WIRTSCHAFTSUNIVERSITÄT WIEN

Vienna University of Economics and Business

Bachelor's Thesis

Titel of Bachelor's Thesis (english)	JDOR – An introduction to Java 2D's drawing classes with ooRexx and BSF4ooRexx
Titel of Bachelor's Thesis (german)	JDOR – Eine Einführung in Java 2D's Zeichenklassen mit ooRexx und BSF4ooRexx
Author (last name, first name):	Blauensteiner, Fabian
Student ID number:	H12021645
Degree program:	Bachelor of Science (WU), BSc (WU) Wirtschafts- und Sozialwissenschaften
Examiner (degree, first name, last name):	ao.Univ.Prof.Mag.Dr.rer.soc.oec 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.

26.02.2023
Date

Fabian Blauensteiner
Signature

Contents

Abstract 1

1. Introduction 2

1.1 Initial Situation 2

1.2 Goal of this Thesis 3

1.3 Structure of the Thesis 3

2. Used Languages and Frameworks 5

2.1 Open Object REXX (ooRexx) 5

2.1.1 Code Example for ooRexx 8

2.2 Java 10

2.2.1 Code Example for Java and Comparison to ooRexx 12

2.3 Bean Scripting Framework for ooRexx (BSF4ooRexx) 15

3. Used Versions 16

3.1 Open Object REXX 5.0 16

3.2 Java 1.8.0_352 17

3.3 BSF4ooRexx 850 17

4. Installation Guide and Troubleshooting 18

4.1 Installation of Open Object REXX 18

4.2 Installation of Java 19

4.3 Installation of BSF4ooRexx 21

4.4 Checking Installations Correctness 22

4.5 Classpath – Variable 23

5. Creating Graphics with Java 26

5.1 Abstract Windowing Toolkit (AWT) 26

5.2 Java 2D API 27

5.3 Java Drawing for ooRexx (JDOR) 30

5.3.1 JDOR – Commands 31

6. JDOR-Examples in Open Object REXX 37
6.1 Drawing Simple Shapes – JDOR_shapes.rxj 38
6.2 Different Strokes in JDOR – JDOR-strokes.rxj 40
6.3 Displaying Text in JDOR – JDOR-Strings.rxj 43
6.4 Using Images in JDOR – JDOR-images.rxj 45
6.5 Rotate, Scale, Translate and Shear in JDOR 48
6.6 Moving Objects in JDOR – JDOR-move.rxj 51
6.7 Combining JDOR with JavaFX – JDORFX.rxj 54
6.8 Creating Arbitrary Shapes with JDOR – JDOR-shapes.rxj 58
7. Conclusion 62
Appendix 64
A 1. JDOR-transform.rxj 64
A 2. JDOR-move.rxj 65
A 3. JDORFX.rxj 67
A 4. JDOR – shapes.rxj 73
References 75

List of Figures

Figure 1: Console output of the used version of Open Object REXX 16
Figure 2: Console output of the used version of Java.....17
Figure 3 ooRexx Installation Process 19
Figure 4: Java version from Bellsoft. 19
Figure 5: Java Installation Process.....20
Figure 6: Installation Steps for BSF4ooRexx.....21
Figure 7: ooRexxTry.rxj coding-tool.....22
Figure 8: BSF4ooRexx error message 23
Figure 9: Setting Classpath-Variable 24
Figure 10: Content of the classpath-variable 25
Figure 11: Coordinate system for User-Space in Java 27
Figure 12: Relation of graphics and methods (Oracle, o.D.-c).....29
Figure 13: Shapes in JDOR - JDOR_shapes.rxj.....38
Figure 14: Output of JDOR_shapes.rxj.....40
Figure 15: Strokes in JDOR - JDOR_strokes.rxj.....41
Figure 16: Output JDOR_strokes.rxj.....42
Figure 17: Writing in JDOR - JDOR-Strings.rxj.....43
Figure 18: Output JDOR-Strings.rxj.....45
Figure 19: Using images in JDOR - JDOR-images.rxj.....46
Figure 20: Snowman.png (left) and output of the code from JDOR-images.rxj (right).....47
Figure 21: Excerpt from JDOR - JDOR_transform.rxj (full code in Appendix - A 1. JDOR-transform.rxj) 49
Figure 22: Output form JDOR-transform.rxj.....50
Figure 23: JDOR-move.rxj (partly – complete version in Appendix - A 2. JDOR-move.rxj).....52
Figure 24: Output of JDOR-move.rxj after three seconds (left) and after two minutes (right) 53
Figure 25: Creating a JavaFX button in ooRexx – Excerpt from JDORFX.rxj.....54
Figure 26: Setting the EventHandler for the Draw-Button – excerpt from JDORFX.rxj.....54
Figure 27: Implemeting the logic of an EventHandler - excerpt from JDORFX.rxj.....55

Figure 28: GUI created by JDORFX.rxj 56
Figure 29: Creating a movement effect - excerpt from JDORFX.rxj 56
Figure 30: Drawings created by JAVA-FX GUI interaction 57
Figure 31: Creating shapes - JDOR-shapes.rxj (excerpt – full code in Appendix in A 4. JDOR – shapes.rxj) 59
Figure 32: Banner created by JDOR-shapes.rxj 61
Figure 33: JDOR-transform.rxj.....64
Figure 34: JDOR-move.rxj 66
Figure 35: JDORFX.rxj 72
Figure 36: JDOR-shapes.rxj 74

List of Tables

Table 1: Comparison between procedural and object-oriented programming (geeksforgeeks, 2022; javatpoint.com, o.D.) 7
Table 2: JDOR-commands and their Java2D - counterparts 36

Abstract

This bachelor thesis demonstrates how the most recent BSF4ooRexx 850 extension, JDOR, can be utilized in ooRexx to generate various images.

For this purpose, "Nutshell-Examples" are presented to illustrate the fundamental operations and how they are carried out. Additionally, a detailed description of the required software components and an installation manual are included.

ooRexx is a further development of the successor of IBMs mainframe languages EXEC and EXEC 2. With the help of the BSF4ooRexx 850 framework, the high functionality of Java can be combined with the simple-to-read syntax of ooRexx to provide access to Java functionalities via ooRexx.

Through JDOR and an underlying interest in simple graphics design, even programmers with limited knowledge in ooRexx and Java can produce sophisticated drawings.

1. Introduction

This chapter gives a short overview of the topic of this bachelor thesis, the related research question and the overall goal of this thesis. Additionally, it provides the methodological approach to write this paper and the main structure of the thesis.

1.1 Initial Situation

“Java” is one of the most popular programming languages today, ranking at number two in the PYPL-Index, whose ranking is based on the number of searches for tutorials on Google (GitHub.io, 2022). It also ranks at number four in the TIOBE-index (taking into account the number of programmers and search-engine usage) and at number three in the Developer Nation index (ranking created through surveys of active developers) (SlashData, 2022; TIBOE, 2023) . Even though Java is one of the most prominent languages, its’ structure and syntax is often, especially for beginners, hard to understand.

Therefore, the courses “Business Programming I” and “Business Programming II” from Professor Rony G. Flatscher at the “Vienna University of Business and Economics” focus on the programming language “ooRexx”, which is a more beginner-friendly and easier to understand language for a layman. Throughout these two courses students learn the basics of programming and how to use the “ooRexx” language to automate certain processes through the “OLE-connection” (=Object Linking and Embedding) of the Microsoft ecosystem.

In the second course students learn how to use objects and functions from Java in ooRexx through the BSF4ooRexx-Framework. It allows the programmer to interact with the Java language while using the syntax and structure from ooRexx.

With the latest instalment of ooRexx and BSF4ooRexx 850 the capabilities of this combination increased even more, including a broader collection of usable functions from Java.

The author of this paper thinks that GUI-elements (=Graphical User Interface) are, apart from the general output-console of most programming languages, an interesting way to show beginners certain mechanisms of these languages. They often invite the programmer to try out new things and play around with these elements. Therefore the

author chose the newly integrated “JDOR”-package (=Java Drawing for ooRexx) for this thesis.

1.2 Goal of this Thesis

This bachelor's thesis' main objective is to program, demonstrate, and explain a few examples using the JDOR-package. Drawings made using the JDOR-package utilize Java's "`Graphics`" and "`Graphics2D`" libraries. JDOR can be used to produce extremely complex drawings with a range of distinct aspects for a variety of use-cases.

The purpose of this thesis is not to demonstrate these incredibly complex drawings, but rather to introduce this broad subject by making specific remarks and explanations on how to use some of the items and functions.

The research question, which this thesis aims to address, is whether JDOR is a useful addition for students learning ooRexx, how the students should learn it, and if it is beneficial to give an additional lesson on this subject.

This bachelor thesis should provide the reader with the basic information of the used programming languages and frameworks. Furthermore, it gives an introduction of the ooRexx programming language and how its’ syntax and structure work.

After reading this thesis the reader should be able to install the needed languages and frameworks, program simple ooRexx-Code and use the JDOR-package to create simple drawings.

1.3 Structure of the Thesis

This bachelor's thesis can be divided into two main sections. The thesis' first section takes a more theoretical approach and is more closely akin to a literature review. To develop this theoretical overview of the subject, several websites and publications were consulted. The sources can be found in the chapter ***References.***

The second section deals with the actual programming. To demonstrate the many aspects of the JDOR package, the author offers specific code samples and their generated output. For this, the author used his previous knowledge of Java and ooRexx and combines it with the newly gained information from the research for the theoretical part of this thesis building a link between those two major parts.

In the following chapter the different programming languages used for this thesis are introduced and compared. ***Chapter 3*** will provide a step-by-step installation guide for all the needed languages and packages as well as a short sub-chapter on some installation problems the author experienced during his own installation process and how they were solved.

In ***chapter 4*** the different packages and libraries that were used in the programming part are discussed in more detail, giving an overview of the underlying structure.

Chapter 5 deals in detail with the Java-2D library, giving overall information and history of the library and providing a comprehensive description of some of the functions used in the code-examples showing the different possible attributes that were and could be used while coding.

Chapter 6 shows the different code-examples and their outputs. It should provide basic knowledge and information about the structure and functionality of the provided examples.

Finally, in the ***last chapter*** the most important information of this thesis will be summarized, and the research question will be answered.

2. Used Languages and Frameworks

To code and execute the different example-programs presented in this bachelor-thesis three main software components need to be installed on the system. These components are the two programming languages “ooRexx” and “Java” as well as the “Bean Scripting Framework for ooRexx” (BSF4ooRexx) extension. This chapter provides more information about these components.

2.1 Open Object REXX (ooRexx)

The programming language Open Object REXX (ooRexx) is the successor and open-source Version of the Object REXX programming language developed by IBM, which itself is the successor of the language REXX. REXX was developed in 1979 by Mike F. Cowlishaw and IBMs’ research facilities (Flatscher, 2013).

The goal of REXX was to create a programming language which is easier to understand for humans that replaces the cryptic IBM-Mainframe languages “EXEC” and “EXEC 2” (Flatscher, 2013). REXX later became the standard batch and scripting language for the SAA (System Application Architecture) for all of IBMs’ operating systems. In the 1980s REXX became increasingly popular even outside IBM, which lead to the development of many different REXX-interpreters (Flatscher, 2013).

At the end of the 1980s and the beginning of the 1990s IBMs’ researchers created the next milestone in the REXX history by developing a REXX-interpreter which extends the REXX language with an object-oriented feature (Flatscher, 2013). This prototype would later become Object REXX and was first distributed in 1997. Different versions of this interpreter were created to be used on the operating system AIX from IBM and Microsoft’s Windows (Flatscher, 2013).

After lengthy negotiations between IBM and the RexxLA, the source code of Object REXX was given to the RexxLA in 2004, which committed itself to its further development. Open Object Rexx is the open source version of Object REXX (Flatscher, 2013).

Since 2004 RexxLA continuously developed new versions of ooRexx. In 2009 ooRexx 4.0 was released providing the user with a newly written kernel which allows porting to virtually all operating systems (Flatscher, 2017).

The most recent version of ooRexx, ooRexx 5.0, was published in December 2022 after spending the previous five years in beta. A number of the fundamental ooRexx 4.0 functionalities have been updated, and new components have been introduced (Flatscher, 2017). However, the basic principles of the simple structure and easy readability of REXX have not been forgotten.

ooRexx drew a lot of inspiration from other object-orientated programming languages. One prime example in this case is “Smalltalk” (Flatscher, 2017). For example, variables have a dynamic type, which means that their type can change throughout the program and all values are seen as objects (Flatscher, 2017).

ooRexx generally perceives each value as a string. The interpreter automatically changes variables to numeric values when they are coupled with mathematical symbols like the plus sign. On the one hand, because the interpreter does part of the logical work, this is especially beneficial for beginners who are not as familiar with the various data types. On the other hand, this might cause some issues for programmers who have exclusively worked with strictly typed programming languages, like Java, up until that point.

ooRexx offers a structure and syntax which is more in line with the English language, making it easier for layman to read and understand the code. The use of a smaller number of built-in functions helps beginners to get a quick overview of the whole system.

Especially for programmers who already have some experience with object-orientated languages, ooRexx will seem very familiar. The use of classes and methods is straight-forward and after a short period of familiarization with its’ syntax it is quickly possible to write more complicated programs. This of course also goes the other way around, making it a good programming language for learning the basics of programming. This is probably one of the reasons why Prof. Dr. Rony G. Flatscher chose ooRexx as the programming language for his two “Business Programming” courses.

Through its’ interpreter ooRexx programs can be written in a procedural and object-oriented way, making it a good starting point for learning the differences of the two big programming language categories.

It can be debated which programming style is better, yet most modern and popular programming languages, like “Java”, “Python” or “C#”, are based on an object-orientated approach making it possible to code in a procedural style as well as an object-orientated style (geeksforgeeks, 2022). Table 1 shows some of the differences between the procedural approach and the object-oriented approach. It can be said that object-oriented programming is based on objects from the real world and is capable to solve more complex problems (geeksforgeeks, 2022; javatpoint.com, o.D.)

	Procedural-Oriented Programming	**Object-Oriented Programming**
Definition:	Program is based on the unreal world and divided into functions	Program is based on the real world and divided into objects
Approach	Follows a top-down approach	Follows a bottom-up approach
Data movements	Data moves freely within the system	Objects communicate with each other over functions
Inheritance	No concept of inheritance an no code reusability	Inheritance is present and features code reusability
Complexity	Not appropriate for complex problems	Appropriate for complex problems
Data hiding	Features no possibility of data hiding	Features a possibility of data hiding
Examples	Fortran, Pascal, C, VB	C#, Java, Python, C++, ooRexx

Table 1: Comparison between procedural and object-oriented programming (geeksforgeeks, 2022; javatpoint.com, o.D.)

2.1.1 Code Example for ooRexx

To give a better picture of the structure and syntax of ooRexx, Code 1 shows a short and simple program that uses some of the basic elements of the programming language. The program creates the factorial of different numbers using a recursive function. A recursive function is a function which calls itself.

Throughout the years new versions of ooRexx were released which brought new ways of declaring certain variables. One of these instances is the declaration of arrays. An array is a structure which holds multiple elements of the same data type. Before ooRexx 5.0 an array must be created like it is shown in line 2 in Code 1. Since ooRexx 5.0 it is possible to use the easier to read method which is shown in line 5. The code in line 5 overwrites the “numbers” array declared in line 2.

In most programming languages an array must be declared with a fixed number of elements, which is needed in order to reserve the storage space on the system. In ooRexx an array is of dynamic size, which allows adding new elements after the declaration of the array. Line 7 in Code 1 shows how a new element is added at the index 8 to the existing “`numbers`”-array.

Outputs in ooRexx can easily be created by the “`SAY`” keyword. The interpreter automatically detects which data type is needed for the used variable making string-parsing operations for numeric values for the outputs obsolete.

Line 11 shows the use of a loop. Most often loops use a certain variable which is increased incrementally for each pass. In this case the variable ‘i’ starts at ‘1’ and increases incrementally till it reaches the size of the array (in this case ‘8’). In each pass the “`FACTOR`”-function is called which passes an element of the array to the function. If ‘i’ is ‘6’ it would pass the sixth element to the function. Each loop must be closed by the “`END`”-keyword.

Functions are a part of a programming code which can be used multiple times. Line 18 in Code 1 shows the creation of such a function. Through the keyword “`PROCEDURE`” the function becomes encapsulated, which gives the function a local scope, meaning that the function cannot access variables from the main program. Each element from

the main program must be passed to the function-code as an argument when calling the function.

Line 21 shows a simple IF-THEN-ELSE structure. If the passed argument is equal to '1' the function will return '1' to the main code. Otherwise, the interpreter will execute the code-block followed by the "ELSE"-keyword. This is the most complex part of Code 1 since the function calls itself again, making it a recursive function.

```
1  /*code for array initalization in ooRexx 4.0 and lower*/
2  numbers = .array~of(1,2,3,4,5,6,7)
3
4  /*new possibility in ooRexx 5.0*/
5  numbers = 1,2,3,4,5,6,7
6
7  numbers[8] = 8
8
9  say "Size of the array" numbers~SIZE
10
11 DO i=1 TO numbers~SIZE
12 factorial = FACTOR( numbers[i])
13 SAY "The factor of" numbers[i] "is" factorial
14 END
15
16 PARSE PULL input
17 EXIT
18
19 FACTOR: PROCEDURE
20 PARSE ARG num
21 IF num = 1 THEN
22 RETURN 1
23 ELSE
24 RETURN num * FACTOR( num - 1 )
```

Code 1: Example Code for factoring in ooRexx

As an example, at the third element of 'numbers', which is the number '3' will be passed to the function as the variable 'num'. Since '3' is unequal to '1' the interpreter will execute the ELSE block, which results in the calculation '3 * FACTOR (3-1)'. The FACTOR function will be called again passing the argument '2', which again is not '1', calling the FACTOR-function again with the argument '2-1'.

Since '2-1' is '1' the function will only return '1', resulting in the final calculations of:

1. FACTOR (3) = 3 * FACTOR (3-1)
2. FACTOR (2) = 2 * FACTOR (2-1)
3. FACTOR (1) = 1
4. -> **3 * 2 * 1 = 6**

Output 1 shows the output resulting from Code 1. First the size of the array was displayed by line 9 in Code 1. After that the different factors were displayed by the loop. Through the calculation done for the number ‘3’ it shows that the program gives the correct result for the factor of ‘3’.

```
Size of the array: 8

The factor of 1 is 1
The factor of 2 is 2
The factor of 3 is 6
The factor of 4 is 24
The factor of 5 is 120
The factor of 6 is 720
```

Output 1: Output from Code 1

2.2 Java

Java is an object-oriented programming language like ooRexx. Java was created by Sun Microsystems’ James Gosling in 1991 and released in 1995. The idea was to develop a language for digital devises for television (Bahtnagar, 2022). The first plan was to use “C++” for the projects code but later was rejected since “C++” had some issues, like the increasing memory use compared to other languages. After some thoughts James Gosling decided to create a new programming language which has a syntax similar to C++ (Britannica, 2022).

After many different names, like “Greentalk” and “Oak”, Java was chosen as the final name for the project, which name is based on a type of espresso bean (Bahtnagar, 2022). After some work, the projects focus shifted to the World Wide Web and its’ premise to link many devices and to run everywhere on each device (Britannica, 2022). In contrary to many other programming languages Java’s compiler converts the code into bytecode, which is than interpreted by the Java Runtime Environment (JRE) or the Java Virtual Machine (JVM) (Britannica, 2022). The JRE thereby acts like a virtual computer which interprets the bytecode and translates it for the host computer (Britannica, 2022).

