STUDENTRAPPORT

NTNU

KANDIDATNUMMER(E)/NAVN:
10015 / Lars Mikkel Lødeng Nilsen

DATO:	FAGKODE:	STUDIUM:	ANT SIDER/BILAG:
12.12.2022	IDAT1001	Programmering 1	14 /

FAGLÆRER(E) :
Muhammad Ali Norozi
Surya Kathayat

TITTEL :
Warehouse rapport

SAMMENDRAG:
Rapport beskriver utviklingen av en programvare for et varehusfirma, Smarthus AS, for å administrere lageret deres. Rapporten er delt inn i tre deler: Del 1 omhandler utviklingen av en entitestklasse for varer, del 2 omhandler utviklingen av en registerklasse for varer og et enkelt brukergrensesnitt, mens del 3 omhandler forbedringer og finpussing av programmet, samt utviklingen av en enum-klasse for kategorier til varene. I tillegg inneholder rapporten en teoretisk bakgrunn om cohesion, coupling, modularisering, aggeresjon og komposisjon samt en diskusjon om hvordan disse prinsippene ble implementert i utviklingen av programvaren. Videre står det litt om metode. Det står blant annet litt om hvilken programvare som ble brukt, og arbeidsprosessen mot det endelige resultatet. Neste avsnitt, resultatet, handler om det endelige produktet, der er alle klassene beskrevet, i tillegg til litt om arbeidsprosessen fram til det endelige resultatet. Det står også en kort brukerveiledning og litt om verktøyene som ble brukt. Til slutt står det litt om hvordan jeg syntes oppgaven ble løst, og om mine erfaringer etter å ha jobbet med dette prosjektarbeidet.

INNHOLD

1 INNLEDNING – PROBLEMSTILLING

1.1 Bakgrunn/Formål og problemstilling

Denne oppgaven var en mappevurdering gitt av faget IDAT1001 ved NTNU Trondheim. Vurderingen vi fikk på oppgaven skulle telle som den endelige karakteren i faget.

I oppgaven skulle det utvikles en programvare som skal brukes av et varehusfirma kalt Smarthus AS for å administrere lageret deres. Oppgaven deles inn i tre deler. I del 1 skulle man lage en entitestklasse som representerer en vare. I del to skulle man lage en registerklasse for varer, i tillegg til et enkelt brukergrensesnitt. I del 3 skulle man gjøre nødvendige forbedringer og finpusse på programmet. Man skulle også i del 3 lage en enum klasse for kategorier til varene. I tillegg til selve programmet skal det også skrives en rapport.

1.2 Teminologi-liste

Klasser:

- Category: Klasse for kategorien til en vare.
- Item: Klasse for en vare.
- ItemRegistry: Klasse for et register med varer.
- Menu: Klasse for en meny.

Metoder:

- set-metoder: Metoder for å endre verdien til bestemte atributter.
- get-metoder: Metoder for å hente verdien til bestemte atributter.

Datatyper:

- ArrayList: En ArrayList er en datatype som lar deg lagre en samling av elementer i en rekkefølge. ArrayList er en dynamisk struktur, noe som betyr at den automatisk justerer størrelsen når elementer legges til eller fjernes. ArrayList bruker indekser for å hente elementer, noe som gjør det enkelt å søke og endre på elementer i listen.

- int (short for integer) er en datatype som brukes for å representere heltall.
- double er en datatype som brukes for å representere flyttall (tall med desimaler).
- string er en datatype som brukes for å representere tekststrenger.
- boolean er en datatype som kan ha to verdier: true (sant) og false (usant).

