WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor's Thesis

Titel of Bachelor's Thesis (english)	ooRexx and JavaFX: A Perfect Match for GUI Development
Titel of Bachelor's Thesis (german)	ooRexx und JavaFX: Eine perfekte Kombination für die GUI Entwicklung
Author (last name, first name):	Dall'Oglio Isabella
Student ID number:	1607760
Degree program:	Bachelor of Science (WU), BSc (WU)
Examiner (degree, first name, last name):	ao.Univ.Prof. Dr. Rony G. Flatscher

I hereby declare that:

1. I have written this Bachelor's thesis myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.
2. This Bachelor's Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.
3. This Bachelor's Thesis is identical with the thesis assessed by the examiner.
4. (only applicable if the thesis was written by more than one author): this Bachelor's thesis was written together with

   The individual contributions of each writer as well as the co-written passages have been indicated.

03/03/2023
Date

Dall'Oglio
Unterschrift

ooRexx and JavaFX: A Perfect Match for GUI Development

Bachelor Thesis

eingereicht bei

ao.Univ.Prof. Dr. Rony G. Flatscher
Institut für Wirtschaftinformatik und Gesellschaft
Wirtschaftsuniversität Wien

Von

Isabella Dall`Oglio

Fachrichtung: Wirtschaftsinformatik
Matrikelnummer: 1607760

Abstract

This thesis explores the combination of the object-oriented scripting language ooRexx and the JavaFX framework to develop graphical user interfaces (GUIs). The thesis provides a comprehensive overview of the history and concepts of JavaFX, including its architecture, application structure and lifecycle, as well as key features such as Scene Builder, FXML and CSS. Additionally, the thesis covers the fundamental language concepts of ooRexx, such as syntax, variables, expressions and instructions and demonstrates how to interact with objects, create classes and utilize built-in classes. The thesis also discusses two JavaFX libraries, JFoenix and ControlsFX, which offer additional styling and widget options for developers. Practical examples and use cases are provided to illustrate GUI application development with JavaFX and ooRexx, including integrating external libraries, using FXML and CSS for GUI design and utilizing JDBC for database connectivity.

List of Figures

List of Tables

List of Snippets

List of Listings

Abbreviations

API	Application Programming Interface
AWT	Abstract Window Toolkit
BSF4ooRexx	Bean Scripting Framework for ooRexx
BSF	Bean Scripting Frameworks
CSS	Cascading Style Sheets
DOM	Document Object Model
FXML	JavaFX Scene Builder markup language
GUI	Graphical User Interface
HTML5	Hypertext Markup Language version 5
JavaFX	Java graphical user interface toolkit
JDBC	Java Database Connectivity
ooRexx	Object-Oriented Rexx language
OpenJFX	Open-source JavaFX
RexxLA	Rexx Language Association
RIA	Rich Internet Application
SVG	Scalable Vector Graphics

Introduction

In the recent years, JavaFX has gained popularity as a modern and versatile platform for developing graphical user interfaces (GUIs) for desktop, web, and mobile applications. At the same time, ooRexx has emerged as a powerful object-oriented scripting language that is easy to learn and use.

The combination of JavaFX and ooRexx provides a powerful and flexible toolset for developing GUI applications that can benefit from the rich library of JavaFX components, the simplicity and expressiveness of the ooRexx language, and the object-oriented paradigm that underlies both technologies.

The thesis explains the history and development of JavaFX, as well as its main features and architecture. It also introduces the basics of the ooRexx language, including its syntax, data types, and control structures, as well as its object-oriented model and built-in classes.

Additionally, practical examples and use cases are presented for developing GUI applications with JavaFX and ooRexx, including the integration of external libraries such as JFoenix and ControlsFX, and the use of FXML and CSS for GUI design and styling, as well as JDBC for database connectivity with JavaFX and ooRexx.

1. History of JavaFX

Sun Microsystems introduced the platform-independent programming language, Java, in 1995. Following its release, the Abstract Window Toolkit (AWT) library was introduced for the development of desktop applications and their graphical user interfaces (GUIs) (Abstract Window Toolkit, 2022). In 1996, the Swing package was introduced as a follow-up to the Abstract Window Toolkit (AWT) for graphical user interface (GUI) development in Java. AWT and Swing remained the standard for Java graphical applications until 2014. To overcome the limitations in media and animation, Oracle developed a new GUI layer and introduced JavaFX in 2008 (JavaFX, 2022).

In the following sections, the features of the two packages AWT and Swing as well as the new development JavaFX are discussed in more detail.

1.1. Abstract Window Toolkit

The Abstract Window Toolkit (AWT) was the first user-interface widget toolkit for the development of graphical user interfaces (GUIs) in Java when the programming language was first introduced (Abstract Window Toolkit, 2022).

The *java.awt* package, provides classes for developing GUI applications in Java, such as TextField, Label, TextArea, RadioButton, CheckBox and List, among others. To access these components, the *java.awt* package must be imported into the development environment using the following import statement: *"import java.awt.*;".*

The appearance of AWT components is dependent on the underlying operating system, so they are considered platform dependent. This means that the components will appear differently on different OS platforms, such as Windows and macOS. Additionally, because AWT components use the native appearance of the OS, they are considered "heavyweight", which means they are closely tied to the system environment (Java-Awt, 2022).

The use of native components in the AWT package has the benefit of improving performance. Additionally, AWT provides a strong event handling system and the ability to customize window layouts with its layout manager. A drawback of using AWT is the difficulty in creating a platform-independent application with a consistent appearance, as the available components are limited to those supported by all platforms (AWT vs. Swing, 2022).

The Java Foundation Classes (JFC) include the Abstract Window Toolkit (AWT) as a standard Application Programming Interface (API) for creating graphical user interfaces in Java (Abstract Window Toolkit, 2022).

1.2. Swing

In 1996, Java Swing was introduced as a GUI widget toolkit, offering a departure from the previous AWT framework. Swing is platform-independent and lightweight, as it is written entirely in Java and draws its own components. This independence allows developers to choose between the look and feel of the underlying system and the uniform look and feel of Java. Although Swing has largely replaced AWT, it still builds on and complements the latter (AWT, 2022).

Swing builds upon the features of AWT and offers a wider range of components, including trees, image buttons, tables, tabbed panes, sliders and more. Unlike AWT, Swing is not dependent on peer components, making it a versatile option for GUI programming in Java (Java Swing, 2022). To utilize the Swing components, developers must import the *javax.swing* package into their development environment. This can be achieved by adding the following import statement: *"import javax.swing.*;".* The Java Swing toolkit is part of the Java Foundation Classes (JFC) (Java Swing, 2022).

1.3. JavaFX

JavaFX is a cutting-edge development in the GUI layer that replaced the aging AWT and Swing. The limitations in media and animation capabilities made it necessary for a new GUI layer to be created. Oracle decided to create JavaFX to meet the modern requirements for graphical interfaces.
The first version of JavaFX was released by Sun Microsystems in 2008. It started as F3, a Java scripting language for GUI development, created by Chris Oliver at SeeBeyond. After being acquired by Sun Microsystems in 2007, F3 was renamed JavaFX. In 2010, Oracle acquired Sun Microsystems and made JavaFX open source in 2013 (What is JavaFX, 2022).
JavaFX is an open-source framework for developing cross-platform Java applications. Its goal is to make it easier to create and distribute interactive multimedia content and GUIs. Desktop applications and Rich Internet Applications (RIAs) can be developed using the JavaFX library and run-on multiple platforms, including web, mobile and desktops. JavaFX provides its own components and is lightweight as it is not dependent on the platform. It supports various operating systems such as Windows, Linux and Mac OS (JavaFX, 2022).

2. JavaFX Concepts

This chapter focuses on the theory behind JavaFX development, including its architecture, application structure, and lifecycle. It also discusses the tools like Scene Builder, FXML and Cascading Style Sheets (CSS).

2.1. Architecture

The design of JavaFX does not rely on the architecture of AWT and Swing. The structure and components of the JavaFX platform are shown and explained in detail in Figure 1.

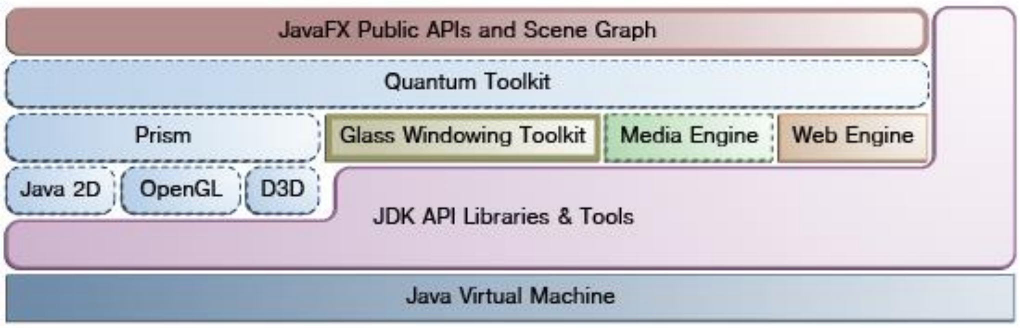


Figure 1 JavaFX Architecture Diagram (JavaFX_Oracle, 2023)

The foundation of the platform is the Java Virtual Machine (JVM), which is part of the Java runtime environment and executes Java bytecode in its own virtual machine. On top of the JVM, the Java Development Kit provides developer tools and extensions, such as Java 2D for creating 2D shapes.

Prism, the rendering engine, works with both hardware and software. The hardware render path is preferred for better performance and requires either DirectX 9 on Windows XP and Vista, DirectX 11 on Windows 7, or OpenGL on Mac, Linux and Embedded.
If hardware rendering is not possible, the software rendering path on Java 2D is used, which is already included in all Java Runtime Environments.

The Glass windowing toolkit sits at the lowest level of the JavaFX graphics stack and provides access to low-level operating system routines, such as managing windows, timers and surfaces. The Glass toolkit also manages the event queue and uses the native operating system's functionality for scheduling thread management.
The Media Engine integrates audio and video. The Web Engine, based on WebKit, supports HTML5, CSS, JavaScript, DOM and SVG. This allows Java applications to: render HTML content from local or remote URLs, support history and provide back and forward navigation, reload content, apply effects to web components, edit HTML content, execute JavaScript commands and handle events.

The Quantum Toolkit combines Prism, Glass Windowing Toolkit, Media Engine and Web Engine and exposes them to the JavaFX API (JavaFX_Oracle, 2022).

The highest level of the architecture provides a complete set of public Java APIs, with the main packages listed in Table 1:

Package	Description
javafx.animation	Provides the set of classes for easy use of transition-based animations.
javafx.application	Provides the application life-cycle classes.
javafx.collections	Contains the essential JavaFX collections and collection utilities.
javafx.event	Provides basic framework for FX events, their delivery and handling.
javafx.fxml	Defines the FXML APIs for the JavaFX UI toolkit.
javafx.geometry	Provides the set of 2D classes for defining and performing operations on objects related to two-dimensional geometry.
javafx.scene	Provides the core set of base classes for the JavaFX Scene Graph API.
javafx.stage	Provides the top-level container classes for JavaFX content.
javafx.util	Contains various utilities and helper classes.

Table 1: Main packages of the JavaFX API (JavaFX, 2022)).

2.2. JavaFX Application Structure

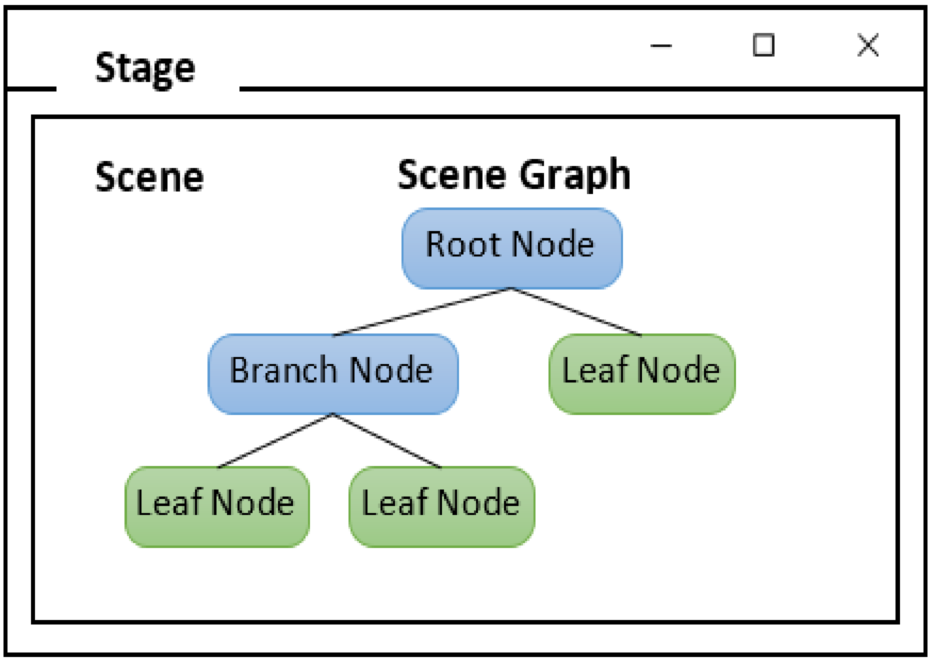


Figure 2: Application Structure (ApplicationStructure, 2023)

The design of graphical applications in JavaFX is based on the concept of a theater. The Stage, defined by the *javafx.stage.Stage* class, is the top-level container for a GUI and can be thought of as a window. The platform creates the primary Stage, while additional Stages can be created by the application. Stage properties are largely read-only, as they can be changed by the underlying platform and are therefore not bound (JavaFX_Stage, 2022).

The Stage is split into the decoration (title bar and frame) and the content area and its size is determined by its width and height parameters. There are five different types of Stages: Decorated, Undecorated, Transparent, Unified and Utility. To display the Stage, the created Stage object is passed as an argument to the *start()* method of the application class (JavaFX_Application, 2022).

A scene, defined by the *javafx.scene.Scene* class, is necessary to visualize the content on the Stage. A Scene contains all the elements of a GUI and can only be assigned to one Stage at a time, although an application can have multiple scenes. The Scene object is created by creating an object of the Scene class and passing it to the constructor of the Stage (JavaFX_Application, 2022).

At the lowest level of the hierarchy is the Scene Graph, a tree-like data structure that manages the individual components of a GUI. The elements of the graph are represented as node objects, defined in the abstract class *javafx.scene.Node* and can include geometrical objects, UI controls, containers and media elements such as audio, video and images.

There are three types of nodes in the Scene Graph: Root Node, Branch Node and Leaf Node. Each node in the graph has a parent and zero or more child nodes, except for the root node. The Branch Node, defined by the abstract class *javafx.scene.Parent*, contains three subclasses: Group, Region and WebView. The properties of a parent node are applied to child nodes when transformations are performed (JavaFX_Application, 2022).

2.3. Lifecycle of JavaFX Application

The *javafx.application.Application* class is a required import for all JavaFX applications. This class has three life cycle methods: *init(), start()* and *stop(),* which can be customized by the application if needed. The *launch()* method, provided by JavaFX, is used to launch the application and eliminates the need for a main method.
Following how the lifecycle methods are executed when a JavaFX application is launched:

- An instance of the Application class is created.
- The *init()* method of the instance is executed, which is empty by default.
- The *start()* method is executed and is passed the stage. This method is abstract and must be overridden.
- The JavaFX runtime waits until the application is terminated, either through calling the *Platform.exit()* method or closing the last window when the *implicitExit* attribute of Platform is set to true.
- Finally, the *stop()* method is executed, which is also empty by default.

Note that the *init()* method cannot create a Stage or Scene and the *stop()* method only needs to be customized if necessary (JavaFX, 2022).

2.4. Scene Builder

The Scene Builder is a visual layout tool for designing JavaFX application interfaces without programming knowledge. It allows users to drag and drop UI components into the workspace and customize their properties using a stylesheet. The scene graph structure is generated automatically in the background and saved in an FXML file. The FXML file can then be connected to the Java project by linking the elements to the applications logic (SceneBuilder, 2022)
Originally distributed by Oracle until Java 8, Scene Builder is now maintained and updated by Gluon within the OpenJFX project.

2.5. FXML

FXML files make it simpler for developers to maintain and modify code by separating the presentation layer from the application logic. To use FXML files in a JavaFX application, developers can load the FXML file through the FXMLLoader class and then pass it to the Scene object to display the user interface in the application window. Additionally, the FXMLLoader class can be used to load a controller object and set it as the controller for the user interface described in the FXML file. The JavaFX runtime processes the FXML files to create the user interface (Learn JavaFX 17, 2023).

2.6. Cascading Style Sheet (CSS)

Cascading Style Sheets (CSS) is a language used for formatting and designing HTML, SVG and XML documents. It is constantly being improved by the World Wide Web Consortium (W3C) and is a key language in the World Wide Web. CSS allows the separation of the content and design of an electronic document by allowing layout, colors and typography to be defined in separate CSS files through the use of stylesheets.
This gives CSS an unlimited level of flexibility (CSS, 2022). JavaFX also supports CSS. JavaFX provides the *javafx.css* package, which contains all the CSS classes for use in JavaFX applications. CSS can be used to customize and design JavaFX controls and scene graph objects. Any compatible CSS parser can easily parse JavaFX CSS stylesheets. (JavaFX_CSS, 2023)

3. The Language Rexx and Open Object Rexx (ooRexx)

REXX is a procedural programming language that enables the structured and organized coding of algorithms and programs. Its main goal was to be user-friendly for both computer experts and non-technical individuals. REXX simplifies the manipulation of common symbolic objects such as words, numbers, names and more. Its features are designed to make symbolic manipulation easier. REXX is designed to be system-independent, although it has the capability to send commands to its host environment and call programs or functions written in other languages. It offers powerful character and arithmetic capabilities in a straightforward framework, making it suitable for both simple and complex programs (Cowlishaw, 1990). Rexx is a user-friendly language with a simplified structure, built-in functions and classes, flexible variable types that can handle any object, strong string manipulation capabilities, decimal arithmetic instead of binary arithmetic, easily understandable error messages and robust debugging tools (ooRexx, 2023).

ooRexx, also known as Open Object Restructured Extended Executor, is a compatible version of Rexx that has been enhanced with object-oriented features. The core elements of Rexx remain unchanged, but ooRexx includes added capabilities that allows typical object-oriented language capabilities, including classes, objects, methods, inheritance, multiple inheritance and messaging. In traditional Rexx, all data were stored as strings, but with ooRexx, variables can now refer to objects other than just strings. The language has a variety of built-in classes, including those for arrays, queues, streams and the String class. Additionally, developers can create their own custom classes that work in conjunction with the built-in classes. Methods are used to manipulate objects in these classes and are accessed by sending a message to the object. The use of object technology has several benefits, including simplified design through object modeling, greater code reuse, rapid prototyping, higher-quality components, easier maintenance, cost savings, increased adaptability and scalability (Ashley W. , et al., 2010)

3.1. History

Mike Cowlishaw developed the Rexx Restructured Extended Executor language in 1979 to replace exec and exec-2. The language was first introduced to the public at the 56th SHARE conference in Houston. Over the years, IBM has integrated Rexx into nearly every operating system.
In 1990, Cathie Dager from SLAC organized the first independent Rexx symposium, which led to the formation of the REXX Language Association. Annual symposiums take place (Rexx, 2023).