Because of the JRE as this second layer, code in Java can be written the same way on many different platforms (“Write once, run anywhere”) – this helped to increase the popularity, since most users work on different systems (Britannica, 2022).

Java uses a “Just in Time Compiler” (JIT), which is responsible for performance optimizations at runtime. Javas JIT compiler interacts with the Java Virtual Machine at runtime and compiles certain bytecode sequences into native machine code (Sharma, 2022). In comparison to having the JVM interpret the bytecode repeatedly, the JIT compiler allows the hardware to execute the native code, which leads to a performance gain (Sharma, 2022). This performance gain is only visible when multiple code sequences are used multiple times since then the code will be interpreted and compiled to machine code once and further executions will use the pre-compiled version (Brihadiswaren, 2020)

Java often has this stigma of being a slower language compared to other programming languages, like ‘C’ and C++’ (Brihadiswaren, 2020). While this statement is not completely wrong, Java’s JIT-compiler performs optimization processes automatically which greatly increases its’ performance. In languages like ‘C’ these optimization flags need to be set explicitly which often result into a faster execution then Java (Brihadiswaren, 2020).

In most other programming languages a compiler first needs to translate the code into instructions for certain specific systems (Britannica, 2022).

Java primarily works with five principles:

- Robustness
- Portability
- Platform Independence
- High Performance
- Multithreading (Bahtnagar, 2022)

Since the release of version 1.0 in the beginning of 1996 many changes were made to the programming language itself and its’ corresponding platforms, increasing its’ features and use-cases. In the following years Java was increasingly used in more systems – personal and professional.

It was even used on NASA's Mars exploration rovers, showing its' versatility (Britannica, 2022). Due do Javas popularity Sun Microsystems released different versions for different purposes, like Java SE for home computers, Java ME for embedded devices and Java EE for internet servers and supercomputers (Britannica, 2022).

In 2010 Oracle took over Javas development when it acquired Sun Microsystems (Britannica, 2022). Currently Java is used as a programming language in internet programming, mobile devises, business solution, games, etc. (Bahtnagar, 2022). The current version of Java, "Java 19", was released in September 2022 and the next version "Java 20" is set to release in March of 2023.

2.2.1 Code Example for Java and Comparison to ooRexx

To show the structure and syntax of Javas code, especially in comparison to ooRexx codes, Code 2 depicts Java's solution to the same factoring problem solved by ooRexx in Code 1.

To give a better comparison the code in Code 2 is built the same way as ooRexxs code in Code 1. This is done to give a better understanding of the similarities and differences between the two programming languages. In reality, Java provides many different functions and structures which could greatly decrease the number of lines of code needed to solve the problem.

For the author one of the most striking differences to Open Object REXX is the use of curly brackets to indicate the start and end of a block or section and semicolons at each end of a line. Another conspicuous feature is the use of strictly typed variables since each datatype of the variable needs to be specified at the declaration of the variable.

Another prominent feature is the division into classes and functions. The use of at least one class with one function (in most simple cases this would be the "main" class and the corresponding "main" function) is mandatory. In comparison, ooRexx can be used with classes and functions but they are not necessarily needed (see Code 1).

Java has three levels of different scopes (accessibility for variables, classes and methods):

- Class Scope (instance variables):
  - Variables which are declared within a certain class (yet outside of the methods) are accessible by all methods in the class. This can be seen in line 6 in Code 2.
- Method Scope (local variables):
  - Variables declared within a method can only be accessed by this method. An example of this is the variable ‘num’ declared in line 25 in Code 2.
- Block Scope (loop variables):
  - If a variable is declared in a for-loop condition it can only be accesses within this loop (see variable ‘i’ on line 19 in Code 2)
    (Codeacademy, o.D.)

The program starts with the declaration of the array ‘`numbers`’ with the ‘`int`’-datatype on line 6 in Code 2. Like in most programming languages an array in Java has a fixed length making it impossible to add other elements afters its’ declaration. Therefore, the command on line 10 would cause an error since the array only has seven elements. To solve this problem, an ‘`ArrayList`’ is used on line 13, which is similar to ooRexxs Array. To add a new element to this ArrayList the add-function of the ArrayList is used.

To write to the console the “`System.out.println()`” keyword is needed. Unlike ooRexx, texts must be in quotation marks and variables must be linked with the ‘+’ sign.

To loop through the array a for-loop is used on line 19 which does the same as a “`DO`” loop in ooRexx, where the variable ‘i’ is created which iteratively takes values starting from ‘1’ to the size of the ArrayList. In Java other loop variants, like a for-each-loop, could be used to iterate through an array-like structure.

In the loop the FACTOR-method with the current element of the ArrayList as an argument is called on line 21. The method is declared on line 25 and uses the 'public' keyword, which would make it possible for other classes to access the method. The method is built similar to the FACTOR-method in ooRexx in Code 1 using the variable 'num' and return '1' if 'num' is equal to '1' or else call the method again.

```
1   import java.util.ArrayList;
2   import java.util.Arrays;
3   public class Main
4   {
5     //Array has a fixed length
6     public int[] numbers = {1,2,3,4,5,6,7};
7
8     public static void main(String[] args)
9     {
10      //numbers[8] = 8; -> would cause an error -> OutofRange
11
12      //ArrayList has a dynamic length
13      ArrayList<Integer> numbers2 = new ArrayList<Integer>();
14      numbers2.addAll(Arrays.asList(1,2,3,4,5,6,7));
15      numbers2.add(8);
16
17      System.out.println("Size of the array" + numbers2.size());
18
19      for (int i=0; i< numbers2.size();i++)
20      {
21          System.out.println("The factor of "+i+" is "+FACTOR(numbers2.get(i)));
22      }
23
24    }
25    public static int FACTOR(var num)
26    {
27      if(num == 1){
28          return 1;
29      }
30      else{
31          return (num * FACTOR(num - 1));
32      }
33    }
34  }
```

Code 2: Example Code for factoring in Java

The code creates the same output as Code 1, which can be seen in Output 1. In comparison, ooRexx's code is easier to read then Java's, yet for some programmers who are already familiar with languages like 'C++' Java's code-structure is also easy to get used to.

2.3 Bean Scripting Framework for ooRexx (BSF4ooRexx)

The Bean Scripting Framework for ooRexx allows interaction between ooRexx programs and Java over the JNI (Java Native Interface) for Windows, macOS and Linux (Flatscher, 2017). It provides programmers with the possibility to implement (abstract) Java methods and other Java functionalities in ooRexx.

BSF4ooRexx was created in Austria and Germany in the year 2000 after IBM decided to no longer support its' OS/2 system (Flatscher, 2012). It uses Javas open-source class-library Bean Scripting Framework (BSF) which allows to implement other script-languages in Java. With the release of ooRexx 4.0 and its' interface to C++ it was possible to implement callbacks from Java to ooRexx (Flatscher, 2012, 2021).

BSF4ooRexx is an external ooRexx function packages, which allows ooRexx to access external functions, even though they are written in C++. For an easier understanding for ooRexx programmers, who are not familiar with Java, the different functions were masked to create the impression that the used Java methods are in fact ooRexx methods (Flatscher, 2012).

Using the BSF4ooRexx package provides the programmers with serval advantages:

- No need for further external function packages if the needed functions are already present in the Java runtime environment
- It allows the use of different Java classes and JRE classes regardless of the operating system that is used.
- It is possible to use Java-classes which use abstract method or expect arguments from different interface-classes.
- If a system has a Java-interface it can be accessed over Open Object REXX
- It is possible to start ooRexx-scripts through Java
  (Flatscher, 2012, 2021)

3. Used Versions

Specific versions of the programming languages and frameworks used are required to recreate the outputs of the programs supplied in the following chapters. Otherwise, the code will either not run at all or not correctly.

Since many of the earlier versions lack specific capabilities, it is extremely important to use the proper versions. Even though the code has only been tested with the versions listed below, using a newer version shouldn't present any issues.

It should also be mentioned that if versions of the programming languages are used that are not those listed below it is imperative that the same architecture version is used (e.g. only use the combination of 32-bit ooRexx with 32-bit Java – the same goes for the 64-bit version).

3.1 Open Object REXX 5.0

Open Object REXX 5.0 is the newest version from ooRexx. It can be downloaded from the website of the REXX language association (https://www.rexxla.org) or from their file-share-portal on SourceForge (https://sourceforge.net/projects/ooRexx/files/ooRexx/5.0.0/). It is imperative to choose the correct version of ooRexx for the used system.

For the programming part of this bachelor thesis a Windows Laptop and the 64-bit version of ooRexx were used. ooRexx is available in a 32-bit and 64-bit version. Using the 32-bit version should not make a difference if the corresponding Java version is used.

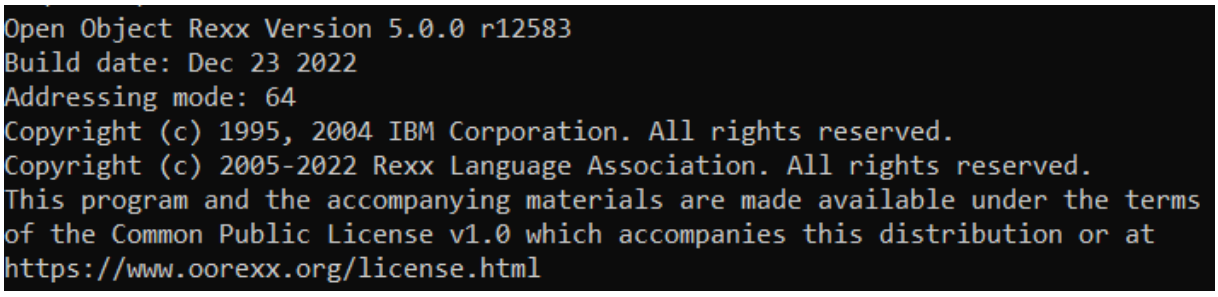

Figure 1: Console output of the used version of Open Object REXX

3.2 Java 1.8.0_352

Even though “Java 8” is not the newest version of Java it still includes all the necessary packages for the project. As of today, “Java 8” is still maintained and should still receive updates in the coming years. Oracle will support “Java 8” through their extended support period till December 2030 (Oracle, 2022). The premier support ended in March 2022 which included regular updates as well as technical support (Oracle, 2022).

The version of Java used for this bachelor thesis is the “Liberica FULL JDK 8 x86 64”-package and can be downloaded through the website of Bellsoft (https://bell-sw.com/pages/downloads/#downloads). Other versions of Java for other operating systems can be accessed through the same link.

```
openjdk version "1.8.0_352"
OpenJDK Runtime Environment (build 1.8.0_352-b08)
OpenJDK 64-Bit Server VM (build 25.352-b08, mixed mode)
```

Figure 2: Console output of the used version of Java

3.3 BSF4ooRexx 850

Version 850 of the BSF4ooRexx is the newest version and successor of BSF4ooRexx 641. While the earlier version of BSF4ooRexx works with “Java 6” and “ooRexx 4.1” or later, version 850 needs at least “Java 8” and “ooRexx 5.0”. This is mainly due to the additional features these two versions bring with them compared to their older versions (Flatscher, 2022a).

BSF4ooRexx 850 is based on the older 641 version and brings along many changes related to “Java 8” and “ooRexx 5.0”. It is needed to start the programs that are presented in this bachelor thesis, since the JDOR-package is only included in the newest version. The JDOR-package makes it possible to use the Java-2D library in ooRexx.

The specific version of BSF4ooRexx 850 used in this project is the version from the 23.12.2022. It can be downloaded on SourceForges’ website (https://sourceforge.net/projects/BSF4ooRexx/files/)

4. Installation Guide and Troubleshooting

In the following chapter an installation guide for the different programming languages and frameworks is presented. While the two programming languages can be installed through different installers (program which leads the user through the installation process) the BSF4ooRexx framework needs a little bit more work.

Even if not absolutely necessary, it is advisable to carry out the installations in the order mentioned here. For the two programming languages, the order would be irrelevant, however the BSF4ooRexx framework has to be installed last.

After the introduction to the installation guide, some problems, the author faced during his installation process are mentioned and possible solutions are provided.

4.1 Installation of Open Object REXX

The installation of ooRexx is easy and can be done through an installer. The download can be started through the following link from SourceForge:

https://sourceforge.net/projects/ooRexx/files/ooRexx/5.0.0/ooRexx-5.0.0-12583.windows.x86_64.exe/download

This link will automatically open the download website where the needed files will be downloaded after a short timer. The download only consists of one ".exe"-file (executable file)

If a previous version of ooRexx is already installed on the system, the installer should be able to detect it. In the process of the installation, the installer will give the user the option to deinstall any previous version of ooRexx. It is advised to deinstall any previous versions since certain problems could occur if two different versions are installed on the system.

During the installation no additional checkboxes were marked throughout the installation process, so no extra settings were enabled or disabled. Some of the installation steps can be seen in Figure 3. The steps of the top two pictures will only happen if a previous version is installed.

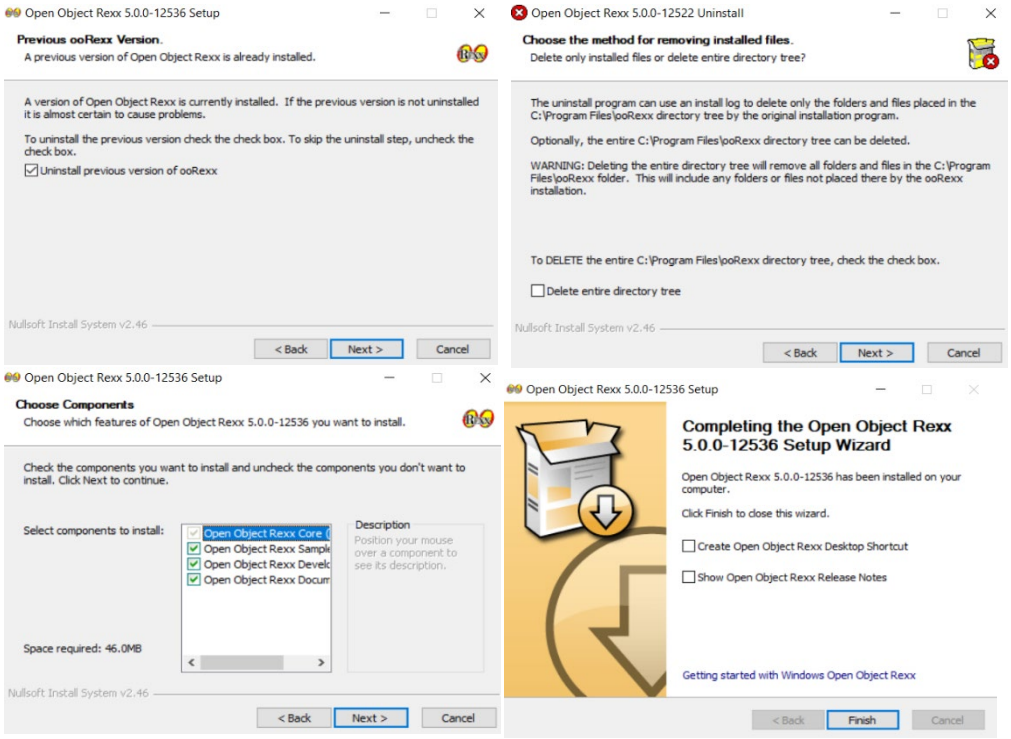

Figure 3 ooRexx Installation Process

4.2 Installation of Java

Bellsoft provides a wide range of different Java versions for different operating systems under the following link:

https://bell-sw.com/pages/downloads/

For this project the “MSI”-version of the Windows x86 64-bit version was downloaded which includes an installer for the chosen Java version (see Figure 4)

Figure 4: Java version from Bellsoft.

If the same version of Java is already installed on the system, the installer will give the user the option to either change, repair or remove the installation (see top left picture in Figure 5).

If a new version of Java is installed, the installer will automatically let the user decide what features should be installed (see top right picture in Figure 5). By default, the entire package is installed - this can be seen when "Liberica Full JDK" is highlighted in blue. If nothing specific is selected, Java will be saved in the "Program Files" folder.

After that the installer will either start the changing or installation process. When this process is complete the installation is finished (see bottom left picture in Figure 5). As with Open Object REXX no additional check-boxes were marked.

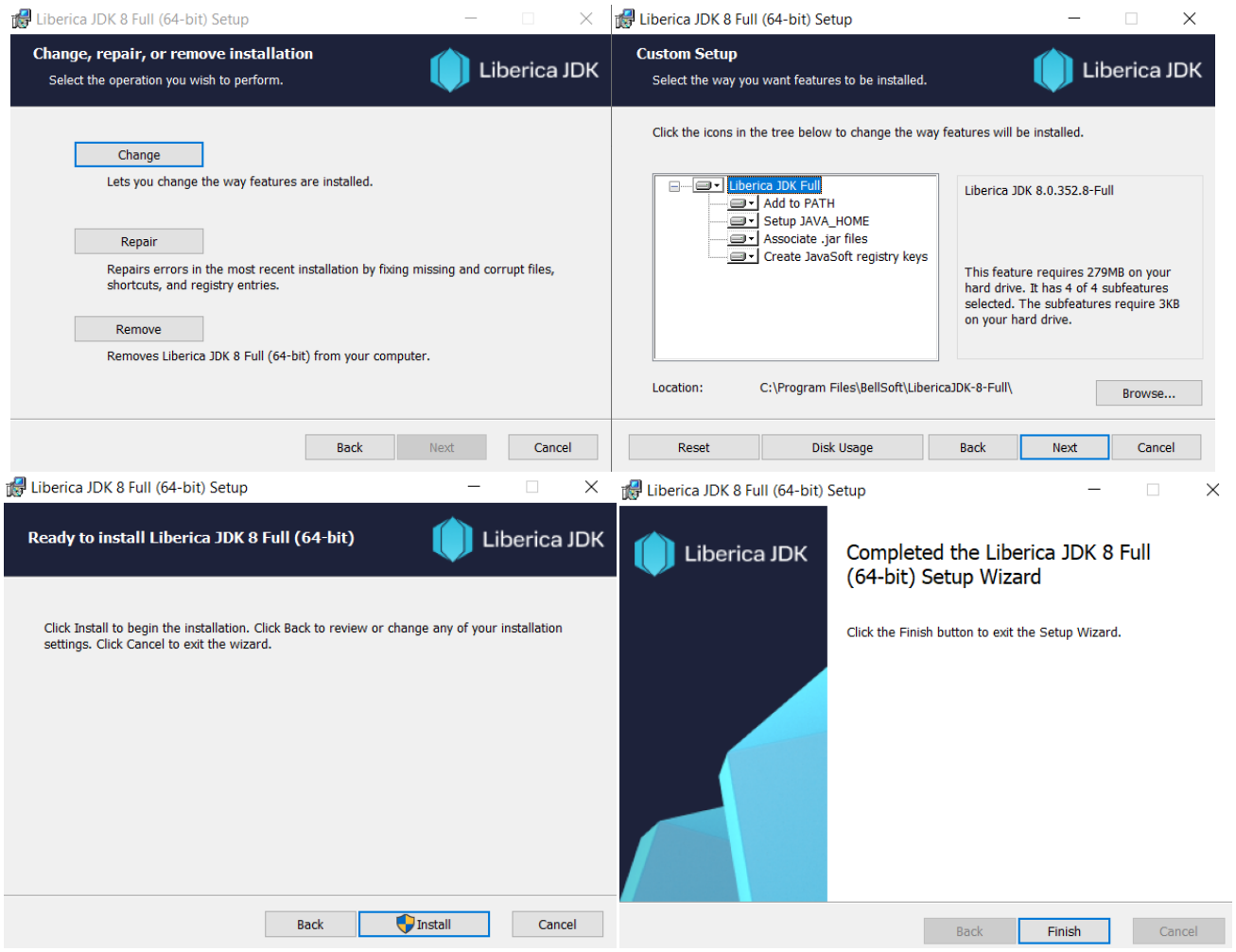

Figure 5: Java Installation Process

4.3 Installation of BSF4ooRexx

The Bean Scripting Framework for Open Object REXX can be downloaded from the website of SourceForge through the following link:

https://sourceforge.net/projects/BSF4ooRexx/files/latest/download

The needed files are downloaded as a .zip-file which must be unzipped before starting the installation process. Depending on the system the unzipping may take a while. The download contains the installations for the different operating systems and through the installation process the framework automatically detects the required architecture (e.g. 32-bit or 64-bit).

