| Kandidatnummer(e)/Navn: 10042 | | | |
| --- | --- | --- | --- |
|  | | | |
| Dato: | Fagkode: | Studium: | Ant sider/bilag: |
|  | IDATA1001 | BIDATA | 3 / |

|  |
| --- |
| Faglærer(e) : |
| Arne Gerhard Styve |

|  |
| --- |
| Tittel : |
| Warehouse Management System (WMS) |

|  |
| --- |
| Sammendrag: |
| Jeg har laget et system som skal hjelpe og holde orden på ett varehus. Systemet har en identitetsklasse som symboliserer en vare i lagerhuset. Denne varen har en rekke felt som beskriver varen. Varene er lagret i ett register som brukes for å oppnå ulike mål med varene.  Systemet er to delt, da jeg har delt det opp i logikken bak applikasjonen, og ett brukergrensesnitt som snakker med den bakenforliggende. Disse lagene er delt inn i pakker, for å organisere og å sikre pakkene.  Videre i rapporten skal det forklares nærmere på hva slags fremgangsmåte er brukt, og mer i dybden på hvorfor det er valgt å gjøre det slikt.  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](#_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 Ingen norsk oversettelse

WMS Warehouse Management System Ingen norsk oversettelse

Int Integer Heltall

E Exception Unntak

UI UserInterface Brukergrensesnitt

# INNLEDNING – PROBLEMSTILLING

## Bakgrunn/Formål og problemstilling

Oppgaven som er gitt vil ha ett system som kan implementeres i ett varehus. Det som er viktig for dette systemet er at det har nødvendige funksjoner som gjør brukerens opplevelse fullstendig. Funksjoner som gjør brukeropplevelsen fullstendig, er metoder som endrer på innholdet til varehuset. Systemet skal skrives i kodespråket Java, og det skal brukes tekstbasert brukergrensesnitt for å informere brukeren.

Problemet er gitt i 3 forskjellig deler, da det er gitt utover ett halvt år. Iterasjon 1 handlet om klassen Item, altså en vare i dette systemet. Denne klassen er selve grunnlaget for resten av applikasjonen, og hva som er nødvendig å legge til av utvidelser av applikasjonen. Iterasjon 2 handlet om registeret, selve klassen som representerer varehuset, og brukergrensesnittet. Her ble det gitt en del krav om hva som var nødvendig for en bruker å ha muligheten til. Iterasjon 3 handlet om at man skulle sette sitt eget preg på koden, og legge til ekstra funksjoner for å gjøre brukeropplevelsen enda bedre.

Kravene i iterasjon 1 var som følger;

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) Gulv

laminater, (2) Vinduer (3) Dører og (4) Trelast. (*Vedlegg 1)*

Kravene i iterasjon 2 var som følger;

Brukergrensesnitt

**Brukergrensesnittet** skal tilby følgende funksjonalitet til brukeren. Brukeren er en person som arbeider på lageret.:

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).

