? ½ẼỊỀ ŋļ ŢĻ ¼ẩ Ǘŋļ Ễ

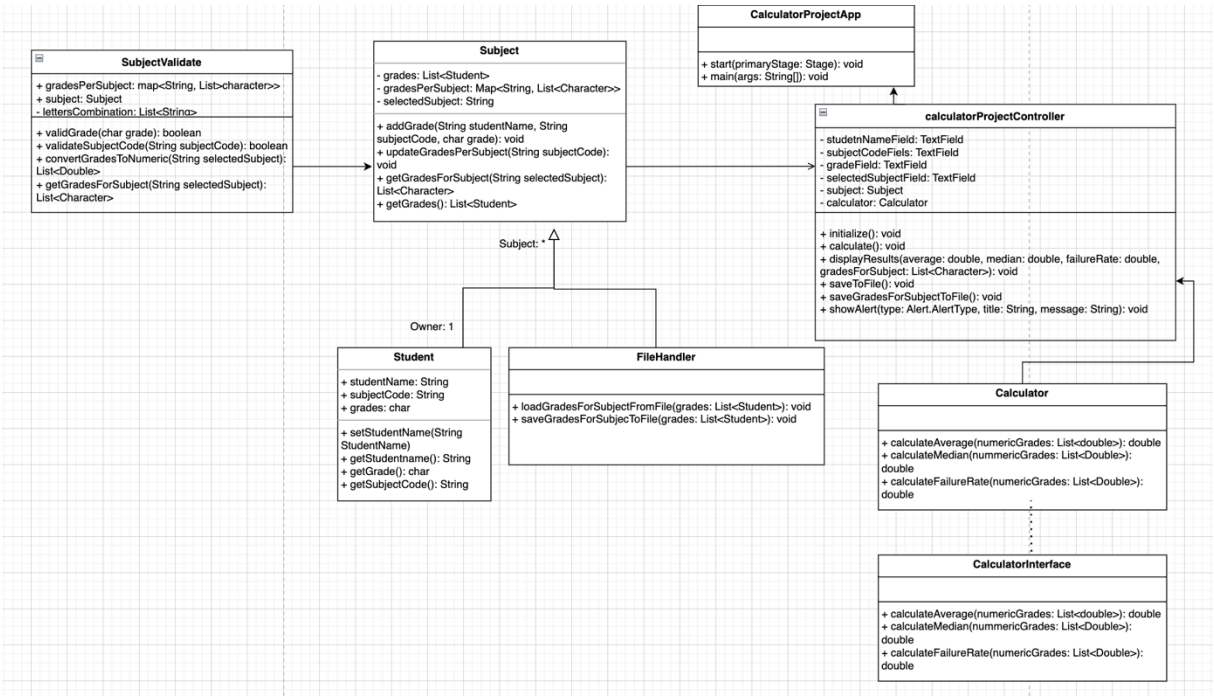
Ề

1. **Beskrivelse av appen**

   Jeg har laget en karakterkalkulator, dette er et verktøy som gir studentene mulighet til å registrere karakterene sine for ulike fag og deretter vil de få en «analyse». Med et brukervennlig grensnitt kan studentene legge inn karakterer for individuelle fag og deretter velge spesifikke emner for å se gjennomsnittskarakter, median og strykprosent. Systemet sørger for at hver student bare kan legge inn én karakter per fag, for å sikre gode analyser. Dataen som hentes inn lagres i en tekstfil slik at en har enkel tilgang til å hente ut alle karakterene til en bestemt emnekode i fremtiden. Dersom studenten har forbedret karakteren i faget i senere tid, vil dette være mulig å legge inn enkelt. Dette for å sikre muligheten på oppdatert data til enhver tid.

   Karakterkalkulatoren vil være et verdifullt verktøy for studenter som ønsker å holde oversikt og analysere sine prestasjoner. Samtidig vil det være verdifullt for studenter som skal ta faget i fremtiden å se hvordan karakterfordelingen har vært i faget tidligere.

2. Diagram



3. Spørsmål

a. *Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For eksempel bruk av arv, interface, delegering osv.)*

Først og fremst har jeg organisert koden i klasser som representerer de ulike konseptene som student, subject, calculator, filehandeling og applikasjonsgrensesnittet. Hvor student-klassen har en konstruktør som oppretter nye studentobjekter, samt instansmetoder for å sette og hente egenskaper som studentname, subjectcode og grades.

