STUDENTRAPPORT NTNU

NTNU | Fakultet for informasjons-teknologi og elektroteknikk
Institutt for datateknologi og informatikk

IDATT1001 Programmering 1

Rapport for mappevurdering

Kandidatnummer 10114

Innhold

1 SAMMENDRAG

Rapportens hensikt er å dokumentere og presentere min løsning av den obligatoriske mappevurderingen gitt i emnet IDAT1001, i form av en oppgave/oppdrag gitt av Smarthus AS. Oppgaven gitt omhandler et program som skal benyttes av en lagerarbeider hos Smarthus AS. Den skal være brukervennlig og intuitiv.

Metodekapitlet ser nærmere på fremgangsmåten som har blitt tatt i bruk både i utarbeiding av kildekoden og rapporten. Programmet og rapporten har blitt utarbeidet gjennom kontinuerlig arbeid fra utgivelsen av den første mappevurderingsdelen utgitt den 4. oktober 2022. Implementasjonen foregikk både på lab med medstudenter og studentassistenter, og også hjemme på kontoret.

I kapitlet om resultater, blir det endelige produktet beskrevet, sett opp mot problemstillingen og lært teori. Resultatene av programmet fremstår som vellykket og gjennomført. Kravene vurderes som tilfredsstilt. Programmet har oppnådd de etterspurte funksjonene som lagerarbeideren trenger for å utføre sitt arbeid på varehuset hos Smarthus AS. Det er blitt benyttet sentrale kodeprinsipper som kobling og kohesjon til å skrive en så god og oversiktlig kode som mulig.

Programmet fremstår som å ha få svakheter sett fra et brukerperspektiv, og virker intuitiv og brukervennlig. Unntakshåndteringer er benyttet for å gi brukeren en så god opplevelse som mulig, uten at programmet avsluttes utilsiktet.

Til slutt ses det nærmere på drøfting rundt den endelige løsningen, og det konkluderes rundt det som prosjektet har gitt av erfaringer.

2 TERMINOLOGI

UML	Unified Modeling Language
WMS	Warehouse Management System
REGEX	Regular Expression
OOP	Object Oriented Programming
IDE	Integrated Development Environment

3 INNLEDNING – PROBLEMSTILLING

3.1 Bakgrunn/Formål og problemstilling

Følgende varehusløsning er utarbeidet som følge av den obligatoriske mappevurderingsformen gitt i emnet IDATT1001. Arbeidsgiveren er «Smarthus AS», leverandør av varer innen bygg-industrien. Rapporten og programmet tar for seg problemstillingen om å utvikle et sofistikert, brukervennlig og pålitelig varehusprogram.

Oppdraget går ut på å utvikle et WMS[1], slik at Smarthus AS kan holde en oversikt over deres varelager. Systemet skal designes for og benyttes av lagerarbeidere. Lagerarbeideren skal ha tilgang til følgende operasjoner: se alle varer, søke etter varer, legge til varer, øke og minke beholdningen til varer, slette varer, og redigere varer.

Kravspesifikasjonen i kortfattet, oversiktlig og kronologisk form følger:

1. Entitetsklassen «Item» skal implementeres, med følgende attributter: varenummer (består av både tall og bokstaver), beskrivelse (tekst som beskriver varer), pris (heltall), merkenavn (tekst som inneholder merke), vekt (kilogram, desimaltall), lengde (desimaltall), høyde (desimaltall), farge (tekst), antall på lager (heltall) og kategori (heltall fra og med 1 til og med 4).
2. Vareregisteret skal implementeres. Vareregisteret skal holde på én eller flere varer (objekter av entitetsklassen). Dette registeret bør fylles opp med et sett med forhåndsbestemte varer, for å kunne teste funksjonaliteten.
3. Et brukergrensesnitt skal utarbeides. Det kan bestå av en tekstbasert meny som har som ansvar å kommunisere med brukeren.
4. Refaktorer der det er nødvendig, for å forbedre løsningen helhetlig.
5. Enum skal benyttes for å håndtere varekategoriene.

3.2 Avgrensninger

Brukeren av programmet, lagerarbeideren, skal ha en god og intuitiv opplevelse gjennom et velutenkt brukergrensesnitt. Det ble ikke satt et krav om at dette brukergrensesnittet skulle forekomme i form av en tekstbasert meny, til tross for at det ble anbefalt grunnet studentenes mangel på kunnskap rundt utarbeiding og implementasjon av grafiske brukergrensesnitt.

Utover dette tillot oppgaven personlige preg og løsninger, dersom alle de grunnleggende kravspesifikasjonene ble oppfylt.

3.3 Begreper/Ordliste

Begrep (Norsk)	Begrep (Engelsk)	Betyding/beskrivelse
Vare	Item	Norsk: Brukes til å beskrive én enkelt enhet. For eksempel: et hvitt vindu med dimensjonene x, y og z fra produsenten «M». Dersom denne enheten var grønn, ville den ha vært registrert som en separat enhet, selv om spesifikasjonene ellers er identiske. English: Used to describe a single kind of unit. For example: a white window with the dimensions x, y and z from the manufacturer "M". If the unit were green, then it would be registered as a separate unit, even though all the other specifications are the same.
Vareregister	Item register	Norsk: Brukes til å beskrive registeret som lagrer alle de registrerte varene i varehuset. Det er dette registeret som kan legge til, fjerne, og endre registrerte varer i varehuset. English: Used to describe the register that stores all the registered items in the warehouse. This register is used to add, remove, and edit registered items in the warehouse.
Varehus	Warehouse	Norsk: Brukes til å beskrive Smarthus AS sitt hovedlager. Engelsk: Used to describe Smarthus AS' main warehouse.

