Java Advanced Objects

CSC 18B

Dr. Mark Lehr

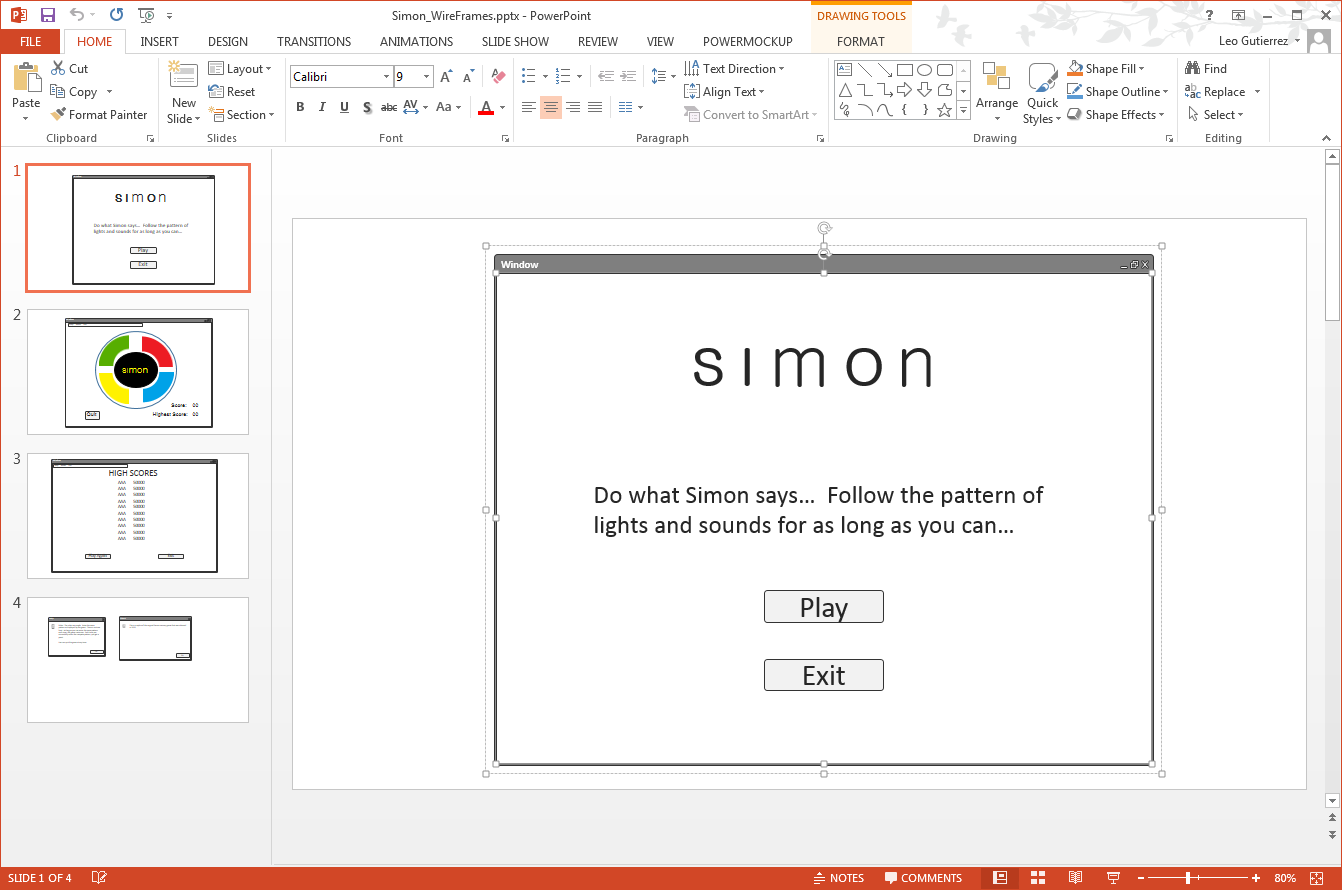
Riverside City College

Spring 2015

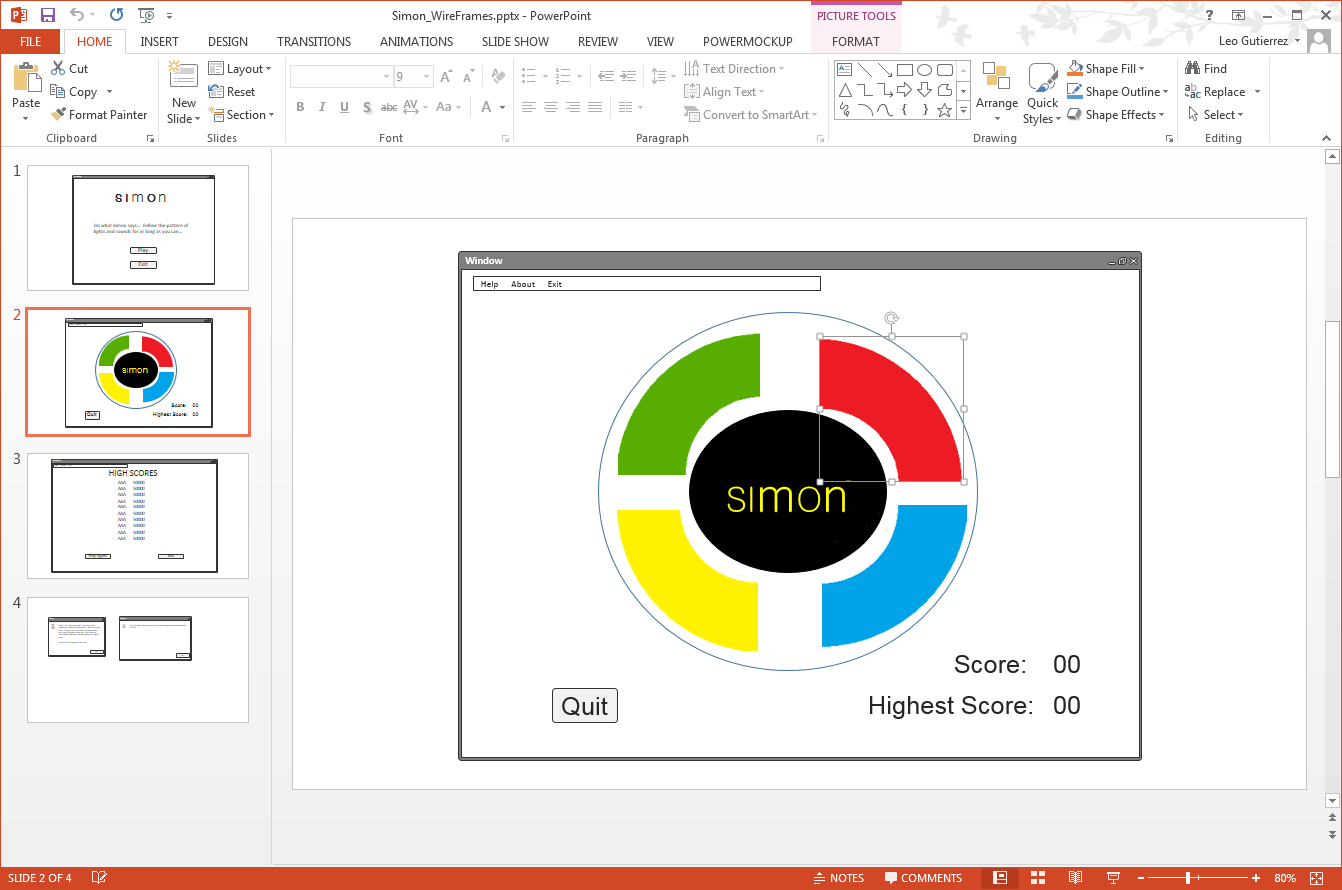
Leo Gutierrez

Assignment 1 Review Chapter 12 GUI part 1.

