| Kandidatnummer(e)/Navn: | | | |
| --- | --- | --- | --- |
| 10124 | | | |
| Dato: | Fagkode: | Studium: | Ant sider/bilag: |
| 12.12.22 | IDATT1001 | Programmering 1 | 4 / |

|  |
| --- |
| Faglærer(e) : |
| Muhammad Ali Norozi |

|  |
| --- |
| 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](#_Toc121779980)

[2 TERMINOLOGI 1](#_Toc121779981)

[3 INNLEDNING – PROBLEMSTILLING 1](#_Toc121779982)

[3.1 Bakgrunn/Formål og problemstilling 1](#_Toc121779983)

[3.1.1 Del 1 1](#_Toc121779984)

[3.1.2 Del 2 1](#_Toc121779985)

[3.1.3 Del 3 2](#_Toc121779986)

[3.2 Avgrensninger 2](#_Toc121779987)

[3.3 Begreper/Ordliste 2](#_Toc121779988)

[3.4 Rapportens oppbygning 4](#_Toc121779989)

[4 BAKGRUNN - TEORETISK GRUNNLAG 4](#_Toc121779990)

[4.1 Coupling 4](#_Toc121779991)

[4.2 Cohesion 4](#_Toc121779992)

[4.3 Design Pattern 4](#_Toc121779993)

[4.3.1 Builder Pattern 5](#_Toc121779994)

[4.4 Modularisering 5](#_Toc121779995)

[4.5 Abstrahering 5](#_Toc121779996)

[5 METODE – DESIGN 6](#_Toc121779997)

[6 RESULTATER 7](#_Toc121779998)

[7 DRØFTING 8](#_Toc121779999)

[8 KONKLUSJON - ERFARING 9](#_Toc121780000)

[9 References 9](#_Toc121780001)

[10 VEDLEGG 10](#_Toc121780002)

[Denne rapporten inneholder ferdigdefinerte **stiler** som du/dere kan benytte for de mest vanlige avsnittene. Følgende stiler er definert:

Heading 1 Overskrift på nivå 1

Heading 2 Overskrift på nivå 2

Heading 3 Overskrift på nivå 3

Brødtekst Standard tekst i et avsnitt. Benytt denne for all ”vanlig” tekst

Definition Benyttes hovedsakelig i avsnittet ”TERMINOLOGI”

References Benyttes i REFERANSER-avsnittet.

Comment Denne grønne teksten. Fjern all tekst av denne typen i rapporten.]

# 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

UP Unified Process

WMS Warehouse Management System

IDE Integrated Development Environment

# INNLEDNING – PROBLEMSTILLING

## Bakgrunn/Formål og problemstilling

Oppgaven var å utvikle en programvare som skulle brukes av et varehus. Programvaren skal kunne brukes til å håndtere varelageret til varehuset som benytter programmet. Programmet skulle til slutt bestå av et register som lagrer informasjon om de forskjellige produktene på lager, og skulle kunne brukes gjennom et tekstbasert brukergrensesnitt. Det skal være mulig å gjøre enkle operasjoner på registeret som gjenspeiler varehus funksjoner i den virkelige verden.

Programmet skulle utvikles igjennom 3 iterasjoner, en del 1, 2 og 3, hvor hver del har sine egne definerte kravspesifikasjoner.

### Del 1

I denne delen skulle man implementere en entitetsklasse som skulle representere en vare. Kravspesifikasjonen for denne delen handlet om informasjonen som skulle registreres, med spesifisert datatype.

### Del 2

I denne delen skulle man implementere et vareregister for å holde på en eller mange varer. Registeret skulle inneholde enkle funksjoner for å modifisere varene i registeret. I tillegg skulle et brukergrensesnitt implementeres for å kunne bruke registeret og funksjonene i registeret. Kravspesifikasjonen for denne delen handlet om de enkle funksjonene som registeret minst skulle ha.

### Del 3

I denne delen skulle man ferdigstille applikasjonen og endre på tidligere kode for å optimalisere den. I tillegg skulle man sette sitt eget preg på løsningen ved å frigjøre seg litt fra de tidligere nevnt kravspesifikasjonene.

[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.)]