Andre begreper:

- Cohesion: Graden av sammenheng og samarbeid mellom de forskjellige elementene i et Java-program eller klasse.
- Kobling: Graden av avhengighet og sammenheng mellom forskjellige deler av et Java-program.
- Aggregering: Sammenheng mellom klasser, hvor klassene er likeverdige og kan eksistere uavhengig av hverandre.
- Komposisjon: Sammenheng mellom klasser der en klasse bruker en eller flere andre klasser, og de er tett integrert og avhengige av hverandre.
- Robuste programmer: Programmer som er stabile og kan motstå endringer.
- Utviklingsmiljø: Et program eller verktøy som brukes for å utvikle og teste kode.
- IntelliJ: Et integrert utviklingsmiljø (IDE) for Java-programmering.
- Kodestandard: En standard eller retningslinje for hvordan kode bør skrives og organisert.
- CheckStyle-IDEA: Et verktøy for å sjekke at kode følger en bestemt kodestandard.
- SonarLint: Et verktøy for å finne og løse problemer i kode, som for eksempel feil eller dårlige programmeringspraksiser.
- UML-verktøyet: Et verktøy for å designe og visualisere klassediagrammer.
- Klassediagram: En type diagram som brukes i objektorientert. programmering for å visualisere relasjoner og interaksjoner mellom klasser og objekter.
- Enum-class: En Java-klasse som inneholder en definert liste med konstanter.

1.3 Begreper/Ordliste

Begrep (Norsk)	Begrep (Engelsk)	Betyding/beskrivelse
Vare	Item	Vare som legges inn i vare register.
varenummer	ItemNumber	Unikt varenummer til vare
beskrivelse	Description	Beskrivelse til vare
pris	Price	Pris til vare
Merkenavn	brandName	Merkenavn til vare
vekt	weight	Vekt til vare
lengde	Length	Lengde til vare
høyde	Height	Høyde til vare
farge	Color	Farge til vare
Antall på lager	AmountInStock	Antall av vare på lager
kategori	Category	kategori til vare
Rabbat prosent	DicountPercentage	Rabatt i prosent til vare
Vare register	ItemRegistry	Vare register
Hoved klasse	Main	Hoved klasse
Kategori klasse	Category	Kategori klasse

2 BAKGRUNN - TEORETISK GRUNNLAG

2.1 Cohesion

I Java er cohesion en betegnelse for graden av sammenheng og samarbeid mellom de forskjellige elementene i et program eller klasse. (Udacity, 2022, [2]) Det vil si at det er en måling på hvor godt de ulike delene av koden jobber sammen for å oppnå et felles mål. Jo mer cohesion et program har, desto enklere vil det være å forstå og vedlikeholde koden. Cohesion kan også øke ytelsen til programmet, da det vil være lettere å identifisere og fikse eventuelle problemer. En god måte å oppnå cohesion i Java er å sørge for at hver klasse har en tydelig definert funksjon og ansvar, og at den kun inneholder de metodene og variablene som er nødvendige for å utføre sitt formål.

2.2 Kobling

I Java refererer kobling til graden av avhengighet og sammenheng mellom forskjellige deler av et program.(Udacity, 2015, [1]) Et Java-program kan ha høy kobling når de ulike delene av programmet er sterkt avhengige av hverandre, mens et program med lav kobling har færre avhengigheter mellom sine komponenter.

En viktig form for kobling i Java er koblingen mellom klasser. En klasse kan ha høy kobling til andre klasser når den avhenger av disse klassene for å fungere korrekt, for eksempel ved å inneholde attributter som er instanser av de andre klassene, eller ved å kalle på metoder i de andre klassene. Lav kobling, derimot, innebærer at klassene har færre avhengigheter mellom seg, slik at endringer i en klasse ikke påvirker de andre klassene i programmet.

Høy kobling kan føre til at endringer i en klasse kan påvirke andre klasser på en uforutsigbar måte, noe som kan gjøre det vanskeligere å vedlikeholde og endre programmet. Derfor er lav kobling generelt ønskelig i Java-programmering, da det kan bidra til mer robuste og fleksible programmer.

2.3 Aggregering og komposisjon

I Java kan klasser benytte seg av aggregering og komposisjon for å organisere og samarbeide med andre klasser.(Trivedi, 2022)

Aggregering oppstår nå en klasse samarbeider med en annen klasse, men de to klassene er likeverdige og kan eksistere uavhengig av hverandre. For eksempel kan en klasse for en bil inneholde attributter for hjul, motor og seter, mens hjul, motor og seter kan være selvstendige klasser som eksisterer uavhengig av bil-klassen.