3.4 Rapportens oppbygning

Rapporten i seg selv ble skrevet parallelt med implementasjon av kildekode, for å skape et så tett bånd mellom koden og rapporten som mulig.

Innledningsvis har det blitt sett på prosjektets problemstilling, bakgrunn og formål. Deretter er det opprettet en ordliste med nødvendige, relevante begreper som tas i bruk videre i rapporten. Det ses som høyt nødvendig at leseren gjør seg innforstått med de nedskrevne begrepene, for å minimere sannsynligheten for misforståelse rundt programmet og dets hensikt.

Rapporten videre er strukturert i omtrent den samme rekkefølgen som programmet ble utviklet i.

Først og fremst var det nødvendig å klargjøre teoretiske grunnlag, noe som er det første som beskrives videre i rapporten. Deretter er utviklingsmetode/fremgangsmåte og arbeidsprosess blitt beskrevet. Etter dette har resultatet blitt presentert og vurdert. Videre er det framstilt en drøfting rundt både metode og resultat. Til slutt ses det nærmere på erfaringer og eventuelle angremoment. I slutten av dokumentet er det vedlagt relevante, utarbeidede UML-diagrammer.

4 BAKGRUNN - TEORETISK GRUNNLAG

Det er, ifølge teori undervist i IDATT1001 av forelesere Muhammad Ali Norozi og Surya Kathayat, særdeles viktig at en klasse og dets metoder er simple nok til at en annen person kan lese og forstå koden. Ergo vil det være nødvendig at alle klasser og metoder har gode navn, og kun har ansvar for et særegent område. Det er altså ikke ønskelig at en klasse både skal fungere som for eksempel en klient og et vareregister. I tillegg til at klassene i seg selv skal ha ansvar for et snevert område av programmet, skal også dets metoder ha det.

Metodene som implementeres i klassene skal kun utføre én viss, bestemt oppgave/logikk. Det er altså ikke ønskelig at en metode både søker etter en vare gjennom å se på varenummeret og varebeskrivelsen.

Dersom det nevnte tas hensyn til, vil koden kunne oppleves enklere å lese og utvikle videre av personer som skal arbeide videre med koden/programmet. Dermed kan en argumentere for at en slik standardisering av navngivning og ansvarsområde, vil føre til at programmet fremstår som mer bærekraftig. Dette fordi et program som er enkelt å forstå, vil ha en større sannsynlighet for å overleve videre, fremfor å fjernes og skrives på nytt senere.

Ifølge undervist materiale i IDATT1001, bør en tenke over hvorvidt koden en skriver vil være av typen aggregering eller komposisjon. Disse sier noe om hva slags assosiasjon klassene har ovenfor hverandre. Aggregering vil medføre at klassene kan eksistere selv dersom en annen klasse ikke gjør det. Ved komposisjon vil båndet mellom klassene være sterkere. Her vil for eksempel den ene klassen ikke kunne eksistere dersom den andre ikke gjør det. Disse prinsippene kan bidra til at vi skriver kode som er lett å håndtere for utviklere som skal arbeide videre på programmet.

Forelesningene ved IDATT1001 har også undervist prinsippet rundt å sette objektvariabler som private felt, utilgjengelige utenfor selve klassen. Dette bygger på prinsippet om innkapsling, hvor dataene til objektene er «gjemt» inne i klassen, og ikke direkte offentlig tilgjengelige. Alle objektvariablene er dog tilgjengelige gjennom offentlige tilgangsmetoder som har som oppgave å kun returnere selve dataen. Den returnerte verdien er av en spesifisert datatype, noe som vil si at vi ikke direkte gir tilgang til objektvariablene til for eksempel et klientprogram.

Enkelte objektvariabler kan det være ønskelig at ikke skal endres etter at de er blitt gitt. I disse tilfellene kan vi benytte oss av «final»-nøkkelordet til Java[1][2]. Dette nøkkelordet vil forhindre at objektvariablene kan endres i etterkant av for eksempel et klientprogram. Objektvariabler som blir deklarert med nøkkelordet «final», kalles *immutable*. Andre kan det være nødvendig at skal kunne endres i etterkant, for eksempel prisen på en vare. Dermed vil denne variabelen lønne seg som *mutabel*, altså med mulighet for endring i etterkant av objektets initiering.

Ifølge undervist teori i IDATT1001, vil et godt skrevet program ha løse koblinger og høy kohesjon. Dette vil si at vi ønsker at ulike enheter[1] i koden vår opptrer så selvstendige som mulig. Altså at de ikke er fullstendig avhengig av andre metoder, klasser eller pakker. For å kunne verifisere i hvilken grad koden er skrevet med løse koblinger, kan blant annet klassediagrammer benyttes.

