Transfer-Dokumentation-Report (TDR)

**Algorithmen und Datenstrukturen**



**Eingereicht bei**

Petra Hieber

**Verfasst von**

Michaelsen, Robin - 638-0-04073

Härtel, Otto - 638-0-04059

638-0-04059

BiB8aE

Eigenständigkeitserklärung

Ich habe die vorliegende Arbeit im Rahmen des Projekt-Kompetenz-Studiums „2018/2021“ selbstständig verfasst und keine anderen als die angegebenen Quellen, Tools und Hilfsmittel benutzt.

Alle Aussagen in der Arbeit, die wörtlich oder sinngemäß aus gedruckten oder elektronischen Veröffentlichungen oder aus anderen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit oder Teile daraus sind noch nicht Leistungsnachweis einer   
vorangegangenen Prüfung gewesen bzw. sind entsprechend als Quelle gekennzeichnet.

Erlangen, den 29.04.2021

Michaelsen, Robin

Erlangen, den 29.04.2021

Härtel, Otto

Erlangen, den 29.04.2021

Rösch, Rene

Inhaltsverzeichnis

[Eigenständigkeitserklärung I](#_Toc69999338)

[Inhaltsverzeichnis II](#_Toc69999339)

[Abkürzungsverzeichnis III](#_Toc69999340)

[Abbildungsverzeichnis IV](#_Toc69999341)

[Tabellenverzeichnis V](#_Toc69999342)

[1 Allgemeine Programmstruktur 1](#_Toc69999343)

[2 Grafische Benutzeroberfläche (in MVP nur der View) 4](#_Toc69999344)

[2.1 Dialoge 5](#_Toc69999345)

[2.2 Fenster 7](#_Toc69999346)

[2.3 Menüs 8](#_Toc69999347)

[2.4 Spezielle Designelemente in der vorliegenden Applikation 9](#_Toc69999348)

[Quellenverzeichnis 11](#_Toc69999349)

[Anhang 11](#_Toc69999350)

Abkürzungsverzeichnis

Abb. Abbildung

Abbildungsverzeichnis

[Abbildung 1: Führungsstile 1](#_Toc63710146)

Tabellenverzeichnis

[Tabelle 1: Personalentwicklungsziele 2](#_Toc63710152)

**//IMMER SICH FRAGEN WURDE DAS WAS? WIE? WARUM?  Designschritte**

**//Überleitungen und Zusammenfassungen & Unternehmerischer Nutzen / Transfer (S.17 / 22 im Balzert)**

# Einleitung

...

# Umgebung

* Java 11.0.9
* Apache Netbeans 12.0
* Apache Derby 10.14.2.0
* Hibernate 5.4.25

# Allgemeine Programmstruktur

Als allgemeine Programmstruktur wurde ein objektorientiertes Verhaltensmuster in Form des MVC- Musters (model view controller pattern) gewählt, bei welchem es sich um eine Verfeinerung des Beobachter (Observer)- Musters handelt.[[1]](#footnote-1)

Beim Beobachter- Muster (observer pattern) handelt es sich um ein objektorientiertes Verhaltensmuster (behavioral pattern). Ziel dieses Musters ist es, dass von einem Objekt abhängige Objekte automatisch benachrichtigt werden, wenn sich das eine Objekt ändert, um ggf. ihre Daten zu aktualisieren. Dies wird durch eine Registrierung der abhängigen Objekte bei dem gewünschten Objekt erreicht. Dieses Muster wird häufig in Verbindung mit GUI- Frameworks und Benutzeroberflächen eingesetzt. Beobachter (Observer) (Objekte die informiert werden wollen) registrieren sich dafür bei den gewünschten Objekten. In Java existiert eine Klasse (Observable, zuständig für Re-/Deregistrierung von Observern, sowie deren Benachrichtigung) und eine Schnittstelle (Observer, für Beanschriftung bei Änderungen an den jeweiligen Observable) um das Observer- Muster umzusetzen.[[2]](#footnote-2)

Eine Anwendung wird durch das MVC- Muster grundlegend in drei Komponenten bzw. Subsysteme aufgeteilt:1

* Model
* View
* Controller

Die Anwendungs- bzw. Geschäftslogik sowie deren Daten wird durch das Model repräsentiert. Die grafische Benutzeroberfläche wird durch die View dargestellt. Die Hauptfunktion des Controllers ist das Steuern der Applikation (u.a. Benutzerinterkation und Veränderung des Models). So fungiert der Controller als Bindeglied und Bekanntmacher zwischen Model und View. Dabei kann ein Model ggf. von mehreren Controllern verändert werden. Zwischen Model und View wird das Observer- Pattern implementiert, so dass sich die jeweiligen Views als Beobachter bei den gewünschten Models registrieren und so bei einer Zustandsänderung des Models benachrichtigt werden, wodurch sie wiederum ihre Darstellung anpassen können.1