## Avgrensninger

Oppgaven spesifiserer datatyper som felt variablene i vare klassen skal ha. Dette er for eksempel at pris skal være heltall, vare nummer skal være en streng med både tall og bokstaver, høyde skal være desimaltall osv. I tillegg skulle koden verifiseres av en Checkstyle-plugin, som sammenligner koden med en kode-standard.

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

## 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) | Betydning/beskrivelse |
| --- | --- | --- |
| Produkt | Product | Varehuset skal håndtere produkter inn og ut av et lager |
| Lager | Storage | Oppbevaringssted for varer |
| Register | Register | Liste over oppgitte varer |
| User Interface | Brukergrensesnitt | Et grensesnitt for å kommunisere mellom bruker og program |
| Iterasjon | Iteration | Utføre en handling flere ganger |
| Klasse | Class | Representerer en gjenstand, et objekt eller en funksjon i den virkelige verden |
| Datatype | Datatype | Beskrivelse av hva slags type en verdi har |
| Variabel | Variable | En verdi som kan variere |
| Kobling | Coupling | Et prinsipp som handler om koblingen mellom klasser |
| Sammenheng | Cohesion | Et prinsipp som handler om hvordan forskjellige funksjoner henger sammen i en klasse |
| Utvidelse | Plugin | En ekstern utvidelse som har en bestemt funksjon |
| Modul | Module | En pakke som inneholder klasser som henger sammen |
| Parameter | Parameter | En verdi som en metode bruker for å utføre funksjonen sin. X-en i F(x). |
| Objekt | Object | En forekomst av en klasse |
| Statisk | Static | Noe som ikke endrer seg |
| Statisk variabel | Static variable | En variabel som er delt mellom alle objekter av en klasse |
| Metode | Method | En funksjon som tilhører et objekt |
| Felt variabler | Field variables | Variablene som defineres av en klasse og som tilhører objekter av klassen |
| Design Mønster | Design Pattern | En generell mal som løser vanlige problemer |
| Abstrakt | Abstract | En klasse som har generelle felt variabler og metoder |
| Mutator metode | Mutator method | En metode av en klasse som kan endre på et felt variabel |
| Aksessor metode | Acessor method | En metode av en klasse som henter informasjon fra klassen sine felt variabler |
| Enum klasse | Enum klasse | En klasse som inneholder statiske definisjoner som tilhører klassen |
| Kompromittert | Compromised | Har blitt gjort sårbar |
| Refaktorisering | Refactoring | En endring på tidligere kode |
| Streng | String | En rekke bokstaver eller tall |

## Rapportens oppbygning

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

# BAKGRUNN - TEORETISK GRUNNLAG

## Coupling

Coupling er et abstrakt konsept som er graden av uavhengighet mellom moduler. Kan ses på som sannsynligheten for hvor mye en endring eller modifisering av en modul, påvirker hvor mye en programmerer må ta for seg en endring i en annen modul. Hvis denne sannsynligheten er høy, så er disse to modulene «highly coupled», som vil si at en endring i den ene modulen påvirker sterkt den andre modulen. Man vil ofte oppnå «loose coupling», som betyr at moduler ikke er sterkt avhengig av hverandre, og at man kan endre på koden uten å påvirke andre moduler. Dette fører til mer uavhengighet og gjør det lettere for en programmerer å implementere nye funksjoner eller fjerne eksisterende funksjoner.

Noe som påvirker coupling kan være en referanse i en klasse til en annen klasse. En intermodulær kobling oppstår når referansen skjer mellom to separate moduler. I tillegg kan kompleksiteten av koblingen være korrelert med høyere coupling. Dette vil si at en kobling fra en klasse som spesifiserer mange parametere har høyere coupling enn en kobling med få parametere. (Yourdon & Constantine, 1975)

## Cohesion

Måten man deler opp et system, i sammenheng med problem strukturen, kan påvirke struktur kompleksiteten og antall intermodulære referanser. Derfor er det viktig og ta hensyn til at man deler opp i moduler som henger godt sammen med funksjonen den ønsker å oppnå. Dette kalles cohesion. Dette gjelder også for funksjonaliteten av individuelle metoder i en klasse. I den virkelige verden kan et isolert element være funksjonelt relatert til en variert mengde av andre elementer. Dette leder ofte til en subjektiv tolkning av problem strukturen, og det oppstår derfor forskjellige løsninger på hvor mye forskjellige elementer henger sammen. Likevel kan man si at hvis alle metoder gjør den spesifikke oppgaven den er ment til å gjøre, så gir dette «høy cohesion». En ulogisk gruppering av funksjoner i en metode kan man mene er «lav cohesion».

