An Example of Integration of Java GUI Desktop Technologies Using the Abstract Factory Pattern for Education Purposes

Ana Korunović [*], Siniša Vlajić [**]

***Keywords**: Design patterns, Graphical user interface, JavaFX, Swing*

Abstract: The integration of Java GUI desktop technologies (Swing and JavaFX), using the Abstract Factory pattern, is explained in this paper. An overview of the basic features of Swing and JavaFX technologies is given, as well as the design pattern concept with an emphasis on the Abstract Factory pattern. In our study example, we have shown how to integrate the existing software system based on the Abstract Factory Pattern and Swing GUI technology with JavaFX GUI technology in a relatively simple way. The integration aims to explain to the students of the Faculty of Organizational Sciences (FON), University of Belgrade, in the course Software Patterns, the application of software patterns in connecting different GUI technologies.

1. INTRODUCTION

When developing software systems, it is very important to create a flexible and sustainable architecture. The architecture consists of interconnected components that are linked to each other via their interfaces [1]. Software system architectures can be implemented using macro-architecture patterns (Enterprise Component Framework (ECF), Model-View-Controller (MVC), etc.) and micro-architecture patterns (GOF design patterns) [1,2]. GOF (*Gang of Four*) design patterns represent generic solutions that can be applied in solving a number of similar problems. These patterns are divided into three groups: creational, structural, and behavioral patterns. The **Abstract Factory pattern** used in this paper is one of the creational design patterns.

The architecture of the software system can be divided into client-side (frontend) and server-side (backend). The frontend part of the software system consists of graphical forms

[*] Ana Korunović is with the Faculty of Organizational Sciences, University of Belgrade, Serbia (phone: 381-64-3320472; e-mail: ak20223702@student.fon.bg.ac.rs).

[**]Siniša Vlajić is with the Faculty of Organizational Sciences, University of Belgrade, Serbia (phone: 381-69-8893133; e-mail: sinisa.vlajic@fon.bg.ac.rs).

and form controllers. Screen forms can be implemented using various GUI (Graphical User Interface) technologies. In this paper, the Java GUI libraries **Swing** and **JavaFX** were used to create a graphical user interface. Swing and JavaFX represent two heterogeneous technologies that can be used together when implementing a graphical user interface. Both technologies are based on the MVC pattern and tend to separate the graphical user interface from the application logic [3].

To explain the integration of Swing and JavaFX GUI technologies, a software system [4] was created in the Java programming language connected to a MySQL database. The above-mentioned integration was performed in order to explain the use of software patterns in connecting different GUI technologies to the students of the Faculty of Organizational Sciences (Information Systems and Technologies program), University of Belgrade, as part of the elective course Software Patterns (4th year of study).

The work consists of six chapters. The introduction to the topic and the goal of this paper is followed by an explanation of the basic concepts of Swing and JavaFX GUI technologies (chapters two and three). Chapter four provides a general definition of the pattern and an explanation of the Abstract Factory pattern. The fifth chapter explains the software system in which the integration of Swing and JavaFX GUI technologies was performed using the Abstract Factory pattern. The sixth chapter contains concluding remarks.

2. SWING

The creation of a graphical user interface in Java is made possible by the use of Java Foundation Classes (JFC), which consist of a group of libraries [5]. JFC consists of the Abstract Window Toolkit (AWT), Java 2D, and Accessibility libraries. The inability to change the appearance of graphical components, platform dependency, non-compliance with the MVC pattern in AWT components, and scarcity have led to the development of Swing [6]. Swing technology was introduced to address the shortcomings of the AWT technology and represents its further development. Although Swing is more advanced than AWT components, it is important to emphasize that event handling is done in the same way in both libraries.

Swing components are written in the Java programming language, and it is possible to change the look of the components and make them independent of the specific implementation platform (pluggable-look-and-feel) [5]. In the context of patterns, Swing supports the Model-View-Controller (MVC) pattern, which separates the control of the display of components on forms (View), the way the components react in interaction with the user (Controller), and the state that the components contain (Model).

