Department of Computer Science
Technical University of Cluj-Napoca

Software Design

Project documentation

Name: Mera Mihai
Group: 30434

Contents

Chapter 1

Requirement

The requirement for this project is:
Problema 26
Dezvoltați o aplicație care poate fi utilizată ca soft educațional pentru studiul cercului. Aplicația va avea 2 tipuri de utilizatori: elev și administrator. Utilizatorii de tip elev pot efectua următoarele operații fără autentificare:

- desenarea interactivă a cercurilor prin înlocuirea creionului și a riglei cu mouse-ul și alegerea stilului de desenare (culoare, stil linie, grosime linie);
- calcularea și afișarea unor proprietăți: aria unui cerc, lungimea unui cerc, aria unui sector de cerc, lungimea unui arc de cerc;
- vizualizarea unor cercuri particulare:
  - cercul circumscris unui poligon (dacă poligonul este inscriptibil)
  - cercul înscris (dacă poligonul este circumscriptibil)
- salvarea/încărcarea unui cerc într-un/dintr-un fișier xml;
- solicitarea unui cont pentru testarea cunoștințelor

Utilizatorii de tip elev pot efectua următoarele operații după autentificare:

- verificarea cunoștințelor prin efectuarea unui test de 10 întrebări (alese aleator dintr-un set de 50 de întrebări) și vizualizarea punctajului obținut după finalizarea testului.

Utilizatorii de tip administrator pot efectua următoarele operații după autentificare:

- Operații CRUD pentru informațiile legate de utilizatorii aplicației care necesită autentificare;
- Vizualizarea listei tuturor utilizatorilor care necesită autentificare.

Chapter 2

Technologies

- Java Swing - Java Swing is a GUI (Graphical User Interface) widget toolkit for Java, which allows developers to create desktop applications with a rich graphical interface. It includes a variety of components such as buttons, text fields, labels, tables, and menus, which can be customized and arranged to create a user-friendly interface. Java Swing is part of the Java Foundation Classes (JFC) and provides a platform-independent API that can be used on different operating systems.
- Postgresql - Postgresql is a powerful open-source relational database management system (RDBMS) that is designed to handle large amounts of data and complex queries. It offers a wide range of features, including support for SQL (Structured Query Language), ACID (Atomicity, Consistency, Isolation, Durability) compliance, and concurrency control. Postgresql can be used for a variety of applications, including web applications, business intelligence, data warehousing, and geospatial data. It is known for its stability, reliability, and scalability, and is widely used in both small and large enterprises.
- Graphics2D - Graphics2D is a powerful 2D graphics API that is part of the Java 2D API. It allows developers to create and manipulate 2D shapes, lines, text, and images in Java. Graphics2D provides a wide range of features, including antialiasing, transformations, compositing, and text rendering, which can be used to create high-quality graphics and animations. Graphics2D is built on top of the Graphics class in Java, which provides basic 2D drawing operations. However, Graphics2D provides a more powerful and flexible API, with additional methods for creating complex shapes, handling transparency and compositing, and applying affine transformations. Graphics2D can be used to create a wide range of graphical applications, including games, scientific simulations, data visualization, and user interfaces.
- JAXB - The Java Architecture for XML Binding (JAXB) is a technology that enables Java developers to map Java classes to XML representations and vice versa. It provides a convenient way to convert XML documents to Java objects and back, which can be used in a variety of scenarios, including data binding, web services, and XML processing. JAXB includes a set of annotations that can be added to Java classes to specify how they should be mapped to XML documents. These annotations can be used to define element names, attribute names, and the order of elements in the XML document, among other things. JAXB also provides a set of APIs that can be used to marshall (convert Java objects to XML) and unmarshall (convert XML to Java objects) XML documents. These APIs make it easy to work with XML documents in Java applications and can be used with a variety of XML parsers.
- Maven Binding - The net.sds.mvvm.bindings.* package contains classes that are