Man ønsker ofte å oppnå høy cohesion. Dette er fordi det leder ofte til en bedre organisert struktur på koden, og betyr at det er lettere å finne frem til metodene som gjør en spesifikk oppgave. I tillegg oppnår man høy cohesion når en klasse inneholder metoder som gir logisk mening for den klassen å ha. (Yourdon & Constantine, 1975)

## Design Pattern

Et design mønster i henhold til programmering er en generell løsning til problemer som ofte oppstår under strukturering og programmering av en applikasjon. Disse er enten en forklaring eller en mal på hvordan man kan løse et problem, og er effektivt når man designer et program eller et system. Subkategorier innenfor design mønstre er for eksempel skapende-mønster, struktur-mønster, oppførsels-mønster osv. (Gamma, Helm, Ralph, & John, 1994)

### Builder Pattern

Builder pattern er et design mønster av subkategori skapende-mønster. Dette vil si at design mønsteret er en løsning på hvordan et objekt er skapt. Intensjonen med å bruke builder pattern er å separere konstruksjonen av et objekt fra representasjonen av objektet, og konstruksjon prosessen kan også derfor brukes til å lage flere representasjoner av objektet. Dette øker modulariteten ved å isolere koden for konstruksjon av objektet fra representasjonen.

Dette design mønsteret kan brukes når algoritmen for å lage et komplekst objekt burde være uavhengig av delene som utgjør objektet og hvordan den konstrueres. Det kan også brukes hvis konstruksjon prosessen tillater at fins flere forskjellige representasjoner av et objekt. (Gamma, Helm, Ralph, & John, 1994)

## Modularisering

Modularisering er en teknikk for å dele et program eller et system opp i mindre, uavhengige deler som kalles moduler. Dette innebærer å identifisere ulike funksjoner eller komponenter, og deretter skille dem ut i egne moduler, hvor hver modul har et definert formål og ansvar. Disse kan kommunisere med andre moduler gjennom intermodulære referanser. På denne måten kan man lett bytte ut eller endre enkelte moduler uten å påvirke resten av systemet.

Modularisering kan også bidra til å øke samarbeidet mellom programmerere ved å gjøre det enklere å dele og koordinere arbeidet på ulike deler av et program eller system. Dette gjør det også mulig og teste og kvalitet sikre individuelle moduler, og kan føre til økt kvalitet og stabilitet.

Dersom en del av et system blir kompromittert kan dette ha store konsekvenser for resten av systemet. Derfor kan modularisering bidra til å øke sikkerheten ved å begrense skadeomfanget dersom det skulle skje noe med en modul. Om bare en modul skulle bli påvirket, blir det lettere for en programmerer å rette opp i problemet.

Ved modularisering av et system oppnår man mer oversiktlighet, en enkel, forståelig og vedlikeholdbar kode, og mer fleksibilitet for å gjøre endringer eller legge til nye funksjonaliteter. Moduler kan også gjenbrukes i andre programmer eller systemer, som kan spare tid og arbeid. (Gosling, Bill, Guy, & Gilad, 2005)

## Abstrahering

Abstrahering i programmering referer til evnen til å samle likheter mellom ulike objekter eller konsepter og definere en generell modell eller struktur som kan følges av objektene. Dette gjøres ved å identifisere de viktigste egenskapene og metodene et objekt kan ha, og deretter lage en abstrakt klasse som inneholder disse felles egenskapene og metodene. Dette gjør det mulig for klasser som kan grupperes som et objekt av den abstrakte klassen til å arve fra denne klassen. Abstrahering gjør det mulig for programmerere å være mer generelle og fleksible, som gjør det enklere å lage og vedlikeholde koden. For eksempel så kan man ha en abstrakt klasse «Dyr», hvor klassene «Løve», «Hund» og «Elefant» arver fra «Dyr» klassen. I dette tilfellet så ville den abstrakte klassen inneholde metoder og felt variabler som er felles for alle dyr, og klasser som arver av denne klasse kan spesifisere mer konkret noen abstrakte metoder som for eksempel en løpe-metode, eller en spise-metode. (Abstraksjon (datavitenskap), 2022)

[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.