Developing a graphical user interface with the Swing library requires knowledge of Components and Containers (Figure 1). Containers are components whose purpose is to connect a group of components, such as JButtom, JComboBox, and JTextField. Swing uses higher-level containers that inherit the Container and Component classes from the AWT library. These containers are not able to change their form and depend on the platform operating system [5].

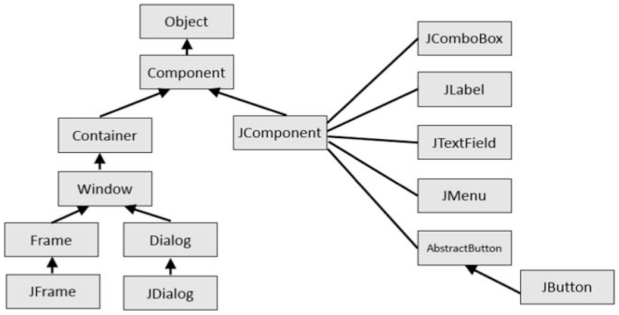

Fig. 1. An example of a Swing library class hierarchy [3]

3. JAVAFX

Swing components are designed for developing graphical user interfaces for desktop applications, while JavaFX technology is also used for building rich Internet and mobile applications [7]. This library was developed by Sun Microsystems and is based on the Java programming language. JavaFX also introduces Java programming language features such as generics, lambda expressions, annotations, and multithreading. An advance over Swing technology is FXML, a scriptable language based on XML. FXML describes the user interface and allows the definition of controllers that respond to events. The logic contained in FXML allows the controller to respond to events in a timely manner without communicating with the user interface, giving rise to the MVC architecture [8].

The graphical user interface in JavaFX is built as a collection of nodes that form a tree [9]. Nodes are an integral part of the scene, i.e., the stage (Figure 2), and can be 2D and/or 3D objects, images, video and/or audio recordings, labels, text input fields, and the like [10].

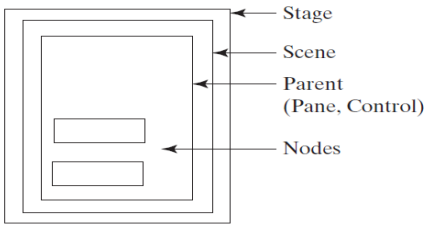

Fig. 2. The relationship between components and nodes in JavaFX [7]

The *javafx.stage.Stage* component represents a parent container created by a particular platform. A stage is a scene consisting of a single node or a tree of graphical user interface nodes (Figure 3).

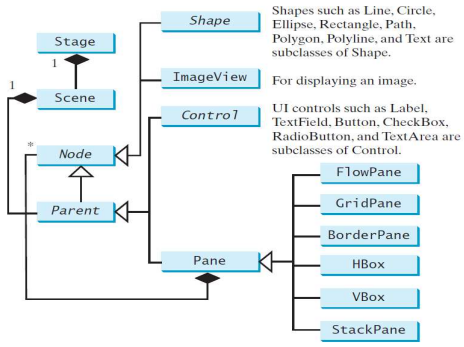

Fig. 3. JavaFX class diagram [7]

JavaFX is controlled by the JavaFX platform, which creates objects and threads the application [11]. All changes to the stage and nodes must be made within the JavaFX application thread in order to be displayed to the user.

4. ABSTRACT FACTORY PATTERN

Software patterns establish a relationship between a context, a system of recurring forces in that context (problem), and a software configuration that enables those forces to establish appropriate relationships (solution) [1]. Patterns provide a general solution to a group of problems when developing a software system, that is, the reusability of once written program code. Software patterns can be divided into three-level patterns, anti-patterns, and meta-patterns. Of the three-level patterns, GOF (*Gang of Four*) design patterns attract special attention.

