| Kandidatnummer(e)/Navn: | | | |
| --- | --- | --- | --- |
| 10120 | | | |
| Dato: | Fagkode: | Studium: | Ant sider/bilag: |
|  | IDATT1001 | Bachelor i ingeniørfag, data - Trondheim | 7 / |

|  |
| --- |
| Faglærer(e) : |
| Muhammad Ali Norozi  Arne Gerhard Styve  Kiran Bylappa Raja  Surya Bahadur Kathayat |

|  |
| --- |
| Tittel : |
| Warehouse management system |

|  |
| --- |
| Sammendrag: |
| Dette prosjektet innebar å lage et “Warehouse management system” for det fiktive selskapet Smarthus AS. Dette er et program som lagrer og endrer på informasjon om et varehus og produktene det har på lager. Dette programmet skulle ha et brukergrensesnitt som ansatte kan bruke for å utføre endringer på systemet. Denne rapporten dokumenterer det teoretiske grunnlaget for hvordan applikasjonen ble utviklet, samt utviklingsprosessen og diskusjon om det endelige resultatet. |

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

INNHOLD

[1 SAMMENDRAG 1](#_Toc116650246)

[2 TERMINOLOGI 1](#_Toc116650247)

[3 INNLEDNING – PROBLEMSTILLING 1](#_Toc116650248)

[3.1 Bakgrunn 1](#_Toc116650249)

[3.2 Formål og problemstilling 1](#_Toc116650250)

[3.3 Avgrensninger 1](#_Toc116650251)

[3.4 Begreper/Ordliste 1](#_Toc116650252)

[3.5 Rapportens oppbyggning 2](#_Toc116650253)

[4 BAKGRUNN - TEORETISK GRUNNLAG 2](#_Toc116650254)

[5 METODE – DESIGN 3](#_Toc116650255)

[6 RESULTATER 3](#_Toc116650256)

[7 DRØFTING 4](#_Toc116650257)

[8 KONKLUSJON - ERFARING 4](#_Toc116650258)

[9 REFERANSER 4](#_Toc116650259)

[10 VEDLEGG 5](#_Toc116650260)

[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

IDE Integrated development environment

VS Code Visual Studio Code

Plugin Tilleggsprogram som utvider funksjonaliteten til en applikasjon.

Setter Metode som endrer på verdien til et felt.

Getter Metode som henter verdien til et felt.

Javadoc Standardformat for å beskrive Java-kode.

JDK Java development kit

Exception/unntak En feilmelding som sier noe om hva som har gått galt. Denne kan bli plukket opp av de som kaller på metoden som utløser den, og de kan bruke den for å håndtere hva som skjer i et slikt tilfelle.

Dyp-kopi Kopi av et objekt hvor all data til det originale objektet har blitt kopiert og lagret et annet sted i minnet.

Grunn-kopi Kopi av et objekt som peker til samme minnet som det originale. Begge objektene kan endre på det samme minnet, som ofte ikke er ønskelig.

Versjonskontroll-

program Programvare for å lagre en kodebase slik at en kan gå tilbake til tidligere punkter i utviklingen og som brukes for å lagre koden i skyen.

Kjøretid Hvor mange operasjoner, i verste tilfellet, som trengs for å utføre en oppgave.

# 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.)]

Dette prosjektet innebar å lage et “Warehouse management system” for det fiktive selskapet “Smarthus AS”. Dette er et program som oppbevarer og endrer på informasjon om et varehus og produktene det har på lager. Programmet skal ha et brukergrensesnitt som ansatte kan bruke for å utføre endringer på systemet. Programmet består altså av tre forskjellige deler: et element som beskriver en vare og håndterer informasjonen til varen, et element som beskriver et inventar som har en samling av varer og funksjoner for å behandle varene og brukergrensesnittet, som tillater en ansatt å bruke funksjonaliteten til inventaret.

I oppgaven ble det oppgitt at Smarthus AS ønsker at et produkt skal holde på 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) Gulvlaminater, (2) Vinduer (3) Dører og (4) Trelast

Det ble også oppgitt av brukergrensesnittet skulle implementere følgende funksjonalitet:

1. Skrive ut all 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 (gitt over) 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 (eksisterende vare). 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 å sette antall varer til 0.
7. Endre rabatt, pris og/eller varebeskrivelse for en vare

Vi stod valgfritt til å bestemme hvordan brukergrensesnittet skulle ta innputt fra brukeren. Det ble også oppgitt at vi stod fri til å legge til og endre på funksjonalitet hvis vi synes det var hensiktsmessig eller ville forbedre brukeropplevelsen.

Sentralt i problemstillingen er at vi skal følge prinsippene for utvikling og design av kode vi har lært om i løpet av semesteret.

