STUDENTRAPPORT

NTNU

KANDIDATNUMMER(E)/NAVN:			
Cathrine Kristiansen, Kandidatnummer: 10101 Meenakshi Jayachandran, Kandidatnummer: 10068			
DATO: 27/05/2026	FAGKODE: IDAATT2003	STUDIUM: BIDATA	ANT SIDER/BILAG: 24 / 0

FAGLÆRER(E):

Emneansvarlig/koordinator:
Atle Olsø

Faglærere:
Arne Gerhard Styve
Ivar Farup
Majid Rouhani

TITTEL :

Millions

SAMMENDRAG:

Denne rapporten beskriver utviklingen av Millions, et aksjespill utviklet i Java som en del av IDATT2003 ved NTNU. Prosjektet fokuserer på objektorientert programmering, JavaFX, designprinsipper, filhåndtering og testing. Resultatet ble en modulær desktop applikasjon hvor brukeren kan kjøpe og selge aksjer gjennom et grafisk brukergrensesnitt. Rapporten drøfter også erfaringer, utfordringer og bruk av teorier og designprinsipper i utvikling av løsningen.

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

Deklarasjon om KI-hjelpemidler

Har det i utarbeidingen av denne rapporten blitt anvendt KI-baserte hjelpemidler?

[ ] Nei

[X] Ja

Hvis *ja*: spesifiser type av verktøy og bruksområde under.

Tekst

[ ] **Stavekontroll**. Er deler av teksten kontrollert av:
Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai eller lignende verktøy?

[ ] **Tekstgenerering**. Er deler av teksten generert av:
ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr eller lignende verktøy?

[X] **Skriveassistanse**. Er en eller flere av ideene eller fremgangsmåtene i oppgaven foreslått av:
ChatGPT, Google Bard, Bing chat, YouChat eller lignende verktøy?

Hvis ja til anvendelse av et tekstverktøy - spesifiser bruken her:

Brukt for å fikse grammatikken.

Kode og algoritmer

[X] **Programmeringsassistanse**. Er deler av koden/algoritmene som i) fremtrer direkte i rapporten eller ii) har blitt anvendt for produksjon av resultater slik som figurer, tabeller eller tallverdier blitt generert av: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

Hvis *ja t*il anvendelse av et programmeringsverktøy - spesifiser bruken her:

KI (chatGPT) har blitt brukt som støtteverktøy til feilsøking, forklaringer av konsepter og forslag til mindre kodebiter. Koden er i hovedsak utviklet, tilpasset og integrert av oss selv.

Bilder og figurer

[X] **Bildegenerering**. Er ett eller flere av bildene/figurene i rapporten blitt generert av: *Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* eller lignende verktøy?

Hvis ja til anvendelse av et bildeverktøy - spesifiser bruken her:

PlantUML som støtte til å lage UML-diagrammer.

Andre KI-verktøy

[ ] **Andre KI-verktøy.** har andre typer av verktøy blitt anvendt? Hvis ja spesifiser bruken her:

X	Jeg er kjent med NTNUs regelverk: *Det er ikke tillatt å generere besvarelse ved hjelp av kunstig intelligens og levere den helt eller delvis som egen besvarelse.* Jeg har derfor redegjort for all anvendelse av kunstig intelligens enten i) direkte i rapporten eller ii) i dette skjemaet



Cathrine Kristiansen, Meenakshi Jayachandran, 26.05.2026, Trondheim

---

Underskrift/Dato/Sted

INNHOLD

Innholdsfortegnelse

1 Introduksjon 1
1.1 Bakgrunn 1
1.2 Kravspesifikasjon 1
1.3 Avgrensninger 4
1.4 Begreper/Ordliste 5
2 Teori 6
3 Metode 8
3.1 Utviklingsprosess 8
3.2 Verktøy 8
3.3 Bruk av KI verktøy 9
4 Resultat 9
4.1 Teknisk Design 9
4.2 Implementasjon 10
4.3 Testing 14
4.4 Utrulling til sluttbruker (deployment) 15
5 Drøfting 16
5.1 Drøfting av løsning/design 16
5.2 Drøfting av prosess 17
5.3 Drøfting av bruken av KI-verktøy 18
6 Konklusjon - erfaring 18

