Prosjektbeskrivelse

Som en del av øvingsopplegget 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.

Dersom dere har spørsmål, spør på Piazza eller send mail til tdt4100-undass@idi.ntnu.no

Noen nøkkelpunkter:

Innleveringsfrist	11. april
Demonstrasjonsfrist	25. april
Gruppestørrelse	1 eller 2 personer

1. Generell informasjon

Prosjektet kan gjøres alene eller i grupper på to personer. Hvis dere gjennomfører prosjektet som par, forventes det at begge er delaktige i **alle** deler av prosjektet, gjennom konsekvent bruk av parprogrammering. Læringsassistentene vil passe på at begge forstår **hele** kodebasen ved demonstrasjonen.

Husk også at dere vurderes på **egen** kode og ikke bruken av eksterne bibliotek og lignende. Vi vil ikke eksplisitt forby bruk av slikt, men man risikerer å ikke vise måloppnåelse hvis man baserer prosjektet i for stor grad på bruk av dette.

Prosjektet er delt opp i følgende deler:

1. Planlegging av prosjektet
2. Grunnklasser og brukergrensesnitt
3. Lagring og filhåndtering
4. Feilhåndtering
5. Testing
6. Dokumentasjon

Innleveringsfristen på prosjektet er **11/4**. I tillegg oppgir vi en valgfri frist på hver del. Det er ikke krav om at dere leverer inn noe til disse frivillige tidspunktene. Hensikten er å gi en anbefaling til omtrentlig tidsbruk og anbefalt rekkefølge på de ulike delene.

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.

Vi anbefaler å vise en enkel plan til læringsassistenten deres før dere begynner. Dette er ikke en omfattende del, men handler om at læringsassistenten kan hjelpe å se at prosjektet er gjennomførbart og dekker kravene. Når prosjektet skal planlegges er det viktig å tenke på hvordan kravene kan overholdes (se del *2. Krav til sluttprodukt*). I begynnelsen av

prosjektperioden, når prosjektet skal planlegges, har ikke alt dette pensumet vært forelest enda, men kravene er beskrevet i dette dokumentet, og det er mulig å bruke læringsassistent for hjelp og tips slik at man er sikker på at prosjektet dekker disse.

Ved innlevering av prosjektet skal dere laste opp koden deres som en zip-fil til Blackboard innen fristen **11/4**. I tillegg må dere demonstrere prosjektet for læringsassistent innen fristen **25/4**. Vi vil komme tilbake med mer detaljert informasjon vedrørende gjennomføringen av dette senere. Dette dokumentet inneholder detaljer for **alle** delene av prosjektet. Vi anbefaler allikevel å fordele arbeidet med prosjektet utover semesteret. Mange av kravene er knyttet til pensum som ikke er forelest ennå, så det kan være krevende å gjøre de mer avanserte delene tidlig. Les derfor gjennom kravene og gjør de delene dere allerede har gått gjennom først.

Anbefalte perioder å jobbe med prosjektet:

Uke	Fra	Til	Beskrivelse
13	24/3	30/3	Grunnklasser og brukergrensesnitt
14	31/3	6/4	Lagring og filhåndtering
15	7/4	11/4	Testing, feilhåndtering og dokumentasjon

2. Krav til sluttprodukt

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 én 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.
    - Minimum én av klassene må implementere et grensesnitt (interface). Dette kan være et grensesnitt du har laget selv, eller et eksisterende grensesnitt som Iterable eller Comparable.
- 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 i JavaFX, med tilhørende Controller- og App-klasser. Disse klassene teller 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.

Disse kravene er utdypet i del 3, sammen med en mer detaljert beskrivelse av hver del av prosjektet.

I tillegg skal appen og koden dokumenteres. Her skal det redegjøres for hvordan appen fungerer og koden skal dokumenteres. Det skal redegjøres for hvordan de tekniske kravene er tilfredsstilt, og en skal benytte seg av begreper og terminologi som er relevant for emnet.

