STUDENTRAPPORT

NTNU

KANDIDATNUMMER(E)/NAVN:			
10021			
DATO: 26.10.2022	FAGKODE: IDATG1001	STUDIUM: DATA INGENIØR BACHELOR	ANT SIDER/BILAG: 14

FAGLÆRER(E):
Kiran Raja

TITTEL:

Rapport: Warehouse Management System for Smarthus AS - Mappevurdering

SAMMENDRAG:

Det skal utvikles en programvare som skal brukes av Smarthus AS. Programvaren skal brukes til å håndtere varelageret til Smarthus AS. Løsningen skal til slutt bestå av et tekstbasert brukergrensesnitt og et register som lagrer informasjon. Denne rapporten handler om utviklingen av dette prosjektet.

Denne oppgaven er en besvarelse utført av student(er) ved NTNU.

INNHOLD

1 SAMMENDRAG 1
2 TERMINOLOGI 1
3 INNLEDNING – PROBLEMSTILLING 1
3.1 Bakgrunn/Formål og problemstilling 1
3.2 Begreper/Ordliste 3
3.3 Rapportens oppbygning 4
4 BAKGRUNN - TEORETISK GRUNNLAG 4
5 METODE – DESIGN 6
6 RESULTATER 8
7 DRØFTING 10
8 KONKLUSJON – ERFARING 11
9 REFERANSER 12

1 SAMMENDRAG

Dette er en rapport som er en del av eksamen i faget IDATG1001, Programmering 1. Det skal utvikles en programvare med kodespråket Java, og det som har blitt lært i løpet av høst semesteret 2022. Programvaren skal brukes til å håndtere varelageret til Smarthus AS (som beskrevet i oppgaven). Dette er et prosjekt som ikke har noen konkret fasit og er dermed ganske åpen. Det finnes mange måter å løse oppgaven på, og det er derfor viktig å løse oppgaven i henhold til oppgavebeskrivelsen på en god måte. Alle vurderinger og valg blir da nødt til å dokumenteres. Denne rapporten er en beskrivelse av både oppgave, og valgte løsninger, samt hva som kunne ha blitt gjort annerledes.

2 TERMINOLOGI

UML	Unified Modeling Language
UP	Unified Process
WMS	Warehouse Management System
UI	User Interface
GUI	Graphical User Interface
AS	Aksjeselskap
dvs	Det vil si
osv	Og så videre

3 INNLEDNING – PROBLEMSTILLING

3.1 Bakgrunn/Formål og problemstilling

Det skal utvikles en programvare som skal brukes til å håndtere varelageret til Smarthus AS. Smarthus AS leverer hovedsakelig varer til bygg-industrien, så typiske varer er laminatgulv, dører og vinduer, lister og annet trevirke. Løsningen skal til slutt bestå av et tekstbasert brukergrensesnitt og et register som lagrer informasjon.

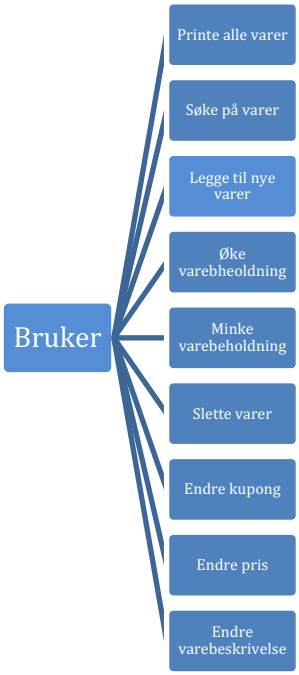
Smarthus AS - Del 1
Smarthus AS ønsker at en vare skal være registrert med følgende informasjon:
1. Varenummer – består av bokstaver og tall
2. Beskrivelse – en tekst som beskriver kort om varen
3. Pris – Heltall
4. Merkenavn – en tekst som inneholder merke (Hunton, Pergo, Egger osv.)
5. Vekt – i kilogram, som et desimaltall
6. Lengde - som et desimaltall
7. Høyde - som et desimaltall
8. Farge – beskrevet som tekst
9. Antall på lager - antall varer på lager. Skal aldri være mindre enn 0.
10. Kategori - et tall som representerer kategori av varen. Bruk følgende: (1) Gulv laminater, (2) Vinduer (3) Dører og (4) Trelast

Følgende krav gjelder til denne delen av oppgaven (del 1):

• Klassen og alle metoder, variabler (felt, parametere, lokale variabler) skal ha gode, beskrivende navn som tydelig gjenspeiler hvilken tjeneste en metode tilbyr, eller hvilken verdi variablene representerer/holder på.
• Alle navn på klasser, metoder og variabler skal være på engelsk.