Komposisjon innebærer at en klasse bruker en eller flere andre klasser, og klassene er tett integrert og avhengige av hverandre. For eksempel kan en klasse for en mus inneholde et attributt for muspeker, og muspekeren kan bare eksistere i sammenheng med mus-klassen. I dette tilfellet kan ikke muspekeren eksistere uavhengig av mus-klassen, og dette representerer en form for komposisjon.

Både aggregering og komposisjon kan være nyttige for å strukturere og organisere klasser i et Java-program, men det er viktig å velge den riktige

relasjonen avhengig av hvordan klassene skal samarbeide. Aggregering gir mer fleksibilitet og mulighet for gjenbruk av kode, mens komposisjon kan være nyttig for å sikre at komponentklasser fungerer sammen på en forutsigbar måte.

2.4 Modulisering

Modulisering i Java handler om å dele opp et program i mindre, selvstendige enheter kalt moduler. Hver modul inneholder en gruppe relaterte klasser, funksjoner og andre komponenter, og kan testes, debugges og vedlikeholdes uavhengig av andre moduler i programmet. Modulisering gir en rekke fordeler, blant annet økt gjenbruk av kode, bedre struktur og organisering av programmet, og enklere testing og vedlikehold. (Lalani, 2021)

I Java er det to hovedmåter å oppnå modulisering på: pakker og moduler. En pakke er en samling av relaterte klasser og andre elementer som er organisert i en logisk struktur, og som kan importeres og brukes av andre deler av programmet. En modul, på sin side, er en gruppe av pakker og andre ressurser som definerer en selvstendig enhet i programmet, og som kan eksporteres og brukes av andre moduler.

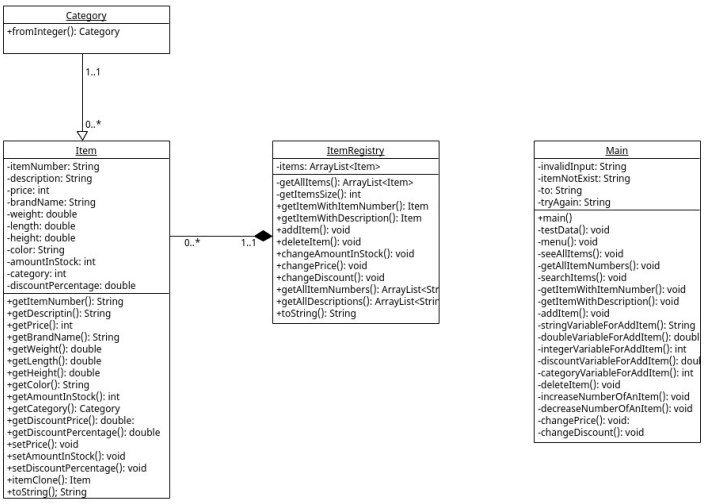
For å bruke modulisering i Java, kan du for eksempel definere en pakke for en gruppe relaterte klasser, og deretter definere en modul som inneholder pakken og andre ressurser som er nødvendige for å utføre en bestemt oppgave i programmet. Deretter kan modulen eksporteres og brukes av andre deler av programmet, slik at du kan oppnå en mer strukturert og fleksibel programvare.

3 METODE – DESIG

I denne oppgaven har det meste av arbeidet vært gjort selvstendig. Etter hver del av mappevurderingen, ble delen kodet og testet, før neste del ble lagt ut. Hele programmet ble skrevet i IntelliJ, siden det er dette utviklingsmiljøet jeg har mest erfaring med å bruke fra før. En annen grunn til at jeg valgte IntelliJ, er at det kan foreslå endringer som fjerner unødvendig kode og gjør programmet lettere å lese. Jeg lastet ned CheckStyle-IDEA, hvor jeg brukte Google sin kodestandard for koden. Jeg lastet også ned SonarLint hvor jeg også der prøvde å rette opp i alle feilene SonarLint fant i koden min. Jeg rettet ikke opp i

feilmeldingene der det sto at jeg skulle bruke logger istedenfor system.out.println, fordi det har vi ikke lært. I tillegg ble det brukt UML-verktøyet for å designe klassediagrammet.

