| Kandidatnummer(e)/Navn: | | | |
| --- | --- | --- | --- |
| Ola Henrik Otterlei Navelsaker | | | |
| Dato: | Fagkode: | Studium: | Ant sider/bilag: |
| 23.11.2022 | IDATG1001 | IDATG | 7 / |

|  |
| --- |
| Faglærer(e) : |
| Kiran Bylappa Raja |

|  |
| --- |
| Tittel : |
|  |

|  |
| --- |
| Sammendrag: |
| Skriv et kort sammendrag av hva du/dere har utviklet og hva denne rapporten handler om |

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

**INNHOLD**

[1 SAMMENDRAG 1](#_Toc121003816)

[2 TERMINOLOGI 1](#_Toc121003817)

[3 INNLEDNING – PROBLEMSTILLING 1](#_Toc121003818)

[3.1 Bakgrunn/Formål og problemstilling 1](#_Toc121003819)

[3.2 Avgrensninger 2](#_Toc121003820)

[3.3 Begreper/Ordliste 2](#_Toc121003821)

[3.4 Rapportens oppbygning 3](#_Toc121003822)

[4 BAKGRUNN - TEORETISK GRUNNLAG 3](#_Toc121003823)

[5 METODE – DESIGN 3](#_Toc121003824)

[6 RESULTATER 4](#_Toc121003825)

[6.1 Beskrivelse av programmet 5](#_Toc121003826)

[6.1.1 Oversikt 5](#_Toc121003827)

[6.1.2 Product klassen 5](#_Toc121003828)

[6.1.3 Register klassen 5](#_Toc121003829)

[6.1.4 ProductRegisterUI klassen 5](#_Toc121003830)

[6.2 Designvalg 5](#_Toc121003831)

[6.3 Robusthet 6](#_Toc121003832)

[6.4 Brukervennlighet 6](#_Toc121003833)

[6.5 Dokumentasjon 6](#_Toc121003834)

[6.6 DRØFTING 7](#_Toc121003835)

[6.6.1 Robusthet 7](#_Toc121003836)

[6.6.2 Brukervennlighet 7](#_Toc121003837)

[6.6.3 Problem? 7](#_Toc121003838)

[7 KONKLUSJON - ERFARING 7](#_Toc121003839)

[8 REFERANSER 8](#_Toc121003840)

[9 VEDLEGG 8](#_Toc121003841)

# SAMMENDRAG

[Et kortfattet sammendrag (abstrakt) av rapporten; om hensikt, oppgave, omfang, framgangsmåte, resultater og konklusjoner.

**Kommentar**: Et sammendrag er som regel lurest å skrive til slutt. Gi her en kort oppsummering av hva rapporten inneholder. For eksempel : Denne rapporten er besvarelsen til obligatorisk innlevering i faget IDAT... Programmering…..]

# TERMINOLOGI

[Definisjoner, begreper og symboler som kan være ukjente for leseren. (Bruk stilen ”Definition”, som vist nedenfor). Utelat dette kapittelet dersom du ikke anvender begreper og symboler som det er behov for å definere.]

UML Unified Modeling Language

IDE Integrated Development Environment

# INNLEDNING – PROBLEMSTILLING

## Bakgrunn/Formål og problemstilling

[Dette er første kapitlet i den faglige rapporten. Det bør behandle bakgrunnen for oppgaven, eventuell oppdragsgiver, problemstillingen og/eller oppgaven som skal løses – og omfanget eller avgrensningen av oppgaven.

**Kommentar**: Det er her du/dere skal presentere selve produktet/problemstillingen som skal løses og eventuelle avgrensninger som gjøres. Merk at det er på den problemstillingen du/dere definere her som resultatdelen og konklusjonen skal vise en løsning for. Dersom oppgaven har fått utdelt en kravspesifikasjon, trenger ikke hele kravspesifikasjonen gjengis her. Henvis i så fall til kravspesifikasjonen, og skriv et sammendrag av kravspesifikasjonen her. Er kravspesifikasjonen relativt kort (1-2 sider) kan den gjerne gjengis i sin helhet her.