used to bind properties between different components of a Model-View-ViewModel (MVVM) architecture. The purpose of the bindings is to provide a way to synchronize data between different parts of the application, so that changes in one component can be propagated to others in a consistent and automatic way. The Binding interface is the foundation of the package, and defines the basic behavior of a binding. A Binding connects two properties together, so that changes in one property are automatically reflected in the other. The Binding interface defines a bind() method that sets up the binding, and an unbind() method that removes the binding. The BindingFactory class is used to create Binding instances. It provides a number of static methods that create bindings for different types of properties, such as BooleanProperty, IntegerProperty, and StringProperty. In addition to the basic Binding interface, the package provides a number of specialized bindings that handle more complex scenarios. For example, the ListBinding interface can be used to bind a list property to a UI component such as a table or list view. The ObjectBinding interface can be used to bind an object property to a UI component such as a form or dialog.

Chapter 3

Diagram description

3.1 Model Geometry

The GeometryObject class is an interface that doesn't define any methods or fields. It's used as a common interface for all geometry objects in this codebase, meaning that any class implementing GeometryObject is considered a geometry object.
The Circle class represents a circle in two-dimensional space. It implements the GeometryObject interface and has two fields: origin and radius. The computeArea() method calculates the area of the circle using its radius and returns it, while the computePerimeter() method calculates the perimeter of the circle and returns it.
The Line class represents a straight line segment in two-dimensional space. It implements the GeometryObject interface and has two fields: start and end, which are both instances of the Point class. The computeSlope() method calculates the slope of the line segment, while the computeLength() method calculates the length of the line segment. The computeAngle() method takes another Line object as an argument and calculates the angle between the two lines. The computeMiddlePoint() method calculates the middle point of the line segment. Finally, the toString() method returns a string representation of the line segment in the format "startPoint - endPoint".
The Point class represents a point in two-dimensional space. It implements the GeometryObject interface and has two fields: x and y. The computeDistance() method takes another Point object as an argument and calculates the distance between the two points. The toString() method returns a string representation of the point in the format "(x, y)".
The Polygon class represents a polygon in two-dimensional space. It implements the GeometryObject interface and has one field: vertices, which is an ArrayList of Point objects. The isInscribed() method returns a boolean indicating whether the polygon is inscribed, which is true if the polygon has exactly three vertices. Otherwise, it calculates the length and angle of each line segment in the polygon and returns false if any two line segments have different lengths or angles. The isCircumscribed() method returns a boolean indicating whether the polygon is circumscribed, which is true if all the vertices of the polygon lie on a circle. Finally, the getCircumcenter() method calculates the center of the circle that circumscribes the polygon by finding the intersection point of the perpendicular bisectors of any two sides of the polygon.

3.2 Model User

The Student class represents a student user in the system. It has five private instance variables: name, surname, nickname, password, and accountStatus. The first four variables store the student's personal information, while the accountStatus variable represents the status of the student's account, which is an instance of the AccountStatus enum. The Student class has two

constructors, one that takes in five parameters to initialize all of the instance variables, and another that takes in four parameters and sets the accountStatus to REQUESTED by default. The class also provides getters and a setter for the name variable.
The AccountStatus enum is used to define the possible states of a user's account. In this case, the possible states are REQUESTED, APPROVED, and ADMIN. The REQUESTED state is assigned by default when a new Student object is created, and it means that the user has requested to join the platform, but their account hasn't been approved yet. The APPROVED state means that the user's account has been approved, and they can now log in and use the platform. The ADMIN state is used to denote the account of an administrator who has special privileges on the platform.