If a previous version of BSF4ooRexx is installed on the system, it is recommended to deinstall the old version before installing the new on. This can be achieved by executing “uninstall.exe” for the correct operating system. For most windows systems the .exe-file can be found in the following folder:

“C:\Program Files\BSF4ooRexx\install\windows”

The installation process can be done manually by first unzipping the file downloaded from SourceForge. After that the “install.cmd”-file can be found in the “install/windows”-folder from the just unzipped download.

Alternatively, the file can be downloaded and installed through the “Command Line Prompt” (cmd) which can be accessed through “Windows-Key + R” combination. After opening the “cmd” the text from Figure 6 can be copied and executed line by line to complete the process. Line 7 will open the installer for BSF4ooRexx 850 which demands some user-inputs that need to be followed.

```
1 curl -L --output downloads/BSF4ooRexx850.zip
https://sourceforge.net/projects/BSF4ooRexx/files/latest/download
2 cd downloads
3 tar -xf BSF4ooRexx850.zip
4 cd BSF4ooRexx
5 cd install
6 cd windows
7 install.cmd
```

Figure 6: Installation Steps for BSF4ooRexx

cURL stands for “Client URL” and is a command line tool to send or request data using URL syntax (Stenberg, o.D.). The command in Line 1 automatically downloads the data from the given https-address and saves it in the “Downloads”-Folder under the name “BSF4ooRexx850.zip”. The different “cd”-commands are used to jump between the different directories. Finally, the “tar”-command in Line 3 extracts / unpacks the .zip-folder in the current directory.

If needed the text from Figure 6 can be copied into a batch-file which will automatically execute all lines.

4.4 Checking Installations Correctness

To check if all the installed parts are working together correctly the “ooRexxTry.rxj” program-file can be executed. The program itself is a coding-platform for ooRexx which included a simple code and output field.

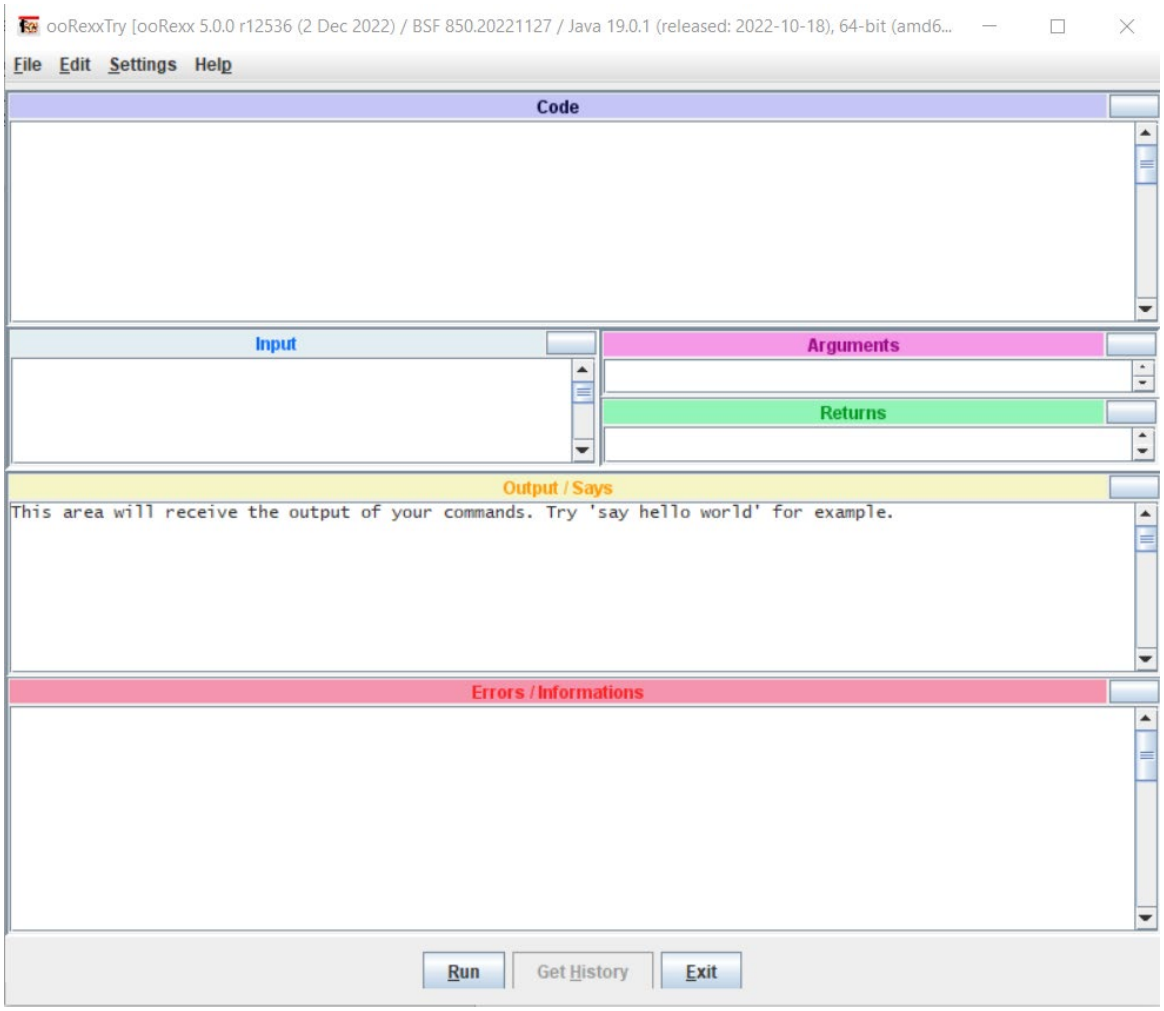

Figure 7: ooRexxTry.rxj coding-tool

4.5 Classpath – Variable

If an older version of BSF4ooRexx was previously installed on the system, it can happen that the system is still referencing these older versions. In that case the “ooRexxTry.rxj”-program or the programs shown in this thesis will not be able to execute correctly since the old references do not include the new and needed libraries that were added with BSF4ooRexx 850.

In most cases, the interpreter will return an error message to show the user that some elements could not be found or loaded. Looking at Figure 8, all the required programming languages and frameworks were installed, yet the system is unable to load the “BSF4ooRexx850” library.

```
C:\Program Files (x86)\BSF4ooRexx850\samples>1-120_JDOR_bw.rxj
   968 *-* ::routine xBSF                  PUBLIC   EXTERNAL "LIBRARY BSF4ooRexx850 BSF
    42 *-* ::requires "BSF.CLS"    -- get ooRexx-Java bridge
    39 *-* call jdor.cls           -- makes routine addJdorHandler([environmentName]) available
Error 98 running C:\Program Files (x86)\BSF4ooRexx850\BSF.CLS line 968:  Execution error.
Error 98.903:  Unable to load library "BSF4ooRexx850".

C:\Program Files (x86)\BSF4ooRexx850\samples>
```

Figure 8: BSF4ooRexx error message

This and similar errors are mostly due to problems during the installation process. It should be noted that these do not always have to be user errors but can also be errors caused by the system itself.

One of the easier solutions that might solve the issue is to use a new command-window. If an old command-window was opened while the old version of BSF4ooRexx was still installed, then this window still points to references from the old version of BSF4ooRexx. Opening a new window after the installation should solve this problem.

If opening a new command-window does not solve the problem, there may be some system-internal referencing issues. For the author, the problem in Figure 8 was solved by deinstalling the current version of BSF4ooRexx 850, deleting all files which were connected to the old version of BSF4ooRexx and by deleting the classpath - variable, which includes the references, from the system.

The easiest way to manage the classpath-variable is to go to the “Advanced System Settings” (pressing “Windows-Key” + “S” and entering “Advanced System Settings”). In the bottom-right corner is a button “Environment Variables”, which when clicked opens a new window, were the user can add, edit or delete certain system-variables. After deleting the classpath-variable it is imperative to click the “OK” – button, otherwise the changes will not be saved.

Figure 9 shows the different steps of this process. Since the authors system has German set as its main language all the menus are also in German. However, the structure and design of the various menu items are language-independent and identical in every language. In this case “Umgebungsvariablen” means “Environmental Variables” and “Löschen” can be translated to “Delete”.

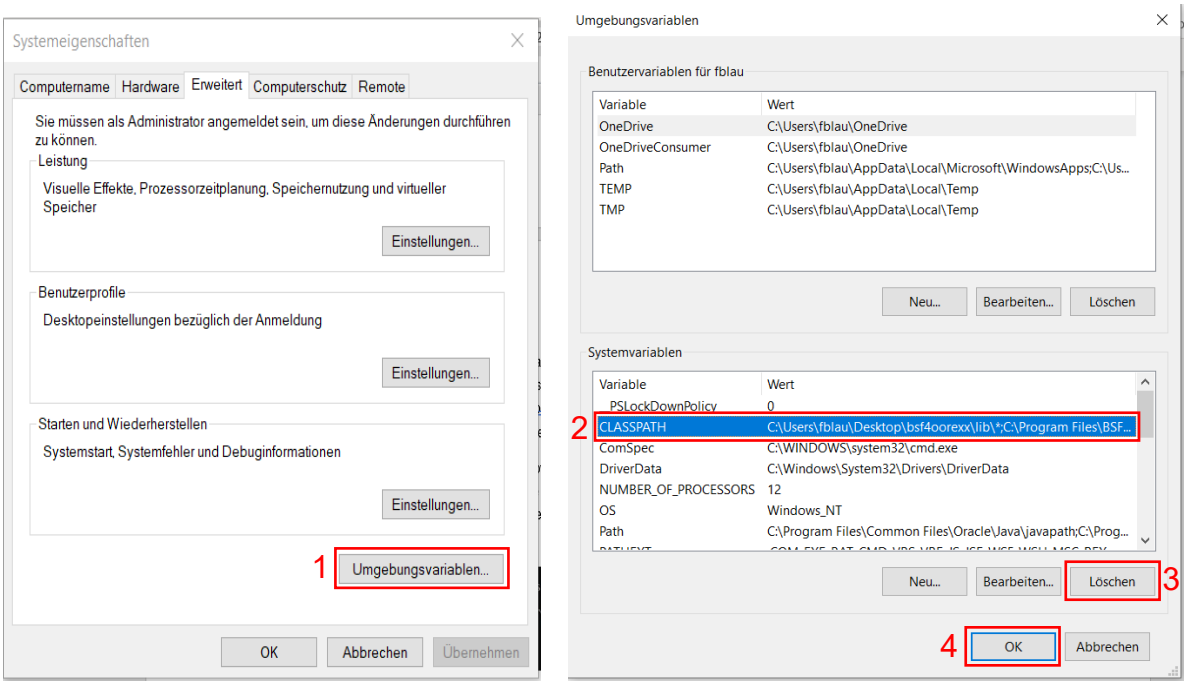

Figure 9: Setting Classpath-Variable

After the classpath-variable is deleted BSF4ooRexx 850 can be installed again. If the classpath-variable is deleted or empty the installation process of BSF4ooRexx 850 should automatically generate a new classpath-variable with the current and correct references.

After this process, the classpath-variable can be viewed by opening the console through “Windows-Key” + “R” and typing “cmd”. In the newly opened window typing “set classpath” will display the content of the classpath-variable. Figure 10 below shows this command and the results for the author’s system.

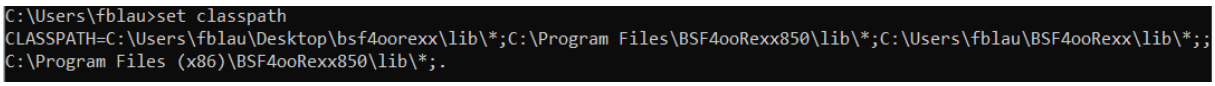

Figure 10: Content of the classpath-variable

As can be seen in Figure 10 the classpath-variable points to references of BSF4ooRexx 850 after the new installation. This indicates that the REXX-Interpreter now uses the correct version of BSF4ooRexx and that the “ooRexxTry.rxj” program should execute correctly.

5. Creating Graphics with Java

Java provides programmers with a variety of different tools and frameworks to create graphics and GUI elements. Most of these tools are collected in the Java Foundation Classes (JFC) which are integrated in Java by default. This thesis focuses on the creation of graphics with the Java 2D API.

The Java 2D API is an extension of the “Abstract Windowing Toolkit” (AWT) (Oracle, o.D.-d). Through the REXX command handler “JDOR” (= Java Drawing for ooRexx) programmers have the capability to use elements from the Java 2D APIs “`Graphics`” and “`Graphics2D`” classes in ooRexx without prior knowledge of the syntax and structure of Java (Flatscher, 2022b).

5.1 Abstract Windowing Toolkit (AWT)

The Abstract Windowing Toolkit is a collection of classes that allows the use of graphical components in a Java program to create graphical user interfaces (GUI). It is part of Java since version 1.0 and can be used on every Java-supported operating system (Schäling, 2010).

AWT is a platform-dependent component which uses subroutines and peer-classes of its native platform. Therefore, an AWT application will look like a standard windows-application when running on a windows system or like a Mac application when running on an Mac system (www.softwaretestinghelp.com, 2022). The reason for this is that the AWT application passes the visualisation and interaction of the elements to classes of the operating system, which then creates the GUI with the existing look of the elements for the operating system (Ullenboom, 2014, pp. 777,778).

Due to the fact that the AWT is available on every platform it only contains elements that are available on each platform (Ullenboom, 2014, p. 777). New elements, that are only available on one platform need to be implemented manually (Ullenboom, 2014, p. 777).

Because each AWT component draws resources from the native platform and these are outside of Java's memory management, these components are called heavyweight components (Ullenboom, 2014, p. 777).

Due to the heavyweight and platform-dependent character of AWT, newer and platform-independent frameworks, such as Swing of JavaFX, are more frequently used in modern applications (www.softwaretestinghelp.com, 2022).

Since Java 1.2 the AWT package is implemented an extended in the Java Foundation Classes (JFC) which among other things includes the Swing components and Java 2D API (Ullenboom, 2014, p. 778).

5.2 Java 2D API

The Java 2D API is a core component since Java 1.2 and allows programmers to create and manipulate shapes, texts, fonts and images (Oracle, o.D.-d). The central class in the Java 2D API is the `java.awt.Graphics2D`, which itself is a subclass of the `java.awt.Graphics` class (Day, 1998).

`Graphics2D` brings a more uniform capability and utility support for manipulating two-dimensional shapes like text, lines and other objects (Day, 1998). In comparison to the `Graphics` class, the `Graphics2D` class provides the programmer with advanced control over geometry, coordinate transformations, colour management and text layout (Oracle, o.D.-a).

A coordinate system is needed to give drawing instructions to a computer. In most cases, the zero point of the coordinate system is in the upper left corner. As can be seen in Figure 11, if the x-value is increased, the drawing shifts to the right and an increase of the y-value shifts the drawing downwards (Oracle, o.D.-b). The Java 2D API distinguishes between two different coordinate systems:

- User Space: A device-independent system that the program uses. Each object created in Java2D is specified in the User-Space.
- Device Space: A device-dependent system that changes according to the used device (screen, window, printer). (Oracle, o.D.-b)

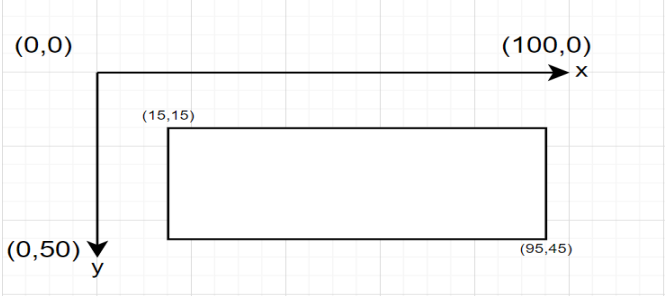

Figure 11: Coordinate system for User-Space in Java

For most cases specifying coordinates as integers is sufficient, yet in some cases floating-values of double-values are needed. Even though the coordinate systems can vary drastically between different devices (e.g. a high-resolution screen and a printer), these differences are invisible to the Java programmer (Oracle, o.D.-b).

In the Java 2D API there are three different levels of configuration information that help the conversion from the device-independent user-space to the device-dependent device-space:

- `GraphicsEnvironment`
- `GraphicsDevice`
- `GraphicsConfiguration` (Sun-Microsystems, 1999)

Through the "`GraphicsEnvironment`" the Java-application gets a collection of all the rendering devices that are connected to the particular platform as well as a list of all the fonts that are available (Sun-Microsystems, 1999). Among others, rendering devices include screens, printers, monitors and image buffers (Sun-Microsystems, 1999).

A visible rendering device is described as a "`GraphicsDevice`". Each "`GraphicsDevice`" can have multiple "`GraphicsConfiguration`". A "`GraphicsConfiguration`" describes a certain mode like 1920x1080 or 1280x720 (Sun-Microsystems, 1999).

So, a "`GraphicsEnvironment`" contains multiple "`GraphicsDevices`" which itself can contain multiple "`GraphicsConfigurations`".

The Java-program automatically detects which device-space is currently needed and performs the transformation from user-space to device-space through its "`AffineTransformation`" object (Sun-Microsystems, 1999). "`AffineTransformation`" defines the rules for the manipulation of coordinates using matrices (Sun-Microsystems, 1999).

For transformations, Java 2D provides the programmer with four basic methods:

- Translate (move)
- Rotate
- Scale
- Shear

“Translate” moves the origin (x=0, y=0) of the graphics context to a new given point. With the “rotate” - command it is possible to rotate a previously created object by a certain angle. The “scale” – command applies a multiplier to both axes for all the following commands (Oracle, o.D.-a). To slant the following drawings the “shear” – command can be used, which applies a multiplier that determines how much the coordinates are shifted / slanted in one axis as a function of their second axis – this method is also known as skewing (Oracle, o.D.-a).

Even though, the Java 2D API provides a number of complex methods for creating different graphics, most Java 2D programs only use a subset of these capabilities found in the “Graphics” class (Oracle, o.D.-c).

Most of the methods within the “Graphics” class can be separated into two different groups:

- Rendering the basic shapes, texts and images through the draw and fill methods
- Setting attributes to those basic drawings and fillings (Oracle, o.D.-c)

Both method-groups can be combined to create a large number of different graphics. Figure 12 shows different graphic objects and how they relate to the two method-groups. While shapes can be created through the draw and fill methods, texts can be edited with the “DrawString” method and the “setFont” and “setColor” method.

