**Projektarbeit Informatik Q11: Roulettespiel**

**mit SQL-Datenbankanbindung**

Teammitglieder: Zollitsch Christoph, Haberl Simon, Sauer Simon, Erben Emanuel

Schwerpunktarbeit:

* SQL-Datenbankanbindung: Simon S. & Simon H.
* Graphische Oberfläche & Controller: Christoph & Emanuel
* Hilfe aber auch Untereinander

**Wasserfallmodel:**

**1.Analyse:**

**Pflichtenheft:**

Ziel der Projektarbeit ist es ein Roulettespiel mit einer Anmelde- und Spieleoberfläche zu entwickeln.

Ferner sollen Benutzerdaten (u.a. Benutzername und Passwort) und das Verlaufsprotokoll des Spiels in einer Datenbank verwaltet werden.

Damit ein reibungsloser Ablauf gewährleistet ist, ist das Projekt auf Github für jeden zu jeder Zeit veränderbar.

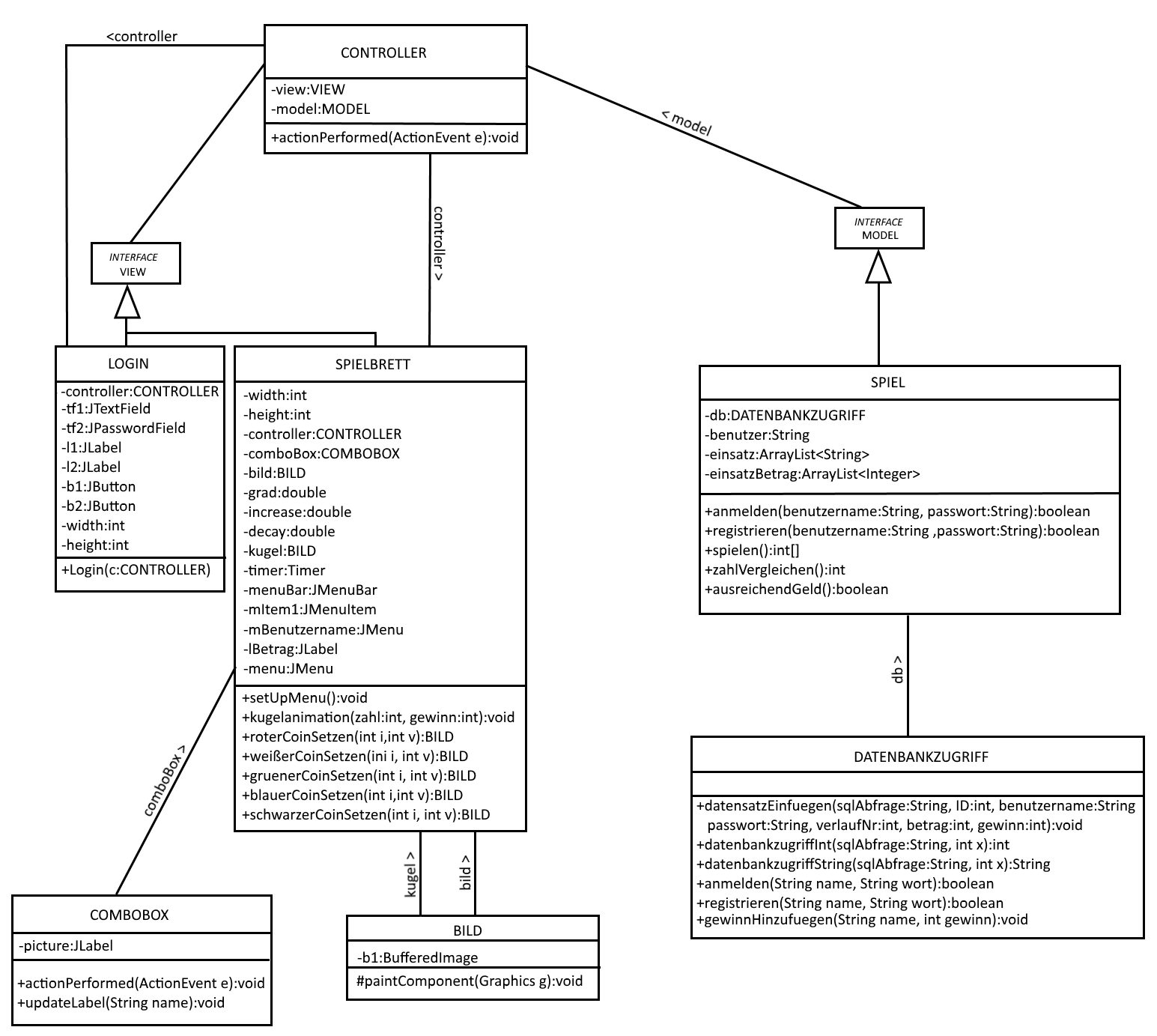
Geplanter Systemablauf:

