Paths - Del 2

I del 1 av Paths-prosjektet implementerte dere grunnleggende klasser og logikk. I denne delen skal dere utvide programmet med filhåndtering og funksjonell programmering. Dere skal også skissere et grafisk brukergrensesnitt. Ta utgangspunkt i egen kode for del 1 når dere løser oppgavene under. Til slutt leverer dere besvarelsen og gjennomfører en godkjennings-samtale.

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

Dere må 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.

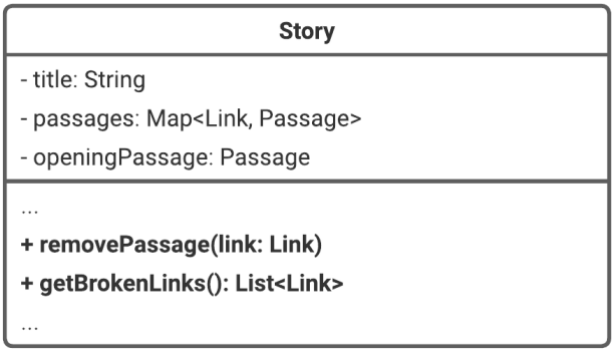
Versjonskontroll

Prosjektet skal være underlagt versjonskontroll. Sjekk inn jevnlig, og husk at 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 kommet i gang med rapporten når dere har godkjenningssamtalen for del 2. Dokumentmal finnes på BB.

Oppgave 1: Nye metoder i Story-klassen



Figur 1: Nye metoder i Story-klassen

I denne oppgaven skal dere legge til to nye metoder i Story-klassen:

- removePassage(link): fjerner en gitt passasje fra passages. Det skal ikke være mulig å fjerne passasjen hvis det finnes andre passasjer som linker til den.
- getBrokenLinks(): finner og returnerer en liste over døde linker. En link er død hvis den refererer til en passasje som ikke finnes i passages.

Dere skal bruke funksjonell programmering og streams for å løse denne oppgaven. Komplementer med enhetstester og husk versjonskontroll.

Oppgave 2: Filhåndtering

Programmet skal ha støtte for filhåndtering:

- Det skal være mulig å lagre en historie i en fil
- Det skal være mulig å lese en historie fra en fil
- Fila skal være en tekstfil og følge et bestemt format (se under)
- Filendelsen skal være «.paths»

Format

En fil skal inneholde informasjon om en – og kun en – historie. Historiens tittel lagres på den første linja, etterfulgt av en tom linje. Resten av fila lister opp passasjene, hvor den første får rollen som openingPassage. Hver passasje kan betraktes som en blokk med tekst som går over flere linjer:

- En ny passasje starter med :: (to kolon) og passasjens tittel
- Den neste linjen holder passasjens innhold (content)
- De siste linjene i en blokk lister opp linkene
- Hver link får sin egen linje og er på formen [text](reference)
- En blokk avsluttes med en tom linje

Eksempel:

```
Haunted House

::Beginnings
You are in a small, dimly lit room. There is a door in front of you.
[Try to open the door](Another room)

::Another room
The door opens to another room. You see a desk with a large, dusty book.
[Open the book](The book of spells)
[Go back](Beginnings)
...
```

Som dere ser støtter ikke formatet actions. Dette er ikke et absolutt krav, men vi anbefaler at dere utvider formatet slik at det også er mulig å knytte en eller flere actions til en link. Hvordan dette gjøres er opp til dere. Støtte for actions som er kodet på en robust måte vil telle positivt i sluttvurderingen av prosjektet.

I denne oppgaven er det ekstra viktig at dere tester godt, da det er mye som kan gå galt når man håndterer filer (feil tegnsett, korrupte data, ugyldig formatering osv).

Et lite tips til slutt: java.util.Scanner har nyttige metoder som gjør det litt enklere å parse tekst.

Oppgave 3: GUI (skisser)

Et grafisk brukergrensesnitt vil gjøre det enkelt å gjennomføre et spill. I denne oppgaven skal du lage en eller flere skisser som illustrerer utseende og layout. Du kan bruke et dedikert verktøy som Balsamiq Wireframes (mer info på BB) eller ganske enkelt tegne med penn og papir og skanne eller ta bilde i etterkant.

Når du utarbeider skissen(e) må du forholde deg til følgende krav og betingelser:

- Det skal være mulig å laste inn en historie fra en .paths-fil.
- Brukergrensesnittet bør opplyse om filens navn, lokasjon og evt døde linker.
- Før man starter et spill må man kunne legge inn informasjon om en spiller (navn, helse og gullbeholdning).
- Før man starter et spill må man kunne legge inn goals. Man skal kunne legge til så mange goals man vil.
- Det skal være mulig å gjennomføre et spill fra start til slutt.
- Spillerens helse, poengsum, gullbeholdning og inventar skal være synlig gjennom hele spillet.
- Det må være mulig å restarte et spill. En restart tar spilleren tilbake til første passasje og nullstiller helse, poengsum, gullbeholdning og inventar.
- Brukeren må kunne få informasjon om hvordan man spiller (brukerveiledning/hjelp).

- Brukergrensesnittet skal ha en oversiktlig layout med tydelige komponenter, god kontrast og fornuftige feilmeldinger.

Implementering av brukergrensesnittet er en oppgave i del 3 av prosjektet, men det er lov å tyvstarte allerede nå. Vi gjør oppmerksom på at GUI skal kodes fullt og helt i JavaFX, og at det ikke er tillatt å bruke FXML i løsningen.

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 og pakke uten at det feiler?
  - Er det mulig å bygge, teste og pakke 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 metodene «removePassage» og «getBrokenLinks» implementert iht oppgavebeskrivelsen?
- Filhåndtering:
  - Er det mulig å lese en historie fra fil på oppgitt format?
  - Er det mulig å skrive en historie til fil på oppgitt format?
  - Hensyntas feil i data, format og filendelse?
  - Lukkes ressurser etter bruk?
  - Har formatet støtte for actions (ikke et absolutt krav, men anbefales løst)
- Er GUI-skissen(e) utarbeidet på en måte som oppfyller kravene og betingelsene i oppgaven?
- Kodekvalitet:
  - Er koden godt dokumentert iht JavaDoc-standard?
  - Er koden robust (unntakshåndtering, validering mm)?
  - 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?