| Kandidatnummer(e)/Navn: | | | |
| --- | --- | --- | --- |
| Sigurd Riseth | | | |
| Dato: | Fagkode: | Studium: | Ant sider/bilag: |
|  | IDATG1003 | BIDATA | 2 / |

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

|  |
| --- |
| Tittel : |
| Rapport for mappevurdering «Train Dispatch System» i IDATG1003 |

|  |
| --- |
| Sammendrag: |
| Skriv et kort sammendrag av hva du/dere har utviklet og hva denne rapporten handler om  Train Dispatch System er et program som lagrer og håndterer togavganger på en enkelt stasjon. Programmet holder orden på tognummer, avgangstid, forsinkelser, spor, linjenavn og destinasjon.  Rapporten beskriver utviklingsprosessen, sluttresultatet og hvorfor valgte løsninger ble brukt. |

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

INNHOLD

[1 SAMMENDRAG 1](#_Toc149558631)

[2 TERMINOLOGI 1](#_Toc149558632)

[3 INNLEDNING – PROBLEMSTILLING 1](#_Toc149558633)

[3.1 Bakgrunn/Formål og problemstilling 1](#_Toc149558634)

[3.2 Avgrensninger 2](#_Toc149558635)

[3.3 Begreper/Ordliste 2](#_Toc149558636)

[3.4 Rapportens oppbygning 3](#_Toc149558637)

[4 BAKGRUNN - TEORETISK GRUNNLAG 3](#_Toc149558638)

[4.1 Abstrahering 4](#_Toc149558639)

[4.2 Modularisering 4](#_Toc149558640)

[4.3 Coupling 4](#_Toc149558641)

[4.4 Cohesion 4](#_Toc149558642)

[4.5 Refaktorering 5](#_Toc149558643)

[4.6 Defensiv programmering 5](#_Toc149558644)

[4.7 Lambda uttrykk 5](#_Toc149558645)

[5 METODE – DESIGN 5](#_Toc149558646)

[6 RESULTATER 6](#_Toc149558647)

[7 DRØFTING 7](#_Toc149558648)

[8 KONKLUSJON - ERFARING 8](#_Toc149558649)

[9 REFERANSER 8](#_Toc149558650)

[10 VEDLEGG 8](#_Toc149558651)

[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…..]

Denne rapporten er besvarelsen til obligatorisk innlevering i faget IDATG1003 Programmering 1. Hensikten med denne rapporten er å presentere oppgaven, fremgangsmåten og resultatene fra løsningen av oppgaven på en ryddig og oversiktlig måte.

Rapporten skal klart beskrive problemstillingen og hvordan den ble løst. (Kort beskrivelse av tekniske utfordringer???). For å tydelig presentere kodens design, vil klassediagrammer bli brukt for å visualisere strukturen og forholdene mellom kodedelene. Til slutt presenteres resultatene og en drøfting og konklusjon av oppgaven/løsningen. Se innholdsfortegnelse for mer informasjon.

Rapporten ble utarbeidet kontinuerlig ved siden av løsning av oppgaven. Git (GitHub) ble brukt til å lagre progresjon av mappen og rapport.

SKRIV OM RESULATER OG KONKLUSJON HER!!!

# 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.]

UI User Interface (Brukergrensesnitt)

AI Artificial Intelligence (Kunstig Intelligens)

