# Integrating R and Java for Enhancing Interactivity of Algorithmic Data Analysis Software Solutions

**Titus Felix FURTUNĂ** (titus@ase.ro)
The Bucharest University of Economic Studies
**Claudiu VINȚE** (claudiu.vinte@ie.ase.ro)
The Bucharest University of Economic Studies

## ABSTRACT

*Conceiving software solutions for statistical processing and algorithmic data analysis involves handling diverse data, fetched from various sources and in different formats, and presenting the results in a suggestive, tailorable manner. Our ongoing research aims to design programming technics for integrating R developing environment with Java programming language for interoperability at a source code level. The goal is to combine the intensive data processing capabilities of R programing language, along with the multitude of statistical function libraries, with the flexibility offered by Java programming language and platform, in terms of graphical user interface and mathematical function libraries.*

*Both developing environments are multiplatform oriented, and can complement each other through interoperability. R is a comprehensive and concise programming language, benefiting from a continuously expanding and evolving set of packages for statistical analysis, developed by the open source community. While is a very efficient environment for statistical data processing, R platform lacks support for developing user friendly, interactive, graphical user interfaces (GUIs). Java on the other hand, is a high level object oriented programming language, which supports designing and developing performant and interactive frameworks for general purpose software solutions, through Java Foundation Classes, JavaFX and various graphical libraries. In this paper we treat both aspects of integration and interoperability that refer to integrating Java code into R applications, and bringing R processing sequences into Java driven software solutions. Our research has been conducted focusing on case studies concerning pattern recognition and cluster analysis.*

**Keywords**: *R, Java, Integration, Interactivity, Algorithmic Data Analysis*
**JEL classification:** *C610, C880*

## INTRODUCTION

More often than would be otherwise expected, developing software solutions for statistical data processing requires various levels of interoperability and integration among a multitude of heterogeneous applications, platforms, data feeds, and so on.

Interoperability refers to that competence of software applications residing on different platforms, different operating systems (OS), written in different programming languages to interchange data seamlessly, as components of a cohesive distributed solution. When it comes to an interactive environment like R, interoperability concerns aspects related to fact that R is multiplatform, offers data import/export capabilities from and towards multiple database systems (Microsoft Excel, Microsoft Access, Oracle, MySQL, SQLite), statistical applications (SAS, SPSS), and generates graphical outputs in various formats (PDF, JPG, PNG, SVG).

Our paper treats interoperability at code level, and concerns the integration of Java code within R applications and, vice versa, integrating R code within Java applications. Regarding R applications, the need for such interoperability is dictated by the very advantages and disadvantages of R's interactive environment. The chief advantages of R lay in its concision, yet offering a comprehensive environment, with a considerable set of packages for statistical analysis, and its high popularity among statisticians. R is open source software that can be freely used and, even more importantly, modified, adapted and enhanced. These advantages make the usage of R code very useful within Java applications, and thus provide a more efficient way of processing data for statistical purposes. The main disadvantages of R regard the less rich graphical user interface, compared to other programming environments. From this perspective, Java, a high level generic programming language, can provide R sophisticated graphical user interface (GUI) frameworks (Java Foundation Classes, JavaFX) and various other graphical libraries. The last, but not the least, Java can offer to R applications access to classes and popular mathematical and statistical libraries written in Java, like The Apache Commons Mathematics Library.

## LIBRARIES, TOOLS AND TECHNIQUES FOR SUPPORTING JAVA WITHIN R

User interactivity with software applications has become highly critical, particularly due to the continuous broadening of computer applications usage in various domains of human society. Therefore, building GUI is considered a crucial part of programming for any software development, being currently included in the area of human computer interaction (HCI). The aim of a GUI is to offer to user a facile and friendly manner of interaction with a given application, improving thus efficiency, usability, and lowering the learning curve.

