Spring 2021

EZ-Translate

Mason D. Beattie
mdbeattie@coastal.edu

EZ-Translate

By

Mason Beattie

Information Systems

Submitted in Partial Fulfillment of the
Requirements for the Degree of Bachelor of Science
In the HTC Honors College at
Coastal Carolina University

Spring 2021

Paul S. Cerkez

Louis E. Keiner	Paul Cerkez
Director of Honors	Associate Visiting Professor
HTC Honors College	Computing Sciences
	Gupta College of Science

Table of Contents

Abstract

This report describes the EZ-Translate Software that was designed and developed by Mason Beattie for the CSCI495-D1 Honors Capstone Project and Course at Coastal Carolina University. This project is built from Java and is designed for the Windows 10 operating system. While running it displays or hides a translation menu when the designated key binds are pressed. Translations supported include select phrases from English, German, and Russian. The application also provides localization support for these three languages. The overall goal of this project to provide translation services while within another application was successfully achieved. The application utilizes an open-source Java external library named JNativeHook to read the keyboard input from the host operating system.

Keywords: Java, Input, Translation, Keyboard, JNativeHook, Menu, Localization, Hooks, Windows 10, Localization.

Chapter 1

Introduction

Background

Within online communities exists many opportunities to engage in intercultural communication or to learn about another language. Digital games and similar mediums provide many situations that parallel what is considered best practices for second language teaching according to language professionals (Sykes, 2018, section 3). On this basis, the author of this report wanted to create an application that makes such learning opportunities more readily available to members of these digital communities. To do so, this author utilized their knowledge in both Java application development and foreign language to develop a new tool that could be used for learning a language.

Problem statement

The author of this project report aimed to further the ability to conduct intercultural communication amongst participants within digital communities. Despite this motive, the quality of intercultural communication provided by this project is not within the scope to investigate; instead, this project addressed the issue of efficiency when conducting intercultural communication. The current methodology for conducting intercultural communication while within a fast-paced multiplayer game is inefficient since it requires leaving the application’s window to access web translations.

Project Goal

The goal for this project was to provide an accessible way for users to translate words or phrases while within a separate application without the need of leaving said application’s window. By reading keyboard input, when a user inputs a specific key combination a translation window would appear that would allow the user to access translation services. This service of reading keyboard input and providing a translation interface was developed as the EZTranslate software. The EZ-Translate software also provides localizations other than English to increase the linguistic diversity of user’s who can use this application.

Relevance and significance

The impact of this project is defined by the conversations and dialogs enabled by increased access to translation methods. On the local level, this product’s impact is limited to the languages that exist around the users. In a linguistically homogenous area, this product will see little impact and usage. Conversely, this product will benefit global online communities that necessitate on-demand translations. The significance of this product will be seen especially in gaming, as many online games provide little time to conduct translation using conventional methods such as phone, desktop, or web applications. If users are given more opportunity to participate in intercultural dialog, then knowledge of other cultures and languages for said users could also increase significantly.

Barriers and Issues

What were the Barriers and Issues?

Key issues within this project's development included access to a free and accessible translation service. The act of creating an intricate translation service was outside of the capabilities of the author within the given timeframe of the project. Instead, an attempt was made to include free and accessible translation APIs (application programming interface) or web translation applications for the project. Barriers that were encountered included paywalls for enterprise level translation API, as well as the inability to discover a reliable open-source translation library for Java. Another issue encountered was the access to a desired application for keyboard input monitoring, and the permission required from the application to monitor its keyboard input. Additionally, accessing the native keyboard input of the host system was impossible within a purely Java application, due to it not having the support built into the language to naturally interact with native input. A barrier encountered during the creation of keyboard input monitoring was the inability to use pure Java to access native keyboard input, and the inability to access another application's source code to modify the build path to allow for input monitoring. Developing solutions for these issues were unobtainable during the project's allocated development time, so alternative solutions were consulted.

How were they accounted for?

The issue of translation API access was accounted for by utilizing a minimal translation service that can only translate a select few words within three given languages.

This was developed to retain the goal of enabling users to translate given words or phrases. Since access to another application’s source code was unobtainable during the project’s development, the reading of global keyboard input was utilized instead to enable users to access the translation service while within it. This issue of allowing Java to read native keyboard input was solved through the usage of the open-source library JNativeHook. This library allows Java applications to monitor the global keyboard input that they cannot naturally read.

Chapter 2

Literature Review

Purpose

This literature review serves as a collection and evaluation of the valid sources and how they relate to the development of the EZ-Translate software. These sources cover a large variety of topics that that were used throughout the development of the EZ-Translate application itself. Topics within this literature review include the keyboard input reading processes, graphical user interface creation, and the focus system of Java menu components. The literature review also includes sources that are relevant to the goal of providing localizations for menu components; furthermore, open-source libraries from GitHub that allow Java to monitor global keyboard input or provide language detection services are also included within this review. Finally, sources that provide methodology and insight into how to properly localize a Java application are present within this review.

Pro Java 8 Programming

This textbook is about higher-level programming techniques offered by the Java language. It was written by Brett Spell and was published by Apress in 2015. The contents of this literature are analyses and descriptions of different tools and components within the Java

8 application development environment. This book was used in consideration for the graphical user interface (GUI) and user interaction with the application itself. The book included Cut-and-Paste functionality that would have been used for the translation interface. Use of a Transferable wrapper class, as defined by the book, would have been utilized to ensure that the text boxes would have Cut-and-Paste support (Spell, 2015, pp. 381-383). Providing Cut-and-Paste support to the application's textboxes would have allowed for the translation service to satisfy the project goal of providing translations to users through this application, since typing a phrase manually would have inherently slowed down the translation process. Unfortunately, this section from the book was not necessary to include within the overall project, due to many of the Java Swing GUI components used already possessing cut and paste support. Because of this, the book's information was not utilized in the final product.

JNativeHook: Global keyboard and mouse listeners for Java

JNativeHook is an open-source global keyboard and mouse listener for Java created by GitHub users Alexander Barker, Aidas Adomkus, Johannes Boczek, and Amir Sangi. The GitHub page itself provides an overview as to what the project is about, the licensing information for the GNU General Public and GNU Lesser General Public licenses, download links, software, and hardware requirements, as well as documentation for how to use the library's API (Barker et al., 2017). As described in the about section, the JNativeHook library is a way for a Java application to monitor the native keyboard and mouse input to the host system that is normally unreadable for the Java language. This library performs this functionality by utilizing native code that is supported by Apple OS X 10.5 – 10.12, Windows 2000-10, and X11 Linux systems (Barker et al., 2017, section 1 para 1). Because this library performs tasks that are needed for the project while also conforming to its hardware and software requirements, it is included as an external library by the project. The global events monitored by this library that are relevant to this project include key press events, key release events, and key typed events (Barker et al., 2017, section 1 list 1). Using the JNativeHook API global key pressed events are monitored by this project to detect when a user presses keys while it is outside of keyboard focus. This GitHub page provides information on how to utilize the JNativeHook API as well as how to install and utilize the project's associated jar file. The GitHub page is weak in that it does not explain in depth the functionality and performance of the library itself.

