Prosjektbeskrivelse

TDT4100 - Objektorientert programmering

Innleveringsfrist: 26/4-2021

Som en del av karaktervurderingen i TDT4100 - Objektorientert Programmering inngår et større prosjekt der dere skal lage en fungerende app. Dette dokumentet beskriver prosjektet, lister opp krav, frister, og generell informasjon som det er viktig å få med seg. Vi anbefaler på det sterkeste å lese **hele** dette dokumentet *før* dere starter å jobbe med prosjektet.

Tabell 1: Oversikt over viktige deler av prosjektet

Andel på karakter	35 %
Innleveringsfrist	26/4-2021
Vurderingsfrist hos studass	26/4-2021
Gruppestørrelser	1 eller 2 personer

Innhold

1 Generell informasjon

Prosjektet teller 35 % på sluttkarakteren i faget. Det kan gjøres alene eller i grupper på 2 personer. Merk at dersom dere gjør prosjektet i par forventes det at begge er delaktige i alle deler av

Tabell 2: Anbefalte arbeidsperioder for prosjektet

Fra	Til	Del	Tema
12/2	26/2	1	Grunnklasser og brukergrensesnitt
12/3	19/3	2-3	Lagring og filhåndtering
16/4	26/4	4	Testing

prosjektet, både i den grad at dere forstår hva som skjer, og med tanke på at begge skal skrive deler av koden. Studass vil passe på at begge forstår hele kodebasen når h*n sjekker prosjektet. Dere bør av den grunn benytte parprogrammering, altså at dere sitter sammen og programmerer, i stedet for å jobbe med hver deres del.

Prosjektet er delt opp i 4 deler. Innleveringsfristen på prosjektet er 26/4. I tillegg oppgir vi en valgfri frist på hver del. Det er ikke krav om at dere lever inn noe til disse tidspunktene, men de har to betydninger:

1. De viser en anbefalt fremdriftsplan for prosjektet i forhold til resten av faget.
2. Dersom dere ønsker tilbakemeldinger fra studass på de forskjellige delene, skal dere levere og snakke med studass innen disse fristene. Merk at dere ikke vil få en liste over alt som er feil i disse delvurderingene, de er ment å gi en generell pekepinn på hvordan dere ligger an.

Ved innlevering av prosjektet skal dere laste opp koden deres som en zip-fil til blackboard. I tillegg må dere demonstrere prosjektet for studass **innen fristen 26/4**. For dere som tar opp faget og derfor ikke har en studass kommer vi tilbake til demonstrasjonsinfo.

Dette dokumentet inneholder detaljer for alle delene av prosjektet. Vi anbefaler allikevel å fordele arbeidet med prosjektet utover semesteret, både fordi det er "hull" i øvingsopplegget satt av til å jobbe med det, og fordi de senere delene av prosjektet krever pensum som ikke gås gjennom før sent i semesteret. Det vil derfor sannsynligvis kreve mye ekstra jobb å gjøre hele prosjektet tidlig i semesteret. Anbefalte perioder å jobbe med prosjektet er listet opp i tabell 2.

2 Krav til sluttproduktet

Når dere er ferdig med prosjektet, skal dere ha produsert en fungerende app som oppfyller følgende krav:

- Appen skal bestå av minimum to interagerende klasser.
- Minimum en av klassene må ha noe funksjonalitet utover ren datalagring, en form for *kalkulasjoner* (i en utvidet betydning av begrepet). Dette kan være funksjonaliteten i et spill, utregning av matematiske uttrykk, kryssjekking av flere datakilder for å finne felles verdier, eller lignende.
- Alle klasser i appen skal ha innkapsling av tilstanden sin, samt validering ved endring av tilstanden der det er relevant.
- Appen skal ha et *brukergrensesnitt* laget i FXML, med tilhørende `Controller-` og `App-`klasser. Disse klassene telles *ikke* som de to klassene dere skal lage selv.
- Appen skal ha mulighet for lesing fra og skriving til fil, og må dermed inneholde noe det gir mening å lagre. Det kan være tilstanden i et spill, en oversikt over varer på lager i en butikk, innlegg i en dagbok, eller lignende.

