Einleitung (Franzi)

Gliederung (Franzi)

1. Aufgabenstellung
2. Projektplanung hinsichtlich der verwendeten Methoden und Werkzeuge
3. Anforderungsanalyse anhand der Aufgabenstellung
4. Programmentwurf mithilfe von Mockups und eines Use-Case Diagramm
5. Umsetzung in Java
6. Kritische Reflexion

Aufgabenstellung (Gregor)

- Eingabemaske, die Eingabe von Kreditdaten ermöglicht
- Zinsen berechnen
- Zurückzuzahlende Betrag einer Liste hinzufügen
- JSON-Datei mit den Kreditdaten der Kredite erstellen

Projektplanung (Max)

- **Vorgehen mit Wasserfallmodell**
  - Einfaches Problem, daher keine stetigen Updates
  - Support Schritt weggelassen, da Programm nicht released werden soll
- **MVC**
- **IntellIJ**
  - Einfache Entwicklung, dank sehr guten Code Completion
  - VCS sehr einfach, durch Git und GitHub Integration, zudem Maven vollintegriert
  - Zudem leichter dank Vorwissen innerhalb Gruppe
- **Git/GitHub**
  - Git in der Gruppe schon bekannt
  - Wahl auf GitHub, da Integration in IDE, zudem GitHub Desktop direkt verknüpft
  - Aufgrund kleinen Umfang, egal welche Plattform gewählt wird, da weitere Funktionen wie CI/CD Pipelines erst bei größeren Projekten interessant werden

Anforderungsanalyse (Gregor)

- Entscheidung für MVC und 3 Schicht-Architektur ohne DB zu befolgen, da Eigenschaften - UI, Anwendungslogik und Datenabruf - gegeben waren
- Model - Vorgabe der Eigenschaften von Krediten
  - Kreditbetrag
  - Zinssatz
  - Laufzeit
  - Rückzahlbetrag
  - Zinsbetrag
- View - Eingabemaske, Liste der Kredite, Trigger der Berechnung
- Controller - Verwaltung der Kredite, Schreiben/Lesen von JSONs, Berechnung der Kredite
- Selbstständige Erweiterung der Aufgabe um folgende Punkte
  - **Kreditarten:** Annuitäts-, Fälligkeits- und Abzahlungskredit
  - Zahlart: monatlich, jährlich
  - Auswahl von Krediten in Liste, erneute Modifikation und Berechnung

Entwurf (Gregor/Franzi)

- Use-Case (Gregor)
- UML-Diagramme

- Mockups

Umsetzung (Max/Gregor)

- Branches zeigen
- Grundsätzlich Maven Projekt, für Extensions
- **Model - Credit.java**
  - Orientiert an Java Spring, um Programm an DB anzuschließen und gleichzeitig eine API zu bieten können (nicht umgesetzt)
  - fast nur Getter/Setter
  - setParameters, um unkompliziert Objekt zu laden.
  - ansonsten Implementierung aller Attribute
- **Controller**
  - würde im Fall der Erweiterung zu einem REST Controller umgebaut werden können
  - Hier Punkt, um API Requests zu verarbeiten
  - Daher wird hier Verarbeitung der Daten gemanaged
  - createObject, saveObject, loadObjectById, loadAllObjects
  - Da keine API, sondern JSON Workaround mit convertObjectToJSON Methode
  - Benutzung von 2 Maven Repos: json-simple, jackson
  - **json-simple für Speichern/Lesen von JSONs**
    - oft genutztes Package für JSON
    - erleichtert Handling von JSON -> keine eigenen Parser nötig
    - Welche Klassen werde genutzt
      - JSONParser - Lesen aus Textdatei
      - JSONList - Erstellen einer Liste aus JSONObject
      - JSONObject enthält Art Key,Value Paar (ID und Object)
  - **ObjectMapper aus Jackson Databind**
    - In vielen Tutorials verwendet
    - Jackson großer Project, daher viele Funktionen, diese auch getestet
    - Wegfall eines Parsers des Objektes
    - Eigener Parser wäre auch denkbar
    - ObjectMapper .readValue erstellt ein Object (Parser), .writeValueAsString schreibt das Objekt als JSON
  - Allgemein - Nutzung von Maven mit externen Repositories erlernen
- **View (Gregor)**
  - Aufbau auf JFrame aus javax.swing Paket
  - Singleton Pattern
  - verschiedene Panels und Layouts
  - BorderLayout - Top, Center, Bottom
  - CenterLayout -
  - Controller benötigt
  - initialize() erzeugt Fenster und initialisiert alle UI Element mit Listeners
  - Ausführen der User Interaktionen über restlichen Funktionen
  - onSaveClick() - Objekt Speichern und Liste neuladen
  - onCalculateClick() - interestAmount berechnen und im UI anzeigen
  - onCancelClick() - Löscht die Werte aus allen Feldern
  - getValuesFromCredit() - Auswählen eines Kredites aus der Liste
- **Calculation (Franzi)**
  - Kalkulation wurde in extra Klasse ausgelagert - Darstellung der Anwendungslogikschicht
  - 3 Methoden zum Berechnen der einzelnen Kreditarten und eine obere Methode, die anhand eines Enums die verschiedenen Methoden triggert
  - theoretisch als einzelne Unit verwendbar
  - Erklärung Berechnung

Kritische Reflexion (Franzi)

- Schnelle Entwicklung des Programmes
- Wenig Planung -> Feature Ideen sind später noch dazu gekommen und mussten eingearbeitet werden
- Sehr gute Zusammenarbeit im Team

- Gute Verwendung von Git, wenn gleich nicht alle Funktionen (Branch Locking, CI/CD) ausprobiert/verwendet wurden
- 3-Schichten Modell nicht strikt, aber projektgemäß umgesetzt
- Einsatz von Maven als Package Management System