Software quality: Concepts and practice

This piece of literature is provided by the IEEE Press and IEEE Computer Society and was written by Daniel Galin. The document was published in 2018 through the IEEE Computer Society. Within the text itself includes definitions and assorted topics pertaining to software quality; additionally, methodologies and management techniques required for maintaining quality during software development are listed. This literature was referenced to assure that the delivered product complies with IEEE quality standards for software development. This includes the definition of the quality policy and SQA (Software Quality Assurance) processes, the tasks conducted by individuals chosen to assure SQA processes, the definition of a managerial communication tool/method, and finally the development of a follow-up and review methodology for said SQA functions (Galin, 2018, p. 107). The SQA processes will be defined within the related Software Requirement Specifications document as well as the Test Plan document of this project. Inclusion of these standards are done to ensure the predefined quality within the product and its compliance with IEEE standards. Furthermore, IEEE recommendations for Software Configuration Management (CM) will be conducted to assure that new document versions in addition to code variations are tracked to conform to configuration management procedures. Tasks that were conducted to comply to this standard were the proper storage of identified versions and other software configuration items, the release of the various software configuration versions, efforts in information systems to ensure safe recovery of stored data, and finally controlled changes to affected software (Galin, 2018, p. 523). This endeavor experienced variation to the requirement documents as well as variations within the software versions. Compliance to this document and Software Quality procedures ensured proper documentation throughout different document versions and product iterations. This source provided a strong quality standard for the project to be based on to ensure that the project itself was delivered to the utmost quality. A certain weakness of this literature would be that it is impossible to comply to every single quality within it. Due to this project's limited team and development window, exact conformity to such extensive documentation requirements were not achievable; however, the document's recommendations were utilized for the creation of archiving and data protection procedures throughout development.

Digital games and language teaching and learning

The article, Digital Games and Language Teaching and Learning, is written for the Foreign Language Annals volume 51, issue 1, and was written by Julie M. Sykes who is an Associate Professor of Linguistics at the University of Oregon. The article was published on March 15, 2018, and it is a brief description of the potential linguistic benefit of the digital gaming industry, regardless of if the digital game was designed for entertainment or educational purposes. Benefits of such incorporation are emphasized within the conclusion of this article. As described by Sykes, the digital world provides a unique language learning environment that benefits learner-directed goal orientation, opportunities for interaction, just-in-time, individualized feedback, relevant narrative, and context, as well as motivation (Sykes, 2018, section 3, para 2). For this project, EZ-Translate, these relevant points of information are utilized to justify the means of creating an application that enables translation services while within a digital game or other computer application. This project's goal of enabling intercultural communication and providing the opportunity for someone to learn a second language while within a game is supported by the benefits listed from Sykes. The article has strengths in explaining the greater learning potential of digital games, as well as providing specific reasons as to why they are useful for learning a second language; however, the article is weak in the depth of its explanation of such topics. All the points about the benefits that it makes are not explained in detail, and the overall content of the article is a review of concepts potential benefits rather than a tangible study.

Java programming tutorial programming graphical user interface (GUI)

The Java programming tutorial programming graphic user interface (GUI) webpage is on a subdomain of the Nanyang Technological University in Singapore and is written by the associate professor Chua Hock-Chuan. The purpose of the website is to provide a tutorial on how to create GUIs that utilize Java resources including JavaFX or Swing. The different sections of the webpage include definitions and concepts of Java GUI using the Abstract Windowing Toolkit (AWT) API, utilizing AWT Event-Handling, the different Layout Managers within the AWT toolkit, a tutorial of Swing and its API, and finally using GUI builders within NetBeans and Eclipse integrated development environments (IDEs) (Chuan, 2018). The information within this webpage serves as the basis for how the GUI of this application is conducted. The literature discusses Swing and how it benefits a system compared to strictly AWT components. For instance, Swing components are described as being lightweight, whereas AWT components are heavyweight. This refers to the overall system impact of the two methodologies and how Swing components are less resource intensive than purely AWT components (Chuan, 2018, Section 7.2). Such explanations were useful for the project, since they helped create a better understanding of what would have been a less resource intensive option when creating the GUI components. The webpage itself is also a useful tool for creating and adding functionality to a Java GUI. The explanations provided were enough to create a working knowledge foundation on the concepts of Java GUI building for use within the project. A weakness of the webpage is that it does not provide extensive information about creating complex GUI systems, but rather provides a strong working foundation for them instead.

Hooks Overview

Hooks Overview is an article on the online Microsoft documentation forum for functionalities and concepts within the Windows Operating System. This forum is posted as official documentation to the Microsoft Documents online forum, where information about multiple topics pertaining to the windows operating system is available. The article has multiple contributors listed on GitHub including Shawn Hickeys, whose username on the forum is “hickeys.” The contents of the article include a brief description of windows hooks as well as information pertaining to hook chains and hook types. Additionally, procedures and example code for how to utilize a hook are included. Hooks are relevant to this project because, as described by the article, a hook can be used to detect inputs, keypresses, or mouse movement while outside of an application (Hickey et al., 2018, para. 1). For this endeavor, the keyboard hook was intended to be used. The WH_KEYBOARD hook is described as a hook that will specifically allow an application to read keyboard button presses and lifts for the windows operating system (Hickey et al., 2018, para. 21). Use of the WH_KEYBOARD hook would have allowed for the translation window to be called while the application is outside of keyboard focus. The forum also includes information pertaining to thread-specific hooks. A thread-specific hook would have been used over a global hook due to the global hook's performance setbacks compared to a thread-specific hook (Hickey et al., 2018, para. 10). In relation to the project a thread-specific keyboard hook would have allowed for the application to require less system resources while also detecting keyboard input from outside of the application. In the end an open-source library was used to detect global keyboard input instead of manually creating the system for the project. Despite this, the documentation provided by this forum was relevant and applicable for the project's needs. If an open-source solution was not discovered, then this webpage would have been referenced throughout the development of this project. The documentation is extensive, and the information provided is up to date and relevant for Windows 10 system; however, the article does not provide intuitive information on how to create these hooks for personal use, and it instead serves as a definition and explanation of the concepts themselves.

Lingua

Lingua is an open-source language detection software on GitHub. The repository was created and is owned by Peter M. Stahl. Other contributors include Bernhard Geisberger and a user by the name of "maltaisn". The GitHub page provides information pertaining to the functionality of the library, the requirements, small tutorials to install and utilize the library, as well as a section dedicated to comparing the accuracy of the product to similar language libraries. As described by the GitHub page, the library translates small words and phrases and provides the detected language using the included language library that contains 74 different languages (Stahl, 2020, para. 3). The output of the library API is also customizable and provides functionality to display what languages the word or phrase is closest to (Stahl, 2020, section 9.1, para. 2). In this project, the Lingua library was intended to be imported so that the language detection service could detect the language of a word or phrase. This feature would have allowed for the original language of a phrase being translated to be detected. The Lingua library itself was also compatible with the project requirements, as it supports Java 6 and higher applications (Stahl, 2020, para. 1). Such satisfactions towards project hardware and software requirements would have allowed the library to be integrated into the project without drastic changes in the requirements. The GitHub page for the library is intuitive and provides substantial information as to what the repository contains and what it is capable of. A weakness of the repository would be that the information provided does not include extensive information about how the product should be compiled without the use of Maven or Gradle.

Class ResourceBundle

Class ResourceBundle is the official Java documentation on the Oracle website for the ResourceBundle class that is included within the Java utility package. Throughout the webpage are detailed documentation for all methods included and required by the abstract ResourceBundle class. Also included within these explanations are examples on how to create and utilize each method and how to implement the ResourceBundle class itself. As defined within the webpage, a ResourceBundle object contains specific locale data for creating language localizations (Oracle, n.d.-a, para. 1). Through implementing a ResourceBundle class, localization functionality was available to an application. The webpage describes the structure of the ResourceBundle class and all other classes that inherit from it as a family that shares a common base name, and these base names can be utilized to determine the different localization that is associated with it, such as Resource_de for German (Oracle, n.d.-a, para. 2). This functionality was utilized for creating different language resources for the localizations available in the application. Within the methodology documentation of this webpage a method named getString is listed. This method can return a localized string based on the key that is used as input. This functionality is identical to the (String) getObject(key) method of any Java object, and because of this can be used to store key values based on a menu item to return the associated localization item (Oracle, n.d.-a, Method Detail 1). Through this, menu strings can be incorporated to return string items based on the given input. The string returned would be based on the current localization using this method and class. This methodology allowed for menu items to be dynamic in their localization setting. This webpage presents a strong description of what the tasks the Java ResourceBundle class performs. The explanations are detailed in explaining each component of the class, and further information about how to utilize the class is presented. A definite weakness of this webpage is that the information is not presented in an intuitive way. Fully understanding the page requires thorough reading and comprehension due to how it is structured.