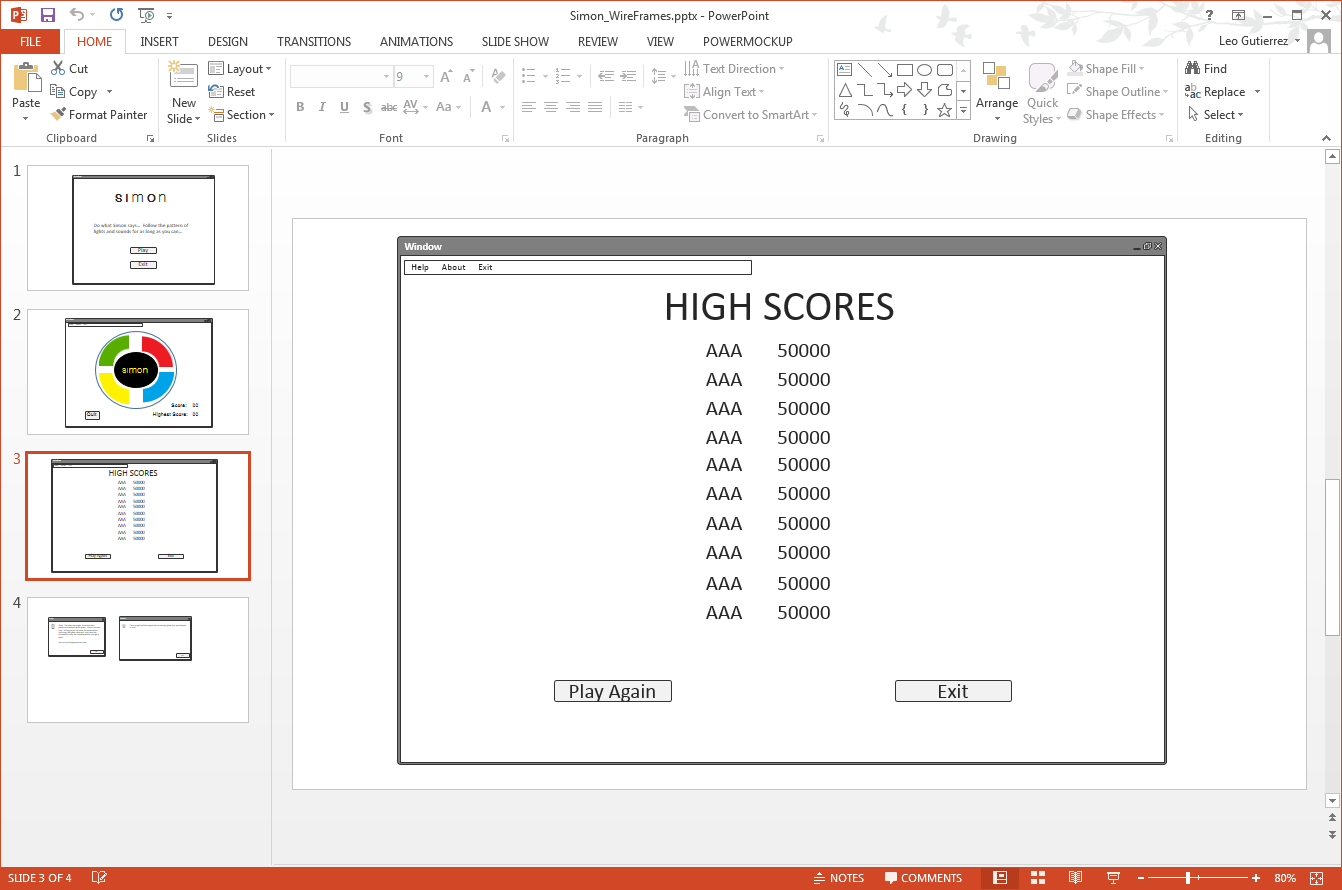
The initial wire frames when the concept of the game was started are below and in the following 3 pages.



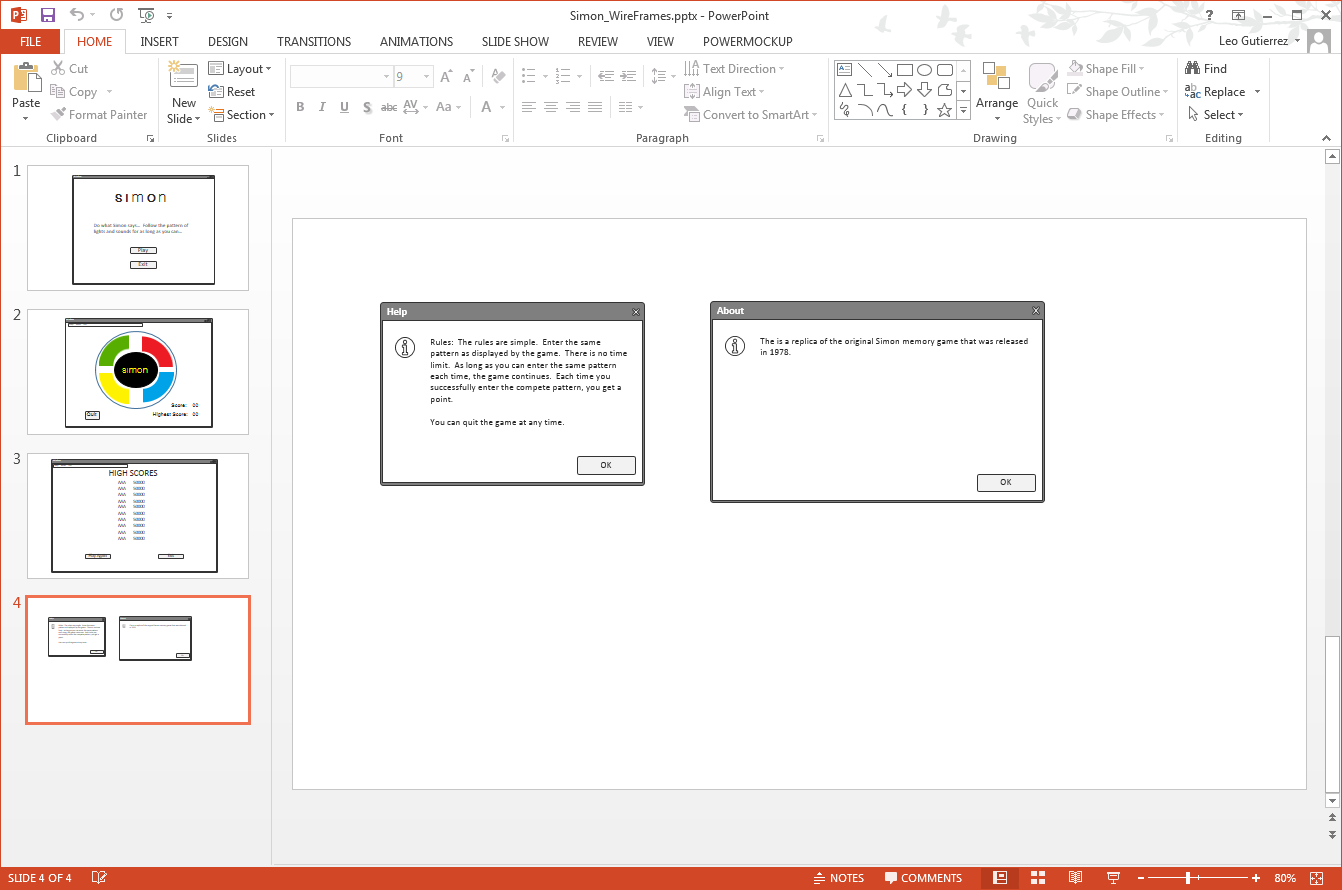
The Simon Game Panel



The Game Score Panel



The help and about menu items.