An object-oriented version of REXX was developed due to the influence of object-oriented programming. Many concepts from the object-oriented, message-based programming language Smalltalk were incorporated. An object-oriented version of REXX was released in the late 1990s. The Rexx Language Association (RexxLA) acquired Object Rexx from IBM and released "Object Rexx (ooRexx) 3.0" as open source in 2004.

In 2009, ooRexx 4.0 was released with a new kernel and a new native interface that allows the C++ programming language to use ooRexx as a scripting language.

In 2010, BSF4ooRexx ("Bean Scripting Framework for ooRexx") was released, which acts as a bidirectional bridge between ooRexx and Java (Flatscher R. G., 2019).

3.2. Fundamental Language Concepts

The main ideas that were consciously used when designing Rexx are listed in the list below.

- **Readability:** Rexx semantics is comparable to regular text semantics. The syntax structure should be simple to read. Upper- and lowercase letters are explicitly supported throughout the language, both for data processing and for the program itself. The Rexx language is written in a free format. This means extra spaces between words and blank lines can be inserted freely throughout the exec without causing an error. Punctuation is only used when it is necessary to remove ambiguity.

- **Natural data typing:** Rexx is not strongly typed, in contrast to many other languages. Types are handled as naturally as possible by Rexx. The meaning of data is entirely dependent on how it is used. All values are specified as strings of characters, or the symbolic notation, that a user would typically use to represent the data. Because the outcomes of all operations have a defined symbolic representation, values can always be inspected. Because numerical computations and all other operations are precisely defined, they will behave consistently and predictably for every correct implementation.

- **Emphasis on symbolic:** Rexx operates with character strings and has a rich set of operators and functions for manipulating them. One of its unique features is the "blank" operator, which concatenates two strings with a blank space in between, along with the conventional concatenation operator "II" that combines two strings without a space.

- **Dynamic scoping:** The scoping of Rexx is entirely dynamic. This implies that it can be interpreted effectively. Rexx scoping follows the programmer-defined order in which Rexx clauses are executed.
- **Nothing to declare:** In Rexx, the declaration of variables is not required. Instead, variables can be created and given a value at the start of a program.

- **System independence:** System and hardware are unrelated to the REXX language. REXX programs need to be able to interact with their environment.

- **Limited span syntactic units:** The clause, which is a piece of program text terminated by a semicolon, is the REXX language's syntactic unit. As a result, syntactic units have a short span, usually one line or less. This means that the syntax parser in the language processor can detect and locate errors quickly, allowing error messages to be precise and concise.

- **Dealing with reality:** Consistency was an important design goal; in practice it leads to unexpected side effects.

- **Be adaptable:** The language allows for the extension of instructions and other language constructs whenever it is possible. Since only a small set of common characters are permitted for variable names (symbols), there is a useful set of common characters available for future extensions. Similar to this, the rules for keyword recognition permit the addition of instructions whenever necessary without jeopardizing the integrity of already-written programs. There are no words that are reserved globally. The language is made more adaptable by including space for growth and modification.

- **Keep the language small:** Every suggested extension to the language has only been taken into consideration if it would benefit a sizable portion of users. Users quickly understand the majority of the language, it has been intentionally kept as small as possible.

- **No defined size or shape limits:** There are no restrictions on the size or shape of any of the language's tokens or data (Cowlishaw, 1990)

3.3. Language Basics

3.3.1. Structure and General Syntax

A Rexx program is made up of clauses that includes the following elements:

- Optional whitespace characters (blanks or horizontal tabs), which are ignored by the processor
- A sequence of tokens
- Optional additional whitespace characters, which are again ignored
- A semicolon (;) delimiter, which may be implied by line end, certain keywords, or the colon (:) symbol.

Before execution, each clause is scanned from left to right and the tokens that make up the clause are identified. During this process, instruction keywords are recognized, comments are removed and whitespace character sequences (except within literal strings) are condensed into single blanks. In addition, any whitespace or special characters adjacent to operator characters are also removed (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.2. Characters

When programming in the REXX language, two sets of characters should be considered. The first set is a relatively small set of characters that are used to write the REXX program itself. This set is explicitly defined by the REXX language to ensure code portability and readability, while avoiding limitations on the character set used for data.

The second set of characters is used as data in a REXX language processor and can generally be any characters. Some characters may only be used within comments or literal data.

When the REXX language manipulates or examines data, such as when performing arithmetic operations, there may be specific requirements for the data character set. For example, numbers must be represented by digits in the set (Cowlishaw, 1990).

3.3.3. Comments

In Rexx, comments are sequences of characters that are ignored by the program but serve as separators. There are two types of comments recognized by the interpreter: line comments and standard comments.

A line comment starts with two consecutive minus signs (--) and ends at the end of the line.

A standard comment is a sequence of characters, on one or more lines, that are surrounded by the delimiters /* and */. Any characters can be used within the delimiters, including nested standard comments as long as each begins and ends with the delimiters. Standard comments can appear anywhere and be any length (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.4. Variables

Variable names can be up to 250 characters in length and have certain naming restrictions. The first character must be an uppercase or lowercase letter, an exclamation mark (!), a question mark (?), or an underscore (_). The rest of the characters can be letters, numbers, exclamation marks, question marks, underscores, or periods (.).

Variable names are case-insensitive, meaning they can be typed and queried in uppercase, lowercase, or mixed-case characters. Rexx automatically converts all lowercase letters in variables to uppercase before use, so "abc", "Abc" and "ABC" all refer to the same variable, "ABC". If a variable is referenced before it has been set, the name in uppercase characters will be returned. All data is treated as objects of different types. Variables can contain any type of object, so there is no need to specify the type of the variable beforehand, such as declaring it as a string or number. Variables can be assigned new values using the ARG, PARSE, or PULL instructions (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

3.3.5. Tokens

Tokens in Rexx are the smallest building blocks of syntax that make up a program. The maximum length of a token may vary based on the implementation, but they can be of any length. Tokens are differentiated from each other by whitespace, comments, or their own nature and are used to construct clauses. There are different classes of tokens in Rexx (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.5.1. Literal String

A literal string in Rexx is a sequence of characters enclosed by either single quotes (') or double quotes ("). To include the same type of quote within the string, two consecutive quotes need to be used. A null string is a literal string with no characters. Literal strings are considered constant and their contents won't change during processing. They must be complete on a single line and their length is limited only by available memory. Additionally, a string followed by a left parenthesis is treated as a function name and if immediately followed by the letter "X" or "x", it's considered a hexadecimal string, or if followed by "B" or "b", it's considered a binary string (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.5.2. Hexadecimal Strings

A hexadecimal string is a type of literal string that represents its encoding using hexadecimal notation. It consists of zero or more hexadecimal digits (0-9, a-f, A-F), grouped in pairs, separated by one or more whitespace characters and enclosed in single or double quotation marks. The symbol x or X must immediately follow the closing quotation mark. The whitespace characters are ignored by the language processor for improved readability. Hexadecimal strings allow the inclusion of characters in a program, even if they cannot be directly entered. When a hexadecimal string is processed, the whitespace is removed and each pair of hexadecimal digits is converted to its equivalent character. The packed length of a hexadecimal string is unlimited (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.5.3. Binary Strings

A binary string is a type of literal string that represents its encoding using binary digits (0 or 1). The binary digits are grouped in bytes (8 digits) or nibbles (4 digits) and can be separated by whitespace characters for readability. The string must be delimited by matching single or double quotation marks and immediately followed by the symbol b or B. The packed length of the binary string (with whitespace removed) is not limited. The leading 0 digits are added to make a multiple of 8 or 4 before packing. Binary strings allow the explicit specification of characters using binary digits (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.5.4. Symbols

A Symbol in ooRexx is a combination of characters that can consist of English letters (both uppercase and lowercase), numbers (0-9) and special characters (".", "!", "?" and "_"). Lowercase letters are automatically converted to uppercase before they are used.
Symbols that do not start with a digit or a period can be used as variables and assigned a value. The value of such symbols is the uppercase version of the symbol's characters. Conversely, symbols that begin with a digit or a period are constant and cannot have a value assigned to them.

ooRexx also supports exponential number representation, with symbols starting with a digit or a period, ending with "E" or "e" and having an optional sign (+ or -) followed by one or more digits. There must be at least one digit and at most one period in the character sequence before the "E" or "e". The sign is considered part of the symbol and is not an operator.

The interpretation of a symbol depends on its usage within a particular context. It can represent a constant value, such as a number, a reserved word, or the identifier of a variable (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.5.5. Environment Symbol

An Environment Symbol in ooRexx is a symbol that begins with a dot (.) followed by a symbol name. The Rexx interpreter converts all alphabetical characters in the symbol name to uppercase. The symbol represents a value in one of the ooRexx runtime environments, which are searched in the following order: "package environment", the local environment and the global environment. If a corresponding value is found, it replaces the environment symbol. If no value is found, the symbol name is converted to uppercase and replaces the symbol.

Examples of ooRexx environment symbols include ".TRUE", ".FALSE", ".NIL", ".LOCAL", ".ENVIRONMENT" and ".SOME.VALUE". Each of these symbols represents a specific value, such as a string or directory, within the ooRexx runtime environments (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

3.3.6. Expressions

Expressions allows the combination and transformation of data, resulting in a final result. The output of an expression is always in the form of an object and can be a modification of the original data used in the expression. Expressions are a fundamental aspect of the Rexx language and are used to perform various operations and calculations.

3.3.6.1. String Concatenation Expressions

The concatenation operators join two strings to create a new string by attaching the second string to the right side of the first string. There are three concatenation operators: (blank), which concatenates terms with one blank in between, ||, which concatenates terms without a blank in between and (abuttal), which is assumed between two terms that are not separated by another operator.

For example, the expression "Hello" || " World" would result in the string "Hello World". On the other hand, the expression "Hello" " World" would result in the same string, with a blank in between the two terms (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.6.2. Arithmetic Expressions

In ooRexx, arithmetic operations can be performed on character strings that are considered valid numbers. The following operators are available:

- + for addition
- - for subtraction
- * for multiplication
- / for division
- % for integer division
- // for remainder
- ** for power
- prefix - and prefix + operators to indicate subtraction and addition respectively.

When performing arithmetic operations, it is important to note that the result may be displayed in exponential notation if rounding has occurred (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.6.3. Comparison

Comparison operators compare two values and return a result of 1 if the comparison is true, or 0 if it is false. The following operators are available:

- = (equal to) - checks if two values are equal
- \ = or ¬= (not equal to) - checks if two values are not equal

- > (greater than) - checks if the left operand is greater than the right operand
- < (less than) - checks if the left operand is less than the right operand
- >< (not equal to, also known as xor) - checks if either the left or right operand is true, but not both
- <> (not equal to) - same as = or ¬=, checks if two values are not equal
- >= (greater than or equal to) - checks if the left operand is greater than or equal to the right operand
- \ < or ¬< (not less than) - checks if the left operand is not less than the right operand
- <= (less than or equal to) - checks if the left operand is less than or equal to the right operand
- \ > or ¬> (not greater than) - checks if the left operand is not greater than the right operand
- == (same as) - checks if two values are the same (identical)
- \ == or ¬== (not same as) - checks if two values are not the same (not identical)
- >> - strictly greater than
- << - strictly less than
- >>= - strictly greater than or equal to
- \ << or ¬<< - strictly not less than
- <<= - strictly not greater than
- \ >> or ¬>> - strictly not greater than

Strict comparison operators (such as == and \ ==) require an exact match between the two strings being compared, including matching length and character-by-character comparison. When comparing two numeric values, a numeric comparison is executed. Otherwise, they are treated as character strings and any leading or trailing white space is ignored and the shorter string is padded with spaces on the right. (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.6.4. Logical

A character string is considered false if it is 0 and true if it is 1. The logical operators work with one or two such values, returning either 0 or 1. Only 0 or 1 are allowed as values (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

& - AND operator returns 1 if both values are true.

| - Inclusive OR operator returns 1 if either value or both values are true.

&& - Exclusive OR operator returns 1 if either value, but not both, is true.

\ , ¬ - Logical NOT operator negates the value, meaning 1 becomes 0 and 0 becomes 1

3.3.7. Instructions

The ooRexx programming language provides a comprehensive set of instructions that enable the interpreter to perform a wide range of actions. These instructions serve as the basic elements of an ooRexx program and include control flow instructions for decision making and repetition, data manipulation instructions for working with variables, as well as a multitude of other operations. Effective use of these instructions is essential for creating robust and efficient programs in ooRexx.

3.3.7.1. Message Instructions

In Open Object Rexx, a message instruction is used to invoke a method on an object, referred to as the "receiver object." The receiver object can be represented by various elements, such as a symbol, an environment symbol, a string, a literal string, a function call, or an expression in parentheses. The message instruction consists of the receiver object, followed by either the single tilde message operator or the double tilde cascading message operator and the name of the method. The method name can be followed by a colon and a symbol or environment symbol that refers to a superclass where the object should search for the method.

Additionally, there can be round parentheses after the method name, which may contain arguments to be supplied to the method (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

If the invoked method returns a value, the message instruction is replaced by the return value, which can serve as the receiver object for another message instruction. Message instructions are processed from left to right. A message term is used when the main purpose of the message is to obtain a result. If there is only a message term, it is sent in the same way as a message instruction. If the message results in a result object, it is assigned to the sender's special variable, RESULT. If the double tilde cascading message operator is used, the receiver object is used as the result. If there is no result object, the RESULT variable becomes uninitialized (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022).

3.3.7.2. Control Structures

The following constructs in ooRexx provide the capability to control the flow of execution in a program (Fosdick, 2005):

1. **If-Then-Else** - This construct allows a programmer to make decisions based on a specified condition. The syntax of this construct is as follows:

```
if condition then
  statements
else
  statements
end
```

Listing 1: If-Then-Else

2. **Select-When** - This construct allows a programmer to make multiple decisions based on a specified value. The syntax of this construct is as follows:

```
select
  when value1 = condition1 then statements
  when value2 = condition2 then statements
  otherwise statements
end
```

Listing 2: Select-When

3. **Do-End** - This construct allows a programmer to create a loop, repeating a set of instructions a specified number of times. The syntax of this construct is as follows:

```
do count = 1 to limit
  statements
end
```

Listing 3: Do-End

4. **Do-Until** - This construct allows a programmer to create a loop that repeats until a specified condition is met. The syntax of this construct is as follows:

```
do until condition
  statements
end
```

Listing 4: Do-Until

5. **Do-While** - This construct allows a programmer to create a loop that repeats while a specified condition is true. The syntax of this construct is as follows:

```
do while condition
  statements
end
```

Listing 5:Do-While

6. **Loop** - This construct allows a programmer to create a loop that repeats an indefinite number of times. The syntax of this construct is as follows:

```
loop
  statements
end
```

Listing 6: Loop

7. **Iterate** - This construct allows a programmer to exit a loop and start the next iteration. The syntax of this construct is as follows: iterate
8. **Leave** - This construct allows a programmer to exit a loop. The syntax of this construct is as follows: leave.

3.3.7.3. Data Manipulation Keyword Instructions

The data manipulation keywords in ooRexx are a set of commands used for processing and manipulating data in a program. These keywords are essential for organizing and transforming data within a program. The following are the commonly used in ooRexx (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022):

1. **SAY** - The Say keyword serves the purpose of outputting a message to the user for the purpose of displaying information regarding the state of the program or to prompt the user for input.
2. **CALL** - The Call keyword is utilized to invoke a Rexx procedure, thereby enabling the reuse of code within the program.
3. **PARSE** - The Parse keyword is employed to extract data from a string, making it useful for parsing and manipulating strings in the program

3.3.7.4. Program Management – Keyword Instructions

ooRexx provides several keywords for program management, which are used to manage and organize ooRexx programs. The following are the commonly used program management keywords in ooRexx (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022):

1. **RETURN** – The Return keyword is used to return a value from a Rexx procedure. This allows to pass data from one procedure to another, or to return a result from a function.
2. **EXIT** – The Exit keyword terminates the execution of a Rexx program. This is useful for prematurely ending a program if a certain condition is met or for explicitly ending the program when desired.
3. **EXPOSE** – The Expose keyword allows access to object variables in the current object’s variable pool in a method. Changes to these variables persist and are immediately visible to other methods sharing the same scope. All other variables are local and dropped upon exit from the method. EXPOSE must be the first instruction in the method.
4. **ADDRESS** – The Address keyword is used to interact with external systems and applications. Using the Address keyword, external commands can be run, scripts can be executed, or interaction with other programs can be performed.
5. **PROCEDURE** – The Procedure keyword is used to define a Rexx procedure. Procedures enable the encapsulation of code, resulting in reusability and organization. They can also be passed parameters, making them more flexible and adaptable.

3.3.7.5. Error Handling – Keyword Instructions

Error handling is a crucial aspect of programming as it enables anticipation and handling of errors and exceptions that may arise during the execution of a program. The following are some of the commonly used ooRexx keywords for error handling (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Reference, 2022):

1. **SIGNAL** – Raises an error or exception in the program to signal a specific condition.
2. **ON** – Enables error handling in the program and specifies the conditions that trigger error handling and the actions to be taken when an error occurs.
3. **OFF** – Disables error handling in the program, which can be useful for temporarily disabling error handling for a specific section of the program.
4. **ERROR** – Indicates that an error has occurred in the program and can be used in combination with the On keyword to specify the actions to be taken when an error occurs.
5. **TRAPS** – Specifies the conditions that trigger error handling in the program, such as specific error codes or exceptions.

3.3.8. Directives

Directives are instructions that provide structure and organization to a Rexx program. They are indicated by two consecutive colons (:: and serve as separators between different sections of code. When a program is executed, the directives are processed first to establish any necessary classes, methods, or routines before the main code block is executed (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

3.3.8.1. The ::ROUTINE

The ::ROUTINE directive in ooRexx is used to create named routines within a program. The directive starts at the beginning of the routine and ends with another directive or the end of the program. The ::ROUTINE directive helps to organize functions that are not related to a specific class type. Additionally, it has a PUBLIC option, which makes the routine accessible to other programs outside of the containing Rexx program. To use the routine, the external program must reference it using a ::REQUIRES directive in the Program that contains the routine (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

The following code snippet in Listing 7 defines a public routine named hello that will output the string “Hello, world!” to the console when called. The double colons before “ROUTINE” indicate that it is a method or function that can be called from outside the script.

```
::ROUTINE hello PUBLIC
    say "Hello, world!"
```

Listing 7: Routine PUBLIC

The following code snippet in Listing 8 shows an routine named "hello", which outputs the string "Hello, world!" to the console when called. This routine is not explicitly defined as public, so it will only be accessible from within the script where it is defined.

```
::ROUTINE hello
    say "Hello, world!"
```

Listing 8: Routine

3.3.8.2. The ::REQUIRES

The ::REQUIRES directive in ooRexx allows a program to access the classes and objects of another program. The directive is written in the following form: ::REQUIRES program_name. The ::REQUIRES directives are processed prior to any other directives and the order of these directives determines the search order for the classes and routines defined in the referenced programs. It is important to note that local routine or class definitions within a program take precedence over any imported routines or classes through ::REQUIRES directives (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

3.4. Introduction to ooRexx

Open Object Rexx, an extension of the traditional Rexx language, includes key characteristics of object-oriented programming such as encapsulation, inheritance and polymorphism. This expansion does not eliminate the use of traditional Rexx functions and programs can still be developed and executed as before. It offers the flexibility to program with just objects, just traditional Rexx instructions, or a mix of both (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

3.4.1. Interacting with Objects

In Object-Oriented Rexx (ooRexx), interaction with objects such as values and instances can only be performed using message instructions. Objects are considered as entities that receive a message name and search for a corresponding method. If the method is found, it is executed by passing any arguments received with the message and returning any value produced by the method. If the method cannot be found, an error occurs.