How to Make Dialogs

How to Make Dialogs is a Java tutorial located on the official Java documentation section of the Oracle website. The webpage provides information on how to create different dialog boxes for a Java application. The overall purpose of a Dialog window in a Java GUI is to provide temporary notice that is separate from a main application frame or window (Oracle, n.d.-b, para. 1). The webpage also describes that the dialog boxes can designed to be modal, which means that when there is a visible dialog box, a user's input to all windows within the program is blocked (Oracle, n.d.-b, section 2 para. 1). The dialog windows described in this webpage can be utilized to create notifications that require a user's attention before the program progresses, which can then be utilized to create startup or shutdown notifications. Java dialog windows can also be modified to display different information based on the showMessageDialog method's input. This input can range from plain messages to error notifications (Oracle, n.d.-b, section 3 images 1-5). Such functionality was used to specify what type of dialog box was being displayed. The information provided by this webpage was thorough in its explanation on how to provide dialog boxes in a Java GUI, and the capabilities of said dialog boxes.

How to Use the Focus Subsystem

How to Use the Focus Subsystem is a webpage that is located within the Java Tutorials section of the Oracle website. It includes specific information about the functionality of frames to change keyboard focus. Throughout the webpage the specifics of such focus methods are explained, and examples of using different methods to gain keyboard focus are shown. The text explains that for a Java window to gain keyboard focus it must use the Window.toFront method (Oracle, n.d.-c, para. 3). The methodology described was utilized by the translation window to bring it to the front of the screen for display. Also, within the webpage is a table of the Focus methods that can be used to change keyboard focus for components within a window. For example, the requestFocusInWindow() requests that the designated component be given window focus (Oracle, n.d.-c, table 1 row 5). This method is employed to bring focus to the translation window, thus ensuring the goal of simplified translation due to the lack of a necessity to select the textbox manually using the mouse. The information provided by the Oracle tutorial webpage allowed incorporating the window focus methodologies as intuitive and efficient as possible without consuming too much time from development.

How to Use Key Bindings

The webpage How to Use Key Bindings is in the Oracle website under the Java tutorials library. Within this webpage is a tutorial of the methodology for writing and listening for key bindings. Also, included within this webpage is how key bindings operate, functions of the key bindings class, and the specifics of the key binding API. The key binding methodology was supposed to be utilized by this application, but since as described on the page, key binds are nonfunctional outside of the focused window of the keyboard (Oracle, n.d.-a, para. 3). Because of this limitation, another method for detecting the global keyboard input was utilized. Also, within the webpage is documentation and utilization for the InputMap and ActionMap classes. As introduced on the page, "An input map binds keystrokes to action names, and an action map specifies the action corresponding to each action name" (Oracle, n.d.-a, para 5). Therefore, the utilization of these classes was intended to satisfy the key bind creation as well as the action associated with the key bind; however, this API proved to be difficult to use outside of a Java component, so another methodology was utilized by the application. Overall, the How to use key bindings webpage provided enough information about key bindings to discovering what was unnecessary with implementation.

How to Write a Key Listener

The How to Write a Key Listener webpage is located within the official Java Tutorials section of the Oracle website. The content of this website is a description and tutorial on how to utilize the Key Listener API to determine Key Events while within a Java component. As described by the webpage, for a Java component to monitor key presses the isFocusable method of the component must return true, and the component must also request the focus when it is appropriate (Oracle, n.d.-e, para. 4). The webpage also includes information about the KeyAdapter that is utilized by the KeyListener interface. The KeyAdapter has three associated methods, keyTyped(), keyPressed(), and keyReleased() that are all used to determine what state the key action is in (Oracle, n.d.-e, table 1). All three of these methods take a parameter of a KeyEvent class, that has an associated getKeyText() and getKeyCode() method (Oracle, n.d.-e, table 2). Both KeyAdapter and KeyEvent classes and their associated methods are relevant to this application as they were used to detect what keys a user was typing during the creation of custom key binds. This webpage was a strong source that provided extensive documentation and resources for the KeyListener interface. It explained in detail what is required to utilize a KeyListener, as well provided additional tutorials on how to include a KeyListener within a Java project.

How to Write Window Listeners

This Webpage, titled How to Write Window Listeners is a Java tutorial located on the Oracle website for official Java documentation. Information on how to utilize the window listeners interface and related classes for Java frame components is included within this webpage. The WindowListener interface and all methods included within it are used to detect and handle actions created by window events within a Java frame component (Oracle, n.d.-f, para. 4). Events that are monitored by a WindowListener include: opening a window, closing a window, iconifying a window, deiconyfying a window, and maximizing a window (Oracle, n.d.-f, para 3 list 1). These events enable Java frames to monitor activity to the frame component itself; therefore, window listening can be implemented in the project to provide customizability to a window component, so that it can determine which event occurred. Methods used by this interface to detect such activity are described within the WindowListener API documentation table of this webpage. For instance, the windowOpened method is called when the window that it is listening to has been shown, and the windowClosing method is called when the user requests the listened window to be closed (Oracle, n.d.-f, table 1). Such implementation allowed for custom window opening and closing events to be triggered based on user's requests to the window. For the application itself such resources were used for performing specific closing tasks when a window was closed to prevent unintended data loss within menus. This webpage has strengths in providing a methodology for creating custom window functionality based on the activity of the window itself. Weaknesses of the webpage include lack of reliable information with the potential on using a WindowListener, and lack of information about whether these listeners work with the Java Swing components that are utilized by this project.

Summary

This literature review covered topics and concepts that are relevant to the EZ-Translate application. Through this information, knowledge about how to implement many different items within Java applications was acquired. Knowledge gained that pertains to Java application development includes how to incorporate a localization to a Java project, different methodologies on how to monitor keyboard input through a Java project, how to create a custom and flexible Java GUI, and the focus requirements for input monitoring. Knowledge gained that does not pertain to Java application development include the benefits of digital games in learning a second language, as well as the practices and structure of creating a quality computing project. Information that remains unknown include how to handle strings that include non-recognizable characters; furthermore, it is unknown as to how to remove keyboard input from a given Java component so that multiple instances of input monitoring do not exist at the same time.

Chapter 3

Methodology

Resources and Documentation

Tools

The EZ-Translate Application was created for and tested in the latest version of Microsoft Windows 10, which at the time of this report's creation was version 10.0.19042. Tools used during this project's development include the Java SE Runtime Environment build 15.0.2+7-27 and the Java Development Kit version 15.0.2. for both the creation of and execution of the application. Further tools used include the Eclipse IDE version 4.19.0 which was used for the creation, and package management for the application itself. Apache Maven version 3.8.1 was briefly utilized in the development of this application, but it was not fully implemented due to time constraints in the latter half of development. In the initial stages of the project's development, Oracle Virtual Machine Virtual Box version 6.1.18 was intended to be utilized as a controlled environment for the application's testing, but it was dropped from the requirements since it was unnecessary to achieve the project's goals.

Development Information

To conform with CM requirements, documents associated with this project were assigned version numbers, and changes to documents were maintained within the document. Previous document versions were also stored within a project archive; likewise, code versions were stored using a GitHub repository created for the project (Beattie, 2021). Commits to the repository were maintained by the GitHub website for version control, and the repository served as an external storage for the project in the case of hard drive failure or other destructive incidents. All measures were taken to ensure project quality and keep track of the overall project development in accordance with the Software Configuration Management definition in the Software quality: Concepts and practice text (Galin, 2018, Frame 25.1). Further documentation was also created alongside this project. These documents included an initial Project Proposal, Software Requirements Specifications, Test Plan, Test Cases, and a Project Development Plan. Such documents were created to track and evaluate the overall project development process.