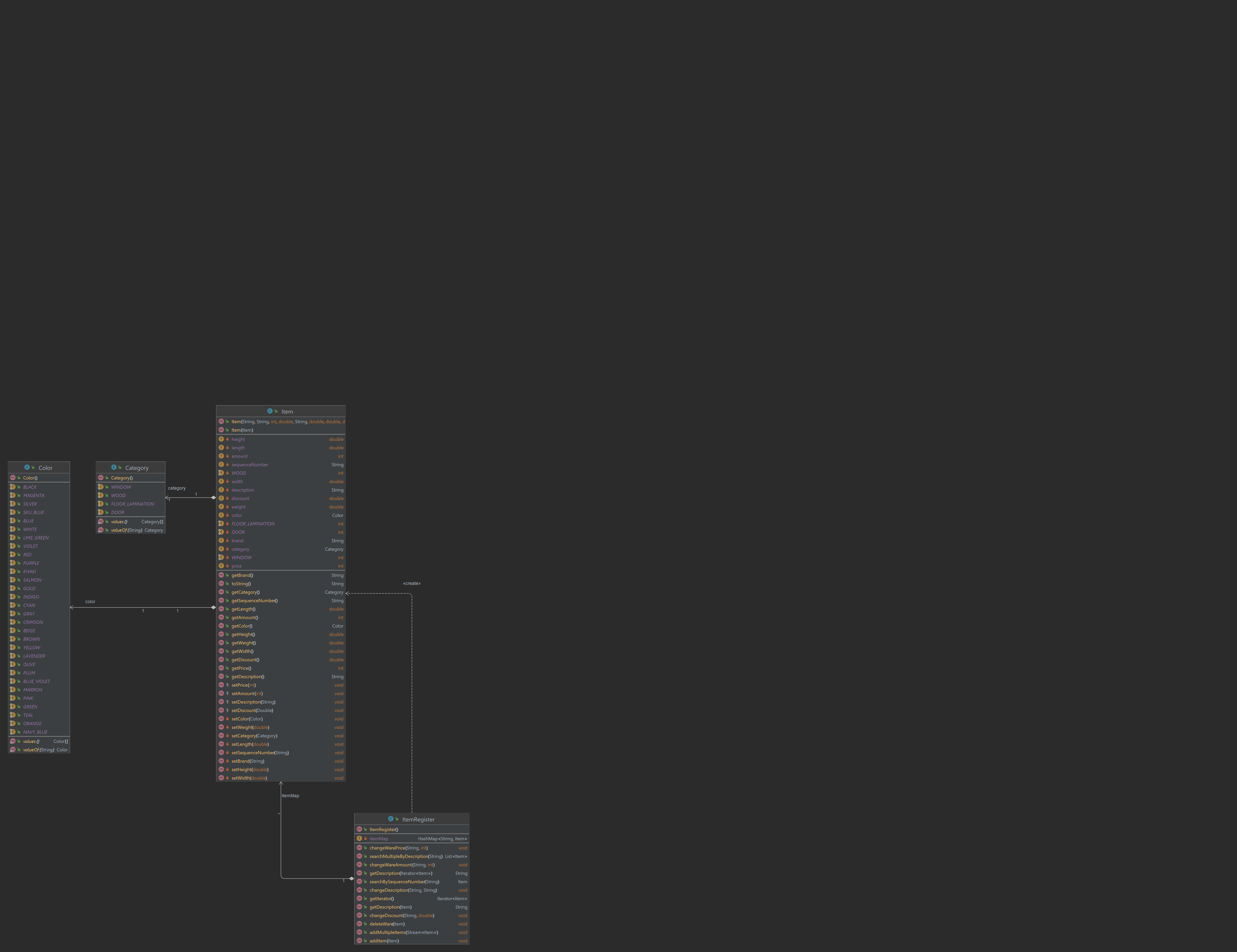
5. Ta ut varer fra varebeholdningen (eksisterende vare). M.a.o. du har en vare med et gitt antall på lageret (f.eks. 20 stk laminatgulv). Du tar så ut 5 stk fra lageret.

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. *(Vedlegg 2)*

I denne iterasjonen var det også søkelys på noen utforming prinsipper. Robust design for eksempel *cohesian, coupling* og *resposibility driven design.*

Under kan dere se ett UML diagram som viser klassene i pakken Logic. Det er disse klassene som utgjør selve logikken bak systemet.



[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

Det er satt litt avgrensninger på de ulike feltene til identitetsklassene. For eksempel, varenummer skal være tall og bokstaver, pris skal være heltall, vekt i kg og skal være desimaltall osv.

Likevel er det ikke sagt noe i kravspesifikasjonen om at feltene til en klasse skal være private, eller om det er noen metoder som skal / burde være det.

I oppgaven står det oppgitt at koden skal være i henhold til google sin kode stil. Dette handler om luft i kode, og hvordan javadokumentasjonen skal skrives. Og at dette skal sjekkes med en checkstyle plugin til ide’en Intellij.