## Avgrensninger

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

I beskrivelsen av hvilke funksjonalitet brukergrensesnittet skal ha, så står det på punkt 7: “Endre rabatt, pris og/eller varebeskrivelse for en vare.” I kravene for vare-klassen, så er rabatt ikke tilstede som noe den skal inneholde. Å endre rabatt er derfor ikke blitt inkludert som noe en kan gjøre i brukergrensesnittet. Et annet argument for hvorfor en rabattfunksjon ikke burde implementeres, er at å gi rabatt er noe en gjør i henhold til salg. Siden dette er et program for håndtering av et varehus, så er håndtering av rabatt noe som ikke hører hjemme i programmet.

## 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 | Inneholder informasjon om en gitt vare. |
| Inventar | Inventory | Inventaret er samlingen av alle produktene som ligger på lageret, og det har ansvaret for å endre informasjonen til enkelte produkter og lagerbeholdningen som helhet. |
| Klient | Client | Klienten er ansvarlig for brukergrensesnittet til inventaret. Klienten tar innputt fra brukeren for å utføre operasjonene som inventaret tilbyr |

## 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

# 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.]

Det har blitt benyttet flere verktøy i sammenheng med å utvikle programmet: VS Code ble brukt som IDE, og Git ble brukt som versjonskontrollprogram. GitHub ble brukt for skylagring av koden. Bruken av Git og GitHub gjør at en kan jobbe på koden fra flere PCer, og det sikrer at koden er trygt lagret i tilfelle PCene skulle bli ødelagt eller informasjonen på de blir slettet av en eller annen grunn. Et av kravene til koden er at en skal følge en kodestil. For dette prosjektet har Google sin blitt brukt. VS Code har en plugin for CheckStyle som gjør det enkelt å sjekke at koden følger stilen.

Oppgaven ble gitt til oss i tre deler. Fremgangsmåten for å løse oppgaven har derfor vært å gjøre hver av de tre delene individuelt i stedet for å gjøre alt i ett. Dette forsikrer at oppgaven blir gjennomført slik faglærer tiltenkte det og minsker sannsynligheten for at en gjør feil, siden en løser oppgaven i mindre deler.

Som beskrevet i 3.1, så består programmet av tre forskjellige deler: en enkeltvare, samlingen av varene og brukergrensesnittet. Dette gir et bra utgangspunkt for hvordan en kan strukturere klassene i Javakoden. En kan lage en klasse som representerer et produkt, en som representerer inventaret og en som inneholder brukergrensesnittet. I denne løsningen er disse klassene henholdsvis: “Product”, “Inventory” og “Client.”

Startet med å implementere product

*robuste design prinsipper som modularisering, coupling, cohesion,*

*responsibility driven design osv.*

For å følge prinsippet om lagdelt arkitektur, ble det bestemt at klienten kun skal ta informasjon fra brukeren og gi det videre til inventory klassen som bruker det til å manipulere inventaret produktene. Dette sikrer også lavere coupling, cohesion og følger responsibility driven design og lagdelt arkitektur.

Videre utvikling tok utgangspunkt i å følge disse prinsippene.

Tok notater og skrev i rapporten underveis.

# 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).]

“Product” er en informasjonsklasse. Den har felter for all informasjonen en vare skulle holde på i følge kravspesifikasjonen. Klassen tilbyr kun metoder for å hente og endre på verdien til feltene. og for å endre på verdiene til alle feltene bortsett fra id’en. Datatypene til feltene ble valgt basert på det som stod i oppgaveteksten, siden beskrivelsen av slik dataen skulle lagres kunne bli direkte oversatt til datatyper i Java.

Jeg har valgt å implementere mutatormetoder for alle feltene bortsett fra id-feltet. Dette er i tilfelle en har lagt inn feil informasjon på noen av feltene. Det vil være en vesentlig bedre brukeropplevelse å kunne endre på ett eller flere felt med feil informasjon enn å måtte lage hele objektet på nytt. Hvis en skulle ha laget et nytt objekt av samme produktet, så må en også slette det eksisterende objektet før en kan legge inn det nye. Å kunne endre feltene på et eksisterende produkt er altså konseptuelt enklere å skulle implementere og det vil føre til en bedre brukeropplevelse.

I koden brukes det hjelpemetoder for å forsikre at verdiene på feltene (som det gir mening eller er mulig å gjøre det for) er korrekte. Disse feltene er price, weight, height, length, category og quantity. Feltene price, weight, height og length blir sjekket om er strengt større enn null. Det gir ingen mening at weight, height eller length skal være null, ettersom dette er fysisk umulig. En kan argumentere for at price skal kunne settes til null, f.eks. for å indikere noe form for salg, men dette gjør funksjonen til feltet utydelig, noe som bør unngås. “category” blir sjekket om er enten 1, 2, 3 eller 4, og quantity blir sjekket om er større eller lik 0. I stedet for å måtte repetere koden for å sjekke om disse verdiene er korrekte, så bruker konstruktøren setter-metodene. En annen måte vi forsikrer at brukeren ikke gir feil verdier, er ved at vi har skrevet hvilke verdier som er akseptert i javadocen til koden. Brukeren vet altså hva slags verdier som er lovlig før de tar i bruk koden, og koden sier i fra til brukeren i tilfelle de skriver feil med et uhell.

