Software-Entwicklung 2 Projekt:

ss401
bb071
jh170

CRM_System [SE2_CRM-System3 master]

Pfad zum Git-Repository :
https://version.mi.hdm-stuttgart.de/git/SE2_CRM-System

CRM-System (Customer Relationship Management):

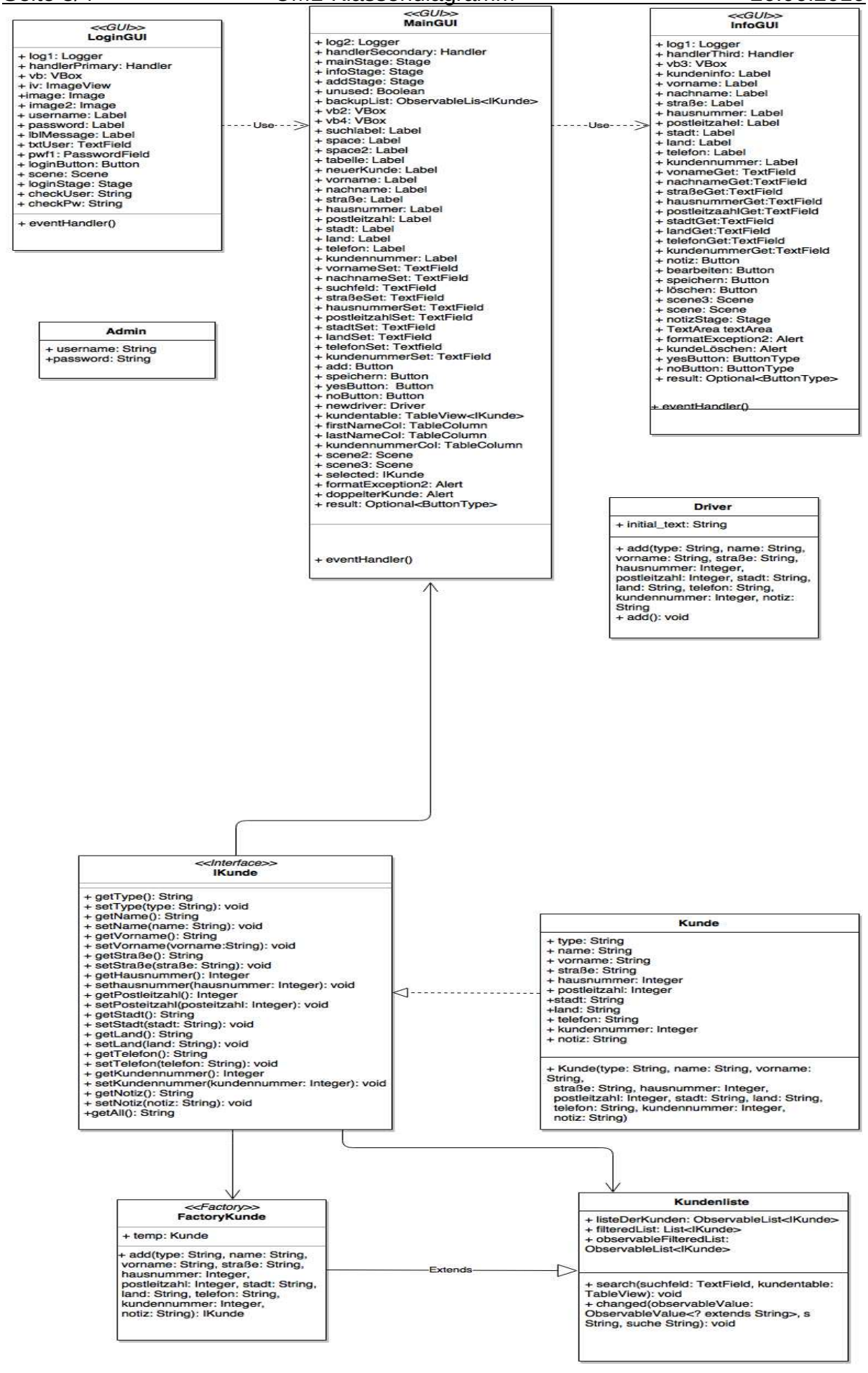
- Das Programm dient der Verwaltung und Pflege von Kundendaten. Um sich einloggen zu können muss man über Administratorrechte verfügen.
- In dem Hauptfenster wird eine Übersicht aller Kunden in einer Tabelle angezeigt. Diese beinhaltet die wichtigsten Daten eines Kunden.
- Für die erleichterte Verwaltung gibt es ein Suchfeld. Dieses filtert die Tabelle nach Kunden, deren Namen, Vornamen oder Kundennummern mit dem Suchbegriff beginnen.
- Beim Anlegen eines neuen Kunden, wird die Kundennummer automatisch generiert, dies vermeidet eine doppelte Vergabe. Des weiteren wird überprüft, ob ein identischer Kunde bereits existiert. Sollte dies der Fall sein wird eine Warnung geöffnet, die den Benutzer darauf aufmerksam macht.
- Es gibt auch eine Fehlermeldung, sollte man für die Postleitzahl oder die Hausnummer aus versehen einen Buchstaben eingeben.
- Durch Doppelklick auf einen Kunden in der Tabelle, wird ein Fenster mit allen Kundendaten geöffnet. Diese lassen sich, bis auf die Kundennummer, bearbeiten und speichern. Durch Drücken des Lösch-Buttons öffnet sich eine Abfrage, die sicherstellen soll das der Benutzer den Kunden auch wirklich löschen will.
- Außerdem lassen sich zu jedem Kunden Notizen anlegen und speichern.
- Durch schließen des Hauptfensters werden auch alle anderen offenen Fenster geschlossen.