Assignment 2 – Implement the GUI interface with the GUI builder provided with NetBeans

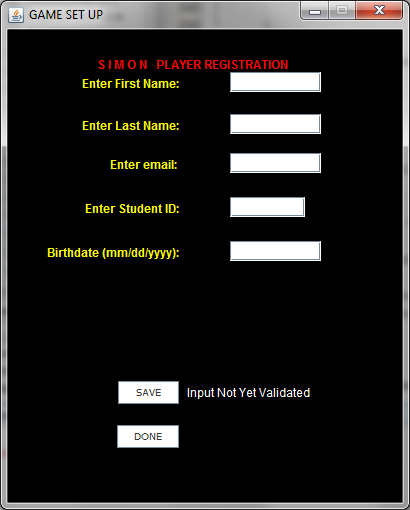
Some of the screen were created using the GUI Builder in NetBeans, some were created with GUI Developer in Eclipse and some were done on the fly without either.

The following components are the complete GUI interface for the Simon Game.

This splash screen will change colors when the game first comes up. Colors will appear to be scrolling from left to right.

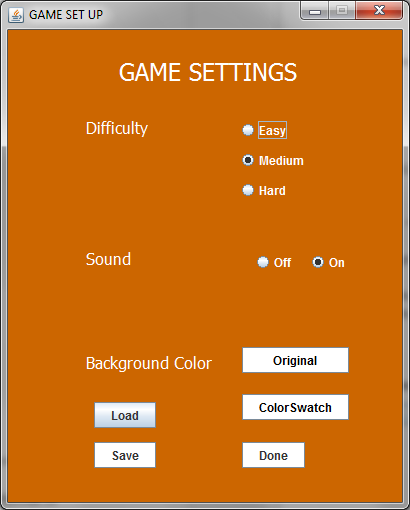


The first screen to appear is the user registration screen. It uses Regex to validate the entry of a user.



1. The First Name and Last Name must each start with a capital letter and be no more than 25 characters long.
2. The email address can be any kind of valid email address so long as it fits the format of [abcde12345@myservice.com](mailto:abcde12345@myservice.com) (or .edu or .net or various other variations).
3. The student id follows the form of RCC’s student id format, which is 2 alphas followed by 7 integers (AA9999999).
4. Birthday must be a valid date in the form of mm/dd/yyyy.
5. SAVE button will save the user to a database
6. DONE button is for indicating that you are done registering. If you want to bypass registering altogether, just click DONE upon entering this screen and leave all fields blank.

After the registration screen, you will come across the Settings screen below.



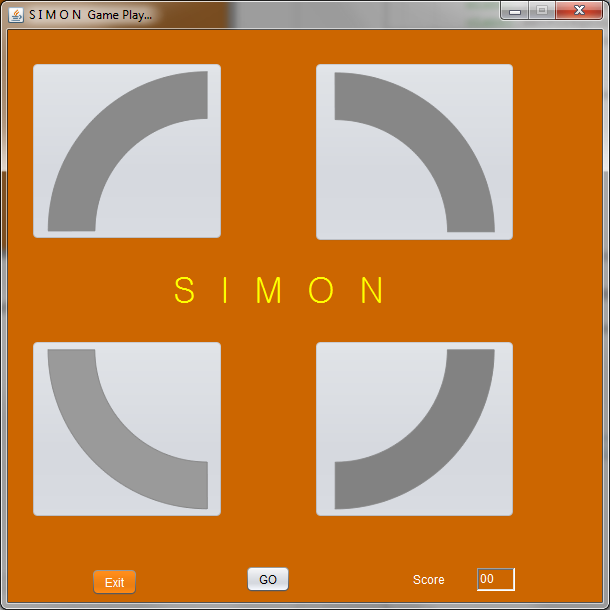
The setting screen allows you to choose a level of difficulty, leaving the sound on and off and changing the color of the playing screen. To use the default colors as seen on the GAME SETTINGS screen, just click the DONE button. If you want to load a previously saved setting, click on the load setting and it will take you to MyDocument or any other folder the user wishes to search. Clicking save will save whatever the current settings are in a default file called settings.ser and will let you choose a file name for yourself to load later.

After the setting screen, the SIMON WELCOME screen will come up.



This screen just gives basic instructions on the game with an EXIT button to leave or PLAY button to start the game.

The game screen looks like below. The game buttons have been disabled until the GO button has been pressed to begin the game.

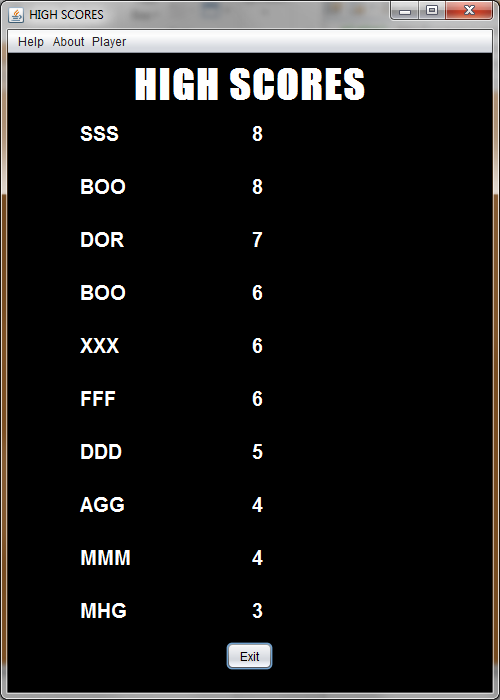


The GO button is to begin the game. The EXIT button will allow you to end the game at any time and the your score is kept on the bottom right.

Once the go button is clicked, the game buttons will become enabled and the first button will have come on and off.

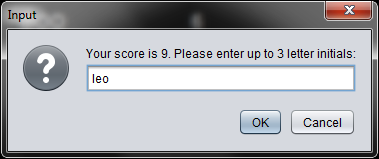


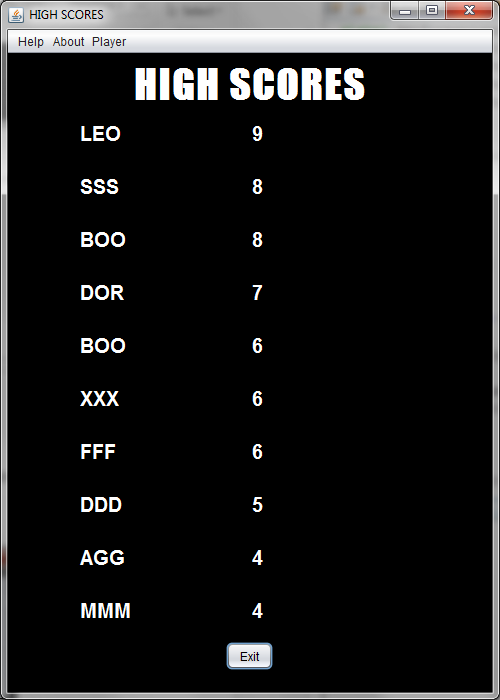
At the end of the game whether the user wins or loses, the score screen comes up.



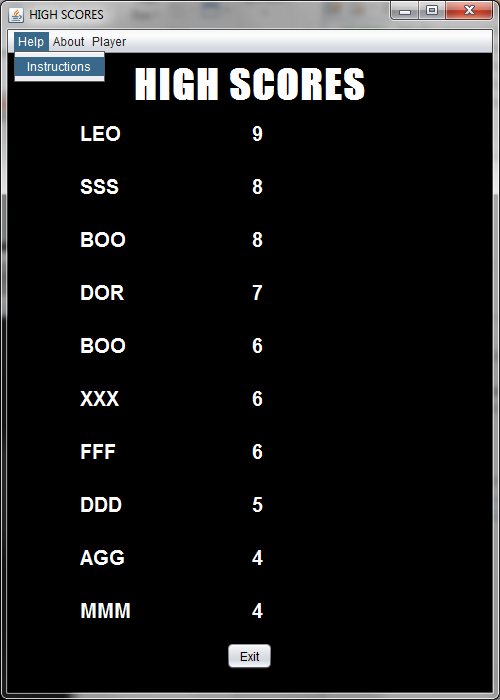
The top 10 scores are displayed being read in from a file and displayed in ascending order.