## Del 2

I denne delen skulle første iterasjon av brukergrensesnittet og registerklassen utarbeides. Registermetoden skal bruke en av samlingene som er tilgjengelige i JDK for å holde på de ulike produktene som blir registrert i varehuset. I oppgaveteksten ble ArrayList, HashMap og HashSet valgt. Av disse tre ble ArrayList og HashMap vurdert, og det endelige valget ble ArrayList.

En av hovedfordelene med HashMap, er at den har konstant tid for get og put. <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html> Det vil si at det tar kun en operasjon (forbedre beskrivelsen av dette) for å finne elementet du vil søke etter i HashMapet. I ArrayList er søketiden lineær. Det vil si at du må på det meste bruke like mange operasjoner som det finnes elementer i listen for finne elementet du søker etter.

Måten HashMap oppnår konstant søketid på, er ved at hvert element blir tildelt en nøkkel. I dette programmet kunne det f.eks. vært id’en til varen. Når en vil hente et element fra HashMapet, så kaller en på en metode til HashMap objektet, og en gir nøkkelen til elementet som argument til metoden. HashMapet gjør så en behandling på nøkkelen som gir den minneadressen til elementet og det kan hente det ut direkte. Siden brukeren skal i dette programmet kunne søke etter en vare ved å gi id’en til varen, så gir dette også en innebygd måte å implementere dette søket.

I en ArrayList er elementene indeksert etter posisjon, og indeksen starter på 0. For å hente ut et element må en altså vite posisjonen til et element i listen. Siden id’en til en vare i dette programmet skal inneholde både bokstaver og tall, er det ikke mulig å hente et element fra en ArrayList direkte med id’en. For å finne et element i listen etter id’en, må en da gå gjennom hvert element i listen og sjekke om id’en til varen på en gitt posisjon stemmer overens med den som er gitt av brukeren. I verste fall, så ligger varen en ser etter sist i listen, derav den lineære søketiden til en ArrayList.

Det er dog flere grunner til at en ArrayList er fornuftig å bruke ovenfor et HashMap i dette programmet. Den første er at antallet elementer som skal representeres i listen, sannsynligvis, er så liten at søketiden gjennom listen ikke kommer til å være merkbart for et menneske. Moderne datamaskiner kan søke gjennom lister på flere millioner elementer på millisekunder (INSERT CITATION). En rimelig antagelse er at et varehus på det aller meste kommer til å ha ulike varetyper målt i titusener (INSERT CITATION). The Home Depot, verdens største Det en sparer i beregningstid veier ikke opp for den mer kompliserte implementasjonen bruken av et HashMap vil føre til.

Alle endringer på inventaret, altså ArrayListen som inneholder produktene som ligger på lageret, skjer i inventar klassen. For at andre klasser skal kunne endre på et produkt i inventaret må det vite id’en til produktet. Klassen inneholder metoder som lar brukere av klassen finne id’er til produktet basert på søk etter id og beskrivelse. Siden en må gi den id’en, så kan inventaret bruke den til å sjekke om det er et eksisterende produkt, og utløse et unntak hvis det ikke eksisterer. Dette garanterer at brukere av inventar klassen ikke kan utføre ugyldige handlinger på lageret. Siden inventar klassen kun utfører handlinger som er relatert til å endre eller hente informasjon om inventaret, så har klassen høy cohesion og deltar i modulariseringen av programmet for øvrig. Den oppnår også lav kobling ved at den ikke er avhengig av andre klasser for å utføre handlingene på inventaret.

Skrive om feilhåndtering av innputt når en skal endre på eller legge til et produkt.

For brukergrensesnittet valgte jeg å bruke terminalen til å hente innputt fra brukeren og for å vise informasjon. Siden vi ikke har lært å lage grafiske grensesnitt, så anså jeg terminalen som den beste måten å lage grensesnittet på.

Feilhåndtering av innputt skjer i klienten og i produkt klassen. Klienten sjekker at det brukeren skriver inn passer til typen som den verdien skal bli brukt til (e.g. hvis brukeren har skrevet en streng der det skal være en int.) Å sjekke at verdiene til produktet ikke blir satt til noe ugyldig skjer i produkt klassen. Dette fører til lavere kobling og enklere gjenbruk av koden, siden noen som bruker klassen ikke trenger å innføre sjekker for ugyldige verdier selv, men trenger bare å håndtere unntaket som blir utløst. Hvis hva som er gyldige verdier endrer seg, så trenger en bare å oppdatere produkt klassen. (Ikke helt sant. Hvis det virkelig skulle vært fullstendig avkopling, så måtte informasjonen om hva som er gyldige verdier som skal printes ut hentes fra produkt klassen, likt som den formaterte strengen.)

# 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.]

# 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.]