3.4.2. Classes

Data types are represented as classes, which define the allowable values (known as attributes) and the operations that can be performed on those values (referred to as methods). The internal workings of the class or the values themselves are kept hidden from the user through the use of the black-box approach. Each class in ooRexx is represented by a class object that holds the attributes and methods of the class. At the start of the program, ooRexx creates class objects for all built-in classes and stores them in the global environment directory to allow for easy access. The attributes of a class object are known as class attributes, while the methods are referred as class methods.

When an object is created from a class, initialization operations can be performed by using a constructor method known as INIT. The INIT method is invoked when the NEW class method is called to create the object and any arguments provided to the NEW method will be passed to the INIT method in the same order. When an object is no longer being used and there are no references to it, it becomes garbage and the ooRexx garbage collector destroys it to release computer resources such as memory. Before destroying the unused object, the garbage collector will call the destructor method UNINIT, if it exists (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

The following code snippet in Listing 9 shows an example ooRexx class named "ExampleClass" with two attributes, variable1 and variable2. This class has a constructor method named "init" and a destructor method named "uninit". It also has a method named "output" which outputs the values of the instance variables.

```
::CLASS ExampleClass

-- Define attributes for the class
::attribute variable1
::attribute variable2

-- The constructor is called when an instance of the class is created
::METHOD init
  expose variable1 variable2
  use arg
  say "An instance of ExampleClass has been created"
  variable1 = "Hello"
  variable2 = "World"

-- The destructor is called when an instance of the class is destroyed
::METHOD uninit
  expose variable1 variable2
  say "An instance of ExampleClass has been destroyed"

-- A method of the class that outputs the variables
::METHOD output
  expose variable1 variable2
  say "variable1:" variable1
  say "variable2:" variable2
```

Listing 9: Example Class

The expose keyword is used to make the instance variables visible to the methods of the class. The use arg statement is used to pass arguments to the constructor method.

3.4.2.1. Organization of Classes

The organization of classes into a class hierarchy serves to simplify the process of creating new classes and searching for methods. Classes in this hierarchy are related to one another based on their position, with one class being either above or below another class. The highest class, which does not have a superclass, is referred to as the Root Class and is commonly named Object. The Class Hierarchy is utilized to search for methods, allowing objects to inherit methods and attributes from superclasses along the hierarchy. As a result, the Root Class, Object, is eventually consulted, making all its methods accessible to all objects in ooRexx (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

3.4.2.2. Inheritance and Polymorphism

In ooRexx, multiple inheritance is supported. This means that a class can have more than one direct superclass. This feature allows for inheriting method implementations from multiple superclasses directly (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

Polymorphism in ooRexx allows objects to respond to the same message in different ways. This is achieved by having each object have its own implementation of a method, such as the REVERSE method. This enables a common interface to be used for different objects, even though the underlying code for each object is different. Rexx keeps track of the methods each object owns, which allows for the reuse of the same method name so that one message can initiate multiple functions. This helps to simplify naming schemes and makes complex programs easier to understand and modify. Polymorphism involves a contract between two objects. One object sends a message to another object expecting a particular result and different objects can implement different versions of this message as long as it meets the expectations of the invoking object (Ashley W. D., et al., ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide, 2022).

3.4.2.3. Unknown Messages

If an object doesn't have a method matching the received message name, the language processor looks for an UNKNOWN method in the object's inheritance tree. If located, the UNKNOWN method is triggered with two arguments: the first being the name of the missing message and the second being an array containing the arguments sent with the original message. If no UNKNOWN method is found, ooRexx raises the NOMETHOD error (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

3.4.3. ooRexx – Built-in Classes

The ooRexx Built-in Classes provide a range of capabilities for developing powerful Rexx programs. The built-in classes can be categorized into several groups including Fundamental Classes, Classic Rexx Classes, Collection Classes and Utility Classes.

Fundamental Classes include Object, Class, Method, Message, Routine and Package, which are used by the ooRexx interpreter to create and run programs.

Classic Rexx Classes, such as Stem, Stream and String, help ooRexx run Rexx programs easily and efficiently.

Collection Classes, like Array, Directory, Relation, Table and Directory, allow for organizing and retrieving objects in different types of containers. The DO m OVER keyword instruction makes it simple to iterate over collections. Utility Classes, such as Alarm, DateTime, File, Monitor, MutableBuffer and TimeSpan, provide useful features and capabilities for programmers (Flatscher R. G., Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..), 2013).

3.5. BSF4ooRexx

BSF4ooRexx, which stands for Bean Scripting Framework for ooRexx, is an external Rexx function package. It consists of an external Rexx function package and an ooRexx package named BSF.CSL, which loads the function package and defines an ooRexx class named BSF. This package enables the use of the Java Runtime Environment functions without prior knowledge of Java programming. The Java class libraries and Java objects have been masked to appear as ooRexx class libraries and ooRexx objects, to which messages can be sent (Flatscher R. G., Automatisierung mit ooRexx und BSF4ooRexx, 2012).

4. JavaFX Libraries

JavaFX provides a number of built-in styling libraries, but there are also several third-party libraries available that can be used to enhance the styling and customization options. There are several third-party styling libraries available for JavaFX, including JFoenix and ControlsFX. By adding them as dependencies to the project, they can easily be integrated into a JavaFX application. Furthermore, they can improve the visual appeal and functionality of JavaFX applications.

4.1. Styling Library – JFoenix

JFoenix is a Java library that provides a set of JavaFX UI controls and design elements that are styled to look like the material design guidelines from Google. This library allows Java developers to create modern and attractive user interfaces using JavaFX, which is a library for creating rich client applications in Java. JFoenix provides a wide range of controls and elements such as buttons, checkboxes, tables, dialogs and more, that can be used to create a modern and consistent look and feel across different parts of an application. It also provides a set of pre-built animations and effects that can be used to enhance the visual appeal of the UI. JFoenix is open-source and can be easily integrated into any JavaFX application. It is actively maintained and has a strong community of developers and contributors (JFoenix, 2023).

4.2. Widget Library – ControlsFX

ControlsFX is a JavaFX library that provides additional UI controls and features that are not included in the standard JavaFX library. It is designed to enhance the functionality of JavaFX and make it easier for developers to create rich and attractive user interfaces.

ControlsFX provides a wide range of controls and features such as:

- Dialogs and Alerts
- Table filtering and sorting
- Text fields with built-in validation and formatting
- Rich text editor
- Master/Detail View
- Notifications and Popup
- Undo/Redo Framework
- and many more.

It also provides a set of pre-built animations and effects that can be used to enhance the visual appeal of the UI.

ControlsFX is open-source and it is developed mainly for the JavaFX versions 8.0 and above, it has a principle that new features or controls will be accepted only if the current code is in a higher version (ControlsFx, 2023).

5. Required Software and Installation

This chapter describes how to install and configure all the essential software components for the nutshell examples. The following installation instructions are tailored for the Windows operating system, but they can also be used on Linux and MacOS provided that the appropriate components are downloaded and installed. The software versions used in this bachelor's thesis are listed below:

- Java Liberica JDK8u362-Full (64 Bit)
- SceneBuilder-8.5.0: BSD License
- ooRexx 5.0.0 (64 Bit): GNU General Public Liicense-version-2.0, Common Public License Version 1.0
- BSF4ooRexx850: Apache License Version 2.0
- sqlite-jdbc-3.41.0.0: Apache License Version 2.0
- jfoenix-8.0.10: Apache License Version 2.0
- controlsfx-8.40.18: BSD 3-Clause License
- DB Browser for SQLite-3.12.2: GNU General Public License Version 3.0

5.1. Java

It is necessary to install Java version 8 on the operating system. It is important to note that the installed version of Java has the same bit rate as ooRexx and includes JavaFX, as JavaFX is necessary for the graphical display of the user interfaces and is not available in every version of Java.

Furthermore, it is important to note that the corporation Oracle, which manages Java, since Java version 8 or later, if Java is used in a commercial context, there may be licensing fees applicable. However, there are free versions of Java available for download at the following link: https://bell-sw.com/pages/downloads/ (accessed 15-02-2023).

Once the appropriate Java version has been downloaded, run the setup file and follow the installation process to complete the installation.

5.2. ooRexx

Once the required version of Java has been installed, the next step is to install ooRexx version 5.0. The required version can be downloaded at the following link: https://sourceforge.net/projects/oorexx/files/ (accessed 15-02-2023).

To avoid any potential program errors, it is important to ensure that the installation file has the same bit architecture as the operating system and Java. After running the .exe file for ooRexx, an installation manager will appear, which should be followed. After completing the installation process, ooRexx will be successfully installed.

5.3. BSF4ooRexx

After successfully installing Java and ooRexx, the next step is to download and install BSF4ooRexx. The appropriate version of BSF4ooRexx can be downloaded from the following link: https://sourceforge.net/projects/bsf4oorexx/ (accessed 15-02-2023).

To install BSF4ooRexx, the downloaded ZIP archive must be extracted to any location within the operating system. After extraction, a subdirectory named "install" will appear, which contains installation files for all relevant operating systems (Windows, Mac, Linux). It is important to ensure that the correct version corresponding to the operating system is executed during the installation process.

5.4. SceneBuilder

In order to use the graphical user interface (GUI) builder for JavaFX applications, called Scene Builder, it must first be installed on the operating system. The installation files for Scene Builder can be downloaded from the following link: https://gluonhq.com/products/scene-builder/#download (accessed 15-02-2023).

After downloading the installation files, run the setup file and follow the installation process. Once the installation is complete, Scene Builder can be accessed through its shortcut in the start menu or by running the "SceneBuilder.exe" file.

It is important to note that Scene Builder requires Java to be installed on the system and that the installed version of Java must match the bit rate of the Scene Builder.

5.5. JavaFX Libraries

In order to use JavaFX libraries JFoenix or ControlsFX in a JavaFX project or in SceneBuilder, the first step is to download the library from the official website. JFoenix can be downloaded from the following URL: https://github.com/sshahine/JFoenix (accessed 15-02-2023), while ControlsFX can be downloaded from the following URL: https://mvnrepository.com/artifact/org.controlsfx/controlsfx/8.40.18 (accessed 15-02-2023).

Once the JAR file has been downloaded, it must be added to the CLASSPATH variable. Detailed instructions are provided in Chapter 5.7.

To use either library in SceneBuilder, open the SceneBuilder and create a new FXML file or open an existing one. Then, select the "Library" tab on the right-hand side of the SceneBuilder window and click on the "Add Library/FXML" button. Navigate to the location where the downloaded JAR file is saved and select it. The library will now be added to the SceneBuilder library list and its components can be used by selecting them from the respective library option in the SceneBuilder controls section.

5.6. SQLite Browser

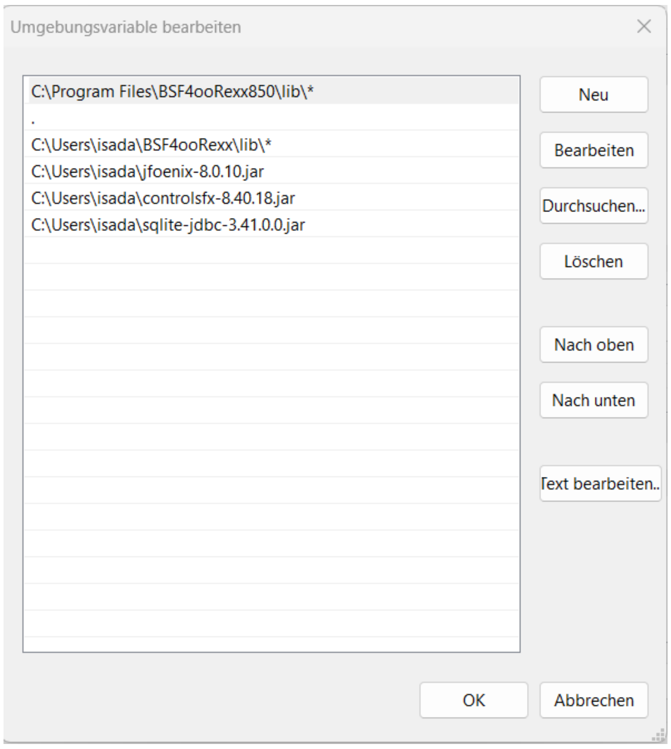
The SQLite Database Browser is a clear and simple database creation tool. The program allows to create, read, edit and delete databases and data sets. The installation files can be downloaded from the following link: https://sqlitebrowser.org/dl/(accessed 15-02-2023). After downloading the installation files, run the setup file and follow the installation process. Before using a SQLite database in a Java project, it's necessary to download the SQLite JDBC library from this https://github.com/xerial/sqlite-jdbc/releases (accessed 15-02-2023) and add the JAR file to the CLASSPATH variable. Detailed instructions are provided in Chapter 5.7.

5.7. CLASSPATH

In order to run Java applications that depend on external libraries, it is necessary to add the library files to the *CLASSPATH*. The *CLASSPATH* is a list of directories and JAR files that the Java Virtual Machine (JVM) uses to look for classes that are not included in the application's own source code (Classpath, 2023).

In order to include JAR files in the *CLASSPATH* on a Windows operating system, follow these steps:

1. Open the "Control Panel" from the Windows Start menu.
2. Select "System and Security", then click on "System".
3. Click on "Advanced system settings" on the right-hand side of the window.
4. In the "System Properties" window, click on the "Environment Variables" button.



```
C:\Program Files\BSF4ooRexx850\lib\*
.
C:\Users\isada\BSF4ooRexx\lib\*
C:\Users\isada\jfoenix-8.0.10.jar
C:\Users\isada\controlsfx-8.40.18.jar
C:\Users\isada\sqlite-jdbc-3.41.0.0.jar
```

Figure 3: Adding New Path to CLASSPATH

5. Under "System Variables", scroll down and find the "CLASSPATH" variable, then click "Edit".
6. In the "Edit Environment Variable" window, click "New" to add a new path to the *CLASSPATH*.
7. Enter the file path of the directory containing the JAR files to include in the *CLASSPATH*.
8. Click "OK" to close all the windows.

6. Nutshell Examples

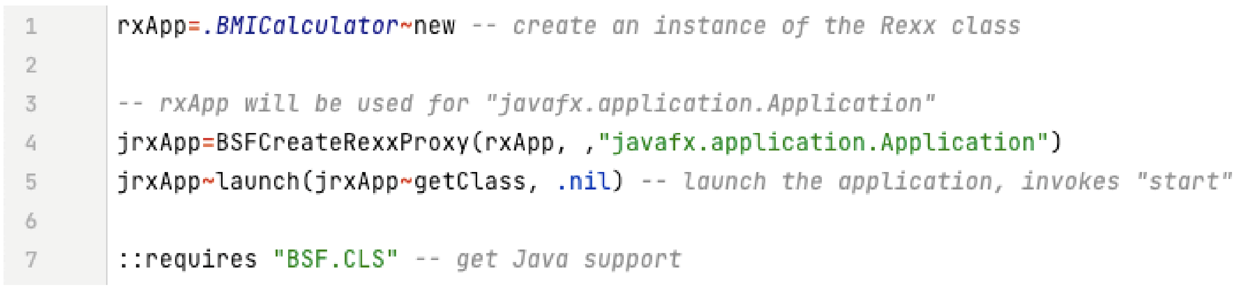
The focus of this chapter is on the development of JavaFX graphical user interface (GUI) applications using various libraries, tools and technologies in connection with the programming language ooRexx. To illustrate the flexibility and diversity of ooRexx programming, each application is developed using a different set of tools and technologies, including FXML, CSS, JDBC, JFoenix Library and ControlsFX Library. The integration of BSF4ooRexx provides a powerful mechanism for developing GUI applications that combine the strengths of both Java and ooRexx.

6.1. JavaFX GUI Application with ooRexx

The Nutshell example is an simple ooRexx script that acts as a BMI calculator and employs the JavaFX library to produce a graphical user interface (GUI) without the use of FXML. The application logic is contained within the script, which generates the GUI components and handles user inputs using event-driven programming. In the following section, the script will be discussed, which comprises two Rexx classes - BMICalculator and RexxButtonHandler. The BMICalculator Rexx class is responsible for implementing the GUI, while the RexxButtonHandler Rexx class offers functions for BMI calculation and button event handling.

To use the *BMICalculator* class for the GUI, an instance of this class is created with the line "*rxApp = .BMICalculator~new*" in the Snippet 1. This creates a new instance of the class and stores it in the variable "*rxApp*". A proxy instance for the Rexx class must also be created so that it can be used in the JavaFX application. This is done using the "*BSFCreateRexxProxy*" method from the Java Bean Scripting Framework (BSF) in line 4 of the Snippet 1. This creates a new proxy object for the Rexx instance, which is stored in the variable "*jrxApp*".

The second argument indicates that the Rexx instance is to be used within a JavaFX application.

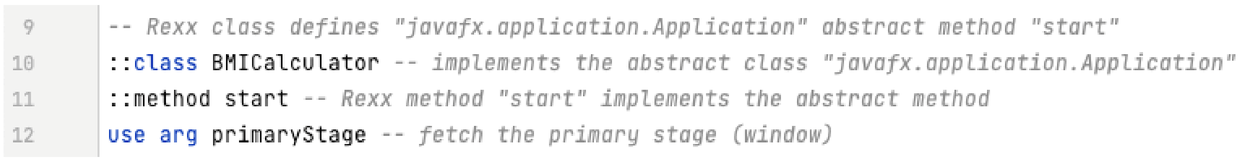


```
rxApp=.BMICalculator~new -- create an instance of the Rexx class

-- rxApp will be used for "javafx.application.Application"
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil) -- launch the application, invokes "start"

::requires "BSF.CLS" -- get Java support
```

Snippet 1:BMI-Calculator - Application launch

To start the application, the "*launch*" method is called on the proxy object: "*jrxApp~getClass, .nil*". This calls the start method of the "*BMICalculator"* class, which implements the abstract start method defined in the *javafx.application.Application* class, that is required for every JavaFX application. The "BSF.CLS" package is loaded to provide Java support for the Rexx code.