In the case that your score is higher than the last score on the list, you will be asked to enter your initials. Whatever is typed in, will be truncated to 3 upper case characters.

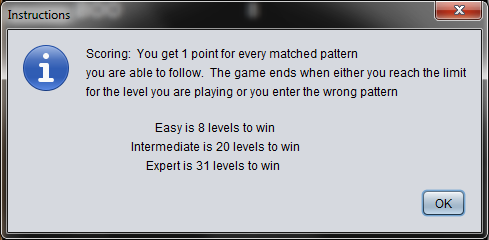




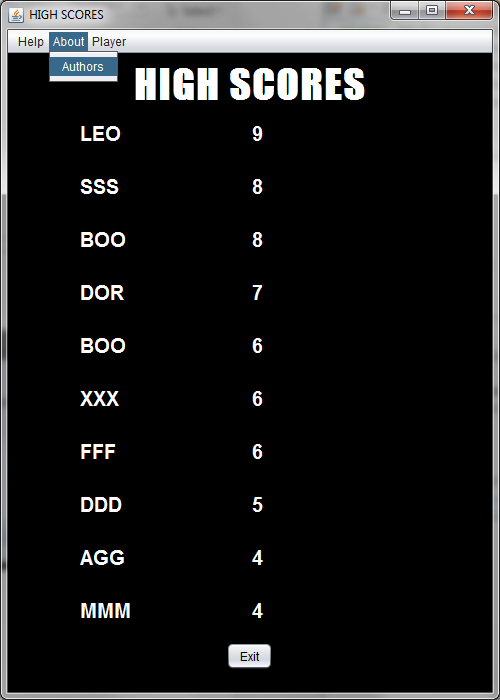
There is a help menu that will have some more detailed instructions to the game.

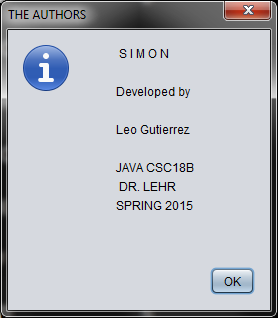


The pop up information pane will appear.

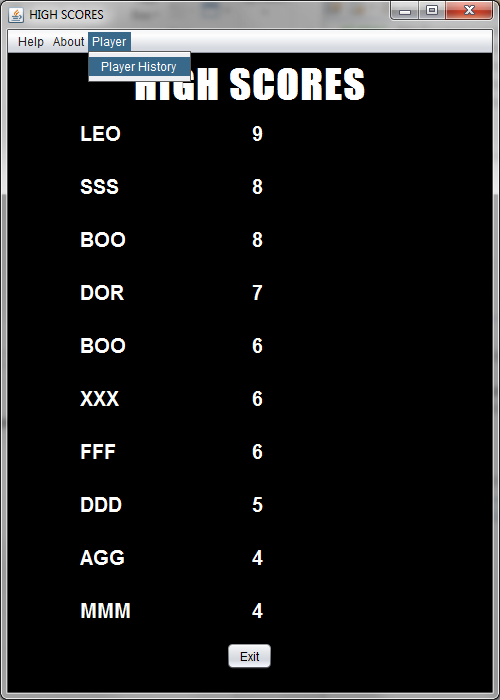


The about menu has information about the author of the game (me) and the course I took.





The last menu item will have a history button to pull the game history for the registered player from a database.



The program will read the database and pull all game history only for the user that is currently playing, displaying their name in the title of the dialogue box.



Assignment 3 – Create a Splash Screen

The following code generates the Splash Screen for the Simon Game as seen on page 6 of this document. The class is called SplashScreen.java.

package SimonPackage;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.Toolkit;

import javax.swing.ImageIcon;

import javax.swing.JWindow;

public class SplashScreen extends JWindow{

//Get image that will be used as the splash screen image.

Image splashImage = Toolkit.getDefaultToolkit().getImage("SimonSplash.gif");

//Declare a new image icon to place on the screen

ImageIcon splash = new ImageIcon(splashImage);

//Declare the player registration frame to call after the splash screen has concluded

RegisterFrame rf = new RegisterFrame();

//Splash screen constructor

public SplashScreen(){

// try statement to display the splash image

try

{

//Set splash image default location

setSize(splash.getIconWidth(),splash.getIconHeight());

setLocationRelativeTo(null);

//Set the splash screen to visible

setVisible(true);

//Set the splash screen to display for 4 seconds

Thread.sleep(4000);

//When the splash screen concludes, close and dump from memory

dispose();

//Set the Player registration frame on for player registration

rf.setVisible(true);

}

catch (Exception e)

{

System.err.println("Exception in loading splash");

e.printStackTrace();

}

}

// Draw the splash screen image

public void paint(Graphics g){

g.drawImage(splashImage,0,0,this);

}

// Main to initiate splash screen

public static void main(String args[]) {

SplashScreen ss = new SplashScreen();

}

}

Assignment 4 Regular Expressions

Use regular expressions on all forms.  You must check for numerical, dates, and strings for your forms.  Yes, you will need to create forms need or not.

The registration screen was created with Regular Expression programming as seen on page 7 of this document. The code that invokes this logic is found in class ValidateInput.java. The program is below.

package SimonPackage;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class ValidateInput {

//private static Pattern pattern;

private static Pattern DATE\_PATTERN =

Pattern.compile("(0?[1-9]|1[012])/(0?[1-9]|[12][0-9]|3[01])/((19|20)\\d\\d)");

private static Matcher matcher;

public ValidateInput(){

}

// Validate first name.

public static boolean validateFirstName(String firstName){

return firstName.matches("^[A-Z][a-zA-Z]{1,20}$");

}

// Validate last name.

public static boolean validateLastName(String lastName){

return lastName.matches("^[A-Z][a-zA-Z-'\\s]{1,20}$");

}

// Validate email

public static boolean validateEmail(String email){

return email.matches("^[\_A-Za-z0-9-]+(\\.[\_A-Za-z0-9-]+)\*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)\*(\\.[A-Za-z]{2,})$");

}

// Validate Student ID

public static boolean validateStudentID(String studentID){

return studentID.matches("^[a-z]{2}\\d{7}$");

}

//Validate date

public static boolean validateDate(final String todayDate){

System.out.println("you are in the validateDate method");

System.out.println("the string date is: " + todayDate);

Matcher matcher = DATE\_PATTERN.matcher(todayDate);

//matcher = pattern.matcher(todayDate);

if(matcher.matches()){

matcher.reset();

if(matcher.find()){

String month = matcher.group(1);

String day = matcher.group(2);

int year = Integer.parseInt(matcher.group(3));

if (day.equals("31") &&

(month.equals("4") || month .equals("6") || month.equals("9") ||

month.equals("11") || month.equals("04") || month .equals("06") ||

month.equals("09"))) {

return false; // only 1,3,5,7,8,10,12 has 31 days

} else if (month.equals("2") || month.equals("02")) {

//leap year

if(year % 4==0){

if(day.equals("30") || day.equals("31")){

return false;

}else{

return true;

}

}else{

if(day.equals("29")||day.equals("30")||day.equals("31")){

return false;

}else{

return true;

}

}

}else{

return true;

}

}else{

return false;

}

}else{

return false;

}

//return date.matches("(0?[1-9]|1[012])/(0?[1-9]|[12][0-9]|3[01])/((19|20)\\d\\d)");

}

}

Chapter 5 – Serialization

There are a couple of files that are being serialized in this game. (1) The scores at the end of the game and (2) the stored settings done at the beginning of the game. The registered users are also a serialized object, but we are using the data base to store those values.