Testing

During the EZ-Translate project's development testing was conducted throughout development, and each test was based on the Test Cases defined within the associated Test Case and Test Plan documents for this project. All tests were used to determine if the project meet the requirements, and executed all tasks required of it. In the initial conception of the project's Tests Cases, some were designed to include automatic testing methods. These methods were dropped due to time constraints in development to create and integrate such methods into the overall application. Additionally, a test to determine the delay of a keypress being registered by the system while global input was being monitoring was intended on being used; however, due to inclusion of an open-source interface for monitoring keyboard input this test was never conducted. The reason for this is that the format in which the time that they key was being pressed and interpreted by JNativeHook was not the same as the methodology for time stamping utilized by Java. The results of both were unusable to determine the time between key press and the system's recognition of the keypress; furthermore, the development time required to create a solution for this issue was not available.

Product and Methods

General Project Structure

The EZ-Translate Application involves four internal packages used to store the main project files, configuration resources, language resources, and service resources used throughout the application; additionally, a library file was included in the project's directory to serve as storage for the project's referenced external Java libraries. The project itself was exported as an executable jar file from the Eclipse IDE using its functionality to export projects as executable jar files.

Running the Application, Startup and Shutdown

The application's main method was inserted into the EZTranslate class, and the sole functionality of this class was to create the ApplicationManager class object and call its startup method. The constructor of the ApplicationManager class was designed to perform the bulk of the application's startup functions. It creates all objects utilized by the application, and initializes all variables used within it. The order of object creation starts with the creation of the Configuration object and the calling its method to read the configuration file's contents. This method returns either TRUE or FALSE based on the success of the file reading, and the return value is stored for later use by the application. After the configuration is created, the localization value is retrieved from the configuration object to determine what initial localization is used by the application. Localization retrieval still functions regardless of the success of the configuration loading, because if a value is not retrieved from the configuration the localization defaults to English. After these two steps the remaining GUI and notification components are created by the constructor. The reason for this structure is because all menus require a localization object to be created to retrieve localization data, and the creation of the localization requires the configuration object. This was not initially the case for the startup process, as it began with creating all objects then reading the file. This did not work for the application's purposes, because the menu items would not start with the appropriate localization received by the configuration file.

Once the ApplicationManager object's construction is complete, the startup method in the EZ-Translate class executes. This method determines if file reading was a success or failure, if it was a success then a confirmation notification is displayed, and then the main menu of the application is displayed. If the file reading was a failure, then an error message is displayed, and the application shuts down.

The shutdown methodology of the application first calls the method to update the configuration file in the configuration object. If updating the file is successful, then a shutdown notification is displayed notifying the user that the file was successfully updated, and the application is shuts down. The main menu is then disposed, and the system exits the application. If updating the file was a failure, then the application displays an error notification and exits the application.

Configuration

The configuration of the application includes the current localization value, and the key binds used to open or close the translation interface. During runtime, these values are stored as private variables to the class, and when these values are changed the configuration file is updated. These values are given relevant methods to access them individually or to return all of them in a specific format. Configuration items are stored and retrieved in a text file to ensure that the application's configuration is maintained between launches. This file is a visible file that contains a single line in which the configuration values are stored. These values are separated by a delimiter defined as "." which is stored as a constant within the class. This delimiter value is applied to the Scanner object so that it may read the values stored between delimiters from the configuration file. The values stored between the delimiters are the values of the localization and key binds. An issue encountered when using this delimiter was that the data was not properly stored and read through it. To fix this, two back spaces "\\" were concatenated to the beginning of the delimiter string before it was attached to the scanning object. After implementing this fix the file reading was successful. The error was caused because a single period "." is already defined as a delimiter, so the escape sequence "\\" must be used to declare that the period is purely a text value.

File reading takes the configuration line and attaches a scanner object that uses the defined delimiter. It then iterates through the string and stores the values internally. If no line or value is found the values default to being assigned as "NOT_CONFIGURED." Storing to the file simply overwrites the current contents and concatenates the values separated by the delimiter. Afterwards, the line is written to the file. If either of the file writing or reading methods encounters an input or output exception, then these methods return FALSE to notify the application manager to display an error notification.

UTF-8 encoding

When creating the file contents, an error related to UTF-8 (Unicode Transformation Format) encoding was encountered. Because of the configuration's values containing either Cyrillic or Germanic symbols, errors ensued that caused randomized strings to be returned rather than the intended language character. After research on the issue was conducted, sources about encoding UTF-8 values in Java were discovered. The methodology for writing UTF-8 encoded values to a file utilized the code described in Appendix B. The methodology encodes by first creating a file output stream to the designated file. This output stream is then inserted into a new output stream writer object alongside the charsets for UTF-8 encoding, then that output stream writer object is used to create a buffered writer object. This buffered writer object then writes a string containing UTF-8 characters to the file.

Reading from the file also had the same encoding issue. The methodology used by the application for reading UTF-8 encoded files is detailed in Appendix C. As described in the code, a file input stream is created through a file object, then an input stream reader object is created through both the file input stream and the UTF-8 standard charsets. Finally, a buffered reader object is then created from the input stream reader object. This reader is then used to store the configuration line for internal use by the application. Through the utilization of both encoding and decoding methodologies a UTF-8 encoded file was successfully read or written by the application without altering the data.

Alongside the issues of reading and writing UTF-8 encoded files was the issue of UTF-8 encoded strings within the application. After continued research a method of encoding UTF-8 to strings was found. The code used to perform this encoding is detailed in Appendix A. The first step was to get an array of byte values using UTF-8 standard charsets encoding as the basis of the byte collection. This encoded byte array is then used alongside the UTF-8 standard charsets to create a new string that now contains UTF-8 characters. Whenever a string in the application is processed that contains UTF-8 characters, this methodology is used to process it to ensure that the special characters are not modified.

Key Binds

Key binds used by the EZ-Translate application were designed to be created by the user and are created for displaying or hiding the translation interface when keyboard input monitoring was active. Key listeners were used to determine what keys the user pressed when configuring a key bind. A key listener could be used to dynamically determine which key was pressed when a component the listener was configured to was focused. Furthermore, the key listener interface when attached to a component creates key event objects (Oracle, n.d.-e, para. 1). A key event possesses methods for receiving the text of which key was pressed (Oracle, n.d.-e, table 2 row3). Because of this, when a key listener creates this key event, text can be received that describes if the key event was “Shift” or a single letter “A”. Through this methodology, the application repeatedly adds the key event’s text to a string if the key listener is active. The application utilizes theses key listeners to attach them to a component so that input to the component can be translated into key binds.

How the application specifically monitors key input when key binds are being created is by adding an action listener to a button that when clicked follows the steps described in Appendix H. The first section describes the action listener created for the key bind configuration menu. The first action is to remove the key listener. This is because during testing it was realized that multiple key listeners could be active on a component if the menu were selected multiple times during runtime. This removal of the listener ensured that multiple listeners were not created on the same component. The internal binding stored within the menu and the key bind retained by the configuration are cleared to ensure replacement is successful. The text field to display the current binding configured is also cleared to provide visual feedback to the user. Then the text field the listener is attached to grabs keyboard focus. The key listener code simply concatenates the key event's text that is received by the listener to the internal binding string stored by the menu. Each key press is also concatenated with a space for separating the individual presses. The key text is retrieved using the KeyEvent's key code value. Testing of keyboard input from different keyboard layouts revealed that the key event does not properly recognize international keys. To accommodate for this the special key event object's key code can be retrieved using the getExtendedKeyCode() method. This method expands what is capable of the original getKeyCode() method to include international characters. The text field is then updated to the current stored bind. The reason why the text field is updated while receiving input is for visual feedback purposes, so the user can visually see which keys they are pressing. Once key binds are saved, input reading ceases and the key bind is verified.

