JavaFX GUI, much simplified introduction

Virtual PC

To simplify the work for those students who, for various reasons, either can not or do not want to install the programs needed for this subject, I created a virtual PC. Since this is a fairly powerful and reliable tool I recommend its use to all students. Another significant advantage is that its use will provide a single environment, exactly the same for all students and for teachers, which simplifies debugging programs and possible troubleshooting.

Virtual PC environment is created in Oracle VM VirtualBox 4.3.12.

First Start

Downloading VirtualBox

The latest version of VirtualBox download from the web
https://www.virtualbox.org/wiki/Downloads.

Yet there is a portable version Portable VirtualBox, which can be installed on a USB memory. How reliably it works and if it is fully compatible, I did not test. Its use is valuable on the computer on which you do not have sufficient rights to install programs.

All versions of personal use VirtualBox are licensed under *GNU* license.

Freshly installed VirtualBox is completely empty, without any programs and even without the operating system. To put it to use, it is necessary to install the virtual machine first. This can be done in several ways. I recommend that you download and install the **appliance**.

Appliance

Our appliance is in file **Tutorial_2014-15.ova** and you can download it on the subject pages.

Its size is about 7 gigabytes and it runs on Windows 7, 32-bit version.

The freshly installed VirtualBox enter *Import Appliance*. This will start the installation program.

On a standard PC installation takes about 10 minutes.

While installing, you can review the license conditions. Of these the most important is that Windows on a virtual machine are not licensed, so it is necessary to activate the license.

Our school has multilicences, activation process can be found on the school web site.

Other programs, especially NetBeans and Scene Builder, are **free** for private use.

MS Project and Enterprise Architect are installed as a **trial** version for the limited number of days. This you must consider when you schedule time to study.

The first thing you need to do after installing the appliance, is adjusting the size of memory. The default memory allocated for virtualbox is only 512MB of memory. While this is sufficient to run, but then the virtual machine is desperately slow.

I recommend to set up at least half of the physical memory. Configuration is performed via System →Settings → Memory shown on *picture*.

Running and Closing

You will run the vitrual computer by button „*zobrazit*“.

Shutdown is surprisingly a bit complicated. If the virtual machine is turned off in the normal way, it saves its state and when runs next time, it is recalled automatically. But for various reasons it may be advantageous to go back to older state. For example, if you accidentally delete a file or if something in the system is not set properly, it is nice to return before bad changes. Virtual machine solves it using images (**snapshot**). View the *images* (red oval) and then, pressing the "camera" (blue oval) can record images. The recorded images can be recalled at any time.

In special situations it is advisable not to store all status changes.

If you do not want to save the changes, simply *force closing* the virtual machine. Next time normally starts the last saved state. No need to worry about "destroying" windows.

NetBeans

Online Compiler

There are cases where you can get by with just a text interface. For example, for studying purposes, there are many examples where you do not need to work with graphics, which can totally make do with text input and output. In such cases, you can work with any on-line compiler. On the Internet you can find a number of them.

Good and tested on-line compiler for Java, see http://www.compileonline.com.

Compiler window is *shown*.

To use it properly, you need to know where to enter values.

If you run a program with arguments, you must specify the parameters of the program in the "Command Line Arguments", see black arrow. Most do not use this.

If you want the program pass any data from the keyboard, you must write it in advance into box for standard input "STDIN Input". It is highlighted by green arrow.

At the top of the form are three tabs. The first one is for editor, where you write code for main class (yellow arrow).

Next is the tab "input.txt" (red arrow). Unfortunately, it is difficult to see, it is covered by combobox with choice for the editor. Into "input.txt" form, type the text that your program should use at runtime as if it were read from "input.txt". In other words, this form emulates a disk file "input.txt".

If we need to work with multiple files, check the "Multiple Files" (blue oval), and we get the opportunity to work with multiple files.

To compile and run your program, press *button* „Compile and Run".

NetBeans in text mode

Using the NetBeans GUI mode we will see later. Now we will first show how to work with NetBeans in the classic text mode. By text mode, I mean that the entire program code is entered from the keyboard in text form. That's the way most programmers work.

First Steps

The new project will start via File → New Project.

For now, we will work in text mode, and so we choose **Java** and **Java Application**. This opens a *dialog*.

Important: It is not good just mindlessly click over the dialog that opens. You must enter a *project name*, **Project Name**. If it is in this form incorrectly, and directory and file name, a wrong and then everything takes a lot of work to clean up.

Note that the class name must be written with a capital letter. Therefore, the file name of this class must have initial capital (in case we ported our project into an operating system that distinguishes between uppercase and lowercase letters).