Smarthus AS - Del 2
Lag et vareregister som implementerer nødvendig funksjonalitet for å kunne brukes av brukergrensesnittet. Vareregisteret bør ha en metode for å fylle registeret med en samling default varer for å kunne teste funksjonaliteten til registeret. Brukergrensesnittet skal tilby følgende funksjonalitet til brukeren som arbeider på lageret:

1. Skrive ut alle varer på lageret
2. Søke etter en gitt vare basert på Varenummer og/eller Beskrivelse
3. Legge en ny vare til registeret. Her skal all informasjon fra 1-10 felter (gittover) innhentes fra bruker input
4. Øke varebeholdningen til eksisterende vare. M.a.o. du har en vare med et gitt antall på lager (f.eks. 10 stk laminatgulv). Du mottar så en ny forsyning av laminatgulv som så skal registreres inn på lageret (f.eks. 20 stk).
5. Ta ut varer fra varebeholdningen (eksisterendevare). M.a.o. du har en vare med et gitt antall på lageret (f.eks. 20 stk laminatgulv). Du tar så ut 5 stk fra lageret.
6. Slette en vare fra varelageret (fordi den for eksempel er utgått eller ikke i produksjon lenger). M.a.o. du skal ikke lenger ha varen "Laminatgulv" i butikken din lenger. NB! Ikke det samme som a˚ sette antall varer til 0
7. Endre rabatt, pris og/eller varebeskrivelse for en vare.



Smarthus AS - Del 3
I **del 3** av mappen skal du ferdigstille applikasjonen din ved å ta i bruk teori gjennomgått mot slutten av emnet:

- Refaktorering, coupling, cohesion, dokumentert kode, kode stil osv.
- Enum for å håndtere **kategorier**.
- Refaktorering for å forbedre løsningen/designet ditt
- Hvordan vil du **teste** løsningen din for sikre at den er robust og godt designet?

I tillegg skal du nå **sette ditt eget preg** på løsningen:

- Med utgangspunkt i opprinnelig kravspesifikasjon, hvilke endringer/forbedringer ville du ha gjort for at applikasjonen skal bli enda mer nyttig for brukeren? Her har du lov til å gå bort 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, og legge til ny funksjonalitet.
- Brukergrensesnitt: du må gjerne tenke alternativt her. Feks, er en meny beste løsning (tatt i betraktning at dere ikke har lært å programmere grafisk brukergrensesnitt enda)?
- Dersom Smarthus ønsker å legge til en ny kategori av varer (feks, Hagemøbler) hvor mye av koden din må du endre? Hva tenker du dette forteller om ditt valgte design?
- 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.

3.2 Begreper/Ordliste

Begrep (Norsk)	**Begrep (Engelsk)**	**Betydning/beskrivelse**
Vare	Item	Vare eller produkt. En vare i varehuset.
Lager	Storage	Et lager som holder på varer.
Vare register	Item register	Et vareregister holder på en eller flere varer.
Brukerinnspill	User input	Det brukeren setter inn.
Varenummer	Item number	Varens unike varenummer.
Varebeskrivelse	Item description	Beskrivelse av varen.
Pris	Price	Prisen til varen.
Merke	Brand	Også kjent som varemerket. Merket varen er produsert av.
Vekt	Weight	Vekten til varen
Lengde	Length	Lengden til varen/bredden til varen.
Høyde	Height	Høyden til varen.
Farge	Colour	Fargen til varen.
Antall	Amount	Antall varer på lager.
Kategori	Category	Varens kategori.
Skriv ut	Print	Skriver ut i terminalen slik at bruker kan lese.
Søk	Search	Søke. Søke på varer i varelageret.
Legg til	Add	Legge til ny vare i varelageret.
Slett	Delete	Slette vare i varelageret.
Inventar	Inventory	Innhold i varelageret.
Beskrivelse	Description	Beskrivelse av varen.
Kupong/Rabatt	Coupon/discount	Kupong/Rabatt, nedsatt pris til varen.

Begrep (Norsk)	Begrep (Engelsk)	Betydning/beskrivelse
Avslutt	Exit	Avslutter programmet.

3.3 Rapportens oppbygning

Rapporten er innledet med et sammendrag som gir kortfattet informasjon som angår oppgave, problemstilling, mål og metode. Videre i rapporten blir det presentert terminologi, så oppgavebeskrivelse, ordliste, bakgrunn, design, resultat og drøfting.