The Abstract Factory pattern is one of the creational patterns within the family of 23 GOF design patterns. Abstract Factory defines an interface for creating related and dependent objects (products) without specifying concrete classes (Figure 4). This type of encapsulation prevents the client from having direct access to specific products and relieves them of the responsibility of creating them. The client manipulates instances by reference to *AbstractFactory*, manages and monitors the process of creating a complex product, and creates a complex product while *AbstractFactory* determines how to create parts of a complex product [1].

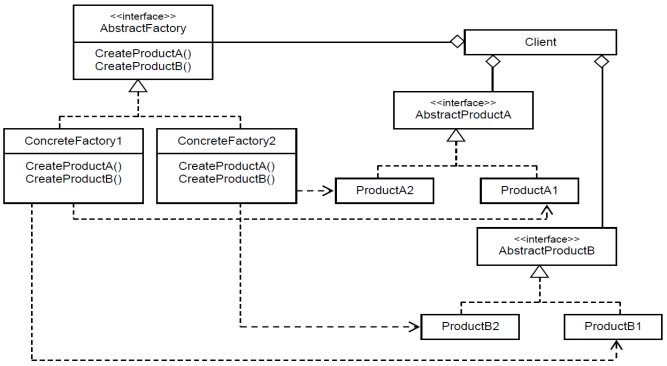

Fig. 4. The structure of the Abstract Factory pattern [2]

The advantage of using the Abstract Factory pattern is the isolation of the concrete products from the client, which allows better control of instantiation. Also, it is easy to change the product since the specific product is only in one place in the software system. The *AbstractFactory* interface limits the collection of products that can be created, so adding a new and different implementation requires changing the interface and the previously created concrete products [2].

5. INTEGRATION OF SWING AND JAVAFX TECHNOLOGIES USING THE ABSTRACT FACTORY PATTERN

The idea of integrating two heterogeneous GUI technologies arose from the desire to give students of FON a practical example of how to use software patterns (Abstract Factory) when combining different GUI technologies (Swing and JavaFX).

It is important to emphasize that the idea of our teaching approach is not only to show students the final structure of sustainable software systems. First, students should learn about the process of transforming unsustainable software systems into sustainable ones using software patterns, so that they can then independently design and implement their own study examples.

In our study example, a software system [4] was developed for processing students' exam registration. The elements of the system are a screen form with fields for receiving and displaying data, a database broker, and a controller for exchanging data between the controller and the database broker. The Abstract Factory *Client* oversees the process of making a complex product and creates the complex product, while the *Designer* is responsible for creating parts of the complex product. The software system of the study example is based on the software system (Figure 5) where the Abstract Factory pattern was applied in the development of the Swing GUI Java application.

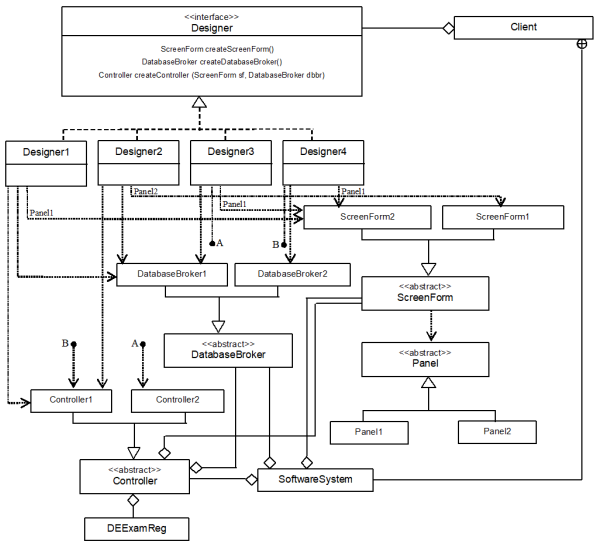

Fig. 5. Class Diagram - Swing GUI Technology

Figure 5 shows that the *Client* initiates the development of a complex product and contains a reference to the *Designer*, i.e., the *AbstractFactory*. Implementations of the Designer interface create their own combinations of parts of a complex product.