Falls eine Applikation komplex aufgebaut ist, kann diese auch aus mehrere kleinere fachliche Sub- Komponenten zusammengesetzt werden. Diese Sub- Komponenten (oftmals eigene View- Controller- Beziehungen) können eigene Verantwortlichkeiten bzgl. der Darstellung oder der Interaktion besitzen. Eine weitere Variante des MVC- Pattern stellt das Model- View- Presenter (MVP) Pattern dar.[[3]](#footnote-3)

Beim MVP- Pattern herrscht eine strenge Trennung zwischen Model und View. Das Model enthält wie beim MVC- Pattern die Geschäftslogik sowie die Daten und kennt weder den View noch den Presenter. Außerdem übernimmt der Presenter die alleinige Steuerung des Models. Der View enthält keinerlei Steuerungslogik. Er akzeptiert nur Benutzereingabe und stellt die gewünschten Daten entsprechend dar. Die Verbindung des Models und des Views erfolgt über den Presenter. Dieser enthält die steuernde Logik, nimmt Benutzereingabe der View entgegen, ändert Daten am Model und leitet diese Änderungen wieder für die Darstellung an den View weiter. Dadurch übernimmt der Presenter die Rolle des Mediators zwischen Model und View. Die View besitzt jedoch keinen Zugriff auf den Presenter oder das Model. Der größte Unterschied zum MVC- Muster besteht darin, dass es beim MVP- Muster keine Assoziation zwischen Model und View gibt.[[4]](#footnote-4)

Beim MVP – Muster kann der View unterschiedliche Ausprägungen besitzen. Einerseits kann der View komplett passiv sein, in dem er bei Benutzerinterkation einen Methodenaufruf beim Presenter veranlasst, der wiederum über Schnittstellen am View und Model Daten aktualisiert und zur Darstellung aufbereitet. Anderseits kann der View auch so implementiert sein, dass er die Möglichkeit besitzt Interaktionen oder Ereignisse selbstständig zu verarbeiten.[[5]](#footnote-5)

In der vorliegenden Applikation wurde eine Kombination aus MVC und MVP- Muster angewendet. Es existierte eine strikte Trennung zwischen Model und View ähnlich wie im MVP- Muster vorhanden ist, jedoch werden die Benutzereingabe bzw. Interaktion wie im MVC- Muster direkt von der, im vorliegenden Falle, als Presenter bezeichneten Instanz wahrgenommen, ohne einen expliziten Aufruf in der jeweiligen View. Zwischen Views und Presenter existiert in der vorliegenden Applikation eine 1:1 Beziehung. Zudem bleibt der Presenter dem dazugehörigem View verborgen. Der Presenter übernimmt als Action- Listener die gesamte Steuerung der Applikation, in dem er u.a. auf Benutzereingaben reagiert, Benutzeroberflächenelemente hinzufügen, aktivieren oder auch deaktivieren kann oder das jeweilige Model ändern kann. Ähnlich wie in dem Buch von Balzert H (S.67f.) beschrieben kann ein Presenter je nach gewünschter Aktion auch die Erzeugung weiterer Presenter bzw. Views und deren Verknüpfung oder das Wechseln des Models übernehmen. Die Datenkonsitenz der Applikation wird durch die Implementierung des Observer- Musters vom MainPresenter, der jeweils als Observer bei allen Ligen hinzugefügt wird, (ähnlich wie im MVC- Pattern (siehe Balzert S.65)) gewährleistet. Jedoch werden einzelne Views nicht direkt mit dem Liga- Model als Observer verknüpft, sondern nur der MainPresenter dient als Observer für das Liga- Modell. Das Liga- Modell wird bei weiteren erzeugten View- Presneter Beziehungen jeweils als Referenz übergeben. Bei Änderungen des Liga- Models wird dadurch nur der MainPresenter benachrichtigt, der die Daten des aktuell anzuzeigenden Modells an einer Stelle aktualisiert und veranlasst den anzuzeigenden View neu zu zeichnen.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Dies geschieht über die Implementierung der ActionListener- Schnittstelle in den jeweiligen Presentern, welche jeweils eine ActionPerformed Methode überschreiben. Zwischen Views und Presenter existiert in der vorliegenden Applikation eine 1:1 Beziehung. Zudem bleibt der jeweiligen View dem dazugehörigen Presenter verborgen. Da der Presenter die jeweiligen Views kennt, fügt sich dieser selbst als Action- Listener für z.B. jeweilige Buttons oder auch Click- Ereignisse in Tabellen oder Listen hinzu. In der jeweiligen View wird jeder Button mit einem passenden ActionCommand versehen, welche anschließend im Presenter in der ActionPerformed Methode verarbeitet wird, um die jeweils korrekte Aktion auszuführen. Der Presenter übernimmt als Action- Listener die gesamte Steuerung der Applikation, in dem er u.a. Benutzeroberflächenelemente hinzufügen, aktivieren oder auch deaktivieren kann. Ähnlich wie in dem Buch von Balzert H (S.67f.) beschrieben kann ein Presenter je nach gewünschter Aktion auch die Erzeugung weiterer Presenter bzw. Views und deren Verknüpfung oder das Wechseln des Models übernehmen. Ein Beispiel für das Wechseln des Models in der vorliegenden Applikation ist u.a. das Umschalten der Liga (z.B. Liga 1 zu Liga 2) über Knöpfe im MainView, welches wiederum den MainPresenter veranlasst das anzuzeigende Datenmodell (Liga) zu ändern, welches anschließend durch den jeweils ausgewählten View in der gewünschten Form (z.B. Tabelle, Spielplan, Clubs) dargestellt wird. Ein Wechsel von Presenter- View Beziehungen liegt u.a. im MainPresenter vor, wenn von der Tabellen- View in den Spielplan- View gewechselt wird. Hierbei wird der View angepasst und der dazu passende Presenter erzeugt. In der vorliegenden Applikation wird mit einem einzigen Datenmodell (Liga) als Grundlage gearbeitet. In diesem Datenmodell sind Clubs als auch Spiele enthalten. Dieses Datenmodell wird beginnend im MainPresenter an die jeweiligen (Sub)- Presenter- View- Beziehungen als Referenz weitergereicht, die ggf. vom MainPresenter ausgewählt bzw. angesteuert werden (z.B. ClubPresenter und ClubView). Die Datenkonsitenz der Applikation wird durch die Implementierung des Observer- Patterns vom MainPresenter, der jeweils als Observer bei allen Ligen hinzugefügt wird, (ähnlich wie im MVC- Pattern (siehe Balzert S.65)) gewährleistet. Jedoch werden einzelne Views nicht direkt mit dem Liga- Model als Observer verknüpft, sondern nur der MainPresenter dient als Observer für das Liga- Modell. Bei Änderungen des Liga- Models wird dadurch nur der MainPresenter benachrichtigt, der die Daten des aktuell anzuzeigenden Modells an einer Stelle aktualisiert und veranlasst den anzuzeigenden View neu zu zeichnen. Dieses System funktioniert über verschiedene View- Presenter- Beziehungen hinweg. Dies ist möglich, da jeweils nur das anzuzeigende Liga- Model aktualisiert wird und kaskadierend an die bestehende View- Presenter- Auswahl weiter gereicht wird.