Det første kapitlet i rapporten omhandler sammendraget, som blir etterfulgt av en terminologi i kapittel to. Kapittel tre består av tre delkapitler som beskriver selve oppgaven, ordliste og rapportens oppbygning. I kapittel fire blir selve bakgrunnen og det teoretiske grunnlaget for oppgaven presentert, hvor blant annet læringsutbytte er nevnt.

I den originale malen er det også et fjerde delkapittel som omhandler avgrensninger. Dette delkapittelet er fjernet bevisst, med tanke på at oppgaven ikke har noen konkrete avgrensninger eller begrensninger annet enn at oppgaveløseren må løse oppgaven i henhold til oppgavebeskrivelsen. Måten oppgaveløseren løser oppgaven på, er dermed åpen og fritt så lenge oppgaven er løst på en god måte.

Kapittel fem tar for seg design og metode. Innledningsvis presenteres utviklingsmiljø, og hjelpemidler under prosjektet. Det blir beskrevet hvilke valg som har blitt tatt under prosjektet, og hvorfor disse valgene ble foretatt. Videre presenteres resultatet av oppgaven i kapittel seks. Det er oppgavens største del, som tar for seg ulike løsninger som er blitt vurdert i prosessen for å komme frem til den endelige valgte løsningen.

Deretter kommer kapittel sju som er drøftingsdelen av prosjektet, hvor vurdering av metode og resultat blir tatt opp. Det er deretter kapittel åtte som tar for seg konklusjonen. Erfaringer oppnådd ved prosjektet blir drøftet og gjort klart i denne delen. Til slutt er det referanser i kapittel ni, hvor alt av bøker og nettsider som er blitt brukt som hjelpemiddel under oppgaveløsing er listet opp.

4 BAKGRUNN - TEORETISK GRUNNLAG

Denne mappevurderingen ble lagt ut gradvis (dvs. Del 1 kom først, så kom del 2 og del 3), som førte til at alle vurderinger og valg ble endret på underveis i prosjektet. Del 1 av oppgaven nevnte for eksempel ikke at det skulle være Enum for kategorier (som ble nevnt i del 3) og da måtte koden endres litt på for å svare på oppgaven. Underveis i prosjektet har det skjedd en del refaktoreringer.

Ved en analyse av problemstillingen som del 1 av oppgaven gir, kommer det frem flere viktige momenter. Ett av disse er at vi må ha en sjekk som sjekker at brukerens innspill er riktig, slik at blant annet krav nr.9 skal bli oppfylt.
Allerede fra det første momentet ser vi at det er mulig å hente en CheckValid klasse som ble utviklet med medelever og faglærer i en av forelesningene under semesteret, hvor det ble laget en sjekker metode for String [2].
Denne klassen kan bli refaktorert slik at den sjekker riktig for String, int og double, og eventuelt andre sjekker som kan bli satt opp i løpet av prosjektet.

I en klasse er det ofte objekter. Klassen beskriver hva slags objekt det er, mens objektene representerer klassen [1].
Når feltene i vare-klassen skal settes opp, er det viktig å bruke riktige begreper. Alle feltene skal være private fordi bare selve klassen skal ha tilgang på det som står i feltene. Når setters og getters skal lages, er det viktig å bruke riktige begrep også her. Getter-metodene er tilgangsmetoder som returnerer verdien av en privat variabel. Andre klasser har da tilgang til verdien som er lagret i denne variabelen, uten å ha direkte tilgang til selve variabelen [1].
Setter-metodene tar en (eller flere) parametere og tilordner dem til en variabel. Nøkkelordet «this» brukes til å referere til det gjeldende objektet [1].

Fra alle de tre delene av oppgaven kombinert, kan en plan settes opp for hvordan oppgaven skal bli løst på en god måte. For å ha en litt ryddig oversikt over koden, kan det være fint å lage ulike pakker som inneholder klasser der klassene snakker med hverandre. For eksempel kan det bli laget en pakke som heter iteminformation, hvor klassen Item, ItemRegister, Category og CategoryEnum har en sammenheng fordi Category snakker med CategoryEnum, og Item snakker med Category, og ItemRegister snakker med Item (dette er tatt i betraktning av del 1 som implementerer Item klassen, del 2 som implementerer ItemRegister klassen og del 3 som implementerer kategori for en vare som Enum).
Å lage slike pakker kan gi en god cohesion, slik at klassene kan gjenbrukes (eller resirkuleres) og for eksempel brukes i en annen kontekst [1].

Når klasser skal snakke med hverandre, er det viktig at de kan snakke med hverandre uten å være tett tilknyttet. Dvs. De må ha «loose coupling» slik at det å endre på en klasse ikke fører til en nødvendighet i å endre på mange andre klasser [1].
Det å ha en loose coupling vil også gjøre refaktorering av koden mye lettere.