For the same reason, the name of the project should begin with a capital letter.

On the left side of the *form*, in the Projects window (black arrow), we see the structure of the project. Currently, there is

only a file with our new project and it is set JavaApplication1.java.

Suffix **.java** have files containing source code.

Into the right half of the forms we write the source. Ready is a place for the program documentation (green arrow). Java is designed so that comments and program documentation is recommended to write directly into the source code.

Program documentation is then automatically generated using **JavaDoc** application.

A place where we will write program code contains the line `//TODO code application logic here` and is indicated by the blue arrow.

Once you have written the code, you will start the compilation and run.

This is done pushing the "arrow", which is highlighted in red in the *oval*.

Hello World

It has become standard that the first program in any language is demonstrated on a simple "Hello World!" which does nothing other than that prints a greeting.

I will also introduce NetBeans on a simple example "Hello World".

Although not yet know the syntax of the programming language, we write a short program to show how NetBeans works.

The source text pane, delete the line `//TODO code application logic here` and instead write `// Demo program`. Note that the line must start with a double slash, because it is a comment, and comments begin with a double slash (... or otherwise, as we shall see later).

Then we write the first line of the program. We want to write `System.out.println (" Hello World ");` which we

could write as well as straight, but we will demonstrate it slowly and in detail to stand out procedure.

We start typing `System`. When dot appears, two auxiliary window are shown that pertain to the written word (in this case the word System). It shows a *figure*.

The bottom window with blue accent line offers us all the options at this point we may write. We can either write "out" straight, or to "out" you can move the highlighting, or finally start writing "on" and gradually, as we write, the selection is gradually narrowing. Finally, we have the whole "out" written. Write dot and the process is repeated for the rest of the line. As for the upper window, it displays a simple on-line help.

If you make a mistake while typing, the NetBeans reports it with *red ring* on the beginning of line. For example, we deliberately deleted terminating quotation mark and the system reported an error.

Important

Note that NetBeans reports an error to where it founds it. This means that the error may lie even a few lines before!

We can compile and run the program clicking the green symbol (it is highlighted by red oval in figure).

Výsledek ukazuje obrázek: program vytiskl pozdrav a zeleným tiskem ohlásil, že všechno proběhlo v pořádku.</CZ>The program compile and run.

The _result_ shows the greeting. Text printed in green ink reports that everything was all right. The whole program we have available in the _file_ from which you can load a copy to NetBeans.

Debugging

Now we will show how to debug programs.

Debugging is a real art and significantly affects both the speed and quality of work of programmers.

The most common errors are **syntax** errors. Usually arise misspelled, omissions and the like.

Their common feature is that NetBeans reveals this kind already during translation, sometimes (as in the previous example)

already when writing code. Therefore, it is easy to find and remove.

Much more difficult are semantic errors, it means errors in the program logic. Such errors reveal a syntax check.

Often, as in the following example, the error manifests failure of the program at runtime. But there are also cases where the program seemingly works well, not collapsing, but still gives incorrect results.

This kind of error founds extremely difficult.

There are two good ways to make it easier to debug. They are **extreme programming** (one of agile programming methodologies) and **stepping** through code. We'll explain both.

As an example, we show the code from the attached *file*.

Although not yet know the Java language, intuitively we sense what is wrong. The program includes three lines (19, 20 and 21) that print something. While lines 19 and 21 print some texts (like in the Hello World), line 20 prints a number. More specifically, it prints result of number 123 divided by zero.

Of course, this is not possible and the program therefore will stop with *runtime error*.

We of course know where the error is. If we did not know, NetBeans still help us by red printed details about the error. That itself is often enough to identify errors.

Now we will show a simple example of extreme programming. We go from the "Hello World", about which we know work.

(1)   As a first step we change the text in quotes from "Hello World" to "This is first line". Compile, run a program obviously works.

(2)   To add a dividing line 20. Although the program compiles correctly, it ends up with a runtime error. We will therefore seek such a mistake could cause the 20th row

(3)	When the error found, add the line 21. Program continues to function properly.

Extreme Programming is suitable for larger and more complex sections of code like for finding errors in database transactions, for complex algorithms etc. Do not add single lines, but the entire blocks of code.

Furthermore, we show stepping through code.

Reload program from a file. Now click on the line number at which you want to stop the program. This will set the **breakpoint**.

Break-point is indicated by a pink square and pink highlighted line. A second click will cancel the breakpoint.

The program will run until a break-point is reached. Just **before** the breakpoint, execution stops and so we have the opportunity to observe what happens in the program.

Important

For stepping program you have to use a different *icon*, or Ctrl + F5.</

Program stops before executing the breakpoint (green arrow). In the "Variables" (red arrow) are the current values of variables.