# METODE – DESIGN

Hele koden har blitt kodet i IDE’en «InteliJ». Dette er en IDE som ble utviklet for å programmere i Java. Github har også blitt benyttet for å dokumentere koden underveis, og for å gjøre sikre endringer på koden uten risikoen for at alt blir kompromittert. Dette er også viktig ved flere refaktoriseringer for å opprettholde versjonskontroll. Applikasjonen har blitt utviklet ved å følge fossefall utviklings modellen gjennom hver iterasjon av oppgaven. Framgangsmåten startet med å analysere kravene som problemstillingen spesifiserte, og deretter opprette et design i forhold til kravene. Designet ble opprettet som et klasse diagram, som gjør det veldig enkelt å implementere. Vedlegget under illustrerer det første utkastet av klasse diagrammet som ble designet etter analyse av oppgaven.

Etter implementasjonen så ble koden verifisert ved å bruke JUnit test klasser. JUnit er en plugin som kan brukes i InteliJ for å danne test klasser som kan verifisere funksjonaliteten til metoder og konstruktører av klasser. I tillegg ble det brukt en Checkstyle-plugin for å sammenligne koden med en kode-standard. Feil funnet i enten JUnit testingen eller Checkstyle ble rettet opp på før videre utvikling av programmet. Disse stegene ble utført for hver iterasjon av oppgaven helt til det ferdige produktet ble ferdig.

[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.]

# RESULTATER

## Item

Item klassen er klassen som skal representere en virkelig-verden vare. Den inneholder felt variabler som ble definert av oppgaven, i tillegg til egen definerte variabler. I tillegg har den mutator og aksessor metoder for disse variablene, og to konstruktører.

### Felt variabler

Alle definerte felt variabler er private for at man ikke skal kunne endre på variablene etter konstruksjon av objektet, med mindre man bruker de eksisterende mutator metodene. Dette fører til loose-coupling. Variabler med nøkkelordet «final» gjør at variabelen ikke kan endres på etter konstruksjon av objektet.

itemNumber En streng variabel som representerer vare nummberet til varen. Denne variabelen har datatypen String fordi et varenummer skal kunne inneholde en kombinasjon av bokstaver og tall. Den har også nøkkelordet final fordi varenummeret skal være ferdig definert.

description En streng som representerer vare beskrivelsen. Denne variabelen har datatypen String fordi en beskrivelse er en tekst.

brandName En streng som representerer merkenavnet til varen. Denne variabelen har datatypen String fordi et navn er en tekst. Den har også nøkkelordet final fordi merkenavnet til en vare endrer seg ikke over tid.

price

A screenshot of a computer

Description automatically generated with medium confidence

Her ser vi at «Item» klassen har blitt definert med felt variablene som ble spesifisert av oppgaven, med relevante mutator og aksessor metoder. Det ble bestemt at variablene for forklaring (desc), antall varer på lager (warehouseStock) og pris (price) skulle ha mutator metoder. Dette er fordi det skal være mulig å endre på disse variablene siden de kan endres over tid, og var de mest åpenbare å implementere med engang. Andre felt variabler skal ikke kunne endres dersom de defineres ved konstruksjon av objektet, og det gir ikke logisk mening at de endres over tid. For eksempel, så endrer ikke høyden eller lengden på et objekt over tid, og derfor er de fast bestemt med engang. I tillegg til «Item» klassen så ble det fort tydelig at det kunne implementeres enum klasser for både kategori (Category) og color (Color).

[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).]

# 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]

# 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.]

# References

*Abstraksjon (datavitenskap)*. (2022, November 22). Hentet fra Wikipedia: https://en.wikipedia.org/wiki/Abstraction\_(computer\_science)

Gamma, E., Helm, R., Ralph, J., & John, V. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Addison-Wesley.

Gosling, J., Bill, J., Guy, S., & Gilad, B. (2005). *The Java Language Specification, Thrid Edition.* Boston: Addison-Wesley.

Yourdon, E., & Constantine, L. L. (1975). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* New York: YOURDON inc.

[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.]