Der Benutzer startet das Spiel auf der Anmeldeoberfläche. Auf dieser gibt er seinen Benutzernamen und das zugehörige Passwort in die jeweiligen Felder ein. Anschließend hat man die Möglichkeit sich anzumelden oder zu registrieren. Registriert sich ein neuer Benutzer, erfolgt ein automatischer Eintrag in eine SQL-Datenbank. Nach erfolgreicher Anmeldung bzw. Registrierung, wird der Benutzer auf die Spieloberfläche weitergeleitet. Auf dieser erscheint ein Auswahlfeld für den Betrag in Form von Jetons. Nachdem der Benutzer den Betrag ausgewählt hat, setzt er diesen auf das gewünschte Feld auf dem Roulettespielfeld. Zudem ist es möglich, mehrere Einsätze zu tätigen. Um eine neue Spielrunde zu starten, muss der der “Spielbutton“ gedrückt werden. Daraufhin beginnt die Kugel auf dem Rouletterad zu rotieren, um auf der zufällig ausgewählten Zahl stehen zu bleiben. Zum Schluss wird der Gewinn bzw. der Verlust graphisch ausgegeben und in der SQL-Datenbank geupdated und eine neue Spielrunde ist möglich.

Neben dem Rouletterad wird der Verlauf des Spielers angezeigt, der ebenfalls in einer SQL-Datenbank gespeichert ist und automatisch aktualisiert wird. Dies soll dem Benutzer einen Überblick über seine Historie verschaffen.

Am oberen Spielfeldrad befinden sich personenbezogene Daten: Betrag und Name des Spielers, eine Abmeldefunktion, sowie ein Menü mit Möglichkeit zur Hilfe und Einstellungen.

**2.Entwurf:**

Klassendiagramm:

**3.Implementierung (Tagebücher):**

**Graphische Oberfläche:**

**1.Anmeldeoberfläche:**

Als erstes haben wir die Klasse LOGIN, welche von der Klasse JFrame erbt, als Anmeldeoberfläche erstellt. Anschließend haben wir einfache Copyright freie Bilder für die Anmeldeoberfläche und das Spielpanel gesucht. Hierbei haben wir zwei gute Bilder gefunden, die zwar eine gute Qualität hatten, jedoch meist nur eine Auflösung von 960 auf 460 Pixel hatten. Wenn man jetzt die Größe verändert hätte, wären Weiße Ränder entstanden, die wir jedoch verhindern wollten. Deshalb haben wir in der Klasse Login die Zeile „setResizable(false)“ hinzugefügt, wodurch die Größe unseres JPanel´s vom Benutzer nicht mehr verändert werden kann. Danach haben wir auf der Anmeldeoberfläche eine Eingabefläche für Benutzername und Passwort hinzuzugefügt. Hierbei haben wir keinen LayoutManager gefunden, der unsere Anforderungen erfüllt hätte. Deshalb haben wir uns dafür entschieden, keinen LayoutManager zu verwenden, und alle Bestandteile unserer Oberfläche eigenhändig, an genauen Positionen am JFrame zu platzieren. Dies war auch kein Problem, da ja die Größe des Frames nicht verändert werden kann, und somit auch die Buttons und andere Elemente auf dem Frame nicht verrutschen können. Da wir Bilder in unserem Projekt benutzen wollten, dafür aber jedes Mal einen anderen Pfad benötigen, haben wir, um den Pfad der Bilder auf verschiedenen Computern auslesen zu können die Methode System.getProperty("user.dir") benutzt, da diese den Projektpfad zurückgibt. Für die Eingabe des Benutzernamens und des Passworts haben wir ein JTextField und ein JPasswordField, welches das Passwort verbiergt, dem Login-JFrame hinzugefügt. Dazu kamen zwei Buttons, mit denen man sich einloggen bzw, registrieren kann. Falls die Anmeldung oder Registrierung fehl schlägt wird ein MessageDialog mit entsprechendem Text angezeigt. Wenn die Daten übereinstimmen, bzw. die Registrierung erfolgreich ist, wird der Benutzer zur Spieloberfläche transferiert.