2.1 Eksempler

Det er laget et eksempelprosjekt dere kan bruke som inspirasjon i arbeidet deres. For å få opp eksempelet i VSCode må dere først lage et eget repo i git.

2.2 Forslag til 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 læringsassistent før dere begynner, for å dobbeltsjekke at temaet er passende. Pass også på å ikke velge et av prosjektene listet under "*2.3 Ikke tillatte prosjekter*". Forslag til prosjekter:

- Rutenettbasert spill som for eksempel 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 læringsassistenten 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, men sørg for å ha en viss mengde brukerinteraksjon (det holder ikke med f.eks. bare én knapp som styrer spillet). 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 booket.

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 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.

Ikke tillatte prosjekter:

- TicTacToe
- Enkel kalkulator
- Eksempelprosjekter i faget som TodoList, Snakebird, Wordle og ExpenseTracker
- Andre prosjekter som gås gjennom i forelesninger/øvingsforelesninger i faget

Det er heller ikke lov å levere tidligere vurderte prosjekter, verken egne eller andres. Prosjektet skal gjøres som en del av dette emnet, i inneværende semester.

2.4 Hvor skal koden skrives?

På GitHub ligger det en mal dere kan bruke for å sette opp en egen kodebase. Beskrivelsen på hvordan dette kan gjøres ligger på Blackboard. Denne malen inneholder prosjektstruktur slik at dere raskere kan komme i gang med deres eget prosjekt. Vi oppfordrer dere til å ta i bruk denne kodebasen for å gjøre prosessen enklere for dere selv, men det er ikke et krav. Dersom dere jobber i par og ønsker å bruke malen, oppretter ett av medlemmene en kodebase og så jobber begge to opp mot dette. Vi anbefaler også å legge til din/deres læringsassistent som medlem i kodebasen slik at veiledning kan gå raskere og enklere.

3. Del-beskrivelser

3.0 Del 0: Planlegging av prosjektet

Vi anbefaler at dere legger inn litt tid til å planlegge appen før dere begynner. Dette er en liten del, men kan brukes for å sikre at prosjektet dere lager er gjennomførbart. Det kan være lurt å fortelle læringsassistenten deres hva slags app dere har tenkt å lage og hvordan dere tenker kravene skal oppfylles.

Planen kan for eksempel inneholde noe i denne duren:

- *Beskrivelse av appen*: Kort om spillet/appen, hva skal den gjøre? Hva er målet med spillet?

- *Grunnklasser*: Fortelle kort hva de to (minimum to, kan ha flere) grunnklassene skal inneholde, og hvilken klasse som skal ha noen form for kalkulasjoner eller annen logikk.

- *Filbehandling*: Kort om hvilken informasjon som skal leses fra og skrives til fil.

- *Testing*: Kort om hva som skal testes i appen. Merk: I den delen av prosjektet som går på testing, krever vi at den viktigste funksjonaliteten dekkes av tester, og ønsker derfor at dere tenker over dette allerede nå.

3.1 Del 1: Grunnklasser og brukergrensesnitt

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.

Minst én av klassene skal implementere et grensesnitt (interface). Dette kan være noe så enkelt som å lage et grensesnitt som gjenspeiler klassens metoder og kun blir implementert av den klassen. Det kan også være å implementere et av de eksisterende grensesnittene i Java, som Iterable eller Comparable.

I tillegg til grunnklassene skal dere lage et brukergrensesnitt i JavaFX. Vi anbefaler at dere tenker litt over dette grensesnittet, gjerne skissér det på forhånd for å se at det virker greit å bruke. Figma er et godt digitalt verktøy som kan brukes for å skissere grensesnittet, men dette er ikke et krav. 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 ifra det i dette faget).