4 RESULTATER



4.1 Klassediagram og arbeidsprosess

Ganske tidlig bestemte jeg meg for hvordan klassene skulle samarbeide med hverandre, og lagde et klassediagram ut ifra det. Figuren over viser det endelige klassediagrammet med alle metoder og variabler jeg har brukt i koden. Klassen Category som representerer kategorien til en vare har en tilknytning til Item klassen. Videre er Item klassen som representerer en vare tilknytta ItemRegistry, som representerer et register med varer. Jeg valgte å knytte Item og ItemRegistry sammen med aggresjon [2.3], fordi jeg ikke ville at en vare skulle eksistere utenom vare registeret. Altså hvis man sletter vare registeret blir også alle varene tilknytta vareregisteret sletta. Jeg oppnådde dette ved å ha en ArrayList inni ItemRegistry klassen, som lagret alle varene. Grunnen til at jeg valgte å bruke ArrayList er fordi jeg har brukt det før til et lignende prosjekt, også visste jeg at det kom til å være nyttig å ha inexer på alle varene i vare registeret, siden jeg skulle bruke det i flere metoder.

Metodene har jeg endret mye på underveis i prosessen. Blant annet har jeg lagd mange nye metoder underveis for kode som jeg brukte i flere metoder. Jeg gjorde

dette for å oppnå at en metode i størst mulig grad hadde et ansvar og gjorde en ting, for å oppnå bedre cohesion[2.1]. Jeg har også måtte endre mye på koden helt til slutt for å oppfylle SonarLint sine krav, og med det gjøre koden lettere å lese.

4.2 Klasser

Den endelige koden min består av fire klasser, en klasse for en vare, en for kategorien til en vare, en for et register av varer, og en klasse for en meny. Disse klassen inneholder til sammen alle metodene man tenger for å utføre operasjonene gitt i kravspesifikasjonen til oppgaven.

Category klassen er en enum-class, som har fire konstanter; gulv laminater, vinduer, dører og trelast. Jeg har også en metode i denne klassen som tar inn et heltall og returnerer en kategori.

Item klassen representerer en varer. Et objekt av denne klassen, altså en vare har 11 attributter; varenummer som består av bokstaver og tall, beskrivelse som er en kort tekst som beskriver varen, pris som er et heltall, merkenavn som er en tekst som inneholder merke til varen, vekt i kilogram som et desimaltall, lengde som et desimaltall, høyde som et desimaltall, farge beskrevet som tekst, antall på lager som aldri skal være mindre enn null, kategori som et heltall mellom en og fire og rabatt som prosent (aldri mindre enn null eller mer enn 100). I denne klassen har jeg get-metoder for alle atributtene som returnerer verdien dens. Også har jeg set-metoder for pris, antall på lager og rabatt prosenten, fordi disse ville jeg at man kan endre verdien på. Jeg har også en metode som heter itemClone som kloner en vare. Denne metoden bruker jeg når jeg lager en dyp kopi av vare registeret i ItemRegistry klassen. Og til slutt har jeg en toString metode som returnerer en tekstrepresentasjon av klassen. I denne klassen har jeg brukt throw new IlligalArgumentException (IAE) for alle attributtene i konstruktøren som skal utføres dersom attributtene ikke er en gyldig verdi. For eksempel må alle varer ha et varenummer, så hvis det ikke har et varenummer skal den throw new IAE. Eller hvis rabatt prosenten er under null eller over 100.