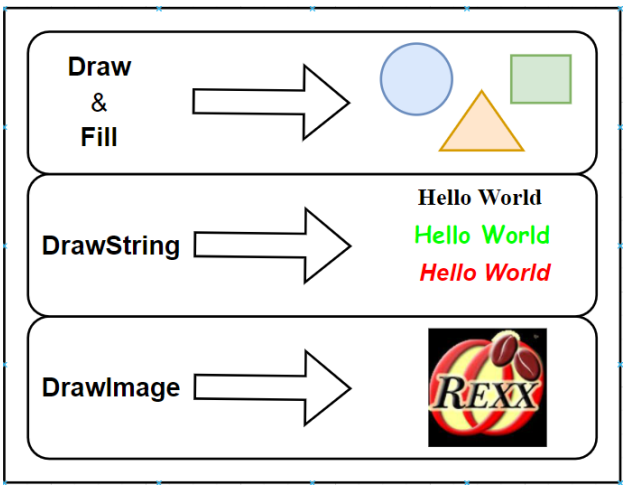

Figure 12: Relation of graphics and methods (Oracle, o.D.-c)

5.3 Java Drawing for ooRexx (JDOR)

The JDOR package allows the Java 2D API environment to be used in ooRexx. With it, programmers are able to exploit the "`Graphics`" and "`Graphics2D`" class of the "`Java.awt.package`" (Flatscher, 2022b, p. 9). Despite this support, the structure and architecture of ooRexx and Java differ greatly (Flatscher, 2022b, p. 7). Due to this fact, ooRexx commands, while using similar wording, look different than their counterparts in Java.

ooRexx and JDOR work with commands that are represented by simple, easy-to-read strings, whose structure and arguments are converted behind the scenes into method calls to Java2D (Flatscher, personal communication, 27.12.2022). Even though the commands differ between the two programming languages, the commands in ooRexx are formulated in such a way that it is also possible for Java programmers to understand and use them after a short time.

JDOR also allows programmers to access the Java2D data with directories and HashMaps of loaded colours, fonts and stroke-types. This access makes it possible to define additional colours, fonts, and stroke-types with nicknames in a HashMap that can later be used by the ooRexx program (Flatscher, 2022b, p. 8).

Through the functions of the JDOR Command Handler, the programmer is enabled to store various images and graphics with ease. Commands can also be stored in a plain text file to create macros for Java2D in ooRexx. To create animations the JDOR Command Handler temporarily halts the execution of the program. (Flatscher, 2022b, p. 9)

Nevertheless, in case of a problem, ooRexx programmers will usually only find code in Java when they search the internet, which they must first convert into ooRexx code. Depending on the problem, this can itself lead to various difficulties and challenges.

To give a better understanding of the different JDOR – commands in ooRexx the following chapter will provide a list of JDOR – commands and their counterpart in Java 2D.

5.3.1 JDOR – Commands

Although the commands written in JDOR / ooRexx and their counterparts in Java often use the same wording, their structure differs in many cases. One of the most striking differences is the absence of brackets and commas.

This chapter is mainly based on the JDOR – documentation from Rony G. Flatscher and the "`Graphics`" and "`Graphics2D`" – documentation from Oracle. The documentation of JDOR can be found in the BSF4ooRexx-folder with the following path:

/BSF4ooRexx850\information\jdor\jdor_doc.html

The "Graphics" and "Graphics2D" documentation can be retrieved from the following two links:

https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html

https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html

JDOR provides ooRexx programmers with a wide range of different commands, yet not each of these commands was used in the programming – part of this bachelor thesis.

Table 2 lists the commands that were used in this bachelor thesis. Each command is described in its scope and additionally the different possible arguments are described in more detail.

The arguments which do not necessarily have to be provided when using the commands are written in square brackets in this table (this is only used to distinguish between necessary and non-necessary arguments). In reality, square brackets are not used when providing additional arguments.

It should also be mentioned that there are multiple combinations of attributes for each command in ooRexx and in Java. While with ooRexx the additional arguments, which can only be used together, are placed together in the square brackets, for Java only the complete command without square brackets is specified.

One of the biggest differences between JDOR and Java2D is JDOR's "`goto`" command. This defines the start coordinates for the following commands and is mainly used to separate these coordinates from the other coordinates and eliminates the need to re-enter the same coordinates several times (Flatscher R., personal communication, 27.12.2022).

In Java a "`Graphics2D`" object must be created to draw shapes. For the Java-commands in Table 2 this object is named "`G2D`".

JDOR	Java	Description for JDOR
`WINSIZE width height`	`setSize(double width, double height);`	Sets the size (width and height) of a new window
`NEW width height [type]`	`public BufferedImage(int width, int height, int Imagetype);`	Creates an image /drawable area on the window. Needed if area should be unequal to 500x500
`WINSHOW`	`setVisible(true);`	Shows the current window
`WINHIDE`	`setVisible(false);`	Hides the current window
`BACKGROUND colornickname`	`G2D.setBackground( Color color);`	Sets the colour of the background. Only works in combination with the command ClearRect
`GOTO x y`	-	Sets the x1 and y1 coordinates for the following commands
`FONTSIZE size`	`G2D.setFont(G2D.getFont(). deriveFont(float size);`	Sets the fontsize for the following commands

FONTSTYLE style	G2D.setFont(G2D.getFont(). deriveFont(int style);	Sets the fontstyle for the following commands. Style-attribute (0: Normal, 1: Bold, 2: Italic, 3: Bold+Italic)
FONT NickName fontname	Font name = new Font (string fontname, int style, float size);	Saves the current font under a nickname with the current style and size. FontName must be in explanation marks
FONT NickName	G2D.setFont(Font name)	Sets a previously saved font as the font for the following commands
STROKE Nickname width [cap] [join] [miter_limit] [dashpattern**[]** dashphase]	BasicStroke name = new BasicStroke(float width, int cap, int join, float miter_limit, float[] dashpattern, float dash_phase);	Saves a stroke under a nickname with a width (in pixels). Cap-Attribute (0: BUTT, 1: ROUND, 2: SQUARE). Join-Attribute (0: BEVEL, 1: MITER, 2: ROUND). Miter_Limit-Attribue (Sets limit for Join-Miter). Dashpattern-Attribute (Array with lengths and spacings). Dashphase-Attribute (Offset for starting point for the first dashpattern)
STROKE strokeNickName	G2D.setStroke(Stroke name);	Sets a previously saved stroke as the stroke for the following commands.
DRAWLINE x y	G2D.drawLine(int x1, int y1, int x2, int y2);	Draws a line from the current coordinates to the given coordinates

DRAWRECT width heigth	G2D.drawRect(int x1, int y1, int width, int height);	Draws a rectangle from the current coordinates (upper-left) with the given width and height
DRAWOVAL width heigth	G2D.drawOval(int x1, int y1, int width, int height);	Draws an oval in an invisible rectangle from the current coordinates (upper-left) with the given width and height
DRAWPOLYLINE xPoints**[]** yPoints**[]** nPoints	G2D.drawPolyline(int[] xPoints, int[] yPoints, int nPoints);	Draws a line between multiple given points. Xpoints-Attribute (Array of x-coordinates) ypoints-Attribute (Array of y-coordinates) npoints-Attribute (number of points – must be equal to both points-arrays size)
DRAWARC width height startAngle arcAngle	G2D.drawArc(int x, int y, int width, int height, int startAngle, int arcAngle);	Draws an arc in an invisible rectangle starting from the current coordinates (upper-left) with the given width and height. The angles are measured in degree starting from the 3 o'clock position going counterclockwise. The arcAngle-Attribute gives the endpoint of the arc.
DRAWSTRING text	G2D.drawString(String text, int x, int y);	Draws a string (=text) at the current coordinates
FILLRECT width height	G2D.fillRect(int x, int y, int width, int height);	Fills a rectangle starting from the current coordinates (upper-left) with the given width an

FILLOVAL width heigth	G2D.fillOval(int x, int y, int width, int height);	Fills an oval in an invisible rectangle starting from the current coordinates (upper-left) with the given width an height.
FILLARC width heigth	G2D.fillArc(int x, int y, int width, int height, int startAngle,int arcAngle);	Fills an arc in an invisible rectangle starting from the current coordinates (upper-left) with the given width an height. (structure of arc see DRAWARC)
LOADIMAGE NickName Path	ImageIO.read(new File(string path);	Saves an image from the given path under the given nickname.
DRAWIMAGE NickName [width height] [bgcolor]	G2D.drawImage(drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer);	Draws a previously saved image. If width and height are provided the image will scale up or down. Bgcolor-Attribute sets the backgroundcolor for transparent image-parts.
TRANSLATE x y	G2D.translate(int x, int y);	Sets a new origin for the coordinate-system.
ROTATE theta [x y]	G2D.rotate(double theta, double x, double y);	Rotates the following drawing in the given theta (=angle in degree) around the origin of the coordinate system. „x“ and „y“ sets a new origin for the rotation
SCALE shx shy	G2D.scale(double shx, double shy)	Applies a multiplier to the coordinates, width and height attributes for the following commands
SHEAR shx shy	G2D.shear(double shx, double shy)	Applies a factor that determines how much an object shifts in relation to its “x” and “y” coordinates.

`CLEARRECT width heigth`	`G2D.clearRect(int x, int y, int width, int height)`	Draws a rectangle with the previously defined backgroundcolour (works only in combination with the background-command)

Table 2: JDOR-commands and their Java2D - counterparts

6. JDOR-Examples in Open Object REXX

This chapter deals with the practical implementation of the information from the previous chapters.

It shows several different small programming examples that are intended to illustrate some of the functionality of the JDOR package in more detail. For this purpose, the different commands shown in Table 2 have been used to create these small programming examples.

Even though the examples only use a small amount of the different ooRexx functions, like loops or question-blocks, it is still recommended to be familiar with the syntax of ooRexx. Nevertheless, the descriptions of the different sections of code should be sufficient to understand the structures and processes.

Each example mainly consists of three parts. First, a small description is given which explains the overall aim of the respective program. Then, the code is shown and described in detail. The last part is a picture of the generated output and its description.

The basic idea is that the respective programs build upon each other - therefore, each program tries to extend some of the functionality of the previous program. The aim of this chapter should be to provide some elementary use-cases and guidance for the different commands so that other programmers can build upon them to create more complex programs.

Since the BSF4ooRexx 850 framework is continually in development some changes, improvements and extensions to the JDOR – package could be published in the future that were not described in this bachelor-thesis because they were not yet publicly available at the time of writing the various chapters.

The JDOR – documentation which is part of the BSF4ooRexx 850 program – folder contains additional information for all the different commands and is revised and supplemented with each new version and contains nearly all available JDOR-commands.

6.1 Drawing Simple Shapes – JDOR_shapes.rxj

This example shows some of the different drawable shapes from the "`Graphics`" and "`Graphics2D`" libraries. It provides examples for drawing and filling rectangles, ovals, lines and round-rectangles in basic colors. It should also show how overlapping shapes act.

```
1   CALL addJdorHandler
2   ADDRESS jdor -- set default environment to JDOR
3
4   --Creating and showing a new window
5   win_width = 500
6   win_height = 300
7   NEW win_width win_height
8   WINSHOW
9
10  --Drawing the different shapes
11  GOTO 50 (win_height / 2 -50) --200
12  DRAWRECT 200 100
13
14  GOTO 200 (win_height / 2 - 25) --225
15  COLOR red
16  DRAWROUNDRECT 150 50 45 45
17  COLOR gray
18  FILLROUNDRECT 150 50 45 45
19
20  GOTO 300 (win_height / 2 - 40) --210
21  COLOR green
22  DRAWOVAL 150 80
23  COLOR red
24  FILLOVAL 150 80
25
26  GOTO 50 (win_height / 2) --250
27  COLOR blue
28  DRAWLINE 450 (win_height / 2)
29
30  SLEEP 40
31  ::REQUIRES "jdor.cls"
```

Figure 13: Shapes in JDOR - JDOR_shapes.rxj

The basics for each JDOR program can be found in line 1,2 and 31 of Figure 13. The code in line 31 starts the "jdor.cls" file before executing the other code-lines. Line 1 loads the JDOR – Command – Handler with its default name "JDOR". Line 2 addresses this handler under its default name, connecting the following commands to the specific handler. These lines are needed to use Javas "`Graphics`" and "`Graphics2D`" libraries in ooRexx.

To draw in ooRexx a new window or frame is needed where these drawings can be displayed. With the code in line 4 and 5 the variables "win_width" and "win_height" are created which are used to save the width and the height of the window. In line 7 a new window is created with the previously saved width and height. Line 8 displays the window on the current screen.

The drawings in this program are all centred around the middle of the y-axis– which in this case is "150" (half of "win_height").

Before the first rectangle is drawn, the starting point ("x" and "y") of the new drawing must first be selected with the "goto" command. The first drawing should be 50 steps away from the origin on the x-axis. To place the rectangle (height 100, length 200) in the centre, half of its height must be subtracted from the y-axis starting point (100 / 2 = 50).

The same idea is followed by the "goto"-command before drawing the oval and round rectangle, with their represented height (see line 14 and 21). The "draw"-command only draws the outlines of the respective shapes in the previously defined colour (e.g line 15 sets the colour to "red" for the "DRAWROUNDRECT"- Command). The "DRAWROUNDRECT" – command draws an empty rectangle with rounded edges in the colour red. The "strength" of the curve-effect is set by the third and fourth argument of the "DRAWROUNDRECT" and "FILLROUNDRECT" command (if both arguments are the same, the edges are perfectly circular – see Figure 14). In line 18, the "FILLROUNDRECT" – command is used to draw a grey rectangle with rounded edges.

Since the "FILLROUNDRECT" – command in line 18 was written after the "DRAWRECT" – command (line 12) the round rectangle overlaps the edged rectangle. The same can be seen with the "DRAWOVAL" and "DRAWLINE". A new object is always drawn on top of all the previously defined objects.

While the shapes, like rectangles and ovals, are defined by their width and height, lines are described by their endpoint (they start from the point set by the previous "goto"-command and end at the given coordinate)

Figure 14 on the next page shows the output of the code from Figure 13. It shows the four different shapes being drawn on top of each other in the middle of the window.

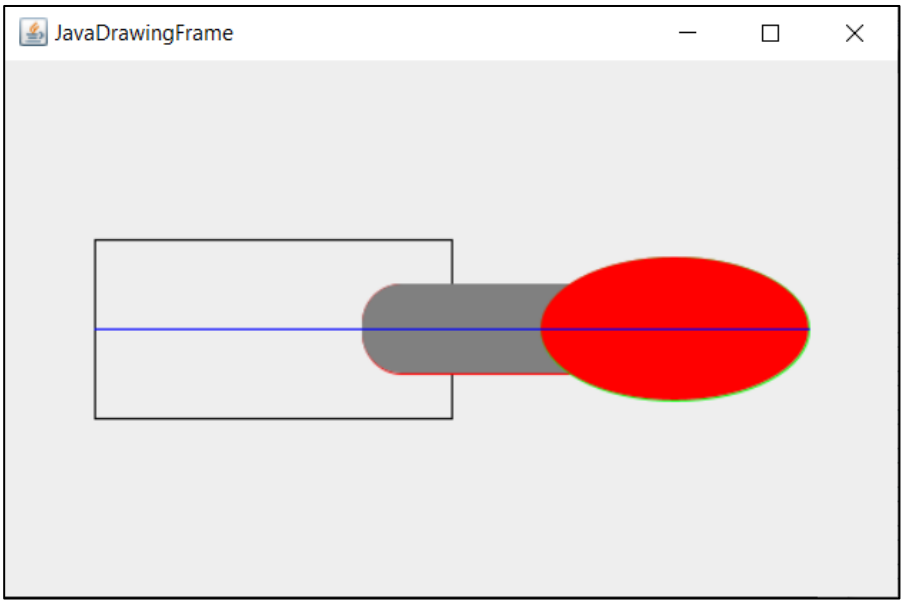

Figure 14: Output of JDOR_shapes.rxj

The standard-width of strokes used by the "DRAW"-commands is one pixel. Due to this small size factor, the different line-colours are hard to see if the shapes are filled with another colour (e.g. the green line around the red oval is only barely visible).

The "STROKE" – command can help in such cases, as it provides the programmers with more customization-possibilities. The next chapter takes a closer look at the different stroke-types and their effects.

6.2 Different Strokes in JDOR – JDOR-strokes.rxj

The following example shows the saving and use of the various stroke-types. For this reason, lines and rectangles with different stroke-widths and stroke-settings are created. In generally, strokes can be used in combination with most of the "DRAW"-commands (e.g. DRAWLINE, DRAWRECT).

Before drawing with different strokes, these strokes must be defined and saved first (this happens in line 11 to 15 in Figure 15 with the "STROKE"-command). The defined strokes are always saved with their nickname in the internal registry of the program and can be used later.

Strokes consist of a width, cap-type, join-typ, miterlimit, dashpattern and dashphase. The "CAP-TYPE" sets the decoration of the starting- and endpoint of a line. The "square" and "round" cap add some pixels to the line, making it a little longer.

Through the "JOIN-TYPE" the connecting behaviour can be set to define how two lines in a shape react if they overlap. By default, two lines, joining in a 90-degree angle, form a straight edge. The "MITERLIMIT" can be used to determine at what point two lines running pointedly towards each other are cut off at the tip.

An array can be used to create a dashed pattern. There are two different ways in ooRexx to create a dashed line. Firstly, a Java-Array can be created with "dashphase_stroke1 = bsf.createJavaArrayOf("float.class", 15, 8, 15,8)" (line 11) which can then be used as an attribute for a new stroke ("dashphase_stroke1" in line 12). Secondly, the array can be specified immediately, in round brackets, as an attribute for a stroke ("(20,10,8,10)" in line 13).

```
1   CALL addJdorHandler --
2   ADDRESS jdor -- set default environment to JDOR
3
4   --Creating and showing a new window with width = 500 and height = 400
5   win_width = 500
6   win_height = 400
7   WINSIZE win_width win_height
8   WINSHOW
9
10  --Creating / Saving strokes
11  dashphase_stroke1=bsf.createJavaArrayOf("float.class", 15, 8, 15,8)
12  STROKE stroke1 3 2 0 10 "dashphase_stroke1" 0
13  STROKE stroke2 5 1 1 10 "(20,10,8,10)" 0
14  STROKE stroke3 8 1 0
15  STROKE stroke4 9 2 2
16
17  --Creating the drawings
18  GOTO 50 50
19  COLOR red
20  STROKE stroke1
21  DRAWLINE 300 50
22
23  GOTO 50 100
24  COLOR orange
25  STROKE stroke2
26  DRAWLINE 300 100
27
28  GOTO 50 150
29  COLOR black
30  STROKE stroke3
31  DRAWRECT 250 200
32  COLOR blue
33  FILLRECT 250 300
34
35  GOTO 350 50
36  COLOR green
37  STROKE stroke4
38  DRAWRECT 100 300
39
40  SLEEP 40
41  ::REQUIRES "jdor.cls"
```

Figure 15: Strokes in JDOR - JDOR_strokes.rxj

The array can be seen as an on-off switch for the drawing-pen, where entries with an even index draw a line and entries with an odd index leave a blank space (note that the array starts with index “0”, which is seen as an even number). In case of “`STROKE2`” the first 20 points are drawn, then 10 points are left blank, then 8 points are drawn and then 10 points are left blank again. After all elements of the array are drawn the pattern starts again from index “0”.