Bruk her gjerne **UML-diagrammer** som **Use-Case**, **Aktivitetsdiagram** osv for å beskrive krav til funksjonalitet (NB! Uten å dra inn hvordan du/dere har løst det.)]

Denne oppgaven er gitt som en mappeoppgave i faget IDATG1001 med tre deler hvor det skal utvikles en programvare som vil benyttes av et varehus – Smarthus AS. Smarthus AS Leverer hovedsakelig varer til bygg-industrien, så typiske varer er laminatgulv, dører og vinduer, lister og annet trevirke. Løsningen skal bestå av et tekstbasert brukergrensesnitt og et register som lagrer informasjon om varene.

Koden skal verifiseres med CheckStyle-plugin for IntelliJ og bruke kodestilen til Google. Koden skal være på engelsk og alle klasser, metoder og variabler skal ha gode, beskrivende navn som gjenspeiler hvilken tjeneste en metode tilbyr, eller hvilken verdi variablene representerer/holder på.

Det skal implementeres en klasse som representerer en vare og holde på følgende informasjon for en vare: varenummer, beskrivelse, pris, merkenavn, vekt, lengde, høyde, farge, antall på lager og kategori. I tillegg skal det implementeres et vareregister som holder på en eller flere varer. Vareregisteret skal ha de nødvendige metodene slik at brukergrensesnittet kan ha følgende funksjonalitet: Skrive ut alle varer på lageret, søke etter en gitt vare basert på varenummer/beskrivelse, legge til en ny vare i registeret, endre varebeholdningen for en vare, slette en vare fra varelageret, endre rabatt/pris/varebeskrivelse for en vare.

## Avgrensninger

[Er det gitt noen avgrensninger/begrensninger i oppgaven? Beskriv i så fall disse her.]

Programvaren må utvikles i programmeringsspråket Java, og koden må skrives på engelsk. Brukergrensesnittet for programvaren må være tekstbasert, vanligvis ville et mer brukervennlig brukergrensesnitt være å foretrekke.

## Begreper/Ordliste

[Når man utvikler programvare for en kunde, er det viktig å etablere en felles forståelse for begreper/terminologi/ord som benyttes av/hos kunden. Det er derfor svært vanlig å lage en "ordliste" og/eller en "Domene modell". Denne ordlisten er også et svært godt utgangspunkt for å finne frem til hvilke mulige **klasser** det kan være aktuelt å implementere i løsningen. Bruk tid på denne slik at du har en god forståelse for begrepene.]

| **Begrep (Norsk)** | **Begrep (Engelsk)** | **Betyding/beskrivelse** |
| --- | --- | --- |
| Produkt | Product | Varehuset skal håndtere produkter inn og ut av et lager |
| Lager | Storage | .... |
|  | Accessor | En type metode som har til hensikt å hente et felt, gjerne privat, tilhørende objektet og returnere det. Gjør at en kan beskytte feltene ved å bruke private. |
|  | Mutator | En type metode som har til hensikt å endre et privat felt tilhørende objektet. |
| Felt | Field | Et felt er en karakteristikk av en klasse. Eksempel på felter i Product klassen er: productNumber, price, description, category. Felter holder på verdier om objektet. |
| Konstruktør | Constructor | Oppretter en instanse av et objekt av en klasse. |
| Pris | Price | Prisen til produktet. Produktet har et felt som heter Price. |
| Beskrivelse | Description | Beskrivelse av produktet. Produktet har et felt som heter Description |
| Produkt nummer | Product number | Produkt nummeret til produktet |
| Produkt spesifikasjoner | Product Specifications | De fysiske dimensjonene til et produkt + fargen. ProductSpecifications klassen holder på dimensjonene til et produkt. |
| Nøkkel | Key | Key blir brukt ved lagring av data i et HashMap. Nøkkelen må være unik og kan brukes til å hente data som er assosiert med den spesifikke nøkkelen. |
| Bibliotek | Library | Libraries inneholder klasser og funksjoner som kan brukes ved å importere de. |
| Programvareutvidelse | Plugin | Plugins gir ekstra funksjonalitet til utviklermiljøet, som for eksempel en automatisk sjekk av kode. |