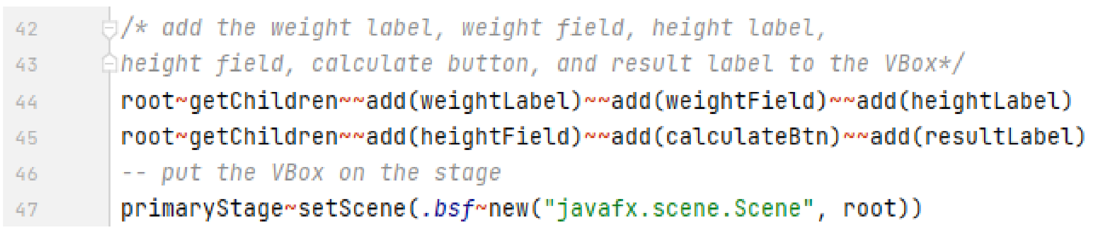
The "start" method in Snippet 2 is defined in the "*BMICalculator*" class and gets a parameter "*primaryStage*", which represents the primary window of the application. The "*use arg*" statement is used to extract the "*primaryStage*" parameter and use it in the method. In the "start" method, various UI elements such as text fields, labels and buttons are created and customized using the JavaFX library. These elements are placed in a VBox container, which is a vertical layout element called "*root*". The VBox container is then added to the scene, which is responsible for displaying the UI elements on the screen.



```
-- Rexx class defines "javafx.application.Application" abstract method "start"
::class BMICalculator -- implements the abstract class "javafx.application.Application"
::method start -- Rexx method "start" implements the abstract method
use arg primaryStage -- fetch the primary stage (window)
```

Snippet 2: BMICalculator Class

The "*getChildren*" method of the VBox object is used to add the various UI elements to the VBox. In line 44 of the Snippet 3 the "*add*" method and the concatenation operator ("~~") are used to add the various elements in the correct order. Once the VBox is created and filled with the UI elements, it is added to the scene by creating a new Scene object.



```
42  /* add the weight label, weight field, height label,
43  height field, calculate button, and result label to the VBox*/
44  root~getChildren~~add(weightLabel)~~add(weightField)~~add(heightLabel)
45  root~getChildren~~add(heightField)~~add(calculateBtn)~~add(resultLabel)
46  -- put the VBox on the stage
47  primaryStage~setScene(.bsf~new("javafx.scene.Scene", root))
```

Snippet 3: BMI- Calculator -Add Components

This Scene object is initialized with the VBox as the root element. Finally, the scene is assigned to the primary window (*primaryStage*)and the window is displayed using the "show" method so that the user can see the application on the screen.

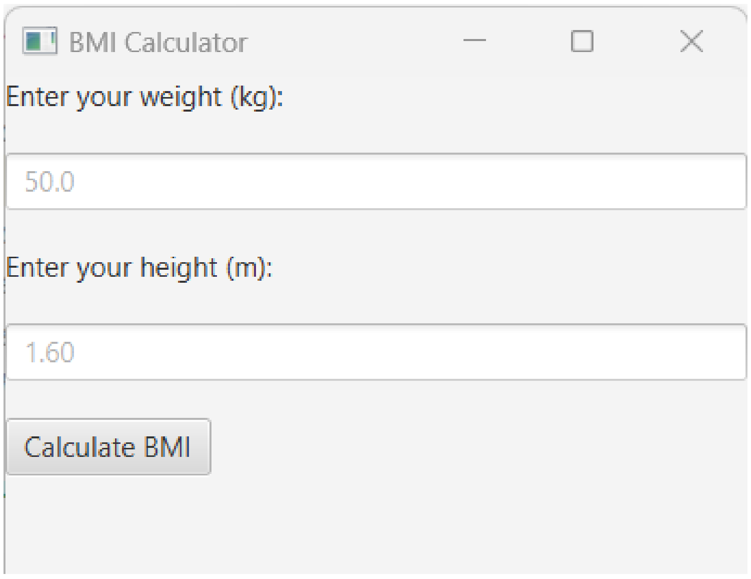
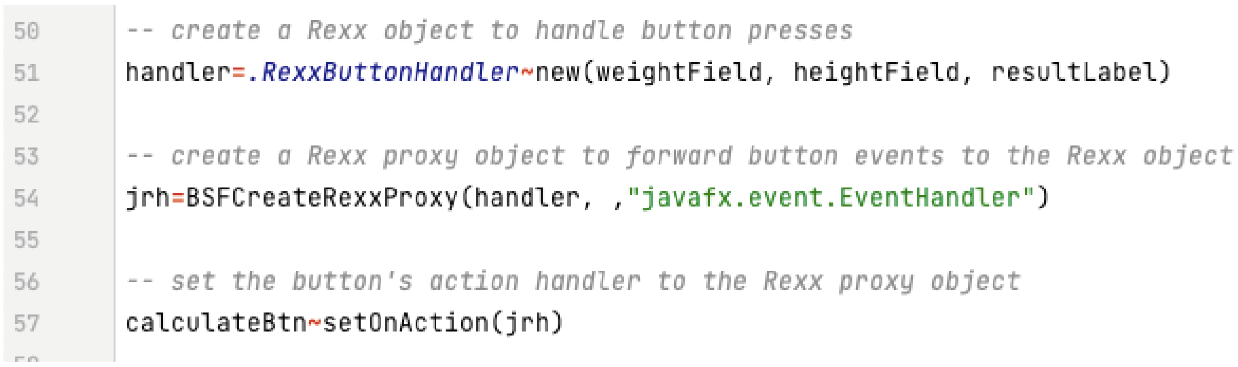


Figure 4:BMI -Calculator - GUI

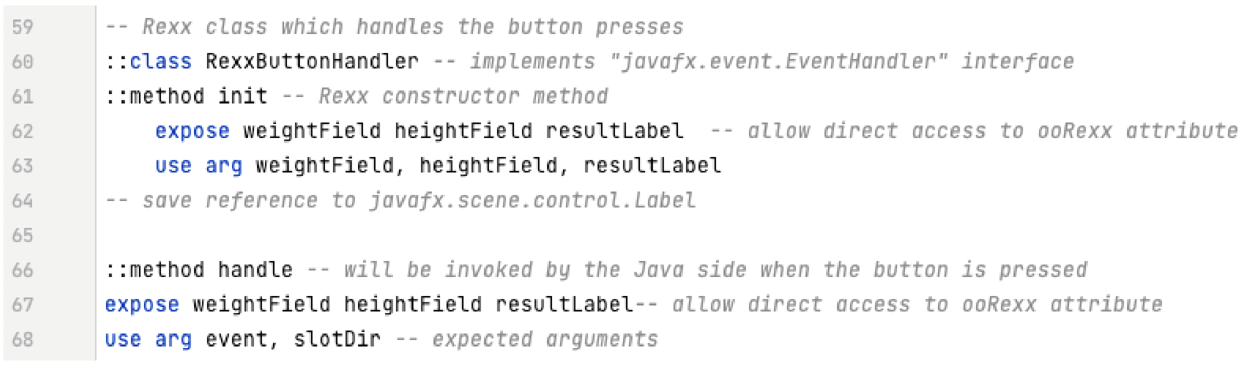
To implement the functionality of the "*Calculate BMI*" button, a button handler is created that is called when the button is pressed. For this purpose, a separate Rexx object called "*handler*" is created. This object is an instance of the “*RexxButtonHandler*” class, which implements the *javafx.event.EventHandler* interface. The parameters weightField, heightfield and resultLabel are passed to this object. Then a Rexx proxy is created that forwards the button events to the Rexx object. This proxy is then set as the handler for the "setOnAction" call of the "calculateBtn" button as in Line 57 of the snippet 4.



```
50  -- create a Rexx object to handle button presses
51  handler=.RexxButtonHandler~new(weightField, heightField, resultLabel)
52
53  -- create a Rexx proxy object to forward button events to the Rexx object
54  jrh=BSFCreateRexxProxy(handler, ,"javafx.event.EventHandler")
55
56  -- set the button's action handler to the Rexx proxy object
57  calculateBtn~setOnAction(jrh)
```

Snippet 4: BMI-Calculator - ButtonHandler

The constructor of the “*RexxButtonHandler*” class get three arguments: weightField, heightfield and resultLabel. The "*handle*" method is called when the button is pressed and performs the BMI calculation based on the values entered in the text fields.



```
-- Rexx class which handles the button presses
::class RexxButtonHandler -- implements "javafx.event.EventHandler" interface
::method init -- Rexx constructor method
    expose weightField heightField resultLabel  -- allow direct access to ooRexx attribute
    use arg weightField, heightField, resultLabel
-- save reference to javafx.scene.control.Label

::method handle -- will be invoked by the Java side when the button is pressed
expose weightField heightField resultLabel-- allow direct access to ooRexx attribute
use arg event, slotDir -- expected arguments
```

Snippet 5:BMI-Calculator, Constructor and Handle Method

6.2. JavaFX GUI Application with FXML in ooRexx

This chapter presents an extension of the nutshell example introduced in Chapter 6.1, demonstrating how to create a user-friendly graphical user interface (GUI) using ooRexx, FXML and CSS. The GUI is created using the Scene Builder, which provides an intuitive visual interface for designing the user interface.

To design the user interface of the application, Gluon's Scene Builder was used. In the Scene Builder, GUIs can be quickly and easily created by dragging and dropping various elements from the tool palette. These elements can be placed and aligned to the desired location by simply dragging and adjusting them. Various properties such as size, font, position and name of the GUI components can be easily modified through the 'Properties' and 'Layout' columns on the right-hand side. To access the elements in the application programmatically, unique IDs must be assigned to these elements in the FXML code. Assigning ids is important so that the code can access and control the elements of the user interface. The ids are marked with the prefix "*fx:id*" and can be assigned by the developer themselves. The three necessary menu entries for making these changes are shown in Figure 5.

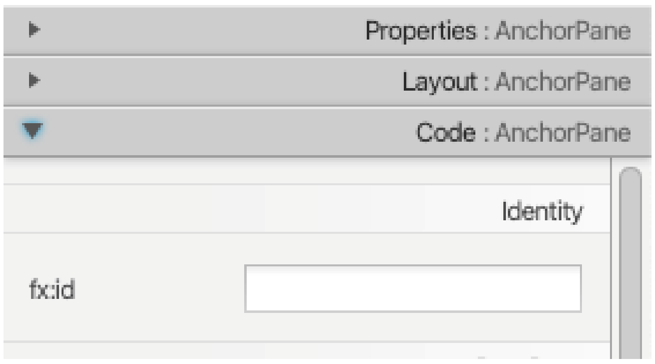
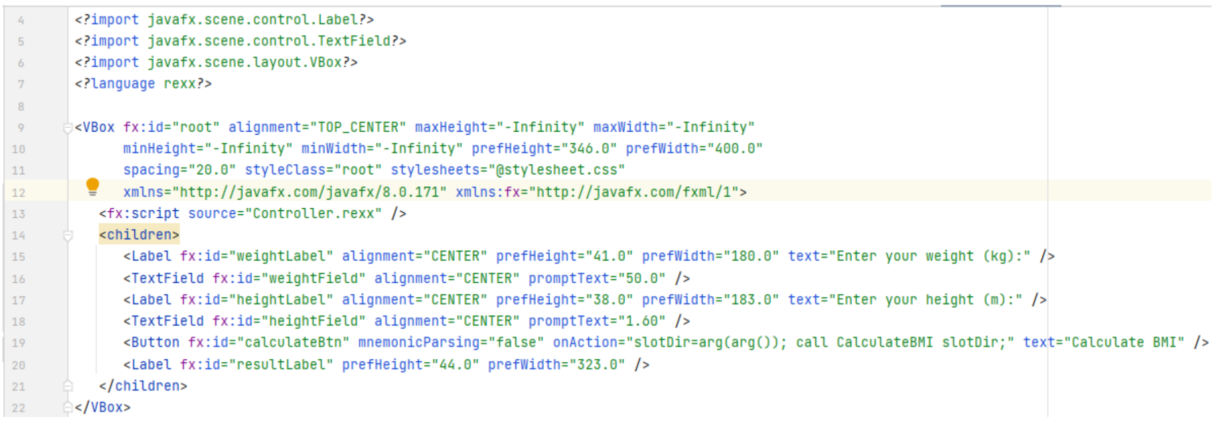


Figure 5: Scene Builder

The Scene Builder also provides a preview feature that allows to view and test the user interface during development.

Once the graphical user interface is created, it can be saved as an FXML file. The FXML code generated by the Scene Builder, is shown in Snippet 6. Before the GUI can be accessed and used by ooRexx, a few modifications must be made to the FXML file. After making these adjustments, the FXML file can be loaded into the ooRexx program and the GUI can be fully functional.



```
4   <?import javafx.scene.control.Label?>
5   <?import javafx.scene.control.TextField?>
6   <?import javafx.scene.layout.VBox?>
7   <?language rexx?>
8
9   <VBox fx:id="root" alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-Infinity"
10        minHeight="-Infinity" minWidth="-Infinity" prefHeight="346.0" prefWidth="400.0"
11        spacing="20.0" styleClass="root" stylesheets="@stylesheet.css"
12        xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1">
13     <fx:script source="Controller.rexx" />
14     <children>
15        <Label fx:id="weightLabel" alignment="CENTER" prefHeight="41.0" prefWidth="180.0" text="Enter your weight (kg):" />
16        <TextField fx:id="weightField" alignment="CENTER" promptText="50.0" />
17        <Label fx:id="heightLabel" alignment="CENTER" prefHeight="38.0" prefWidth="183.0" text="Enter your height (m):" />
18        <TextField fx:id="heightField" alignment="CENTER" promptText="1.60" />
19        <Button fx:id="calculateBtn" mnemonicParsing="false" onAction="slotDir=arg(arg()); call CalculateBMI slotDir;" text="Calculate BMI" />
20        <Label fx:id="resultLabel" prefHeight="44.0" prefWidth="323.0" />
21     </children>
22  </VBox>
```

Snippet 6: BMI Calculator-FXML

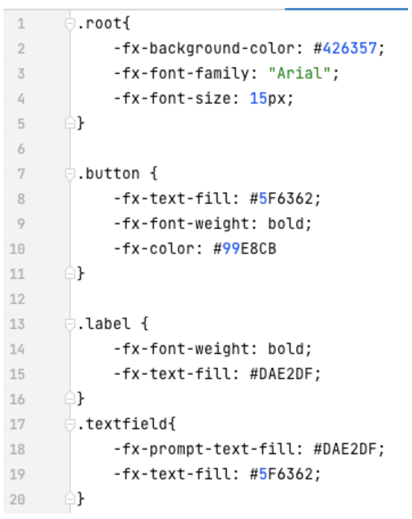
The first modification that needs to be made is in line 7, which concerns the definition of the JavaScript engine "*rexx*" that is used in case of code triggered by an event. The second modification in line 16 defines such an event, named "*onAction*", triggered by a click on the button. The Rexx code contained in this event is called by JavaFX in Rexx, as shown in line 19. This Rexx code calls a routine named "*CalculateBMI*" and passes the argument "*slotDir*", which contains all the information required to access the elements used in the GUI. The routine specified in line 13 can now be called by JavaFX. (Flatscher R. , 2023).

The Scene Builder allows the user to apply CSS rules directly to the GUI elements. This can be used, for example, to change the appearance of buttons, labels, or other controls. The CSS settings can be made in the right-hand side panel under the "CSS" tab. Various properties such as font, background color, or text color can be modified.

The FXML file can also be manually extended with CSS rules. To add CSS rules, the "styleClass" property of the respective element must be set in the FXML file code. This property gives the element one or more class names that can be used in the CSS file to define the formatting of the element. Afterwards, the CSS rules can be defined in a separate CSS file, which is then included in the FXML file.

CSS rules can format various attributes of the element, such as background color, text color, font size, margins and borders. In addition, CSS rules can also use pseudo-selectors to format certain states of the element, such as hover or active state. Embedding CSS rules directly in the code of the FXML file is possible, but it can lead to cluttered code (Flatscher R. , 2023).

To better illustrate this, the stylesheet of the nutshell example is shown in Snippet 7 and in Figure 7 the styled application.



```css
.root{
    -fx-background-color: #426357;
    -fx-font-family: "Arial";
    -fx-font-size: 15px;
}

.button {
    -fx-text-fill: #5F6362;
    -fx-font-weight: bold;
    -fx-color: #99E8CB
}

.label {
    -fx-font-weight: bold;
    -fx-text-fill: #DAE2DF;
}
.textfield{
    -fx-prompt-text-fill: #DAE2DF;
    -fx-text-fill: #5F6362;
}
```

Snippet 7:BMI Calculator-Stylesheet

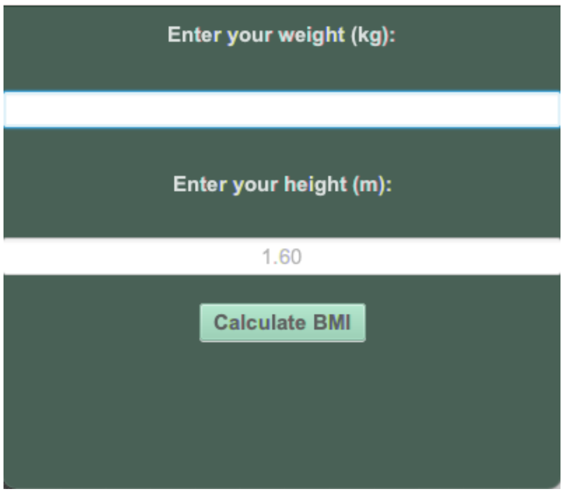
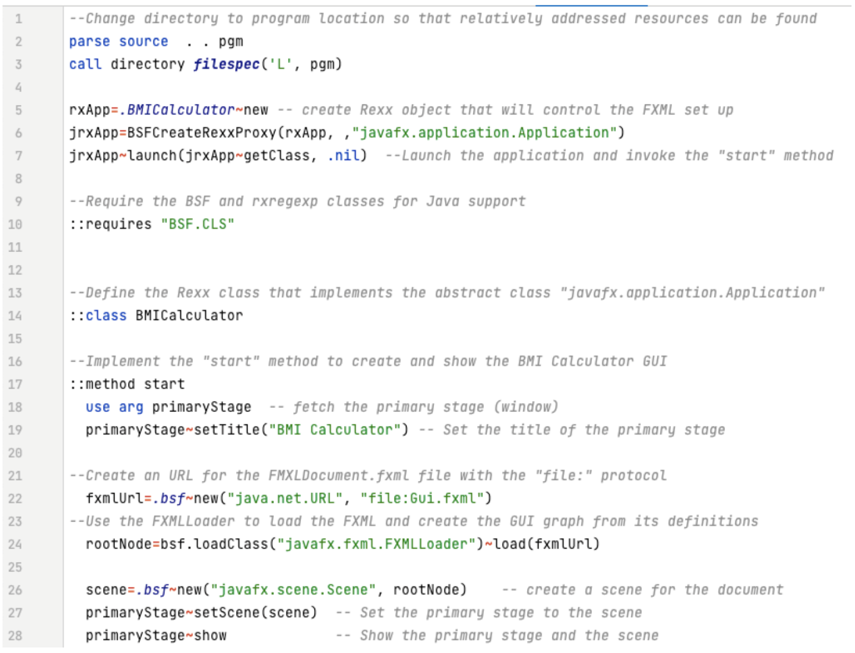


Figure 6:BMI Calculator GUI

The code snippet 8 demonstrates the implementation of a basic BMI calculator application in ooRexx. This is achieved by defining a class named "BMICalculator" and creating an instance of this class, which includes the FXMLLoader to integrate the graphical user interface.



```
--Change directory to program location so that relatively addressed resources can be found
parse source  . . pgm
call directory filespec('L', pgm)

rxApp=.BMICalculator~new -- create Rexx object that will control the FXML set up
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil)  --Launch the application and invoke the "start" method

--Require the BSF and rxregexp classes for Java support
::requires "BSF.CLS"


--Define the Rexx class that implements the abstract class "javafx.application.Application"
::class BMICalculator

--Implement the "start" method to create and show the BMI Calculator GUI
::method start
  use arg primaryStage  -- fetch the primary stage (window)
  primaryStage~setTitle("BMI Calculator") -- Set the title of the primary stage

--Create an URL for the FMXLDocument.fxml file with the "file:" protocol
  fxmlUrl=.bsf~new("java.net.URL", "file:Gui.fxml")
--Use the FXMLLoader to load the FXML and create the GUI graph from its definitions
  rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)

  scene=.bsf~new("javafx.scene.Scene", rootNode)    -- create a scene for the document
  primaryStage~setScene(scene)  -- Set the primary stage to the scene
  primaryStage~show             -- Show the primary stage and the scene
```

Snippet 8: BMI-Calculator- Class

First an object of the BMICalculator class is created and passed as a proxy object to the Java Virtual Machine. The "*launch*" method is called to start the application and invoke the "*start*" method of the *BMICalculator* class.