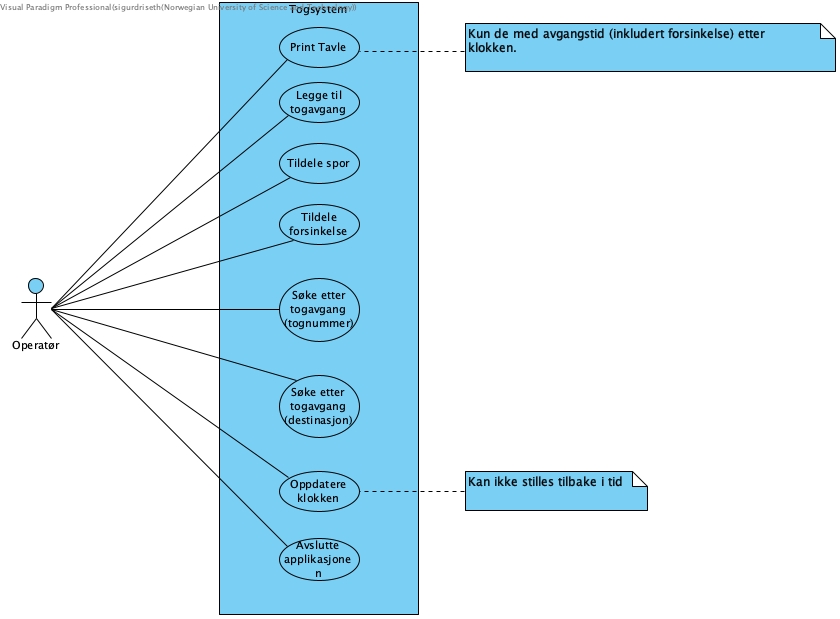
# INNLEDNING – PROBLEMSTILLING

## 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.)]

Prosjektoppgaven ble gitt i tre deler utover semesteret. Problemstillingen var å utvikle et forenklet system for avvikling av togavganger. Hoved funksjonaliteten til systemet var å presentere en informasjonstavle med kommende togavganger. Men systemet hadde krav til andre funksjoner og.

Figur 1 Use-Case Diagram

Oppgaven stilte krav til et tekstbasert brukergrensesnitt i form av en meny. Fra denne menyen skal bruker kunne gjøre følgende oppgaver:

* Vise/skrive ut oversikt over togavganger, sortert etter avreisetidspunkt (informasjonstavle).
* Legge inn en ny togavgang – det skal ikke være mulig å legge inn et tog med tognummer tilsvarende eksisterende tog i listen.
* Tildele spor til en togavgang – ved først å søke opp togavgang basert på tognummer, og så sette spor.
* Legg inn forsinkelse på en togavgang – ved å først søke etter en gitt togavgang basert på tognummer, og deretter legge til forsinkelse.
* Søke etter en togavgang basert på Tognummer.
* Søke etter togavgang basert på destinasjon.
* Oppdatere klokken (tidspunktet på dagen) – ved å spørre bruker etter nytt klokkeslett.
* Avslutte applikasjonen

Togavganger med avreisetidspunkt (inkludert forsinkelse) før klokken skal automatisk fjernes fra tavlen. Klokken skal ikke kunne settes til et tidligere klokkeslett enn det gjeldende.

Informasjonstavlen måtte presenteres i følgende rekkefølge:

1. Avgangstid på formatet «hh:mm» (for eksempel 09:30)
2. Linje
3. Tognummer
4. Destinasjon
5. Forsinkelse
6. Spor

Forsinkelse og spor skal kun vises i informasjonstavlen om de har en tildelt verdi.

Benytte UML Diagrammer for å beskrive krav/oppgave?

## Avgrensninger

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

Systemet har følgende avgrensninger tildelt i oppgaveteksten:

* Systemet skal støtte kun en stasjon (altså tog som kjører fra en bestemt stasjon).
* Systemet tar ikke hensyn til dato, kun tidspunktet innenfor en enkelt dag.
* «Klokken» oppdateres manuelt fra brukermenyen (ingen bruk av systemklokke).

## 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.]

Nedenfor er det en liste med begreper som blir brukt i applikasjonen og hva de betyr.

| Begrep (Norsk) | Begrep (Engelsk) | Betydning/beskrivelse |
| --- | --- | --- |
| Togavgang | Train departure | Et tog som skal gå fra stasjonen, identifisert med et unikt tognummer. |
| Spor | Track | Det tildelte sporet for et bestemt tog |
| Tognummer | Train number | Et unikt identifikasjonsnummer for hvert tog. |
| Linje | Line | Navn på linjen en gitt togavgang kjører på. Dette er relatert til destinasjonen. |
| Destinasjon | Destination | Endestasjonen for en gitt togavgang. |
| Avgangstid | Departure time | Togets planlagte avgangstid. |
| Forsinkelse | Delay | Eventuell forsinkelse på toget, registrert i formatet «hh:mm». |
| Stasjon | Station | Stasjonen hvor togene avreiser fra. |
| Klokke/Tiden | Clock/Time | Tidspunktet på dagen ved stasjonen. |

