NTNU
Kunnskap for en bedre verden

Mappeoppgave
IDATx1003 Programmering 1 - Del 2

Høst 2024

Dette dokumentet beskriver Del 2 av 3 av Mappe-prosjektet i IDATx1003.

Innholdsfortegnelse

Noen avklaringer fra Del 1

Her følger noen oppklaringer/avklaringer fra Del 1 etter innkomne kommentarer/spørsmål fra dere studenter.

Nivå 1 og 2

Som du sikkert husker fra prosjektbeskrivelsen (i Del 1), definerte vi *2 nivåer* på oppgaven:

- Nivå 1: Minimum Viable Product (MVP) – matvarer og kjøleskap/matlager
- Nivå 2: Utvidet funksjonalitet – Matoppskrift og kokebok

For å oppnå topp karakter (A) på mappen må du ha implementert begge nivåene. Har du implementert Nivå 1 og ikke Nivå 2 er du i utgangspunktet typisk «midt på treet», rundt karakteren C. Husk allikevel at det ikke er implementert funksjonalitet alene som bestemmer karakteren (se sensorveiledning).

Sjekkpunkter fra Del 1

Fikk du gjort følgende i Del 1?:

1. Satt opp prosjektet riktig som nevnt på Blackboard kunngjøringer, på et egnet sted på din harddisk slik at du har full kontroll på hvor prosjektet befinner seg på din datamaskin?
2. Lagt prosjektet ditt til **versjonskontroll** på GitHub/GitLab via GitHub Classroom?
3. Implementert **entitetsklassen** for matvare/ingrediens med alle felt, gode konstruktør(er), aksessor- og eventuelle mutator-metoder? Og med god **dokumentasjon** av både klasse- og **samtlige** public metoder iht. JavaDoc standarden? M.a.o. gir CheckStyle feilmeldinger?
4. Opprettet **enhetstest-klasse** for entitetsklassen, der du har laget gode **positive- og negative tester**? Husk at testdekning (test coverage) på 100% IKKE er et kvalitetsmål for testene dine overhodet. Det er kun en eventuell verifisering for å hjelpe deg til å avdekke hvor stor del av kodebasen din som blir testet av testene dine (hvilket også er viktig), men den sier **ingenting** om hvor gode testene er (kvalitet).
5. Gjennomført tilbakemeldings-samtale med faglærer eller læringsassistent?

Hvis ikke: IKKE gå videre til Del 2 før dette er på plass!

Mappe del 2 (av 3)

Del 2 av mappen bygger videre på Del 1.

Basert på tilbakemeldingen du har fått på del 1: gjennomfør nødvendig **refaktorisering (eng: refactoring)** av koden din fra del 1.

I denne delen av mappen skal du implementere **registeret** som skal holde på alle **matvarene i kjøleskapet/varelageret, (nivå 1)** og **kokeboken med oppskrifter (nivå 2)** samt tilhørende **enhetstest-klasse(r)** som tester registerklassen(e) både med negative og positive tester.

Nivå 1 – Minimum Viable Product (MVP)

Register over matvarer (kjøleskap/matlager)

Implementer en klasse som er ansvarlig for å holde på en samling av **matvarer** med tilhørende funksjonalitet.

Du skal selv:

- Velge navn på klassen som skal representere registeret.
- Vurdere hvilken klasse fra Java SDK du tenker er passende å bruke for å lagre alle matvarene i (ArrayList, HashSet, HashMap osv). Husk å begrunn i **rapporten** hvorfor du valgte nettopp denne klassen fra SDK'en.

Fra listen over **funksjonelle krav** fra kravspesifikasjonen i Del 1, bør register-klassen som *minimum* ha støtte for følgende funksjonalitet:

- En metode for å legge til en matvare i registeret. Dersom det allerede finnes en matvare av samme type som den varen som forsøkes lagt til, skal mengden av varen i kjøleskapet økes tilsvarende mengden som blir forsøkt lagt til.
- En metode for å søke opp en matvare basert på navn.
- En metode som fjerner en *mengde* av en matvare. Blir beholdningen av varen 0 etter fjerning, skal hele varen fjernes fra registeret.
- En metode som returnerer alle varer som har en best-før dato før angitt dato. F.eks. skal bruker kunne be om å få en liste over alle varer som har best før dato før f.eks. 2024-10-20.
- En metode som returnerer alle matvarer som en **sortert liste** av matvarer sortert stigende på navn (alfabetisk). Merk at vi ikke ber om at registeret i seg selv nødvendigvis må være sortert til enhver tid, men at denne metoden skal returnere en sortert samling (eller en iterator til en sortert samling) basert på matvarene i registeret ditt.
  NB! Beskriv i rapporten løsningen du valgte her. Beskriv hvordan du kom frem til løsningen (søkte på nett, ChatGPT, GitHub CoPilot, spurte en venn osv), og forklar med egne ord koden du har implementert (dette for å vise at du forstår koden du har implementert).
  Dersom kode er inspirert av en løsning fra andre (f.eks, nettside) er det viktig at kilde henvises til og valg begrunnes.

Du må gjerne utvide klassen med flere metoder som du tenker er nyttig/nødvendig for å løse oppgaven.

TIPS: For å forenkle utprøving av løsningen din underveis i utviklingen, er det lurt å lage en metode som fyller matlageret/kjøleskapet med f.eks. 5-10 ulike matvarer. Du vurderer selv hvor (i hvilken klasse) det er mest hensiktsmessig å legge denne metoden, og hva metoden bør hete.

Enhetstest av registerklassen for kjøleskap

Opprett enhets-tester for å teste registerklassen. Husk at det er smart å planlegge på forhånd **hva** du skal teste i registerklassen og **hvordan**, og spesielt hvordan du skal teste for å få utført **negativ** testing. Dette er det lurt å skrive i JavaDoc'en til testklassen **FØR** du begynner å kode testene.