**2.Spieloberfläche:**

Das SPIELBRETT-JFrame ist ebenfalls nicht größenveränderbar und hat auch keinen LayoutManager. Es enthält eine Menüleiste, eine COMBOBOX um den Einsatz auszuwählen, einen "Spielen"- Button (um die Roulette-Scheibe zu drehen/die Zufallszahlauszugeben), und ein Feld mit den verschiedenen Einsatzmöglichkeiten, auf denen durch Linksklick auf einen der 49 unsichtbaren Buttons der Einsatz platziert werden kann. Der Einsatz wird in Form des ausgewählten Coins, am Platz des ausgewählten Feldes visualisiert. Für die Menüleiste haben wir JMenuBar verwendet und dort sowohl ein JMenu mit dem Text "Menü", welches die JMenuItems "Einstellungen" und "Hilfe" enthält, als auch ein JMenu mit dem Benutzernamen als Text, welches das JMenuItem "Abmelden" enthält, hinzugefügt. Die Klasse BILD erbt von JPanel und zeichnet ein dem Konstruktor übergebenes BufferedImage.

Die Klasse COMBOBOX erbt von JPanel und enthält eine JComboBox und ein JLabel. Abhängig von dem ausgewählten Betrag der JComboBox, wird das Icon des JLabels, als ein Bild des entsprechenden Coins gesetzt. Wir haben außerdem ein Rouletterad auf dem sich eine Kugel dreht hinzugefügt. Damit die Kugel sich in der gewünschten Bahn bewegt, haben wir einen 5ms Timer erstellt und immer nach Ablauf der Zeit die Gradzahl erhöht. Die X und Y Koordinaten der Kugel haben wir mithilfe cos und sin berechnet. Je nach Zahl lassen wir die Kugel an der entsprechenden Stelle stehnbleiben.

**Controller:**

Der CONTROLLER implementiert das Interface ActionListener. Bei allen Aktionen, die auf dem Frame durchgeführt werden, ist er dafür verantwortlich, diese Events in eine Reaktion zu verwandeln. Er ist dafür verantwortlich, dass nach dem Login der Konstruktor von SPIELBRETT aufgerufen wird. Er ruft bei Anmelde- bzw. Registrierfehler die Methode der Klasse LOGIN auf, die eine Fehlermeldung in Form eines MessageDialogs ausgibt. Des Weiteren reagiert er auf ActionEvents des Menüs und der Buttons auf dem SPIELBRETT.