Figurliste

Figur 1: Use Case diagram 4
Figur 2: MVC/Arkitekturdiagram 10
Figur 3: Aktivitetsdiagram for spillflyt 11
Figur 4: Klassediagram som vider modell laget 12
Figur 5: Klassediagram som viser transaksjonssystemet 13
Figur 6: Sekvensdiagram som viser kjøp av askjer 13
Figur 7: Observer diagram 14
Figur 8: Brukertesting 15

Tabelliste

Tabell 1 Begreper og ordliste 6
Tabell 2: Verktøy 9

1 INTRODUKSJON

1.1 Bakgrunn

Investering og aksjehandel har blitt mer tilgjengelig gjennom digitale plattformer og applikasjoner. Samtidig kan det være vanskelig for nye brukere å forstå hvordan aksjemarkedet fungerer og hvordan investeringer påvirkes av prisendringer, risiko og strategi. Digitale simuleringer og spill kan bidra til økt forståelse ved å gi brukeren mulighet til å utforske aksjehandel i et mer kontrollert miljø.

Denne oppgaven som er en del av IDATT2003 ved NTNU, går ut på å utvikle et aksjespill hvor brukeren kan kjøpe og selge aksjer, følge utviklingen i markedet og administrere en egen portefølje gjennom et grafisk brukergrensesnitt.

1.2 Kravspesifikasjon

Funksjonelle krav:

Oppstart av spill:

- Starte et nytt spill
- Registrere spillnavn
- Velge startkapital

Aksjemarked og statistikk:

- Vise tilgjengelige aksjer på børsen
- Søke etter aksjer basert på navn eller symbol
- Vise statistikk for aksjer:
    - Høyeste pris
    - Laveste pris
    - Prisutvikling
    - Ukentlig endring

- Vise ukens vinnere og tapere

Handel

- Kjøpe aksjer
- Selge aksjer
- Validering av:
    - Tilstrekkelig saldo ved kjøp
    - Eier aksjer ved salg
- Vise kostnader og verdier før handel gjennomføres
- Vise kvittering eller fullført transaksjon

Portefølje og status

- Vise brukerens portefølje
- Vise nettoverdi i sanntid
- Vise spillerstatus basert på progresjon

Transaksjon

- Lagre transaksjonshistorikk
- Vise tidligere transaksjoner

Spillflyt

- Avansere til neste uke
- Oppdatere aksjepriser automatisk
- Avslutte spiller

Ikke-funksjonelle krav:

Brukervennlighet

- Intuitivt grafisk brukergrensesnitt
- Informasjon være lett tilgjengelig
- Gi tydelig feilmeldinger ved ugyldig input
- Responsiv og oversiktlig layout