I UserInput delen av prosjektet kan det bli laget en egen pakke kalt input. Denne pakken kan inneholde kommandoer en bruker kan kunne bruke, og selve valgene en bruker har i applikasjonen.
I en av forelesningene mot slutten av semesteret, lagde vi et World of Zuul spill hvor vi brukte en Enum klasse for kommandoer [2].
For å gjøre kommando koden «enkel» slik at det å legge til en ny kommando ikke blir mye arbeid, kan det lages et HashMap som holder på kommandoene slik at vi ikke trenger å lage separate variabler. Når dette blir gjort, kan koden inneholde mange kommandoer og det kan bli lagt til nye kommandoer uten at det blir mye ekstra arbeid [1].

Det finnes veldig mange måter å løse oppgaven på. De valgene som blir tatt med grunnlag på teori, er bare vurdert ut fra en tolkning av teorien. Det vil si at det ikke alltid er alle valg som funker like bra, eller at valgene er gode selv om teorien er god. Det er viktig å se sammenhengen mellom flere teorier (fordi koding har ikke bare én teori) slik at de valgene som blir tatt skal fungere.

5 METODE – DESIGN

I prosjektet ble det benyttet utviklingsmiljøet IntelliJ med CheckStyle og SonarLint verktøy til å utføre oppgaven. Med CheckStyle er den importerte sjekkstilen «IDATX1001» brukt.

Del 1 av mappevurderingen startet med en implementering av Item klassen. Item klassen definerer en vare med følgende informasjon:
String itemNumber for varenummer som består av bokstaver og tall,
String itemDescription for beskrivelse, en kort tekst som beskriver varen,
int price for prisen til varen som består av heltall
String brand for merkenavn, en tekst som inneholder varemerket til en vare,
double itemWeight for vekt med desimaltall (her blir det også nevnt i oppgaven at vekten skal være oppgitt i kilogram. Da ble det satt opp i utskrift av vare at kg skal bli skrevet ut med utskrift av vekt),
double for itemHeight og itemLength for tall med desimal (her står det ikke i oppgaveteksten noen måleenhet, men jeg lagde allikevel en utskrift slik at meter skal bli skrevet ut med utskrift av høyde og lengde),
String itemColor for farge beskrevet som tekst,
int storageAmount for antall på lager beskrevet med tall som aldri skal være mindre enn 0,
int itemCategory for kategori av en vare representert med tall.
De datatypene jeg har valgt for feltene til klassen har jeg valgt etter det jeg mener passer best og litt etter hva oppgaven ber om.

Deretter ble det laget en konstruktør med 10 parametere. Ifølge CheckStyle var det bare lov med syv parametere og da ble det et valg om hvilke vare informasjon som ikke skulle bli lagt til i konstruktøren, men som en setter istedenfor.

Det ble valgt at konstruktøren ikke skulle inneholde lengde (itemWeight), høyde (itemHeight) eller antall på lager (storageAmount). Vareinformasjonen som ble valgt å ikke ha med i parameteren til konstruktøren ble heller brukt som setters.

Da alle felter og konstruktøren var lagd, ble setters og getters lagt til. Alle getter metodene er mutator-metoder, tilgangsmetoder som returnerer verdien av en privat variabel slik at andre klasser har tilgang til verdien, men ikke selve variabelen. Alle feltene som er satt opp har en slik getter metode.

For String, int og double ble det laget en ny klasse med navn CheckValid for å sikre at det ikke er mulig å angi ugyldige verdier. CheckValid klassen er med på å sikre at Item klassen er robust. Denne klassen ble puttet i en egen pakke kalt utility. Item klassen ble også da flyttet til sin egen pakke kalt iteminformation.

Til slutt be det laget en main klasse for testing av metoder og funksjoner som fikk navn WarehouseManagementSystem. Dette var det endelige resultatet av del 1 av oppgaven.

Da del 2 av mappevurderingen ble lagt ut, ble det implementert en ny klasse kalt ItemRegister som skal representere vareregisteret. Denne klassen ble lagt inn i samme pakke som Item, altså iteminformation pakken. ItemRegister skal holde på alle varene i varelageret. I ItemRegister ble det opprettet et HashMap som holder på alle objekter av

klassen Item (som da er alle varene i varelageret). HashMap ble brukt fordi det holder på key og values, hvor Item er valgt som value og itemNumber er valgt som key. Hver vare har et unikt varenummer, og HashMap er lett og enkelt å bruke til de oppgavene som skal brukes som å søke på varer, legge til varer og slette varer.