- Det skal være implementert passende feilhåndtering på hensiktsmessige steder i appen.
- Det skal være laget et sett med JUnit-tester som sjekker at funksjonaliteten i appen fungerer som tenkt.

Kravene er repetert og utdypt i de deloppgavene som omhandler kravet.

2.1 Eksempel

Det er laget et eksempelprosjekt dere kan bruke som inspirasjon i arbeidet deres. For å få opp eksempelet i Eclipse må dere først oppdatere filene fra git (Høyreklikk på et av prosjektene og trykk *Team* → *Pull*). Deretter må prosjektet importeres:

1. Trykk *File* → *Import...*
2. Under *Git* i vinduet som kommer opp velger dere *Projects from Git*
3. I neste vindu velger dere *Existing local repository*, og trykker *Next*
4. Velg *v2021* i lista som kommer opp og gå videre
5. Trykk *Next* i neste vindu uten å endre på noe
6. Huk av for *todolist-example* i lista over prosjekter som kommer opp, og trykk *Finish*

Du skal nå ha et nytt prosjekt som heter *todolist-example* i Eclipse. Kjør en *Maven* → *Update Project* på dette, så skal det være klart til å kjøres. Koden for eksempelet kan dere også se på `https://gitlab.stud.idi.ntnu.no/tdt4100/v2021/students/-/tree/master/todolist-example`.

2.2 Forslag på temaer

Vi har laget noen forslag til temaer dere kan bygge appen deres rundt. Dere står fritt til å velge et av disse, eller et annet tema dere selv ønsker. Om dere velger et annet tema, må dere huske på at appen må være kompleks nok til å inneholde flere klasser, og kunne støtte lagring av tilstand. Vi anbefaler i så fall at dere snakker med studass før dere begynner, for å dobbeltsjekke at temaet er passende. Pass også på å **ikke** velge et av prosjektene listet under Ikke tillatte prosjekter.

Forslag til prosjekter:

- Rutenettbasert spill som Minesweeper eller Battleship. Siden spill kan være kompliserte å implementere, trenger dere ikke nødvendigvis å implementere alle deler, dersom dere føler at appen er kompleks nok. Snakk med studassen deres om dette.
- Digital dagbok/journal. Merk at dere må ha mer funksjonalitet enn å bare skrive innlegg, f.eks. kan dere implementere filtrering og sortering av innlegg, e.l.
- Enkelt kortspill, som f.eks. Blackjack. Vi anbefaler ikke å begynne på noe som krever en veldig intelligent AI-motspiller, da det fort blir veldig mye jobb. Husk også å tenke på hva som skal lagres.
- Karakter-system der man kan fylle inn karakterer, og så regner det ut snitt, median, finner beste og verste fag, e.l. Bonusoppgave her kan være å ha støtte for flere studenter, så man kan finne karakterfordeling per fag.
- Et bookingsystem for et hotell, der man kan legge inn bestillinger og få en oversikt over hva som er bestilt så langt.

2.3 Ikke tillatte prosjekter

Fra tidligere år har vi sett at en del velger prosjekter som enten ikke går an å utvide med funksjonaliteten som kommer i senere deler av prosjektet (lesing fra og skriving til fil), eller prosjekter som man finner helt ferdige løsninger på på nettet. Dette ønsker vi å unngå, og vi har derfor valgt å forby noen spesifikke prosjekter.

MERK: Selv om et prosjekt ikke er listet her, er det ikke lov å plagiere. Om dere sliter med enkelte ting er internett en god kilde, men pass på å ikke kopiere store deler av koden direkte. Dersom dere velger å kopiere noe kode bør dere referere til kilden med kommentarer i koden deres. Om store deler av prosjektet er copy-paste kan det føre til underkjennelse, og ved plagiat (altså uten kildereferanser) kan det håndteres som juks med de følger det kan få.