There are tools for building GUI-s in R, and this approach relies on external packages integration. The following list succinctly describes some the most popular libraries:

- **RGtk2** is delivered as a R package for programming GUI-s in R; RGtk2 toolkit provides access to GTK+ 2.0 graphical library, which a is an open source project written in C [1].
- **tcltk2** package, developed by Philippe Grosjean [3], is built upon Tcl/Tk [4] library, which is a mature multiplatform graphical library (widgets), which continues to evolve and to be enhanced; Tcl is a scripting language that allows integration with code written in other programming languages; Java interactivity is ensured through Tcl/Java project, which comprise of two components: Jacl and Tcl Blend; Jacl is entirely developed in Java and provide support for writing Tcl scripts within Java applications; Tcl Blend is Tcl extension which facilitates Java communication via JNI.
- **wxWidgets** library, written in C++, contains graphical components developed in native code for various platforms, preserving the platform specific look-and-feel; currently there are various bindings for wxWifgets in Python, Perl, Ruby, and other programming languages [5]; for accessing wxWidgets graphical components in R can be used packages like Rcpp and RInside.
- **rJava** is package that allows for running Java within R applications via JNI; having such a low level specification there can be employed Java graphical components like Swing and SWT.
- **JRI** is a Java-R interface which facilitates running R code within Java applications; currently JRI specification is included and gets to be installed with rJava package [9].
- **Rengine** is a Java interpreter for R. Through Rengine, R code can be executed within java applications [10]. One of the chief Rengine is the fact that the R interpreter is itself a Java module which can run within any Java application, without the need of loading dynamic libraries, as is the case with JRI.
- **Rserve** is a TCP/IP server which provides the means for using R code in applications written in other programing languages; the application based on Rserve are client-server applications that rely on creating an R connection through which client commands are executed [11]. Currently there implementations available on the client side for C/C++, PHP, and Java.

Interactive visualization allows for a more dynamic interaction with data, and data exploration. Currently, there are R packages that support access to JavaScript libraries, such as *D3*, which don't even require having JavaScript specific knowledges for using such a library for graphical purposes in R.

## R CODE INTEGRATION WITH JAVAFX

JavaFX is a set a Java packages which contains classes and interfaces designed for supporting graphical and media applications development on various platforms [12][13]. JavaFX is a newer alternative to Java Swing, and it is built upon the concept of virtual reality. Among the advantages offered by JavaFX, compared to Java Swing, can be mentioned the followings:

- The event-listener interaction mechanism is better designed, more consistently, and oriented on properties.
- Graphical components structure is less heavy; layout, shape, and controls belong now to a unified structure based on *Node* class; the control sets is more divers comparing to Swing.
- JavaFX employs widely the concepts of property; a property is an object that can be monitored through event-listener mechanism.
- Java applications look-and-feel can be customized using CSS files.
- JavaFX allows for applying special effects to any *Node* type component.
- JavaFX offers direct support for animation and geometrical transformations.

Being a relatively new technology, Oracle released version 2.1 of JavaFX on April 27, 2012, there are not yet specialized libraries for accessing R code in JavaFX and vice versa. One possible solution would be to create an adaptor API by extending JavaFX classes, in order to access more conveniently the interface offered by JRI or Rengine libraries, and rJava package respectively.

### 3.1 Integrating JavaFX within R code

JavaFX is developed in Java, and therefore JavaFX applications can naturally integrate any Java API. JavaFX applications are implemented as self-contained applications, having their own children within Java Runtime and JavaFX Runtime environments.

Using rJava package there can be instantiated Java objects in R, and have access to their fields and methods. A standard JavaFX application is a javafx.application.Application object type [12][13]. JavaFX Runtime creates the Application object, and then calls orderly the methods *init()* and *start(Stage)* of Application class. Later on, *stop()* method is executed when the application ends. An JavaFX application ends either when the static method *exit(),* belonging to Platform class is called, or when all the application windows are closed. Within this architecture, the Application object cannot be accessed from R through functions offered by rJava package. The work around that we identified is to instantiate a Toolkit object, for multithreading management, rendering and event handling. Then can be created JavaFX graphical objects required by the application. This approach is implemented by creating a static block, as follows:

```
static {
      PlatformImpl.startup(()->{});
      Platform.setImplicitExit(false);
  }
```

The main functions provided by rJava package [8], through which Java objects can be accessed, are the followings:

- .jinit() – for initializing JVM;
- .jaddClassPath(path) – provides an additional path to directories hosting JAR filesalong with the default one;
- .jclassPath() – lists the directories from the current path;
- .jnew(class, ..., check=TRUE, silent=!check) – instantiates an Java object;
- .jcall(obj, returnSig = "V", method, ... ) – for calling a method belong to an Java class, object instantiation respectively.