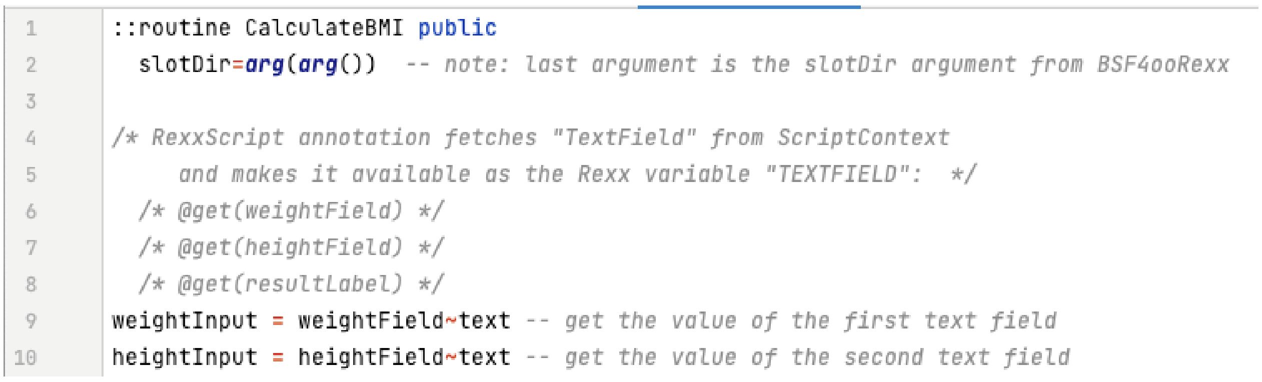
In the "start" method, the main window is created and the title "*BMI Calculator*" is set. Then, a URL instance is created to load an FXML file named "*Gui*.fxml" that defines the user interface of the BMI calculator application. The URL is created with the "file:" protocol and passed as an argument to the "*java.net.URL*" class.

Next, the "*FXMLLoader*" class is called to load the FXML file and create a GUI element from it. The "*load*" method of the "*FXMLLoader*" class is called and the URL instance is passed as an argument. The loaded GUI element is stored in a variable called "*rootNode*".

After that, a new "*Scene*" instance is created and the previously loaded GUI element is passed as an argument to the constructor. The "*Scene*" instance is then passed as an argument to the "*setScene*" method of the "*primaryStage*" object to set the created scene as the main display area of the application.

Finally, the "*show*" method of the "*primaryStage*" object is called to display the main window with the created user interface. The code uses the "BSF" library to create and use Java objects within the ooRexx program.

The code snippet 9 presents the 'CalculateBMI' method that is a part of the Rexx controller in the nutshell example and performs the calculation of BMI.



```
::routine CalculateBMI public
  slotDir=arg(arg())  -- note: last argument is the slotDir argument from BSF4ooRexx

/* RexxScript annotation fetches "TextField" from ScriptContext
     and makes it available as the Rexx variable "TEXTFIELD":  */
  /* @get(weightField) */
  /* @get(heightField) */
  /* @get(resultLabel) */
weightInput = weightField~text -- get the value of the first text field
heightInput = heightField~text -- get the value of the second text field
```

Snippet 9:Routine CalculateBMI

In this example, the "*CalculateBMI*" method is invoked when the user clicks a button on the user interface. The method then retrieves the values from the input fields. In the second line, the *'slotDir'* parameter, supplied by the event, is initialized. The *'slotDir'* parameter is crucial as it provides access to the GUI's control elements through their 'fx:id'. The instruction "*/@get(weightField)/*" makes the textfield with the fx:id "weightField" available and initializes it as a variable in Rexx named "weightInput". The controller class in ooRexx can contain a variety of methods to manage different functions of the user interface. It can also call other classes and methods to process data or access external resources.

6.3. JavaFX GUI Application using JFoenix Library

Although JavaFX provides many predefined controls and functions for creating user interfaces, there may be situations where additional controls or styles are needed that are not available by default. To meet these requirements, external libraries like JFoenix can be used. JFoenix is a JavaFX library that provides additional controls and styles to make it easier to create modern user interfaces.

The nutshell example of this chapter presents a program for calculating a person's BMI and calorie requirement, which is styled using the JFoenix library. A start menu has been added to improve usability, allowing the user to choose between the two calculators. In this example, a StageHandler is also used to manage the different scenes. The following section explain the implementation of the StageHandler and show how GUI styling was achieved with JFoenix.

6.3.1. StageHandler

The StageHandler is an important concept in the development of JavaFX applications. It is an object that facilitates the management of windows and scenes in an application. The StageHandler provides a simple way to manage the different windows and scenes and navigate between them.

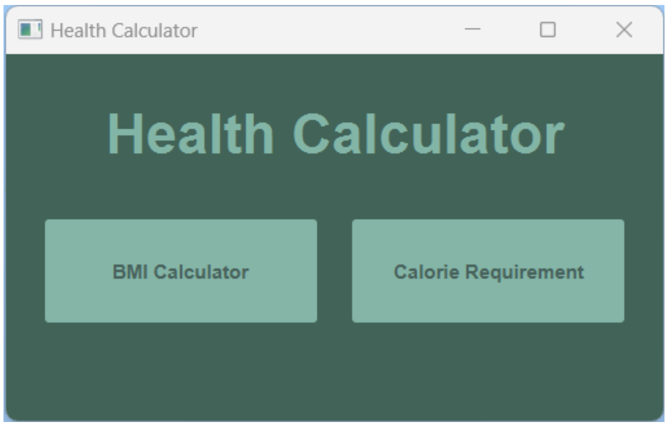
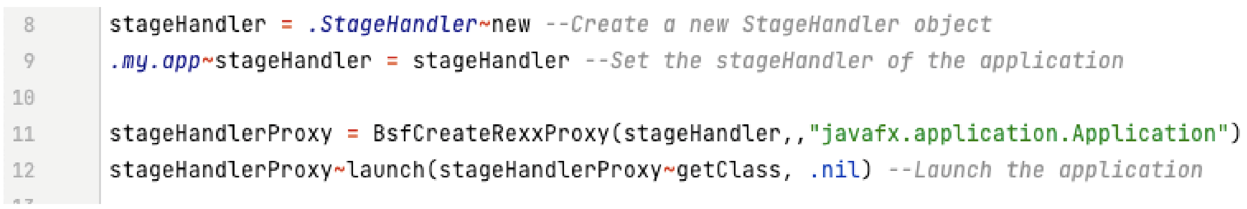


Figure 7: Health Calculator Menu

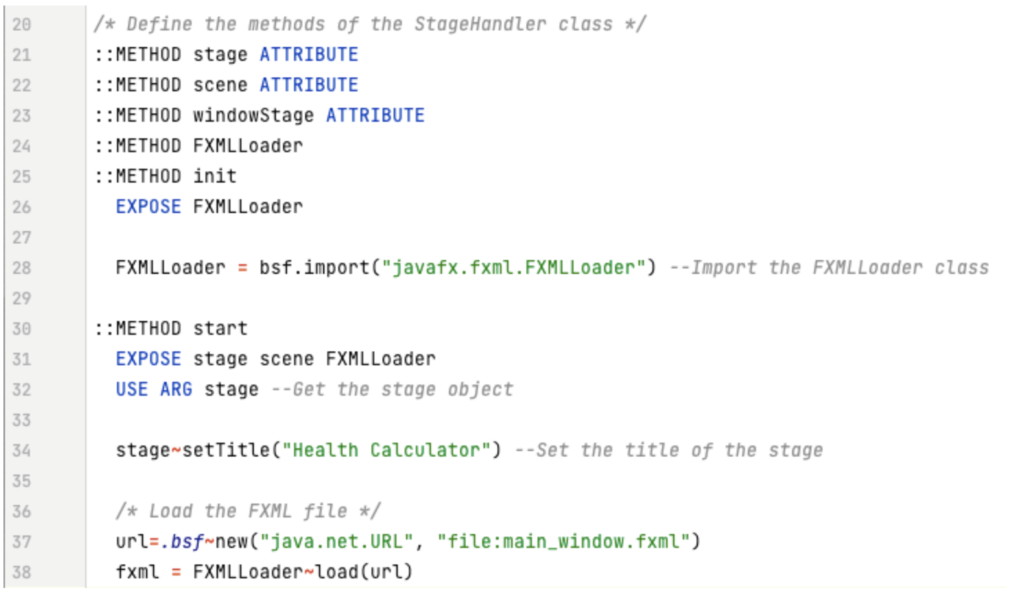
As presented in snippet 10, an instance of the StageHandler class is instantiated to create the StageHandler. Next, a proxy object called *stageHandlerProxy* is created that contains the StageHandler. This proxy object is used to call the *launch()* method, which starts the application.



```
stageHandler = .StageHandler~new --Create a new StageHandler object
.my.app~stageHandler = stageHandler --Set the stageHandler of the application

stageHandlerProxy = BsfCreateRexxProxy(stageHandler,,"javafx.application.Application")
stageHandlerProxy~launch(stageHandlerProxy~getClass, .nil) --Launch the application
```

Snippet 10:Launch StageHandler

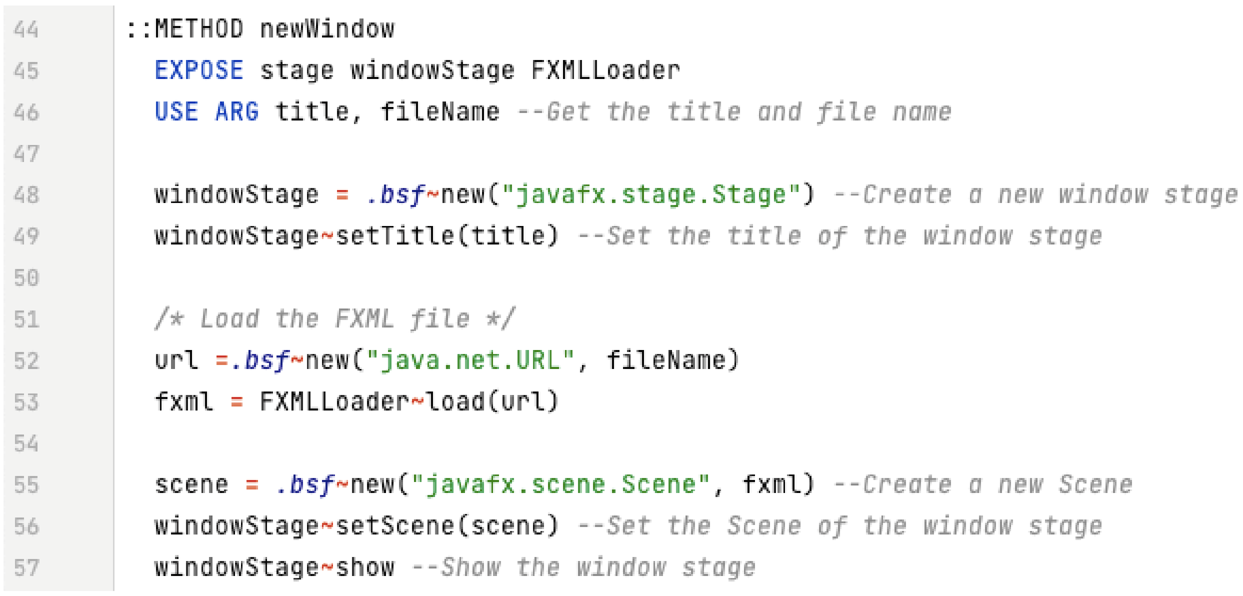
The *StageHandler* class, as shown in Snippet11, defines various methods used for window management. The *start()* method is called to create and display the main window. First, the title of the window is set and then the FXML document main_window.fxml is loaded in line 37. The loaded file is set as the content of the window.



```
20  /* Define the methods of the StageHandler class */
21  ::METHOD stage ATTRIBUTE
22  ::METHOD scene ATTRIBUTE
23  ::METHOD windowStage ATTRIBUTE
24  ::METHOD FXMLLoader
25  ::METHOD init
26    EXPOSE FXMLLoader
27
28    FXMLLoader = bsf.import("javafx.fxml.FXMLLoader") --Import the FXMLLoader class
29
30  ::METHOD start
31    EXPOSE stage scene FXMLLoader
32    USE ARG stage --Get the stage object
33
34    stage~setTitle("Health Calculator") --Set the title of the stage
35
36    /* Load the FXML file */
37    url=.bsf~new("java.net.URL", "file:main_window.fxml")
38    fxml = FXMLLoader~load(url)
```

Snippet 11:StageHandler Class

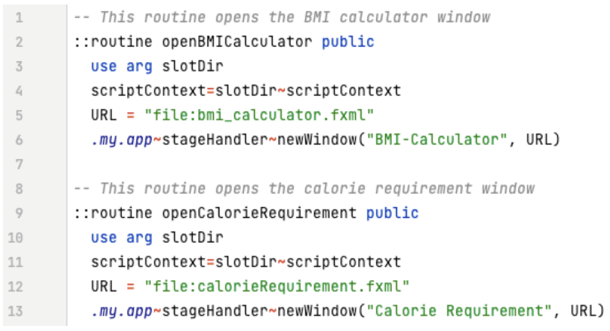
To create a new window, the newWindow() method must be called on the StageHandler class. It receives the title of the window and the name of the FXML file as arguments. The FXML document is loaded and set as the content of the new window. This is shown in the Snippet 12. To use the StageHandler class in JavaFX, the BSF.CLS library is required to be able to call Java code from Rexx code.



```
::METHOD newWindow
  EXPOSE stage windowStage FXMLLoader
  USE ARG title, fileName --Get the title and file name

  windowStage = .bsf~new("javafx.stage.Stage") --Create a new window stage
  windowStage~setTitle(title) --Set the title of the window stage

  /* Load the FXML file */
  url =.bsf~new("java.net.URL", fileName)
  fxml = FXMLLoader~load(url)

  scene = .bsf~new("javafx.scene.Scene", fxml) --Create a new Scene
  windowStage~setScene(scene) --Set the Scene of the window stage
  windowStage~show --Show the window stage
```

Snippet 12:Method newWindow

The Rexx contoller contains two routines, shown in the Snippet 13, each of which is called when one of the buttons in the application is pressed. Both routines use the “slotDir” argument. Both routines work according to the same scheme when a button in the application is pressed. First, the “*newWindow*” method of the StageHandler object is called and the URL of the FXML layout is passed. Also, the title of the window is set. Overall, these routines provide a user-friendly application, as the user can directly access the desired calculator by pressing the corresponding button, without having to close and restart the application.



```
-- This routine opens the BMI calculator window
::routine openBMICalculator public
  use arg slotDir
  scriptContext=slotDir~scriptContext
  URL = "file:bmi_calculator.fxml"
  .my.app~stageHandler~newWindow("BMI-Calculator", URL)

-- This routine opens the calorie requirement window
::routine openCalorieRequirement public
  use arg slotDir
  scriptContext=slotDir~scriptContext
  URL = "file:calorieRequirement.fxml"
  .my.app~stageHandler~newWindow("Calorie Requirement", URL)
```

Snippet 13:Routines to open a window

6.3.2. JFoenix Library

As described in section 5.5, after successful installation, the components can be inserted into the application through drag and drop in the Scene Builder. In the present example, slider, button and textfield components were used from the library.

If the Scene Builder is not used for creating the user interface, the components must be imported in the FXML file. The implementation is shown in the following Snippet 14.

```
<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXSlider?>
<?import com.jfoenix.controls.JFXTextField?>
```

Snippet 14:JFoenix Import

One of the main features of JFoenix is the support for CSS styling for UI components.

In the nutshell example, the slider was used for inputting user data such as size, weight and age. With JFoenix, the slider style can be easily customized by using various CSS properties. For example, the colors for the slider track and slider thumb can be changed by using the " *-fx-background-color*" and " *-fx-background*" properties.

Additionally, the size of the slider thumb can be adjusted by using the " *-fx-background-size*" property.