ItemRegistry klassen representerer vare registeret. Klassen har en ArrayList som heter items hvor alle varene lagres. Denne klassen tilbyr en metode som returnerer alle varer klona i en dyp kopiert ArrayList. Klassen tilbyr også en metode for å hente størrelsen på items ArrayListen, som vil være det samme

som antall varer. Klassen tilbyr en metode som returnerer en vare gitt varenummeret, og en metode gitt beskrivelsen til en vare. Klassen tilbyr også en metode for å legge til en ny vare med parametere for alle variablene i vare klassen, i tillegg til en metode for å slette en vare fra registeret gitt varenummeret. Klassen har en metode for å endre antall varer av en type på lager, en for å endre pris på en vare og en for å endre rabatt på en vare. Klassen har en metode som returnerer alle varenumrene i en ArrayList, og en metode som returnerer alle beskrivelsene i en ArrayList. I tillegg har klassen en toString metode som bruker toString metoden fra Item klassen til å liste opp alle varene i vare registeret.

Main klassen er klassen hvor main metoden er, i tillegg til mage metoder som endte printer ut noe, eller leser inn input fra brukeren. I main metoden kjører jeg en metode som legger inn test data i vare registeret og en metode for menyen. Meny metoden printer ut en liste med alle valgene du har, så kan brukere skrive inn tall for å gjøre valg i menyen. Videre utfører meny den metoden du har valgt. Main klassen har en metode for å få se alle varene i registeret. Klassen har en metode som returnerer aller varenumrene. Klassen har også en metode som kan brukes til å søke etter varer basert på varenummer eller beskrivelse. Denne metoden bruker to andre metoder; en som henter en vare basert på varenummer og en som henter en vare basert på beskrivelse. Klassen har en metode for å legge til en vare i registeret. Denne metoden bruker fire andre metoder for å gjøre det. Den bruker en metode for å lese inn heltall input fra brukeren, en for å lese inn desimaltall input, en for å lese inn tekst input, en for å lese inn rabatt (rabatt kan ikke være mindre enn null eller større enn 100) og en for å lese inn heltall for kategori (kategori kan ikke være mindre enn en eller større enn fire). Videre har klassen en metode for å slette en vare fra vareregisteret. Klassen har også metoder for å endre antall av en vare på lager, en for å øke og en for å minske. I tillegg har klassen en metode for å endre pris på en vare, og en metode for å endre rabatten på en vare. I tillegg har klassen flere final Stringer som lagrer tekst som er brukt på flere steder i denne klassen, så man slipper å skrive samme strenger om igjen. For alle metodene bruker jeg try catch for å hindre at brukeren skriver inn feil type data. I tillegg til if setninger for å hindre at brukere skriver inn ugyldige verdier for en variabel. Programmet har også whilelooper, så brukeren får prøve på nytt hvis de skriver inn feil type data eller en ugyldig verdi.

4.3 Brukerveiledning

Når du starter programmet, vil du få ni forskjellige valg å velge mellom. Her kan du velge hvilke operasjoner du vil utføre. Etter du har skrevet inn noe i terminalen må du trykke enter å utføre. I hele program vil du få opp tekst på terminalen som floteller deg hva du må gjøre. Hvis du skriver noe feil får du en melding i terminalen om hvilke verdier som er lov å skrive inn, og du vil få nye sjanser til å skrive inn gyldig input. For operasjonen slett en vare kan du når som helst skrive 1, og du vil komme tilbake til menyen. Alle andre operasjoner må du utføre for å komme tilbake til menyen. Hvis du angrer kan du da utføre operasjonen igjen hvor du endrer endringene dine tilbake til sånn det vare. Hvis du legger til en vare du angrer på kan du slette den igjen. Når du er i menyen, kan du trykke 9 for å avslutte programmet.

4.4 Verktøy

Det er brukt SonarLint for å gjøre programmet mer lesbart og robust, i tillegg har jeg også testet alle funksjonene programmet har å tilby for å sjekke om du kan skrive noe feil som krasjer programmet.

Jeg brukte en plugin som het CheckStyle-IDEA, hvor jeg valgte og brukte Google sin standard for kodestil i programmet. Ved å rette opp i alle feilmeldingene CheckStyle pluginnen ga meg sørget jeg for at koden min var i henhold til Google sin kodestil standard.