### 3.2 Integrating R code within an JavaFX application

Currently, the most reliable way to integrate R code within Java is offered by JRI API.

In order to access R code from a JavaFX application we employed Java libraries JRI and Rengine. JRI library implements a parsing and interpreting engine for R, and then returns the results to Java. The main class of the specification is JRIEngine, through which a REngine object can be instantiated and then used for parsing and executing R code. A Java application can create only one instance of REngine, which is thread-safe, and therefore its methods can be invoked safely in multiple threads.

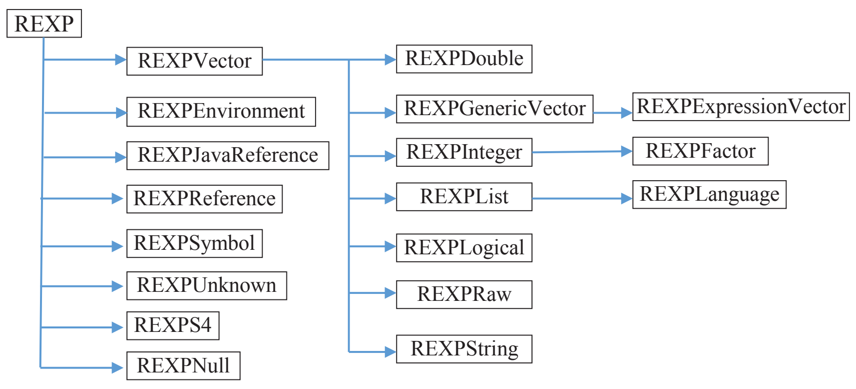
**JRIEngine** class, at instantiation, through the appropriate constructor, there may be specified the option for creating a message stacks, for callback style command processing; otherwise the commands are executed write away. The link between R and Java objects is provided through REXP class and its subclasses [9]. Through REXP class is modeled any type of R object, at the most generic level. The subclasses detail the concept as follows:

- REXPNull - represents NULL object in R;
- REXPUnknown - models an unknown type; it is design to handle the eventuality of dealing with types not supported by the engine.
- REXPReference – is a reference to an R object; it is obtained by calling *parse* methods with parameter *resolve=true*.
- REXPVector – is an abstract class, from which the vector type classes are derived; the subclasses are extensions corresponding to R primitive data types: real, integer, string, factor, list, logical and raw bytes.

Figure 1 depicts the REXP class hierarchy.

**REXP class hierarchy**

*Figure 1*



The methods defined for *REXP* class implement the handling of R objects modeled through REXP, and may be categorized as:

• methods to obtain the content for all data types mentioned above, such as:

```
public JavaType[] asJavaTypes() throws REXPMismatchException;
```

The method names contain the prefix *as* and the plural of the given data type, for *asStrings()*.

• methonds for checking the data type:

```
public boolean isRType();
```

Where *RType* is a R data type which may be *Numeric*, *Integer*, *Logical*, *Complex*, *Factor*, *List* etc.

• methods for handling attributes:

```
public boolean hasAttribute(java.lang.String name);
```

Verifies the existence of the attribute.

```
public REXP getAttribute(java.lang.String name);
```

Supplies the value of the specified attribute.

• methods for creating *DataFrame* and *Matrix* data structures:

```
public static REXP createDoubleMatrix(double[][] matrix);
```

Creates an R *Matrix* object from the specified Java multidimensional array.

```
public static REXP createDataFrame(RList l) throws
REXPMismatchException;
```

Creates an R *DataFrame* object from the supplied Java JRI *RList* object.

*RList* class is Java implementation for any R data type list [9]. The class constructors may receive as parameters instances of *Collection* class or arrays of *REXP* type, as values for the elements of lists and arrays, and arrays of *String* for the name of those elements. Considering the usage frequency of lists in R, it would be extremely useful to build graphical controls for handling lists in Java applications that call R code. Currently, there are implementations in Java Swing for graphical visualization of lists and *DataFrame* type of objects in JGR API (**J**ava **G**UI for **R** API) [7]. These implementations rely on *JTable* class from Java Swing. In JavaFX, on the other hand, data table visualization is achieved through *TableView* class. In order to have tabular perspective of R objects like *data.frame* and *matrix*, one option would be to extend *TableView* class. As part of our research, we propose an implementation of a graphical user interface that implies an extension of JavaFX *TableView* class, in order to handle R objects like *data.frame* and *matrix*.