Robusthet

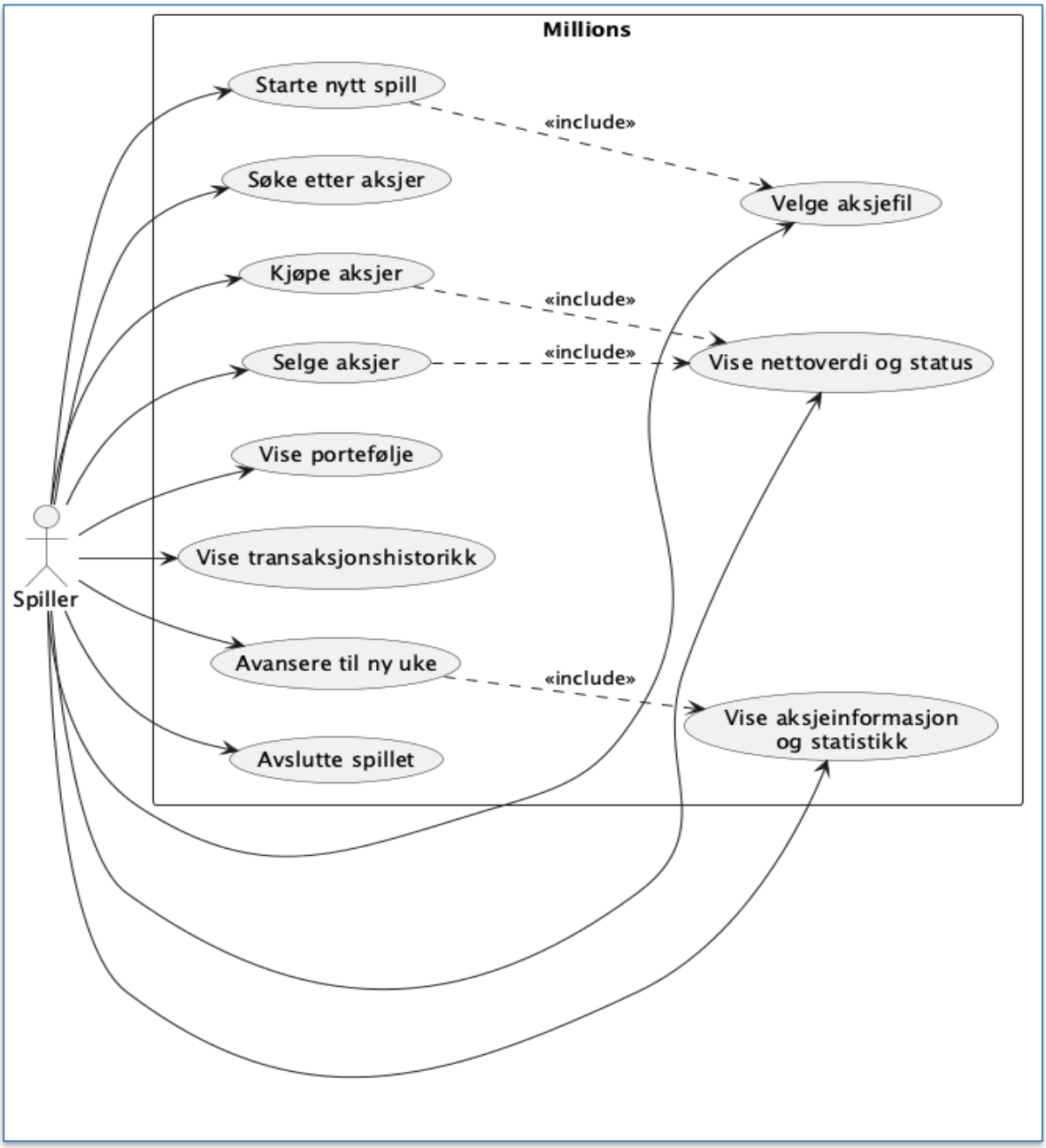
- Håndtere ugyldig input uten krasj
- Feilsituasjon håndteres med informative meldinger

Kodekvalitet

- Modulær og lett å vedlikeholde
- Tydelig og strukturert kode
- Kode i tråd med Checkstyle
- JavaDoc på klasser og offentlige metoder

Kjøring

- Bygges og kjøres med maven



Figur 1: Use Case diagram

1.3 Avgrensninger

Prosjektet er utviklet innenfor rammene og kravene gitt i mappevurderingen for IDATT2003. Derfor er løsningen avgrenset til teknologier, funksjonalitet og designkrav spesifisert i oppgaven, bl.a. bruk av Java, JavaFX, Maven og lokal filhåndtering gjennom CSV filer

