Wargames - Del 2

I del 1 av Wargames-prosjektet implementerte du enheter og simuleringslogikk. I denne delen skal du videreutvikle programmet ditt med filhåndtering og funksjonell programmering, samt skissere et grafisk brukergrensesnitt. Du skal ta utgangspunkt i din egen kode for del 1 når du løser oppgavene under. Til slutt leverer du besvarelsen i Blackboard. Du vil få tilbakemelding på besvarelsen din i etterkant og kan gjøre forbedringer helt frem til endelig mappeinnlevering.

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

Før du begynner

Følgende krav og betingelser gjelder for kodeoppgavene:

Enhetstesting

Du 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.

Maven

Prosjektet skal være et Maven-prosjekt med en fornuftig groupId og artefactId, og følge JDK v11 eller høyere. Filer skal lagres iht standard Maven-oppsett. Det skal være mulig å bygge, teste og kjøre med Maven uten feil.

Versjonskontroll

I del 1 ble koden lagt under lokal versjonskontroll. Nå skal det lokale repoet også kobles mot et sentralt repo. Krav til versjonskontroll er ytterligere beskrevet i oppgave 1.

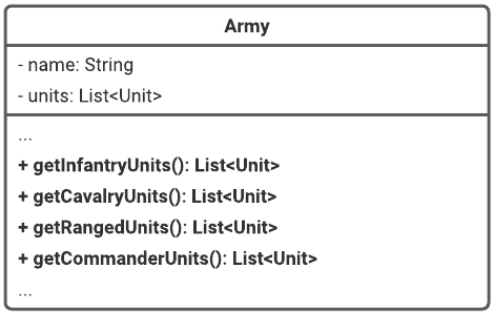
Oppgave 1: Versjonskontroll med git

Prosjektet skal være underlagt versjonskontroll og være koblet mot et sentralt repo:

- Legg koden under lokal versjonskontroll hvis dette ikke allerede er gjort.
- Opprett så et nytt sentralt repo (tomt prosjekt) med samme navn på GitLab (studenter på campus Ålesund skal bruke GitHub Classroom).
- Til slutt kobler du lokalt repo mot sentralt repo.

For hver av oppgavene under skal du gjøre minst én innsjekk (commit) lokalt. Til slutt laster du opp alle endringene til sentralt repo (push). Hver commit-melding skal beskrive endringene som er gjort på en kort og konsis måte. Hvis du sjekker inn på slutten av en oppgave, men senere trenger å gjøre endringer i koden og sjekke inn på nytt, så er det selvfølgelig helt greit.

Oppgave 2: Nye metoder i Army-klassen



Figur 1: Nye metoder i Army-klassen

Klassediagrammet over lister fire nye metoder for å hente spesialiserte enheter fra en armé. Metoden getInfantryUnits() skal returnere de instansene i units som er av typen InfantryUnit, getRangedUnits() skal returnere instansene av typen RangedUnit osv. Enheter av typen CommanderUnit skal kun returneres fra metoden getCommanderUnits(), selv om slike enheter gjennom arv også er en type CavalryUnit.

Implementer de nye metodene i Army-klassen. Du skal bruke funksjonell programmering (stream og lambda) når du løser oppgaven. Komplementer med enhetstester og sjekk inn i versjonskontroll.

Oppgave 3: Filhåndtering

I denne oppgaven skal du implementere filhåndtering i programmet ditt:

- Det skal være mulig å lagre en armé i en fil
- Det skal være mulig å lese en armé fra en fil
- Fila skal være en tekstfil og følge et bestemt format (se under)

Format

En fil skal inneholde informasjon om en – og kun en – armé. Navnet på arméen lagres på den første linja. Resten av fila skal liste opp alle enhetene, med én enhet per linje:

Generisk format
filename.csv

```
army name
unit type,unit name,unit health
unit type,unit name,unit health
unit type,unit name,unit health
…
```

Eksempel
human-army.csv

```
Human army
InfantryUnit,Footman,100
InfantryUnit,Footman,100
CommanderUnit,Mountain King,180
…
```

Siden verdiene er kommaseparerte sier vi at dette er en CSV-fil (Comma Separated Values). Filendelsen skal derfor være «.csv».

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

Oppgave 4: GUI

Programmet vårt trenger et grafisk brukergrensesnitt. 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:

- Et slag involverer to arméer. Disse må presenteres på en oversiktlig måte i GUIet.
- Man må kunne se detaljer om hver armé. Dette inkluderer arméens navn, antall enheter totalt, antall enheter av hver type, samt informasjon om de enkelte enhetene (type, navn, helse osv).
- Det skal være mulig å laste inn arméene fra filer (en fil per armé). Brukergrensesnittet bør opplyse om lokasjon/navn til filene man har lest fra.
- Simuleringen er viktig. Som et minimum må man kunne starte simuleringen og bli opplyst om det endelige resultatet. Man må også kunne se tilstanden til de to arméene etter at slaget er over.
- Når man har kjørt en simulering må man kjapt og enkelt kunne resette arméene til opprinnelig tilstand og kjøre en ny simulering.
- Brukergrensesnittet må ha en oversiktlig layout med tydelige komponenter, god kontrast og fornuftige feilmeldinger.
- Det er et pluss, men ikke et absolutt krav, om brukergrensesnittet viser endringer i tilstanden til de to arméene samtidig som simuleringen kjører. Dette er ikke nødvendigvis lett å få til, og krever blant annet at man setter ned farten på simuleringen (f.eks. med Thread.sleep) og oppdaterer GUI-komponentene underveis.

Selve implementeringen av brukergrensesnittet er en oppgave i del 3 av prosjektet, men hvis du ønsker å tyvstarte nå så er det lov. Da er det verdt å merke seg at brukergrensesnittet skal kodes i JavaFX, og at man selv kan velge om man vil bruke FXML eller kode alt manuelt (ren Javakode).

Selv om skissen(e) ikke er kode er det fornuftig å også legge disse under versjonskontroll.

Viktige sjekkpunkter

Følgende momenter bør dobbeltsjekkes:

- Maven:
  - Er prosjektet et Maven-prosjekt med fornuftige prosjekt-verdier og gyldig katalogstruktur?
  - Kan man kjøre Maven-kommandoer for å bygge, installere, teste osv uten at det feiler?
- Versjonskontroll med git:
  - Er prosjektet underlagt versjonskontroll med lokalt repo?
  - Er det lokale repoet koblet mot et sentralt repo?
  - Finnes det minst én commit per kodeoppgave?
  - Beskriver commit-meldingene endringene på en kort og konsis 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 de fire nye metodene i Army-klassen implementert iht oppgavebeskrivelsen?
- Filhåndtering:
  - Er det mulig å lese en armé fra en fil på oppgitt format?
  - Er det mulig å skrive en armé til en fil på oppgitt format?
  - Hensyntas feil i format og filendelse?
  - Lukkes filene etter bruk?
- GUI-skisse:
  - Presenteres arméene på en oversiktlig måte og kan man se detaljert info iht krav og betingelser?
  - Er det mulig å laste inn arméene fra filer og oppgis filnavn/filsti?
  - Kan man starte simuleringen?
  - Illustreres situasjonen etter en simulering?
  - Er det mulig å resette arméene kjapt og enkelt for ny simulering?
  - Er layouten oversiktlig med tydelige komponenter og god kontrast?
- Kodekvalitet:
  - Er koden godt dokumentert iht JavaDoc-standard?
  - Er koden robust (verifiseres parametere før de brukes, er unntakshåndteringen solid 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?