The constructor of the derived class takes as parameter an REXP type of object, and checks the parameter for being an R object of type *data.frame* or *matrix*. Within constructor, there are identified the actual object type and the variable names, if they were supplied. This information is obtained by invoking *getAttribute* methods of REXP class.

An REXP object is an R *data.frame*, if it has the following attributes:

- *names*,
- *class*,
- *row.names,*

and the *class* attribute has *data.frame* as value.

An REXP object represents an R *matrix*, if it the following attributes:

- *dim*,
- *dimnames*.

The attribute *dimnames* may be missing, case which is treated by the constructor as well.

In order to provide a great flexibility to such a component, the rows are of type *List<Object>*, and the *factory* objects for the columns are of the following type:
**`Callback<TableColumn.CellDataFeatures<List,String>,ObservableValue<String>>`**

As an example, bellow is presented the method that creates the column-type elements for the *TableView* object.

```
private void createColumns() {
      TableColumn<List,String> col0 = new
TableColumn<>("");
      col0.setCellValueFactory(
(TableColumn.CellDataFeatures<List,String> param)->
new SimpleObjectProperty<>(param.getValue().get(0).
toString()));
      this.getColumns().add(col0);
      for (int i = 0; i < noC; i++) {
            final int k = i;
            TableColumn<List, Object> column = new
TableColumn<>(names[i]);
            column.setCellValueFactory((param) -> {
                return new SimpleObjectProperty<>
                  (param.getValue().get(k + 1));
            });
            this.getColumns().add(column);
}
}
```

The method that adds rows to the table verifies the type of data for each column.

For *data.frame* type of REXP objects, this check is done as follows:

```
switch (rFrame.asList().at(j).getClass().getSimpleName())
{
case "REXPDouble":
            ...
            break;
      case "REXPInteger":
            ....
            break;
      ...
}
```

Where *rFrame* is a reference to an REXP object.

For *matrix* type of REXP objects, the check is done as follows:

```
switch (rFrame.getClass().getSimpleName()) {
case "REXPDouble":
            ...
            break;
case "REXPInteger":
            ...
            break;
...
}
```

## CASE STUDY UPON HIERARCHICAL CLUSTER ANALYSIS

Our ongoing research has been focused on programming technics for integrating R developing environment with Java programming language for interoperability at a source code level. In this paper we present a software solution for creating interactive dendrograms for hierarchical cluster analysis using JavaFX.

Hierarchical cluster analysis is an exploratory tool designed to reveal, within a data set, natural groupings, agglomerations or clusters that would otherwise not be apparent. The objects in hierarchical cluster analysis can be observations of categorial or numerical values, depending on the type of the analysis: case classification or examination of relationships between the variables [6].

The case study presented herein uses hierarchical cluster analysis for seismic risk analysis of buildings in Bucharest, Romania.

The data input consist of instances (buildings in Bucharest) and variables:

- year of construction (numeric value),
- number of floors (numeric),
- historical monument (categorical value),
- structural frame type (categorical),
- risk assigned to the masonry type (categorical),
- risk assigned to the type of structural frame (categorical),
- number of apartments in the building (numerical),
- number of people in the building during daytime (numerical),
- number of people in the building during nighttime (numerical),
- number of people working in the building (numerical),
- number of people visiting the building (numerical).

First, we present the integration of Java (JavaFX) code for creating the interactive dendrogram within an R script. The calling R script is showed below.