TOP 10 SCORES class (TopScores.java)

code below:

package SimonPackage;

import java.io.Serializable;

//Class to define the top scores file.

public class TopScores implements Serializable{

// Declare class variables

public String initials;

public int score;

// Set and Get methods for the initials and scores

public TopScores(String initials, int score){

this.initials = initials;

this.score = score;

}

// Set initials

public void setInitials(String initials){

this.initials = initials;

}

// Get initials

public String getInitials(){

return initials;

}

// Set Score

public void setScore(int score){

this.score = score;

}

// Get Score

public int getScore(){

return score;

}

// The toString method for the scores file

@Override

public String toString(){

return String.format("%s %d,",

initials, score);

}

}

GAME SETTINGS class (GameSettings.java)

code below:

package SimonPackage;

import java.io.Serializable;

public class GameSettings implements Serializable{

private int fileStatus;

private int difficulty;

private int sound;

private int R;

private int G;

private int B;

public GameSettings(){

this(0, 0, 0, 0, 0, 0);

}

public GameSettings(int fileStatus, int difficulty, int sound, int R, int G, int B){

this.fileStatus = fileStatus;

this.difficulty = difficulty;

this.sound = sound;

this.R = R;

this.G = G;

this.B = B;

}

public void setFileStatus(int fileStatus){

this.fileStatus = fileStatus;

}

public int getFileStatus(){

return fileStatus;

}

public void setDifficulty(int difficulty){

this.difficulty = difficulty;

}

public int getDifficulty(){

return difficulty;

}

public void setSound(int sound){

this.sound = sound;

}

public int getSound(){

return sound;

}

public void setR(int R){

this.R = R;

}

public int getR(){

return R;

}

public void setG(int G){

this.G = G;

}

public int getG(){

return G;

}

public void setB(int B){

this.B = B;

}

public int getB(){

return B;

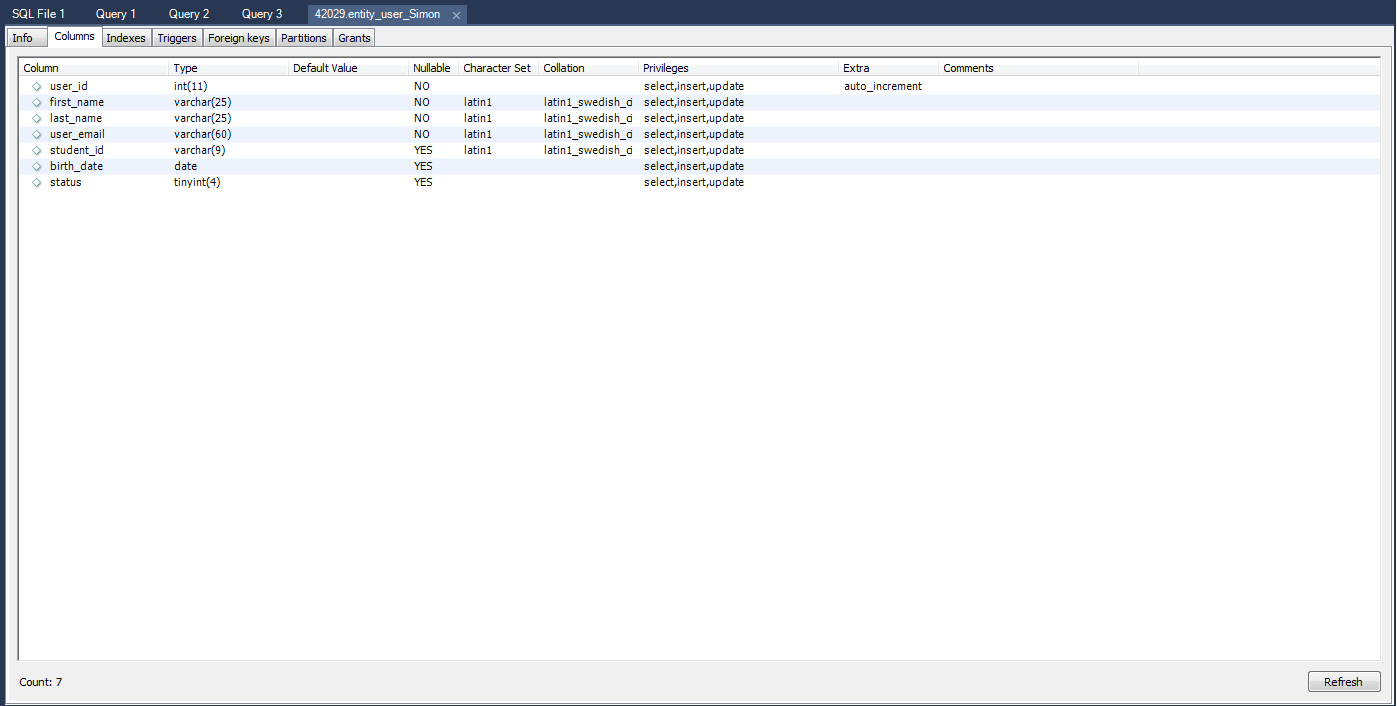
}

}

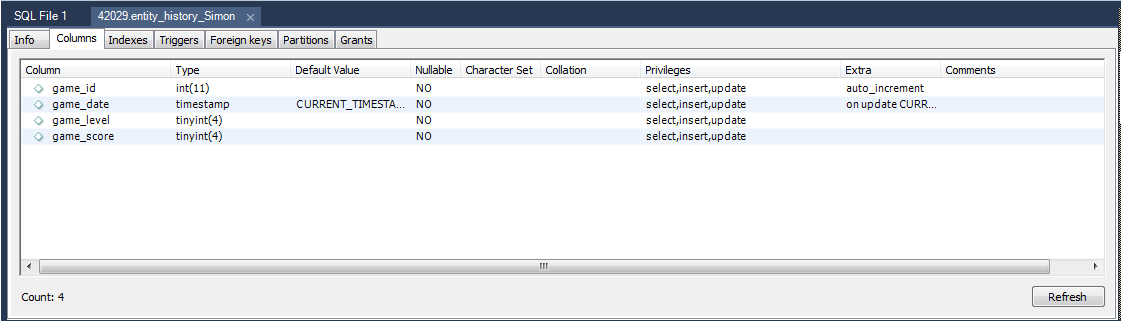
Assignment 6 – Determine what tables needed for the game

Here are the final set of tables that are being used for the Simon game. A user table, a game history table, a user game cross reference table, and a enumerated status table.

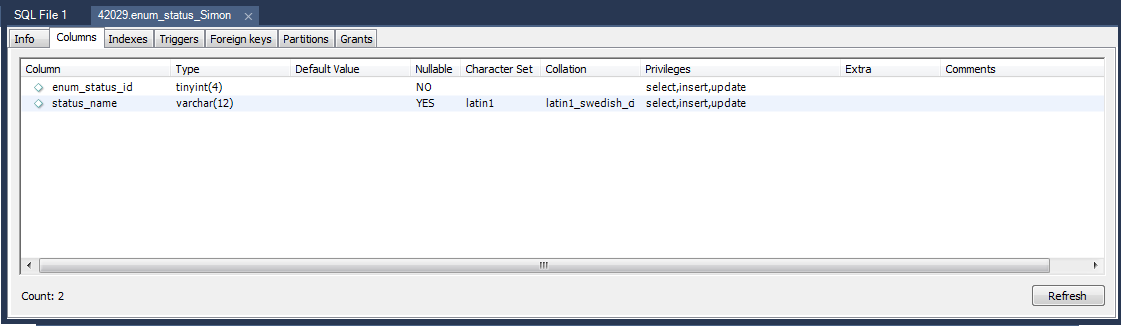
entity\_user\_Simon table



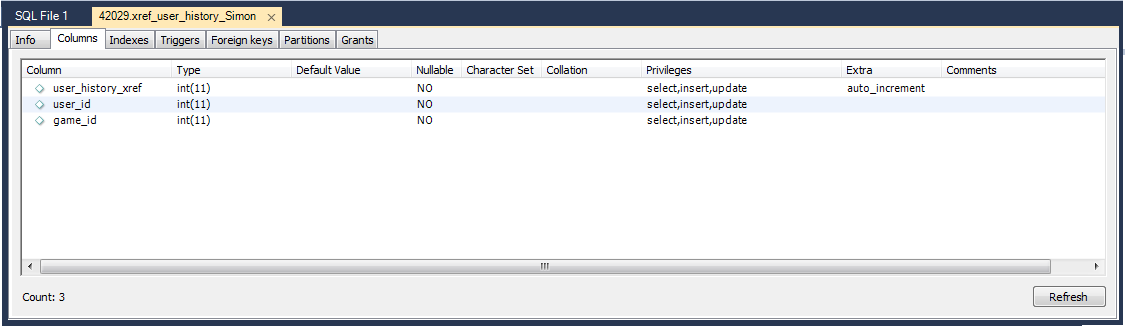
entity\_history\_Simon table



enum\_status\_Simon table



xref\_user\_history\_Simon



Assignments 7 and 8 – Add classes to program and run query against the database. Fill the database with data and run against the remote database.