Dere skal lage Controller- og App-klasser som starter appen og kobler sammen brukergrensesnitt og underliggende klasser. Vi ønsker at dere bygger prosjektet etter Model-View-Controller-prinsippet. Det betyr at det skal være et skille mellom modell (model), brukergrensesnitt (view), og kontroller (controller) i appen. Her er brukergrensesnittet definert av JavaFX-elementene, og modellene er de underliggende klassene. Kontrollerens jobb er å binde sammen brukergrensesnitt og modeller, altså å holde verdiene i brukergrensesnittet oppdatert i forhold til modellene, og å reagere på brukerinput i brukergrensesnittet og kalle passende metoder i modellene. (Eksempel på dette prinsippet her: https://www.ntnu.no/wiki/display/tdt4100/Innkapsling+-+Kalkulator-oppgave) En kontroller trenger for øvrig ikke være helt tom når det gjelder tilstand. Et eksempel på noe som kunne være lovlig er at modellen inneholder en Car-klasse, mens kontrolleren håndterer en liste med slike Car-objekter som en så kan velge mellom.

Detaljerte krav for denne delen:

- Appen skal ha et brukergrensesnitt i JavaFX. Vi anbefaler å bruke FXML for å definere brukergrensesnittet. Merk at det viktigste i dette prosjektet er de underliggende klassende, så det stilles ikke krav om at brukergrensesnittet er komplisert. Gjør det så enkelt dere kan for å realisere funksjonaliteten dere trenger.

- Appen skal bestå av minimum to interagerende klasser, i tillegg til Controller- og App-klassene.

- 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. Det meste av koden som brukes for å håndtere tilstand skal befinne seg i modellen. En vanlig feil er å legge for mye logikk i kontrolleren. Det er derimot ikke et krav at kontrolleren er strippet helt for tilstand – se eksempel like før denne punktlisten.

3.2 Del 2: Filbehandling

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. Dere må lage minimum én ny klasse (altså **ikke** en av klassene dere har lagd tidligere i prosjektet) som håndterer lagringen.

Detaljerte krav for denne delen:

- Minimum én ny klasse (altså **ikke** en av klassene dere har lagd tidligere i prosjektet) som håndterer skriving til og lesing fra fil. 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.
- Klassen kan ikke bruke innebygde funksjoner eller pakker som serialiserer objekter direkte, som ObjectInputStream, ObjectOutputStream eller GSON. Dere må altså programmere overgangen fra objekt til tekst og tilbake selv, om dere skal lagre objekter.

3.3 Del 3: Feilhåndtering

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: (a) Hvor i appen deres det kan oppstå feil. (b) Hvordan appen bør håndtere disse feilene. Deretter må dere implementere feilhåndteringen dere har funnet ut at er hensiktsmessig.

Hensiktsmessig feilhåndtering betyr ikke bare at dere skal forhindre appen fra å krasje. Det skal foregå på en brukervennlig måte. Eksempler på hensiktsmessig feilhåndtering er:

- En bruker skriver inn bokstaver i et felt som forventer tall, hun får da en melding om at det er feil, enten med en popup, eller rød skrift under feltet. (Brukeren blir varslet)

- Programmet prøver å laste inn forrige økt fra en fil, men det er første gang programmet blir kjørt. Programmet starter da en ny, fersk økt. (Brukeren blir ikke varslet, men det blir håndtert på en måte som gir mening)

Eksempler på lite hensiktsmessig håndtering:

- Programmet krasjer. (Feil som ikke blir håndtert)

- Brukeren trykker på lagreknappen. Lagringen feiler, men brukeren får ikke beskjed. (Mangel på varsling)

- Brukeren skriver tekst i et felt som forventer et tall, dette blir tolket som “0”. (Håndtering som ikke gir mening for brukeren)

Detaljerte krav til denne delen:

- Det skal være implementert hensiktsmessig feilhåndtering i alle utsatte deler av appen deres.