After a stroke is defined it can be set by the “`STROKE nickname`”-command (see line 20) and used for each following command till a new stroke is set. For example, the line drawn by the “`DRAWLINE 300 50`”-command on line 21 is drawn in a red colour (set on line 19) and with stroke “`stroke1`” (set on line 20).

Figure 16 shows the output from the code in Figure 15. In comparison to the green line around the red oval in Figure 14, the black line around the blue rectangle is very well visible thanks to the increased width by the “`STROKE`”-command. It also shows the different dash-patterns from “`STROKE1`” and “`STROKE2`”.

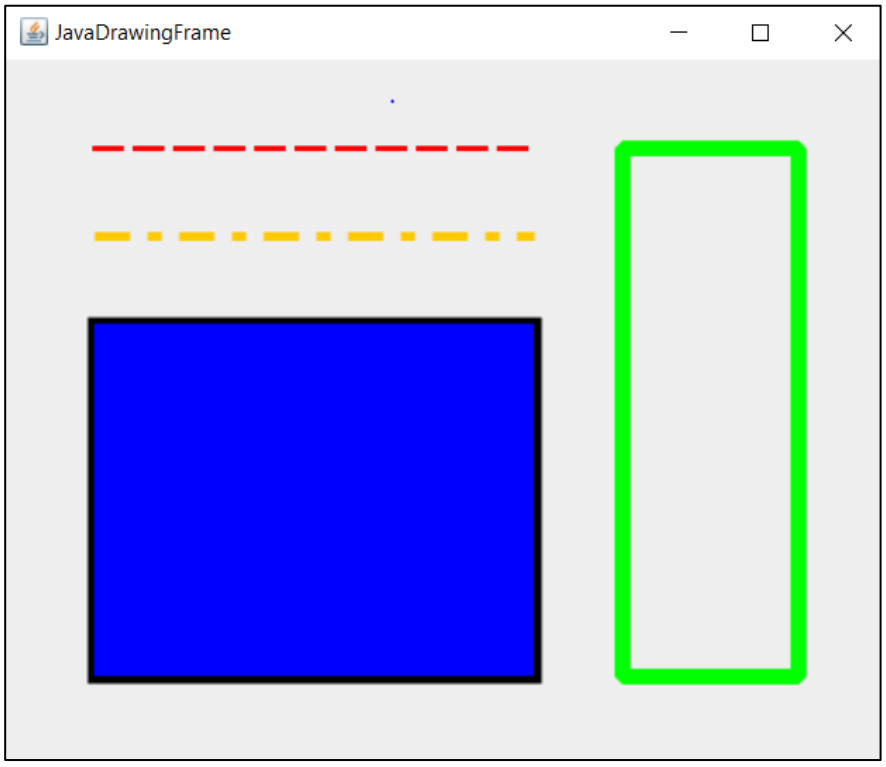

Figure 16: Output JDOR_strokes.rxj

6.3 Displaying Text in JDOR – JDOR-Strings.rxj

In addition to various forms, no graphic interface can usually do without some form of texts and fonts. The "Graphics" and "Graphics2D" classes also offer various possibilities for displaying texts in a window. In addition to the font, their size and style can also be adapted to the respective situation.

This small program shows, how texts can be displayed in JDOR using the "DRAWSTRING"-command and how they can be combined with rectangles and lines to create textboxes and underlined texts.

```
1   CALL addJdorHandler -- load
2   ADDRESS jdor -- set default environment to JDOR
3
4   --Creating and showing a new window
5   win_width = 500
6   win_height = 200
7   WINSIZE win_width win_height
8   WINSHOW
9
10  --Creating and saving different fonts
11  FONTSIZE 15
12  FONTSTYLE 1 -- 1=BOLD
13  FONT BOLD_15_COMIC_S "Comic Sans MS"
14  FONTSIZE 20
15  FONTSTYLE 2 -- 2=ITALIC
16  FONT ITALIC_20_BLACKOUT "Blackout"
17  FONT Arial_BOLDITALIC_20 "Arial-BOLDITALIC-20" -- BOLD + ITALIC
18
19  GOTO 50 50
20  COLOR blue
21  FONT BOLD_15_COMIC_S
22  DRAWSTRING "This is a text written in the font Comic Sans MS"
23  STRINGBOUNDS "This is a text written in the font Comic Sans MS"
24  PARSE VAR rc x " " y " " width " " height
25  SAY width
26  COLOR black
27  DRAWLINE 50+width 50
28
29  GOTO 50 100
30  FONT ITALIC_20_BLACKOUT
31  DRAWSTRING "This is a text written in the font Blackout"
32
33  GOTO 50 150
34  COLOR red
35  FONT Arial_BoldITALIC_20
36  DRAWSTRING "This is a text written in the font Arial"
37  STRINGBOUNDS "This is a text written in the font Arial"
38  PARSE VAR rc x " " y " " width " " height
39  SAY rc
40  GOTO 50 150-height
41  DRAWRECT width height
42
43  SLEEP 40
44  ::REQUIRES "jdor.cls"
```

Figure 17: Writing in JDOR - JDOR-Strings.rxj

Before a certain font can be used in JDOR, it needs to be defined first. In JDOR, there are generally two ways a new font can be defined. However, only fonts that are stored on the respective system can be used with both variants. To obtain the names of the fonts stored in the system, the program "2-110_JDOR_listShowPrintFonts.rxj" should be launched, which can be found in the sample-folder of the BSF4ooRexx850 installation.

By default, the "`FONTSIZE`" is set to "12" and the "`FONTSTYLE`" to "0". Both parameters can be changed by their respective command (see line 11 and 12 in Figure 17). With the command "`FONT nickname fontname`" a new font is saved with the currently used size and style. So, in line 15 a new font is saved under the name "BOLD_15_COMIC_S" which uses the font "Comic Sans MS" with the size 15 and style 1 (=BOLD). A faster way would be to set all three parameters in one line which is shown in line 17 in Figure 17. All three parameters can be specified in one line. After the nickname for the new font is chosen, the three parameters are then separated by a hyphen.

To use the font for a new "`DRAWSTRING`"-command the font must be set with the "`FONT nickname`"-command which looks up the nickname in the system's registry.

The command "`STRINGBOUNDS`" (see line 23) returns the coordinates, width and height of an invisible rectangle that is drawn around a certain text. This information can be accessed with the "`rc`"-variable, which can be split into its respective information with the "`PARSE` VAR" command (line 24).

When drawing underlines or textboxes it is crucial to understand that text is written on top of the current position, whereas shapes are drawn below the current position. So, a line drawn on the same height as a text is displayed right underneath the text (see line 27 in Figure 17– only the x-coordinate (=50+width) of the endpoint changes, while the y-coordinate stays the same). The result can be seen in the first text in Figure 18.

When drawing a rectangle around a text, the starting point of this rectangle must first be shifted upwards by the height of the text (see the "`GOTO`"-command on line 14). After that, the rectangle, with the width and height provided by the "`STRINGBOUNDS`"-command is drawn perfectly around the text. It needs to be noted, that the

“STRINGBOUNDS”-command adds a small spacing on top of the text (see the third text in Figure 18)

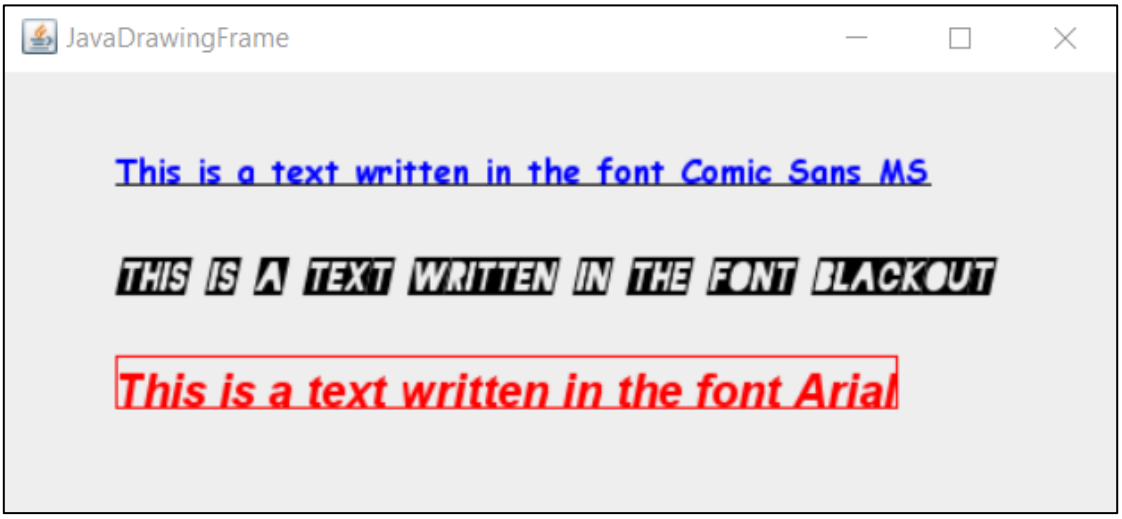

Figure 18: Output JDOR-Strings.rxj

Figure 18 shows the output from the code in Figure 17. As it is described in the code, the first line of text is written in bold and underlined with a black line and the second line of text is written in an italic style. The third line is written in both bold and italic and outlined with a rectangle.

6.4 Using Images in JDOR – JDOR-images.rxj

Images are an important component of many modern graphical user interfaces. Even though the “`Graphics`” and “`Graphics2D`” libraries are not the newest additions to the Java-class system, they do provide valuable functions to use and manipulate images. JDOR allows to use these functions in ooRexx.

The code in this example intends to show the basics of image processing in JDOR. To do this, a previously created image is imported into JDOR, then decorated with a frame and finally saved on the system.

JDOR provides a variety of different commands related to image-processing. The focus in this example lies on the “`LOADIMAGE`”, “`DRAWIMAGE`” and “`SAVEIMAGE`” commands. These are three of the most important commands when using and saving images. Additionally, this example also uses commands shown in the two previous examples, to create different strokes and texts.

```
1   CALL addJdorHandler -- load and add the Java Rexx command handler,
2   ADDRESS jdor -- set default environment to JDOR
3
4   --Creating and showing a new window
5   win_width = 600
6   win_height = 400
7   WINSIZE win_width win_height
8   WINSHOW
9
10  --setting the color and stroke for the frame
11  COLOR woodbrown 139 90 43 255
12  STROKE ImageFrame 30 0 2
13
14  LOADIMAGE Stickman_and_Snowman "Snowman.png" --nickname and path
15  IMAGESIZE Stickman_and_Snowman
16  PARSE VAR rc width_img " " height_img --300x300
17
18  --Drawing the frame in the center of the window
19  GOTO (win_width-width_img)/2 (win_height-height_img)/2 -- 150 50
20  DRAWIMAGE Stickman_and_Snowman
21  DRAWRECT width_img height_img
22
23  --Draw a string on the bottom of the image onto a gold rectangle
24  --String should be displayed in the middle
25  STRINGBOUNDS "WINTERWONDERLAND"
26  PARSE VAR rc x " " y " " width " " height
27  GOTO (win_width-width)/2 (win_height-height_img)/2+height_img-height
28  COLOR gold 255 215 0 255
29  FILLRECT width height
30  COLOR black
31  GOTO (win_width-width)/2 (win_height-height_img)/2+height_img
32  DRAWSTRING "WINTERWONDERLAND"
33
34  --Saving the created image in the same directory
35  SAVEIMAGE "framed_Snowman.png" --path
36
37  SLLEP 40
38  ::REQUIRES "jdor.cls"
```

Figure 19: Using images in JDOR - JDOR-images.rxj

To load an image into JDORs registry the “LOADIMAGE”-command is used in line 14 in Figure 19 to save the “Snowman.png” image under the nickname “Stickman_and_Snowman”. To centre the image in the middle of the frame, the dimensions of the image are needed, which can be accessed through the “IMAGESIZE nickname”-command, which saves the width and height into the “rc”-variable (see line 15 and 16). In combination with the windowsize and image-dimensions the starting point for the image can be calculated and set with the “goto”-command (see line 19). With the “DRAWIMAGE Stickman_and_Snowman”-command the image gets drawn at the current location.

A new width and height for the image can be set with the “DRAWIMAGE” command, yet the dimensions accessed by the “IMAGESIZE” command won’t be affected.

To draw a frame around the picture the "`DRAWRECT`"-command is used in line 21 with the width and height received by the "`IMAGESIZE`"-command. The rectangle is drawn in the colour set by the RGB-Code (139,90,43,255) in line 11 and with a stroke of width 30 and a round join-behaviour set in line 12.

Drawing a rectangle with a wider stroke-size will draw half the strokes width outside of the rectangle's set size and half its width inside of it. Thus, the greater the width of the strokes, the more image area is covered by the newly drawn rectangle. This can be seen in Figure 20 where parts of the outer pixels of the "Snowman.png" are covered by the rectangle in the newly created output-image.

The "Winterwonderland" text inside the rectangle in Figure 20 is created in the same way as the third text-line described in chapter 6.4 "Using images in JDOR – JDOR-images.rxj". Using the window-dimensions, image-size and text-size the string and the golden rectangle can be displayed in the middle of the lower edge of the image through the "`GOTO`", "`DRAWSTRING`" and "`DRAWRECT`" commands (see line 25 to 32). The result can be seen on the right side of Figure 20.

With the "`SAVEIMAGE "framed_snowman.png"`" the resulting image is saved under the name "framed_snowman.png" in the current path. If the image needs to be saved in another location the path could be added to the filename (e.g. "`SAVEIMAGE "C:/Users/Pictures/framed_Snowman.png"`").

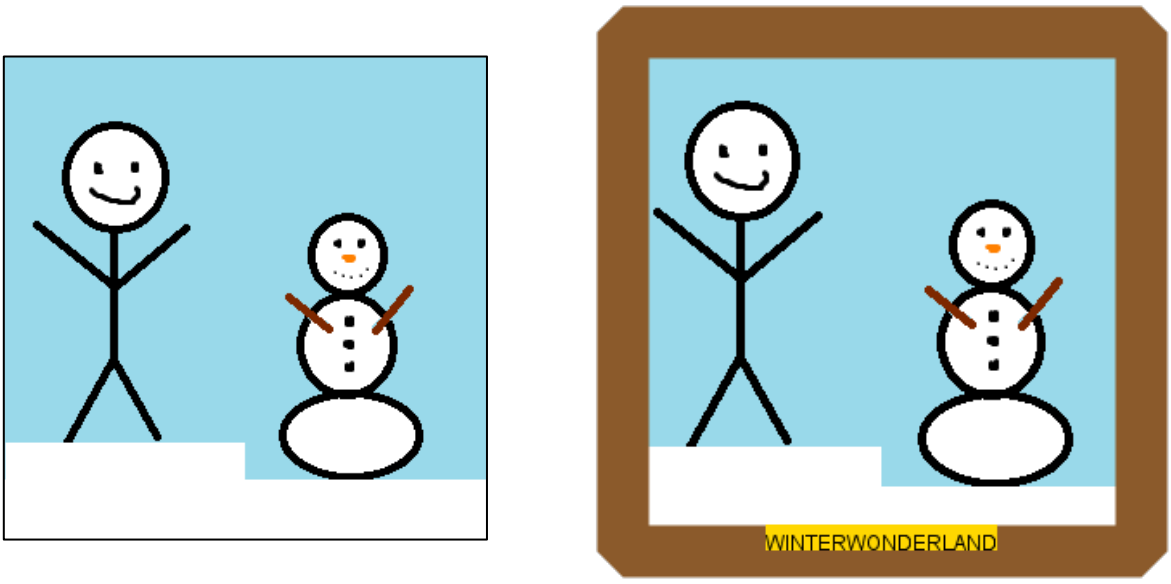

Figure 20: Snowman.png (left) and output of the code from JDOR-images.rxj (right)

6.5 Rotate, Scale, Translate and Shear in JDOR

This example shows the basics of rotating, scaling, translating and shearing in JDOR. These four methods together build the basis for "`AffineTransform`" in "`GRAPHICS`" and "`GRAPHICS2D`". Rotating objects is needed if the goal is to draw a shape which is not parallel to the x-axis or y-axis. Scaling applies a multiplier to both axes which affects the size and coordinates of the following drawings. Translating lets the programmer move the origin of the coordinate-system from the current window. Shearing is the most complex of the four methods and allows to shift and "stretch" objects.

To give a better example of these transforming methods. First, the lines of a coordinate-system are drawn in a window. After that, the different methods are applied to various rectangles.

For the creation of the coordinate-grid two loops are needed. The first loop created the vertical lines and the second loop the horizontal lines. The lines should be placed 25 points apart. Loops in ooRexx often use a counter-variable and a step. In case of the "`DO i=0 TO win_width BY 25`" in line 16 in Figure 21 the counter-variable "i" starts at "0" and iteratively goes up by "25" till it reaches the value of "win_width" (=500). At each step, a vertical line with the height of the window is drawn. The same process is applied to the horizontal lines respectively. The two lines that form the center of the grid are drawn in black (line 24 to 28).

Through the "`TRANSLATE win_width/2 win_height/2`" command on line 31 the origin of the coordinate system is set to the middle of the window. Consequently, it is possible to select starting points with the "`GOTO`"-command which have negative coordinates. If that is the case, the "`GOTO`"-command must be written in exclamation marks (see line 32).

The "`ROTATE`"-command in JDOR allows to rotate objects around a certain point. This point can be provided through the "x" and "y" coordinates, like "`ROTATE 45 100 100`". If no such point is provided, as in line 34 in Figure 21, then the objects are rotated around the origin. Through the "`ROTATE 45`" command in line 34 all following objects are rotated clockwise at a 45° angle towards the origin (in this case the center of the frame) respective to the current starting point selected by the "`GOTO -25 -50`" command.

“ROTATE” commands do not overwrite each other but rather add their values up. So, a “ROTATE 45” followed by a “ROTATE 90” will result in an overall rotation of 135°. To reset a rotation either the “RESET” command (which also resets the origin to the top left) or the “ROTATE” command can be used. Vital to mention is that if the rotate command is used, it needs to add up to 360° or 0°. So, the “ROTATE 315” in line 36 will reset the rotation set by “ROTATE 45” in line 34.

The rectangle drawn by the “FILLRECT 50 50” in line 38, at the starting point 50,50 proofs that no rotation is in effect (it is drawn in the exact location set by “GOTO 50 50” – see the green rectangle in Figure 22).

```
10  --setting the colors
11  COLOR coordinate_system 190 190 190 200
12  COLOR middle 0 0 0 255
13
14  --Drawing the system
15  COLOR coordinate_system
16  DO i=0 TO win_width BY 25
17      GOTO i 0
18      DRAWLINE i win_height
19  END
20  DO i=0 TO win_height BY 25
21      GOTO 0 i
22      DRAWLINE win_width i
23  END
24  COLOR middle
25  GOTO win_width/2 0
26  DRAWLINE win_width/2 win_height
27  GOTO 0 win_height/2
28  DRAWLINE win_width win_height/2
29
30  -- Applying methods
31  TRANSLATE win_width/2 win_height/2
32  "GOTO -25 -100"
33  DRAWRECT 50 50
34  ROTATE 45
35  DRAWRECT 50 50
36  ROTATE 315
37  GOTO 50 50
38  COLOR GREEN
39  FILLRECT 50 50
40  "SHEAR -1 0"
41  COLOR BLACK
42  FILLRECT 50 50
43  CLEAR
44  TRANSLATE win_width/2 win_height/2
45  SCALE 2 2
46  "GOTO -50 50"
47  COLOR blue
48  FILLRECT 25 25
```

Figure 21: Excerpt from JDOR - JDOR_transform.rxj (full code in Appendix - A 1. JDOR-transform.rxj)

Through the “SHEAR” command objects / shapes can be shifted or skewed. Objects can be sheared in the x-direction and / or y-direction through their respective argument. For a shear in the x-direction only the x-coordinates change while the y-coordinates stay the same ($x' = x + Shx * y,\ y' = y$). By performing a shear in the y-direction, y-coordinates change while x-coordinates stay the same ($x' = x,\ y' = y + Shy * x$). When applying a shear in both directions, both coordinates change ($x' = x + Shx * y\ ,\ y' = y + Shy * x$).