## Rapportens oppbygning

[I vitenskapelige rapporter er det svært vanlig å gi et sammendrag her om hvordan rapporten er bygget opp. Typisk "]

# BAKGRUNN - TEORETISK GRUNNLAG

[Oppgaver og problemstillinger står i en sammenheng. Denne delen skal vise at en har oversikt over denne sammenhengen, at en er eller har gjort seg kjent med tidligere resultater og andres forslag til eller forsøk på løsninger. Det er altså tale om å gi et faglig underlag for ens eget arbeid, evt. en beskrivelse av teoretiske forutsetninger, med referanse til litteratur og andre kilder en støtter seg til.

**Kommentar:** Presenter den teorien som er relevant for de vurderinger som skal gi en god løsning på problemstillingen, som for eksempel teori rundt hvilke metoder som benyttes for å analysere kravspesifikasjon og identifisere gode kandidater til klasser og objekter. Det viktige her er å få fram det teoretiske grunnlaget du/dere senere skal bruke til å vurdere og argumentere for at din foreslåtte løsning er utviklet etter gode designprinsipper og kvalitetskriterier.

Alle vurderinger du/dere gjør senere i besvarelsen skal ha referanse til dette kapittelet. Det er særdeles viktig å ha tydelige referanser til de kildene du/dere bruker når du/dere skriver dette kapittelet. All teori du/dere beskriver her skal altså ha en referanse, og denne skal skrives inn i teksten.

**Eksempel:**

”…et viktig moment ved analysen av problemstillingen er å identifisere kandidater til objekter som senere danner grunnlag for klassene. En mye benyttet metode i følge læreboka [1]…..etc.”

Tilsvarende skal det under referanser være et tall som ramser opp forfatter, årstall, tittel på bok eller artikkel osv. (se punktet om referanser). Forelesninger kan også refereres til, da med tittel på fag og navn på foreleser.]

Typiske teorier i programmering:

* Hver klasse, **et** ansvar/en rolle
* Hver metode kun **en** oppgave
* Felt i klasser **skal** være private
* Modularisering og abstrahering
* Osv.

Cohesion

Coupling

En metode en oppgave

En klasse et ansvar

Modularisering og abstrahering

Positive and negative testing?

# METODE – DESIGN

[Denne delen skal redegjøre for hvordan man planla å gå fram / har gått fram for å løse oppgaven og sannsynliggjøre framgangsmåten. Framgangsmåten kan være en utviklingsoppgave, for eksempel utvikling av et datasystem, databasesystem, en grafikkrutine, et kontrollprogram osv. Ta da også med valg av utviklingsmetode, framdriftsplan, organisering og rapportering, hvordan arbeidet utprøves, kontrolleres og korrigeres, om systemdokumentasjon og brukerveiledning, om overlevering til oppdragsgiver og avtalte kriterier for fullført oppgave osv.]

[Beskriv også hvilke verktøy du/dere har benyttet for å løse oppgaven, hvilket utviklingsmiljø du/dere har jobbet i (BlueJ, Netbeans, Eclipse, IntelliJ, CheckStyle, SonarLint Bitbucket, Git, Wiki, Issue-tracking som JIRA el.l. osv.]

Valgte å bruke IntelliJ og java versjon 15?

# RESULTATER

[Dette er rapportens største del. Ved oppgave som omfatter teorigjennomgang, analyse eller teknisk/vitenskapelig undersøkelse: resultater av undersøkelsen - uten vurdering (disse kommer under drøfting). Ved oppgave som omfatter utviklingsoppgave: beskrivelse av løsning, bruksmåte, installasjon, drift og sikkerhet.