Current data as of the making of this document:

1. entity\_user\_Simon table Data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| user\_id | first\_name | last\_name | user\_email | student\_id | birth\_date | Status |
| 1 | Andres | Gutierrez | aGutierrez@email.com | ag3003030 | 1/18/1998 | 1 |
| 2 | Andres | Gut | ag@email.com | ag0000000 | 1/18/1933 | 2 |
| 3 | Dorothy | Chan | dc@email.com | dc0303030 | 1/14/1961 | 1 |
| 4 | Ryan | Chan | rc@email.com | rc0033333 | 1/3/1988 | 1 |
| 5 | Katie | Wong | kw@email.com | kw3938838 | 1/19/1978 | 1 |
| 6 | Leo | Gutierrez | lg@email.com | lg2334999 | 1/19/1968 | 1 |
| 7 | Mario | Gutierrez | mg@email.com | mg0344000 | 1/24/1956 | 1 |
| 8 | Minnie | Mouse | mm@hotmail.com | mm4949494 | 1/10/1933 | 1 |
| 9 | Robert | Alva | ra@email.com | ra3030303 | 1/19/1969 | 1 |

1. entity\_history\_table Data

|  |  |  |  |
| --- | --- | --- | --- |
| game\_id | game\_date | game\_level | game\_score |
| 1 | 6/10/2015 0:00 | 2 | 6 |
| 2 | 6/10/2015 0:00 | 2 | 5 |
| 3 | 6/10/2015 0:00 | 2 | 3 |
| 4 | 6/10/2015 0:00 | 2 | 6 |
| 5 | 6/10/2015 0:00 | 2 | 4 |
| 6 | 6/10/2015 0:00 | 2 | 4 |
| 7 | 6/10/2015 0:00 | 2 | 11 |
| 8 | 6/10/2015 0:00 | 2 | 1 |
| 9 | 6/10/2015 0:00 | 2 | 1 |
| 10 | 6/10/2015 0:00 | 2 | 7 |
| 11 | 6/10/2015 0:00 | 2 | 3 |
| 12 | 6/10/2015 0:00 | 1 | 4 |
| 13 | 6/10/2015 0:00 | 1 | 2 |
| 14 | 6/10/2015 0:00 | 1 | 2 |
| 15 | 6/10/2015 0:00 | 1 | 2 |
| 16 | 6/10/2015 21:35 | 2 | 1 |
| 17 | 6/10/2015 0:00 | 2 | 9 |
|  |  |  |  |

1. entity\_status\_id Data

|  |  |
| --- | --- |
| enum\_status\_id | status\_name |
| 1 | Beginner |
| 2 | Casual |
| 3 | Intermediate |
| 4 | Advanced |
| 5 | Expert |

1. xref\_user\_history\_Simon Data

|  |  |  |
| --- | --- | --- |
| user\_history\_xref | user\_id | game\_id |
| 1 | 3 | 1 |
| 2 | 3 | 2 |
| 3 | 3 | 3 |
| 4 | 1 | 4 |
| 5 | 2 | 5 |
| 6 | 3 | 6 |
| 7 | 4 | 7 |
| 8 | 4 | 8 |
| 9 | 5 | 9 |
| 10 | 5 | 10 |
| 11 | 7 | 11 |
| 12 | 8 | 12 |
| 13 | 8 | 13 |
| 14 | 8 | 14 |
| 15 | 8 | 15 |
| 16 | 1 | 16 |
| 17 | 9 | 17 |

The following query is the query used in the program to view user history from the score screen.

SELECT entity\_user\_Simon.first\_name, entity\_user\_Simon.last\_name, entity\_history\_Simon.game\_score, entity\_history\_Simon.game\_level, enum\_status\_Simon.status\_name

FROM xref\_user\_history\_Simon, entity\_history\_Simon, entity\_user\_Simon, enum\_status\_Simon

WHERE xref\_user\_history\_Simon.game\_id = entity\_history\_Simon.game\_id

AND xref\_user\_history\_Simon.user\_id = entity\_user\_Simon.user\_id

AND enum\_status\_Simon.enum\_status\_id = entity\_user\_Simon.status

AND entity\_user\_Simon.user\_id = 3;

However, the query in the Simon game is not looking at a specific user\_id as in the example above, which is looking to pull only table rows that pertain to user\_id 3. We will use user\_id 3 as an example to show here.

The following is the result of a query in MySQL extracting the values for user\_id 3 for the query above just after user 3 player her game. The results, you will see, match the output from the game itself on the next page:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| first\_name | last\_name | game\_score | game\_level | status\_name |
| Dorothy | Chan | 6 | 2 | Beginner |
| Dorothy | Chan | 5 | 2 | Beginner |
| Dorothy | Chan | 3 | 2 | Beginner |
| Dorothy | Chan | 4 | 2 | Beginner |
| Dorothy | Chan | 7 | 2 | Beginner |

The following example is the result after player with user\_id of 3 plays her latest game.

Here is the SQL statement used in the Simon game with the result of the query from the game just below.

private String DEFAULT\_QUERY = "SELECT entity\_user\_Simon.first\_name, "

+ "entity\_user\_Simon.last\_name, "

+ "entity\_history\_Simon.game\_score, "

+ "entity\_history\_Simon.game\_level, "

+ "enum\_status\_Simon.status\_name "

+ "FROM xref\_user\_history\_Simon, "

+ "entity\_history\_Simon, "

+ "entity\_user\_Simon, "

+ "enum\_status\_Simon "

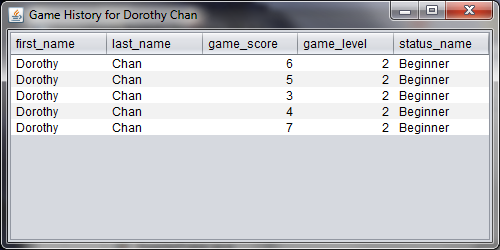
+ "WHERE xref\_user\_history\_Simon.game\_id = entity\_history\_Simon.game\_id "

+ "AND xref\_user\_history\_Simon.user\_id = entity\_user\_Simon.user\_id "

+ "AND enum\_status\_Simon.enum\_status\_id = entity\_user\_Simon.status "

+ "AND entity\_user\_Simon.user\_id = " + sf.getCurrentUserId();

The sf.getCurrentUserId() is used to obtain the current user id from the ScoreFrame class. It will then be used to match up to the database which was already updated, just moments ago, by the ScoreFrame class.



The follow is the GameHistoryFrame.java class that is used to obtain the display the table above for the user history. It is used in conjunction with the ResultSetTableModel.java class that follows right after.