3.3 Model Quiz

Difficulty enum is an enumeration that represents the difficulty level of a quiz question. It has three possible values: EASY, MEDIUM, and HARD, each with an associated integer value representing the difficulty level.
Question class is a class that represents a quiz question. It has four instance variables: a String representing the question text, a Difficulty representing the question difficulty level, an ArrayList of Strings representing the answer options, and a String representing the file name of an optional image associated with the question. It has a constructor that takes these four parameters and sets them as instance variables. It also has methods to get each instance variable, as well as a method to randomize the order of the answer options and return the index of the correct answer.
Test class is a class that represents a quiz test. It has three instance variables: an ArrayList of Questions representing the quiz questions, an integer representing the maximum number of points possible, and an ArrayList of integers representing the indices of the correct answer for each question. It has a constructor that generates a random set of 10 questions from a question database, computes the maximum number of points possible based on the difficulty level of each question, and shuffles the answer options for each question. It also has methods to get each instance variable.
TestTableEntry class is a class that represents an entry in a quiz test results table. It has five instance variables: an integer representing the index of the entry, two Strings representing the name and surname of the test taker, an integer representing the number of points earned on the test, and a Timestamp representing the time the test was taken. It has a constructor that takes these five parameters and sets them as instance variables. It also has methods to get each instance variable.

Chapter 4

Implementation details

4.1 Resources

DrawingCanvas extends the Canvas class in Java's AWT package. It represents a canvas for drawing and allows the user to draw points and circles. The DrawingCanvas class also handles user interactions, such as mouse clicks and key presses, to create and draw points and circles on the canvas.
The class has several fields, such as point, circle, containsPolygon, containsCircle, x, y, startX, startY, endX, and endY, which are used to keep track of the state of the canvas and the drawn shapes. The color and stroke fields represent the current color and stroke used for drawing.
The class has several public methods, such as displayError, setContainsPolygon, setContainsCircle, repaint, getColor, setColor, getStroke, and setStroke, which allow the user to interact with the canvas and modify its properties.
The Palette class is an enumeration that represents a color palette with various shades of blue. Each color in the palette is represented as an enum constant with a corresponding hexadecimal color code. The constructor of the enum takes a string parameter which is the hexadecimal representation of the color. The color() method returns a java.awt.Color object that corresponds to the enum constant's color. The class is useful for providing a set of pre-defined colors for a graphical user interface or for other visual applications.

4.2 Database

The class named "DatabaseCreator" is located in the package "repo". It has several methods for creating and filling tables in a database using SQL queries, as well as checking for existing tables.
The class contains four public methods: "createTableStudent()", "fillTableQuestions(String filename)", "createTestsTable()", and "insertAdmin()".
The method "createTableStudent()" creates a table named "STUDENTS" if it doesn't already exist. The table has columns for student name, surname, nickname, password, and account status. It returns a boolean value indicating whether the table was successfully created and an admin user was inserted.
The method "fillTableQuestions(String filename)" reads question data from a file and populates a table named "QUESTIONS". The table has columns for the question, its difficulty, and four possible answers, as well as an optional image file. It returns a boolean value indicating whether the table was successfully created and filled with data.
The method "createTestsTable()" creates a table named "TESTS" if it doesn't already exist. The table has columns for the student ID, test score, and the time the test was taken. It returns a boolean value indicating whether the table was successfully created.

The method "insertAdmin()" inserts a default admin user into the "STUDENTS" table, with a username, password, and account status of "ADMIN". It returns a boolean value indicating whether the insertion was successful.
The class uses another class named "Persistence" for handling the database connection and SQL queries. It also uses a class named "QuestionsPersistence" for inserting question data into the "QUESTIONS" table.

The enum class named DBConnectionInfo contains the details necessary for establishing a database connection. It has four constants named DRIVER, URL, USER, and PASSWORD, each representing the driver class name, the database URL, the username and password required for accessing the database, respectively. The constants are assigned their respective values in their constructors, which take a single String parameter representing the value to be assigned. The class also has a method getValue() which returns the value of the constant. This class is typically used in conjunction with a JDBC driver to establish a connection to a PostgreSQL database.