[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 |
| --- | --- | --- |
| Vare | Item | Varehuset skal håndtere produkter inn og ut av et lager |
| Klasse | Class | Noe som representerer ett objekt eller en gjenstand i den virkelige verden, eller har en funksjon |
| Felt | Field | En klasse variabel |
| Register | Register | En liste med informasjon |
| Klasse variabel | Field variable | En klasse variabel |
| kobling | coupling | Ett designprinsipp, handler om kodens sammensveising |
| Entydige/ samhold | Cohesian | Ett designprinsipp, handler om kodens samhold |
| Kildekode | Source code | Den faktiske koden i .java format |
| Iterasjon | Iteration | En betegnelse for å utføre |
| Utvidelse | Plugin | En utvidelse til ett program |
| Modul | Module | En pakke som inneholder klasser som henger sammen |
| Metode | Method | En java «funksjon» |
| Oppbevaringssted | Repository/ Repo | Et sted for å oppbevare noe, her; kode |
| Refaktorisere | Refactor | Gjøre om igjen, eller endre på tidligere kode |
| Begå | Commit | Her; brukes om endringer som skal tas i betraktning av eksternt repo |

## Rapportens oppbygning

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

# BAKGRUNN - TEORETISK GRUNNLAG

WMS er bygget opp med en rekke teoretiske prinsipper for å sikre ett godt sluttresultat.

En av disse prinsippene er sentralisert rundt klasse variablene. Alle klasser variabler skal være private. Dette gjøres primært av to grunner, for å oppnå *low coupling,* eller lav kobling, og sikkerhet. Disse to problemene er egentlig 2 sider av samme mynt. Grunnen til at vi vil ha høy sikkerhet i identitetsklassene samtidig som vi vil ha *low coupling* er for å minske muligheten for feilmeldinger eller ulovlige operasjoner. Med dette menes at, de øvrige klassene ikke har mulighet til å endre på klassevariablene direkte, men er nødt til å gå gjennom klassens definerte metoder for å endre på objektet. Dette er viktig for å sikre *low coupling* i koden. Med *low coupling* menes at koden ikke er sterkt avhengig av hverandre. Det er ett abstrakt konsept om hvordan koden henger sammen. Man kan beskrive det som graden av det som må endres for å få implementert nye eller refaktorisere gamle funksjoner. Selv om en klasse som bruker item har den ikke behov for å kunne vite om alle de private feltene eller metoden, adgangen til item blir redusert slik at klassen kun kan se det som er definert som synlige(*public*). Ved å opprettholde slikt prinsipp vil man ved ett senere tidspunkt om man skulle ønske å refaktorisere koden, vil man enkelt kunne endre på den opprinnelige klassen uten å at det oppstår mange problemer på klassene som bygges oppå denne. (Barnes & Kölling, 2017)

Ett annet design prinsipp som er viktig for å få en god kildekode er høy cohesian, med dette menes høy entydig kode. Dette prinsippet går up på at hver klasse og eller hver metode kun har ett ansvarsområdet. Det vil si at hver metode gjør kun en oppgave og at hver klasse har kun en oppgave. Fordelen med å følge dette prinsippet er at det blir lettere å abstrahere koden, med det menes at det er lettere å gjøre spesifikke problemer til generelle problemer. Ved å generalisere problemene kan man enklere løse problemene og man kan bruke denne løsningen flere ganger. Entydig kode gjør også at det er lettere for leseren og eller folk som skal bruke koden videre, til å få en oversikt. Høy cohesian bidrar til god struktur og god organisering i metodene samt i klassene. Ett annet problem som minker om man, sørger for høy cohesian i koden er kodeduplikat. Ved å løse generalisere problemene og løse det en gang, kan man bruke det flere ganger for å løse lignende problemer. (Barnes & Kölling, 2017)

Design prinsipp angående pakker er også ett godt virkemiddel om man vil ha god kilde kode. Ved å dele kilde koden inn i pakker, kan man strukturer en god lagdeling. Dette fører til at man kan ta i bruk *protected* som er en synlighet parameter for felt og metoder. Denne kan brukes for å skjule metoder for eksterne biblioteker, brukere eller andre pakker. God mappestruktur er også korrelert med leselighet, og struktur i prosjektet. Pakkestruktur faller under også under modulisering prinsippet. (Barnes & Kölling, 2017)