Es wurde bewusst diese Art der Datenkonsistenz zwischen Model und View gewählt, da die Applikation einen komplexen Aufbau aus verschiedensten Presenter – View – Beziehungen aufweist, die wiederum weitere Presenter- View – Beziehungen erzeugen können. So ist die Datenkonsistenz über alle Views zu jedem Zeitpunkt gegeben.

# Grafische Benutzeroberfläche (in MVP nur der View)

//Analyse und Designschritte der Benutzeroberfläche (ca. 1 Seite)

//S. 192 / S. 369 (Primärer und Sekundärer Dialog, modaler, nicht modaler Dialog) / objekt orientierte/ funktionsorientierte Bedienung Lehrbuch der Objektmodellierung, Fenster gestaltung  Dialoganapssung)  bei uns nur: Fenster und Dialoge, jedoch zu keinem Lebenszeitpunkt der Applikation mehr als 2 Fenster gleichzeitg offen, von denen eines ein Dialogfenster ist und das andere ein Hauptfesnter ist

 bei uns Objektorientierte Bedingung: wählen über die jeweiligen Liga- Knöpfe das Objekt aus und entscheiden dann was damit gemacht werden soll.

**//wichtig: Analyse- und Designschritte beschreiben (Dialog Gestaltung)**

**//warum wo welche Elemente in der Benutzeroberfläche? (beispielhaft)**

**//**Analyse und Designschritte der Benutzeroberfläche:

Plan für GUI- Erklärung mittels Lehrbuch Objektmodellierung Analyse und Entwurf LE9 Kapitel 5 Teil 1:

1. Theoretische Erklärung des **Dialoges** (modal/ nicht modal, primärer/ sekundärer Dialog) und anschließend praktische Beschreibung/ Anwendung auf vorliegendes Projekt (anhand eines Beispiels)
2. Theoretische Erklärung des **Fensters** (Primärfenster (inkl. Unterfenster bei uns keine wirklichen Unterfenster sondern nur Anpassung des Arbeitsbereich), Sekundärfenster(auch Mitteilungsfenster)) und anschließend praktische Beschreibung / Anwendung auf vorliegendes Projekt (anhand eines Beispiels)
3. Theoretische Erklärung **Menüs** (Menübalken / pop- up Menüs) und anschließend praktische Beschreibung / Anwendung auf vorliegendes Projekt (anhand eines Beispiels)
4. Spezielle Design Merkmale (Bsp.: welches Pannel wurde verwendet für das automatische hinzufügen von Buttons, bzw. welches Layout und warum  Box Layout damit Buttons jeweils übereinander angeordnet werden)

Zusätzliche Informationen aus Kapitel 5 Teil2:

1. Gestaltungsregeln (siehe LE 10, welche davon beachtet z.B. Symmetrie..)

Im folgenden wird genauer auf die Gestaltungsweise der grafischen Benutzeroberfläche mit Hilfe der theoretischen Grundlagen von Frau Prof. Balzert eingegangen.