**SQL-Datenbankanbindung:**

**Datenbankverbindung:**

Am Anfang wollten wir Access als Datenbank und den vorgegebenen Programmtext zur Datenbankverbindung nutzen. Allerdings stießen wir dabei auf einige Probleme: Der Access-Treiber konnte nicht geladen werden, weshalb wir versuchten im Internet Lösungen zu finden. Wir konnten aber keinen geeigneten Treiber laden bzw. finden. Deshalb erstellten wir eine neue Testklasse mit einer Methode, in der wir alle Treiber auflisteten, die uns zur Verfügung stehen. Daraufhin entschieden wir uns anhand des Treibers für eine neue Datenbank: SQLite, die wir mithilfe einer portablen Variante nutzten, damit wir sowohl in der Schule als auch zu Hause effektiv am Projekt arbeiten konnten. Den sqlite-odbc-Treiber haben wir dann unter BlueJ eingebunden und der Datenbankzugriff funktionierte nach anfänglichen Schwierigkeiten problemlos. In der Datenbank wollten wir zum einen die Benutzerdaten ~~(des Programms)~~ speichern, zum anderen aber auch die Verlaufsdaten in jeder Spielrunde. Wir brauchten also für die Datenbank hauptsächlich zwei größere Methoden: Die eine zum Einfügen eines neuen Datensatzes. Die andere zur Ausgabe bzw. Abgleichen der Datensätze, um beispielsweise die Anmeldedaten mit den auf der Anmeldeoberfläche eingegebenen Daten abzugleichen. ~~, um somit sich anzumelden~~. Diese Methoden bauen den Datenbankzugriff auf, führen die SQL Abfrage aus und schließen anschließend die Datenbankverbindung wieder. Da die Vergleichswerte sowohl vom Typ Integer als auch String sind, implementierten wir zwei verschiedene Methoden, die wir je nach SQL Abfrage aufrufen und unterschiedliche Rückgabewerte besitzen (String und Integer). Die meisten anderen Methoden der Klasse DATENBANKZUGRIFF greifen mit unterschiedlichen Eingangsparametern und SQL Abfragen auf diese Methoden zu. Dazu gehören Methoden „registrieren“, „anmelden", „betragGeben", "Idkreieren" (um für die einzelnen Spielerdatensätze Primärschlüssel zu generieren) oder auch "datensatzVergleichen". Desweiteren implementierten wir eine weitere Methode, die den Datenbankzugriff aufbaut, eine SQL Abfrage ausführt und anschließend die Datenbankverbindung wieder schließt, wobei diese Methode keinen Rückgabewert (void) besitzt, weshalb diese etwas anders aufgebaut ist. Methoden wie „gewinnHinzufuegen“, also Methoden, die eine Updatefunktion besitzen, greifen darauf zu.

**Spiel:**

In der Klasse SPIEL implementierten wir zunächst die Methode „zahlVergleichen“ (siehe Klassendiagramm), welche durch bedingte Anweisungen prüft, ob der Benutzer gewonnen hat und berechnet dementsprechend den Gewinn, der anschließend ausgegeben wird. Dabei greift sie auch auf Hilfsmethoden wie beispielsweise „istGerade“ oder „isterstesDrittel“ zu. Außerdem ist eine Methode „ausreichendGeld“, welche \_\_\_, vorhanden. Natürlich ist auch die Methode „spielen“, die zunächst eine Zufallszahl generiert, die Methode „zahlVergleichen“ aufruft und schließlich den Gewinn bzw. Verlust hinzufügt indem sie die Dazugehörige Methode in der Klasse DATENBANKZUGRIFF aufruft, implementiert. Um zu speichern, welcher Benutzer angemeldet ist und die richtigen Parameter an die Klasse DATENBANKZUGRIFF weitergeben zu können, führten wir globale Variablen ein, welche durch dazugehörige Methoden automatisch aktualisiert werden.

private ArrayList<String> einsatz = new ArrayList<String>();

private ArrayList<Integer> einsatzBetrag = new ArrayList<Integer>(); **(?)**

**Model:**

---Ich hab jetzt dein Text mal durchgelesen und noch verändert und halt vor allem weitergeschrieben. Hab die Sachen, die ich gemacht habe rot markiert, dass du das siehst. Bei paar Sachen wusste ich nicht genau was ich schreiben sollte oder von wem die Methode aufgerufen wird etc.; vllt kannst du die Stellen noch fertig schreiben und nochmal drüberlesen, ob man das alles so schreiben kann. Und morgen können wir ja nochmal skypen und dann wirklich alles perfekt machen.---

Und ich hab allgemein noch bisschen formatiert, wenn das so ok ist