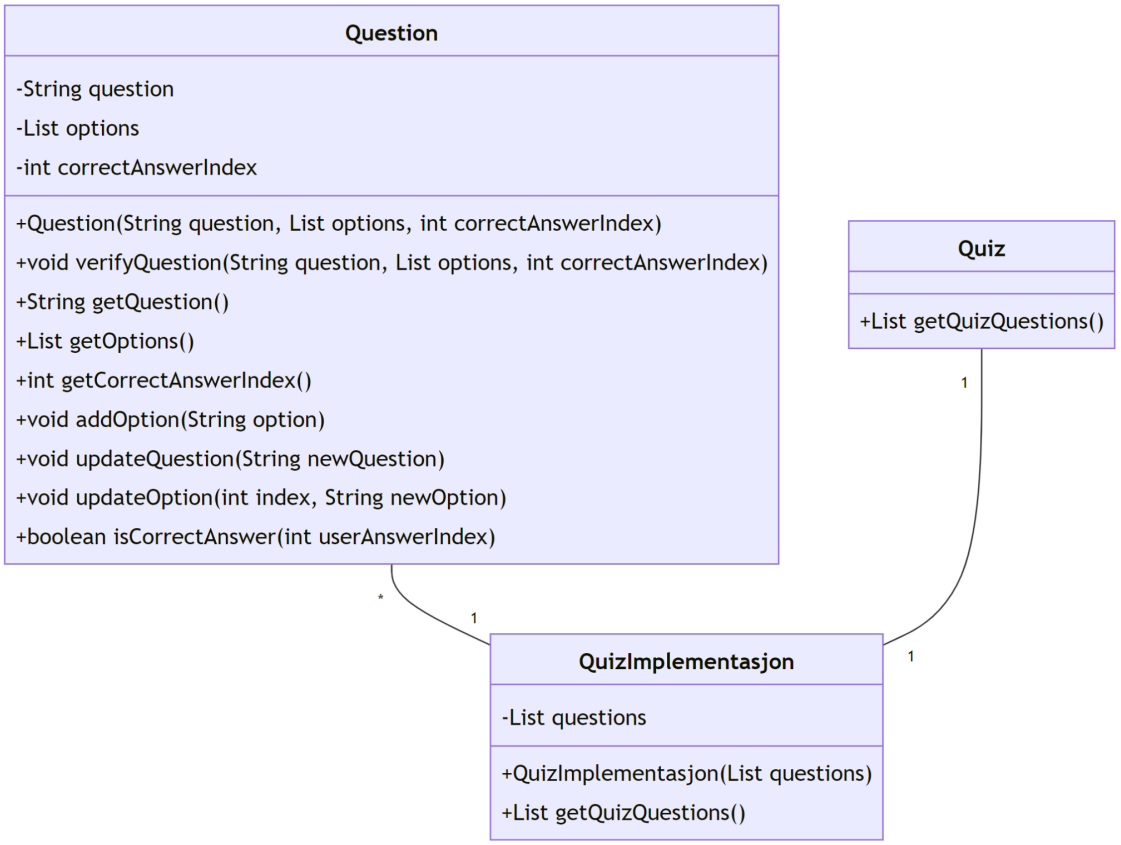
Beskrivelse av appen:

Kort om spillet/appen, hva skal den gjøre?

Vi skal lage en quiz-app. Appen skal kunne ta inn spørsmål og alternativer skilt av semikolon (;) i en tekstfil og vise spørsmålet og alternativene i et eget vindu. Riktig alternativ skal kunne spesifiseres i tekstfilen og alternativene skal blandes slik at de ikke alltid dukker opp i samme rekkefølge.

Når appen starter skal brukeren bli møtt av en knapp markert med “start” og en markert med “avslutt”. Knappen markert med “avslutt” skal la brukeren avslutte quizen og lukke vinduet. Knappen markert med “start” skal fjerne knappene og legge til spørsmål, svaralternativer og knapp for å lagre svar. Alternativene skal være i form av små knapper hvor bare én kan krysses av for hvert spørsmål. Når svaret lagres ved å trykke på knappen skal det ikke være mulig å endre svar, det skal gis ett poeng for riktig svar, ingen for feil svar.