Die grafische Benutzeroberfläche (GUI, Graphical User Interface) entspricht der View- Komponente im MVC bzw. MVP- Pattern. Sie erzeugt Fenster, Menüs sowie die Dialogführung. Zu einer GUI gehören jeweils eine E/A- Komponente (Gestaltung der Information) sowie eine Dialogkomponente (Bedienungsabläufe). Diese grafische Benutzeroberfläche wird mittels eines GUI- Systems (in dem vorliegenden Anwendungsfall Windows) verwaltet. [[6]](#footnote-6)

* 1. Dialoge

Für die Interaktion zwischen einem Dialogsystem und einem Benutzer sowie die Erreichung eines bestimmten Zieles damit wird ein Dialog verwendet. Dabei existieren zwei Arten von Dialogen: Primär- und Sekundärdialog. Primärdialoge dienen der direkten Aufgabenerfüllung und werden bei Fertigstellung der zu bearbeitenden Aufgabe beendet. Sekundärdialoge dienen zum darstellen und anzeigen von Zusatzinformationen. Dialoge können außerdem in unterschiedliche Modi dargestellt werden. So gibt es modale Dialoge, die beendet werden müssen, bevor ein anderes Fenster der Applikation geöffnet werden kann. Außerdem können nicht – modale Dialoge existieren, welche es dem Benutzer ermöglichen neue Dialoge zu erzeugen, auch wenn bereits weitere Dialogfenster geöffnet sind. Die Verwendung von möglichst vielen nicht- modalen Dialogen erhöht die Handlungsflexibilität des Benutzers. Jedoch kann dies im Fehlerfall unter Umstände nicht gewollt sein, wodurch hierbei auf modale Dialoge zurückgegriffen wird, um eine Weiterarbeit erst nach Beendigung des Fehlers zu ermöglichen.[[7]](#footnote-7)

In der vorliegenden Applikation stellt das Hauptfenster (Klasse MainView) den Primärdialog dar von welchem aus weitere Dialoge angesteuert bzw. erzeugt werden können. Die Applikation besitzt wenig Sekundärdialoge (z.B. Kader-View- bzw. Dialog oder auch den PlanAddGame- View bzw. Dialog). Die Sekundärdialoge, die existieren sind zudem alle nicht- modal. Dies wurde gewählt, damit die Datenkonsitenz zu jedem Zeitpunkt der Applikation gegeben ist. So ist es z.B. nicht möglich mehrere Club-Dialoge mit jeweils einem Transaktionsdialog (im ClubEdit-View bzw. Dialog) gleichzeitig zu erzeugen, was ggf. zu Dateninkonsitenzen bei Transaktionen führen kann die noch nicht oder nur Teilweise abgeschlossen sind, und deren Ergebnisse wiederum in anderen offenen ClubEdit-View Dialogen verwendet werden.

Für die Bedienung der Dialoge existieren verschiedene Bedienungsarten:[[8]](#footnote-8)

* Objektorientierte Bedienung (zuerst Auswahl des zu bearbeitenden Objektes, anschließend Auswahl der anzuwendenden Funktion auf das Objekt)
* Funktionsorientierte Bedienung (zuerst Auswahl der gewünschten Funktion, anschließend Auswahl des für diese Funktion zu nutzendem Objekt)

Bei der objektorientierten Bedienung gibt es außerdem die Möglichkeiten Objekte direkt (z.B. Programmstart mittels doppel Click in Windows) oder auch über Fenster bzw. Menüs (z.B. Programmstart über Startmenü in Windows) zu manipulieren. Oftmals werden in der Praxis Kombinationen aus diesen Bedienungsarten verwendet.7

In der vorliegenden Applikation wird beispielsweise eine objektorientierte Bedienung im MainView- bzw. MainPresenter bei der Auswahl der Liga bzw. des Liga- Models (über die vorhandenen Liga 1 bis Liga 3 Buttons) gewählt. Dies wurde so umgesetzt, um die größtmögliche Flexibilität zwischen anzuzeigenden Darstellungen sowie den anzuzeigenden Daten zu realisieren. So ist außerdem eine benutzerfreundliche Bedienung gewährleistet.

Eine funktionsorientierte Beziehung ist bei dem Dialog zum Hinzufügen von Spielen vorzufinden. Hierbei wird zuerst die auszuführende Funktion ausgewählt (Spiel hinzufügen), welches wiederum das öffnen einen neuen Sekundär-Dialoges veranlasst. In diesem Sekundär- Dialog werden anschließend die Objekte, in diesem Fall die zwei Clubs, die gegeneinander spielen, ausgewählt. Bei der Bestätigung dieses Dialogfensters, wird anschließend eine Funktion ausgeführt, die ein Spiel zwischen den ausgewählten Clubs zu dem gewünschten Datum erstellt und zu den jeweiligen Ligen hinzufügt.