Below a snippet from the CSS file showing how the slider was styled in the application.

```
/* Styling the slider track */
.jfx-slider > .track {
     -fx-background-color: #84B5A7;
}
```

```
/* Styling the slider thumb */
.jfx-slider > .thumb {
	-fx-background-color: #84B5A7;
}

/* Styling the filled track */
.jfx-slider > .colored-track {
	-fx-background-color: #84B5A7;
}

/* Styling the animated thumb */
.jfx-slider > .animated-thumb {
	-fx-background-color: #84B5A7;
}

/* Styling the slider value text inside animated thumb */
.jfx-slider .slider-value {
	-fx-fill: #426357;
	-fx-stroke: #426357;
}
```

Snippet 15:Slider CSS – Stylesheet

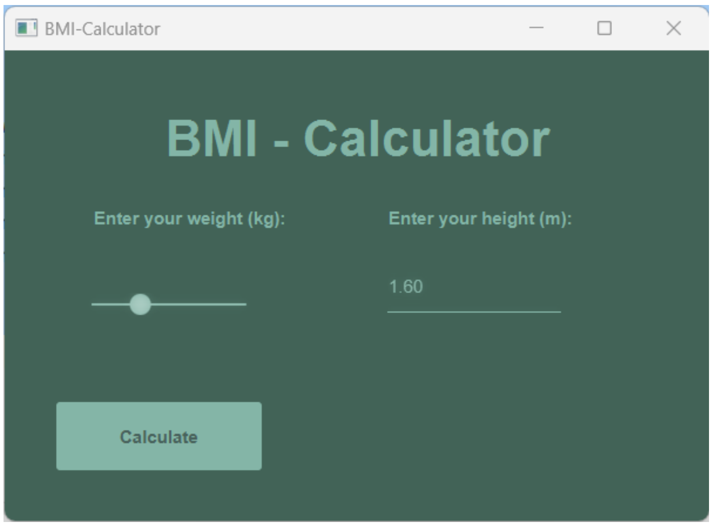


Figure 8: BMI- Calculator JFoenix

6.4. JavaFX GUI Application using ControlsFX Library

The upcoming chapter explores how to use ControlsFX to enhance the user interface of a JavaFX application and implement SQLite and JDBC with a DatabaseHandler for data management. ControlsFX is a valuable library for extending the UI components of JavaFX applications, which will be integrated into a form. Additionally, storing and retrieving data is crucial for JavaFX applications and this is where SQLite and JDBC come in. The chapter presents a nutshell example of a form that stores user data in a database and is styled using the ControlsFX library.

6.4.1. ControlsFX

As described in section 5.5, after successful installation, the components can be inserted into the application through drag and drop in the Scene Builder. In the present example, the rating and textfield components were used from the library.

If the Scene Builder is not used for creating the user interface, the components must be imported in the FXML file. The implementation is shown in the following Snippet 16.

```
<?import org.controlsfx.control.Rating?>
<?import org.controlsfx.control.textfield.CustomTextField?>
```

Snippet 16: ControlsFX Import

Textfields and ratings are essential components for collecting user input and displaying feedback in modern applications. Textfields in ControlsFX are an enhanced version of the standard JavaFX textfield control. They provide additional features such as auto-completion, validation and masking. Ratings in ControlsFX are an easy-to-use and customizable control that allow users to rate items on a scale of 1 to 5 stars. They provide a visual representation of the user's rating and can be styled to fit the applications design. The styling of the rating system is kept very simple and only a few elements have been adapted. Snippet 17 displays a section of the CSS file that illustrates how the Rating component was styled within the application.

```
.rating > .container .button {
           -fx-pref-width: 35 ;
           -fx-pref-height: 35 ;
           -fx-background-size: cover;
           -fx-background-color:#426357;
           -fx-padding: 15;
}
```

Snippet 17:Rating CSS Stylesheet

Following styles were added to the elements:

- "-fx-pref-width" and "-fx-pref-height" are set to 35, which sets the preferred width and height of the button to 35 pixels.
- "-fx-background-size" is set to "cover", which scales the background image to cover the entire button area.
- "-fx-background-color" is set to #426357, which sets the background color of the button to a dark shade of green.
- "-fx-padding" is set to 15, which adds 15 pixels of padding around the content of the button.

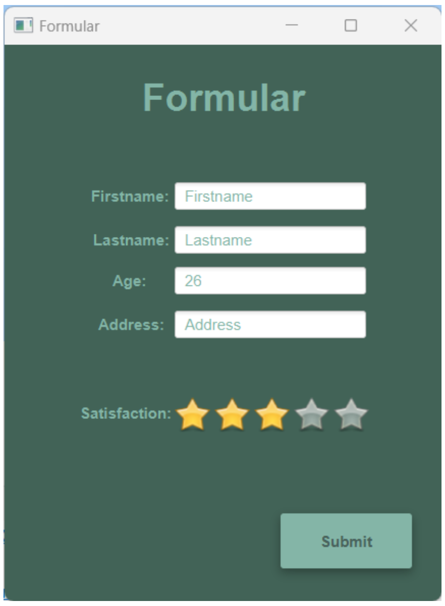


Figure 9: ControlsFX Form

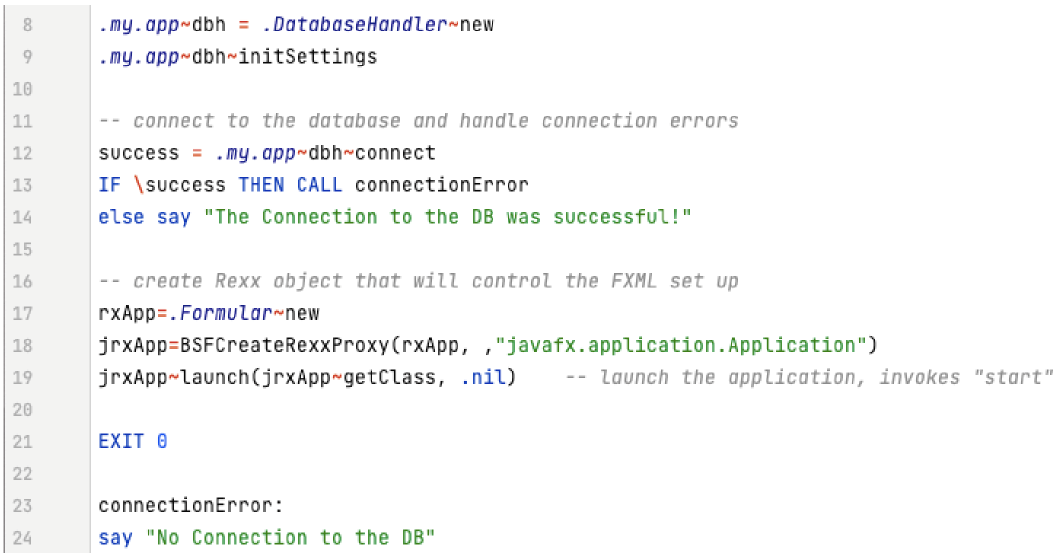
6.4.2. DatabaseHandler

In this example, utilize a lightweight relational database, SQLite, which is widely used in many applications. JDBC (Java Database Connectivity) was used to establish the connection between the database and the application. It is a Java API that provides a common interface between Java applications and various databases. When it comes to using databases in a JavaFX application, JDBC can be used to create a robust and user-friendly application that can store and retrieve data in a SQLite database (JDBC, 2023). For creating SQLite databases, JDBrowser can be used.

JDBrowser is a powerful tool for managing JDBC databases. It provides a graphical user interface (GUI) for connecting to different databases and allows users to execute SQL queries, view, edit and delete tables and data, as well as manage stored procedures and functions. With JDBrowser, users can also perform schema and data exports and create complex queries to retrieve specific information (DB Browser, 2023).

At the beginning, a database is created using the JDBrowser. In this example, the database consists of a single table named "Person". The table has a primary key called "id" and attributes such as "First Name", "Last Name", "Age", "Address" and "Satisfaction".

To access the database, a DatabaseHandler is required. Initially, an instance of the *DatabaseHandler* object, as in line 8 of the snippet 18, is created and its settings are initialized. Then, an attempt is made to establish a connection to the database. If the connection is successful, a success message is displayed, otherwise the "*connectionError*" function is called.

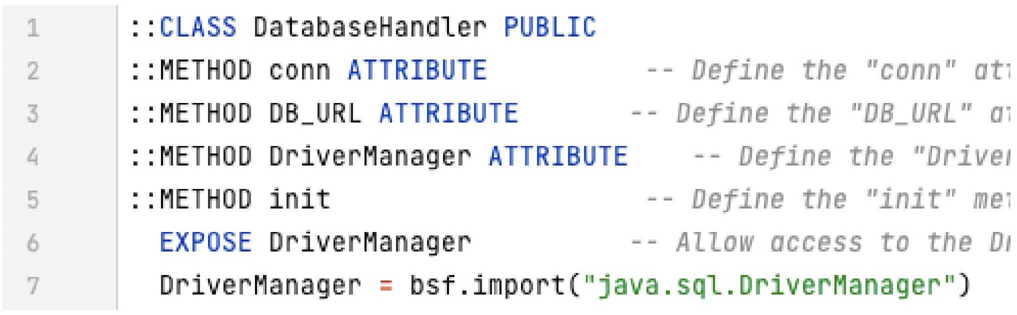


```
 8  .my.app~dbh = .DatabaseHandler~new
 9  .my.app~dbh~initSettings
10
11  -- connect to the database and handle connection errors
12  success = .my.app~dbh~connect
13  IF \success THEN CALL connectionError
14  else say "The Connection to the DB was successful!"
15
16  -- create Rexx object that will control the FXML set up
17  rxApp=.Formular~new
18  jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
19  jrxApp~launch(jrxApp~getClass, .nil)    -- launch the application, invokes "start"
20
21  EXIT 0
22
23  connectionError:
24  say "No Connection to the DB"
```

Snippet 18:Database Connection

To establish and interact with a SQLite database, a DatabaseHandler is created in a CSL file. The "*DatabaseHandler*" Rexx class defines various methods and attributes to facilitate access to the database.

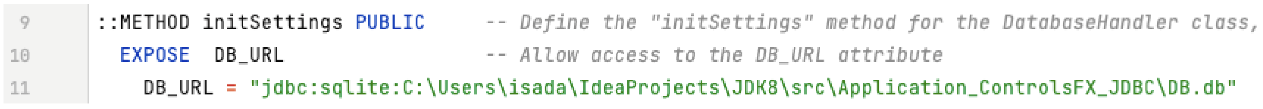
Initially, three attributes are defined: "*conn*", "*DB_URL*" and "*DriverManager*". The "*conn*" attribute stores the connection to the database, the "*DB_URL*" attribute stores the URL of the SQLite database and the "*DriverManager*" attribute is used to establish the connection to the database. The "*init*" method enables the import of the DriverManager object from the Java sql package class.



```
::CLASS DatabaseHandler PUBLIC
::METHOD conn ATTRIBUTE           -- Define the "conn" at
::METHOD DB_URL ATTRIBUTE        -- Define the "DB_URL" a
::METHOD DriverManager ATTRIBUTE    -- Define the "Drive
::METHOD init                     -- Define the "init" me
  EXPOSE DriverManager           -- Allow access to the D
  DriverManager = bsf.import("java.sql.DriverManager")
```

Snippet 19:DatabaseHandler Class

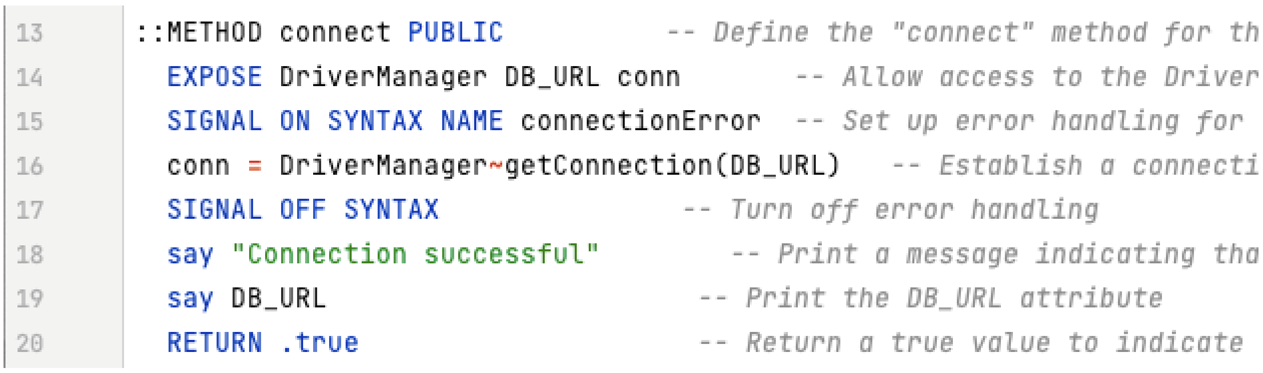
Next, the "initSettings" method is defined, which sets the DB_URL attribute to the path of the SQLite database.



```
::METHOD initSettings PUBLIC     -- Define the "initSettings" method for the DatabaseHandler class,
  EXPOSE  DB_URL                 -- Allow access to the DB_URL attribute
    DB_URL = "jdbc:sqlite:C:\Users\isada\IdeaProjects\JDK8\src\Application_ControlsFX_JDBC\DB.db"
```

Snippet 20:Method initSettings

To establish a connection to the database, the "*connect*" method is defined. The method calls the "*getConnection*" method of the DriverManager object to establish a connection to the database. If the connection is successfully established, the method outputs a success message and returns a value of "true". Otherwise, an error is handled and an error message is output.



```
::METHOD connect PUBLIC          -- Define the "connect" method for th
  EXPOSE DriverManager DB_URL conn       -- Allow access to the Driver
  SIGNAL ON SYNTAX NAME connectionError  -- Set up error handling for
  conn = DriverManager~getConnection(DB_URL)   -- Establish a connecti
  SIGNAL OFF SYNTAX                 -- Turn off error handling
  say "Connection successful"          -- Print a message indicating tha
  say DB_URL                          -- Print the DB_URL attribute
  RETURN .true                        -- Return a true value to indicate
```

Snippet 21:Method Connect

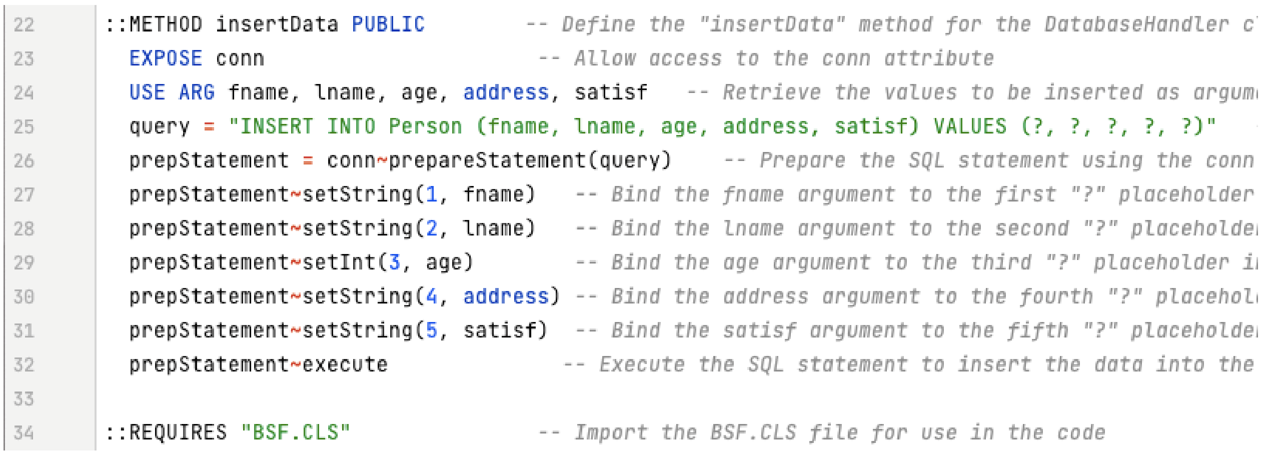
SQL can perform various operations on databases, including inserting, deleting, updating and querying data. The "INSERT INTO" command is used to insert data into a table, allowing records to be inserted into an existing table. To delete data from a table, the "DELETE FROM" command is used, which removes records from a table that match specified conditions. The "UPDATE" command is used to modify data in a table, allowing the updating of records within a table. To query data from a table, the "SELECT" command is used, allowing data to be retrieved from one or more tables that meet certain conditions.

In the nutshell example, the user fills out a form and the data are inserted into the SQLite database using the "*insertData*" method. The method accepts five arguments: "fname", "lname", "age", "address"and "satisf", which contain the values to be inserted into the database.

To insert the data, an SQL statement must first be defined. The statement in line 25 of the snippet 22 uses placeholders that will be replaced by the arguments during processing. The *PreparedStatement* object is then created by calling the "*prepareStatement*" method of the "*conn*" attribute, which is used to execute the SQL statement.

The arguments are then bound to the *PreparedStatement* object by calling the "*setString*" or "*setInt*" method to replace the corresponding placeholders with the argument values. Finally, the SQL statement is executed using the "*execute*" method of the *PreparedStatement* object to insert the data into the SQLite database.

The code also includes a line that imports the "BSF.CLS" file, which is required to use BSF (Bean Scripting Framework) in the code.



```
::METHOD insertData PUBLIC       -- Define the "insertData" method for the DatabaseHandler c
  EXPOSE conn                     -- Allow access to the conn attribute
  USE ARG fname, lname, age, address, satisf   -- Retrieve the values to be inserted as argum
  query = "INSERT INTO Person (fname, lname, age, address, satisf) VALUES (?, ?, ?, ?, ?)"
  prepStatement = conn~prepareStatement(query)    -- Prepare the SQL statement using the conn
  prepStatement~setString(1, fname)   -- Bind the fname argument to the first "?" placeholder
  prepStatement~setString(2, lname)   -- Bind the lname argument to the second "?" placeholde
  prepStatement~setInt(3, age)        -- Bind the age argument to the third "?" placeholder i
  prepStatement~setString(4, address) -- Bind the address argument to the fourth "?" placehol
  prepStatement~setString(5, satisf)  -- Bind the satisf argument to the fifth "?" placeholde
  prepStatement~execute              -- Execute the SQL statement to insert the data into the

::REQUIRES "BSF.CLS"              -- Import the BSF.CLS file for use in the code
```

Snippet 22:Method insertData

7. Conclusion

This bachelor thesis has investigated the potential of using ooRexx in combination with the JavaFX framework for developing graphical user interfaces. The study delved into the history and concepts of JavaFX, as well as the fundamental language concepts of ooRexx. Additionally, the thesis provided detailed instructions on how to install the necessary software for developing ooRexx and JavaFX applications, along with the required libraries.

The thesis demonstrated the ease and versatility of developing GUI applications with ooRexx and JavaFX through various examples, such as integrating the JFoenix styling library and ControlsFX widget library, using FXML and CSS for GUI design and utilizing JDBC for database connectivity. Overall, this thesis has shown that ooRexx is a viable option for creating GUI applications with JavaFX and its use can simplify the development process and enhance productivity.

8. References

Abstract Window Toolkit. (2022, Dezember 12). Retrieved Dezember 12, 2022, from Wikipedia.org: https://de.wikipedia.org/wiki/Abstract_Window_Toolkit

ApplicationStructure. (2022, Dezember 12). Retrieved Dezember 12, 2022, from NTU: https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html

Ashley , W., Flatscher, R., Hessling, M., McGuire, R., Miesfeld, M., Peedin, L., & Wolfers, J. (2010). *Open Object Rexx TM: Programming Guide.*

Ashley, W. D., Flatscher, R. G., Hessling, M., McGuire, R., Peedin, L., Sims, O., . . . Wolfers, J. (2022). *ooRexx Documentation 5.0.0 Open Object Rexx Programmer Guide.*

Ashley, W. D., Flatscher, R. G., Hessling, M., McGuire, R., Peedin, L., Sims, O., . . . Wolfers, J. (2022). *ooRexx Documentation 5.0.0 Open Object Rexx Reference.*

AWT. (2022, Dezember 12). Retrieved Dezember 12, 2022, from Betriebswirtschaft-lernen: https://www.betriebswirtschaft-lernen.net/erklaerung/abstract-window-toolkit-awt/

AWT vs Swing. (2022, Dezember 12). Retrieved Dezember 12, 2022, from Education-Wiki: https://de.education-wiki.com/1224717-awt-vs-swing

bsf4ooRexx. (2023, Februar 15). Retrieved Februar 15, 2023, from Sourceforge: https://sourceforge.net/projects/bsf4oorexx/

Classpath. (2023, Februar 15). Retrieved Februar 15, 2023, from Javatpoint: https://www.javatpoint.com/how-to-set-classpath-in-java

ControlsFx. (2023, Februar 13). Retrieved Februar 13, 2023, from GitHub: https://github.com/controlsfx/controlsfx

ControlsFX. (2023, Februar 15). Retrieved Februar 15, 2023, from MVN Repository: https://mvnrepository.com/artifact/org.controlsfx/controlsfx/8.40.18

Cowlishaw, M. (1990). *The REXX Language A Practical Approach to Programming.* Prentice Hall.