Husk både **positiv(e)** og **negativ(e)** tester. Kontroller gjerne dekningsgrad (test-coverage) for å sjekke at mest mulig av koden i register-klassen dekkes av testene. Bare husk at 100% dekning ikke er ensbetydende med at testene dine er gode.

Oppdatere applikasjonsklassen

Oppdater **init()** og **start()** metodene i klassen som representerer applikasjonen (og på sikt vil representere det tekstbaserte brukergrensesnittet):

- **init()**: Her legger du inn all kode som er nødvendig for å **initialisere** applikasjonen ved oppstart, som f.eks. å opprette instansen av register-klassen.
- **start()**: Oppdater denne metoden til å ta i bruk den nye register klassen. Opprett 3-4 matvarer i kode som du legger til i registeret. Test deretter noe av funksjonaliteten til registeret. Du kan f.eks. allerede nå tenke på å opprette en metode som skriver ut **oversikt over matvarer** til konsollet basert på registrerte matvarer. Det er lurt å lage en egen metode for å skrive ut denne oversikten. Vent med å implementere full meny med input fra brukeren. Dette kommer i del 3.

Nivå 2: Utvidet funksjonalitet – Matoppskrift og kokebok

Matoppskrift

Implementer en klasse som representerer en **oppskrift**. En oppskrift skal minimum inneholde følgende informasjon:

- Navn på rett
- En kort beskrivelse av retten
- Fremgangsmåte (et tekstfelt)
- En liste over matvarer/ingredienser som inngår for å lage retten. Her må du selv velge hviken klasse fra JDK'en du tenker passer (ArrayList, HashMap, HashSet osv). Begrunn valget i rapporten.
- Antall personer oppskriften er myntet på (ofte er 4 standard)

Kokebok

Opprett en klasse som representerer en **Kokebok** (engelsk: *Cookbook*). Du må selv bestemme hvilken klasse fra Java biblioteket du vil bruke (ArrayList, HashMap osv) (husk å begrunn valget i rapporten din).

Du skal selv tenke ut og implementere fornuftige metoder i en slik klasse for å kunne oppfylle brukerkravene under (fra prosjektbeskrivelsen).

Brukerkrav

Som bruker må jeg kunne:

- Opprette en **matoppskrift** (for en matrett). En matoppskrift (engelsk: *Recipe*) består typisk av følgende elementer: Et **navn** på oppskriften, en kort **beskrivelse** av hva oppskriften lager, en **fremgangsmåte** og en liste av **ingredienser** (inkludert mengde).
- Sjekke om kjøleskapet inneholder nok varer/ingredienser til å lage en bestemt matrett.
- Legge oppskriften inn i en **kokebok** for senere bruk.
- Få forslag til hvilke retter som kan lages fra rettene i kokeboken med varene/ingrediensene som finnes i kjøleskapet. (Avansert!)

NB! Det er ikke sikkert at alle disse funksjonene lar seg løse bare med klassen for oppskrift og klassen for kokebok. I så fall må du vurdere om du trenger flere klasser for å løse kravene (NB! Vi tenker her på forretningslogikken, ikke brukergrensesnittet).

Enhetstester av klassen for kokebok og oppskrift

Opprett enhets-tester for å teste klassen for kokebok og klassen for oppskrift.

Husk både **positiv(e)** og **negativ(e)** tester. Kontroller gjerne dekningsgrad (test-coverage) for å sjekke at mest mulig av koden i register-klassen dekkes av testene. Bare husk at 100% dekning ikke er ensbetydende med at testene dine er gode.

Oppdatere applikasjonsklassen

Oppdater **init()** og **start()** metodene i klassen som representerer applikasjonen (og på sikt vil representere det tekstbaserte brukergrensesnittet):

- **init()**: Her legger du inn all kode som er nødvendig for å **initialisere** applikasjonen ved oppstart, som f.eks. å opprette instansen av kokebok-klassen.
- **start()**: Oppdater denne metoden til å ta i bruk de nye klassene for *oppskrift* og *kokebok*. Opprett et par oppskrifter i kode som du legger til i kokeboken. Test deretter noe av funksjonaliteten til registeret. Vent med å implementere full meny med input fra brukeren. Dette kommer i del 3.

Viktige sjekkpunkter

Når du løser oppgaven, bør du dobbeltsjekke følgende:

- Versjonskontroll med git:
  - Er prosjektet underlagt versjonskontroll med sentralt repository i GitHub, og har du holdt GitHub jevnlig oppdatert gjennom ***git push***?
  - 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 matlager/kjøleskap og eventuelt oppskrift og kokebok implementert iht oppgavebeskrivelsen?
- Kodekvalitet:
  - Er koden godt dokumentert iht JavaDoc-standard, og skrevet i henhold til Google sin kodestil? (Tips: bruk CheckStyle)
  - Er koden robust (validering mm)?
  - Har variabler, metoder og klasser beskrivende navn?
- Er klassene gruppert i en logisk pakkestruktur?

Nyttig referanser

- Kodekvalitet og beste praksis: https://www.ntnu.no/wiki/display/idatx1001/Kodekvalitet+og+beste+praksis
- Kodestilsjekk med CheckStyle og SonarLint for IntelliJ: https://www.ntnu.no/wiki/display/idatx1001/CheckStyle+and+SonarLint+for+IntelliJ
- Kildekodekontroll med GitHub/GitLab: https://www.ntnu.no/wiki/pages/viewpage.action?pageId=235998273
- JUnit-testing og Maven: https://www.ntnu.no/wiki/pages/viewpage.action?pageId=240747880