Del 2 ba også om å implementere et enkelt brukergrensesnitt. Da ble det laget en klasse kalt UserInterface. Etter hvert som jeg har jobbet videre med denne oppgaven har UserInterface blitt delt i flere deler. Først var det bare en enkel klasse som inneholdt både Scanner, metoder for å lese String som int og double, alle system.out.println og system.err.println, og alle kommandoer en bruker kan bruke. Nå er klassen bare en brukeropplevelses klasse hvor alle system.out.println og system.err.println er. Alt som blir skrevet ut i terminalen står da i UserInterface. Scanner ble laget i sin egen klasse kalt UserInput, og kommando ordene ble laget i to egne klasser, CommandWordsEnum og CommandWords. CommandWordsEnum er en Enum klasse som inneholder alle de kommandoene en bruker kan bruke. CommandWords er en klasse med et HashMap som holder på alle disse kommandoene. I UserInput klassen er det også metoder for at Scanner skal kunne lese brukerens input som en String og oversette til int eller double der det er nødvendig (som når en bruker legger til en ny vare i varelageret hvor blant annet pris tar int og blant annet vekt tar double).

Det ble laget en ny klasse kalt MenuOptions, hvor det ble laget en switch-case for brukerens valg, noe som var mye mer optimaliserende enn å skrive mange if-else settinger for alle de ulike kommandoene. I switch-case er det brukt Enum, hvor Enum ordene er hentet fra CommandWordsEnum. UserInput, MenuOptions, CommandWordsEnum og CommandWords ble lagt i en egen klasse kalt input. I UserInterface, som ble flyttet til en egen pakke kalt userinterface, ble det laget en del feil-meldinger. Disse ble brukt i UserInput klassen i switch-case slik at når en bruker skriver inn feil kommando, eller om brukeren skriver bokstaver istedenfor tall, så vil terminalen skrive ut en feil melding som enten tvinger brukeren til å skrive på nytt, eller kaster brukeren helt ut og tilbake til hovedmenyen.

I UserInterface klassen ble det laget konstanter, slik at det ikke ble mye gjentakelser i koden. For eksempel ble måleenheter og valuta skrevet som konstanter. Dette var en fin løsning, som fører til at om noen ønsker å endre på valuta eller måleverdier, så kan de lett bare skrive om konstanten, og da vil dette endres automatisk for alle utskrivninger av varer.

I item klassen ble det laget en setter metode for rabatt. Her måtte «Math» bli brukt fordi integer rundet tallet opp eller ned (fordi det er heltall) slik at utregningen ikke ble riktig.

I MenuOptions klassen ble det laget en while-løkke som sikrer at applikasjonen ikke bare lukker seg automatisk, men at det hele tiden venter på input fra brukeren. Dette fører også til at programmet ikke lukker seg ved mindre brukeren skriver «exit». Dette er ikke svært praktisk da jeg personlig har ofte glemt å skrive exit som fører til at applikasjonen bare fortsetter å stå på i bakgrunnen når jeg gjør andre ting. Det ville vært fint å ha en timer her hvor, hvis brukeren ikke skriver noe på en halvtime så vil programmet lukkes automatisk.

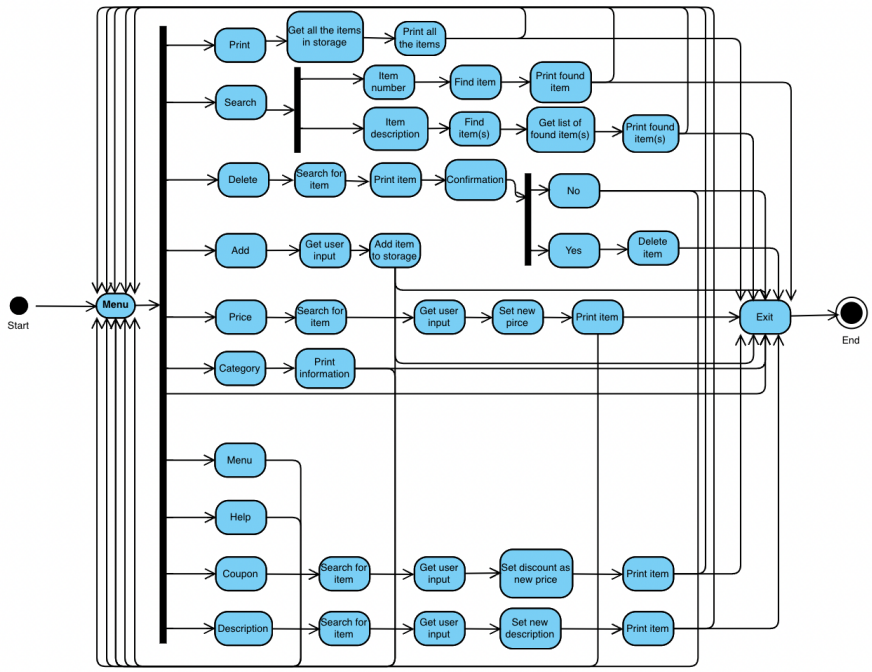
Etter at del 3 av oppgaven kom ut, måtte jeg refaktorere koden litt. Dette ble gjort uten store problemer. Jeg måtte lage en ny Enum-klasse til å håndtere kategorier for en vare.