Key bind verification was designed to prevent any major key bind used by the operating system this application is designed for. This decision was made to prevent any potential conflicts between the two systems when a key bind also used by the Windows 10 operating system was pressed. The code detailed in Appendix D describes what is not allowed to be included within a key bind, not what specific key combinations are allowed. The Microsoft webpage detailing the used system key binds for the Windows 10 operating system was used as a reference for creating this verification system (Microsoft, n.d.). Determining what combination is not allowed utilizes a switch and case system from Java, while the methodology to detect what is not allowed to be used within a bind checks if the key is within the key bind string. The verification also checks if the key bind is empty, which also results in a verification failure. If a key bind passes, the configuration stores the result and updates the file. If it does not pass verification then it is updated and stored as “NOT_CONFIGURED”.

Localization

The localization of the application utilizes a resource bundle stored within the application manager. This resource bundle is accessed by all GUI items that display text within the application to localize the text contained within them. By implementing the resource bundle class, the methods for handling the getting of object or retrieving of the keys that are assigned to localized values were required (Oracle, n.d.-a, para. 11). Code examples of how to implement a ResourceBundle class are described within Appendix I. The application based the localization class design off the structure defined in the appendix. The method to handle key sets was not included, due to lack of requirement for overriding key implementation for the application; otherwise, the method structure of handleGetObject() and getKeys() was replicated. The handleGetObject() method stores values by a repeated if statement where the value of the parameter key is evaluated for similatiries. Upon a match the method returns the string associated with the matched key. In the getKets() method all key values defied in the prior method are stored within a hash set for string-based arrays. The return of the method is the stored list. These two methods are replicated in the application to ensure localized values are accessible by menu components.

Within the resource package of the application exists four localization classes, one that implements the resource bundle interface and provides the default localization. In the case of this application the default localization is English. The other three are sub classes of it, and they contain data specific to either English, German, or Russian. During development specific subsections of these languages were not considered due to development constraints, like British English or Swiss German. Because of their implementation of the resource bundle class the handleGetObjects() and getKeys() method are included in all localization classes. The key value relationship in the get objects method returns the localized string based on the language of the localization class for the respective key. The localized information was obtained by initially listing all English labels, then converting the words and phrases into the appropriate language for the localization.

Changes to the localization are retrieved by the localization menu and stored within the configuration file. During runtime, a ResourceBundle object is created within the application manager that is set to the current localization. Initially the value stored for determining the localization would have been the actual value of the localization, rather than a numerical value; however, testing proved that this was prone to error. When using the text value method, the localization would always default to English, even though the configuration file was storing it as either German or Russian. The localization assignment in response to this error was changed to instead receive a numerical value. Based on if the value were 0, 1, or 2 it would assign the appropriate resource bundle of English, German, or Russian, respectively. This assignment was through the resource bundle's static method of retrieving a bundle, where the package structure of the localization file desired was inserted into the ResourceBundle's getBundle() method. If an incorrect numerical value was received, then the application would default to the base localization which was English. This methodology performed better than the prior method, so it was utilized throughout the rest of development. After localization determination was complete, the application manager would notify all menus to update their text to make the localization change immediate. When updating text, all menus reuse the setText() method of the component to update the text within it, however there was difficulty in changing the localization for the JComboBox components used. Setting the text through this method would not work, so a temporary array is created that stores the choices from the localization. The contents of the combo box are then removed, and each item from the newly localized array is added to the box.

GUI of the Application

All GUI components of this application were designed to inherit from the same Menu class, which functioned as a generic class used to define all commonalities between interface classes. The generic Menu class also inherits from the Java Swing JFrame class to allow access to JFrame methods throughout all interface sub classes that inherit from it. This design was realized during initial GUI development when common traits between all menu classes were noticed, and the Menu class was implemented to keep common changes to GUI components consistent throughout development. The Menu class provides implementation for hiding or displaying a menu by calling the inherited JFrame method setVisible. Also, within this menu class are abstract methods that either add or remove the action listeners for input components, as well as an abstract method used to update the text of a menu item when the localization is changed.

The code structure of each menu was initially based on the example provided by Chua Hock-Chuan in his webpage tutorial and explanation of Java Swing components (Chuan, 2018, section 8.4). The example creates a Java Swing container object that is then assigned a specific layout. The GUI components are then initialized, and subsequently added to the Container. Afterwards, the visibility, title, and size of the container is set using the inherited super methods from the JFrame class. The template this structure is based on is displayed within Appendix F. This template was utilized to create the initial concept of each GUI item of this application, as the design was generic enough to allow for flexible development. The layout utilized by this application was decided as the FlowLayout, since it required the least amount of configuration for creating interfaces. Initially this layout was intended on being changed, but it was kept due to a distinct lack of requirement for it to be updated to a more complex layout. Each menu contains different Java Swing components such as JButtons, JTextFields, JTextBoxes, and JComboBoxes. These components are assigned their labels and values, then added to the content pane.

At the beginning of development, the GUI followed the same AWT Event-Handling structure for creating action listeners for menu components as defined in Chuan's template from Appendix F. This structure creates the action listener and adds it to the component within the same code block. While this structure did work throughout most of the development, it began to fail when menus that read keyboard input were redisplayed after being closed. The action listener associated with a button was unintentionally duplicated in this process. This error caused an action to be interpreted twice, even though it was only activated once. To overcome this error, all menu objects used for action listening were moved from being declared in the constructor, to being declared as a private object outside of the constructor. The initialization of the ActionListener object was then included within the constructor, and the adding of the action listeners to components was moved to its own separate method. This newly created method became the abstract method for all menu classes, and soon after, an abstract method to remove the listeners was included in the generic menu class also. The method to add action listeners to components was then added to the display method, and the method to remove them was added to the hide method. This implementation fixed the duplicate action listener error, and thus allowed for menus to be opened and closed without creating multiple action listener objects.

Once this error was solved another was quickly noticed. Initially the JFrame setDefaultCloseOperation super method was configured to hide all menus when they were closed through the frame’s close button. When the frame was reopened the duplicate action listener error reoccurred. When consulting Chuan’s webpage, his solution to change the default close operation of a window was to add a window listener interface to the container and call the windowClosing method from it to detect window activity (Chuan, 2018, Section 3.3). To understand the window interface to improve its implementation in the program, a tutorial on the Oracle documentation website was consulted. To utilize the window listener interface in the application, a WindowAdapter object is required, since WindowAdapter objects implement window listener functionality (Oracle, n.d.-f, para. 1). Because of this functionality, a window adapter object was created in all menus to detect when the frame window was closed. The default close operation of all menus that implemented a window listener was also changed to do nothing so that the listener handled the action performed instead. This solution fixed the issue, and it also opened the possibility of changing the default close operation of the main menu. Through this it could perform all normal shutdown functions for integrity of the configuration when closing the window through the window’s frame.

Menus

The main menu of the EZ-Translate application was created for the user to access the configuration settings or start the detection service. When the menu is closed the application begins the shutdown process through the application manager. When the start button is selected, the configuration menu button is disabled, and the button to start the keyboard input monitoring is replaced with a button to stop it. Depending on the button selected the application manager is notified to either start detecting keyboard input or stop detecting it. When selecting the configuration button, the application manager is notified to display the configuration menu.

The configuration menu displays the current configuration settings within a non-editable text field. When configuration settings are changed, the text field is updated to include the new values for both key binds and the current localization. The menu also includes two buttons that display the either the key bind configuration menu or the localization configuration menu. Initially this menu was to also include and display current applications that are connected for input monitoring purposes, but that button to access the application configuration menu was removed when connecting applications for input monitoring was removed from the requirements. When the configuration menu is exited, both the key bind configuration menu and localization menu are also closed to ensure configurations can only be changed while within the configuration menu.