Startklasse: Die main-Methode befindet sich in der LoginGUI

Besonderheiten:

Probleme:

- Beim Beenden unseres Programms wird der backup Thread nicht beendet, stattdessen führt er weiterhin die Ausgabe auf der Console durch.
  Befehle wie: thread.stop() , thread.interrupt() , thread.setDeamon(true) , System.exit() haben alle nicht funktioniert.
- Die Sichtbarkeit unserer Klasse Kunde kann nicht auf default gesetzt werden. Wird das Programm gestartet, während das public entfernt ist, kommt diese Fehlermeldung: `java.lang.RuntimeException: java.lang.IllegalAccessException: Class sun.reflect.misc.Trampoline can not access a member of class persons.Kunde with modifiers "public"`
  Da Tobias die Vermutung hatte, dass der Fehler an dem HTMLEdtior in unserem Kunde liegt, haben wir diesen entfernt. Der Fehler tritt jedoch weiterhin auf.
- Da eine ObservableList nicht Thread sicher ist, wollten wir unsere Threads in „guarded Blocks“ setzten. Bei der synchronized() Methode trat jedoch folgender Fehler auf: „Syntax error, insert "EnumBody" to complete BlockStatement“.
  Um einen Fehler bei gleichzeitigem Zugriff beider Threads auf die ObservableList zu vermeiden, haben wir nun die Liste synchronisiert.