Det ble også laget en egen klasse med et HashMap som holder på disse kategoriene og hvilke tall en kategori representerer. Så var det bare å bytte ut int category med CategoryEnum (den nye Enum klassen for kategori representasjon) category.

For å teste om applikasjonen jeg utviklet var god, utførte jeg først og fremst tester selv. Da jeg hadde fikset alle de problemene jeg selv så, utførte jeg tester på andre. Jeg spurte blant annet faglærer i fagnær ingeniørfag (Ivar Farup), til å teste applikasjonen min. Jeg spurte også studasser og medelever om testing slik at jeg fikk et bredt perspektiv. Jeg spurte også om hjelp hjemme til testing, slik at jeg fikk et perspektiv fra utsiden av mennesker som ikke har samme teoretiske kunnskap og bakgrunn for oppgaven. Dette ga meg en god del gode tilbakemeldinger og forbedringer som jeg senere implementerte i den endelige løsningen av applikasjonen. De fleste innspill og testinger jeg foretok var også bare basert på selve bruker opplevelse. Dermed er ikke selve koden og kodedesignet like mye testet.

6 RESULTATER

Aktivitets diagram basert på den ferdig utviklede applikasjonen:



Etter at det ble foretatt brukeropplevelse tester, ble det lagt til en bekreftelsesdel i sletting av en vare. Dette ble lagt til for å sikre at ingen kan slette en vare ved et uhell, og at en bruker har da mulighet til å først finne riktig vare før varen kan bli slettet.

I oppgavebeskrivelsen av del 2 står det blant annet i punkt nr. 2 at bruker skal kunne søke på en gitt vare basert på varenummer OG/eller beskrivelse. Her er det blitt tatt et bevisst valg om at bruker ikke skal trenge å søke på både varenummer og beskrivelse fordi en vare er definert slik at en trenger ikke å søke på de to vareinformasjonene samtidig for å finne en spesifikk vare. Det er derfor bare et valg om en bruker vil søke på en gitt vare basert på varenummer (hvor brukeren vil da få denne gitte varen) eller at en bruker kan søke på en gitt vare basert på vare beskrivelse (hvor brukeren vil få alle varer som har lik beskrivelse).

Jeg tror ikke jeg har vurdert så mange ulike løsninger. For det meste har jeg bare programmert underveis og lagt til funksjoner og gjort refaktoreringer i henhold til det oppgaven ber om. Jeg føler dessuten at alt av vurderinger av løsninger, samt teoriene mot slutten av boka ikke ble så godt gjennomgått mot slutten av semesteret, så jeg har ikke følt at jeg har kunnet det så godt. Det er også en god del utfordringer i hvordan jeg skal skrive ned i en rapport hvilke valg jeg har tatt og hvorfor. Det er vanskelig å formulere på en god måte slik at det er mulig å forstå hva jeg har gjort og hvorfor.

Det endelige resultatet jeg står igjen med, er et program hvor en kan legge til varer, slette varer, finne varer ved hjelp av enten beskrivelse eller varenummer, en kan til og med endre på vare beskrivelse, pris, og legge til rabatt slik at varen er blir billigere. En ønsket løsning er også å få endret fargen til prisen etter at det er lagt på rabatt, til for eksempel grønn, og så få en hevet skrift slik at brukeren kan se at varen har en rabatt.

WarehouseManagementSystem applikasjonen er ganske lett å bruke. Brukeren må bare kunne lese og følge med på det terminalen sier. Det meste av informasjon blir gitt der, blant annet hva brukeren kan skrive og om brukeren trenger å skrive med store bokstaver eller ikke. For eksempel skiller applikasjonen mellom store og små bokstaver når en skriver varenummeret til en vare, men ikke når en skal skrive en vanlig kommando.

For å sikre at programmet ikke krasjer tilfeldig, har jeg foretatt tester av applikasjonen både selv og på andre. Jeg har for eksempel gitt applikasjonen til en annen faglærer (Ivar Farup) og sagt, «Værsågod, prøv å ødelegge eller krasje programmet». Slik har jeg fikset at koden min funker som jeg vil.

