NTNU
Kunnskap for en bedre verden

Mappeoppgave
IDATx1003 Programmering 1 - Del 3

Høst 2023

Mappe del 3 (av 3)

Del 3 av mappen bygger videre på Del 1 og 2.

Basert på tilbakemeldingen du har fått på del 2: gjennomfør nødvendig **refaktorisering (eng: refactoring)** av koden din fra del 2.

Denne delen gjelder resten av mappe-perioden frem til innlevering i uke 50. Dere får tilbud om tilbakemelding på Mappen Del 3 i uke 47, etter dette er dere på egen hånd.

I denne siste delen av Mappeoppgaven skal du **ferdigstille** applikasjonen din og **rapporten**. I tillegg skal du nå **sette ditt eget preg på applikasjonen**:

- Brukergrensesnitt: Du skal implementere et komplett. Du må gjerne tenke alternativt her. F.eks, er en meny beste løsning, eller er kommandoer fra kommandolinjen et bedre alternativ (tilsvarende "World-of-zuul"-prosjektet)? Grafisk brukergrensesnitt er pensum først i Programmering 2, så det gis ingen ekstrapoeng dersom du velger å implementere et grafisk brukergrensesnitt.
  Husk at her er det viktigere at du vektlegger *brukervennlighet* og et *robust grensesnitt*, enn et fancy (grafisk) brukergrensesnitt. Anbefales at du holder deg til pensum i emnet 😉
- Med utgangspunkt i opprinnelig kravspesifikasjon, hvilke endringer/forbedringer ville du ha gjort for at applikasjonen skal bli enda mer nyttig og brukervennlig for brukeren? Her har du lov til å avvike noe fra opprinnelig kravspesifikasjon og tilføre dine tanker og ideer. Løsningen din må selvsagt fortsatt oppfylle de grunnleggende funksjonelle brukerkrav, men du står fritt til å endre på designet av klasser, valg av datatyper om du mener forslagene oppgitt i oppgaveteksten ikke er gunstige, og legge til ny funksjonalitet.
- Med utgangspunkt i designprinsippene **coupling** og **cohesion**, hvilke eventuelle endringer vil du gjøre i ditt design? Kan det være aktuelt å introdusere flere klasser for å oppfylle cohesion-prinsippet bedre, eventuelt innføre flere metoder (delegere)?
- Hvordan møter din løsning prinsippet om **lagdelt arkitektur**?
- Beskriv i rapporten hva du har foreslått av endringer og hvordan du har valgt å implementere disse, og hvorfor disse endringene gir et bedre design i trå med designprinsippene vi har lært i emnet.

Hvilke læringsmål vi ønsker å teste i del 3

I tillegg til at du med mappen skal kunne demonstrere samtlige læringsmål i emnebeskrivelsen, vil vi i del 3 ha spesielt søkelys på:

- evnen til å levere et selvstendig arbeid
- evnen til å re-designe (refaktorere) egen løsning uten å tilføre nye feil
- evnen til å skrive fail-safe/robust kode
- evnen til å skrive kode som følger en etablert kodestil (Google), og som har god lesbarhet og er dokumentert i henhold til anbefalte industristandarder (JavaDoc)
- evnen til å implementere et robust design etter prinsipper som *modularisering*, *coupling*, *cohesion*, *responsibility driven design*
- brukervennlighet/god brukerinteraksjon

Krav til del 3

Krav til Programmet/koden

Følgende krav gjelder (fortsatt) til denne delen av oppgaven:

- Koden skal følge en bestemt ***kodestil*** (fortrinnsvis **Google** sin stil).
- Kodestilen **skal** verifiseres med **CheckStyle**-plugin (for IntelliJ, VSCode osv) og vise ingen/minimalt med regelbrudd ved levering.
- **Klassene** samt alle **metoder** og **variabler** (felt, parametere, lokale variabler) **skal** ha gode, beskrivende navn som tydelig gjenspeiler hvilken tjeneste en metode tilbyr, eller hvilken type verdi variablene representerer/holder på.
- Alle navn på klasser, metoder og variabler **skal** være på **engelsk**.
- Koden skal være dokumentert (på engelsk) iht standarden for JavaDoc (se hvordan JDK'en er dokumentert, for inspirasjon og som referanse).

Du velger selv hvilken IDE (utviklingsverktøy) du vil bruke på prosjektet (IntelliJ, VS Code, Eclipse etc), men prosjektet skal være satt opp som et **Maven-prosjekt**.

Git og GitHub/GitLab

Prosjektet ditt **skal** ha både et lokalt repository, og være koblet til et sentralt repository på GitHub eller GitLab.

Underveis i prosjektet skal du:

- Foreta hyppige **commits** til lokalt repository med gode commit-meldinger, samt **push** til sentralt repository.
- Bruke **tagger** for å markere "release"-versjoner.

Prosjektet skal til slutt pushes i sin helhet til sentralt repository (GitLab/GitHub), i tillegg til å levers som ZIP-fil i Inspera.

Krav til rapport

Rapporten fra del 2 videreføres med tilbakemeldingene du fikk fra LA/Faglærer.

Rapporten skal være **maks 2500 ord**!