The sustainability of the structure was achieved by introducing abstractions and moving the event processing logic to the controller via the screen form. This overcomes Swing's shortcoming of automatically generating program code and forcing the programmer to place the logic for responding to events in screen forms, resulting in inflexible and unsustainable structures.

The abstractions in the application are classes: *Panel*, *ScreenForm*, *Controller*, and *DatabaseBroker*. The methods contained in the abstract classes *Panel* and *ScreenForm* refer exclusively to styling the appearance of the user interface, the setup of data input fields, and components that can react to events. By moving event listeners outside the user interface, the application logic is moved to a higher level, namely the controller level. In this way, the three-tier architecture of the application, which tends to the MVC pattern, was achieved (Figure 6).

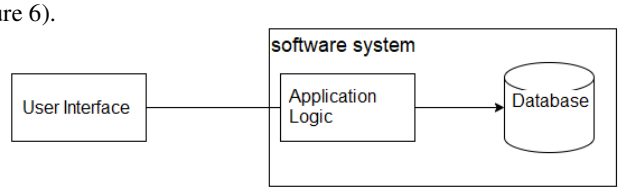

Fig. 6. Three-tier architecture of the software system [12]

The three-tier architecture of the application consists of the following elements [12]:

- The user interface was developed using Swing technology. On the screen forms it is possible to enter a new exam application, change, search and delete an existing application;
- The application logic layer contains the controller, which is the link between the user interface and the database broker. The application logic controller is responsible for placing listeners on-screen forms and responding to events raised by those listeners by calling methods of the database broker. The database broker contains methods that perform operations with the database using domain-general objects;
- The database level contains data on exam applications;

By introducing abstractions, it is possible to separate various screen forms and panels from the control of the application logic. The use of patterns eliminated places in the program that were difficult to maintain and update. The result is generic components that can be reused in the development of other similar software systems.

The idea of developing an existing system evolved even though the software system described earlier gave students insight into how to recognize the places of mutable and immutable program code and when is the right time to include abstractions and generic structures. The goal was to integrate the existing system (Figure 5) with JavaFX GUI technology to enable the implementation of screen forms using JavaFX technology. By integrating Swing and JavaFX using the Abstract Factory pattern (Figure 7), a matrix structure of screen forms and different GUI technologies was created.

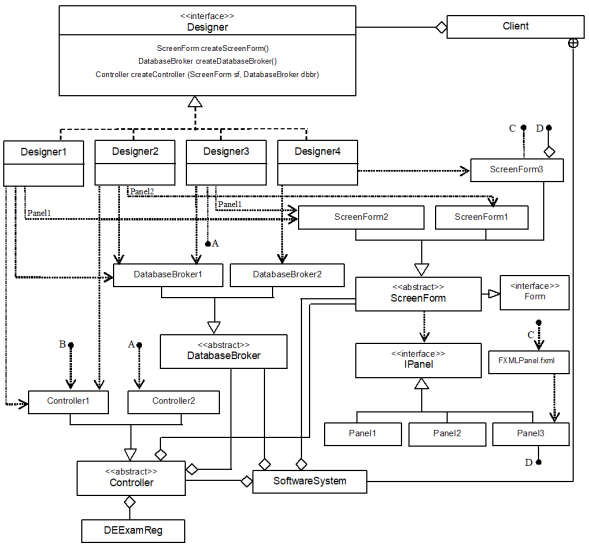

Fig. 7. Class Diagram - Swing and JavaFX GUI technologies

The structure created using patterns is easily expandable, but the incorporation of new technologies requires the adaptation of certain components. The differences in the implementation of Swing and JavaFX require that the abstract class *Panel* be replaced with the interface *IPanel*, thus achieving a higher level of abstraction. In the abstractions, there are no longer references to the classes of the *javax.swing* package, but the superclass of all *Object* classes is used. By creating the *Form* interface, the abstraction level of screen forms has been raised.