Slik ser alle klassene i programmet ut:

Item	Representerer en vare i varelageret.
ItemRegister	Vareregisteret som holder på varene og implementerer brukerens ønsker.
CheckValid	Sjekker at det er riktig verdier som blir puttet inn av bruker.
UserInterface	Printer ut all informasjon til terminalen.
Category	Kategoriene fra Enum klassen holdt i et HashMap.
CategoryEnum	Enum klasse som representerer tilgjengelige kategorier.
UserInput	Leser brukerens innspill, oversetter slik at programmet forstår brukerens ønsker.
MenuOptions	Implementerer brukerens valgte kommando.
CommandWords	Kommandoene fra Enum klassen holdt i et HashMap.
CommandWordsEnum	Enum klasse som representerer tilgjengelige kommandoer.
WarehouseManagementSystem	Main klassen som kjører programmet.

I kravspesifikasjonen av oppgaven i del 1 og del 2 står det følgende:

- Koden skal følge en bestemt ***kodestil*** (enten Google eller "BlueJ"-stilen gitt i regelfilen "IDATx1001" for Chekstyle)
- Kodestilen **skal** verifiseres med **CheckStyle**-plugin (for BlueJ, IntelliJ osv) og vise ingen regelbrudd ved levering.
- **Klassen** og alle **metoder**, **variabler** (felt, parametre, lokale variabler) **skal** ha gode, beskrivende navn som tydelig gjenspeiler hvilken tjeneste en metode tilbyr, eller hvilken verdi variablene representerer/holder på.
- Alle navn på klasser, metoder og variabler **skal** være på **engelsk**.

Kodestilen i programmet har blitt fulgt, med CheckStyle og regelfilen IDATX1001. Kodestilen skal vise ingen regelbrudd, noe min kode gjør. Dette har da blitt avklart med faglærer, om at dette skulle gå greit. De feilene jeg har i CheckStyle er godkjent. Alle klassene, metodene og variablene jeg har laget har gode og beskrivende navn, slik at en nesten kan tyde hva de gjør. Alle navn er på engelsk slik kravet krever.

I del 3 står de samme kravene, men med ett ekstra punkt:

- Koden skal være dokumentert iht standarden for JavaDoc (se hvordan JDK'en er dokumentert, for inspirasjon og som referanse).

I henhold til alle kravspesifikasjonene oppgaven gir, har jeg gjort det som er blitt bedt om. Jeg sitter igjen med et program som funker, og gjør det den blir bedt om å gjøre. Om det er den beste løsningen, er jeg usikker på. Det finnes forbedringer i koden selvfølgelig, men også i selve brukeropplevelsen. Mye av oppsettet kunne vært litt enklere (og da mener jeg i meny osv.) slik at det ikke blir så mye å lese og overveldende for en bruker. Likevel synes jeg at løsningen min er grei, den fungerer ok i henhold til de testene jeg har utført.

7 DRØFTING

Det var mye utfordring å lage et bra system når det gjaldt antall varer og kategorier. Med unike varenumre, hvordan skal man definere hvor mange varer det finnes av de ulike varene i varelageret?

En vare i dette prosjektet er definert som følgende:

Varenummer er unikt for en vare, men ikke for like varer. Det vil si, en blå dør vil ha samme varenummer som en annen blå dør, så lenge all annen informasjon er likt. Når en vare blir skrevet ut fra varelageret, vil det bare bli skrevet ut informasjonen som beskriver alle varene av denne spesifikke varen i lageret.

En blå dør, fra Dior, som koster 500 kr, vil for eksempel representere alle de blå dørene fra Dior som koster 500 kr. Utskriften av varer vil derfor ikke skrive ut en rekke identiske varer, men en representasjon av varen, og så vil antall på lager beskrive hvor mange det finnes av akkurat den varen i varelageret.

Del 1 av oppgaven var ganske enkel. Det som skulle gjøres, var å lage selve klassen som representerer en vare. Da skulle all informasjon til en vare bli lagt inn i denne klassen. Og det var egentlig det. Allerede i del 1 av oppgaven la jeg til en egen klasse kalt CheckValid, for å sikre at informasjonen en vare er beskrevet med, er riktig. Ut ifra det oppgaven i Del 1 ber om, mener jeg at jeg har besvart oppgaven. I tillegg har jeg gjort det kravet til Del 1 ber om, blant annet er alt på engelsk, og jeg mener at alle klasser, variabler og metoder jeg har gitt navn til har fått gode og beskrivende navn.