Kohesjon sier oss noe om hvor lesbar koden vår kan være. God og høy kohesjon vil bidra til at koden kan fremstå som enklere å forstå og sette seg inn i. Dette oppnås gjennom å avgrense en enhets ansvarsområde så snevert som mulig. Én enhet skal kun stå for å løse én bestemt oppgave. I tillegg vil dette bidra til at det blir enklere å navngi enhetene.

[1] Klasse, metode eller pakke

I tillegg til forelesninger og relevante, troverdige nettsteder, er det tatt inspirasjon fra svigerfar. Han arbeider fulltid som lagermedarbeider for Ortomedic AS. Her ble det vurdert ulike irritasjonsmoment som han og kollegaene hans har rundt bruken av deres varehus-program. Dette har bidratt til å høyne fokuset rundt å utvikle et brukervennlig og intuitivt program.

5 METODE – DESIGN

Oppgaven ble utgitt i tre deler, fordelt utover flere uker. Dermed var det ikke mulig å få et komplett overblikk over programmets endelige hensikt og bruksområde. Det anvendte integrerte utviklingsmiljøet var IntelliJ IDEA Ultimate Edition, utviklet av JetBrains.
Selve oppgavedelene ble løst på følgende måte:

Del 1: Implementasjon av klassen «Item»
Det ble satt krav til hvilke ti attributter klassen skulle inneholde, og dermed ble disse opprettet. I tillegg ble det naturlig å vurdere hvilke attributter som skulle være endelige, og hvilke som skulle kunne endres i senere tid. Det ble i denne delen gitt krav om at koden skulle følge en bestemt kodestil. Her ble det valgt å benytte seg av kodestil-filen «IDATx1001». Klasse-, metode- og variabelnavn skulle være beskrivende, slik at de gjenspeiler deres funksjon. Disse skulle også være på engelsk.

Del 2: Implementasjon av registerklassen «ItemRegister»
Denne delen innebar å utarbeide en klasse som skulle ha ansvar for et register av alle lagerets varer. I tillegg ble det gitt krav om hvilke metoder/operasjoner denne klassen skulle kunne utføre. Dette ble det naturlig å fremstille i form av en intuitiv og tekstbasert meny, slik at lagerarbeideren enkelt skulle kunne finne frem til de ønskede handlingene.

Del 3: «Refaktorering» og bærekraftighet
Den siste delen setter fokus på å vurdere hvorvidt den skrevne koden vil fungere bærekraftig over tid. Dette gjennom å minimere antallet og graden av nødvendige endringer dersom Smarthus AS for eksempel vil legge til en ny eller fjerne en varekategori. Her var det sentralt at dette ble testet, ved å forsøke å legge til og å fjerne varekategorier. Samtidig ble koden gjennomgått med et kritisk blikk.

Løsningen har blitt utarbeidet gjennom et kontinuerlig arbeid, både på lab med og uten medstudenter, og hjemme. Det ble lagt et ønske om å jobbe med prosjektet i små deler om gangen, slik at en helhetlig oversikt over løsningen kunne ivaretas i så god grad som mulig.

Tilbakemeldinger fra studentassistener har vært sentrale i utviklingen av programmet. Gode, kunnskapsrike og konstruktive tilbakemeldinger har blitt brukt aktivt gjennom hele prosjektet.

Det ble opprettet en uoffisiell og enkel framdriftsplan[a] for prosjektet. Den ble overholdt gjennom å arbeide effektivt i begynnelsen, da timeplanen ellers var relativt ledig. Prosjektplanen fungerte godt som et hjelpemiddel for å se progresjonen rundt både implementasjon av kildekoden, samt utarbeiding av rapporten.

Et annet viktig hjelpemiddel har vært det å aktivt teste programmet og rydde opp i feilmeldinger og hindringer så fort som mulig. Programmet har blitt skrevet bit for bit, og dermed lagt til rette for enklere «debugging» av alt ifra syntaks til generell logikk i koden.

I tillegg har det blitt brukt gode kilder på internett, slik som W3Schools, Stack Overflow og andre lignende nettsteder. Disse har bidratt da ukjente og kompliserte feil dukket opp. Samtidig har det blitt hentet inspirasjon fra disse nettstedene med tanke på bruk av ulike klasser ifra JDK-biblioteket, slik som for eksempel «ArrayList».

Et av kravene til prosjektet var at all kode skal tilfredsstille syntaks- og konvensjonsregler ifra én av regelfilene Google Check, Sun Check og IDATX1001. Her ble sistnevnte valgt, da det virket fornuftig med blant annet å ha den første krøll-parentesen under klasse-/metodenavnene. Feilmeldinger produsert av denne regelfilen har blitt hyppig behandlet og rettet opp i, slik at det ikke skulle hope seg opp med alle ulike typer feil.

UML-diagrammer slik som klassediagram og sekvensdiagram ble benyttet under utviklingen av kildekoden. Dette ble brukt som et referansemiddel dersom det skulle oppstå uklarheter rundt hvordan klassene og metodene skulle samarbeide, der det var nødvendig å framlegge en tydelig oversikt.