Now, by pressing **F7** or **F8** always perform one line of program while watching how it changes the values of variables.

The difference between F7 and F8 is that F8 does not stop within a function, it processes it in full.

In our example, the program stops when you try to execute the line 20.

If using F7, we could find the exact place in the library where the error was detected. Stepping through own program usually has sense, stepping libraries usually not (because we do not understand what is going on there).

Basic I/O

Most applications need some form of input or output. The inputs and outputs in graphical (visual) form another chapter discusses, here we do not deal with graphics.

When speaking about a text (or data) inputs and outputs, they are usually divided into two categories: the input / output to the console and input / output to discs. The console is a control device of the computer; in ancient times it was a teletype, later keyboard with text screen. Today, the console can be compared to the system the DOS window, as we know from CMD.EXE program.

The principal advantage is that all inputs / outputs operate according to universal scheme. Used is a universal model, which is called a "pipe". It is an abstraction at which we imagine that at one end of the "pipe" we fill the data and at the other end of the "pipe" data appear on disk, the screen, and the like.

What is happening with the data while passing through the "pipe" proces, is hidden from the programmer, because it is job for the operating system to handle it.

Because the interface of "pipe" is always the same, regardless of where the "pipe" is connected, working with I / O devices is significantly unified.

Important

We do not know anything about programming language JAVA, its commands or spelling. But somehow we need to start from the beginning and we need to somehow communicate with our programs. Therefore, I have prepared a few basic schemes. What means will be explained later.

For now, memorize them as a poem.

JAVA language distinguishes between uppercase and lowercase letters. **Therefore, remember that size matters** ☺.

Console I/O

- The „console" can mean three system I / O:
  - System **output** (`System.out`), a screen
  - System **error output** (`System.err`), screen - usually with a different text color, for example red
  - System **input** (`System.in`), a keyboard in our case

Scheme for text output:

```
System.out.println("This is a clever sentence.");
```

Scheme for keyboard input:

```
import java.util.Scanner;
…
Scanner sc = new Scanner(System.in);     // create a "scanner"
String s = sc.nextLine();                // read line into string "s"
```

Example

Working example for NetBeans is in file ..\files\03-19 KeyboardIO.zip

File I/O

File input / output is more complicated because we have to do more complex preparation than with a keyboard. Important parts are in bold.

Reading a file „a.txt"

```
 import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
…
```

```java
public static void main(String[] args)
throws IOException {
        FileReader fr = new
FileReader("a.txt");
        BufferedReader orig = new
BufferedReader(fr);
        String s;
…
        s = orig.readLine();
```

Writing to file „b.txt"

```java
 import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
```

```java
import java.io.FileWriter;
import java.io.IOException;
…
public static void main(String[] args)
throws IOException {
        FileWriter fw = new
FileWriter("b.txt");
        BufferedWriter copy = new
BufferedWriter(fw);
        String s;
…
        copy.write(s);
        copy.newLine();
…
        fr.close();
```

```
      copy.close();
```

Example

Working example for NetBeans is in file ..\files\03-20 FileIO.zip

JavaFX graphic interface

JavaFX

 JavaFX is a modern framework for creating rich window applications. Rich is meant here visually. JavaFX provides support for images, videos, music, graphics, CSS, and other technologies to ensure that the resulting application is really pleasing. At the same time the emphasis is on ease of creation, all things are mentioned in JavěFX in the base. JavaFX is suitable for both desktop applications and web applets or mobile applications.

Three ways to create graphical interface

In the JavaFX, you can develop similarly as in older Swing, you create an instance of form elements (buttons, text fields). You then insert into the so-called *layouts*, which are actually containers on the form components.

The second way is FXML. FXML is a language for the forms design. As you expect by the title, it's another language derived from XML. Using XML to design the presentation of the application (this is the part that communicates with the user) is not new, on the contrary it is a proven principle of web

applications. Here, Java adapts principles of HTML and CSS to desktop applications.

The third option is to create a graphical interface in the graphics. The "clicking" method is by far the most visual creation and also the simplest. This third way is implemented by **Java Scene Builder**.

Basic "vital functions" of each application are already integrated into an integrated graphical environment, so the programmer does not need to program them. The only thing that has to do is to handle **events** that occur at runtime. Such an event, for example, may be pressing a button. The programmer writes

handlers to handle individual events. This method of programming is called **events-driven**.

Java Scene Builder

Java Scene Builder is a direct part of NetBeans, but it is a standalone application. You can JavaFX Scene Builder download from the server www.oracle.com .

Scene Builder Java is already preinstalled on your tutorial virtual machine.