Key binds for opening and closing the translation interface are configured in the key bind configuration menu. When a button to create a key bind is selected, the user's keyboard input is monitored using the aforementioned methods to create the key bind string. Both key binds can be configured simultaneously, and input ceases to be read when the save key binds button is selected.

Changing the localization is based on the localization configuration menu. The dropdown box contains the current available localizations of English, German, and Russian. When the user selects an option and pressed the button to update the localization to the selected language, the application manager automatically changes the current localization used by the application.

The translation interface was designed to include an editable text field component, where the user inserts a word or phrase, then selects a language abbreviation from the dropdown box to choose which language the word or phrase would be translated to. The languages supported by the translation services of this application include English, German, and Russian. When the translate button is selected, the text field data and the numerical index of the combo box is sent to the application manager which then retrieves the translation results to be displayed. Error results are also displayed within the text box. Errors include index issues from the dropdown menu, or unrecognized words or phrases being inserted.

To satisfy requirements, a methodology for focusing on the translation window when displayed was needed. For bringing the window to the front of the view two methods from the Oracle webpage on utilizing focus in window components were used. The Window.toFront method is inherited from the JFrame class, and when called on frame components they are shifted to the front of view so that they can be granted keyboard focus (Oracle, n.d.-c, para. 3). Furthermore, the frame also calls the requestFocusInWindow() which was also inherited from the JFrame class so that focus can be brought to the menu for display over other windows (Oracle, n.d.-c, table 1 row 4). Through these two methods the translation interface moved to the front of all other menu components, and it was also granted keyboard focus for input reading.

Notifications

Throughout the application, a necessity to prompt the user on different internal events was recognized. To incorporate such a system into the program the Java Swing JOptionPane class was used. This solution was found on the Oracle website and provided a way to display messages to the user in a way that was separate from the main GUI (Oracle, n.d.-b, para. 1). These dialog boxes were utilized by the program to display brief information about the startup and shutdown of the application, as well as the startup and shutdown of the global keyboard input detection service. The code used for these notifications was based the code defined in Appendix G. The code utilizes a static method from the JOptionPane class to display a given input message with a modifier to describe it as an error message or not. The utilization of these

JOptionPane dialog boxes enabled the program to dynamically display the system's status on an operation. As with all displayed text in this application, the messages are localized to the current localization. These notifications are explicitly used by the application manger class since most critical events of this application are handled through it.

Translation

High quality translations were never intended on being provided by the EZ-Translate software, since the initial plan was to utilize the translation API from Google Translate and similar enterprise level services. An open-source translation service discovered that matched the project's requirements was the open-source Java supported language detection library Lingua version 1.0.3 by Peter M. Stahl. The Lingua detection library was intended to be used by the application to detect the language of a given phrase before it was translated into the desired language. During development however, inclusion of the library was becoming too difficult due to how the downloading of the zipped archive from the internet caused the language files to become improperly encoded. Additionally, the library when downloaded to eclipse would not be included within the project's build path and was reporting errors with imports within the code. Due to the development time constraints Lingua was dropped from the requirements.

The translation solution developed for EZ-Translate for translating a word of a given language into another was by simply determining the equality of two String objects. In the translation service class three private arrays consisting of String objects were stored. Exact arrays used and words or phrases supported by the application are defined within Appendix E. These lists were obtained through initially starting with the English phrases that were intended on being used, then converting them to the corresponding language. These arrays contained words or phrases from either the English, German, or Russian languages. Each equivalent phrase from each of the three languages was stored at the same index in all three arrays. Before translation begins the desired language to translate to was determined by interpreting the received index.

This value was either 0, 1, or 2 and represented English, German, or Russian, respectively. Depending on the intended language to translate to, the array of words would be copied into a temporary array for later use. The translation occurred by searching through these arrays for the input word. If a match was found, then the method would then return the value at the same index in the array stored prior. This methodology proved to be simple and efficient, and provided a concept for translation as a placeholder for when a more complex method is acquired. If the string was not found, then a message saying the word or phrase was not supported was returned instead.

Keyboard Input Detection

For the application to detect global keyboard input the open-source JNativeHook external library was utilized. Through the implementation of this library global keyboard presses could be detected by a Java application. Appendix J is the example code provided by the GitHub repository for JNativeHook, and it details the steps on how to utilize the JNativeHook global keyboard listener interface. It should be noted that the import structure used as defined in the appendix was different than what used to access the library. Using the "org.jnativehook" packages allowed the application to access the required class files from JNatieHook. The "com.github.kwhat.jnativehook" import described was not properly importing the external library's interface, so it was altered to the import structure described prior.

To access JNativeHook within the application's detection service class, a subclass within the file was created. This subclass was named GlobalKeyListener, and it implemented the native key listener interface described in the appendix. This interface was functionally identical to the key listener interface utilized for key bind creation, so understanding its functionality did not consume development time. One issue occurred during the integration of JNativeHook was which listener methodology to use for detecting input. It was noticed that the native key released method included within the interface would register ctrl values between key presses. Initially measures to prevent this occurrence, and to ensure that the data being interpreted by the application was as accurate to what the user was pressing as possible were employed. It was later discovered that the native key pressed method in the interface was preferable, since it did not add ctrl key presses in between what was being typed. Another configuration that needed to be changed while incorporating JNativeHook was the output to the console. Naturally JNativeHook returns all detected actions to the console for display. To discover how to remove this functionality the repository's documentation was consulted. As depicted in Appendix K, the logger class from the Java utility package is used to disable both the logging and the parent handlers from the library's global screen class. After this methodology was employed no console activity was displayed by the library's functions.

The detection service class of EZ-Translate included methods to interpret the global key presses that were being sent from JNativeHook. Before input is read, the application manager first checks if key binds are configured. If no key binds are configured prior to the starting of the application, then an error notification is displayed, and keyboard input monitoring does not begin. When the monitoring does begin however, it first receives the configured key bind strings and utilizes the string method replaceAll(). Within each string, each space is replaced with an empty string, thus removing all whitespace from each string. This is done for when comparing the strings to what is being processed by JNativeHook, the process needs not to consider white spaces from the stored key bind strings. After strings are cleared of white spaces, it is determined if the strings are the same. If they are the same, then a value that determines if the strings are the same is set to TRUE. Otherwise, this value remains FALSE. After this process, the Global Screen class within JNativeHook registers a native hook for input reading. The subclass that implemented the native key listener interface is initialized, then the subclass object is added to the global screen through the method to add native key listeners. Closing the input monitoring follows the reverse, where the subclass is removed as a native key listener, and the global screen unregisters the native hook.

The way that Ez-Translate interprets the input from JNativeHook is through the handle input method defined in Appendix L. This method is called within the native key listener method when a key is pressed, and it is called every time input is received by the keyboard. The key text of the native key event is used a parameter to the handling method. The method begins by concatenating the input native key event text to an internal string used to store the input being received. Then the method checks if input is contained within the open and close key strings. If it is not, then the input string is cleared. This is to ensure that the string does not become lengthy and that if a key is pressed that is not included within the defined key binds the input is cleared. If the string is contained, then there is a check to see if the key binds are the same. If the binds are the same, then the input string is checked to determine if it contains the open key bind, and that the menu is not open. If this is true, then the translation interface is displayed and the value that determines if the menu is open is set to true. The input string is also cleared in this statement. Otherwise, if the input string contains the close key bind string and the menu is open, then the translation interface is closed. The open value is also set to FALSE, and then the input string is cleared. If the strings are not the same, then the input string is checked to see if it contains the open key string or the close key string. Depending on if the open or close key bind is present in the input string the translation interface is either opened or closed respectively, and in both instances the input string is cleared.