6 RESULTATER

Klassenes navn ble valgt på bakgrunn av deres overordnede funksjon. Klientklassen har som formål å både verifisere funksjonaliteten til de andre klassene og kommunisere med brukeren av programmet. Denne oppgaven vil kunne framstå tydelig gjennom klassenavnet «Client». Klassen som skal representere en vare, vil nødvendigvis kunne hete «Item». Registerklassen som har som oppgave å lagre på flere objekter av klassen «Item», vil dermed kunne kalles for «ItemRegister». I tillegg ble det laget en separat fil til den spesielle klassen «enum», som bygger videre på «Enumeration». Denne har som oppgave å lagre de ulike varekategoriene varehuset for øyeblikket lagerfører. Her kan det legges til flere kategorier i etterkant, samt fjerne eldre dersom det skulle vise seg nødvendig.

Videre følger en tabell med oversikt over de ulike klassene og deres formål.

Klasse	**Hensikt/oppgave**
Client	Brukerkommunikasjon. Styrer programflyten gjennom å presentere en meny for brukeren og ta imot input. Alle funksjonene lagerarbeideren trenger for å holde oversikt over varehuset og dets varer, utføres her. Betraktes derfor som programmets «front-end», også ofte kalt *frontend* på norsk.
Item	Kan opprette vare-objekter og behandle disse objektene gjennom tilgangs- og mutatormetoder.
ItemRegister	Lagrer objekter av klassen «Item» i en liste. Denne listen kan brukeren få indirekte tilgang til, og dermed holde oversikt over varehusets registrerte varer. Denne klassen har ansvar for å blant annet legge til og fjerne varer.
ItemCategory	Lagrer på konstantene som representerer de ulike varekategoriene.

Programmet skulle, ifølge kravspesifikasjonen og oppgaveteksten, kunne kommunisere med en bruker gjennom en tekstbasert meny. Brukeren skulle også ha mulighet til å utføre operasjoner slik som å: vise alle registrerte varer, søke etter varer på bakgrunn av varenummer og varebeskrivelser, registrere nye varer, øke og minke antallet av en vare på lager, fjerne en vare, og endre en vares rabatt, pris og beskrivelse.

Den tekstbaserte menyen ble løst gjennom implementasjonen av klientklassen «Client», og benytter seg av den importerte java.util-klassen «Scanner» til å motta input fra brukeren gjennom terminalen i IntelliJ. Her styres programflyten gjennom at brukeren skriver inn den ønskede tjenestens tilhørende tallverdi, og deretter vil den gitte tallverdien gjennomgå en «switch-case»-løkke, der programmet utfører den ønskede tjenesten på bakgrunn av tallverdien. Disse «switch-casene» vil kalle på tilhørende metoder og videre be om input fra brukeren der det trengs.

Vareklassen ble løst gjennom å opprette standardiserte tilgangs- og mutatormetoder, samt en konstruktør som oppretter/initierer et objekt på bakgrunn av de gitte parameterne. Denne klassen kan også behandle varenes attributter slik som rabatt, pris og antallet varehuset har på lager.

Vareregisteret ble løst ved hjelp av den importerte klassen «ArrayList», som fungerer som en dynamisk liste som lagrer på ulike objekter av vareklassen. Denne listen kan vises for å gi lagerarbeideren en oversikt over alle de registrerte varene. Det er denne listen det letes gjennom når en bruker ønsker å søke etter varer. Iterering over de lagrede varene i registeret gjør det enkelt å holde god kontroll på varehusets varer.

Varekategorikonstanene ble opprettet med navn som tilsvarer de ulike varekategoriene oppgaven la til rette for og spesifiserte i oppgaveteksten.

Løsningen i seg selv består av tre hoved-klasser og én mindre enum-klasse. Hovedklassene samarbeider basert på brukerens input. Vare- og vareregisterklassene har som ansvar å returnere de ønskede datatypene tilbake til klientprogrammet. Dermed kan vi si at disse to klassene opptrer som «back-end»-klasser.

Programmet er tiltenkt brukt av en lagerarbeider som har som ansvar å holde oversikt over varehusets varer. Denne brukeren skal ta imot varer ifra ulike leverandører, og legge disse mottatte varene til i programmet gjennom den tilhørende funksjonen for å legge til varer, eller øke et antall dersom varen allerede er registrert i varehuset. Brukeren er tiltenkt som en aktiv bruker av programmet i løpet av en ordinær arbeidsdag. Programmet kan i tillegg benyttes av økonomiansvarlige i Smarthus AS, slik at de ansvarlige kan følge utviklingen av inntekter og kostnader. Disse vil mest sannsynlig benytte seg av funksjonen som viser frem alle de registrerte varene, deres priser og antall på lager.

Videreutvikling av programmet kan være nødvendig dersom Smarthus AS for eksempel ønsker å endre på varehusoppsettet sitt, eller ytterligere funksjonaliteter er ønsket av lagerarbeiderne. Denne forutsetningen ble tatt i betraktning da programmet ble utviklet, gjennom at kildekoden skal være så enkel og intuitiv å forstå som mulig.

Diagrammene har blitt utviklet på bakgrunn av undervist teori i IDATT1001 av forelesere Muhammad Ali Norozi og Surya Kathayat. Klassediagrammet[b] viser de ulike klassenes navn, objektvariabler og metoder, samt forholdet mellom disse klassene.