CSS. (2022, Dezember 13). Retrieved Dezember 13, 2022, from Wikipedia: https://de.wikipedia.org/wiki/Cascading_Style_Sheets

DB Browser. (2023, Februar 12). Retrieved Februar 12, 2023, from Heise: https://www.heise.de/download/product/db-browser-for-sqlite-41685

Flatscher, R. (2013). *Introduction to REXX and ooRexx From REXX to Open Object REXX (ooREXX).* Facultas .

Flatscher, R. (2023). *JavaFX for ooRexx – Creating Powerful Portable GUIs for ooRexx*. Retrieved from Rexxla: https://www.rexxla.org/presentations/2017/201711-ooRexx-JavaFX-Article.pdf

Flatscher, R. G. (2012). Automatisierung mit ooRexx und BSF4ooRexx. In *Proceedings der GMDS 2012 / Informatik 2012* (pp. 1-12). Braunschweig: Gesellschaft für Informatik, Bonn.

Flatscher, R. G. (2013). *Introduction to Rexx and ooRexx (coloured illustration): from Rexx to open object Rexx (ooRexx) (1. ed..).* Wien: Facultas Verl.- u. Buchhandels-AG.

Flatscher, R. G. (2019). *Flatscher, R. G., & Müller, G. (2019)ooRexx 5 Yielding Swiss Army Knife Usability.*

Fosdick, H. (2005). *Rexx Programmer's Reference.* Wiley Publishing, Inc.
Java-AWT. (2022, Dezember 12). Retrieved Dezember 12, 2022, from Javatpoint: https://www.javatpoint.com/java-awt

JavaFX. (2022, Dezember 14). Retrieved Dezember 14, 2022, from Wikipedia.org: https://de.wikipedia.org/wiki/JavaFX

JavaFX. (2022, Dezember). Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=JavaFX&oldid=947700994

JavaFX Architecture. (2022, Dezember 12). Retrieved Dezember 12, 2022, from Oracle: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#JFXST788

JavaFX_Application. (2022, Dezember 13). Retrieved Dezember 13, 2022, from Tutorialspoint: https://www.tutorialspoint.com/javafx/javafx_application.htm

JavaFX_CSS. (2023, Jänner 15). Retrieved Jänner 15, 2023, from JavatPoint: https://www.javatpoint.com/javafx-css

JDBC. (2023, Februar 15). Retrieved Februar 15, 2023, from Developer: https://www.developer.com/database/working-with-the-javafx-ui-and-jdbc-applications/

JFoenix. (2023, Februar 16). Retrieved Februar 16, 2023, from GitHub: https://github.com/sshahine/JFoenix

Learn JavaFX 17. (2023). In K. Sharan, & P. Späth, *Learn JavaFX 17 - Building User Experience and Interfaces with Java* (p. 851). Apress.

ooRexx. (2023, Januar 18). Retrieved Januar 18, 2023, from ooRexx: https://www.oorexx.org/about.html

ooRexx. (2023, Februar 15). Retrieved Februar 15, 2023, from Sourceforge: https://sourceforge.net/projects/oorexx/files/

OpenJDK. (2023, Februar 15). Retrieved Februar 15, 2023, from Bellsoft: https://bell-sw.com/pages/downloads/

Rexx. (2023, Januar 18). Retrieved Januar 18, 2023, from Wikipedia: https://en.wikipedia.org/wiki/Rexx#History

Scene Builder. (2022, Dezember 15). Retrieved Dezember 15, 2022, from RipTutorial: https://riptutorial.com/javafx/topic/5445/scene-builder

SceneBuilder. (2023, Februar 15). Retrieved Februar 15, 2023, from Gluon: https://gluonhq.com/products/scene-builder/#download

SQLite Browser. (2023, Februar 15). Retrieved Februar 15, 2023, from SQLite Browser: https://sqlitebrowser.org/dl/

SQLite-JDBC. (2023, Februar 2023). Retrieved Februar 15, 2023, from Github: https://github.com/xerial/sqlite-jdbc/releases

Stage. (2022, Dezember 15). Retrieved Dezember 2022, 2022, from Oracle: https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html

Swing. (2022, Dezember 12). Retrieved Dezember 12, 2022, from techopedia: https://www.techopedia.com/definition/26102/java-swing

What is JavaFX. (2022, Dezember 13). Retrieved Dezember 13, 2022, from Oracle-Patches: https://oracle-patches.com/en/coding/what-is-javafx

Appendix

Example Nr.	Name	Site number
Example 1	BMI Calculator	**65**
Example 2	BMI Calculator with FXML	**67**
Example 3	Health Calculator with JFoenix	**70**
Example 4	Formular with ControlsFX and JDBC	**76**

Example 1-BMI Calculator

The program is a body mass index (BMI) calculator implemented in the Rexx programming language, utilizing the JavaFX library to create a graphical user interface. It allows the user to input their weight and height to calculate and display their BMI and BMI category. The program's primary functions involve creating a user-friendly interface using JavaFX's layout nodes such as VBox, TextField, Label, and Button. It also involves implementing a JavaFX abstract method called "start" in the Rexx class, which initiates the primary process of the application and displays the window and its contents. Furthermore, the program utilizes BSF (Bean Scripting Framework) and RexxProxy to pass events from the JavaFX side to the Rexx side and to manage events such as clicking on the "Calculate BMI" button. Lastly, the program also implements the Rexx class "RexxButtonHandler," which performs the BMI calculation and displays the result on the user interface.

```
rxApp=.BMICalculator~new -- create an instance of the Rexx class

-- rxApp will be used for "javafx.application.Application"
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil) -- launch the application, invokes "start"

::requires "BSF.CLS" -- get Java support

-- Rexx class defines "javafx.application.Application" abstract method "start"
::class BMICalculator -- implements the abstract class "javafx.application.Application"
::method start -- Rexx method "start" implements the abstract method
use arg primaryStage -- fetch the primary stage (window)

primaryStage~setTitle("BMI Calculator") -- set the title of the window

root=.bsf~new("javafx.scene.layout.VBox") -- create the root node
root~prefHeight=400 -- set the preferred height of the VBox
```

```
root~prefWidth=400 -- set the preferred width of the VBox
root~setSpacing(20)  -- set the spacing between nodes in the VBox

-- create two text fields for user input
weightField=.bsf~new("javafx.scene.control.TextField")
weightField~setPromptText("50.0")
heightField=.bsf~new("javafx.scene.control.TextField")
heightField~setPromptText("1.60")

-- create a label for the weight field
weightLabel=.bsf~new("javafx.scene.control.Label")
weightLabel~text("Enter your weight (kg):")

-- create a label for the height field
heightLabel=.bsf~new("javafx.scene.control.Label")
heightLabel~text("Enter your height (m):")

-- create a button for performing the BMI calculation
calculateBtn=.bsf~new("javafx.scene.control.Button")
calculateBtn~text="Calculate BMI"

-- create a label for displaying the result
resultLabel=.bsf~new("javafx.scene.control.Label")

/* add the weight label, weight field, height label,
height field, calculate button, and result label to the VBox*/
root~getChildren~~add(weightLabel)~~add(weightField)~~add(heightLabel)
root~getChildren~~add(heightField)~~add(calculateBtn)~~add(resultLabel)
-- put the VBox on the stage
primaryStage~setScene(.bsf~new("javafx.scene.Scene", root))

primaryStage~show -- show the stage (window) with the scene

-- create a Rexx object to handle button presses
handler=.RexxButtonHandler~new(weightField, heightField, resultLabel)

-- create a Rexx proxy object to forward button events to the Rexx object
jrh=BSFCreateRexxProxy(handler, ,"javafx.event.EventHandler")

-- set the button's action handler to the Rexx proxy object
calculateBtn~setOnAction(jrh)

-- Rexx class which handles the button presses
::class RexxButtonHandler -- implements "javafx.event.EventHandler" interface
::method init -- Rexx constructor method
    expose weightField heightField resultLabel  -- allow direct access to ooRexx attribute
    use arg weightField, heightField, resultLabel -- save reference to javafx.scene.control.Label

::method handle -- will be invoked by the Java side when the button is pressed
expose weightField heightField resultLabel-- allow direct access to ooRexx attribute
use arg event, slotDir -- expected arguments

weightInput = weightField~getText -- get the value of the first text field
heightInput = heightField~getText -- get the value of the second text field

if weightInput="" | heightInput="" then do -- check if both fields have been filled out
    resultLabel~text = "Please enter the weight and height."
    return -- exit the method
    end

--Calculate BMI
weight = weightInput
height = heightInput
bmi = weight / (height * height)

--Determine BMI category
if bmi < 18.5 then
    category = "Underweight"
else if bmi < 25 then
    category = "Normalweight"
else
    category = "Overweight"
```

```
-- Output the result
resultLabel~text = "Your BMI is" bmi~format(, "0.00") "and you have " category "."
```

Listing 10: Example 1 - ooRexx_Gui.rexx

Example 2 -BMI Calculator with FXML

The BMI Calculator is a program created using JavaFX technology and the FXML format. It provides a simple user interface that allows the user to enter their weight and height. The program calculates the Body Mass Index (BMI) and also displays the corresponding BMI category (e.g. underweight, normal, overweight). The program uses CSS style sheets to customize the appearance of the user interface.

The stylesheet (Listing 11) contains CSS rules for formatting various elements in the user interface of the BMI calculator.

```
.root{
    -fx-background-color: #426357;
    -fx-font-family: "Arial";
    -fx-font-size: 15px;
}

.button {
    -fx-text-fill: #5F6362;
    -fx-font-weight: bold;
    -fx-color: #99E8CB
}

.label {
    -fx-font-weight: bold;
    -fx-text-fill: #DAE2DF;
}
.textfield{
    -fx-prompt-text-fill: #DAE2DF;
    -fx-text-fill: #5F6362;
}
```

Listing 11: Example 2- stylesheet.css

The gui file (Listing 12) is the FXML file of the BMI-Calculator. It defines how the frontend should look and which components are used in the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.VBox?>
<?language rexx?>

<VBox fx:id="root" alignment="TOP_CENTER" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="346.0" prefWidth="400.0" spacing="20.0"
styleClass="root" stylesheets="@stylesheet.css" xmlns="http://javafx.com/javafx/8.0.171"
xmlns:fx="http://javafx.com/fxml/1">
   <fx:script source="Controller.rexx" />
   <children>
      <Label fx:id="weightLabel" alignment="CENTER" prefHeight="41.0" prefWidth="180.0"
text="Enter your weight (kg):" />
      <TextField fx:id="weightField" alignment="CENTER" promptText="50.0" />
      <Label fx:id="heightLabel" alignment="CENTER" prefHeight="38.0" prefWidth="183.0"
text="Enter your height (m):" />
      <TextField fx:id="heightField" alignment="CENTER" promptText="1.60" />
      <Button fx:id="calculateBtn" mnemonicParsing="false" onAction="slotDir=arg(arg()); call
CalculateBMI slotDir;" text="Calculate BMI" />
      <Label fx:id="resultLabel" prefHeight="44.0" prefWidth="323.0" />
   </children>
</VBox>
```

Listing 12:Example 2 - Gui.fxml

The main functions of the program (Listing 13) include creating and showing the GUI, loading the FXML file using the FXMLLoader, and calculating the BMI using the RexxProxy and BSF framework. The program launches by invoking the "start" method in the Rexx class that implements the "javafx.application.Application" abstract class.

```
--Change directory to program location so that relatively addressed resources can be found
parse source   . . pgm
call directory filespec('L', pgm)

rxApp=.BMICalculator~new -- create Rexx object that will control the FXML set up
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil)  --Launch the application and invoke the "start" method

--Require the BSF and rxregexp classes for Java support
::requires "BSF.CLS"

--Define the Rexx class that implements the abstract class "javafx.application.Application"
::class BMICalculator

--Implement the "start" method to create and show the BMI Calculator GUI
::method start
  use arg primaryStage  -- fetch the primary stage (window)
  primaryStage~setTitle("BMI Calculator") -- Set the title of the primary stage

--Create an URL for the FMXLDocument.fxml file with the "file:" protocol
  fxmlUrl=.bsf~new("java.net.URL", "file:Gui.fxml")
--Use the FXMLLoader to load the FXML and create the GUI graph from its definitions
  rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
  scene=.bsf~new("javafx.scene.Scene", rootNode)    -- create a scene for the document
  primaryStage~setScene(scene)  -- Set the primary stage to the scene
  primaryStage~show             -- Show the primary stage and the scene
```

Listing 13: Example 2 - main.rexx

The code in Listing 14 contains a method named CalculateBMI which calculates the BMI based on the weight and height values entered by the user in the weightField and heightField text fields. It also determines the BMI category based on the calculated BMI and outputs the result in the resultLabel label.

```
::routine CalculateBMI public
  slotDir=arg(arg())  -- note: last argument is the slotDir argument from BSF4ooRexx

/* RexxScript annotation fetches "TextField" from ScriptContext
     and makes it available as the Rexx variable "TEXTFIELD":  */
  /* @get(weightField) */
  /* @get(heightField) */
  /* @get(resultLabel) */
weightInput = weightField~text -- get the value of the first text field
heightInput = heightField~text -- get the value of the second text field

if weightInput="" | heightInput="" then do -- check if both fields have been filled out
    resultLabel~text = "Please enter the weight and height."
    return -- exit the method
    end

    --Calculate BMI
    weight = weightInput
    height = heightInput
    bmi = weight / (height * height)


    --Determine BMI category
    if bmi < 18.5 then
        category = "Underweight"
    else if bmi < 25 then
        category = "Normalweight"
    else
        category = "Overweight"

-- Output the result
/* RexxScript annotation fetches "label" from ScriptContext
and makes it available as the Rexx variable "LABEL": */
/* @get(resultLabel) */
resultLabel~text = "Your BMI is " || bmi~format(, "0.00") || " and you have " || category || "."
```

Listing 14: Example 2 - controller.rexx

Example 3 – Health Calculator with JFoenix

The application is a health calculator that features a main menu with two buttons. The first button leads to the BMI calculator, and the second leads to the basal metabolic rate calculator. Each calculator opens in its own window, allowing users to input their data. This functionality enables users to calculate their BMI and basal metabolic rate.

The bmi_calculator file (Listing 15) is the FXML file of the BMI Calculator window. It defines how the frontend should look and which components are used in the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXSlider?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.effect.Glow?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?language rexx?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="400.0" prefWidth="600.0" stylesheets="@stylesheet.css"
xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1">
    <fx:script source="controller.rexx" />
   <children>
      <Label alignment="CENTER" layoutX="119.0" layoutY="42.0" prefHeight="64.0" prefWidth="363.0"
text="BMI - Calculator" textAlignment="CENTER">
         <font>
            <Font name="System Bold" size="43.0" />
         </font></Label>
      <JFXSlider fx:id="slider_weight" blockIncrement="200.0" indicatorPosition="RIGHT"
layoutX="70.0" layoutY="207.0" max="200.0" styleClass="jfx-slider-style"
stylesheets="@stylesheet.css" value="60.0">
         <effect>
            <Glow />
         </effect></JFXSlider>
      <Label fx:id="label_weight" alignment="CENTER" layoutX="70.0" layoutY="132.0"
prefHeight="21.0" prefWidth="175.0" text="Enter your weight (kg):" />
      <Label fx:id="label_height" alignment="CENTER" layoutX="326.0" layoutY="132.0"
prefHeight="21.0" prefWidth="158.0" text="Enter your height (m):" />
      <Label fx:id="resultLabel" alignment="CENTER" layoutX="225.0" layoutY="304.0"
prefHeight="48.0" prefWidth="322.0" />
      <JFXTextField fx:id="textField_height" layoutX="326.0" layoutY="179.0" prefHeight="42.0"
prefWidth="148.0" promptText="1.60" unFocusColor="#84b5a7">
         <effect>
            <Glow />
         </effect></JFXTextField>
      <JFXButton fx:id="btn_calculate" layoutX="44.0" layoutY="299.0"
onAction="slotDir=arg(arg()); call calculateBMI slotDir;" prefHeight="58.0" prefWidth="175.0"
stylesheets="@stylesheet.css" text="Calculate" />
   </children>

</AnchorPane>
```

Listing 15: Example 3 - bmi_calulator.fxml

The calorieRequirement file (Listing 16) is the FXML file of the calorie requirement calculator window. It defines how the frontend should look and which components are used in the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXSlider?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.effect.Glow?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?language rexx?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="400.0" prefWidth="600.0" styleClass="root"
stylesheets="@stylesheet.css" xmlns="http://javafx.com/javafx/8.0.171"
xmlns:fx="http://javafx.com/fxml/1">
    <fx:script source="controller.rexx" />
    <children>
        <Label alignment="CENTER" layoutX="70.0" layoutY="41.0" prefHeight="64.0"
prefWidth="461.0" stylesheets="@stylesheet.css" text="Calorie - Requirement"
textAlignment="CENTER">
            <font>
                <Font name="System Bold" size="43.0" />
            </font></Label>
        <JFXSlider fx:id="slider_weight" layoutX="43.0" layoutY="232.0" max="200.0"
stylesheets="@stylesheet.css" value="60.0">
            <effect>
                <Glow />
            </effect></JFXSlider>
        <Label fx:id="label_weight" alignment="CENTER" layoutX="33.0" layoutY="150.0"
prefHeight="21.0" prefWidth="175.0" text="Enter your weight (kg):" />
        <Label fx:id="label_height" alignment="CENTER" layoutX="235.0" layoutY="150.0"
prefHeight="21.0" prefWidth="158.0" text="Enter your height (m):" />
        <Label fx:id="resultLabel" alignment="CENTER" layoutX="225.0" layoutY="304.0"
prefHeight="48.0" prefWidth="322.0" />
        <JFXTextField fx:id="textField_height" layoutX="235.0" layoutY="199.0"
prefHeight="42.0" prefWidth="148.0" promptText="1.60">
            <effect>
                <Glow />
            </effect></JFXTextField>
        <JFXButton fx:id="btn_calculate" layoutX="45.0" layoutY="313.0"
onAction="slotDir=arg(arg()); call calculateCalorieRequirment slotDir;" prefHeight="56.0"
prefWidth="158.0" stylesheets="@stylesheet.css" text="Calculate" />
      <Label fx:id="label_age" alignment="CENTER" layoutX="407.0" layoutY="150.0"
prefHeight="21.0" prefWidth="140.0" text="Age:" />
      <JFXSlider fx:id="slider_age" layoutX="437.0" layoutY="232.0" value="25.0">
         <effect>
            <Glow />
         </effect>
      </JFXSlider>
    </children>