3.4 Del 4: Testing

Siste del av prosjektet handler om å skrive enhetstester til appen deres. Dere bør teste metoder med funksjonalitet utover helt enkle get- og set-metoder. Dere trenger ikke å teste Controller- og App-klassene deres, da det skal være minimalt med logikk i disse. Har dere for mye logikk liggende i Controller-klassen bør dere flytte dette til en passende underliggende klasse i *modellen* 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. Typiske eksempel er å teste for input fra tomme felt, eller strenger det en forventer tall. Slike edge-cases bør det skrives tester for, i tillegg til å teste for at funksjonaliteten fungerer med “typiske” verdier. Til slutt er det viktig at testene skal sjekke at koden gjør det den er ment å skulle gjøre. Du skal ikke skrive tester som får den koden du har laget til å passere – med feil og alt. Hvis dere er sikre på at testen er riktig, men den feiler, så må dere sannsynligvis endre koden i klassen som testes – ikke testen i seg selv.

Detaljerte krav for denne delen:

- Lag *minst* 6 enhetstester, der minst 1 skal teste fillagring.

- Dokumenter valg (se neste avsnitt)

- Enhetstestene skal skrives i JUnit 5.

3.5 Del 5: Dokumentasjon

Som en del av prosjektet skal dere også levere en dokumentasjon til applikasjonen dere har laget. Et mål med denne dokumentasjonen er at dere skal få mulighet til å vise forståelse for konsepter og mekanismer i objektorientert programmering.

Dersom dere har jobbet i par med prosjektet, skal dere også levere dokumentasjonen sammen.

Dokumentasjonen kan være i form av en PDF- eller en Markdown-fil (.md) og skal inneholde følgende deler:

1. Beskrivelse av appen
2. Diagram
3. Spørsmål

Beskrivelse av appen

Her skal dere gi en kort beskrivelse av appen dere har laget og hva den gjør.

Krav: 150-200 ord

Diagram

Her skal dere vise et bilde av et diagram. Dere kan selv velge blant de fire diagramtypene i pensum; klassediagram, objektdiagram, objekttilstandsdiagram, og sekvensdiagram. Diagrammet skal vise en **interessant/viktig del av appen**. Dersom dere gjør noen antagelser som er viktige for å forstå diagrammet deres, skal disse også noteres ned her.

Krav: Kun ett diagram.

Spørsmål

I denne delen skal dere reflektere rundt egne valg, og vise forståelsen dere har for objektorientert programmering og pensum i emnet. Dere skal besvare følgende spørsmål:

1. Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For eksempel bruk av arv, interface, delegering osv.)
2. Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?
3. Hvordan forholder koden deres seg til Model-View-Controller-prinsippet? (Merk: det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller standarder. Det er mulig (og bra) å reflektere rundt svakheter i egen kode)
4. Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke, hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at dere skal reflektere rundt egen bruk av tester)

Krav: 500-800 ord

4. Innlevering

Fristen for innlevering av prosjektet er **11/4**. Dere skal levere en zip-fil med hele prosjektet deres på Blackboard. Nærmere informasjon om innlevering hvordan dette gjennomføres kommer når fristen nærmer seg.

Dere må også demonstrere prosjektet for en læringsassistent innen **25/4**. Under demonstrasjonen vil læringsassistenten gå gjennom prosjektet i lys av kravene i denne oppgaveteksten. Merk at læringsassistenten vil stille spørsmål om koden underveis, og kan bestemme hvem som skal svare på spørsmålet dersom dere har jobbet sammen. Fagets læringsmål krever at begge gruppemedlemmer forstår hele kodebasen.

NB! Dersom dere jobber i par er det veldig viktig at begge to leverer! Da skal dere også vise frem prosjektet samtidig. Hvis dere ikke er på samme Læringsassistentgruppe viser dere frem prosjektet til én av læringsassistentne, pass på at begge fortsatt levere prosjektet på blackboard.