Eventuelle UML-diagrammer som klassediagrammer, sekvensdiagrammer osv med tilhørende forklaringer/begrunnelser for valg kan inngå her.

Det er også her viktig å få frem ulike løsninger man har vurdert i prosessen for å komme frem til endelig valgt løsning. Begrunnelse skal gis for hvorfor den ene løsningene ble valgt fremfor den andre.

Beskriv også det endelige resultatet; hva ble til slutt produsert/utviklet i prosjektet? En kort brukerveiledning kan også være på sin plass. Er løsningen stor, kan brukerveiledningen legges ved som et vedlegg og henvises til fra dette kapittelet.]

Hva ble endret i forhold til opprinnelige design contra det siste designet som ble innlevert (**refaktorering**)? Her kan dere benytte klassediagram, sekvensdiagram, osv for å illustrere endringer underveis.

Hva som ble gjort for å oppnå

* robust programvare (bruk av testing, debugging osv, SonarLint)
* godt dokumentert kode (CheckStyle)
* «idiot-sikker» programvare (fail-safe)
* bruker-vennlig design og graceful termination for å ikke krasje program tilfeldig

**Kommentar**: Det er her du/dere skal bearbeide arbeidet ut fra de teorier og metoder som er nevnt i de to foregående kapitlene, og som kan gi et forslag til løsning på den problemstillingen som er definert i innledningen. Merk at det da er nødvendig å gjøre en del henvisninger tilbake til disse to kapitlene for at den som leser rapporten skal kunne følge bakgrunnen for de vurderinger du/dere nå gjør. Husk at du/dere aldri må gjøre vurderinger og analyser uten at dette er dokumentert i teori kapittelet. Ubegrunnet synsing er fullstendig verdiløst. I en oppgave som denne der selve læreprosessen er vesentlig, bør du/dere være flinke til å formulere de tanker og vurderinger som gjøres i selve argumenteringen, altså beskrive både prosess og løsning. Som en huskeregel kan du/dere tenke at normalt har man en tendens til å ikke skrive ned nok rundt selve prosessen med argumentering.]

[**Konkret for Programmeringsemner**:

Her beskriver du/dere hvilke kandidater til klasser du/dere har funnet med bakgrunn i kravspesifikasjonen og begreps-kapittelet. Beskriv også hvilke funksjonalitet som det er stilt krav til i kravspesifikasjonen og hvordan denne er løst. For hver klasse du/dere har identifisert, skriv kort hva som er klassens ansvar/rolle i systemet (gjerne i form av en tabell).]

## Beskrivelse av programmet

### Oversikt

Programmet har hovedsakelig tre klasser: Product, Register og ProductRegisterUI. Product klassen representerer et produkt som bedriften har på lageret og har metoder for å innhente og endre nødvendig informasjon om produktet. Register klassen representerer selve lageret i bedriften ved å lagre og håndtere flere objekter av Product klassen. ProductRegisterUI klassen inneholder det tekstbaserte brukergrensesnittet som hjelper brukere med manipulering av data. For at Product klassen ikke skal holde på for mange felter blir det tatt i bruk en ekstra klasse, «ProductSpecifications», som holder på vekt, lengde, høyde og farge. Denne klassen bruker setters, som Product klassen, slik at vekt, lengde og høyde ikke kan være negative tall. I tillegg holder enum klassen, ProductCategory, på en liste av godkjente kategorier.

### Product klassen

Product klassen inneholder informasjon om: produkt nummer (productNumber), beskrivelse (description), pris (price), merke (brand), vekt (weight), lengde (length), høyde (height), farge (color), antall varer (stock), og kategori (category). Klassen har også de accessor og mutator metodene som er nødvendige for å oppnå kravspesifikasjonene, i tillegg til en metode som returnerer all informasjonen om et produkt slik at det er mulig å gi brukeren en tekstbasert representasjon av produktet. I konstruktøren til klassen blir enkelte setters benyttet slik at verdier som «price» og «stock» ikke kan bli satt til å være negative tall. Det samme blir gjort med lengde, høyde og vekt, men dette håndteres i en annen klasse kalt «ProductSpecifications». For å være sikker på at riktige kategorier blir opgitt benyttes en enum klasse kalt «ProductCategories». Når et objekt av Product klassen lages, vil kun kategorier som finnes i ProductCategories kunne benyttes.