With the “`SHEAR -1 0`” in line 40 only a shear in the x-direction is applied. The following rectangle “`FILLRECT 50 50`” appears shifted (see the black parallelogram in Figure 22. For example, the starting point set by the “`GOTO 50 50`” is changed to (0,50) (mathematical equation: $x' = 50 - 1 * 50 => x' = 0\ ,\ y' = 50$).

Through the following “`CLEAR`”-command in line 43 all transformations, including the “`SHEAR`” and “`TRANSLATE`” are reset. Therefore, the “`TRANSLATE`”-command must be reset in line 44.

The “`SCALE 2 2`” in line 45 applies a double-multiplier to all coordinate-related following commands (“x” and “y” coordinates). Due to this fact the “`GOTO -50 50`” in line 46 actually moves the starting point to (-100,100) and the “`FILLRECT 25 25`” in line 48 draws a rectangle with the size 50x50 (see the blue rectangle in Figure 22.

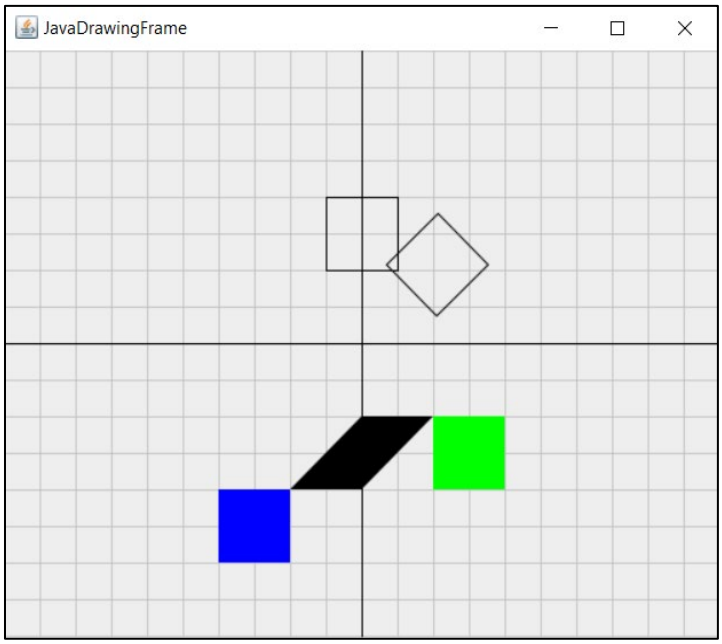

Figure 22: Output form JDOR-transform.rxj

6.6 Moving Objects in JDOR – JDOR-move.rxj

The following example shows how to “animate” objects in ooRexx through JDOR. Animating in this context means creating the appearance that an object is moving. In reality, however, an object is drawn repeatedly in such short time-intervals in different places, that it looks to a human observer as if this object is moving from one place to another.

The goal of this example is to show that simple animations can be created with ease through just one loop, in which only the coordinates of the drawings are changing. In addition, it shows how a “rebound effect” (redirecting an object after it collides with another object) can be achieved through simple calculations. The idea is that a circle, which is drawn in the middle of the window, starts moving in the direction of a random angle leaving a trail on its path.

To place a circle exactly in the middle of the window, the window-size and circle-diameter are needed. In this example the diameter of the circle is “50” (set by the variable “`ball_dia`” in line 8 in). In its essence a circle is an oval with identical width and height - so the “`FILLOVAL 50 50`” on line 20 draws a circle.

To give the circle a direction, trigonometric functions are needed. By default, ooRexx does not have any trigonometric functions, so they must be imported through a Java class. In line 4 the “`java.lang.Math`” class is imported through BSF4ooRexx 850 which allows the use of each method in this class via “`".calc~methodname`”. Respectively, this is used to convert the random number (“`RANDOM(1,360)`” – picks a number between 1 and 360) into radians and then to apply the cosines and sinus in the lines 21 to 23. The cosines and sinus are needed to calculate the movement in the “x” and “y” direction.

Constantly changing the same values is important to create animations. To achieve this, an endless loop is created with the “`DO FOREVER`” command. To get the current position on the window the “`GET STATE`” command is needed, which creates a table of information that is accessible through the “`RC`” variable. The coordinates can then be accessed through their respective index in the table (see “`rc["CURRX"]`” in line 28).

```
4   CALL bsf.import "java.lang.Math", "calc" --import Java.lang.Math as calc
5   --Setting variables
6   win_width = 500
7   win_height = 500
8   ball_dia = 50
9   speed = 5
10  colors = "red", "black", "green", "blue", "orange", "gray", "yellow"
11  --Create Window
12  WINSIZE win_width win_height
13  NEW win_width win_height
14  BACKGROUND white
15  CLEARRECT win_width win_height
16  WINSHOW
17
18  COLOR black
19  GOTO win_width/2-ball_dia/2 win_height/2-ball_dia/2 --  500/2 - 50/2 = 225
20  FILLOVAL 50 50
21  angle = .calc~toRadians(RANDOM(1,360)) --random angle between 1 and 360
22  MOVE_Y = .calc~cos(angle) * speed  --Cosines needs radians
23  MOVE_X = .calc~sin(angle) * speed  --Sinus needs radians
24  SAY angle
25  SAY MOVE_X " : "MOVE_Y
26  DO FOREVER -- loop will be executed forever till window is closed
27     getState
28     currX =  rc["CURRX"]
29     currY =  rc["CURRY"]
30     next = collison_detection(currX, currY, MOVE_X, MOVE_Y)
31     MOVE_X = next[1]
32     MOVE_Y = next[2]
41     COLOR next[3]
42     GOTO currX+MOVE_X currY+MOVE_Y
43     FILLOVAL  ball_dia ball_dia
46  END
47  sleep 4 -- line is never executed since loop is never exited
48  EXIT -- same as line 47 (exists only for consistency with other examples)
49
50  collison_detection: PROCEDURE EXPOSE win_width win_height ball_dia colors
53  PARSE ARG CurrX, CurrY, move_X, move_Y
57  IF CurrX + move_X < 0 THEN  --left Border
58   DO
59       move_X = -move_X
60       col = colors[RANDOM(1,colors~size)]
61   END
62  IF CurrX + move_X + ball_dia > win_width THEN  --right Border
63   DO
64       move_X = -move_X
65       col = colors[RANDOM(1,colors~size)]
66   END
67  IF CurrY + move_Y < 0 THEN  --top Border
68   DO
69       move_Y = -move_Y
70       col = colors[RANDOM(1,colors~size)]
71   END
72  IF CurrY + move_Y + ball_dia > win_height THEN  --bottom Border
73   DO
74       move_Y = -move_Y
75       col = colors[RANDOM(1,colors~size)]
76   END
77  erg = move_X, move_Y, col --Creating an array to return multiple values
78  RETURN erg
```

Figure 23: JDOR-move.rxj (partly – complete version in Appendix - A 2. JDOR-move.rxj)

The actual calculations in which direction the circle moves are done in the “collision_detection” method. The method is created in line 50 and called on line 30. The method uses the current location (“`currX`” and “`curry`”) and the movement parameters (“`MOVE_X`” and “`MOVE_Y`”) as arguments to calculate whether the next location would be outside the frame-borders. Through the “`EXPOSE`”-keyword it accesses the window-dimensions and circle-diameter (line 50) and returns two new directions and a colour (line 78).

For the top and left border this means that the current location plus the respective directional move would lead to a value below “0” (see line 57 and 67). Since the current location is always on the top left corner of the circle, the circles diameter must be added to the current location and the next moves direction when checking if the circle moves across the right and bottom border (see line 62 and 72).

If the method detects a collision than the respective direction is changed (e.g. for a top or bottom collision the “`y`”-direction is changed). Additionally, the colour for the circle is changed by picking a random colour out of the “`colours`”-array. The direction and colour are return via an array which is stored in the “next” variable (line 30). The program then uses these variables to calculate the next move and colour (line 31,32 and 41). After that, a new oval is drawn with “`FILLOVAL ball_dia ball_dia`” in line 43.

Figure 24 shows the output after a few seconds (left) and after two minutes (right). It can be seen that with each collision, the line changes its colour and direction.

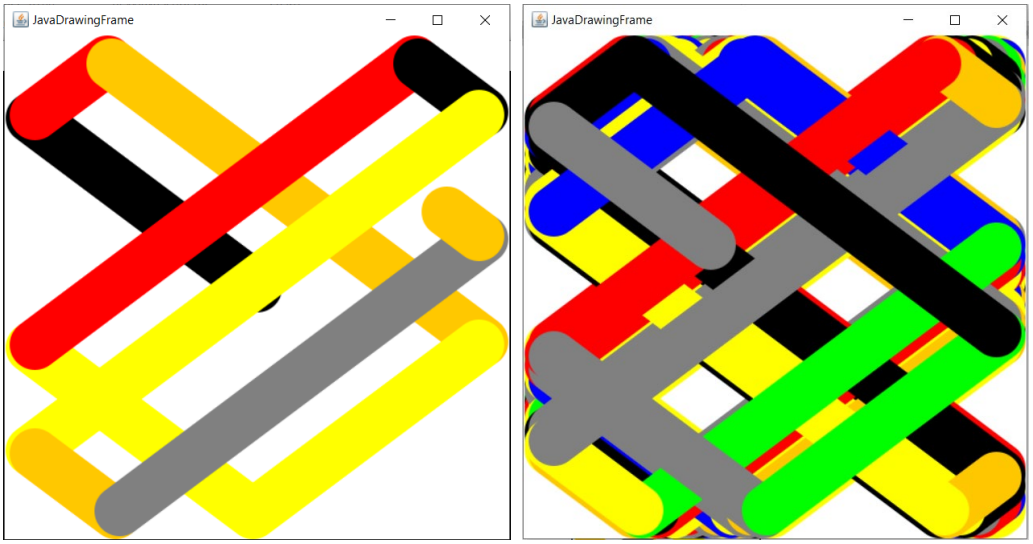

Figure 24: Output of JDOR-move.rxj after three seconds (left) and after two minutes (right)

6.7 Combining JDOR with JavaFX – JDORFX.rxj

This example shows how to combine JDOR with JavaFX. JavaFX is a system-independent lightweight platform that enables the creation of desktop applications and Rich Internet Applications (RIA) (Javatpoint, o.D.). FXML is a version of XML that is being used to create and represent its graphical user interface. Of essence is the fact it is detached from the actual program (Javatpoint, o.D.).

With JavaFX a small GUI with buttons and text fields is created which allows one to draw and move objects in JDOR. The GUI-elements can be defined in an FXML file or directly in the ooRexx-code. Due to the small number of elements used in the GUI, the elements were defined directly in ooRexx. Figure 25 shows how a button is created with an ID (`"setId("btnDraw")"`), position (`"setLayoutX()"` and `"setLayoutY()"`) and text (`"setText()"`). A similar syntax is used for the creation of the other elements as well.

```
73  btnDraw=.bsf~new("javafx.scene.control.Button")~~setId("btnDraw")-
74                        ~~setTextFill(colorClz~BLACK)               -
75                        ~~setLayoutX(460)                           -
76                        ~~setLayoutY(25)                            -
77                        ~~setText("Draw object")
```

Figure 25: Creating a JavaFX button in ooRexx – Excerpt from JDORFX.rxj

For each button an "`EventHandler`" is created which waits for a user-input (e.g. clicking the button) and then performs an action (e.g. drawing an object in JDOR).

Figure 26 shows how the "`EventHandler`" "RexxButtonDrawHandler" with multiple arguments ("`txtWidth`", "`txtHeight`",…) is created and how it is linked to the Draw-button with the command "`btnDraw~setOnAction(bhDraw)`".

```
106  bhDraw=BSFCreateRexxProxy(.RexxButtonDrawHandler~new(txtWidth,
     txtHeight, choice, Information), ,"javafx.event.EventHandler")
107  btnDraw~setOnAction(bhDraw)
```

Figure 26: Setting the EventHandler for the Draw-Button – excerpt from JDORFX.rxj

Figure 27 shows the logic of the "RexxbuttonDrawHandler". If the "Draw"-button is clicked, the handler accesses the texts from the "`txtwidth`", "`txtheight`" and "`choice`" elements ("`EXPOSE`" and "`USE ARG`" in lines 211,212 and 214) and saves it in the "`information`"-array (line 216 to 218).

To use JDOR functionalities in a new class, the current JDOR instance must be addressed first. This is done in line 215 and must be implemented in each new class. This allows multiple classes to access the same JDOR instance. Therefore, different "`EventHandlers`" have access to the same information regarding the current JDOR instance.

Through the "`PUSHIMAGE currentState`" in line 219 the current state of the frame (positions of the different objects) is saved as an image which can later be used to restore the state of the frame before the new object was drawn. Then, on line 221 the handler creates a drawing of the chosen object ("`choice~getValue()`") at position 50,50 with the entered size ("`txtWidth~getText`" and "`txtHeight~getText`").

```
209  ::class RexxButtonDrawHandler
210   ::method init
211     EXPOSE txtWidth txtHeight choice information
212     USE ARG txtWidth, txtHeight, choice, information
213   ::method handle -- will be invoked by the Java side
214     EXPOSE txtWidth txtHeight choice information
215    ADDRESS JDOR
216    information[1] = choice~getValue()
217    information[2] =  txtWidth~getText
218    information[3] =  txtHeight~getText
219    PUSHIMAGE currentState
220    GOTO 50 50
221    choice~getValue() txtWidth~getText txtHeight~getText
```

Figure 27: Implemeting the logic of an EventHandler - excerpt from JDORFX.rxj

The implementation of JavaFX in ooRexx uses an object-oriented approach, where every element is created in a class and each "EventHandler" is a class itself.

In total, the GUI has eleven elements. One selection box to set which object should be drawn next. Two text fields to enter the width and height of the object. One button to draw the selected object. One button to reset the last drawn object (deleting the object from the current frame). Four buttons to move the object. One text field to add a path if the image should be saved on the system and one button to save the image.

Figure 28 shows the resulting GUI with its elements and layout. Due to the lack of a colour selector, it is only possible to draw in black. A colour selector could be implemented in a similar way to the object selector, whereby different colour options are displayed after clicking on the selector.

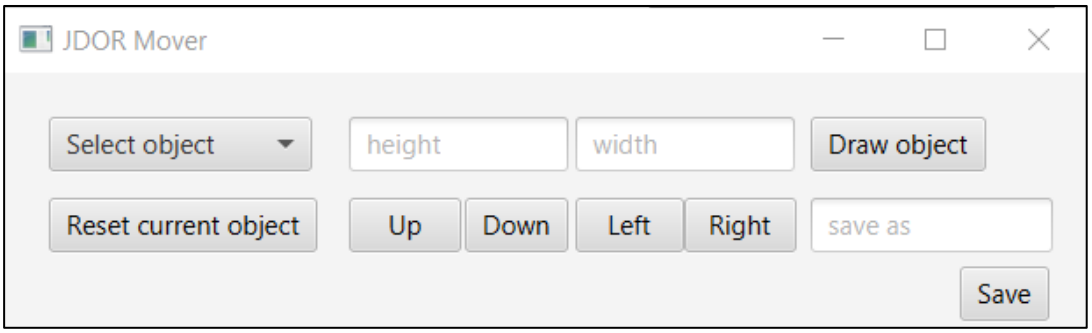

Figure 28: GUI created by JDORFX.rxj

To create the “moving”-effect with the four directions buttons, the object must be redrawn with each click and the old object must be erased. To erase the object, the old object must be painted over with a clear composition (line 140 and 141 in Figure 29). The composition sets the behaviour for overlapping objects. With the “`COMPOSITE src_over`” on line 142 the default composition is set so that new pixels are drawn on top of old pixels. The "`DRAWIMAGE currentState`" command is used to prevent newly painted objects from erasing old objects when moved, since all old objects are redrawn with each move.

With the “`information[1] information[2] information[3]`” line the contents of the “information”-array (Object, Width, Height) are used to draw an object, since the combination would form a string like “`DRAWRECT 60 60`”

```
129  ::class RexxButtonUpHandler
130  ::method init
131   EXPOSE information
132   USE ARG information
133  ::method handle -- will be invoked by the Java side
134   EXPOSE information
135   ADDRESS JDOR
136   GETSTATE
137   curr_x = rc["CURRX"]
138   curr_y = rc["CURRY"]
139   WINUPDATE .false
140   COMPOSITE clear
141   FILLRECT information[2]+1 information[3]+1
142   COMPOSITE src_over
143   GOTO 0 0
144   DRAWIMAGE currentState
145   GOTO curr_x curr_y-5
146   information[1] information[2] information[3]
147   WINUPDATE .true
```

Figure 29: Creating a movement effect - excerpt from JDORFX.rxj

The complete code can be found in the Appendix – in chapter A 3. JDORFX.rxj

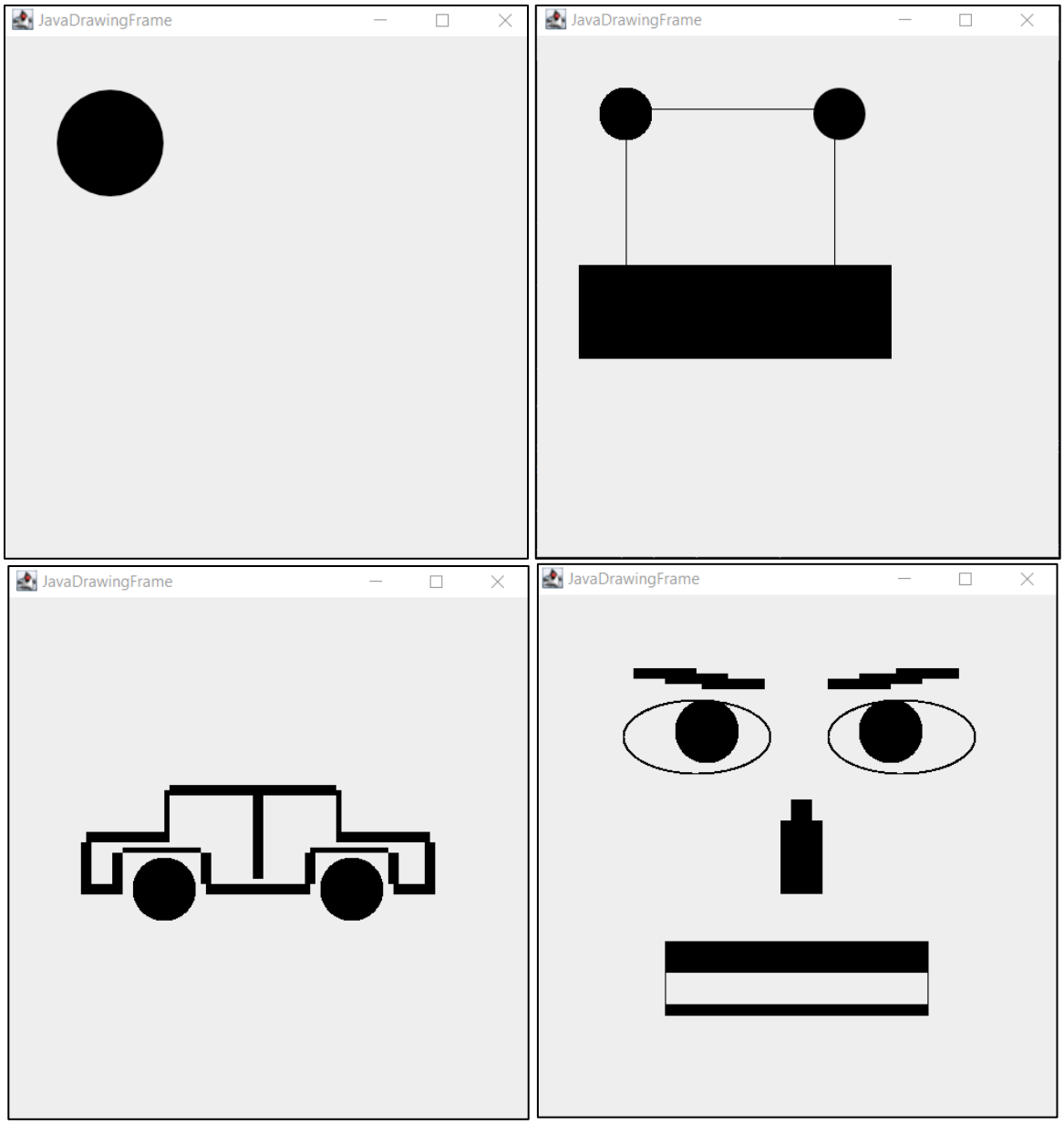

Figure 30: Drawings created by JAVA-FX GUI interaction

Figure 30 above shows some pictures created via the Java-FX GUI. When a new object is drawn, it is automatically placed at the location 50,50 (see upper left image in Figure 30). Although, the user can only choose between rectangles and ovals, it is still possible to create more detailed drawings.