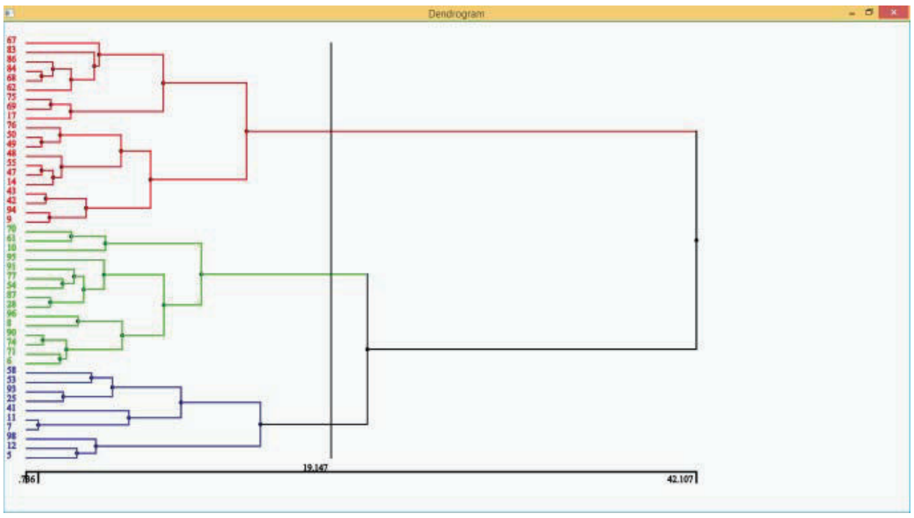
```
source("ClusterJ.R")
table=read.table("D:\\Titus\\Articole\\InLucru\\
RInteroperability\\
      ClusterAnalysis\\ProiectB.
csv",header=T,sep=",",row.names=1)
k=list(2,3,4,5)
obj=hclustFX(table,k,"ward.D")
partition = .jcall(obj, "[I", "currentClassification")
print(partition)

hclustFX=function(table,k,method)
{
  for(var in k){
    table[,var] = as.numeric(as.factor(table[,var]))
  }
  d = dist(as.matrix(table))
  clust = hclust(d,method = method)
  labels=rownames(table)
  index=clust$merge
  distJ=clust$height
  if(!"package:rJava"%in%search()){
    library(rJava)
  }
  .jinit(force.init = TRUE)
.jaddClassPath("D:\\Titus\\Articole\\InLucru\\
RInteroperability\\
ClusterJava\\dist\\ClusterJava.jar")
```

```
  hjw <- .jnew("rjava/javafx/HCFX",TRUE)
  .jcall(hjw, "V", "setData",index,distJ,labels)
  .jcall(hjw, "V", "createDendrogram")
  .jcall(hjw, "V", "show")
  return(hjw)
}
```

Running the above R script has as output the dendrogram pictured in figured 2, created in JavaFX.

**Dendrogram built in JavaFX 2D Graphics**

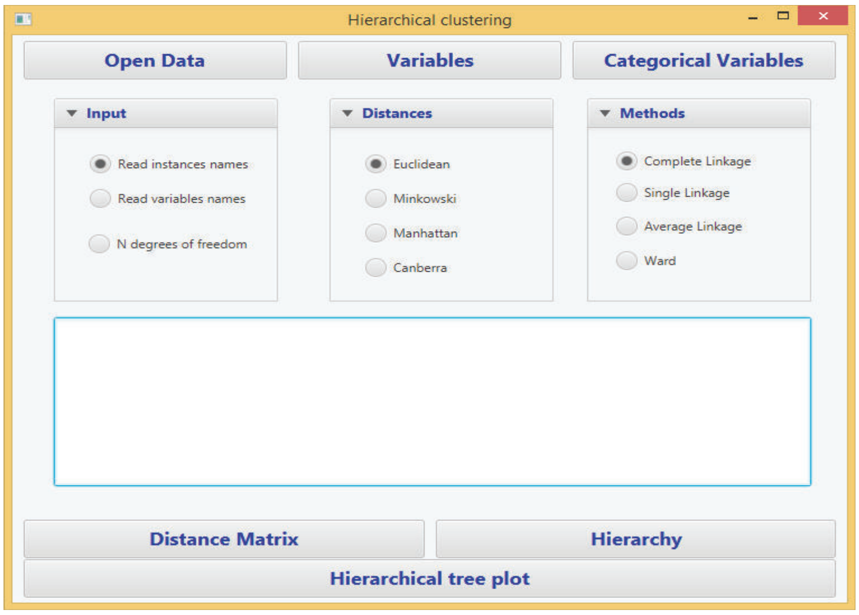
*Figure 2*



Second, within a JavaFX application, we invoke R code through Rengine for the hierarchical cluster analysis processing, and then create the interactive dendrogram. We developed a GUI in JavaFX, in order to offer a friendly interface for choosing the values of the parameters to pass to the R *hclust* method. Employing JavaFX graphic controls, the application calls corresponding R scripts for clustering (figure 3).