* 1. Fenster

Das Fenster ist das zentrale Element der Dialoggestaltung. Unter Windows werden folgende Fenstertypen differenziert:[[9]](#footnote-9)

* Sekundärfenster (Durchführung Sekundäraktivitäten (Sekundärdialoge), Optionseingabe)
* Primärfenster (Durchführung Hauptaktivitäten (Primärdialoge))

Das Anwendungsfenster, aus welchem sich heraus bei Bedarf weitere Fenster öffnen lassen können, ist das wichtigste Primärfenster. Der Arbeitsbereich, der Menübalken sowie der Titelbalken sind mindestens im Primärfenster enthalten. Bei einer Schließung des Primärfensters erfolgt außerdem das Beenden aller offenen Sekundärfenster.[[10]](#footnote-10)

In der vorliegenden Applikation besteht das Primärfenster der Applikation (MainView) aus einem Titelbalken (Titel: Bundesligaverwaltung), einem zwei reihigem Menübalken, bestehend aus den Auswahlknöpfen für Liga 1, Liga 2, Liga 3 sowie den Auswahlknöpfen für Tabelle, Spielplan und Clubs und dem Arbeitsbereich, der je nach der getroffenen Auswahl die Darstellung bzw. die anzuzeigenden Daten im Arbeitsbereich anpasst. So wird z.B. bei einer Auswahl der Knöpfe Liga 1 und Tabelle die Tabelle der ersten Bundesliga im JPanel des MainView dargestellt.

Dialogfenster (dialog box) und Mitteilungsfenster (message box) sind u.a. verwendbare Arten von Sekundärfenster in Windows. Für die Darstellung von Sekundärdialogen werden Dialogfenster benötigt. Daher sind diese Sekundärfenster meist mittels modaler Dialoge realisiert (können jedoch auch nicht modal sein). Sekundärdialoge realisieren häufig Dateneingaben über Interaktionselemente. Eine mögliche Spezialisierung von Dialogfenstern sind u.a. Mitteilungsfenster. Dem Benutzer stehen bei einem Mitteilungsfenster keine Interaktionselemente zur Datenselektion- oder Manipulation zur Verfügung. Mittels einer Aktion kann der Benutzer auf Mitteilungen reagieren.[[11]](#footnote-11)

//Bsp.: wie bei uns  Dialog: neues Spiel hinzufügen, Mitteilung die Betsätigung des hinzugefügten Speiles oder die Fehlermeldung

In der vorliegenden Applikation wurden Sekundärfenster als Dialoge z.B. beim hinzufügen eines neuen Spiels zwischen zwei Teams verwendet. Dabei kann der Benutzer die jeweilige Liga mittels einer JComboBox auswählen, in denen sich z.B. das erste oder das zweite Team befindet und anschließend auch über eine JComboBox die jeweiligen Teams aus den ausgewählten Ligen, die gegeneinander spielen sollen. Es wurde diese Team Auswahl über JComboBoxen gewählt, um eine möglichst einfache Bedienung zu gewährleisten und mögliche Fehlerquellen, die durch eine manuelle Benutzereingabe entstehen könnten im Voraus auszuschließen. Die vorgegebenen Auswahlmöglichkeiten für die Ligen wird dabei an die jeweilige Liga angepasst, in der das Spiel hinzugefügt werden soll. So ist es z.B. möglich in Liga 1 Spiele zwischen Teams aus Liga 1 und Liga 2 zu organisieren während es beim hinzufügen von Spielen in Liga 2 möglich ist Spiele zwischen Teams aus Liga 1, Liga 2 oder Liga 3 zu speichern. Nach Auswahl der jeweiligen gegnerischen Teams wird anschließend das Datum und die Uhrzeit vom Benutzer manuell eingegeben. Das korrekte Datumsformat steht dabei unter dem Eingabefeld angeben, um dem Benutzer die korrekte Datumseingabe zu erleichtern.

Nach vollständiger Auswahl und Eingabe aller erforderlichen Parameter werden diese überprüft. Kann das spiel erfolgreich hinzugefügt werden, erscheint ein MessageDialog (mittels JOptionPane) mit dem Hinweis, dass das Spiel erfolgreich gespeichert wurde. Falls es zu Problemen beim Anlegen des Spiels kommt, werden auch MessageDialoge erzeugt, die den Benutzer auf mögliche Fehler (z.B. Auswahl gleicher Teams als gegnerische Teams) bei der Parametereingabe hinweisen.

* 1. Menüs

Eine übersichtliche und zumeist vorbestimmte Auswahl von Menüoptionen sind häufig Bestandteile von Menüs. Dabei besteht für den Benutzer die Möglichkeit eine oder mehrere Optionen auszuwählen. Menüs können als Aktionsmenü (Verzweigung in andere Menüs oder Auslösen von Anwendungsfunktionen) oder als Eigenschaftsmenü (Einstellung von Parametern zum Beeinflussung des Verhaltens der Anwendung, auch mehr Selektion möglich) in Applikationen vorkommen.[[12]](#footnote-12)

Es gibt u.a. folgende Menüarten:11

* Menübalken (besitzt alle Menütitel, ist immer sichtbar, Anwendungsfenster hat immer einen Menübalken, Menübalken sind nicht in Dialog- oder Mitteilungsfenster vorhanden, Menübalken in Unterfenster möglich, falls Unterfenster keine eigenen Menübalken, gilt der Menübalken des Anwendungsfensters)
* Pop-up Menüs (wird an aktueller Mauszeigerposition erzeugt, Bezugspunkt ist Objekt, auf das es aktiviert wird, nur sichtbar, wenn geöffnet, Menüoptionen gelten nur für selektiertes Objekt)