De syv første begrepene i denne tabellen tilhører togavgangene, mens de to siste tilhører stasjonen. Det vil si at enhver togavgang kan ha et unikt spor, tognummer, linje, destinasjon, avgangstid og forsinkelse. Men de opererer på samme stasjon og i samme klokkeslett.

## 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.

Mange faktorer gir en indikasjon på hva som utgjør god kode. I løpet av dette semesteret har vi blitt presentert for beste praksis for å oppnå akseptable løsninger.

I kjernen av objektorientert programmering finnes to sentrale konsepter: objekter og klasser [1]. En klasse gir en abstrakt definisjon av alle objekter av samme type [1]. Objekter opprettes basert på disse klassene og inkluderer feltene og metodene definert i klassen [1]. Det er denne bruk av objekter fra klasser som skiller objektorientert programmering fra funksjonell programmering. Objektorientert programmering tilbyr en tilnærming som generelt letter utviklingen av prosjekter (med noen unntak, spesielt innenfor matematikk). I dette avsnittet vil jeg presentere relevant teori som har blitt dekket i løpet av kurset.

## Innkapsling

Java innkapsling er en måte å skjule implementeringsdetaljene til en klasse fra ekstern tilgang og bare eksponere et offentlig grensesnitt som kan brukes for å samhandle med klassen. Innkapsling blir oppnådd ved å deklarere felter i klassen som private, og håndtere tilgang utenifra igjennom getter og setter metoder. [2]

## Abstrahering

Abstraksjon er evnen til å ignorere detaljer om deler, for å rette oppmerksomheten mot et høyere nivå av et problem [1]. Utfordringene deles da opp i mindre og mindre biter til de blir lettere å løse.

## Modularisering

Modularisering er prosessen med å dele et helt system inn i mindre og godt definerte deler som kan bygges og utforskes separat, samtidig som de samhandler på forhåndsdefinerte måter. [1]

## Kobling

Kobling refererer til forholdet mellom ulike klasser eller komponenter i et program. Dette konseptet dreier seg om hvor tett eller løst disse komponentene er knyttet til hverandre. Målet er å oppnå løse koblinger mellom klassene, noe som betyr at de primært er uavhengige av hverandre og kun kommuniserer via et begrenset og godt definert grensesnitt.

## Kohesjon

Kohesjon beskriver hvor godt en kode er knyttet til en spesifikk oppgave. I et program med høy kohesjon vil hver kodeenhet (metode, klasse eller modul) være ansvarlig for en veldefinert oppgave eller entitet. Kohesjon gir oss huskeregelen «en klasse, ett ansvar» og «en metode, en oppgave». Godt klassedesign inneholder en høy grad av kohesjon.

## Refaktorering

Refaktorering er prosessen med å reorganisere og endre eksisterende kode for å forbedre dens struktur, lesbarhet, ytelse og vedlikeholdsevne. Dette gjøres ofte ved å dele klasser eller metoder i flere deler for å øke dens kohesjon.

## Defensiv programmering

Defensiv programmering er å produsere kode som beskytter seg selv mot mulige feil og unntak. Hovedmålet med defensiv programmering er å gjøre systemet mindre åpent for feil. Noen måter å skrive defensiv kode er med error handling (håndtere feil på en måte som sikrer mot sammenbrudd og data korrupsjon) og input validation (sjekke at gitt parameter stemmer med det som forventes).

## Lambda uttrykk

Lambda uttrykk/funksjoner er korte funksjoner som vanligvis brukes for enkle operasjoner og kan defineres i en enkelt linje. Lambda-funksjoner er ikke ment for kompleks logikk eller operasjoner som må gjenbrukes på flere steder i koden din. De brukes vanligvis for kortvarige, spesifikke oppgaver.