Det ble ikke vurdert løsninger svært ulike den endelige løsningen. Små ulikheter ble likevel sett opp mot hverandre, og valgt på bakgrunn av hvilken som både vil fungere optimalt med tanke på opplevelsen til lagerarbeideren, og fremstå bærekraftig gjennom å være enkel å forstå, lese og videreutvikle. For eksempel ble det vurdert rundt hvorvidt vare-objektet skulle opprettes/initieres i klientklassen eller vareregisterklassen. Sistnevnte ble valgt, da dette vil føre til at vare-objektet ikke har et livsløp utenfor registerklassen.

Det ble refaktorert deler av metodene som søker etter én eller flere varer på bakgrunn av enten varenummer eller varebeskrivelse. I begynnelsen hadde metodene kun mulighet til å returnere én vare, den varen i registeret som først matchet med det spesifiserte varenummeret eller beskrivelsen. Disse metodene ble refaktorert, slik at logikken deres tillot å legge til flere vare-matcher i en liste, som senere vises til brukeren dersom det oppstår treff i søket.

Koden består av flere ulike «try-catch»-blokker. Disse blokkene har som ansvar å håndtere de ulike unntakene som kan oppstå gjennom for eksempel brukerens input. Dersom en bruker skriver inn en bokstav der det spørres etter et tall, vil disse unntakene fanges i «catch»-blokken, og håndteres manuelt og starte programmet på nytt. Det er implementert egne unntaksmeldinger til de ulike unntakene som kan oppstå når en ny vare skal registreres, slik at brukeren blir klar over hva det var som gikk galt i opprettingen av varen.

Koden er basert på å ha så løse koblinger som mulig, samt ha høy kohesjon. Gjennom å ha klasser og metoder med snevre ansvarsområder og beskrivende navn, vil koden fremstå som enkel å forstå og

lese. Løse koblinger er oppnådd i så god grad som mulig gjennom å ikke ha for mange direkte koblinger i metodene mellom klassene. Metodene i de ulike klassene vil kunne «stå på egne ben».

Koden er dokumentert ved hjelp av Javadoc. Dette er kommentarer med en konvensjon som spesifiserer en viss oppbygning og struktur på kommentarene som det brukes til å dokumentere. En metode sine parametere og returverdier er inkluderte i disse kommentarene. Dokumentasjonen skal være tilfredsstillende beskrivende og gjøre det enklere for en som leser koden å forstå hva de enkelte klassene og metodene gjør.

Det er ikke garantert at koden er idiot-sikker, da det er vanskelig å konkludere rundt om alle mulige ønskede hendelser er testet i programmet. Klientprogrammet skal derimot ikke ha tilgang til å manipulere hverken vare-objekters attributter eller vareregisterets liste direkte. Dette medfører at koden fremstår som sikker ovenfor varehusets register og oversikt. Slik kan de være trygge på at klientprogrammet ikke kan interferere eller påvirke registeret utilsiktet.

Programmet kan gjennom menyen avslutte programmet gjennom at «while»-løkken menyen ligger i vil slutte løkken etter at boolean-variabelen «finished» endres til «true» når brukeren gir en input tilsvarende den tilhørende tallverdien. Etter at denne løkken har kjørt sin siste løkke, vil programmet avsluttes kontrollert. Dette kalles for «graceful termination»[3]. Programmet vil oppleves mer kontrollert og gjennomført ettersom at programmet ikke vil avslutte når et unntak oppstår, men heller kun oppstår når brukeren selv ønsker det.

En brukerveiledning for programmet ligger vedlagt under kapitlet «10 Vedlegg»[1].

7 DRØFTING

Jeg vurderer valget av arbeidsmetode og endelig resultat som god. Det er ulike momenter som kunne ha blitt gjort bedre, gitt bedre tid og mer kunnskap. Det hadde vært ønskelig med en mer grafisk fremstilling av de registrerte varene og programmet generelt, men dette er kunnskap vi ikke innhenter før emnet IDATT2001 til våren. Jeg er fornøyd med brukeropplevelsen av resultatet, og har forsøkt å la familiemedlemmer benytte seg av programmet. Det virker som at de mottar god oversikt over hva programmet tilbyr av tjenester og hva det representerer i en varehus-setting. Til senere vil jeg vurdere å jobbe mer aktivt med mine medstudenter, ettersom at det ikke ble altfor mye tid til dette.

Det oppstod ingen særlige avvik i forhold til utgangspunktet til prosjektet. Hindringene ble håndtert, og arbeidet var svært enkelt å navigere seg gjennom ved bruk av prosjektplanen og oppgaveteksten.

Eventuelle feilkilder som kunne ha oppstått underveis i prosjektet, er mangelen på innhenting av ny kunnskap rundt å optimalisere metoder. Det kunne kanskje ha vært importert og brukt flere klasser fra JDK-biblioteket, for å gjøre koden mer lesbar og effektiv. Samtidig kan dette være en enklere tanke å tenke i retrospekt, da en har innhentet nye og mer effektive kodekunnskaper.