Forbutte prosjekter:

- TicTacToe
- Enkel kalkulator
- TodoList
- Prosjekter som gås gjennom i forelesninger/øvingsforelesninger i faget

2.4 Hvor skal koden skrives?

Her har dere to alternativer: Det første er å lage en ny *pakke* i et av `ovinger` og `minegenkode`-prosjektene i Eclipse, og skrive koden der. Husk i så fall å ha samme pakkenavn i `src/main/java` og `src/test/java` (vi anbefaler å bruke pakkenavnet `project`). Alternativ 2, som er det vi anbefaler, er å bruke et eget git-repo. Vi har generert et repo til hver av dere på GitLab, som dere kan importere i Eclipse og bruke. Det er laget videoguider på hvordan dere setter opp disse prosjektene på Blackboard, i tillegg til noen introvideoer på bruk av git generelt. Om dere jobber i par er det spesielt anbefalt å bruke dette alternativet, men merk da at dere bør velge ett av repoene deres til koden i stedet for å bruke hver deres.

For ordens skyld gjør vi oppmerksom på at alle studasser, samt fagstaben, har tilgang til repoene deres. Det betyr at en enkel måte å dele koden med studass er ved å dele lenka til repoet dere bruker. Samtidig betyr det også at dere ikke bør legge opp noe hemmelig i disse repoene.

3 Del-beskrivelser

3.1 Del 1: Grunnklasser og brukergrensesnitt

Frivillig innleveringsfrist: 26/2

I denne delen skal dere lage de grunnleggende klassene i appen deres. Klassene skal realisere hovedfunksjonaliteten i appen, men dere trenger ikke å implementere feilhåndtering og lagring i denne delen (det kommer senere). Tilstanden til objektene i appen deres skal være innkapslet, og et objekt skal administrere sin egen tilstand.

I tillegg til grunnklassene skal dere lage et brukergrensesnitt i FXML. Vi anbefaler at dere tenker litt over dette grensesnittet, gjerne skissér det på forhånd for å se at det virker greit å bruke. Om dere har tatt eller tar *TDT4180 - Menneske-maskin-interaksjon* kan det være lærerikt

å bruke noen av konseptene dere har lært der (men merk at dere ikke vurderes ut i fra det i dette faget).

Dere skal også lage `Controller`- og `App`-klasser som starter appen og kobler sammen brukergrensesnitt og underliggende klasser. Merk at appen deres **skal** bygges etter Model-View-Controller-prinsippet. Det betyr at det skal være et klart skille mellom modell (model), grensesnitt (view), og kontroller (controller) i appen. Her er grensesnittet definert av FXML-filene deres, og modellene er de underliggende klassene. Kontrollerens jobb er å binde sammen grensesnitt og modeller, altså å holde verdiene i grensesnittet oppdatert i forhold til modellene, og å reagere på brukerinput i grensesnittet og kalle passende metoder i modellene.

Detaljerte krav for denne delen:

- Appen skal bestå av minimum to interagerende klasser, i tillegg til `Controller`- og `App`-klassene, og brukergrensesnittet laget i FXML. Merk at det viktigste i dette prosjektet er de underliggende klassende, så det stilles ikke krav om at grensesnittet er komplisert. Gjør det så enkelt dere kan for å realisere funksjonaliteten dere trenger.
- Minimum en av klassene må ha noe funksjonalitet utover ren datalagring, en form for *kalkulasjoner* (i en utvidet betydning av begrepet). Dette kan være funksjonaliteten i et spill, utregning av matematiske uttrykk, kryssjekking av flere datakilder for å finne felles verdier, eller lignende.
- Det skal implementeres korrekt innkapsling og validering for tilstandene til objektene i appen.
- Appen skal organiseres etter Model-View-Controller prinsippet, som beskrevet ovenfor.

3.2 Del 2: Filbehandling

Frivillig innleveringsfrist: 19/3

Her skal dere utvide appen til å kunne lese fra og skrive til filer. Dere må selv bestemme hva som skal lagres, og formatet det skal lagres på. Altså må dere finne en måte å strukturere tilstanden i objektene deres som en tekstfil.