1.4 Begreper/Ordliste

Begrep (Norsk)	Begrep (Engelsk)	Betydning/Beskrivelse
Aksje	Stock	Aksje som kan handles på børsen. Inneholder symbol, navn og prisutvikling.
Andel	Share	En mengde aksje eid av en spiller. Inneholder informasjon om kvantitet og kjøpspris.
Spiller	Player	Brukeren i spillet.
Portefølje	Portfolio	Andeler som spilleren eier.
Transaksjon	Transaksjon	Kjøp eller salg av aksjer.
Kjøp	Purchase	Transaksjon hvor spilleren kjøper aksjer fra porteføljen sin.
Salg	Sale	Transaksjon hvor spilleren selger aksjer fra porteføljen sin.
Børs	Exchange	Markedet hvor aksjer handles og prisene oppdateres ukentlig.
Nettoverdi	Net Worth	Summen av spilleren sin penger og verdien av porteføljen.
Kurtasje	Commission	Avgift som betales ved kjøp eller salg av aksjer.
Skatt	Tax	Avgift som trekkes ved gevinst på salg av aksjer.
Status	Status	Spilleren sin progresjon i spillet.
Handelsuke	Trading Week	Runde i spillet hvor aksjepriser kan endres og transaksjoner gjennomføres.
Prisendring	Price Change	Differansen mellom tidligere og nåværende aksjepris.
Novise	Novice	Startnivå i spillet.
Investor	Investor	Statusnivå for spillere som har handler i minst 10 uker og økt nettoverdien sin med 20%.
Spekulant	Speculator	Høyeste statusnivå, krever minst 20 handelsuker og at spilleren har doblet nettoverdien sin.

Tabell 1 Begreper og ordliste

2 TEORI

Ved utvikling av Millions spillet har det blitt brukt flere teorier innen objektorientert programvareutvikling. Teoriene danner grunnlaget for hvordan robuste, vedlikeholdbare og fleksible systemer kan designes og implementeres. Nedenfor har jeg lagt fram teoriene som ligger i grunn for designet.

Responsiblity Driven Design (RDD)

RDD er en objektorientert programvare designmetode som modellerer systemer ved å tilpasse spesifikke ansvarsområder til objekter basert på deres roller og interaksjoner.[1] Hva objekter gjør og hvordan de samarbeider for å oppfylle systemmål, er det som bli prioritert.

Model View Controller – MVC

MVC mønsteret er en programvare arkitekt metode som separer applikasjonen i tre hoved komponenter. [2] Hver komponent har ansvar for en spesifikk aspekt spillet sin funksjonalitet: Model, View og Control.

Arv og Polymorfi

Arv er prosessen med å opprette en primærklasse (foreldreklasse) som andre klasser (barneklasser) kan opprettes fra. Som et resultat får vi polymorfisme, og refererer til prosessen der en barneklasse modifiserer det den arver fra en foreldreklasse.[3] Når man bruker de sammen, reduserer mengden av kode du må skrive i et program og gir en mer bedre kohesjon mellom klassene.

Robusthet og Exception Handling

Unntakshåndtering er metoden for å bygge et system fr å oppdage og gjenopprette fra eksepsjonelle forhold.[4] Dette betyr at koden skal kunne validere argumenter, bruk av Exceptions til å signalisere feil, returnere kopier for å beskytte interne data..

Filhåndtering

Filhåndtering betyr å jobbe med filer, som få opprette dem, lese data, skrive data eller slette dem.[5] Dette hjelper et program med å bruke informasjonen eller lagre den permanent i datamaskinen.

Brukergrensesnitt - GUI

GUI er et grensesnitt mellom en bruker og en datamaskin eller et nettsted som bruker en rekke visuelle elementer for å utføre datakommandoer. Et GUI lar brukere uten programmeringskunnskaper samhandle med datamaskinen.

SOLID-prinsippene

Består av fem essensielle retningslinjer for å forbedre programvaredesignet og gjøre koden mer fleksibel og vedlike holdbar.[6]

Single Responsiblity Principle – S

Prinsippet går ut på at en laksklasse se skal kun ha ett ansvar/hensikt. Som gjør det lettere å håndtere og teste klassene.

Open/Closed Principle – O

Prinsippet går ut på at programvarenheter skal være åpne for utvidelse, men ikke åpen for modifikasjoner av andre klaser. Så når en funksjonalitet legges til, skal ikke den eksisterende koden endres, men utvidet gjennom nye komponenter.

Kobling og Kohesjon

Kobling referer til graden av gjensidig avhengighet mellom programvaremoduler. [7] Høy kobling gjør modulene avhengig av hverandre, der en forandring påvirker andre moduler. Dermed er lav kobling ønsket.