 Although we do not know anything about Java, we can now show the basics of Scene Builder, as we will not need knowledge of the language.

Introduction

New Project

First, you create a project in NetBeans. The project is called the place (usually a directory) in which to gather all information, files, settings, and translated programs related to the problem addressed.

Open your NetBeans and choose **File->New Project**, choose **JavaFX**, and then choose **JavaFX FXML Application** (!), see [figure](). This creates a very simple JavaFX app that includes a main application class, a controller class to provide the actual backing logic for the window defined in Scene Builder, and the

FX Markup Language (FXML) file containing our window definition (XML) code. You do not need to read nor understand any of these files.

On next page you **must** fill the [project name](). If you forget to enter project name, a default one will be used – but it is a very difficult job to rename it. So you have better to use a proper name now.

- On the upper left hand side of NetBeans window, in [Projects pane](), you can see three files: **EasyProject.java** contains the main application class. We won't do anything with this class for our example, because its primary purpose in life is to load the window definition code contained in the FXML file and

then show the main stage/scene. You'll keep the JavaFX terms straight with ease if you relate them to a theater: a platform holds a stage, which contains scenes.

- **FXMLDocumentController.java** is our controller class, which provides the "brains" behind the graphical interface. If you open the SampleController, you'll see that it includes a property and a method tagged with @FXML. This tag enables the integration of the visual controls and elements you define using Scene Builder, which are stored in an FXML file.
- **FXMLDocument.fxml** is the definition file for our sample window. You can right-click and edit the file name in the tree to view the underlying FXML—and you may need to do that if

you change file names or properties by hand—or you can double-click it to open it (visually) in Scene Builder.

At this moment, we can compile and run the project (Run → Run Project). You can see that new project is not empty. As usual, it contains a simple application „Hello World!“.

We can compile and run the program clicking the green symbol ▷(it is highlighted by red oval in figure).

Right click file FXMLDocument.fxml and select **Open**. This automatically opens Scene Builder – an application for graphic editations.

Drag Button somewhere else to see that it moves. After this, **we save the scene (Ctrl+S),** open NetBeans and try to run the program again.

Oops!!! Position of Button didn't change!

The explanation is simple. NetBeans and SceneBuilder are not connected directly, but indirectly, through the FXML file. When we saved the scene in SceneBuilder, the new location is written in FXML file, but NetBeans still does not know about it.

We must read changes in FXML first. We do so by right clicking on *FXMLDocument.fxml* and selecting [Edit]. Btw, this also opens editor where we could edit FXML file manually…if we could.

Now we can run program again and everything works OK.

Attributes or Properties

Switch to SceneBuilder again and select Button. In the righr part of screen we see all features of Button. As an example, feature Text contains text string with label of Button. Try to change it and make sure, it changed in program as well.

Completely analogous: the scene has a label Label. The picture label you can not see because it is empty, but we see it in the navigation bar on the right. When you select it there, you'll see it onscene. If we change the property Rotate to 45, the title is rotated by 45 degrees. Rotate property, as well as all the

properties that relate to appearance, can be found on the Layout tab

Editing

The last example refers to adding objects to scenes.

On the "Library" pane on the left, select the Controls tab. Find CheckBox and drag it to the scene. Then run the program in the usual way and you will see that the check box is functional. Just for us it is useless because we can not write code that would use it. By the way, the file was added FXML kind of CheckBox

For Stunts

Who wants can continue.

Switch to tab *FXMLDocumentController.java*. There are lines

```
private void handleButtonAction(ActionEvent
event) {
    System.out.println("You clicked me!");
    label.setText("Hello World!");
```

`handleButtonAction` says that this code runs if you activate (press) the Button.

we see that there is a line `System.out.println` that prints out given text. Also, there is a line

```
label.setText("Hello World!");
```

We can try to understand it. It says that in `label` we set text to „Hello world!".

Let's write our first command. Append line

`label.setRotate(label.getRotate()+ 45);`
Its meaning is as follows: take current rotation of label
`label.getRotation()`, add 45 degrees and store result back to label's rotation `label.setRotate(…)` .

You can find the project in file ../files/03-43%20SimpleProgram.zip.

Excercise

Open NetBeans and create a simple application to print out two identical lines „Hello World!".

Demonstrate a step-by step debugging.

Demonstrate a run-time error.

Same application compile and run using an *on-line* compiler-

In *SceneBuilder* create this [scene](#) (semaphore). Compile it to see that there are no errors.

Write a short application that reads lines from keyboard and prints them on the screen, each line 2x.

Write a short application that reads lines from file and prints them on the screen.

Write a short application that reads lines from keyboard and prints them into file TEST.TXT.

Content