For example, the bottom two pictures in Figure 30 show drawings of a car and a face by combining several different rectangles or ovals in different sizes.

While the drawings are not the most colourful nor complex it is still clear that it is possible to produce simple visuals that can be recognized by others.

6.8 Creating Arbitrary Shapes with JDOR – JDOR-shapes.rxj

This chapter takes a closer look at the shapes and Path2D commands added in the latest version of BSF4ooRexx. For this purpose, a small example will be created to show some of the functionalities.

Using the Shapes and Path2D commands, different shapes can be combined to create new completely and arbitrary ones. The resulting shape can then be drawn, filled and transformed.

Since the functions of "SHAPES" and "PATH2D" were only added with a version of BSF4ooRexx during the later stages of the writing process of this bachelor thesis, the commands were not described in chapter 5.3.1 JDOR – Commands. A complete description of the commands can be found in the "JDOR_doc.html" file in the information-folder in the BSF4ooRexx installation-folder or on the website of WU's "Information Systems & Society" institute (https://wi.wu.ac.at/rgf/rexx/misc/jdor_doc.tmp/jdor_doc.html#cmdNew).

Even though the "`SHAPES`" function is very similar to the objects shown in the previous chapters (e.g. "`DRAWRECT`", "`DRAWARC`"), they still differ in syntax and structure. A "`PATH2D`" object is in itself a shape that can consist of different elements and "`SHAPES`", which together form a new individual "`SHAPE`".

In contrast to the standard "`DRAW`" commands, "`SHAPES`" are not drawn immediately, but are first created and then drawn with an additional command. The "`GOTO x y`" is also omitted, as the starting point is directly set when the "`SHAPE`" is created (e.g. "`SHAPE my_rect Rectangle x y width height`"). The object can then be drawn via the "`DRAWSHAPE my_rect`"-command, which will draw the "`my_rect`" object at the given location (x,y).

In this example rectangles, arcs and ellipses are combined to create two "`PATH2D`" objects which together form a banner / image. The aim is to show that with "`SHAPES`" and "`PATH2D`" complex drawings can be created in just a few lines of code.

```
7   STROKE str3 3
8   STROKE str1 1
9   --TranslateX TranslateY ScaleX ScaleY ShearX ScaleY
10  "TRANSFORM trans1 . . . . -0.1 -0.25"
11  "TRANSFORM trans2 . . . . 0.1 0.25"
12
13  --Creating the different shapes
14  SHAPE path1 path
15  SHAPE path2 path
16  SHAPE rect_bg Rectangle 100 50 300 400
17  SHAPE rect_right Rectangle 275 150 50 75
18  SHAPE rect_left  Rectangle 188 22 50 75
19  SHAPE rect_test Rectangle 100 100 200 200
20  SHAPE circle Ellipse 225 200 50 50
21  SHAPE arc_right  arc2d 350 350 50 50  90 90 open
22  "SHAPE arc_left   arc2d 100 350 50 50  90 -90 open"
23
24  --Creating the path1 shape (appending lines and shapes)
25  PATHMOVETO path1 250 250
26  PATHLINETO path1 350 350
27  PATHAPPEND path1 arc_right
28  PATHLINETO path1 350 350
29  PATHMOVETO path1 250 250
30  PATHAPPEND path1 circle
31  PATHLINETO path1 225 225
32  PATHMOVETO path1 250 250
33  PATHLINETO path1 150 350
34  PATHAPPEND path1 arc_left
35  PATHLINETO path1 150 350
36  PATHMOVETO path1 250 250
37  PATHLINETO path1 250 200
38  PATHLINETO path1 250 300
39  PATHQUADTO path1 200 280 250 400
40  PATHQUADTO path1 300 280 250 300
41  PATHLINETO path1 250 175
42
43  --Creating the path2 shape (using transformation)
44  PATHITERATOR rect_right trans1 --applying transform from line 11
45  PATHAPPEND path2 "rc" --transformed shape is saved in rc
46  PATHLINETO path2 250 175
47  PATHMOVETO path2 189.3 69.75
48  PATHITERATOR rect_left trans2 --applying transform from line 12
49  PATHAPPEND path2 "rc" --transformed shape is saved in rc
50  PATHMOVETO path2 239.3 82.25
51  PATHLINETO path2 250 175
52  PATHMOVETO path2 260 81.25
53  PATHLINETO path2 250 175
54
55  --Drawing the shapes
56  COLOR white
57  FILLSHAPE rect_bg
58  COLOR red
59  FILLSHAPE path1
60  COLOR orange
61  FILLSHAPE path2
62  STROKE str3
63  COLOR black
64  DRAWSHAPE path1
65  DRAWSHAPE rect_bg
66  DRAWSHAPE path2
```

Figure 31: Creating shapes - JDOR-shapes.rxj (excerpt – full code in Appendix in A 4. JDOR – shapes.rxj)

Figure 31 on the page above shows the code for this example. In line 10 and 11 two transformations with a shear effect are set. The syntax follows "`TRANSFORM nickname translateX translateY scaleX scaleY shearX shearY`". A single dot means that the value has not been changed and the current value is being used – in this case the default value.

Shapes are all created with the "SHAPE" keyword followed by a nickname, type and attributes. For example, an arc at the location 350,350 with a width and height of 50 and an open arch with 90° is created with the command "`SHAPE arc_right arc2d 350 350 50 50 90 90 open`" in line 21.

After a shape is created, it can either be drawn directly (e.g. "`DRAWSHAPE rect_bg`" on line 65) or it can be appended to a path (line 27).

With the "`PATHMOVETO path1 250 250`" in line 25 the current position of "path1" is set to 250,250. In comparison, the "`PATHLINETO path1 350 350`" will also set the current position to 350,350 and will draw a line from the old position to the new position.

With the "`PATHAPPEND path1 arc_right`" on line 27 the previously created "`arc_right`" shape is appended to "path1". When appending shapes, it is crucial to know where JDOR starts drawing these shapes. For example, the starting-point of an ellipse is at its 3 o'clock position – therfore a line will automatically be drawn to this location before drawing the ellipse.

The "`PATHQUADTO path1 200 280 250 400`" in line 39 is used to draw a quadratic-curve to a new position (in this case 200,280). On its "route" from the current to the new position it will touch the second point (in this case 250,400).

To apply a transformation to a shape, the "`PATHITERATOR`"-command can be used. On line 39 the transformation "`trans1`" is applied to the shape "`rect_right`". To append the transformed shape to a path the "`rc`" variable needs to be used instead of the shapes name (e.g "`PATHAPPEND path1 "rc"`" in line 45).

To draw or fill a shape the "DRAWSHAPE" or "FILLSHAPE" followed by the nickname of the needed shape can be used. A colour or stroke-type can only be applied to a whole shape. So, if a larger object with multiple colours is needed, the different sections must be split into different shapes.

Due to this fact, the drawing was split into three shapes. A white background created by the shape "rect_bg", The red lower part of the banner created by "path1" and the orange upper part created by "path2".

Figure 32 shows the resulting image of "JDOR-shapes.rxj" All enclosed parts of the "path1" shape are filled with a red colour. Also, the shear-effect of both transformations for the shape "path" created the two slightly angled rectangles at the upper part of the image is visible.

As with all the other commands, JDOR always draws one layer above the other. So, the lines in the red circle in the middle are only visible since the "DRAWSHAPE path1" command was executed after the "FILLSHAPE path1" command. Otherwise, the "FILLSHAPE" command would have painted over the black lines.

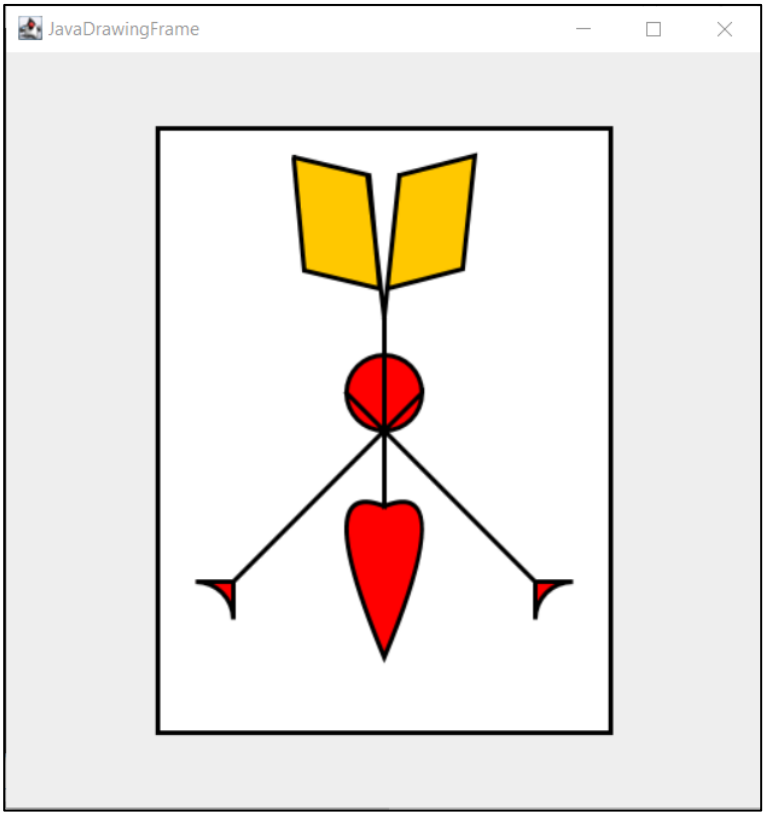

Figure 32: Banner created by JDOR-shapes.rxj

7. Conclusion

By no means are Java and the 20-year-older ooRexx the most recent programming languages available. But, whereas Java is still widely used and appreciated in the development community, ooRexx has lost its popularity over time.

The most recent version of ooRexx, however, was only made available in December of 2022 and through extensions, notably BSF4ooRexx 850, the number of functionalities has increased significantly.

Allowing programs to use the complex features of the Java programming language in a simple-to-read programming language like ooRexx is especially beneficial for beginners, but it also has advantages for experienced developers.

The syntax and structure of ooRexx, like many other programming languages available, may take some getting accustomed to. However, results are often attained rather rapidly, and the usage is typically rewarding.

JDOR, the practical subject of this thesis, is simple to learn and gives programmers the ability to create basic drawings after a little learning period, especially when combined with its documentation. In particular, beginners may learn about graphic design by employing the various elements and engaging in active learning. Through the latest addition, “`SHAPES`” and “`PATH2D`”, even complex drawings can be created with ease.

Unfortunately, the lack of online support is one of ooRexx's greatest drawbacks. In most cases, it is up to the individual to identify their errors. Therefore, even though knowledge of Java is not required to utilize the BSF4ooRexx 850 framework, it undoubtedly has certain benefits to know how to solve issues in Java. Additionally, "successors" to "AWTs" "`Graphics`" and "`Graphics2D`," such as "Swing" and "JavaFX," are generally more popular and better suited in their respective sectors.

However, one should not undervalue JDOR's potential. JDOR, like JavaFX to a certain extent, are fascinating additions for students and encourage experimentation if there is an underlying interest in drawing or graphic design. Nevertheless, it is debatable whether a course needs to have a distinct JDOR unit. A few slides outlining the fundamental ideas, would suffice to spark some students' curiosity on this topic.

A certain excitement of exploration may emerge rather fast, especially if one goes deeper into the subject and tries out the other commands in the JDOR documentation.

Even though it was occasionally difficult, the author of this bachelor thesis generally enjoyed the designing and creation-process of the provided examples.

In conclusion, it is debatable to what degree learning the "`Graphics`" and "`Graphics2D`" components via JDOR in ooRexx provides a graduate with a competitive edge in the workforce. However, the AWT package serves as the foundation upon which the other graphical Java classes, like Swing and JavaFX, are build on, and as such, it provides a useful look behind the scene. Hence, the author believes that learning JDOR is especially worthwhile for beginners. But rather than adding a new course subject, it would probably be sufficient to provide students with slides that they could read on their own.

With these slides as a basis and enough creativity students are able to create numerous drawings and may also awaken the interest in more complex design classes or frameworks.

Appendix

This chapter contains the complete code for the examples that were only partly shown in the thesis.

A 1. JDOR-transform.rxj

Figure 32 shows the complete code from example JDOR-transform.rxj shown and described in chapter ***6.5 Rotate, Scale, Translate and Shear in JDOR***

```
CALL addJdorHandler -- load
ADDRESS jdor -- set default environment to JDOR

--Creating and showing a new window
win_width = 500
win_height = 400
WINSIZE win_width win_height
WINSHOW

--setting the colors
COLOR coordinate_system 190 190 190 200
COLOR middle 0 0 0 255

--Drawing the system
COLOR coordinate_system
DO i=0 TO win_width BY 25
    GOTO i 0
    DRAWLINE i win_height
END
DO i=0 TO win_height BY 25
    GOTO 0 i
    DRAWLINE win_width i
END
COLOR middle
GOTO win_width/2 0
DRAWLINE win_width/2 win_height
GOTO 0 win_height/2
DRAWLINE win_width win_height/2

-- Applying methods
TRANSLATE win_width/2 win_height/2
"GOTO -25 -100"
DRAWRECT 50 50
ROTATE 45
DRAWRECT 50 50
ROTATE 315
GOTO 50 50
COLOR GREEN
FILLRECT 50 50
"SHEAR -1 0"
COLOR BLACK
FILLRECT 50 50
CLEAR
TRANSLATE win_width/2 win_height/2
SCALE 2 2
"GOTO -50 50"
COLOR blue
FILLRECT 25 25

sleep 400
::REQUIRES "jdor.cls"
```

Figure 33: JDOR-transform.rxj

A 2. JDOR-move.rxj

Figure 34 shows the complete code of the “JDOR-move.rxj” example shown in chapter ***6.6 Moving Objects in JDOR – JDOR-move.rxj.*** In addition to the code shown in the chapter the complete code contains a commented section (line 34 to 40) which, if included, would erase the trail left by the circle.

```
1   CALL addJdorHandler -- load the JDOR handler
2   ADDRESS jdor -- set default environment to JDOR
3
4   CALL bsf.import "java.lang.Math", "calc" --import Java.lang.Math as calc
5   --Setting variables
6   win_width = 500
7   win_height = 500
8   ball_dia = 50
9   speed = 5
10  colors = "red", "black", "green", "blue", "orange", "gray", "yellow"
11  --Create Window
12  WINSIZE win_width win_height
13  NEW win_width win_height
14  BACKGROUND white
15  CLEARRECT win_width win_height
16  WINSHOW
17
18  COLOR black
19  GOTO win_width/2-ball_dia/2 win_height/2-ball_dia/2 --  500/2 - 50/2 =
20  FILLOVAL 50 50
21  angle = .calc~toRadians(RANDOM(1,360)) --random angle between 1 and 360
22  MOVE_Y = .calc~cos(angle) * speed  --Cosines needs radians
23  MOVE_X = .calc~sin(angle) * speed  --Sinus needs radians
24  SAY angle
25  SAY MOVE_X " : "MOVE_Y
26  DO FOREVER -- loop will be executed forever till window is closed
27   getState
28   currX =  rc["CURRX"]
29   currY =  rc["CURRY"]
30   next = collison_detection(currX, currY, MOVE_X, MOVE_Y)
31   MOVE_X = next[1]
32   MOVE_Y = next[2]
33   WINUPDATE .false
34   --COMPOSITE clear
35   --FILLRECT ball_dia ball_dia
36   --COMPOSITE src_over
37   --To clear the window before drawing a new circle
38   --It would appear as the circle is moving without a trail
39   --GOTO 0 0
40      --CLEARRECT ball_dia ball_dia
41   COLOR next[3]
42   GOTO currX+MOVE_X currY+MOVE_Y
43   FILLOVAL  ball dia ball dia
```

```
44      WINUPDATE .true
45      --sleep 0.01
46  END
47  sleep 4 -- line is never executed since loop is never exited
48  EXIT -- same as line 47 (exists only for consistency with other examples)
49
50  collison_detection: PROCEDURE EXPOSE win_width win_height ball_dia colors
51  --"EXPOSE" makes public variables accessible in the method
52  -- -> changing them in the method would also change them everywhere else
53  PARSE ARG CurrX, CurrY, move_X, move_Y
54
55  -- angle of entrence = angle of exit
56  -- Only changing one direction (direction * -1) while the other stays
57  IF CurrX + move_X < 0 THEN  --left Border
58      DO
59          move_X = -move_X
60          col = colors[RANDOM(1,colors~size)]
61      END
62  IF CurrX + move_X + ball_dia > win_width THEN  --right Border
63      DO
64          move_X = -move_X
65          col = colors[RANDOM(1,colors~size)]
66      END
67  IF CurrY + move_Y < 0 THEN  --top Border
68      DO
69          move_Y = -move_Y
70          col = colors[RANDOM(1,colors~size)]
71      END
72  IF CurrY + move_Y + ball_dia > win_height THEN  --bottom Border
73      DO
74          move_Y = -move_Y
75          col = colors[RANDOM(1,colors~size)]
76      END
77  erg = move_X, move_Y, col --Creating an array to return multiple values
78  RETURN erg --returning the array
79
80  ::requires "jdor.cls"
```

Figure 34: JDOR-move.rxj

To achieve this, each time the circle is moved, the complete image is first painted over with the background colour. This would also paint over any previously drawn circle.

