter 3

Use in Source Code

As soon as the library of I18n4Java is included in `CLASSPATH` a translator can be get by a simple call of

```
private static final Translator translator =
Translator.getTranslator(<class>.class);
```

Translations can be retreived by

```
translator.i18n(``<message with parameters>'',
<param1>, <param2>...);
```

That's almost all of it! The rest can be done graphically with I18nLinguist, which is the tool to scan the sources, to do the translations and to release the translations into language files. The Translator automatically translates into the language which is administrated in the underlaying OS. With

```
Translator.setDefault(<locale>);
```

the current language can be changed.

Nomenclature

ASCII American Standard Code for Information Interchange - Code table for a number to character mapping

Design Patterns Design Patterns (in german: Entwurfsmuster) are generalized approaches to repeating programming issues. For the most issues there is a fitting approach to solve the issue with a generalized design. Design patterns are well known with generalized names and software which implements and documents these patterns are very easy mainanable. Other programmers have a good chance to understand the code and the design of the software in short time. (see: [1, 2])

RefactoringRefactoring Refactoring is the process of permanent code improvement even after implementing functionality. This process is essential to keep the software code in a clean state for later use. Without Refactoring the code will get out of shape and the implementation process for new functionallity will become more expensive. Therefore, Refactoring is a way to keep code maintanable. (see: [3, 4])

Bibliography

[1] Eric Freeman & Elisabeth Freeman. *Head First Design Patterns*. O'Reilly, 1st edition, Oct 2004.

[2] Erich Gamma & Richard Helm & Ralph Johnson & John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 36th edition, July 2008.

[3] Andrew Hunt & David Thomas. *The Pragmatic Programmer*. Addison-Wesley, 22nd edition, January 2008.

[4] Martin Fowler. *Refactoring – Improving the Design of Existing Code*. Addison-Wesley, 22nd edition, August 2008.

Index