Arbeidsinnsatsen har vært gjennomgående god og effektiv. Det har vært sene kvelder med fokusert arbeid som har ført til ferdigstillingen av kildekoden samt rapport. Arbeidet kunne vært strukturert bedre mot slutten, ettersom at det oppstod et avbrekk før matematikkeksamen. Dette kan det være vanskelig å ha unngått, ettersom at fristen for denne mappevurderingen lå lengre frem i tid. Det måtte rett og slett prioriteres i ukene før matematikkeksamen. Hadde jeg lagt til rette for kortere arbeidsøkter innimellom, kunne jeg kanskje ha kommet enda lengre med programmet. Samtidig mener jeg at alle krav er besvart, så dette kan tyde på at jeg har hatt tilstrekkelig med tid til å gjennomføre oppgaven.

Alt i alt vurderes problemstillingen som godt besvart. Alle de beskrevne kravene i oppgavetekstene er implementert i koden, og programmet fungerer godt og fremstår intuitivt. Det oppstod ingen større, uforutsette utfordringer underveis i prosjektet.

Jeg mener at koden kunne ha hatt en enda løsere kobling mellom klassene og metodene. Det kan virke som at de avhenger litt for mye av hverandre, til tross for at løsningen på dette ikke er helt åpenbar enda. Kodestilen og konvensjoner mener jeg at er fulgt godt opp. Det er blitt benyttet store bokstaver der det skal, og alle navn er beskrivende og tydelige ovenfor dets funksjon/ansvar i programmet.

8 KONKLUSJON – ERFARING

Dersom jeg kunne ha begynt på nytt, ville jeg ha laget en enda tydeligere «skisse» over hvordan jeg så den endelige løsningen for meg. Denne har delvis kommet tydeligere og tydeligere fram mot slutten. I tillegg ville jeg ha forsøkt å lete etter mer effektive måter å behandle vareregisteret på, slik at det kanskje kunne ha fremstått som enda mer lesbart og effektivt.

Programmet har noen få begrensninger i koden, og jeg er ikke kjent med noen spesifikke begrensninger ved brukeropplevelsen. Koden består av flere metoder som kanskje kunne vært delt opp i flere deler, slik at programmet hadde vært enda bedre oppstykket og lesbart. I retrospekt, kunne dette ha blitt gjort dersom tiden hadde strukket til enda litt bedre.

En fremtidig utvikler av dette programmet kunne ha forsøkt å implementere en grafisk utgave, samt stykket opp metodene i klientklassen enda mer. Hen kunne også ha jobbet med å løsne koblingene mellom metodene og klassene mer.

Prosjektet i sin helhet har bidratt til å gi meg et innblikk i hvor stort omfang en slik oppgave kan være. Det kan omtrent arbeides i evigheten, hvor kreativiteten er det eneste som setter grensen. Selve implementasjonen av kildekoden og testingen av programmet har vært svært lærerikt, og jeg har plukket opp på ulike triks og nye måter å løse ting på underveis, både gjennom studentassistener og nettsteder som W3Schools og Stack Overflow.

9 REFERANSER

[1] "Core Java Volume I – Fundamentals", Eleventh edition, av Cay S. Horstmann. ISBN: 9780135166307.

[2] Java-nøkkelordet "final":

https://www.geeksforgeeks.org/final-keyword-in-java/

[3] "Graceful termination":

