Paths - Del 3

I denne siste delen av Paths-prosjektet skal dere anvende designmønster og implementere et grafisk brukergrensesnitt basert på skissene fra del 2. I tillegg får dere mulighet til å utvide programmet basert på egne ideer. Ta utgangspunkt i egen kode fra del 2 når dere løser oppgavene. Som vanlig skal dere levere besvarelsen i Blackboard og gjennomføre en godkjenningssamtale.

Det er mulig å jobbe med prosjektet helt frem til endelig innlevering i Inspera. Inspera vil være åpen for innlevering i tidsrommet 22.05 09:00 - 23.05 14:00.

Det kan være lurt å lese gjennom hele dokumentet før dere begynner på oppgavene.

Før dere begynner

Følgende krav og betingelser gjelder for alle oppgavene:

Enhetstesting

Det er ekstra viktig å lage enhetstester for den delen av koden som er forretningskritisk, altså for den koden som er viktigst for å oppfylle sentrale krav. Feil her vil få store negative konsekvenser for programmet.

Unntakshåndtering

Uønskede hendelser og tilstander som forstyrrer normal flyt skal håndteres på en god måte.

Versjonskontroll

Prosjektet skal være underlagt versjonskontroll. Sjekk inn jevnlig. Hver commit skal beskrive endringene som er gjort på en kort og konsis måte. Branching skal benyttes der det er hensiktsmessig.

Prosjektrapport

Det skal skrives en rapport for prosjektet. I rapporten skal dere forklare hvordan løsningen er bygget opp, hvilke valg som er tatt underveis, hvordan dere har anvendt designprinsipper osv. Rapporten skal være på maks 2500 ord. I tillegg kommer figurer og eventuelt små kodesnutter for å vise hvordan utvalgte problemer er løst. Det forventes at dere har innhold for alle kapitlene når dere har godkjenningssamtalen for del 3. Dokumentmal finnes på BB.

Oppgave 1: Builder

Anvend Builder design pattern på klassen Player, slik at det blir mulig å opprette ulike representasjoner av spillere med en dedikert builder.

Oppgave 2: GUI

I del 2 av prosjektet laget dere skisser for et grafisk brukergrensesnitt. Dere skal nå kode brukergrensesnittet i JavaFX. Vi gjør oppmerksom på at det ikke er tillatt å bruke FXML i løsningen, og at det skal være mulig å kjøre programmet fra terminalen med «mvn javafx:run». Detaljert kravliste for GUI er oppført i del 2.

Oppgave 3: Videre arbeid

Frem til nå har dere jobbet utifra veldefinerte oppgaver og krav. Underveis har dere kanskje fått egne ideer. I denne oppgaven oppfordrer vi dere til å være kreative og gå utenfor de begrensningene vi har satt. Det er dere som bestemmer, men her er noen forslag:

- Passasjer med multimedia (bilder, lyd, video, animasjoner)
- Kan en passasje være et lite program, f.eks. et minispill?
- Mulighet til å angre og gå tilbake i spillet (undo)
- Funksjonalitet for å opprette og redigere en historie direkte i GUI
- Persistering av goals slik at man slipper å legge til dette for hvert spill
- Støtte for andre filformater
- Hva med støtte for flere språk (i18n)?

Det forventes at dere tar prosjektet videre med egne idéer, men poenget er ikke å legge til mest mulig funksjonalitet. Kvalitet er viktigere enn kvantitet.

Viktige sjekkpunkter

Når dere løser oppgaven bør dere dobbeltsjekke følgende:

- Maven:
    - Er prosjektet et Maven-prosjekt med fornuftige prosjekt-verdier og gyldig katalogstruktur?
    - Kan man kjøre Maven-kommandoer for å bygge, teste, pakke, samt starte GUI uten at det feiler?
    - Er det mulig å bygge, teste, pakke, samt starte GUI fra terminalen med mvn?
- Versjonskontroll med git:
    - Er prosjektet underlagt versjonskontroll med sentralt repo?
    - Sjekkes det inn jevnlig?
    - Beskriver commit-meldingene endringene på en kort og konsis måte?
    - Benyttes brancher på en hensiktsmessig måte?
- Enhetstester:
    - Har enhetstestene beskrivende navn som dokumenterer hva testene gjør?
    - Følger de mønstret Arrange-Act-Assert?
    - Tas det hensyn til både positive og negative tilfeller?

    - Er testdekningen god nok?
- Er builder implementert for Player-klassen iht Builder design pattern?
- Oppfyller GUI kravene og betingelsene som beskrevet i del 2 og 3?
- Kodekvalitet:
    - Er koden godt dokumentert iht JavaDoc-standard?
    - Er koden robust (unntakshåndtering, validering med mer)?
    - Har variabler, metoder og klasser beskrivende navn?
    - Er klassene gruppert i en logisk pakkestruktur?
    - Har koden god struktur, med løse koblinger og høy kohesjon?
    - Benyttes linter (f.eks. SonarLint) og tilsvarende verktøy (f.eks. CheckStyle) for kvalitetssjekk?