### Register klassen

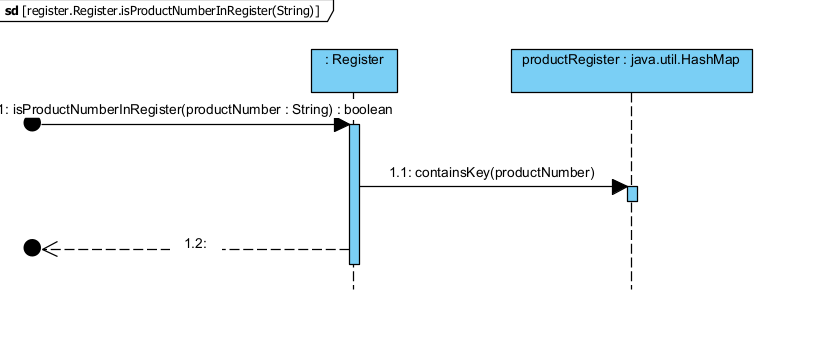
Register klassen lagrer flere produkt i et HashMap hvor productNumber feltet blir brukt som Key og Product objektet blir brukt som Value i HashMappen. Manipulering av data i registeret skjer i denne klassen. Klassen har metoder for å legge et product i HashMappen, finne et produkt gjennom beskrivelsen/produktnummer, skrive ut registeret, fjerne et produkt og sjekke om et produktnummer er i registeret. I tillegg har klassen accessor metoder for å returnere en liste med kategorier, finne kategori og en iterator for gjenbruk.

### ProductRegisterUI klassen

Alt som har med det tekstbaserte brukergrensesnittet og brukere å gjøre, skjer i denne klassen. Den har metoder som start(), searchProductMenu() og changeProductInformationMenu() som skriver ut brukergrensesnittet og bruker metoden getMenuChoice() for å hente hvilket meny element brukeren velger. Klassen inneholder alle metodene brukeren kan benytte i programmet i henhold til programspesifikasjonene. Det betyr at brukeren kan endre Price, Stock og Description feltene til produktet, han kan søke etter et produkt gjennom beskrivelsen eller produkt nummer, fjerne et produkt fra registeret, legge til et produkt til registeret og skrive ut alle produkt i registeret.

## Designvalg

For å representere selve lageret valgte jeg å bruke HashMap i stedet for å bruke ArrayList. HashMap lagrer data ved å assosiere en verdi (produktet i dette tilfellet) med en unik «Key» (produkt nummeret), sammenlignet med ArrayList som lagrer data ved å legge data i en liste og holde på indeksen til objektet som blir lagt til listen. Det at «Key»en må være unik gjør at en ikke kan legge til duplikate «Key»er, men man kan legge til duplikate verdier (Produkt) med forskjellig «Key». For mitt program er dette en fordel, da det ikke er nødvendig å lagre duplikate verdier og gjør at det blir enkelt å sjekke om et produkt allerede finnes i registeret ved å bruke containsKey() funksjonen innebygd i HashMap biblioteket.



Figur 1: isProductNumberInRegister()

Sekvensdiagrammet illustrerer hvor enkelt det er å bruke HashMap til dette formålet. containsKey() metoden returnerer true eller false basert på om produkt nummeret som blir sendt med er i registeret eller ikke. Hvis dette skulle blitt gjort ved hjelp av ArrayList, måtte man ha laget en metode som sjekker om produkt nummeret er likt det produkt nummeret som blir sendt med, for hvert objekt i listen til den finner et likt produkt nummer.