The Persistence class is used to connect to a PostgreSQL database and execute SQL queries.
The class has a constructor that tries to load the PostgreSQL JDBC driver using the value of the DRIVER constant defined in the DBConnectionInfo enum.
The class has a method called "connect()" that creates a connection to the PostgreSQL database using the values of the URL, USER, and PASSWORD constants defined in the DBConnectionInfo enum.
The class has a method called "close()" that closes the connection to the PostgreSQL database.
The class has a method called "executeQuery(String query)" that takes an SQL query as an argument, connects to the database using the "connect()" method, executes the query using a Statement object, and returns a boolean value indicating whether the query was successful or not.
The class has a method called "getDataTable(String query)" that takes an SQL query as an argument, connects to the database using the "connect()" method, executes the query using a Statement object, and returns a ResultSet object containing the result of the query.
Both the "executeQuery()" and "getDataTable()" methods use the "connect()" and "close()" methods to ensure that the connection to the database is properly established and closed after each query.

4.3 MVVM architecture

At a high level, the MVVM pattern is designed to help separate the concerns of a GUI application into three distinct components: the Model, the View, and the ViewModel. Each of these components has a specific role to play in the overall design of the application.
Here's a breakdown of what each of the components of MVVM is responsible for:
Model
The Model is responsible for representing the underlying data and business logic of the application. It might contain things like database access code, web service calls, or other data-related operations. Essentially, the Model is where all the "real work" of the application happens.

View
The View is responsible for presenting the user interface to the user. It is typically implemented using some combination of HTML, CSS, and/or other graphical technologies. The View knows how to display data to the user, and how to respond to user input.

ViewModel
The ViewModel sits between the Model and the View, acting as a kind of mediator between the two. It is responsible for exposing data and operations from the Model to the View in a way that is easy for the View to consume. The ViewModel also responds to user input from the View, and updates the Model accordingly.

The Command design pattern is a behavioral pattern that is commonly used in object-oriented programming. It is used to encapsulate a request as an object, thereby allowing the request to be treated as a first-class citizen within the program. This allows for greater flexibility and extensibility, as commands can be easily composed and manipulated at runtime.
At a high level, the Command design pattern consists of the following components:
Command interface
This is an interface that defines the basic contract for a command. It typically contains a single method, such as "execute" or "run", that is responsible for performing the command.

Concrete command classes
These are concrete implementations of the Command interface. Each concrete command class represents a specific command that the program can execute.

4.3.1 VMAccount

The VMAccount class is a ViewModel that manages the state and behavior of the AccountView in a Java Swing application. It encapsulates the logic and data required for the view, providing a separation of concerns between the UI and the underlying data and logic.
The class contains properties for the user's nickname and a table model, which are implemented using the Property class from the net.sds.mvvm.properties package. These properties provide change notification and binding support, allowing other components to subscribe to changes in the values and update the UI accordingly. By using bindings, the VMAccount class can communicate with other ViewModel classes and Views in a decoupled manner.
Additionally, the class provides commands for going back, retrieving tests, and taking a test. These commands are implemented using the ICommand interface, which allows the ViewModel to respond to user actions in a testable and maintainable way. The goBack command is responsible for navigating back to the previous view, while the retrieveTests and takeTest commands handle the retrieval and taking of tests respectively.

4.3.2 VMAdmin

The VMAdmin class is a ViewModel that represents the functionality of an admin view in the application. The ViewModel has properties for the table model, row, and commands for approving, deleting, seeing all, and seeing requests. The properties are bound to the view using the PropertyFactory class provided by the MVVM framework.
The model property is of type DefaultTableModel, which represents a table model for the view. The row property is of type Integer and represents the selected row in the table. The ViewModel also has commands that are executed when certain actions are taken by the user. The commands are bound to the corresponding buttons in the view.
The ViewModel acts as a mediator between the view and the underlying data model. It pro-

vides the necessary data and behavior for the view to operate correctly. The bindings between the ViewModel and the view ensure that the view is updated automatically when the data changes. Overall, the VMAdmin class provides a clean and efficient way to implement the functionality of an admin view in a software application.