4.5 Idiot-sikkert

Et mål var å gjøre programmet så idiot-sikkert og brukervennlig som mulig. Jeg har for alle inputer brukt try catch for å hindre at brukeren skriver inn feil datatype. Det er brukt if setninger for å hindre at brukere skriver inn ugyldig verdi for en variabel. Det er gode beskrivende feilmeldinger som printes til terminalen om man skriver inn feil datatype eller ugyldige verdier. I dissen meldingene står det for eksempel hvilke tall programmet aksepterer. Programmet bruker også whileloops for å gi brukeren nye sjanser om de skriver feil datatype eller ugyldig input.

5 DRØFTING

Jeg er fornøyd med at jeg valgte å bruke Intellij til dette prosjektet, og SonarLint og CheckStyle-IDEA fungerte også veldig fint. Det var lurt å lage et klassediagram tidlig i prosessen for å få litt oversikt over hvilke klasser og metoder jeg trengt. Jeg vil si jeg er fornøyd med kildene jeg har brukt. Når jeg skulle finne ut av noe prøvde jeg alltid å bruke flere kilder, og se om det kildene sa stemte overens. Det jeg angrer mest på under dette prosjektarbeidet er at jeg jobbet mye selvstendig. Jeg burde heller samarbeidet mer med andre i klassen, og spurt de om ting hvis det var noe jeg lurte på.

Jeg tror jeg løste oppgaven ganske bra, koden min følger ihvertfall kravene til mappevurderinga ganske bra. Hadde jeg skulle jobbe mer med programmet ville jeg nok prøvd å strukturere metodene mine litt bedre, så de er lettere å lese.

Det virker som koden har en bra kodestil. Alle metoder og klasser har javadoc. Koden i alle klassene er uniform og relativt lett å lese, bortsett fra noen metoder som ble litt lange å mange if setninger og looper inni værandre, men programmet er ihvertfall i henhold til SonarLink sin kodestandard.

6 KONKLUSJON – ERFARING

Prosjektet har vært veldig lærerikt. Det har vært en veldig fin måte å få prøvd ut alt jeg har lært i år, og utvikle evnene mine videre. Vansklihetsgraden til oppgaven var helt passe. Jeg vil si at klient-klassen og enum klassen kanskje var det vanskeligste. Jeg syntes jeg løste oppgaven greit og fulgte kravspesifikasjonene, men det er fortsatt mye forbedringspotensial, spesielt med tanke på finpussing av metoder. Hvis jeg kunne begynt på nytt ville jeg nok ha jobba mer tidlig i prosjektet, så jeg slapp å jobbe så mye tett på innleveringsfristen. Jeg ville også ha prøvd å jobbe mindre selvstendig. Programmet kunne nok vært mye bedre om jeg hadde samarbeidet mer med medstudenter, og få hjelp av dem om det var noe jeg lurte på. Jeg burde nok også ha gått igjennom koden min med er læringsassistent litt mer enn en gang og litt seinere i prosjektet enn det jeg gjorde.

7 REFERANSER

Janssen, T. (2022, 8.januar)

OOP Concept for Beginners: What is Encapsulation. Stackify.

https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/

Lalani, S. (2021, 4.juni)

Modularization in Java: What Is It?. Xperti.

https://xperti.io/blogs/modularization-in-java-what-is-it/

Pagade, G. (2022, 9.november)

Difference Between Cohesion and Coupling. Baeldung.

https://www.baeldung.com/cs/cohesion-vs-coupling#:~:text=High%20cohesion%20correlates%20with%20loose,a%20sign%20of%20low%20cohesion

Trivedi, J. (2022, 12.januar)

Association, Aggregation and Composition. C-sharpcorner.

https://www.c-sharpcorner.com/UploadFile/ff2f08/association-aggregation-and-composition/

Udacity. [1]. (2015, 23.februar)

Coupling Between Classes – Intro to Java Programming. Youtube.

https://www.youtube.com/watch?v=P3mVhTQI8F8

Udacity. [2]. (2015, 23.februar)

Coupling and Cohesion. Youtube.

https://www.youtube.com/watch?v=Df0WVO-c3Sw