GAME HISTORY CLASS

package SimonPackage;

//

// This program taken from example in the text and modified heavily to fit the needs

// of this game. Used in conjunction with the ResultSetTableModel class

//

//

import java.awt.BorderLayout;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import javax.swing.JFrame;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JOptionPane;

import javax.swing.table.TableRowSorter;

import javax.swing.table.TableModel;

public class GameHistoryFrame extends JFrame{

//variables to connect to a mysql database remotely

private static final String DATABASE\_URL = "jdbc:mysql://209.129.8.4:3306/";

private static final String USERNAME = "42029";

private static final String PASSWORD = "42029csc18b";

private static final String dbName = "42029";

private static final String driver = "com.mysql.jdbc.Driver";

private static String firstName = "firstName";

private static String lastName = "lastName";

private static String windowTitle;

private List<String> userArrayList;

private ScoreFrame sf = new ScoreFrame();

// default query retrieves all data from authors table

//private static final String DEFAULT\_QUERY = "SELECT \* FROM authors";

private String DEFAULT\_QUERY = "SELECT entity\_user\_Simon.first\_name, "

+ "entity\_user\_Simon.last\_name, "

+ "entity\_history\_Simon.game\_score, "

+ "entity\_history\_Simon.game\_level, "

+ "enum\_status\_Simon.status\_name "

+ "FROM xref\_user\_history\_Simon, "

+ "entity\_history\_Simon, "

+ "entity\_user\_Simon, "

+ "enum\_status\_Simon "

+ "WHERE xref\_user\_history\_Simon.game\_id = entity\_history\_Simon.game\_id "

+ "AND xref\_user\_history\_Simon.user\_id = entity\_user\_Simon.user\_id "

+ "AND enum\_status\_Simon.enum\_status\_id = entity\_user\_Simon.status "

+ "AND entity\_user\_Simon.user\_id = " + sf.getCurrentUserId();

private PreparedStatement searchSimonUser;

private static ResultSetTableModel tableModel;

public GameHistoryFrame(){

ResultSet resultSet = null;

// create ResultSetTableModel and display database table

try

{

// create TableModel for results of query SELECT \* FROM authors

tableModel = new ResultSetTableModel(

DATABASE\_URL+dbName, USERNAME, PASSWORD, DEFAULT\_QUERY);

Connection connection = DriverManager.getConnection(DATABASE\_URL+dbName, USERNAME, PASSWORD);

//search the simon user to get first and lat name

searchSimonUser = connection.prepareStatement("SELECT first\_name, last\_name FROM entity\_user\_Simon WHERE user\_id = ?");

searchSimonUser.setInt(1, sf.getCurrentUserId());

resultSet = searchSimonUser.executeQuery();

userArrayList = new ArrayList<String>();

while (resultSet.next()){

firstName = resultSet.getString("first\_name");

lastName = resultSet.getString("last\_name");

}

// create JTable based on the tableModel

JTable resultTable = new JTable(tableModel);

windowTitle = "Game History for " + firstName + " " + lastName;

this.setTitle(windowTitle);

add(new JScrollPane(resultTable), BorderLayout.CENTER);

final TableRowSorter<TableModel> sorter =

new TableRowSorter<TableModel>(tableModel);

resultTable.setRowSorter(sorter);

// ensure database is closed when user quits application

addWindowListener(

new WindowAdapter()

{

public void windowClosed(WindowEvent event)

{

tableModel.disconnectFromDatabase();

System.exit(0);

}

}

);

}

catch (SQLException sqlException)

{

JOptionPane.showMessageDialog(null, sqlException.getMessage(),

"Database error", JOptionPane.ERROR\_MESSAGE);

tableModel.disconnectFromDatabase();

System.exit(1); // terminate application

}

}

}

RESULT SET TABLE MODEL class

package SimonPackage;

//A TableModel that supplies ResultSet data to a JTable.

import java.sql.Connection;

import java.sql.Statement;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import javax.swing.table.AbstractTableModel;

//ResultSet rows and columns are counted from 1 and JTable

//rows and columns are counted from 0. When processing

//ResultSet rows or columns for use in a JTable, it is

//necessary to add 1 to the row or column number to manipulate

//the appropriate ResultSet column (i.e., JTable column 0 is

//ResultSet column 1 and JTable row 0 is ResultSet row 1).

public class ResultSetTableModel extends AbstractTableModel{

private final Connection connection;

private final Statement statement;

private ResultSet resultSet;

private ResultSetMetaData metaData;

private int numberOfRows;

//keep track of database connection status

private boolean connectedToDatabase = false;

//constructor initializes resultSet and obtains its meta data object;

//determines number of rows

public ResultSetTableModel(String url, String username,

String password, String query) throws SQLException

{

// connect to database

connection = DriverManager.getConnection(url, username, password);

// create Statement to query database

statement = connection.createStatement(

ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.CONCUR\_READ\_ONLY);

// update database connection status

connectedToDatabase = true;

// set query and execute it

setQuery(query);

}

//get class that represents column type

public Class getColumnClass(int column) throws IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

// determine Java class of column

try

{

String className = metaData.getColumnClassName(column + 1);

// return Class object that represents className

return Class.forName(className);

}

catch (Exception exception)

{

exception.printStackTrace();

}

return Object.class; // if problems occur above, assume type Object

}

//get number of columns in ResultSet

public int getColumnCount() throws IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

// determine number of columns

try

{

return metaData.getColumnCount();

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

}

return 0; // if problems occur above, return 0 for number of columns

}

//get name of a particular column in ResultSet

public String getColumnName(int column) throws IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

// determine column name

try

{

return metaData.getColumnName(column + 1);

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

}

return ""; // if problems, return empty string for column name

}

//return number of rows in ResultSet

public int getRowCount() throws IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

return numberOfRows;

}

//obtain value in particular row and column

public Object getValueAt(int row, int column)

throws IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

// obtain a value at specified ResultSet row and column

try

{

resultSet.absolute(row + 1);

return resultSet.getObject(column + 1);

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

}

return ""; // if problems, return empty string object

}

//set new database query string

public void setQuery(String query)

throws SQLException, IllegalStateException

{

// ensure database connection is available

if (!connectedToDatabase)

throw new IllegalStateException("Not Connected to Database");

// specify query and execute it

resultSet = statement.executeQuery(query);

// obtain meta data for ResultSet

metaData = resultSet.getMetaData();

// determine number of rows in ResultSet

resultSet.last(); // move to last row

numberOfRows = resultSet.getRow(); // get row number

// notify JTable that model has changed

fireTableStructureChanged();

}

//close Statement and Connection

public void disconnectFromDatabase()

{

if (connectedToDatabase)

{

// close Statement and Connection

try

{

resultSet.close();

statement.close();

connection.close();

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

}

finally // update database connection status

{

connectedToDatabase = false;

}

}

}

} // end class ResultSetTableModel

TABLE INSERTS

There were also several table INSERTS done in the game. The registration and score screens were used to update all necessary tables for the game history menu item to function properly.

Below is code from the RegisterPanel.java class that contains the java code for saving a registered user. It does 3 things, it first queries the database to see if a user is already registered. It does this by comparing email addresses. If an email address is already in the database, it flags the current user as a user that is already registered. It will then bypass the next two database functions and continue with the game with a registered user. If the email address is not found in the user table, then the two function immediately following are run. It updates the database with the new user information just received and then it rereads the database to obtain the most current user id by using the same logic as the first database read… it looks for the newly entered email address and then pulls the auto incremented user id just created. This user id will be saved by itself in a file to be passed into the user history function at the end of the game.

The following is the code from the RegisterPanel.java used to perform the 3 database functions.