Kohesjon refererer i hvilken grad elementer i en modul fungerer sammen for å oppfylle et enkelt, veldefinert formål.[7] Lav Kohesjon betyr at det utføres man ulike oppgaver som ikke henger sammen, som fører til kompleksitet og sårbarhet i koden. Derfor er høy kohesjon ønsket, men med ett tydelig og avgrenset mål.

3 METODE

3.1 Utviklingsprosess

Millions sin utviklingsprosess fulgte en iterativ utviklingsprosess, der den ble utviklet gjennom flere deler av mappevurderingen. Prosessen gjorde det mulig det mulig å forbedre og videreutvikle løsningen kontinuerlig gjennom semesteret.

Arbeidet startet med planlegging av grunnleggende modellklasser og forretningslogikk. Videre ble løsningen utvidet med filhåndtering, statistikk og statusberegninger. Til siste del ble fokus rettet mot GUI utvikling med JavaFX, MVC arkitektur, designmønstre og videreutvikling av funksjonalitet. Underveis var det tilbakemeldingssamtaler som var en viktig del av utviklingsprosessen.

Prosessen fokuserte på å utvikle, teste og forbedre, som ga oss god kontroll over fremdriften våres. Det ble brukt forskjellige verktøy som SonarLint, JavaDoc, Checkstyle og navngivningskonvensjoner som støtte for kodekvalitet og struktur.

3.2 Verktøy

Verktøy	Bruksområde
Java	Implementasjon av programmet
JavaFX	Grafisk brukergrensesnitt
Maven	Bygging, Testing og pakking av programmet
JUnit	Enhetstesting
Git	Versjonskontroll
IntelliJ IDEA	Utviklingsmiljø
SonarLint / CheckStyle	Kvalitetssikring av kode

Tabell 2: Verktøy

3.3 Bruk av KI verktøy

Brukt som støtte under utviklingen, spesielt feilsøking, forklaring av konsepter, og forslag til mindre kodebiter og løsninger. Spesielt nyttig ved arbeid med JavaFX, GUI design og løsninger. KI ble brukt som støtte ved utarbeiding av UML diagrammer, README og strukturering av rapporten. All kode og tekst som ble brukt i prosjektet ble gjennomgått, tilpasset og integrert manuelt.

4 RESULTAT

4.1 Teknisk Design

Løsningen er basert på Model-View-Control, hvor forretningslogikk, brukergrensesnitt og kontrollflyt er separert i egne komponenter. Arkitekturen ble valgt for å skape en mer strukturert, vedlikeholdbar og modulær løsning. Som bidrar til lav kobling mellom komponentene og gjør løsningen lettere å teste, videreutvikle og vedlikeholde.

Teknisk design for Millions spillet bygger på MVC-arkitekturmodellen, hvor forretningslogikk, brukergrensesnitt og kontrollflyt er separert i egne komponenter.

1. Modell-laget

   Håndterer forretningslogikk og data i systemet. Som f.eks. aksjer, porteføljer, transaksjoner og spillerinformasjon. Inneholder kjernelogikken i spillet og er ansvarlig for behandling og lagring av data knyttet til aksjehandel og statusberegninger.