After that, the circle is repainted on a new location. To make this process look more fluent for the viewer the window wont physically update (line 33,44) during these steps which should give the impression that these intermediate steps do not take place

A 3. JDORFX.rxj

The complete code for the program shown in chapter 6.7 Combining JDOR with JavaFX – JDORFX.rxj is shown in Figure 34 over the next pages. In its essence, all the parts, that are a new addition from the previous chapters were described in chapter 6.7.

More information is provided for the button handlers for the direction changes (Up, Down, Left, and Right). In these, the old object is entirely painted over using the command "`FILLRECT information[2]+1 information[3]+1`" before the corresponding object is redrawn in the appropriate direction (+/- 5). "`FILLRECT`" is utilized because, despite whether it is a circle or a rectangle, it can truly paint over any form due to its wider height and width.

The “+1” in the “`FILLRECT`” width and height is used since the dimension of a rectangle drawn with the “`FILLRECT`” command is one point smaller than the given width and height. So, the size of a rectangle drawn by “`FILLRECT 300 300`” is in fact only 299x299. The "FILLRECT" command, in a sense, paints the region inside an invisible rectangle without touching the borders. A rectangle drawn by “DRAWRECT 300 300” has a size of 300x300.

So, by giving the “`FILLRECT`”-command a larger size (+1) it can paint over every object (oval and rectangle), regardless if it was drawn by a “`DRAW`” or “`FILL`” command

```
CALL addJdorHandler -- load
ADDRESS jdor -- set default environment to JDOR

 NEW 500 500
 WINSHOW

rexxHandler=.RexxAppHandler~new -- create Rexx object that will control the FXML
 -- instantiate the abstract JavaFX class
rxApp=BSFCreateRexxProxy(rexxHandler, ,"javafx.application.Application")
 -- launch the application, invoke "start" and then stay up until the application
rxApp~launch(rxApp~getClass, .nil)
::REQUIRES "BSF.CLS" -- get Java support
::REQUIRES "JDOR.CLS" -- get JDOR support


::CLASS RexxAppHandler -- the Rexx handler for javafx.application.Application
::METHOD start -- will be called by JavaFX, allows to setup everything
 USE ARG primaryStage, questions
 ADDRESS JDOR
 primaryStage~setTitle("JDOR Mover") -- setting the name of the JavaFX frame
 colorClz=bsf.loadClass("javafx.scene.paint.Color") -- access to JavaFX colors


information = "Object","Width","Height"
 --wT = bsf.loadClass("javafx.scene.control.Labeled.wrapText")

 --PANE:
 root=.bsf~new("javafx.scene.layout.AnchorPane") -- create the root node
 root~prefHeight=150
 root~prefWidth=620


choice = .bsf~new("javafx.scene.control.ComboBox")~~setId("choice") -
                     ~~setLayoutX(26) -
                     ~~setLayoutY(25) -
                     ~~setPrefWidth(150) -
                     ~~setPromptText("Select object")
choice~getItems()~add("FILLOVAL")
choice~getItems()~add("FILLRECT")
choice~getItems()~add("DRAWOVAL")
choice~getItems()~add("DRAWRECT")

btnUP=.bsf~new("javafx.scene.control.Button")~~setId("btnUP")          -
                     ~~setTextFill(colorClz~BLACK)                     -
                     ~~setLayoutX(197)                                 -
                    ~~setLayoutY(71)                                   -
                    ~~setText("   Up   ")
btnDown=.bsf~new("javafx.scene.control.Button")~~setId("btnDown")      -
                     ~~setTextFill(colorClz~BLACK)                     -
                     ~~setLayoutX(263)                                 -
                     ~~setLayoutY(71)                                  -
                     ~~setText("Down")
 btnLeft=.bsf~new("javafx.scene.control.Button")~~setId("btnLeft")     -
                     ~~setTextFill(colorClz~BLACK)                     -
                     ~~setLayoutX(326)                                 -
                     ~~setLayoutY(71)                                  -
                     ~~setText("  Left  ")
 btnRight=.bsf~new("javafx.scene.control.Button")~~setId("btnRight")   -
                     ~~setTextFill(colorClz~BLACK)                     -
                     ~~setLayoutX(388)                                 -
                     ~~setLayoutY(71)                                  -
                     ~~setText(" Right ")
```

```
 btnReset=.bsf~new("javafx.scene.control.Button")~~setId("btnReset")     -
                      ~~setTextFill(colorClz~BLACK)                      -
                      ~~setLayoutX(26)                                   -
                      ~~setLayoutY(71)                                   -
                      ~~setText("Reset current object")
 btnSave=.bsf~new("javafx.scene.control.Button")~~setId("btnSave")     -
                      ~~setTextFill(colorClz~BLACK)                      -
                      ~~setLayoutX(545)                                  -
                      ~~setLayoutY(110)                                  -
                      ~~setText("Save")
btnDraw=.bsf~new("javafx.scene.control.Button")~~setId("btnDraw")     -
                      ~~setTextFill(colorClz~BLACK)                      -
                      ~~setLayoutX(460)                                  -
                      ~~setLayoutY(25)                                   -
                      ~~setText("Draw object")
 txtHeight = .bsf~new("javafx.scene.control.TextField")~~setId("txtHeight")-
                      ~~setPromptText("height")-
                      ~~setLayoutX(197)-
                      ~~setLayoutY(25)-
                      ~~setPrefHeight(26)-
                      ~~setPrefWidth(125)
  txtWidth = .bsf~new("javafx.scene.control.TextField")~~setId("txtWidth")-
                      ~~setPromptText("width")-
                      ~~setLayoutX(326)-
                      ~~setLayoutY(25)-
                      ~~setPrefHeight(26)-
                      ~~setPrefWidth(125)
  txtSave = .bsf~new("javafx.scene.control.TextField")~~setId("txtSave")-
                      ~~setPromptText("save as")-
                      ~~setLayoutX(460)-
                      ~~setLayoutY(71)-
                      ~~setPrefHeight(26)-
                      ~~setPrefWidth(138)

--BUTTONHANDLER
bhUp=BSFCreateRexxProxy(.RexxButtonUpHandler~new(information),
,"javafx.event.EventHandler")
btnUP~setOnAction(bhUp)
bhDown=BSFCreateRexxProxy(.RexxButtonDownHandler~new(information),
,"javafx.event.EventHandler")
btnDown~setOnAction(bhDown)
bhLeft=BSFCreateRexxProxy(.RexxButtonLeftHandler~new(information),
,"javafx.event.EventHandler")
btnLeft~setOnAction(bhLeft)
bhRight=BSFCreateRexxProxy(.RexxButtonRightHandler~new(information),
,"javafx.event.EventHandler")
btnRight~setOnAction(bhRight)
bhDraw=BSFCreateRexxProxy(.RexxButtonDrawHandler~new(txtWidth,
txtHeight, choice, Information), ,"javafx.event.EventHandler")
btnDraw~setOnAction(bhDraw)
bhReset=BSFCreateRexxProxy(.RexxButtonResetHandler~new(information),
,"javafx.event.EventHandler")
btnReset~setOnAction(bhReset)
bhSave=BSFCreateRexxProxy(.RexxButtonSaveHandler~new(txtSave),
,"javafx.event.EventHandler")
btnSave~setOnAction(bhSave)
--Adding the elements to the AnchorPane
root~getChildren~~add(btnUP)   -
~~add(btnDown)   -
~~add(choice)    -
~~add(btnLeft)   -
~~add(btnRight) -
```

```
~~add(btnReset)-
~~add(btnSave) -
~~add(btnDraw) -
~~add(txtHeight)-
~~add(txtWidth)-
~~add(txtSave)


 -- put the scene on the stage (using AnchorPane's height and width)
 primaryStage~setScene(.bsf~new("javafx.scene.Scene", root))
 primaryStage~show
 ::CLASS RexxButtonUpHandler
 ::METHOD init
  EXPOSE information
  USE ARG information
 ::METHOD handle -- will be invoked by the Java side
   EXPOSE information
   ADDRESS JDOR
   GETSTATE
   curr_x = rc["CURRX"]
   curr_y = rc["CURRY"]
   WINUPDATE .false
   COMPOSITE clear
   FILLRECT information[2]+1 information[3]+1
   COMPOSITE src_over
   GOTO 0 0
   DRAWIMAGE currentState
   GOTO curr_x curr_y-5
   information[1] information[2] information[3]
   WINUPDATE .true

  ::CLASS RexxButtonDownHandler
 ::METHOD init
  expose information
  use arg information
 ::METHOD handle -- will be invoked by the Java side
   EXPOSE information
   ADDRESS JDOR
   GETSTATE
   curr_x = rc["CURRX"]
   curr_y = rc["CURRY"]
   WINUPDATE .false
   COMPOSITE clear
   FILLRECT information[2]+1 information[3]+1
   COMPOSITE src_over
   GOTO 0 0
   DRAWIMAGE currentState
   GOTO curr_x curr_y+5
   information[1] information[2] information[3]
   WINUPDATE .true

 ::CLASS RexxButtonLeftHandler
 ::METHOD init
  EXPOSE information
  USE ARG information
 ::METHOD handle -- will be invoked by the Java side
   EXPOSE information
   ADDRESS JDOR
   GETSTATE
   curr_x = rc["CURRX"]
   curr_y = rc["CURRY"]
   WINUPDATE .false
```

```
180     COMPOSITE clear
181     FILLRECT information[2]+1 information[3]+1
182     COMPOSITE src_over
183     GOTO 0 0
184     DRAWIMAGE currentState
185     GOTO curr_x-5 curr_y
186     information[1] information[2] information[3]
187     WINUPDATE .true
188
189   ::CLASS RexxButtonRightHandler
190    ::METHOD init
191     EXPOSE information
192     USE ARG information
193    ::METHOD handle -- will be invoked by the Java side
194     EXPOSE information
195     ADDRESS JDOR
196     GETSTATE
197     curr_x = rc["CURRX"]
198     curr_y = rc["CURRY"]
199     WINUPDATE .false
200     COMPOSITE clear
201     FILLRECT information[2]+1 information[3]+1
202     COMPOSITE src_over
203     GOTO 0 0
204     DRAWIMAGE currentState
205     GOTO curr_x+5 curr_y
206     information[1] information[2] information[3]
207     WINUPDATE .true
208
209   ::CLASS RexxButtonDrawHandler
210    ::METHOD init
211      EXPOSE txtWidth txtHeight choice information
212      USE ARG txtWidth, txtHeight, choice, information
213    ::METHOD handle -- will be invoked by the Java side
214      EXPOSE txtWidth txtHeight choice information
215      ADDRESS JDOR
216      information[1] = choice~getValue()
217      information[2] =  txtWidth~getText
218      information[3] =  txtHeight~getText
219      PUSHIMAGE currentState
220      GOTO 50 50
221      choice~getValue() txtWidth~getText txtHeight~getText
222
223   ::CLASS RexxButtonResetHandler
224    ::METHOD init
225      EXPOSE information
226      USE ARG information
227    ::METHOD handle -- will be invoked by the Java side
228     EXPOSE information
229     ADDRESS JDOR
230     WINUPDATE .false
231     COMPOSITE clear
232     FILLRECT information[2]+1 information[3]+1
233     COMPOSITE src_over
234     GOTO 0 0
235     DRAWIMAGE currentState
236     WINUPDATE .true
237
```

```
238  ::CLASS RexxButtonSaveHandler
239  ::METHOD init
240    EXPOSE txtSave
241    USE ARG txtSave
242  ::METHOD handle  -- will be invoked by the Java side
243    EXPOSE txtSave
244    ADDRESS JDOR
245    SAVEIMAGE txtSave~getText
```

Figure 35: JDORFX.rxj

A 4. JDOR – shapes.rxj

Figure 36 contains the full code of the “JDOR-shapes.rxj” examples shown in chapter ***6.8 Creating arbitrary Shapes with JDOR – JDOR-shapes.rxj.***

In addition to the code shown in the chapter Figure 36 includes the lines 1 to 6 where the JDOR-handler is loaded and a window with the size 500x500 is created

The structure and logic of the example was described in the chapter.

```
1   CALL addJdorHandler -- load
2    ADDRESS jdor -- set default environment to JDOR
3
4    NEW 500 500
5    WINSHOW
6
7    STROKE str3 3
8    STROKE str1 1
9    --TranslateX TranslateY ScaleX ScaleY ShearX ScaleY
10   "TRANSFORM trans1 . . . . -0.1 -0.25"
11   "TRANSFORM trans2 . . . . 0.1 0.25"
12
13   --Creating the different shapes
14   SHAPE path1 path
15   SHAPE path2 path
16   SHAPE rect_bg Rectangle 100 50 300 400
17   SHAPE rect_right Rectangle 275 150 50 75
18   SHAPE rect_left  Rectangle 188 22 50 75
19   SHAPE rect_test Rectangle 100 100 200 200
20   SHAPE circle Ellipse 225 200 50 50
21   SHAPE arc_right  arc2d 350 350 50 50  90 90 open
22   "SHAPE arc_left     arc2d 100 350 50 50  90 -90 open"
23
24   --Creating the path1 shape (appending lines and shapes)
25   PATHMOVETO path1 250 250
26   PATHLINETO path1 350 350
27   PATHAPPEND path1 arc_right
28   PATHLINETO path1 350 350
29   PATHMOVETO path1 250 250
30   PATHAPPEND path1 circle
31   PATHLINETO path1 225 225
32   PATHMOVETO path1 250 250
33   PATHLINETO path1 150 350
34   PATHAPPEND path1 arc_left
35   PATHLINETO path1 150 350
36   PATHMOVETO path1 250 250
37   PATHLINETO path1 250 200
38   PATHLINETO path1 250 300
39   PATHQUADTO path1 200 280 250 400
40   PATHQUADTO path1 300 280 250 300
41   PATHLINETO path1 250 175
42
```

```
43  --Creating the path2 shape (using transformation)
44   PATHITERATOR rect_right trans1 --applying transform from line 11
45   PATHAPPEND path2 "rc" --transformed shape is saved in rc
46   PATHLINETO path2 250 175
47   PATHMOVETO path2 189.3 69.75
48   PATHITERATOR rect_left trans2 --applying transform from line 12
49   PATHAPPEND path2 "rc" --transformed shape is saved in rc
50   PATHMOVETO path2 239.3 82.25
51   PATHLINETO path2 250 175
52   PATHMOVETO path2 260 81.25
53   PATHLINETO path2 250 175
54
55   --Drawing the shapes
56   COLOR white
57   FILLSHAPE rect_bg
58   COLOR red
59   FILLSHAPE path1
60   COLOR orange
61   FILLSHAPE path2
62   STROKE str3
63   COLOR black
64   DRAWSHAPE path1
65   DRAWSHAPE rect_bg
66   DRAWSHAPE path2
67   SLEEP 15
68   ::REQUIRES "JDOR.CLS"
```

Figure 36: JDOR-shapes.rxj

References

Bahtnagar, A. (2022). *The complete history of Java programming language*. www.geeksforgeeks.com. Retrieved 09.12.2022 from https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/

Brihadiswaren, G. (2020). *A performance comparison between C, Java, and Python*. www.medium.com. Retrieved 09.12.2022 from https://medium.com/swlh/a-performance-comparison-between-c-java-and-python-df3890545f6d

Britannica. (2022). *Java*. Encyclopidia Britannica. Retrieved 09.12.2022 from https://www.britannica.com/technology/Java-computer-programming-language

Codeacademy. (o.D.). *Scope in Java*. www.codeacedemy.com. Retrieved 11.12.2022 from https://www.codecademy.com/article/variable-scope-in-java

Day, B. (1998). *Getting started with Java 2D*. JavaWorld. Retrieved 26.12.2022 from https://www.infoworld.com/article/2076715/getting-started-with-java-2d.html

Flatscher, R. G. (2012). Automatisierung mit ooRexx und BSF4ooRexx. 1-12.

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

Flatscher, R. G. (2017). Automatisierungssprache Open Object Rexx 5.0 vor der Tür. *iX*, *11*, 66-70.

Flatscher, R. G. (2021). BSF4ooRexx 6.41 Going GA. 1-39.

Flatscher, R. G. (2022a). *BSF4ooRexx: From 641 GA Update to 850 Beta* International RexxLA Symposium 2022-09, https://www.rexxla.org/presentations/2022/202209_B4r641_to_B4r850.pdf

Flatscher, R. G. (2022b). *BSF4ooRexx: Introducing the JDOR Rexx Command*

Handler for Easy Creation of Bitmaps and

Bitmap Manipulations on Windows, Mac and Linux International RexxLA Symposium, 2022-09, https://www.rexxla.org/presentations/2022/202209_JDOR_command_handler.pdf

geeksforgeeks. (2022). *Differences between Procedural and Object Oriented Programming*. geekforgeeks. Retrieved 07.12.2022 from https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/#:~:text=Object%2Doriented%20programming%20provides%20data,of%20data%20hiding%20and%20inheritance.

GitHub.io. (2022). *PYPL PopularitY of Programming Language*. Retrieved 06.12.2022 from https://pypl.github.io/PYPL.html

Javatpoint. (o.D.). *JavaFX Tutorial*. Retrieved 20.01.2023 from https://www.javatpoint.com/javafx-tutorial

javatpoint.com. (o.D.). *Difference between procedural programming and object-oriented programming*. www.javatpoint.com. Retrieved 10.12.2022 from https://www.javatpoint.com/procedural-programming-vs-object-oriented-programming

Oracle. (2022). *Oracle Java SE Support Roadmap*. www.oracle.com. Retrieved 13.12.2022 from https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Oracle. (o.D.-a). *Class Graphics2D*. Oracle. Retrieved 27.12.2022 from https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html

Oracle. (o.D.-b). *Coordinates*. Oracle. Retrieved 27.12.2022 from https://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html
Oracle. (o.D.-c). *Lesson: Getting Started with Graphics*. Oracle. Retrieved 28.12.2022 from https://docs.oracle.com/javase/tutorial/2d/basic2d/index.html
Oracle. (o.D.-d). *Lesson: Overview of the Java 2D API Concepts*. www.docs.oracle.com. Retrieved 25.12.2022 from https://docs.oracle.com/javase/tutorial/2d/overview/index.html
Schäling, B. (2010). *Programmieren in Java: Aufbau*. www.highscore.de.
Sharma, S. (2022). *Just in Time compiler*. Retrieved 09.12.2022 from https://www.geeksforgeeks.org/just-in-time-compiler/
SlashData. (2022). State of the Developer Nation 22nd Edition. *22*. Retrieved 30.01.2023, from https://slashdata-website-cms.s3.amazonaws.com/sample_reports/VZtJWxZw5Q9NDSAQ.pdf
Stenberg, D. (o.D.). *What is cURL?* www.curl.se. Retrieved 20.12.2022 from https://curl.se/docs/faq.html#What_is_cURL
Sun-Microsystems. (1999). *1.2 Rendering Model*. Nickerson Group at University of Washington. Retrieved 27.12.2022 from https://nick-lab.gs.washington.edu/java/jdk1.3.1/guide/2d/spec/j2d-intro.fm2.html
TIBOE. (2023). *TIOBE Index for January 2023*. TIBOE. Retrieved 30.01.2023 from https://www.tiobe.com/tiobe-index/
Ullenboom, C. (2014). *Java SE 8 Standard-Bibliothek: das Handbuch für Java-Entwickler*. Galileo Press.
www.softwaretestinghelp.com. (2022). *What Is Java AWT (Abstract Window Toolkit)*. www.softwaretestinghelp.com. Retrieved 25.12.2022 from https://www.softwaretestinghelp.com/java-awt-abstract-window-toolkit/