<<GUI>>
LoginGUI
+ log1: Logger
+ handlerPrimary: Handler
+ vb: VBox
+ iv: ImageView
+image: Image
+ image2: Image
+ username: Label
+ password: Label
+ lblMessage: Label
+ txtUser: TextField
+ pwf1: PasswordField
+ loginButton: Button
+ scene: Scene
+ loginStage: Stage
+ checkUser: String
+ checkPw: String
+ eventHandler()
Use
<<GUI>>
MainGUI
+ log2: Logger
+ handlerSecondary: Handler
+ mainStage: Stage
+ infoStage: Stage
+ addStage: Stage
+ unused: Boolean
+ backupList: ObservableLis<IKunde>
+ vb2: VBox
+ vb4: VBox
+ suchlabel: Label
+ space: Label
+ space2: Label
+ tabelle: Label
+ neuerKunde: Label
+ vorname: Label
+ nachname: Label
+ straße: Label
+ hausnummer: Label
+ postleitzahl: Label
+ stadt: Label
+ land: Label
+ telefon: Label
+ kundennummer: Label
+ vornameSet: TextField
+ nachnameSet: TextField
+ suchfeld: TextField
+ straßeSet: TextField
+ hausnummerSet: TextField
+ postleitzahlSet: TextField
+ stadtSet: TextField
+ landSet: TextField
+ telefonSet: Textfield
+ kundenummerSet: TextField
+ add: Button
+ speichern: Button
+ yesButton: Button
+ noButton: Button
+ newdriver: Driver
+ kundentable: TableView<IKunde>
+ firstNameCol: TableColumn
+ lastNameCol: TableColumn
+ kundennummerCol: TableColumn
+ scene2: Scene
+ scene3: Scene
+ selected: IKunde
+ formatException2: Alert
+ doppelterKunde: Alert
+ result: Optional<ButtonType>
+ eventHandler()
Use
<<GUI>>
InfoGUI
+ log1: Logger
+ handlerThird: Handler
+ vb3: VBox
+ kundeninfo: Label
+ vorname: Label
+ nachname: Label
+ straße: Label
+ hausnummer: Label
+ postleitzahel: Label
+ stadt: Label
+ land: Label
+ telefon: Label
+ kundennummer: Label
+ vonameGet: TextField
+ nachnameGet:TextField
+ straßeGet:TextField
+ hausnummerGet:TextField
+ postleitzaahlGet:TextField
+ stadtGet:TextField
+ landGet:TextField
+ telefonGet:TextField
+ kundenummerGet:TextField
+ notiz: Button
+ bearbeiten: Button
+ speichern: Button
+ löschen: Button
+ scene3: Scene
+ scene: Scene
+ notizStage: Stage
+ TextArea textArea
+ formatException2: Alert
+ kundeLöschen: Alert
+ yesButton: ButtonType
+ noButton: ButtonType
+ result: Optional<ButtonType>
+ eventHandler()
Admin
+ username: String
+password: String
Driver
+ initial_text: String
+ add(type: String, name: String, vorname: String, straße: String, hausnummer: Integer, postleitzahl: Integer, stadt: String, land: String, telefon: String, kundennummer: Integer, notiz: String
+ add(): void
<<Interface>>
IKunde
+ getType(): String
+ setType(type: String): void
+ getName(): String
+ setName(name: String): void
+ getVorname(): String
+ setVorname(vorname:String): void
+ getStraße(): String
+ setStraße(straße: String): void
+ getHausnummer(): Integer
+ sethausnummer(hausnummer: Integer): void
+ getPostleitzahl(): Integer
+ setPosteitzahl(posteitzahl: Integer): void
+ getStadt(): String
+ setStadt(stadt: String): void
+ getLand(): String
+ setLand(land: String): void
+ getTelefon(): String
+ setTelefon(telefon: String): void
+ getKundennummer(): Integer
+ setKundennummer(kundennummer: Integer): void
+ getNotiz(): String
+ setNotiz(notiz: String): void
+getAll(): String
Kunde
+ type: String
+ name: String
+ vorname: String
+ straße: String
+ hausnummer: Integer
+ postleitzahl: Integer
+stadt: String
+land: String
+ telefon: String
+ kundennummer: Integer
+ notiz: String
+ Kunde(type: String, name: String, vorname: String, straße: String, hausnummer: Integer, postleitzahl: Integer, stadt: String, land: String, telefon: String, kundennummer: Integer, notiz: String)
<<Factory>>
FactoryKunde
+ temp: Kunde
+ add(type: String, name: String, vorname: String, straße: String, hausnummer: Integer, postleitzahl: Integer, stadt: String, land: String, telefon: String, kundennummer: Integer, notiz: String): IKunde
Extends
Kundenliste
+ listeDerKunden: ObservableList<IKunde>
+ filteredList: List<IKunde>
+ observableFilteredList: ObservableList<IKunde>
+ search(suchfeld: TextField, kundentable: TableView): void
+ changed(observableValue: ObservableValue<? extends String>, s String, suche String): void

Erweiterte Funktionalitäten:

Lambda Expressions:
Bei der Erstellung unserer Threads haben wir Lambda-Ausdrücke benutzt um die run() Methoden zu überschreiben.

Streams:
Beim Speichern eines neuen Kunden verwenden wir einen Stream um unsere Kundenliste nach einem identischen Kunden zu durchsuchen.
In unserer Suchfunktion wird der Stream verwendet um die Kundenliste nach einem Kunden zu durchsuchen, der auf den Suchbegriff passt.

Exception Handling:

Bei dem Anlegen eines neuen Kunden oder der Bearbeitung von Kundendaten kann es zu einer FormatException kommen, wenn man für die Plz oder die Hausnummer einen String eingibt. Diese wird abgefangen und es wird eine Fehlermeldung geöffnet, die dem Benutzer sagt, er soll eine gültige Zahl eingeben.

Multi Threading:

Durch Synchronisation unserer Kundenliste, können der backupThread und der refreshThread gleichzeitig arbeiten.