2. View-laget

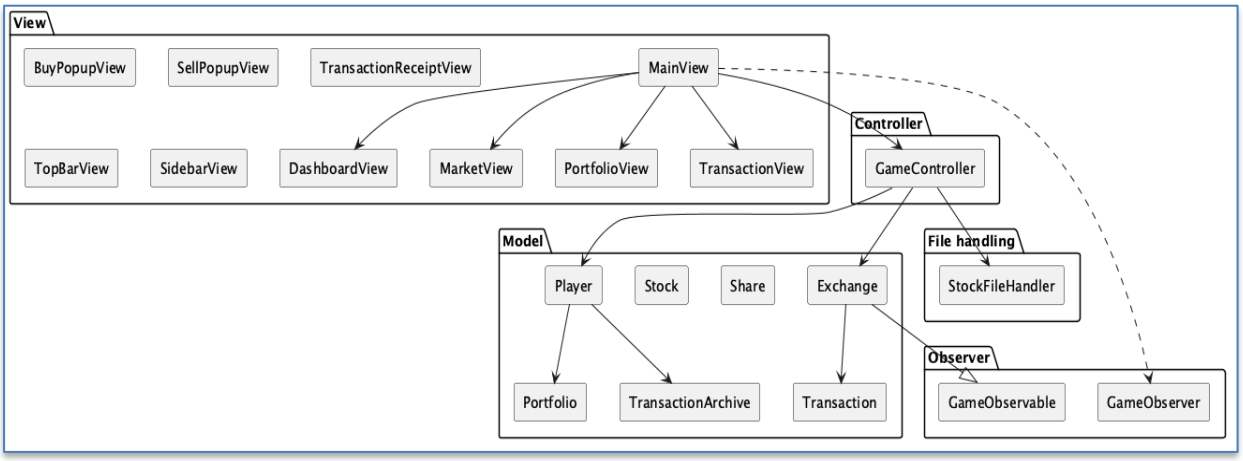
   JavaFX baserte brukergrensesnitt som presenterer informasjon og mottar input fra brukeren. GUI-et er delt opp i flere separate komponenter for å gjøre løsningen mer oversiktlig.

3. Controller-laget

Mellomledd mellom modell og View, og håndterer brukerhandlinger og sørger for at endringer i modellen reflekteres i brukergrensesnittet.

Aksjedata håndteres gjennom lokal filhåndtering ved bruk av CSV-filer. Filhåndteringen er separert av resten av forretningslogikken for å gjøre løsningen mer modulær og enklere å utvide.

Brukergrensesnittet benytter observerbare komponenter for å støtte dynamiske komponenter. Dette betyr at HUI-et oppdateres automatisk når data i modellen endre, uten av View komponentene trenger tilgang til den interne logikken.



Figur 2: MVC/Arkitekturdiagram

4.2 Implementasjon

Implementasjon av millions består av flere komponenter som samarbeider for å håndtere aksjehandel, brukerinteraksjon og oppdatering av systemet. Systemet er delt opp i flere pakker basert på deres ansvar og rolle i applikasjonen, bl.a. modellklasser, kontrollklasser, filhåndtering, transaksjoner og brukergrensesnittet.

Modell-laget inneholder sentrale domeneklasser i systemet. Klassene håndterer forretningslogikken på markedet.

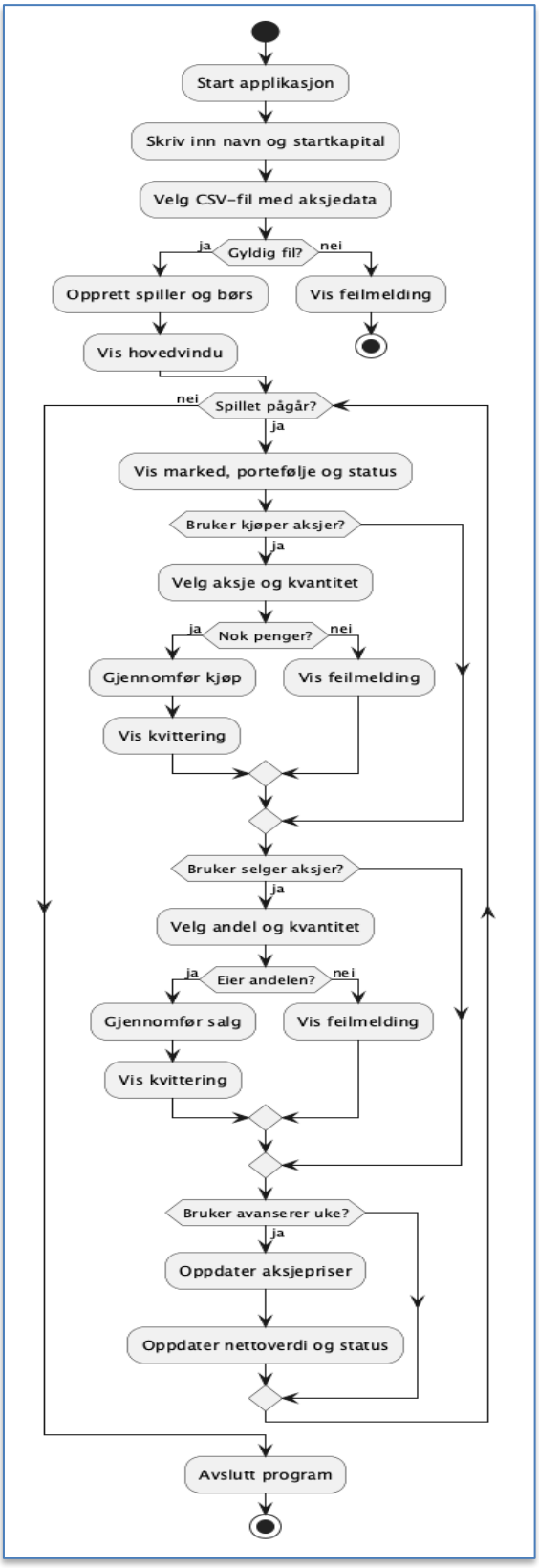
Transaksjonssystemet er bygget rundt en abstrakt Transaction klasse med subklassene Purchase og Sale. Som gjør det mulig å samle felles funksjonalitet i superklassen samtidig som at sub-klassene kan implementere ulik logikk. Egne kalkulatorklasser brukes for å beregne verdier, kurtasje og skatt knyttet til transaksjoner.