Chapter 4

Results and Discussion

Results

Due to the nature of the project itself no tangible test data could be received from the application; but rather, the successfulness of certain tasks being performed was observed. For the displaying of the translation interface over a full screened application, tests proved that the textbox would not be displayed over another application if said application was full screened. Despite the use of a Window.toFront method for the frame, the translation interface would not display over the other application; however, the translation interface would be displayed if the other application were windowed. Despite this interaction with other applications, the window was successfully displayed or hidden if the corresponding key binds were pressed. Key bind configuration testing also passed as appropriate keys were accepted while key binds that were deemed invalid sent the appropriate error message and were not stored. The localization of the application also properly updated and changed based on user selection, and localization setting was consistent throughout application. The translation service of the application properly translates a recognized word or phrase and would also provide proper feedback if text input were unrecognized. File contents would also be properly updated and stored upon starting, closing, or changing an application's setting. All menus were also properly displayed or hidden when corresponding buttons were pressed.

An unintended result of utilizing JNativeHook is that while key binds could be created with UTF-8 supported characters, JNativeHook does not support extended character encoding. This resulted in a key bind that utilizes Cyrillic being unresponsive when displaying the translation interface. Furthermore, the configuration that is displayed within the configuration menu includes English characters for labelling. This is noticeable when the localization changes from English, and English words are still used for labelling the settings. The language translation provided is also limited to only 10 words from each language which is limited for a language translation service.

Discussion

The project's outcome met the goal of providing a method for accessing translation services while the user was within another application. This success was achieved through the application passing all current and valid test cases and meeting the expected translation standards. The results of the EZ-Translate application serve as a proof of concept for translating information from an application into a desired language and returning results for use back into said application. The overall results of the current application are satisfactory, as it upholds what was initially desired from the application. Given results shown through the functionality and performance of the application prove that the project was an overall successful endeavor. Unintended results with using JNativeHook does in fact reduce useability of this application for users who do not use a keyboard containing English characters. Additionally, the limited translation possibilities reduce the overall potential this application has in real world translation situations.

Chapter 5

Conclusion and Recommendations

Conclusion

In summary, the EZ-Translate application reached its goal of providing accessibility to translation services while within another application. While barriers were encountered during the project's development, measures were taken to overcome them while also meeting the initial project goal. These barriers were avoided, and a functional working example was provided within the development timeframe. While the project's working application is limited, it does provide a foundation for future development so that the systems that are present could be improved upon. The EZ-Translate application overall was a successful endeavor, and it achieved all planned project goals.

Recommendations

Recommendations for the project moving forward are to develop a native input detection service that is designed specifically for the EZ-Translate application. Building this feature for the application specifically rather than utilizing open-source code would allow for more characters to be supported for keyboard input monitoring services. JNativeHook is also infrequently maintained, so later-on in this project's life cycle other methods for monitoring global keyboard input should be utilized. The configuration storage should also become more intricate to allow for more potential configuration settings in the future. A more advanced language translation service should also be integrated into the project, to make the project more suitable for real world translation scenarios. The GUI of the application must also be redesigned to be more visually appealing for users. The ability to connect specific applications to EZ-Translate for individual input reading should also be re-added in the future to reduce the overall impact the global input reading system has on the system. Finally, more localizations should be added to ensure that the application is accessible to a wider variety of language speakers.

References

Baeldung. (2021). *Encode a String to UTF-8 in Java*. Retrieved April 16, 2021 from https://www.baeldung.com/java-string-encode-utf-8

Barker, A., et al, (2017). *JNativeHook*. (Version 2.1.0). GitHub. Retrieved April 22, 2021, from https://github.com/kwhat/jnativehook

Beattie, M. (2021). *EZ-Translate*. (Version 1.0). GitHub. Retrieved April 30, 2021, from https://github.com/masonBeattie99/EZ-Translate

Chuan, C. (2018). *Java Programming Tutorial Programming Graphical User Interface (GUI). Yet Another Insignificant Programming Notes*. Retrieved February 22, 2021, from https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

Galin, D. (2018). *Software quality: Concepts and practice*. ProQuest Ebook Central https://doi.org/10.1002/9781119134527

Hickey, S.[hickeys], et al. (2018). *Hooks Overview*. Microsoft. Retrieved February 22, 2021, from https://docs.microsoft.com/en-us/windows/win32/winmsg/about-hooks

Microsoft (n.d.). *Keyboard shortcuts in Windows*. Retrieved April 29, 2021 from https://support.microsoft.com/en-us/windows/keyboard-shortcuts-in-windows-dcc61a578ff0-cffe-9796-cb9706c75eec

Mkyong (2020-a). *How to read a UTF-8 file in Java*. Retrieved April 16, 2021 from https://mkyong.com/java/how-to-read-utf-8-encoded-data-from-a-file-java/

Mkyong (2020-b). *How to write a UTF-8 file in Java*. Retrieved April 16, 2021 from https://mkyong.com/java/how-to-write-utf-8-encoded-data-into-a-file-java/

Oracle. (n.d.-a). *Class: ResourceBundle*. Retrieved March 29, 2021, from https://docs.oracle.com/javase/7/docs/api/java/util/ResourceBundle.html

Oracle. (n.d.-b). *How to Make Dialogs*. Retrieved March 31, 2021, from https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

Oracle. (n.d.-c). *How to Use the Focus Subsystem*. Retrieved February 22, 2021, from https://docs.oracle.com/javase/tutorial/uiswing/misc/focus.html

Oracle. (n.d.-d). *How to Use Key Bindings*. Retrieved February 22, 2021, from https://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html

Oracle. (n.d.-e) *How to Write a Key Listener*. Retrieved April 12, 2021, from https://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html

Oracle. (n.d.-f). *How to Write Window Listeners*. Retrieved April 22, 2021, from https://docs.oracle.com/javase/tutorial/uiswing/events/windowlistener.html

Spell, B. (2015). *Pro Java 8 Programming*. Apress https://doi.org/10.1007/978-1-4842-0641-6

Sykes, J. M. (2018). *Digital games and language teaching and learning*. Foreign Language Annals, 51(1), 219–224. https://doi.org/10.1111/flan.12325

Stahl, P. (2020). *Lingua*. (Version 1.0.3). GitHub. Retrieved February 22, 2021, from https://github.com/pemistahl/lingua

Appendix A

(Baeldung, 2021, section 3)

```
String rawString = "Entwickeln Sie mit Vergnügen"; byte[] bytes =
rawString.getBytes(StandardCharsets.UTF_8); String utf8EncodedString
= new String(bytes, StandardCharsets.UTF_8); assertEquals(rawString,
utf8EncodedString);
```

Appendix B

(Mkyong, 2020-b, section 1)

```
try (FileOutputStream fos = new FileOutputStream(file);
          OutputStreamWriter osw = new OutputStreamWriter(fos, StandardCharsets.UTF_8);
BufferedWriter writer = new BufferedWriter(osw)
      ) {
          for (String line : lines) {
writer.append(line);
              writer.newLine();
          }
      } catch (IOException e) {
e.printStackTrace();
      }
```

Appendix C

(Mkyong, 2020-a, section 2)

```
public static void readUnicodeClassic(String fileName) {
File file = new File(fileName);
        try (FileInputStream fis = new FileInputStream(file);
             InputStreamReader isr = new InputStreamReader(fis, StandardCharsets.UTF_8);
BufferedReader reader = new BufferedReader(isr)
        ) {
            String str;
            while ((str = reader.readLine()) != null) {
                System.out.println(str);
            }
        } catch (IOException e) {
e.printStackTrace();
        }
    }
```