## Robusthet

Målet, når det gjelder robusthet, er å sørge for at programmet ikke kan krasje. Kanskje den største trusselen når det gjelder dette er personen som bruker programmet. Derfor blir det benyttet input validation på alle verdier brukeren skriver inn i programmet. Input validation er prosessen hvor en sjekker at det som blir sendt inn til programmet er riktig skrevet inn, og at det ikke vil forårsake en feil. Programmet inneholder funksjoner som intChecker() og floatChecker() for å verifisere og sørge for at brukeren skriver inn gyldig tall der det er nødvendig. Hvis brukeren fikk lov å skrive inn hva som helst i et felt som krever datatypen int, hadde programmet krasjet og data kunne gått tapt.

I tillegg til å sørge for at programmet ikke krasjer, brukes input validation for å sørge for at brukeren skriver inn verdier som gir mening. Når brukeren forsøker å legge inn et nytt produkt, blir det sjekket at produkt nummeret ikke er tomt, at produkt nummeret ikke allerede finnes i lageret, og at tallverdier som pris og antall på lageret ikke er negative siden dette ikke hadde gitt mening.

En viktig del av robusthet er at brukeren ikke klarer å sette seg fast i programmet slik at han ikke kommer seg videre uten å avslutte og gjenåpne programmet. For å sørge for at dette ikke skjer er det gjort slik at brukeren har et bestemt antall forsøk på å skrive inn det som kreves av funksjonen for å komme seg videre. Hvis brukeren overstiger dette antallet forsøk vil programmet avbryte funksjonen, skrive ut en feilmelding til brukeren, og så gå tilbake til hovedmenyen for at brukeren kan fortsette å benytte seg av programmet.

Har benyttet SonarLint pluginen som en verifisering på at koden ikke inneholder unødvendige ting som kan redusere effektiviteten til programmet. SonarLint ble også brukt til å oppdage potensielle farer slik at det ikke oppstår problemer senere i prosessen hvor det er mer tidkrevende å fikse slike problemer.

## Brukervennlighet

For å sørge for at brukeren enkelt skjønner hva som skjer og hva som kreves av han når han bruker programmet, er det viktig å gi brukeren gode og deskriptive tilbakemeldinger mens han bruker programmet. Hvis brukeren gjør noe feil vil det komme en ny tilbakemelding som forklarer hva som kreves for det steget i funksjonen brukeren har valgt. Det gis også tilbakemelding når det ikke går som planlagt ved å bruke feilmeldinger med rød skrift, for eksempel når brukeren prøver å velge et menyvalg som ikke eksisterer, eller prøver å fjerne et produkt som ikke finnes i lageret.

Der det er enkelt å sette seg fast vil brukeren bli sendt tilbake til hovedmenyen hvis han ikke greier å forsyne programmet med den nødvendige informasjonen innen et gitt antall forsøk. Brukeren kan da fortsette å bruke programmet etter at han setter seg fast og trenger ikke å avslutte og gjenåpne programmet.

## Dokumentasjon

God dokumentasjon er viktig for at programmet skal kunne brukes effektivt og opprettholdes. Det er ikke nødvendigvis personen som utvikler programmet som skal vedlikeholde programmet, og da er det en stor fordel for personen som vedlikeholder programmet at det er godt dokumentert og at metoder og felt har gode og beskrivende navn.

Alle metoder, klasser og konstruktører har derfor Javadoc som kort forklarer hva hensikten med metoden/klassen/konstruktøren er, og hva som blir returnert hvis dette er aktuelt. Metoder som er lange og vanskelig å lese har kommentarer inne i metoden for å forklare hva som skjer og gjør det enklere for personen som leser koden å skjønne hva som faktisk skjer i metoden.

Har brukt Google style gjennom CheckStyle pluginen for å sikre at koden er skrevet på riktig vis.