I Del 2 av oppgaven kom det litt flere krav til hvordan dette systemet skal brukes på varelageret. Funksjonen hvor en bruker kan søke på en vare ble implementert først. Det var det jeg opplevde som lettest å implementere. Deretter ble sletting av varer implementert, så utskrift av alle varer, og til slutt endring av varelagerbeholdning, pris, beskrivelse og rabatt. Her gjenbrukte jeg en søkemetode jeg hadde laget for søking av varer i ItemRegister klassen, slik at jeg ikke trengte å skrive en ny kode for hvert eneste søk av varer i de forskjellige funksjonalitetene.

Del 3 av oppgaven ber om å håndtere kategorier som Enum. Etter del 1 var det allerede valgt int for kategori, men Enum var lett å implementere istedenfor int.

Jeg ble godt fornøyd med løsningen min, men etter å ha lekt litt rundt med funksjonalitetene og prøvd ut programmet selv ser jeg en god del rom for forbedringer. Generelt ser jeg selv at koden min ser litt rotete ut. Kodestilen mener jeg at jeg har fulgt til beste evne, da vi har brukt CheckStyle og jeg (for det meste) har ingen feil eller mangler. Jeg mener at jeg har lite kode duplikasjon, da jeg har vært flink til å gjenbruke koder jeg har skrevet til andre ting (som søke-koden som ble brukt i flere kontekster).

Det er helt klart et forbedringspotensial i oppsett av koden. Jeg kunne oppnådd enda bedre loose coupling, samt enda høyere cohesion. Jeg tror med mer tid, så kunne jeg fått et mye bedre responsibility-driven design. Jeg har jo fått til alle funksjonene som oppgaven ber om, men jeg tror ikke jeg har fått det til på en veldig god måte. Dette er nok et prosjekt jeg kanskje vil leke meg litt videre med samtidig som vi går videre i utdanningsløpet slik at jeg kan forbedre koden og gjøre den mer robust, og gi den enda bedre design.

8 KONKLUSJON – ERFARING

Det innleverte prosjektet er ikke et endelig resultat av et lagerstyringssystem. Applikasjoner utvikler seg mye med tiden, og det vil dette prosjektet også gjøre. Det blir aldri helt «ferdig».

Dersom jeg kunne gjort hele dette prosjektet på nytt, hadde det vært høyest ønskelig å kunne ha tid til å lære ordentlig grafisk brukergrensesnitt slik at selve bruker opplevelsen blir bedre. Det er også en god del justeringer i koden som kunne gjort den både finere, lettere, og enda mer robust. De erfaringene jeg har fått av å jobbe med dette prosjektet er at det finnes mange mulige løsninger. Det er ingen fasit til hvordan en slik oppgave skal løses, så det er mange muligheter.

Hadde utviklingen blitt tatt videre hadde det skjedd mye refaktorering i koden. Det er en del funksjoner vi ikke har lært enda, som kan gjøre lange kodelinjer mye kortere. Også er det helt sikkert mye sikrere koder, som gjør at det «nesten» ikke går an å gjøre feil. Etter å selv ha testet litt rundt på koden så ser jeg for eksempel, at programmet krasjer hvis jeg fjerner alle varene i varelageret og printer ut alle varene etterpå. Det skulle vært

en sjekk som sjekker om varelageret (HashMap) er tomt, og hvis det er tomt så skriver det ut at det ikke er varer på varelageret.

Med den kompetansen jeg har fra før, og fra det som har blitt lært i løpet av et semester på 5 måneder er jeg fornøyd med min valgte løsning. Jeg kommer helt sikkert til å se tilbake på denne løsningen senere, om noen år kanskje, når jeg har lært meg enda mer og se at det er veldig mye jeg kunne gjort annerledes og forbedret. Kanskje jeg vil se på løsningen min og tenke «Herre gud, lærte jeg ingen ting?» eller så vil jeg bare se på løsningen som en «You did the best with what you had». Jeg likte veldig godt å jobbe med dette prosjektet, og jeg føler jeg har lært ganske mye underveis i arbeidet. Koden min funker i hvert fall, så jeg er superfornøyd.

9 REFERANSER

[1] David J. Barnes and Michael Kölling, 2017,” Objects First with Java”, Sixth edition, ISBN.

[2] IDATG1001 Programmering 1, Kiran Raja, forelesninger.

[3] https://www.w3schools.com/java/java_intro.asp

[4] https://beginnersbook.com/2017/10/java-8-filter-a-map-by-keys-and-values/

[5] https://docs.oracle.com/javase/8/docs/api/overview-summary.html