Appendix D

```
if(keyBind.length() != 0) {
                    if(keyBind.contains("Windows")) {
                        return false;
                    }
                    else if(keyBind.contains("End")) {
                        return false;
                    }
                    else if(keyBind.contains("Home")) {
                        return false;
                    }
                    else if(keyBind.contains("Backspace")) {
                        return false;
                    }
                    else if(keyBind.contains("Space")) {
                        return false;
                    }
                    else if(keyBind.contains("Insert")) {
                        return false;
                    }
                    else if(keyBind.contains("Escape")) {
                        return false;
                    }
                    else if(keyBind.contains("Tab")) {
                        return false;
                    }
                    else {
                        //do nothing and move on to specific combinations
                    }

                    //add further validation code here
                    switch(keyBind) {
                    case "Ctrl X ":
                        return false;
                    case "Ctrl C ":
                        return false;
                    case "Ctrl V ":
                        return false;
                    case "Ctrl Z ":
                        return false;
                    case "Ctrl F ":
                        return false;
                    case "Ctrl N ":
```

```
                return false;
        case "Ctrl Y":
                return false;
        case "Alt D ":
                return false;
        case "Ctrl W":
                return false;
        case "Alt Tab ":
                return false;
        case "Alt F4 ":
                return false;
        case "F1 ":
                return false;
        case "F2 ":
                return false;
        case "F3 ":
                return false;
        case "F4 ":
                return false;
        case "F5 ":
                return false;
        case "F6 ":
                return false;
        case "F7 ":
                return false;
        case "F8 ":
                return false;
        case "F9 ":
                return false;
        case "F10 ":
                return false;
        case "F11 ":
                return false;
        case "F12 ":
                return false;
        case "Ctrl Alt Delete ":
                return false;
        case "Ctrl Alt Tab":
                return false;
        }
        return true;
}
else {
        return false;
}
```

Appendix E

```
String[] engList = {
                    "Hello", "How are you", "Good morning", "Good day", "Good evening",
"Good night", "Goodbye", "What time is it", "Where are you from", "Thank you"};

String[] gerList = {
                    "Hallo", "Wie gehts", "Guten Morgen","Guten Tag", ”Guten Abend",
"Guten Nacht", "Auf Wiedersehen", "wie spät ist es", "Woher kommst du", "Vielen Dank"};

String[] rusList = {
                    "Привет", "Как дела", "Доброе утро", "Добрый день", "Добрый
вечер", "Спокойной ночи", "До свидания", "который сейчас час", "Откуда ты",
"Спасибо"};
```

Appendix F

(Chuan, 2018, section 8.4)

```
import java.awt.*;        // Using AWT layouts import java.awt.event.*; //
Using AWT event classes and listener interfaces import javax.swing.*;     //
Using Swing components and containers
// A Swing GUI application inherits from top-level container javax.swing.JFrame
public class SwingTemplate extends JFrame {
  // Private instance variables
  // ......
  // Constructor to setup the GUI components and event handlers
public SwingTemplate() {
    // Retrieve the top-level content-pane from JFrame
    Container cp = getContentPane();
// Content-pane sets layout
cp.setLayout(new ....Layout());       //
Allocate the GUI components
    // .....
    // Content-pane adds components
    cp.add( ... )
    // Source object adds listener
    // .....
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Exit the program when the close-window button clicked
    setTitle("......");      //    "super"    JFrame    sets    title
setSize(300, 150);     // "super" JFrame sets initial size
setVisible(true);    // "super" JFrame shows
```

```
  }
  // The entry main() method    public
static void main(String[] args) {
    // Run GUI codes in Event-Dispatching thread for thread-safety
    SwingUtilities.invokeLater(new Runnable() {
      @Override          public void run() {             new
SwingTemplate(); // Let the constructor do the job
      }
    });
  }
}
```

Appendix G

(Oracle, n.d.-b, Creating and Showing Simple Dialogs ex. 1)

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.");
```

(Oracle, n.d.-b, Creating and Showing Simple Dialogs ex. 3)

```
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);
```

Appendix H

```
openKeyAL = new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent evt) {
openKeyInputField.removeKeyListener(openKeyAda);
                currentOpenBind = "";
am.accessConfig().clearOpenKey();
openKeyInputField.setText(currentOpenBind);
openKeyInputField.grabFocus();
                openKeyInputField.addKeyListener(openKeyAda);
                                }
};//update open key
openKeyAda = new KeyListener() {
                @Override
                public void keyPressed(final KeyEvent e) {
                currentOpenBind += KeyEvent.getKeyText(e.getExtendedKeyCode()) + " ";
                openKeyInputField.setText(currentOpenBind);
                }
                @Override
                public void keyTyped(KeyEvent e) {
                }
                @Override
                public void keyReleased(KeyEvent e) {
                }
};
```

Appendix I

(Oracle, n.d.-a, Example)

```
 public class MyResources extends ResourceBundle {
public Object handleGetObject(String key) {          if
(key.equals("okKey")) return "Ok";          if
(key.equals("cancelKey")) return "Cancel";
return null;
    }
    public Enumeration<String> getKeys() {
return Collections.enumeration(keySet());
    }
    // Overrides handleKeySet() so that the getKeys() implementation
    // can rely on the keySet() value.
    protected Set<String> handleKeySet() {          return new
HashSet<String>(Arrays.asList("okKey", "cancelKey"));
    }
 }
 // German language public class MyResources_de
extends MyResources {      public Object
handleGetObject(String key) {          // don't need
okKey, since parent level handles it.          if
(key.equals("cancelKey")) return "Abbrechen";
        return null;

    }
    protected Set<String> handleKeySet() {
        return new HashSet<String>(Arrays.asList("cancelKey"));
}
```

Appendix J

(Barker et al, 2017, jnativehook/doc/Keyboard.md)

```
import com.github.kwhat.jnativehook.GlobalScreen; import
com.github.kwhat.jnativehook.NativeHookException; import
com.github.kwhat.jnativehook.keyboard.NativeKeyEvent; import
com.github.kwhat.jnativehook.keyboard.NativeKeyListener; public class
GlobalKeyListenerExample implements NativeKeyListener {
        public void nativeKeyPressed(NativeKeyEvent e) {
                System.out.println("Key Pressed: " +
NativeKeyEvent.getKeyText(e.getKeyCode()));
                if (e.getKeyCode() == NativeKeyEvent.VC_ESCAPE) {
        try {
                        GlobalScreen.unregisterNativeHook();
                        } catch (NativeHookException nativeHookException) {
                        nativeHookException.printStackTrace();
                        }
      }
        }

        public void nativeKeyReleased(NativeKeyEvent e) {
                System.out.println("Key Released: " +
NativeKeyEvent.getKeyText(e.getKeyCode()));
        }
        public void nativeKeyTyped(NativeKeyEvent e) {
                System.out.println("Key Typed: " + e.getKeyText(e.getKeyCode()));
        }
        public static void main(String[] args) {
                try {
                        GlobalScreen.registerNativeHook();
                }
                catch (NativeHookException ex) {
                        System.err.println("There was a problem registering the native hook.");
                        System.err.println(ex.getMessage());
                        System.exit(1);
                }
                GlobalScreen.addNativeKeyListener(new GlobalKeyListenerExample());
        }
}
```

Appendix K

(Barker et al, 2017, jnativehook/doc/ConsoleOutput.md)

```
// Get the logger for "com.github.kwhat.jnativehook" and set the level to off. Logger
logger = Logger.getLogger(GlobalScreen.class.getPackage().getName());
logger.setLevel(Level.OFF);

// Don't forget to disable the parent handlers.
logger.setUseParentHandlers(false);
```

Appendix L

```
public void handleInput(String input) {
        inputString += input;
        if(!openKeyString.contains(input) && !closeKeyString.contains(input)) {
            inputString = "";
        }
        else {
            if(same) {
                if(inputString.contains(openKeyString) && !open) {
                    openAction();
                open = true;
                    inputString = "";
                }
                else if(inputString.contains(closeKeyString) && open){
                    closeAction();
                open = false;
                    inputString = "";
                }
                else {
        //do nothing
                }
            }
            else {
                if(inputString.contains(openKeyString)){
        openAction();
                    inputString = "";
                }
                if(inputString.contains(closeKeyString)) {
        closeAction();
                    inputString = "";
                }
            }
        }
    }//handleInput
```