For at brukeren ikke skal få en feilkode eller at programmet terminerer uforventet er det viktig å tenke på en *failsafe* kode. I dette legges at man skal skrive kode som håndterer feil mens den kjører, og gir bruker en sjanse til å rette opp feilen, uten at koden terminerer. Dette gjøres på litt forskjellige måter, en av måtene som ofte brukes er med *exceptions.* Ved å kaste ett unntak i de lavere nivåene ungår man at feil oppstår. Disse unntakene må tas imot senere i programmet for å unngå brå slutt på programmet. Java har allerede en del innebygde unntak, som er lett å ta i bruk, men om man trenger flere unntak er dette mulig å opprette. Ved å kaste disse unntakene åpner man for at feil i koden kan løses på ett høyere nivå. For eksempel om man skal opprette et objekt med definerte felt gitt av brukeren, men brukeren har skrevet inn argumenter som ikke er godkjente, kan man ta imot ett unntak og gi brukeren muligheten til å skrive inn et nytt argument som er godkjent. En måte på å sjekke om objektene og metodene opererer riktig, kan man ta i bruk testklasser. Med testklasser tester man de ulike oppgavene til metodene og klassene, for å forsikre seg om at objektene og metodene håndterer unntak på riktig måte. Tester kan gjøres automatiske og kjøres hver gang man pakker programmet. Det som er viktig med tanke på testing er at man både tester for argumenter som er gyldige og ugyldige. På denne måten kan man få en idé om hvor godt logikklaget jobber og om det er noe som trengs å forbedres eller gjøres annerledes. (Barnes & Kölling, 2017)

[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

Koden som er blitt skrevet, er utviklet i Intellij. Dette er en IDE som er lagd for Java og den er utviklet av JetBrains. Ved hjelp av Intellij ble et maven prosjekt opprettet. Det ble planlagt mapper hvor filene skulle ligge med en gruppe id relatert til faget. Prosjektet er lagret i pakker i henhold til maven sine forhåndsdefinerte protokoller. Det er også brukt en Intellij plugin som het *check-style* som er brukt for å kontrollere at koden er skrevet i henhold til angitt kodestil. Denne plugin’en sjekker blant annet luften i koden er der hvor den skal være, at det ikke er mer luft en det nødvendige, alt av dokumentasjon av koden er skrevet i henhold til stilen osv. Javadokumentasjon er skrevet etter hvert som koden utviklet seg, og det oppstår nye metoder og eller klasser.

Selve koden er utviklet utefra agil metode, i dette ligger det at prosjektet er utviklet under flere iterasjoner. Den agile utviklingsmodellen er en modell som er basert rundt iterasjoner eller sprinter, som er ulike gjøremål eller ferdigfremstillinger av delprosjekter som skal være fastsatte til en frist. Som overnevnt har dette prosjektet foregått i 3 iterasjoner, da den første iterasjonen var kun en start fase for prosjektet og de 2 neste var mer utfyllende iterasjoner. Grunnen til dette er fordi at vi som studenter får ikke tilgang til hele prosjektet med en gang. Med tanke på det passer den agile fremgangsmetoden bedre enn fossefallsmetoden, fordi den agile metoden er mer agil og fleksibelt, og kan fort endre visjonen om iterasjonene om det trengs. Iterasjon 1 handlet mye om fundamentet i prosjektet, sette opp ett repo, sette opp filstruktur, koble prosjektet opp med skyen, lage første identitetsklasse. Iterasjon 2 gikk ut på å ferdigstille logikken bak applikasjonen og ferdigstille brukergrensesnittet, samt lage ett første utkast av den endelige rapporten. Den tredje iterasjonen handlet om refaktorisering av koden, sette sitt eget preg på koden og ferdigstille rapporten.

Under prosjektet er det gjort en rekke endringer på koden. Det er gjort mange refaktoriseringer, derfor er det viktig med god plan for å opprettholde versjons kontroll. For å løse dette er det brukt ett eksternt oppbevaringssted, GitHub. For hver gang det er gjort noen endringer på prosjektet, er dette blitt lagt inn som en endring i formen *commit* og lastet opp på nette. Dette er en god praksis da det gjør at man kan både være sikker på at kildekoden holdes trygg, og man kan nå koden fra flere lokasjoner om det skulle være nødvendig. Med disse *commit­-* ene som er lagt inn ligger det også med en kort melding som skal beskrive hva som er gjort i de ulike arbeidsøktene.

[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

[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.]

# Bibliografi

David J. Barnes, M. K. (2017). *Objects First with Java.* Harlow: Pearson.

[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.]

# 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.]