https://www.techtarget.com/whatis/definition/graceful-shutdown-and-hard-shutdown

10 VEDLEGG

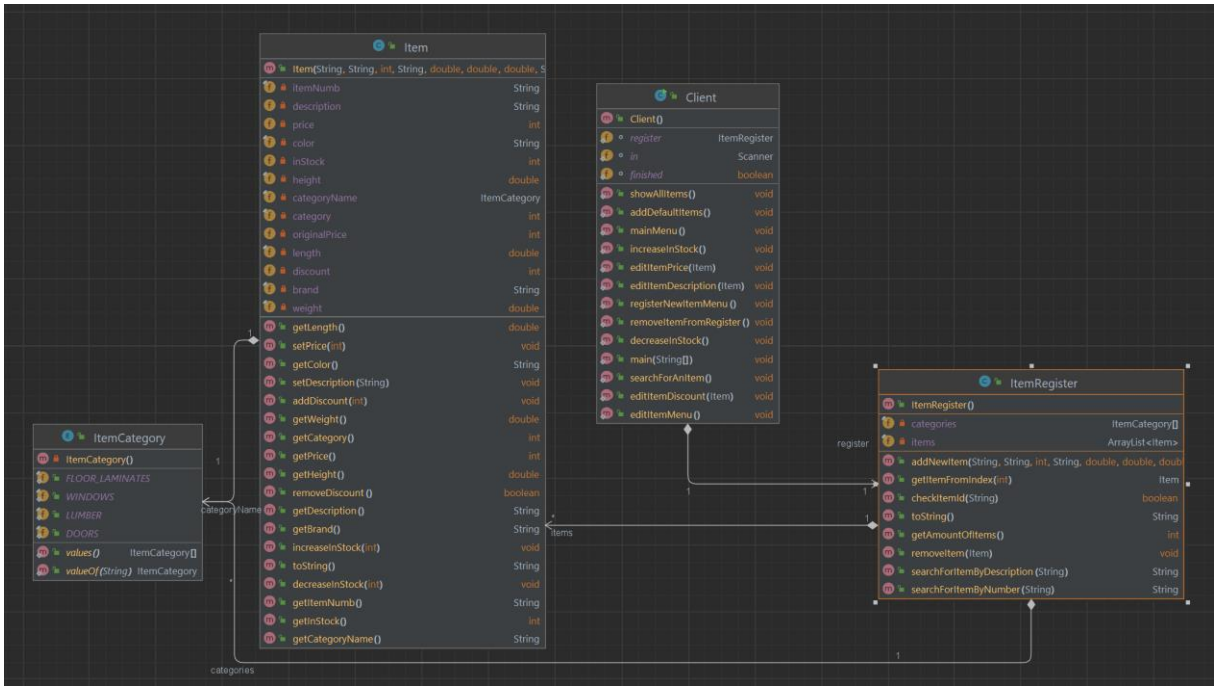
[1] Brukerveiledning

Brukeren anbefales å sette seg inn i programmets hensikt før det tas i bruk.

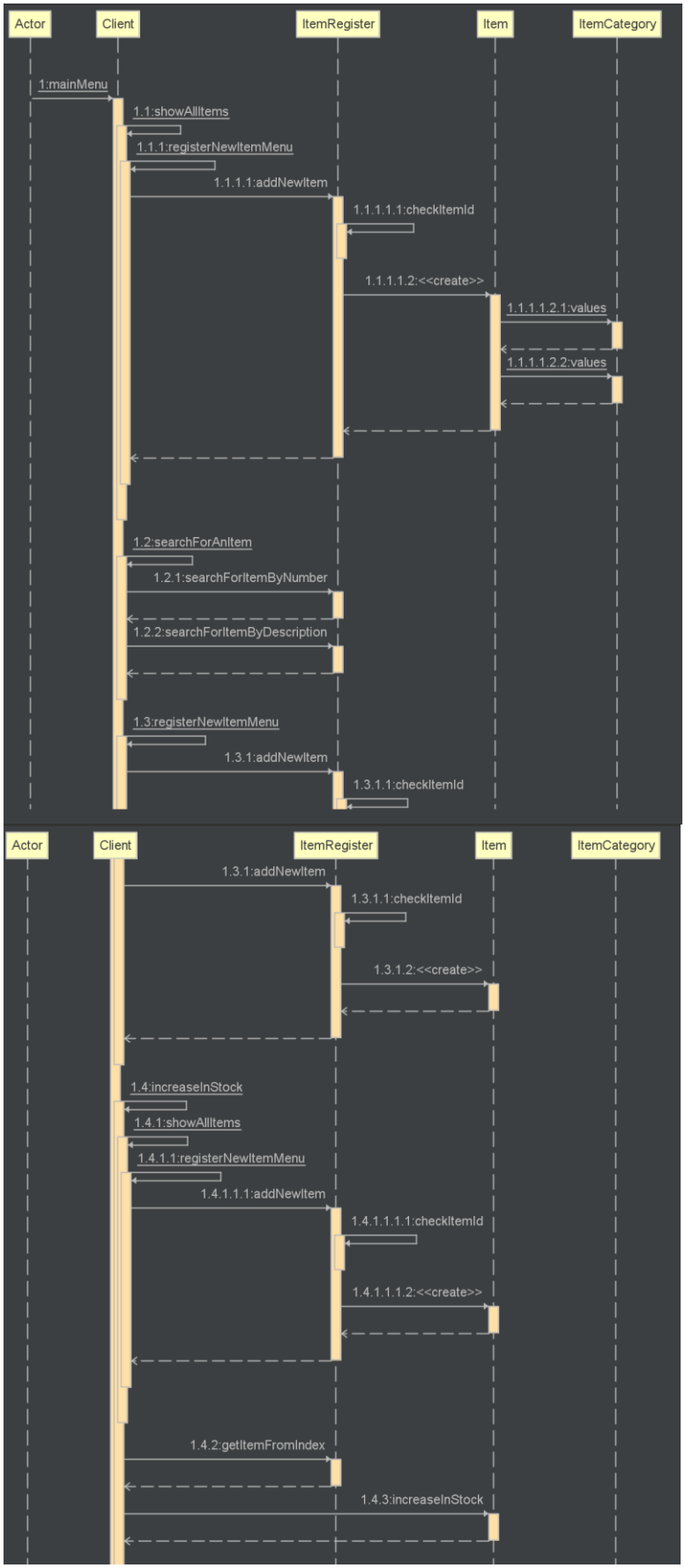
For å begynne, må programmet startes som normalt. Deretter skal brukeren velge én av de presenterte valgene ifra hovedmenyen. Valget tas ved å skrive inn tallet til venstre for den ønskede operasjonen i terminalen.