# 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.]

Første løsning ble å opprette en klasse for togavganger som skulle holde på feltene spor, tognummer, linje, destinasjon, avgangstid og forsinkelse. Denne klassen skulle ha enkle (getter og setter) metoder for å endre feltene og få de returnert. Klassen fikk navnet TrainDeparture og har som ansvar å lagre hver enkelt togavgang.

Det ble også opprettet en ny klasse - Station - som skulle lagre instansene av TrainDeparture i et HashMap og ha kontroll på klokkeslettet på stasjonen. Denne klassen brukes også til å opprette nye instanser av TrainDeparture og endre eksisterende. All kommunikasjon mellom UI-klassene og resten av systemet skjer igjennom Station.

UI-klassen (UserInterface) lages og kjøres ved start av programmet. Den har to funksjoner; Init() kjøres først og initialiserer programmet ved å opprette et instans av Station og andre nødvendige prosedyrer. Etter dette kjøres start() som kommuniserer med bruker og håndterer inputs.

Hovedgrunnen til at disse klasse ble valgt er fordi det gir en høy grad av kohesjon, ettersom klassene kun har ett ansvar. UserInterface, tolke brukerinput og sende dette videre til Station. Station skal behandle instansene av TrainDeparture og klokken. Mens TrainDeparture brukes for å lagre togavgangene.

Senere ble det lagt til to andre klasser, Printer og UserInputHandler. Dette ble gjort for å øke graden av kohesjon i programmet. UserInputHandler har metoder som returnerer brukerinput.

Printer klassen har som ansvar å printe ut all informasjon til brukeren. Ettersom all informasjon brukeren mottar nå er i én enkelt klasse er det enklere å legge til informasjon eller oversette programmet. Det blir også enklere å refaktorere programmet hvis du senere ønsker å kommunisere med bruker i et annet system enn terminalen.

Systemet har blitt utviklet i IntelliJ og skrevet på språket Java. Følgende tilleggsprogram (plugins) har vært brukt:

* SonarLint – varsler om dårlig kodepraksis og dårlige løsninger.
* CheckStyle – brukt til å kontrollere at kode er skrevet i henhold til google standard.
* GitHub CoPilot – AI-verktøy som kommer med forslag og automatiske kodeutsnitt. Brukt for å øke produktivitet.

Mappen ble opprettet i GitHub og kontinuerlig lagret og opplastet dit under utvikling ved bruk av Git. Tilbakemelding på kode ble gitt av læringsassistent igjennom GitHub.

LA TIL PRINTER KLASSE FORDI DET GJØR ENDRINGER I FREMTAIDEN LETTERE. TRENGER BARE ENDRE PRINTER KLASSEN FOR Å FÅ DET PÅ ANNET SPRÅK ELLER HVIS DU NESTE GANG SKAL PRINTE UT PÅ EN XBOX OG IKKE TREMINALEN.

NÅR DU OPPRETTER EN NY TRAINDEPARTURE LAG DEN I ADDTRAINDEPARTURE TO HASHMAP.  
  
GRACEFULL TERMINATION OG IDIOTSIKKERT PROGRAM MED TRY/CATCH.

Telefonregister.addContact(new Contact(firstName, lastName, phoneNumber)); anonymt object.

# 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).]

Klasser og deres ansvar:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TrainDispatchApp | TrainDeparture | Station | UserInterface | InputHandler | Printer |
| Opprette UI-klassen og starte programmet. | Entitetsklassen. Håndtere dataen til én enkelt togavgang. | Behandle liste av togavganger og stasjonsklokken. | Initialisere programmet og kjøre brukermenyen. | Etterspørre og tolke brukers input. | Printe til terminalen/ brukeren. |

UserInterface klassen benytter en switch case for å håndtere

# 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. Harsh Agarwal. (2023). *Encapsulation in Java.* Geeksforgeeks. https://www.geeksforgeeks.org/encapsulation-in-java/

# Bibliografi

Eirik Rossen. (2020). Objekt (IT). SNL. <https://snl.no/objekt_-_IT>

# 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.]