In der vorliegenden Applikation wurde auf klassische Menüleiste, wie in den meisten Windows- Applikationen vorzufinden ist verzichtet. Stattdessen wurden beispielsweise im Hauptfenster (MainView) zwei Menüreihen zur Auswahl der anzuzeigenden Daten (über Liga 1 bis Liga 3 selektierbar) und der passenden Ansicht (Tabelle, Spielplan, Clubs) zu den Daten genutzt. Dies ermöglicht sowohl eine große Flexibilität in der Auswahl als auch eine einfache Bedienung, da so auf Verzweigungen in andere Menüs verzichtet werden konnte und gleichzeitig der Benutzer zu jedem Zeitpunkt im Hauptfenster durch farbliches Hervorheben der jeweiligen Knöpfe erkennt, welche Kombination aus Liga bzw. Ansicht ausgewählt ist. Eine weitere Anwendung des Menübalken wurde in der Ansicht zum bearbeiten von jeweils einzelnen Clubs (ClubEditView) gewählt, bei welchem jeweils du Kader, Transaktion oder Spieler hinzufügen einzelne Information über den Club abgerufen (z.B. durch Kader) oder geändert (z.B. durch Transaktion oder Spieler hinzufügen) werden können.

* 1. Spezielle Designelemente in der vorliegenden Applikation

//inkl. Gestaltungsmerkmale

//Bsp.: wann listen und wann z.B. über Suchfeld (Bsp.: Transfer)

In jedem GUI- System stehen verschiedene Elemente zur Interaktion mit bzw. Steuerung der Applikation bereit. Dabei können diese Elemente bzgl. ihres Aussehens und ggf. bzgl. ihrer Funktionsweise vom jeweiligen Betriebssystem abhängig sein, auf welchem die Applikation ausgeführt wird.[[13]](#footnote-13)

Im Folgenden wird auf eine Auswahl eingesetzter grafischer Interaktionselemente genauer eingegangen, die in der vorliegenden Applikation (laufend unter dem Betriebssystem Microsoft Windows) verwendet wurden.

**Tabellen bzw. Listenelemente** stellen Erweiterungen eines Listenfeldes (list box) dar. Für die Darstellung der Daten in der Tabellenform existieren verschiedene Darstellungsvarianten:[[14]](#footnote-14)

* beliebige Positionierung von (Mini)-Piktogramm mit Beschriftung im Listenelemente
* jeder Eintrag in Liste bestehend aus Beschriftung und Minipiktogramm spaltenweise sortiert
* Report (pro Zeile ein Eintrag, jeder Eintrag bestehend aus mehreren Spalten, Spaltenbreite anpassbar)

Da die vorliegende Applikation mit Java programmiert wurde, wurde für die Darstellung von Tabellen (z.B. im TableView oder in der Spiel- Historie) die Java eigene JTable gewählt. Diese stellt die Daten in der Form des oben beschriebenen Reportes dar. Damit die Daten korrekt dargestellt werden, benötigt die JTable ein eigenes Model, welches die Tabellenstruktur (Spaltenbeschriftung und ggf. Zeilen) vorgibt. Die Tabellenmodelle für die verwendeten JTable in der vorliegenden Applikation wurden in den jeweiligen View- Klassen mit den gewünschten Spalten initialisiert. Die Befüllung der jeweiligen Tabelle mit den anzuzeigenden Daten erfolgt durch den entsprechenden Controller, der für den jeweiligen View verantwortlich ist, in welchem sich die Tabelle befindet. In der vorliegenden Applikation sind alle vorhanden Tabellen spaltenweise sortierbar, jedoch ist keine Tabelle an sich editierbar. Die Möglichkeit der Veränderung von Daten direkt in der Tabelle wurde deaktiviert, um die Daten zwischen dem Model und der jeweiligen View, die die Tabelle anzeigt konsistent zu halten. In einige Tabellen z.B. im ClubView oder im KaderView sind Zeilen bzw. Spalten selektierbar und über Pop-Up Menüs editierbar. Diese Möglichkeit, wurde bewusst nur in diesen Views bzw. Tabellen gewählt, da die JTables zum Anzeigen der allgemeinen Bundesligatabelle bzw. der Spielhistorie nicht verändert werden dürfen, da sie nur bereits feststehende bzw. verarbeitete Informationen anzeigen.