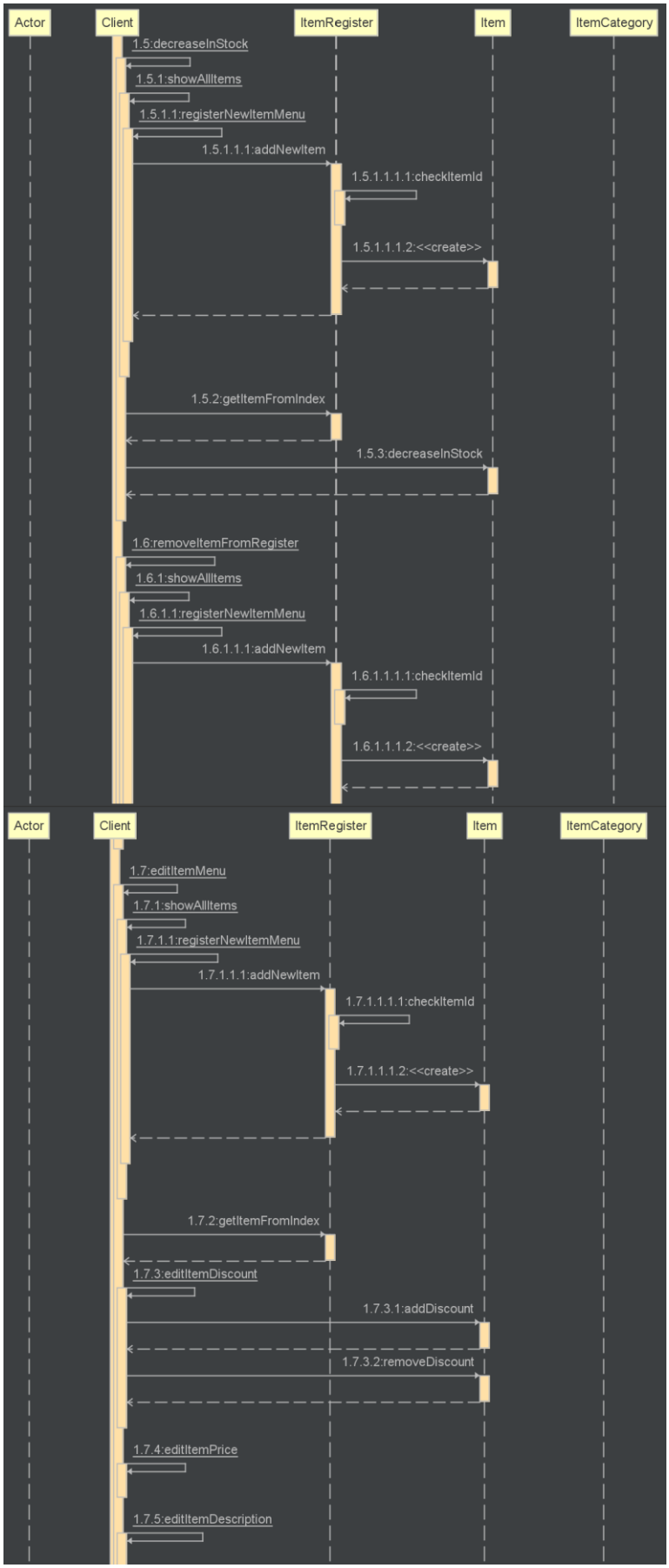
Programmet fungerer altså gjennom at brukeren leser av tallene som ligger til venstre for teksten, og skriver inn dette og trykke på Enter-knappen. Slik kan brukeren navigere seg gjennom menyen, og benytte seg av programmets funksjoner.

[2a] Klassediagram (endelig utgave generert i IntelliJ)



[2b] Sekvensdiagram for hovedmenyen i klientklassen (endelig utgave generert i IntelliJ)




Actor
Client
ItemRegister
Item
ItemCategory
1.5:decreaseInStock
1.5.1:showAllItems
1.5.1.1:registerNewItemMenu
1.5.1.1.1:addNewItem
1.5.1.1.1.1:checkItemId
1.5.1.1.1.2:<<create>>
1.5.2:getItemFromIndex
1.5.3:decreaseInStock
1.6:removeItemFromRegister
1.6.1:showAllItems
1.6.1.1:registerNewItemMenu
1.6.1.1.1:addNewItem
1.6.1.1.1.1:checkItemId
1.6.1.1.1.2:<<create>>
Actor
Client
ItemRegister
Item
ItemCategory
1.7:editItemMenu
1.7.1:showAllItems
1.7.1.1:registerNewItemMenu
1.7.1.1.1:addNewItem
1.7.1.1.1.1:checkItemId
1.7.1.1.1.2:<<create>>
1.7.2:getItemFromIndex
1.7.3:editItemDiscount
1.7.3.1:addDiscount
1.7.3.2:removeDiscount
1.7.4:editItemPrice
1.7.5:editItemDescription

[3] Fremdriftsplan (overordnet)

Fremdriftsplan for IDATT1001 Mappevurdering

Milepæl:	Dato planlagt:	Dato oppnådd:
Implementere entitetsklassen	10.10.2022	06.10.2022
Begynne med rapport del 1	11.10.2022	07.10.2022
Implementere vareregisteret	05.11.2022	30.10.2022
Begynne med rapport del 2	06.11.2022	02.10.2022
Kontroll av løsning så langt	10.11.2022	10.10.2022
Refaktorere og ferdigstille løsning	30.11.2022	06.12.2022
Ferdigstille rapport	10.12.2022	11.12.2022