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;
- vizualizarea unor cercuri particulare:
  - cercul circumscris unui poligon (dacă poligonul este inscriptibil)
  - cercul înscris (dacă poligonul este circumscriptibil)
- 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.

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 class has a DrawingCanvasPresenter instance, which is used to interact with the presenter layer of the program. 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 getDrawingCanvasPresenter method returns the DrawingCanvasPresenter instance associated with the canvas.
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.

Overall, this code provides a simple and reusable way to connect to a PostgreSQL database and execute SQL queries.

4.3 Presenter

The AccountPresenter class is part of the presenter layer and serves as the intermediary between the AccountView and the data access layer. The class has a reference to an instance of AccountView and provides methods to retrieve student tests, take a test, and go back to the previous screen.

The retrieveStudentTests method retrieves the tests of the current student and returns them as a DefaultTableModel object, which can be used to populate a JTable. The method uses an instance of StudentPersistence to find the student ID based on their nickname, and an instance of TestPersistence to retrieve the list of TestTableEntry objects associated with that student

ID. The method then creates a new DefaultTableModel object and adds columns for index, name, surname, points, and timestamp. It then loops through the TestTableEntry objects and adds a new row to the DefaultTableModel for each entry, using the addRow method.

The takeTest method is called when the user clicks the "Take Test" button in the AccountView. The method uses an instance of StudentPersistence to find the student ID based on their nickname, creates a new TestView object with that ID as a parameter, and sets the visibility of the current AccountView to false. The TestView is a new window where the user can take a quiz.

The goBack method is called when the user clicks the "Back" button in the AccountView. The method sets the visibility of the CanvasView (the previous screen) to true and disposes of the current AccountView.

The provided code shows the implementation of the AdminPresenter class, which serves as an intermediary between the AdminView and the StudentPersistence and TestPersistence repositories. The class provides methods for approving, deleting, and viewing all students and their requests. The methods use the DefaultTableModel class to present the data in a tabular format, and display relevant messages using JOptionPane. Additionally, the class provides a back() method to return to the previous view. Overall, the class is designed to facilitate the management of student accounts by the administrator.

The CanvasGeometryPresenter is responsible for handling user interactions with the CanvasView, which is a graphical user interface element that displays geometric shapes drawn on a canvas.

The CanvasGeometryPresenter class contains several methods for modifying the appearance of the shapes on the canvas, such as changing their color, stroke width, and stroke pattern. It also provides methods for displaying information about a circle drawn on the canvas, and for drawing a circumscribed or inscribed circle around a polygon. Overall, this class acts as an intermediary between the user interface and the underlying data model, allowing the user to interact with the shapes on the canvas in a meaningful way.

The CanvasUserPresenter is a presenter class that is responsible for handling user interactions related to logging in and creating a new account. It has access to a CanvasView object, which it uses to hide the current view and display the appropriate login or new account view.

The openLogin() method is called when the user clicks on the "Login" button in the canvas view. It hides the canvas view and creates a new LoginView object, passing in the CanvasView object as a parameter. The LoginView is then displayed, allowing the user to enter their login credentials.

Similarly, the openNewAccount() method is called when the user clicks on the "New Account" button in the canvas view. It hides the canvas view and creates a new NewAccountView object, passing in the CanvasView object as a parameter. The NewAccountView is then displayed, allowing the user to create a new account.

The class contains methods to add these shapes to the canvas, create a polygon using a set of points, clear the canvas, and retrieve the circle or polygon object currently on the canvas. The class also handles the graphics rendering by getting the graphics object from the canvas and using it to draw the shapes.

The code follows the Model-View-Presenter (MVP) pattern, where the presenter acts as an intermediary between the model (geometric shapes) and the view (drawing canvas). The presenter receives user input from the view, manipulates the model objects accordingly, and updates the view with the new data. This separation of concerns makes the code more modular

and easier to maintain.

The LoginPresenter class handles the user login process. It receives input from the LoginView and interacts with the StudentPersistence to retrieve a Student object from the database. If the Student is found, it checks the AccountStatus of the Student. If the AccountStatus is ADMIN, it creates a new AdminView and hides the LoginView. If the AccountStatus is APPROVED, it creates a new AccountView with the student's information and hides the LoginView. If the Student is not found in the database, it displays an error message on the LoginView. The reset() method is called to clear the text fields in the LoginView.

The LoginPresenter is responsible for controlling the flow of the login process and handling any errors that may occur. It interacts with the StudentPersistence to retrieve data from the database and with the LoginView to update the UI. The LoginPresenter ensures that the appropriate view is displayed based on the AccountStatus of the Student. The reset() method is used to ensure that the LoginView is cleared after a user logs in or encounters an error. The LoginPresenter is an important component of the login process and helps to ensure that the user experience is smooth and error-free.

The NewAccountPresenter class handles the actions related to the creation of a new account in the system. It interacts with the NewAccountView class to retrieve user input and display information to the user.

The class contains a constructor that takes an instance of the NewAccountView class and sets it to a private field. Additionally, it sets the visibility of the canvas view to false.

The request() method handles the creation of a new account. It retrieves the user input from the view and validates it. If the nickname entered by the user is not already in use, the method creates a new Student object and inserts it into the database using the StudentPersistence class. If the insertion is successful, the method displays a message to the user informing that the request has been sent and shows the canvas view. Otherwise, it displays an error message. The reset() method clears the input fields in the view.

Overall, the NewAccountPresenter class provides a way for users to create new accounts in the system and manages the communication between the view and the persistence layer.
The TestPresenter class is responsible for controlling the behavior of the TestView, which is the graphical interface where the user can take a quiz. The TestPresenter handles the creation of a new Test object, which contains a list of Question objects and the corresponding correct answers. The TestPresenter also keeps track of the user's progress through the quiz, calculates their score, and saves the score to the database when the quiz is completed.

The start() method initializes the Test object and sets the index and points to zero. The next() method is responsible for moving to the next question in the quiz. It first checks the user's answer for the current question and updates their score accordingly. It then displays the next question in the TestView, filling in the question text, options, and any associated image. The end() method is called when the user has completed the quiz, and it calculates the final score and saves it to the database.