Når siste spørsmålet er svart på, skal resultatet dukke opp og en knapp for å starte quizen på nytt. Da blir resultatet automatisk lagt til i en Scores.txt fil som skal samle alle scores fra tidligere forsøk.



Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte?
Blant pensum som er dekket i prosjektet er interface, delegering, enkapsulasjon, validering og Collections. Quizimplementasjon implementerer interfacet Quiz. Dette lar oss lage forskjellige implementeringer av Quiz med ulike regler og spørsmål. Selv om det ikke er direkte åpenbart, delegerer Quizimplementasjon oppgaven med å lese spørsmål fra fil til en egen metode getQuizQuestions. Vi bruker enkapsulasjon med private og offentlige metoder samt variabler slik at brukeren ikke skal kunne endre på hva som helst. I tillegg bruker vi validering i verifyQuestion().

Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?
Andre teknikker som kunne inkluderes er f.eks. arv, Iterator og Observer. Arv kunne blitt implementert for å lage flere typer quizer, men det er ikke så relevant i vårt tilfelle. Iterator kunne vært nyttig ved å iterere gjennom og få tilgang til spørsmålene og alternativene i quizen. Observer kunne blitt brukt til å varsle andre deler av appen når spesielle hendelser skjer. Det kan være for eksempel å varsle brukeren når en ny quiz er tilgjengelig når forrige quiz er fullført. En abstrakt Question-klasse kunne tjene som grunnlag for spesialiserte underklasser som MultipleChoiceQuestion og TrueFalseQuestion, hver med tilpassede valideringsmetoder og interaksjonsmekanismer.

Hvordan forholder koden deres seg til Model-View-Controller-prinsippet?
Med tanke på Model-View-Controll i prosjektet vårt, er Model (modellen) representert av QuizImplementasjon og Question. QuizImplementasjon håndterer logikken for å hente spørsmål og alternativer fra fil og organisere dem i lister, mens Question representerer hvert enkelt spørsmål. Selv om AppController fungerer effektivt som bindeledd mellom modell og visning, vil en strengere adskillelse av ansvar forbedre både skalerbarheten og vedlikeholdbarheten av appen. For eksempel kunne vi ha opprettet separate kontrollere for forskjellige funksjonalitetsområder, som brukeradministrasjon og quiz-styring. Dette ville redusert kompleksiteten i hver kontroller og gjort det lettere å håndtere, oppdatere, og teste hver del av systemet isolert.

View (visningen) er representert av App.fxml. Filen definererer struktur og utseende. AppController fungerer som et bindeledd mellom modell og visning. Den håndterer brukerinteraksjon og oppdaterer visning basert på data i modellen. En svakhet er at mye av logikken relatert til visning og brukerinteraksjon er implementert direkte i AppController, noe som kan føre til en viss blanding av ansvar.

Controller (Kontroller) er representert av AppController og håndterer brukerinteraksjon og oppdaterer visningen basert på brukerens handlinger. En svakhet er at AppController får mye ansvar da store deler av koden som gjør appen

funksjonell ligger der. I betrakning hadde det vært bedre å følge MVC-prinsippet mer strengt slik at man får et mer tydelig skille.

Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har?
Da vi skulle lage tester fokuserte vi først og fremst på Question-klassen fordi spørsmålene er en sentral del av en quiz appen. Vi valgte å teste konstruktøren i Question klassen, spesielt gyldighet av spørsmål og svaralternativer, samt håndtering av ulovlige argumenter som negative indekser eller en for stor indeks. Vi testet “ConstructorInvalidIndex” og “testOptionsIsEmpty” for å sikre at systemet kaster de forventede unntakene. Dette sørger for at appens brukervennlighet maksimeres. .

Ikke alle deler av koden er dekket like grundig. Vi valgte å fokusere på kjernefunksjonalitet og input validering fremfor mindre kritiske funksjoner.