For å gjøre det enklere å bytte filformat på et senere tidspunkt, eller å gjøre det mulig å lagre i forskjellige formater, skal dere lage et grensesnitt (`interface`) med metoder for lesing fra og skriving til fil. Deretter må dere lage minimum en ny klasse (altså ikke en av klassene dere har lagd tidligere i prosjektet) som implementerer grensesnittet og realiserer funksjonaliteten.

Detaljerte krav for denne delen:

- Et grensesnitt som minimum har en metode for lesing fra og en for skriving til fil.
- Minimum en ny klasse (altså ikke en av klassene dere har lagd tidligere i prosjketet) som implementerer grensesnittet. Klassen skal altså lagre (deler av) tilstanden til appen deres til et valgfritt format og lese inn det samme formatet til appen.
- Brukergrensesnittet i appen må utvides med mulighet for å skrive tilstanden til og lese fra fil.

3.3 Del 3: Feilhåndtering

Frivillig innleveringsfrist: 19/3

I de aller fleste programmer vil det være mulighet for at feil skjer under kjøring, uavhengig av om koden er skrevet riktig. Det kan f.eks. være ved ugyldig input fra brukere, filer som ikke eksisterer, manglende internett, eller lignende. Om denne type feil ikke håndteres, vil det enten føre til at appen oppfører seg feil, eller at den krasjer.

I denne delen av prosjektet skal dere utvide appen deres til å håndtere feil på passende måter. For å gjøre dette må dere finne ut:

- Hvor i appen deres det kan oppstå feil.
- Hvordan appen bør respondere på slik feil.

Deretter må dere implementere feilhånderingen dere har funnet ut at er hensiktsmessig.

Detaljerte krav til denne delen:

- Det skal være implementert hensiktsmessig feilhåndtering i alle utsatte deler av appen deres.
  - Om dere har en eksepsjonelt stor app, kan dere, **etter avtale med studass**, slippe unna med å bare feilhåndtere deler av koden.

3.4 Del 4: Testing

Innleveringsfrist: 26/4

Siste del av prosjektet handler om å skrive enhetstester til appen deres. Dere bør teste alle metoder med funksjonalitet utover helt enkle `get-` og `set`-metoder. Dere trenger heller ikke å teste `Controller-` og `App`-klassene deres, da det ikke skal være logikk i disse. Har dere logikk liggende i `Controller`-klassen bør dere flytte dette til en passende underliggende klasse før dere begynner på denne delen.

Når dere skriver testene bør dere først tenke over hvilke deler av koden det er viktig å sjekke at fungerer som den skal. Deretter bør dere prøve å komme på (kombinasjoner av) input-verdier og tilstander som kan føre til uventet oppførsel i disse delene. Slike *edge-cases* bør det skrives tester for, i tillegg til å teste for at funksjonaliteten fungerer med "snillere" verdier. Til slutt er det viktig at testene skal sjekke at koden fungerer, ikke motsatt. Om dere er sikre på at en testen er riktig, men den feiler, så må dere sannsynligvis endre koden i klassen som testes – ikke testen deres.

Detaljerte krav for denne delen:

- Alle relevante deler av koden deres skal enhetstestes.
  - Om dere har en eksepsjonelt stor app, kan dere, **etter avtale med studass**, slippe unna med å bare teste deler av koden.
- Enhetstestene skal skrives i JUnit 5.

4 Innlevering

Fristen for innlevering av prosjektet er **26/4**. Dere skal levere en zip-fil med hele prosjektet deres på Blackboard. Om dere har brukt `git` under arbeidet med prosjektet kan dere gjerne også dele det repoet med studass, men dere **må uansett** levere zip-fila på Blackboard.

Dere må også demonstrere prosjektet for studass **innen samme dato, 26/4**. Under demonstrasjonen vil studass gå gjennom prosjektet i forhold til kravene i denne oppgaveteksten. Merk at studass sannsynligvis vil stille spørsmål om koden, *og kan bestemme hvem som skal svare på spørsmålet dersom dere har jobbet sammen.* Derfor er det viktig at begge gruppemedlemmer forstår hele kodebasen.