ScreenForm3 and *Panel3* differ from other forms and panels by the technology they use. *ScreenForm3* inherits the *javafx.application.Application* and contains **start**(*javafx.stage.Stage*) method. Inside this method, an FXML file loads. *Panel3* contains an FXML annotation declaring the controller members of the *javafx.scene.control* package initialized by *FXMLLoader*.

Minor changes were made at the controller level, but the core remained the same. The event processing logic was moved from the screen forms to the *Controller*, the *Software System* includes a reference to the parts of the complex product, and the decision about instantiation of the parts of the complex product is made by the *Designer*.

6. CONCLUSION

For a software system to be sustainable and efficient, it must be built on a stable foundation. The core of the software system should be such that it can allow adding new and changing existing functionalities, as well as adding new technologies, with minimal changes. This is made possible by using the Abstract Factory pattern, which is based on the MVC pattern. In our study example, we have shown how, relatively easily, the existing software system (Figure 5) can be integrated with JavaFX GUI technology (Figure 7).

The further research directions refer to the extension of the observed matrix structure achieved by integrating JavaFX and Swing on the one hand and different implementations of screen forms on the other hand. Namely, a further step in the development of this structure will be made by the introduction of new Java graphic user interface technologies.

In this way, there will be an integration of different desktop, Android, and web GUI technologies in the application. The goal of the research is to obtain a program code generator that, based on the choice of the desired technology and the appearance of the screen forms, will produce a program code that meets the defined requirements.

REFERENCES

[1] S. Vlajić, *Softverski Paterni*, ISBN: 978-86-86887-30-6, Zlatni Presek, 2014.

[2] E. Gamma, R. Helm, R. Johnson, R. E. Johnson & J. Vlissides, "Creational Patterns," in *Design patterns: elements of reusable object-oriented software*, Pearson Deutschland GmbH, 1995.

[3] 3K. Sage, *Concise Guide to Object-Oriented Programming,* Springer International Publishing, 2019.

[4] A. Korunović. (2022). SoftwareSystem-FON [Source code]. https://github.com/AnaKorunovic/SoftwareSystem-FON.

[5] S. Vlajić, D. Savić, V. Stanojević, I. Antović & M. Milić, *Napredne Java tehnologije*, ISBN: 978-86-86887-03-0, Zlatni presek, 2008. S. Vlajić, *Softverski Paterni*, ISBN: 978-86-86887-30-6, Zlatni Presek, 2014.

[6] H. Schildt, *Java The Complete Reference*, *7th ed*., The McGraw-Hill Companies, 2007.
[7] Y. D. Liang, "JavaFx Basics," in *Introduction To Java Programming, Comprehensive Version,* Pearson Education India, 2009, pp 535-584.
[8] G. Kruk, O. Alves, L. Molinari, E. Roux, " Best practices for efficient development of JavaFX applications," Proceedings of the 16th International Conference on Accelerator and Large Experimental Control Systems, ICALEPCS 2017, Barcelona, Spain, 2017.
[9] K. Sharan, Beginning Java 8 APIs, Extensions and Libraries: Swing, JavaFx, JavaScript, JDBC and Network programming APIs, Apress, 2014.
[10] P. Xiao, "Java programming for Windows Application," in *Practical Java programming for IoT, AI, and Blockchain*, 1st ed., John Wiley & Sons, Inc., 2019, pp 99-127.
[11] S. S. Chin, J. Vos, J. Weaver, "JavaFx Fundamentals," in *The Definitive Guide to Modern Java Clients with JavaFX,* Apress, 2019, pp 33-80
[12] S. Vlajić, *Projektovanje Softvera (Skripta – Radni material)*, Dr Siniša Vlajić, 2020.