Eine platzsparende Variante der Liste ist das **Dropdown- Listenfeld** bzw. die Klappliste, bei welcher ein Aufklappen der Liste vor der Selektion eines Listenelementes notwendig ist. Nach der Selektion ist die Liste wieder unsichtbar und das selektierte Element wird permanent angezeigt. Dabei ist eine Überdeckung anderer Interaktionselemente durch die aufgeklappte Liste möglich. Für die Gestaltung eines Dropdown- Listenfeldes gelten die Regeln eines Listenfeldes, zusätzlich kann aufgrund des ähnliches Aufbaus ein Textfeld mit dem Dropdown- Listenfeld kombiniert werden und es kann eine Vorauswahl in dem Drop- Down Listenfeld getroffen werden.14

In der vorliegenden Applikation wurden Dropdown- Listenfelder in Form von JComboBoxen beispielsweise beim Ersteller neuer Spiele im PlanAddGameView verwendet. Hierbei ist sowohl die auswählbare Liga als auch das jeweils auswählbare Team über JComboBoxen realisiert. Die jeweils selektierbaren Teams richten sich dabei nach der ausgewählten Liga. Diese Interaktionselemente wurde an dieser Stelle bewusst gewählt, um den Benutzer, durch vorgegebene Auswahlmöglichkeiten in den JComboBoxen eine einfache Bedienung der Applikation zu ermöglich und zu gleich die Wahrscheinlichkeit für fehlerhafte Benutzereingaben, die z.B. durch einen direkten Input durch Textfelder entstehen könnten, zu reduzieren.

Für die vertikale Darstellung von mehreren alphanummerischen oder grafischen Einträgen können **Listenfelder** dienen. Als Auswahlmodi stehen dabei single selection (nur ein Listenelement selektierbar),multi selection (mehrere Listenelemente selektierbar) oder extended selection (mehrere zusammenhängende Bereiche) zur Verfügung. Auf horizontale Rollbalken sollte in einer Liste verzichtet werden, außerdem sollten aus Gründen der Lesefreundlichkeit mindestens vier Listenelemente gleichzeitig sichtbar sein. Listen werden oft durch die Anwendung gefüllt, wodurch Listenfelder geeignet sind, um Elementkollektionen darzustellen, deren Anzahl von Elementen variable und umfangreich sind.[[15]](#footnote-15)

In der vorliegenden Applikation wurde ein Listenfeld z.B. bei der Ergebniseingabe für Spiele im ErgebnisInputView verwendet. Hierbei kommt die Java eigene JList für die Darstellung der jeweiligen Spieler der jeweiligen Clubs zur Anwendung, so dass die geschossenen Tore spielergranular erfasst werden können. Hierbei wurde das Listenfeld als Interaktionselement für den Benutzer gewählt, da die Spieleranzahl für jeden Club ggf. unterschiedlich sein kann und dem Benutzer die Spielerauswahl, durch die vorgegebene Spielerliste erleichtert werden sollte.

**Textfelder** dienen zur Ein- bzw. Ausgabe von Text sowie numerischen Daten in einer Zeile. Dabei sollten alle Zeichen in dem Textfeld darstellbar sein sowie. Die Dimensionen des Textfeldes sind so zu wählen, dass die Mehrheit voraussichtlicher Eingaben komplett im Textfeld Platz finden. Außerdem sollte der Benutzer zwischen optionalen und verpflichtenden unterscheiden können (z.B. durch Untergrundtöne). Im Feld sollten häufig vorkommende Eingaben als Standardvorbelegung für das Feld vordefiniert sein. Text sollte in einem Textfeld linksbündig angeordnet sein, während Zahlen rechtbündig angeordnet. Außerdem sollten alle nicht-nummerischen Eingaben bei Zahlenfelder nicht möglich sein.[[16]](#footnote-16)

In der vorliegenden Applikation wurden u.a. Textfelder für das Anlegen eines Neuen Spielers in einem Club. In dem dafür notwendigen SpielerAddView existieren zwei einzeilige JTextfelder. Zu erst wird der Name des Spielers als alphanummerischen Wert eingeben. Dafür besitzt dieses Textfeld wie im Balzert beschrieben eine links-bündige Ausrichtung. Anschließend erfolgt die Angabe der bisherigen Toranzahl für den jeweiligen Spieler als Zahlenwert, wodurch wie im Balzert beschrieben eine rechtsbündige Textausrichtung gewählt wurde. Notwendigen Überprüfungen, ob z.B. das Textfeld für die Namenseingabe nicht leer ist oder ob bei der bisherigen Toranzahl ein Zahlenwert eingeben wurde, erfolgt im SpielerAddPresenter, der nach erfolgreicher Eingabe einen neuen Spieler anlegt und diesen in dem dazugehörigen Club Objekt speichert.