REGISTER PANEL class

code below:

//Action for the SAVE button

private class SwingAction extends AbstractAction {

public SwingAction() {

putValue(NAME, "SAVE");

putValue(SHORT\_DESCRIPTION, "Input Complete");

}

public void actionPerformed(ActionEvent e) {

//Get the values entered by the user.

firstName = firstNameTextField.getText().trim();

lastName = lastNameTextField.getText().trim();

email = emailTextField.getText().trim();

studentID = studentIDTextField.getText().trim();

birthDate = dateTextField.getText().trim();

//Perform the verifyInput method to test the input by the user

verifyInput();

// add database save here

if (inputVerification){

ResultSet resultSet = null;

//Insert into database

try

{

String DATABASE\_URL = "jdbc:mysql://209.129.8.4:3306/42029";

String USERNAME = "42029";

String PASSWORD = "42029csc18b";

Connection connection = DriverManager.getConnection(DATABASE\_URL, USERNAME, PASSWORD);

// SEE IF A USER ALREADY REGISTERED

//See if user is already registered by looking at the email address. If the email address is already

// in the database, then inform the user that there is already somebody registered with that

// same email.

searchEmailAddress = connection.prepareStatement("SELECT user\_email, user\_id FROM entity\_user\_Simon WHERE user\_email = ?");

searchEmailAddress.setString(1, email);

resultSet = searchEmailAddress.executeQuery();

emailArrayList = new ArrayList<String>();

while (resultSet.next()){

emailArrayList.add(resultSet.getString("user\_email"));

if (emailArrayList.get(0).equals(email)){

JOptionPane.showMessageDialog(null, "You are already registered. Thank you.",

"Registration Complete", JOptionPane.INFORMATION\_MESSAGE);

currentUserId = resultSet.getInt("user\_id");

result = 1;

userAlreadyRegistered = true;

}

}

// ADD A USER IF USER HASN'T REGISTERED

if (!userAlreadyRegistered){

// create PREPARED STATEMENT to insert that adds a new entry into the database

insertNewRegistrant = connection.prepareStatement(

"INSERT INTO entity\_user\_Simon" +

"(first\_name, last\_name, user\_email, student\_id, birth\_date, status)" +

"VALUES (?, ?, ?, ?, ?, ?)");

reformatTheDate();

insertNewRegistrant.setString(1, firstName);

insertNewRegistrant.setString(2, lastName);

insertNewRegistrant.setString(3, email);

insertNewRegistrant.setString(4, studentID);

insertNewRegistrant.setDate(5, sqlDate);

insertNewRegistrant.setInt(6, status);

//insert the new entry; returns the number of rows updated

result = insertNewRegistrant.executeUpdate();

}

// Get the user\_id of the user just added

// Use the email entered above to get the user\_id just created

if (!userAlreadyRegistered){

searchEmailAddress = connection.prepareStatement("SELECT user\_email, user\_id FROM entity\_user\_Simon WHERE user\_email = ?");

searchEmailAddress.setString(1, email);

resultSet = searchEmailAddress.executeQuery();

emailArrayList = new ArrayList<String>();

while (resultSet.next()){

emailArrayList.add(resultSet.getString("user\_email"));

if (emailArrayList.get(0).equals(email)){

currentUserId = resultSet.getInt("user\_id");

result = 1;

}

}

}

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

System.exit(1);

}

if (result == 1)

thankYouLabel.setText("User has been registered!!!");

else

thankYouLabel.setText("User has not been added");

firstNameTextField.setText("");

lastNameTextField.setText("");

emailTextField.setText("");

studentIDTextField.setText("");

dateTextField.setText("");

// Update the file with the current user

saveTheCurrentUser();

inputVerification = false;

userAlreadyRegistered = false;

somebodyRegistered = true;

}

}

}

UPDATE THE DATABASE

The following piece of code is from the ScoreFrame.java class where it updates the score information into the database.

private void updateTheDatabase(){

ResultSet resultSet = null;

//Insert into database

try

{

// Database information

String DATABASE\_URL = "jdbc:mysql://209.129.8.4:3306/42029";

String USERNAME = "42029";

String PASSWORD = "42029csc18b";

Connection connection = DriverManager.getConnection(DATABASE\_URL, USERNAME, PASSWORD);

// Assign values to the database fields

gameLevel = currentLevel;

gameScore = myScore;

gameUser = currentUserId;

// UPDATE the game history data base storing score level and user information

// This PrepareStatement also will return the last update to be used to

// INSERT into the Xref table below

insertNewHistoryRec = connection.prepareStatement(

"INSERT INTO entity\_history\_Simon" +

"(game\_date, game\_level, game\_score)" +

"VALUES (?, ?, ?)",PreparedStatement.RETURN\_GENERATED\_KEYS);

insertNewHistoryRec.setDate(1, getCurrentDate());

insertNewHistoryRec.setInt(2, gameLevel);

insertNewHistoryRec.setInt(3, gameScore);

// Insert the new entry; returns the number of rows updated

result = insertNewHistoryRec.executeUpdate();

// Get the last generated key for the last GameId value

ResultSet keyResultSet = insertNewHistoryRec.getGeneratedKeys();

int newGameId = 0;

if (keyResultSet.next()){

newGameId = (int) keyResultSet.getInt(1);

gameId = newGameId;

}

// UPDATE the Xref Table for game user and game number as per previous PrepareStatement

insertNewHistoryXref = connection.prepareStatement(

"INSERT INTO xref\_user\_history\_Simon" +

"(user\_id, game\_id)" +

"VALUES (?, ?)");

insertNewHistoryXref.setInt(1, gameUser);

insertNewHistoryXref.setInt(2, gameId);

//insert the new entry; returns the number of rows updated

result = insertNewHistoryXref.executeUpdate();

}

catch (SQLException sqlException)

{

sqlException.printStackTrace();

System.exit(1);

}

}

private static java.sql.Date getCurrentDate(){

System.out.println("You are reformatting the date");

java.util.Date today = new java.util.Date();

return new java.sql.Date(today.getTime());

}

Assignment 9 – POSIX threads and parallel programming. Implement multithreading into the game.

As it turns out, the game absolutely requires threading in one way or another. Originally, the game was coded without threading and the loop that would show the user the Simon pattern to repeat would execute fine, but the button would remain in the “pressed” position on the doClick() method. So after hours of researching, I finally found a method to thread the program that would allow the buttons to go on and off during the doClick(). That code is below:

From SimonGamePanel.java class. Enclosing the for loop with the new Thread(new Runnable() statements finally got the buttons to “press” and come on and off. However, there is still a glitch in here. Most of the time it is working. It currently fails to work whenever the loop ends in the exact same color as the beginning button. For example, if the first iteration is press RED, then the user would see the RED button go on and off. The user would then press the RED button with no problem. When the 2nd iteration starts, the RED button is bypassed and the button that follows, let us say, BLUE, goes off by itself. What I have to do is click any button first to get the counters within the program where they should be and then begin the iteration which is RED then BLUE. On the 3rd iteration, it will work perfectly… RED will go on and off, BLUE will go on and off and whatever randomly comes next will come on and off. It will fail again when the last button iteration is RED… so if on the fifth iteration it comes out to be RED, BLUE, YELLOW, YELLOW, RED, the fifth iteration will all click fine… but because the 5th iteration ended with the color RED, in the 6th iteration, the first button, RED, will not come on and it will start with BLUE. In that case again, when it comes time for the user to enter the pattern, I have to press any button and then start the iteration… RED, BLUE, YELLO, YELLOW, RED, etc… If the last button ends up being RED again, then the next iteration, the same thing happens… it has to be a threading issue because it doesn’t matter which color comes up as the first one, when the last button in the iteration as the first, it ends up skipping the first button on the subsequent iteration. Eventually I will figure out the proper way to thread this. But for now, this has got me 90% of the way there.

From SimonGamePanel.java class

if ((n == simonArraySize) || (!arraysGood) || (!onTime)){

System.out.println("Do not play the next iteration. Bring up the score panel");

} else {

System.out.println("The current thread is: " + Thread.currentThread().getName());

new Thread(new Runnable(){

public void run(){

for (int i = 0; i < j; i++){

playSimonArray(i);

}

}

}).start();

}

Assignment 10 – Networking

The only item that is has not been implemented into the game is the networking element.

APPENDIX

GITHUB

Individual Project: <https://github.com/lgutier111/Gutierrez_Leonardo_CSC18B>

Group Project: <https://github.com/lgutier111/SpaceInvaders_CSC18B>