## DRØFTING

[Vurdering av metode og oppnådde resultater. Begrensninger, endringer eller avvik i prosjekt i forhold til plan / opprinnelig problemstilling - mulige feilkilder. Resultatenes betydning.

**Kommentar**: Her kan man for eksempel gjøre seg tanker rundt kvaliteten av det arbeidet som er nedlagt. Er de kildene du/dere bruker pålitelige, er det sprik mellom forskjellige kilder (og i så fall hvorfor), er det andre forhold som kan være med å gjøre noen av de vurderinger og valg du/dere har gjort usikre?]

[Konkret for programmeringsemner: Her oppsummerer du/dere oppgaven. Hvor langt kom du/dere (resultat)? Hva fikk du/dere ikke gjort i forhold til oppgaveteksten ? Hva var de store utfordringene/problemene du/dere møtte, etc..

Spesielt viktig er det å drøfte din egen løsning i forhold til det du har lært om gode prinsipper for programmering (robust kode, kodestil, designprinsipper osv) som beskrevet i teori-kapittelet]

### Robusthet

Når jeg skulle sjekke om programmet var sikkert og at det fungerte som det skulle, gikk jeg manuelt gjennom hver funksjon brukeren kunne nå og forsøkte å krasje programmet ved å gi dårlig inputs samtidig som jeg sjekket at programmet gjorde det som var forventet når riktig input ble gitt. Dette er tidkrevende og ikke veldig effektivt. Allerede for et lite program som dette blir det fort tungvint å manuelt sørge for at programmet funker som det skal når noe endres, men for et større program med flere funksjoner som må sjekkes er dette helt urealistisk å gjennomføre. Det jeg burde gjort, og ville gjort for å forbedre programmet er å implementere automatiske tester som kan kjøres for å verifisere at programmet fungerer som forventet når både riktig og feil input blir gitt, dette er et konsept kalt «positive and negative testing».

For at brukeren ikke skal kunne klare å sette seg fast hadde jeg noen plasser implementert en fail-safe hvor programmet ville gå tilbake til hovedmenyen hvis brukeren ikke klarte å gi programmet den nødvendige informasjonen innen et gitt antall forsøk. Denne typen fail-safe kunne vært implementert flere steder i koden, for eksempel når brukeren legger til et produkt i lageret.

### Brukervennlighet

For å øke brukervennligheten av programmet skulle jeg gjerne implementert en måte for brukeren å gå tilbake et steg i menyen.

### Problem?

# KONKLUSJON - ERFARING

[Overbevisninger /erfaring som en er kommet fram til på grunnlag av det presenterte materialet.

* Hva ville du ha gjort annerledes dersom du kunne begynn på nytt?
* Hva slags begrensninger kan en forvente når en bruker løsningen?
* Hva skal tas opp i fremtidige arbeid dersom du eller noen andre ville ha tatt utvikling videre?

**Kommentar**: Her skal du/dere presentere de viktigste resultatene fra arbeidet sammen med de erfaringer du/dere har opparbeidet i prosessen.]

# REFERANSER

[Forfatter, årstall, tittel på bok eller artikkel, navn på tidsskrift eller forlag/utgiver, nr. eller dato for tidsskrift, sted som det vises til eller refereres fra i oppgaven.

**Kommentar**: se eksempel under]

[Konkret for programmeringsemner: Regner med at du/dere kommer til å måtte slå opp litt i læreboka, så den er en innlysende referanse. Dersom du/dere i tillegg benytter internett, så list URL’er til sidene du/dere har benyttet.]

1. ”Objects First With Java”, Sixth edition, av Barnes og Kölling. ISBN ….
2. http://.....
3. Osv.

# VEDLEGG

[Materiell som er utarbeidet eller innsamlet i tilknytning til rapporten, men som det ikke er naturlig eller hensiktsmessig å ta inn i hoveddelen, skal tas inn som vedlegg.

Vedleggene skal være nummererte og ha en overskrift.

Har du/dere ingen vedlegg, så droppes dette kapittelet.]