//bei uns erklären: wir nutzen für uns eine Kombination aus MVC und MVP wir nutzen die strikte Trennung von Model und View aus dem MVP Pattern, jedoch die Verarbeitung von Benutzereingaben, die durch den Presenter geschenen aus dem MVC Pattern  für die Datenkonsitenz nutzen wir das Observerpattern (MainPresenter registriert sich beim LigaModel  ist das Hauptmodel an Liga ist alles dran)  Quelle die die Anwendung des Observer Patterns im MVC Bereich beschreibt

 Ziel: keine Beziehung zwischen Model und View & App- Weite Datenkonsitenz über mehrere View- Presenter- Beziehung durch ein einheitliches Hauptmodels (durch Aktualisierung des MainPresenters bei Änderung des Liga Models)

 warum geben wir wie was mit  manche Fenster Dialoge  brauchen ein Vater Fenster  Hauptinstanz die Alles Kontrolliert: MainPresenter  hier allgemeine Anzeige im Hauptfenter (und damit auch weitere Intialisierung von View- Presenter Beziehungen)  & Aktualisierung der Daten über das Observer pattern  app Funktioniert und weitere Fenster bleiben offen und nur Datenaktualisierung (Bsp.: Hinzufügen von Spielern (im ClubView), weil wir bei Aktualisierung im MainPresenter nicht dessen Zustand ändern d.h. alle offenen Presnter- View Beziehungen, die durch den MainPresenter initlaiisert wurden bleiben so bestehen  das gleiche gilt auch für alle nachfolgenden Presenter- View Beziehungen  sinnvoll wegen der Komplexittät der Applikation (models können von mehreren Controllern aktualisiert werden, und von mehreren Views dargestellt werden)

 bei uns: wir arbeiten mit einem einheitlichen Datenmodell (der ausgwählten Liga)  deswegen auch nur eine einheitliche, zentrale Update Schnittstelle

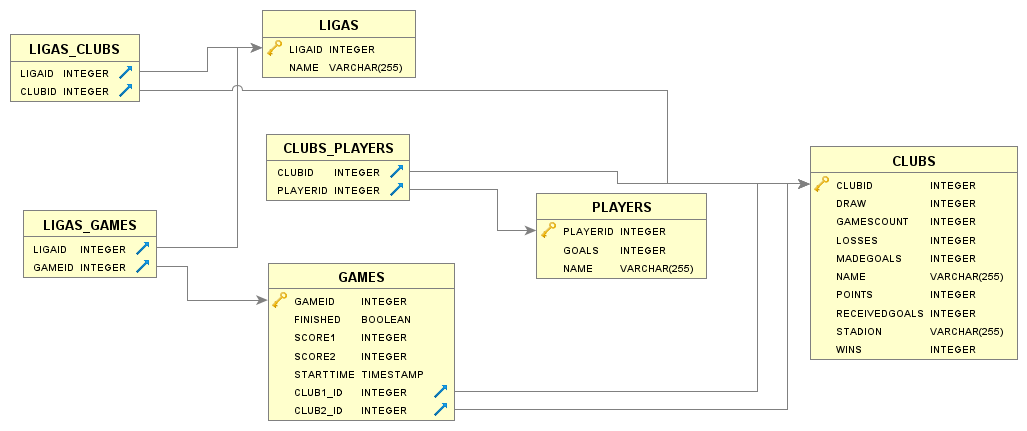
 siehe S. 449 Subsystem der Benutzeroberfläche

Anhang

Anhang 1 Klassendiagramm (Eigene Darstellung)

Anhang 2 Paketdiagramm (Eigene Darstellung)

Anhang 3 Datenbank-Diagramm (Eigene Darstellung)



1. Literaturverzeichnis

1 Balzert, Heide (Lehrbuch der Objektmodellierung; 1999): Lehrbuch der Objektmodellierung: Analyse und Entwurf, Spektrum Akademischer Verlag, Heidelberg, Berlin, 1999

2 Balzert, Helmut (Lehrbuch der Softwaretechnik; 2011): Lehrbuch der Softwaretechnik, Spektrum Akademischer Verlag, Heidelberg, 2011

3 Gharbi, Mahbouda; Koschel, Arne; Rausch, Andreas; Starke, Gernot (Basiswissen für Softwarearchitekten; 2018): Basiswissen für Softwarearchitekten, dpunkt Verlag, Heidelberg, 2018

1. Vgl. Balzert, H.; Lehrbuch der Softwaretechnik; 2011; S. 62f. [↑](#footnote-ref-1)
2. Vgl. Balzert, H.; Lehrbuch der Softwaretechnik; 2011; S.54-56 [↑](#footnote-ref-2)
3. Vgl. Balzert, H.; Lehrbuch der Softwaretechnik; 2011; S. 67f. [↑](#footnote-ref-3)
4. Vgl. Gharbi, M./ Koschel, A./ Rausch, A., et al.; Basiswissen für Softwarearchitekten; 2018; S. 79 [↑](#footnote-ref-4)
5. Vgl. Balzert, H.; Lehrbuch der Softwaretechnik; 2011; S. 458 [↑](#footnote-ref-5)
6. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 194 [↑](#footnote-ref-6)
7. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 195 [↑](#footnote-ref-7)
8. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 197-199 [↑](#footnote-ref-8)
9. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 199 [↑](#footnote-ref-9)
10. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 200 [↑](#footnote-ref-10)
11. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 201 [↑](#footnote-ref-11)
12. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S.202f. [↑](#footnote-ref-12)
13. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 215 [↑](#footnote-ref-13)
14. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 218 [↑](#footnote-ref-14)
15. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 217 [↑](#footnote-ref-15)
16. Vgl. Balzert, H.; Lehrbuch der Objektmodellierung; 1999; S. 215f. [↑](#footnote-ref-16)