Underveis i koden har jeg implementer unntakshåndtering for å validere input og håndtere feil. For eksempel kastes unntak hvis studentname or subjectcode er ugyldig, eller dersom karakteren ikke er gyldig. Interface er brukt i for av 'CalculatorInterface'. Grensesnittet definerer tre metoder: 'calculateAverage', 'calculateMedian', og 'calculateFailureRate'. Metodene er implementert og utvidet i 'Calculator'-klassen.

Jeg har brukt 'List' og 'Map'-samlinger for å lagre og organisere karakterene til studentene og karakterer for emnekoder. Dataen fra studentene skrives til og fra fil, dette skjer i klassen 'FileHandler' her lagres data mellom hver

gang. 'Calculator'-klassen gjennomfører beregningene av gjennomsnittskarakter, median og strykprosent.

Jeg har brukt JavaFX til å opprette et grafisk brukergrensesnitt i 'CalculatorProjectController' og 'CalcualtorProjectApp'-klassene. For å designe layouten er filen AppCalculator.fxml brukt.

b. *Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?*
I koden min har jeg ikke brukt arv, da jeg følte jeg ikke hadde noe naturlig sted å bruke det og det kunne heller ført til unødvendig kompleksitet. Dette kunne jeg brukt dersom jeg hadde hatt klasser som deler egenskaper eller funksjonalitet. Måten jeg kunne implementert arv er for eksempel å utvide kalkulatorfunkjsonaliteten. Dersom jeg skulle anvendt ulike kalkulatorer for ulike typer beregninger, kunne jeg opprettet en klasse kalt 'Calculator' med grunnleggende funksjonalitet, deretter kan andre arve fra denne klassen for å lage spesialiserte og ulike typer kalkulatorer.

c. *Hvordan forholder koden deres seg til Model-View-Controller-prinsippet? (Merk: det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller standarder. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)*
Jeg har gjort et forsøk på å implementere Model-View-Controller prinsippet.
<u>Model:</u>
- Her har jeg 'Student' og 'Subject'-klassene som fungerer som modeller i applikasjonen. Klassene representerer data. 'Calculator'-klassen regnes også som en del av modellen, klassen utfører beregninger basert på karakterdataen som gis.

<u>View:</u>

- Det er 'CalculatorProjectController'-klassen og 'AppCalculator.fxml'-fila som håndtere brukergrensesnittet. De definerer hvordan brukergrensesnittet skal se ut og hvordan brukeren kan interagere med det.

Controller:

- 'CalculaotrProjectController' – klassen har funksjon som en controller i applikasjonen. Den kobler dataen sammen med brukergrensesnittet, og tilpasser seg brukerens handlinger gjennom knappetrykk og kan oppdatere visning deretter. Dette er muligens den klassen jeg tror ansvaret kan være litt blandet og ikke best bygget opp. Da jeg føler klassen både kontrollerer View-logikk og Controller-logikk. Her hadde det kanskje vært en ide å dele inn i flere klasser for å følge prinsippet på en bedre måte samt gjøre koden lettere å forstå.

d. *Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)*

Jeg har ikke laget enhetstester til alle deler av koden, men jeg har testet appens funksjon i praksis og vil si jeg ikke har oppdaget noen feil ved bruk. Jeg valgte å lage enhetstester til kalkulatorfunksjonen av programmet. Dette fordi det er en karakterkalkulator og da svært viktig at denne delen av programmet fungerer riktig og gir tilbake riktige svar til studentene. Når det er mye data som er samlet vil dette også være mer tidkrevende å teste gjennom bare brukertest og ikke en enhetstest. Har derfor enhetstest for å teste gjennomsnitt, median for partall og oddetall samt strykprosent. I tillegg har jeg en enhetstest for konstruktøren i student-klassen. Dette fordi her opprettes objektet student som er svært viktig og relevant i resten av koden. Følte derfor det var viktig å lage ne test for å kontrollere at ting blir gjort riktig fra start. Samtidig har jeg skrevet enhetstest på skriving til og fra fil.