**JavaFX GUI for hierarchical clustering**

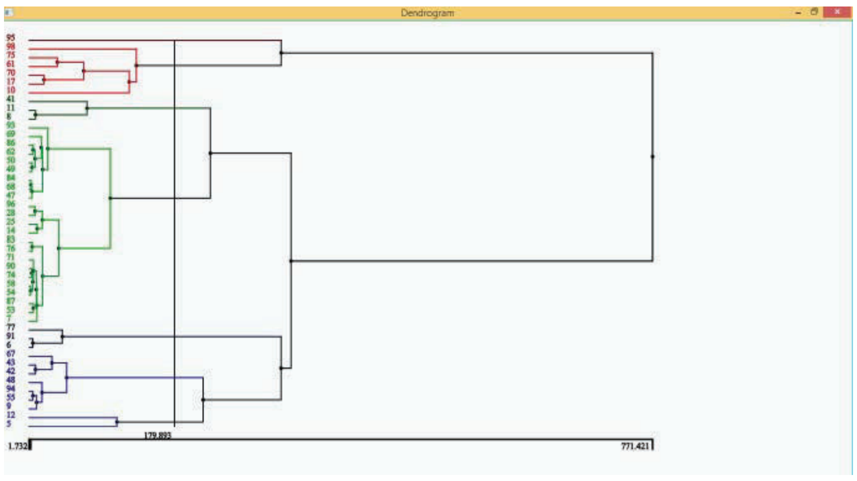
*Figure 3*



The corresponding dendrogram is then generated within the Java application, see figure 4.

**Dendrogram built in JavaFX 2D Graphics**

*Figure 4*



## CONCLUSIONS AND FURTHER RESEARCH

In this paper we treat both aspects of integration and interoperability that refer to integrating Java code into R applications, and bringing R processing sequences into Java driven software solutions.

Our ongoing research aims to design programming technics for integrating R developing environment with Java programming language for interoperability at a source code level. The goal is to create an API wrapped around rJava and JRI, that would readily allow for combining the intensive data processing capabilities of R programing language, along with the multitude of statistical function libraries, with the flexibility offered by Java programming language, in order to develop interactive GUI-s based on JavaFX as presentation platform.

Our research will continue to explore for graphical tools developed through R and Java integration for supporting pattern recognition and cluster analysis.

**REFERENCES**

1. **Michael Lawrence, Duncan Temple Lang,** 2010, *RGtk2: A Graphical User Interface Toolkit for R*, Journal of Statistical Software, 37(8), pp. 1-52.
2. **Verzani J**, 2009, *gWidgets: gWidgets API for Building Toolkit-independent, Interactive GUIs*, R package version 0.0-37, ***http://CRAN.R-project.org/package=gWidgets***
3. **Grosjean P**, 2010, *tcltk2: Tcl/Tk Additions*, R package version 1.1-2, **http://CRAN.R-project.org/package=tcltk2**.
4. **Ousterhout J**, 1994, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, USA.
5. **Smart J, Hock K, Csomor S,** 2005, *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
6. **Aldenderfer, M. S.,** and **R. K. Blashfield**, 1984, *Cluster Analysis*. Newbury Park: Sage Publications.
7. **Helbig M, Theus M, Urbanek S**, 2005, *JGR: Java GUI to R,* Statistical Computing and Graphics Newsletter, 16(2), dec. 2005, ***http://stat-computing.org/newsletter/issues/scgn-16-2.pdf***
8. **Urbanek S**, 2009, *rJava: Low-Level R to Java Interface*, R package version 0.8-1, ***http://CRAN.R-project.org/package=rJava***.
9. *JRI - Java/R Interface*, ***https://rforge.net/JRI/***
10. *Renjin developer documentation*, **http://docs.renjin.org/en/latest/**
1. **Simon Urbanek**, 2003, *Rserve -- A Fast Way to Provide R Functionality to Applications*, Distributed Statistical Computing (DSC 2003), Technische Universität Wien, **https://www.r-project.org/conferences/DSC-2003/Proceedings/**
11. **Weaver, J., Gao, W., Chin, S., Iverson, D., Vos, J.**, 2014, *Pro JavaFX 8. A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*, Apress
12. *Getting Started with JavaFX*, ***http://docs.oracle.com/javafx/2/ get_started / jfxpub-get_started.htm***
13. *Using JavaFX UI Controls*, ***http://docs.oracle.com/javafx/2/ui_controls/ jfxpub-ui_controls.htm***