</AnchorPane>
```

Listing 16:Example 3 - calorieRequirement.fxml

The main window file (Listing 17) is the FXML file of the main window of the Health-Calculator. It defines how the frontend should look and which components are used in the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.effect.Glow?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?language rexx?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="284.0" prefWidth="507.0" stylesheets="@stylesheet.css"
xmlns="http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1">
   <fx:script source="controller.rexx" />
   <children>
      <Label alignment="CENTER" layoutX="30.0" layoutY="29.0" prefHeight="64.0"
prefWidth="447.0" styleClass="root" stylesheets="@stylesheet.css" text="Health Calculator">
         <font>
            <Font name="System Bold" size="43.0" />
         </font>
      </Label>
      <JFXButton fx:id="btn_bmi" alignment="CENTER" layoutX="30.0" layoutY="128.0"
onAction="slotDir=arg(arg()); call openBMICalculator slotDir;" prefHeight="80.0"
prefWidth="210.0" text="BMI Calculator" textAlignment="CENTER">
         <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
         </padding>
         <effect>
            <Glow />
         </effect>
      </JFXButton>
      <JFXButton fx:id="btn_calorie" alignment="CENTER" layoutX="267.0" layoutY="128.0"
onAction="slotDir=arg(arg()); call openCalorieRequirement slotDir;" prefHeight="80.0"
prefWidth="210.0" text="Calorie Requirement" textAlignment="CENTER">
         <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
         </padding>
         <effect>
            <Glow />
         </effect>
      </JFXButton>
   </children>
   <padding>
      <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
   </padding>
</AnchorPane>
```

Listing 17:Example 3 - main_window.fxml

The stylesheet (Listing 18) contains CSS rules for formatting various elements in the user interface, like the JFoenix slider.

```css
.root{
    -fx-background-color: #426357;
    -fx-font-family: "Arial";
}

.button {
    -fx-text-fill: #49635C;
    -fx-font-weight: bold;
    -fx-background-color: #84B5A7;
}

.button:active {
    -fx-color: #A9B0AD;
}

.label {
    -fx-font-weight: bold;
    -fx-text-fill: #84B5A7;
}
.jfx-text-field{
    -fx-prompt-text-fill: #84B5A7;
    -fx-text-fill: #49635C;
    -jfx-focus-color: #64E8C2;
    -jfx-unfocus-color: #84B5A7;
}



/* Styling the slider track */
.jfx-slider > .track {
     -fx-background-color: #84B5A7;
}

/* Styling the slider thumb */
.jfx-slider > .thumb {
     -fx-background-color: #84B5A7;
}

/* Styling the filled track */
.jfx-slider > .colored-track {
     -fx-background-color: #84B5A7;
}

/* Styling the animated thumb */
.jfx-slider > .animated-thumb {
     -fx-background-color: #84B5A7;
}

/* Styling the slider value text inside animated thumb */
.jfx-slider .slider-value {
     -fx-fill: #426357;
     -fx-stroke: #426357;
}
```

Listing 18: Example 3 - sylesheet.css

The code in Listing 19 involve setting up the application environment, defining the StageHandler class and its methods, and loading FXML files to create the UI components of the application.

```
/* Parse the source and get the full path */
PARSE SOURCE . . fullPath
CALL directory filespec('L', fullPath)

.environment~setEntry("my.app", .directory~new) --Set the environment variable 'my.app'
.my.app~homeDir = filespec('Location',fullPath) --Set the home directory of the application

stageHandler = .StageHandler~new --Create a new StageHandler object
.my.app~stageHandler = stageHandler --Set the stageHandler of the application
--Create a StageHandlerProxy object
stageHandlerProxy = BsfCreateRexxProxy(stageHandler,,"javafx.application.Application")
--Launch the application
stageHandlerProxy~launch(stageHandlerProxy~getClass, .nil)

/* Exit the program */
EXIT 0

/* Define the StageHandler class */
::CLASS StageHandler

/* Define the methods of the StageHandler class */
::METHOD stage ATTRIBUTE
::METHOD scene ATTRIBUTE
::METHOD windowStage ATTRIBUTE
::METHOD FXMLLoader
::METHOD init
  EXPOSE FXMLLoader

  FXMLLoader = bsf.import("javafx.fxml.FXMLLoader") --Import the FXMLLoader class

::METHOD start
  EXPOSE stage scene FXMLLoader
  USE ARG stage --Get the stage object

  stage~setTitle("Health Calculator") --Set the title of the stage

  /* Load the FXML file */
  url=.bsf~new("java.net.URL", "file:main_window.fxml")
  fxml = FXMLLoader~load(url)

  scene = .bsf~new("javafx.scene.Scene", fxml) --Create a new Scene
  stage~setScene(scene) --Set the Scene of the stage
  stage~show --Show the stage

::METHOD newWindow
  EXPOSE stage windowStage FXMLLoader
  USE ARG title, fileName --Get the title and file name

  windowStage = .bsf~new("javafx.stage.Stage") --Create a new window stage
  windowStage~setTitle(title) --Set the title of the window stage

  /* Load the FXML file */
  url =.bsf~new("java.net.URL", fileName)
  fxml = FXMLLoader~load(url)

  scene = .bsf~new("javafx.scene.Scene", fxml) --Create a new Scene
  windowStage~setScene(scene) --Set the Scene of the window stage
  windowStage~show --Show the window stage

::REQUIRES "BSF.CLS" -- get Java support
```

Listing 19:Example 3 - main.rexx

The code in Listing 20 includes several routines that perform different functions. These functions include opening windows for the BMI calculator and calorie requirement calculator, as well as calculating the user's BMI and daily calorie requirements based on their weight, height, and age inputs.

```
-- This routine opens the BMI calculator window
::routine openBMICalculator public
  use arg slotDir
  scriptContext=slotDir~scriptContext
  URL = "file:bmi_calculator.fxml"
  .my.app~stageHandler~newWindow("BMI-Calculator", URL)

-- This routine opens the calorie requirement window
::routine openCalorieRequirement public
  use arg slotDir
  scriptContext=slotDir~scriptContext
  URL = "file:calorieRequirement.fxml"
  .my.app~stageHandler~newWindow("Calorie Requirement", URL)

/*The routine calculates the BMI based on the weight and height values entered by the
user.*/
::routine calculateBMI public
  use arg slotDir

/* RexxScript annotation fetches "TextField" from ScriptContext
     and makes it available as the Rexx variable "TEXTFIELD":  */
  /* @get(slider_weight) */
  /* @get(textField_height) */
weightInput = slider_weight~getValue -- get the value of the first text field
heightInput = textField_height~text -- get the value of the second text field

if heightInput="" then do -- check if the height field have been filled out
    -- Output the result
        /* RexxScript annotation fetches "label" from ScriptContext
        and makes it available as the Rexx variable "LABEL": */
        /* @get(resultLabel) */
        resultLabel~text = "Please enter a height."
    return -- exit the method
    end

    --Calculate BMI
    weight = weightInput
    height = heightInput
    bmi = weight / (height * height)


    --Determine BMI category
    if bmi < 18.5 then
        category = "underweight"
    else if bmi < 25 then
        category = "normalweight"
    else
        category = "overweight"

    -- Output the result
    /* RexxScript annotation fetches "label" from ScriptContext
    and makes it available as the Rexx variable "LABEL": */
    /* @get(resultLabel) */
    resultLabel~text = "Your BMI is " || bmi~format(, "0.00") || " and you have " ||
category || "."

/*The routine calculates the daily calorie requirement based on the weight, height, and age
values entered by the user.*/
::routine calculateCalorieRequirment public
  use arg slotDir

/* RexxScript annotation fetches "TextField" from ScriptContext
     and makes it available as the Rexx variable "TEXTFIELD":  */
  /* @get(slider_weight) */
  /* @get(textField_height) */
  /* @get(slider_age) */
weightInput = slider_weight~getValue -- get the value of the first text field
heightInput = textField_height~text -- get the value of the second text field
ageInput = slider_age~getValue

if heightInput="" then do -- check if both fields have been filled out
    -- Output the result
        /* RexxScript annotation fetches "label" from ScriptContext
        and makes it available as the Rexx variable "LABEL": */
        /* @get(resultLabel) */
        resultLabel~text = "Please enter a height."
    return -- exit the method
    end

    --Calculate Calorie Requirement
    weight = weightInput
    height = heightInput*100
    age = ageInput

    calorieRequirment = 655.1 + (9.6 * weight) + (1.8 * height) - (4.7 * age)


    -- Output the result
    /* RexxScript annotation fetches "label" from ScriptContext
    and makes it available as the Rexx variable "LABEL": */
    /* @get(resultLabel) */
    resultLabel~text = "Your Calorie Requirment is " calorieRequirment~format(, "0") " kcal.
"

::REQUIRES "BSF.CLS"
```

Listing 20:Example 3 - controller.rexx

Example 4 – Formular with ControlsFX and JDBC

The program creates a GUI that contains a form, and the entered data are stored in an SQLite database.
The gui file (Listing 21) is the FXML file of the Formular. It defines how the frontend should look and which components are used in the GUI.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.effect.Glow?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import org.controlsfx.control.Rating?>
<?import org.controlsfx.control.textfield.CustomTextField?>
<?language rexx?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="546.0" prefWidth="429.0" styleClass="root"
stylesheets="@stylesheet.css" xmlns="http://javafx.com/javafx/8.0.171"
xmlns:fx="http://javafx.com/fxml/1">
<fx:script source="controller.rexx" />
<children>
```

```
      <Label alignment="CENTER" layoutX="50.0" layoutY="24.0" prefHeight="54.0"
prefWidth="330.0" text="Formular" textAlignment="CENTER">
         <font>
            <Font size="37.0" />
         </font>
      </Label>
      <CustomTextField fx:id="textfieldFName" layoutX="167.0" layoutY="135.0"
promptText="Firstname" styleClass="jfx-text-field" />
      <CustomTextField fx:id="textfieldLName" layoutX="167.0" layoutY="178.0"
promptText="Lastname" styleClass="jfx-text-field" />
      <CustomTextField fx:id="textfieldAddress" layoutX="167.0" layoutY="261.0"
promptText="Address" styleClass="jfx-text-field" />
      <JFXButton fx:id="btn_submit" buttonType="RAISED" layoutX="271.0" layoutY="460.0"
onAction="slotDir=arg(arg()); call InsertData slotDir;" prefHeight="54.0" prefWidth="129.0"
ripplerFill="#a9b0ad" text="Submit">
         <effect>
            <Glow />
         </effect></JFXButton>
      <Rating fx:id="rating" layoutX="167.0" layoutY="345.0" rating="3.0"
styleClass="rating" stylesheets="@stylesheet.css" />
      <Label fx:id="label_fname" layoutX="85.0" layoutY="140.0" text="Firstname:" />
      <Label fx:id="label_lname" layoutX="87.0" layoutY="183.0" text="Lastname:" />
      <Label fx:id="label_age" layoutX="106.0" layoutY="223.0" text="Age:" />
      <Label fx:id="label_address" layoutX="92.0" layoutY="266.0" text="Address:" />
      <Label fx:id="label_satisf" layoutX="74.0" layoutY="346.0" prefHeight="31.0"
prefWidth="91.0" text="Satisfaction:" />
      <CustomTextField fx:id="textfield_age" layoutX="167.0" layoutY="218.0" promptText="26"
styleClass="jfx-text-field" />
      <Label fx:id="resultLabel" layoutX="29.0" layoutY="466.0" prefHeight="42.0"
prefWidth="208.0" />
   </children>
</AnchorPane>
```

Listing 21:Example 4 - gui.fxml

The stylesheet (Listing 22) contains CSS rules for formatting various elements in the user interface.

```
.root {
    -fx-background-color: #426357;
    -fx-font-family: "Arial";
}

.button {
    -fx-text-fill: #49635C;
    -fx-font-weight: bold;
    -fx-background-color: #84B5A7;
}

.label {
    -fx-font-weight: bold;
    -fx-text-fill: #84B5A7;
}

.jfx-text-field {
    -fx-prompt-text-fill: #84B5A7;
    -fx-text-fill: #49635C;
    -jfx-focus-color: #64E8C2;
    -jfx-unfocus-color: #84B5A7;
}


 .rating > .container .button {
```

```
                -fx-pref-width: 35 ;
                -fx-pref-height: 35 ;
                -fx-background-size: cover;
                -fx-background-color:#426357;
                -fx-padding: 15;
 }
```

Listing 22:Example 4 - stylesheet.css

The code in Listing 23 is responsible for connecting to the database and creating the user interface.

```
/*change directory to program location such that relatively addressed resources can be
found*/
parse source  . . pgm
call directory filespec('L', pgm)    -- change to the directory where the program resides

-- set up application environment
.environment~setEntry("my.app", .directory~new)
.my.app~homeDir = filespec('Location',fullPath)
.my.app~dbh = .DatabaseHandler~new
.my.app~dbh~initSettings

-- connect to the database and handle connection errors
success = .my.app~dbh~connect
IF \success THEN CALL connectionError
else say "The Connection to the DB was successful!"

-- create Rexx object that will control the FXML set up
rxApp=.Formular~new
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil)     -- launch the application, invokes "start"

EXIT 0

connectionError:
say "No Connection to the DB"


::REQUIRES "DatabaseHandler.CLS"
::REQUIRES "BSF.CLS"


-- Rexx class defines "javafx.application.Application" abstract method "start"
::class Formular -- implements the abstract class "javafx.application.Application"

-- Rexx method "start" implements the abstract method
::method start
  use arg primaryStage   -- fetch the primary stage (window)
  primaryStage~setTitle("Formular")

  -- create an URL for the FMXLDocument.fxml file (hence the protocol "file:")
  fxmlUrl=.bsf~new("java.net.URL", "file:gui.fxml")
  -- use FXMLLoader to load the FXML and create the GUI graph from its definitions:
  rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)

  scene=.bsf~new("javafx.scene.Scene", rootNode)     -- create a scene for the document
  primaryStage~setScene(scene)   -- set the stage to the scene
  primaryStage~show                  -- show the stage (and thereby the scene)
```

Listing 23:Example 4 -main.rexx

The code in Listing 24 is responsible for inserting data into the database using the insertData method. It also ensures that all required fields are filled out before inserting the data into the database.

```
::routine insertData public
  use arg slotDir
  scriptContext=slotDir~scriptContext -- Get the slotDir entry

  /* Get the values of the textfields and rating component */
  /* @get(textfieldFName) */
  /* @get(textfieldLName) */
  /* @get(textfield_age) */
  /* @get(textfieldAddress) */
  /* @get(rating) */
  /* @get(resultLabel) */

  /* Assign the values to variables */
  fname = textfieldFName~text
  lname = textfieldLName~text
  age = textfield_age~text
  address = textfieldAddress~text
  ratingValue = rating~getRating

-- check if the fields have been filled out
  if fname="" | lname="" | age="" | address="" then do
    resultLabel~text = "Please fill out all fields."
    return -- exit the method
    end

  /* Call the insertData method of the DatabaseHandler instance */
  .my.app~dbh~insertData(fname,lname,age,address,ratingValue)


  /* Require the DatabaseHandler and BSF classes */
  ::REQUIRES "DatabaseHandler.CLS"
  ::REQUIRES "BSF.CLS"
```

Listing 24:Example 4 - controller.rexx

The code in Listing 23 defines and implements the DatabaseHandler class for connecting and interacting with the SQLite database. The class includes methods for initializing the database URL, establishing a connection to the database, and inserting data into the database using SQL statements.

```
::CLASS DatabaseHandler PUBLIC
-- Define the "conn" attribute for the DatabaseHandler class
::METHOD conn ATTRIBUTE
-- Define the "DB_URL" attribute for the DatabaseHandler class
::METHOD DB_URL ATTRIBUTE
-- Define the "DriverManager" attribute for the DatabaseHandler class
::METHOD DriverManager ATTRIBUTE
-- Define the "init" method for the DatabaseHandler class
::METHOD init
  EXPOSE DriverManager  -- Allow access to the DriverManager object
  -- Import the DriverManager object from the Java sql package using BSF
  DriverManager = bsf.import("java.sql.DriverManager")
/*Define the "initSettings" method for the DatabaseHandler class, which sets the DB_URL
attribute*/
::METHOD initSettings PUBLIC
  EXPOSE  DB_URL  -- Allow access to the DB_URL attribute
  -- Set the DB_URL attribute to the path of the SQLite database
```

```
    DB_URL = "jdbc:sqlite:C:\Users\isada\IdeaProjects\JDK8\src\Application_ControlsFX_JDBC\DB.db"

/*Define the "connect" method for the DatabaseHandler class, which establishes a connection to the
SQLite database*/
::METHOD connect PUBLIC
  EXPOSE DriverManager DB_URL conn /* Allow access to the DriverManager, DB_URL, and conn
attributes */
  SIGNAL ON SYNTAX NAME connectionError  -- Set up error handling for the connection
  conn = DriverManager~getConnection(DB_URL)  /* Establish a connection to the SQLite database
using the DB_URL attribute */
  SIGNAL OFF SYNTAX  -- Turn off error handling
  say "Connection successful" -- Print a message indicating that the connection was successful
  say DB_URL -- Print the DB_URL attribute
  RETURN .true -- Return a true value to indicate that the connection was successful

/* Define the "insertData" method for the DatabaseHandler class, which inserts data into the
SQLite database */
::METHOD insertData PUBLIC
  EXPOSE conn -- Allow access to the conn attribute
  -- Retrieve the values to be inserted as arguments
  USE ARG fname, lname, age, address, satisf
  -- Define the SQL query to insert the data
  query = "INSERT INTO Person (fname, lname, age, address, satisf) VALUES (?, ?, ?, ?, ?)"
  -- Prepare the SQL statement using the conn attribute
  prepStatement = conn~prepareStatement(query)
  -- Bind the fname argument to the first "?" placeholder in the query
  prepStatement~setString(1, fname)
  -- Bind the lname argument to the second "?" placeholder in the query
  prepStatement~setString(2, lname)
  -- Bind the age argument to the third "?" placeholder in the query
  prepStatement~setInt(3, age)
  -- Bind the address argument to the fourth "?" placeholder in the query
  prepStatement~setString(4, address)
  -- Bind the satisf argument to the fifth "?" placeholder in the query
  prepStatement~setString(5, satisf)
  -- Execute the SQL statement to insert the data into the SQLite database
  prepStatement~execute

::REQUIRES "BSF.CLS"                -- Import the BSF.CLS file for use in the code
```

Listing 25:Example 4 - DatabaseHandler.CSL