4.3.3 VMCanvas

The VMCanvas class is a ViewModel class that connects the CanvasView, the view responsible for drawing on the canvas, with the commands that manipulate it. It contains various ICommand instances that are used to change the color, pattern, and stroke of the drawings, display information, and draw shapes like inscribed and circumscribed circles. The class also includes commands for opening the login and account views, uploading and downloading XML files, and other functionalities.
The class follows the MVVM (Model-View-ViewModel) architecture, which separates the view and the model using the ViewModel to bind the two. The CanvasView is responsible for the drawing, while the VMCanvas contains the commands that manipulate the view. The ICommand instances encapsulate the operations that can be performed on the view and are bound to the view's events through data binding. Overall, the VMCanvas class plays an essential role in connecting the view with the commands that control its behavior, making it an integral part of the application's functionality.

4.3.4 VMDrawingCanvas

The VMDrawingCanvas class is a ViewModel class that handles the business logic of the DrawingCanvas view. It contains commands that correspond to the user actions performed on the DrawingCanvas, such as adding a circle, a line, a point, or creating a polygon. The class also maintains an ArrayList of GeometryObject instances, which represent the objects drawn on the DrawingCanvas. The class provides methods for adding and retrieving Circle and Polygon instances from the ArrayList.
The class uses the Command design pattern to encapsulate the user actions as commands that can be executed or undone. The class instantiates the Command objects and assigns them to the corresponding commands declared as public fields. When a command is executed, it calls the appropriate method in the VMDrawingCanvas class, which modifies the ArrayList of GeometryObject instances accordingly. The DrawingCanvas view listens to changes in the ArrayList and updates the display accordingly.
The VMDrawingCanvas class is tightly bound to the DrawingCanvas view and provides a way to separate the view logic from the business logic. The class can be easily unit tested and maintained separately from the view.

4.3.5 VMLogin

The VMLogin class is responsible for managing the data and behavior of the LoginView. It contains properties for the nickname and password fields, as well as commands for the login, reset and go back actions.
The nickname and password properties are created using the PropertyFactory class from the net.sds.mvvm.properties package. These properties are used to store the values entered by the user in the corresponding fields of the LoginView.

The login, reset and goBack commands are instances of classes that implement the ICommand interface. These commands are used to handle the login, reset and go back actions respectively. When the user interacts with the LoginView and triggers one of these actions, the corresponding command is executed.

4.3.6 VMNewAccount

This code defines the VMNewAccount class, which is a ViewModel class responsible for handling the user input and interaction with the NewAccountView. The class has four properties, name, surname, nickname, and password, which are all instances of the Property class defined in the net.sds.mvvm.properties package. The class also has two ICommand fields, request and reset, which represent the commands to request a new account and to reset the form fields.
The constructor of the VMNewAccount class initializes the fields of the class, including the name, surname, nickname, and password properties and the request and reset commands. The constructor takes a NewAccountView parameter, which represents the view associated with this ViewModel.
The class provides getter methods for each of its properties and the associated view. These methods allow other parts of the application to retrieve the values of the properties and the associated view.

4.3.7 VMTest

This code defines the VMNewAccount class, which is a ViewModel class responsible for handling the user input and interaction with the NewAccountView. The class has four properties, name, surname, nickname, and password, which are all instances of the Property class defined in the net.sds.mvvm.properties package. The class also has two ICommand fields, request and reset, which represent the commands to request a new account and to reset the form fields.
The constructor of the VMNewAccount class initializes the fields of the class, including the name, surname, nickname, and password properties and the request and reset commands. The constructor takes a NewAccountView parameter, which represents the view associated with this ViewModel.
The class provides getter methods for each of its properties and the associated view. These methods allow other parts of the application to retrieve the values of the properties and the associated view.