Til samtalen for tilbakemelding på del 3:

- bør rapporten i størst mulig grad ferdigstilles. Kapitlene resultat og drøfting skal nærme seg ferdigstillelse, tilstrekkelig til at det er mulig å gi tilbakemelding.
- det er viktig at dere dokumenterer opprinnelige funksjoner/klasser contra refaktorerte funksjoner/klasser, og ikke minst hvorfor du valgte å refaktorere.

Noen kommentarer til rapportmalen

Rapportmalen dere har fått utlevert, er egentlig en mal som også benyttes til Bachelor oppgaver 3. året. Målet med å gi denne allerede i 1. semester, er at dere skal bli vant til å tenke **akademisk rapport** fra starten av, slik at det å skrive en slik rapport forhåpentligvis blir enklere for dere når dere kommer til Bachelor-oppgaven.

Alle akademiske/vitenskapelige rapporter følger omtrent samme mønster, så også denne malen:

1. *Innledning*: Her skriver dere kort om prosjektet som rapporten omhandler.
2. *Bakgrunn* – teoretisk grunnlag: All utvikling er basert på noen teorier og bakgrunnsinformasjon (f.eks. fra andre publikasjoner eller kilder/bøker). Her skal dere kort beskrive de **teoriene** som dere har benyttet når dere har løst mappen, og henvise til **kildene** der dere har teoriene ifra (bøker, nettressurser osv). IKKE skriv om teorier her som der IKKE har benyttet, og ikke omtaler senere i rapporten!
3. *Metode-design*: Her beskriver dere kort hvilke **verktøy** dere har benyttet i prosjektet (IDE'er, osv), samt en kort beskrivelse av **arbeidsmetoden** dere har jobbet etter.
4. *Resultat*: Dette er det viktigste og største kapitlet i rapporten. Her skal dere beskrive den endelige løsninge dere har kommet frem til. Bruk UML diagrammer som use-case diagramm(er), klassediagramm(er), sekvensdiagramm(er) osv. Beskriv kort hver klasse i forhold til ansvar/rolle (ikke gå i detalj på alle metoder og felt). Begrunn ditt designvalg med referanse til **teoriene** du har beskrevet i teoridelen.
   Ikke drøft løsningen din i dette kapittelet, det kommer i neste kapittel☺
5. *Drøfting*: Her skal du nå se tilbake på din løsning og analysere og drøfte både den og de valg du har tatt i forhold til teoriene og metodene du har anvendt (og beskrevet i teori- og metode kapittelet.) Tenk som følger: «Det at jeg valgte å designe min løsning i henhold til teoriene om.....ved å bruke verktøy og metoder som....har medført at min løsning har endt opp å bli enkel å vedlikeholde, lett å utvide.....fordi...»
6. *Konklusjon*: En kort oppsummering av prosjektet som helhet. Maks en halv side.

Generelt:

- Pass på den **røde tråden** gjennom kapitlene; har du beskrevet en teori i teoridelen, så sørg for å henvis til denne i resultat- og/eller diskusjonsdelen av rapporten. Teorier som ikke henvises til, skal ikke stå i rapporten heller.
- Terminologier osv: Dersom du ikke har benyttet deg av forkortelser/termer i rapproten, dropp kapittelet ☺

Bruk av AI-verktøy som GitHub CoPilot og ChatGPT

Dersom du har benyttet deg av AI-verktøy som støtte/hjelp til å løse mappeoppgaven, **skal** dette dokumenteres tydelig **både** i koden (ved kommentarer) **og** i rapporten.

I rapporten skal du beskrive:

- Hvilke AI-verktøy du har benyttet (ChatGPT, GitHub CoPilot osv).
- Hva du har brukt AI-verktøy til (generere kode, stille spørsmål, hjelp til rapport osv)
- Der du har brukt AI-verktøy for hjelp til kode: dokumenter i rapporten de delene av koden som du har fått hjelp til av AI-verktøy, og beskriv **hvorfor** du mener foreslått kode løser problemet ditt, og hva koden gjør (du må forstå og kunne beskrive hva koden gjør).

Viktige sjekkpunkter

Når du løser oppgaven, bør du dobbeltsjekke følgende:

- Kompilering/bygging og prosjektstruktur:
  - Er prosjektet et Maven-prosjekt med en ryddig og riktig katalogstruktur?
  - Kan prosjektet **kompilere** uten feil (*mvn compile*)?
  - Kan samtlige enhets-tester i prosjektet kjøres uten at de feiler?
- Versjonskontroll med git:
  - Er prosjektet underlagt versjonskontroll med sentralt repository i GitHub eller GitLab?
  - Finnes det flere innsjekkinger (commits) ?
  - 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 testdekningen god nok?
  - Kjører samtlige tester uten feil?
- Er klassene for togavgang, register og applikasjon implementert iht oppgavebeskrivelsen?
- Kodekvalitet:
  - Er koden godt dokumentert iht JavaDoc-standard, og skrevet i henhold til Google sin kodestil. Husk at også get- og set metoder skal dokumenteres ☺? (Tips: bruk CheckStyle)
  - Er koden robust (validering mm)?
  - Har variabler, metoder og klasser beskrivende navn?
- Er klassene gruppert i en logisk pakkestruktur?

Lykke til!