GUI er implementert med JavaFX og følger MVC-arkitekturen. Den er delt opp i flere View klasser som håndterer ulike deler av systemet.

Controller fungerer som et mellomledd mellom modell og view håndterer brukerhandlinger. Som gjør at GUI-et ikke trenger direkte tilgang til forretningslogikken i systemet.

Observer brukes for å støtte dynamiske oppdateringer av brukergrensesnittet. Når data endres, varsles observatør slik at relevant informasjon i GUI oppdateres sanntid.

Factory er implementert gjennom egne komponenter som håndterer lesing og skriving av aksjedata til CSV filer.

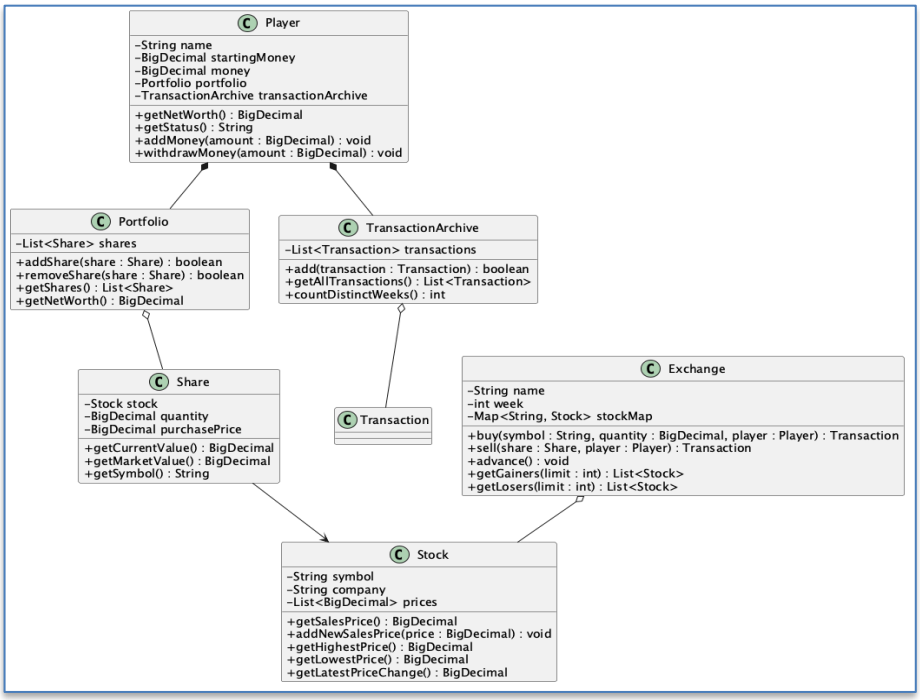


Figur 3: Aktivitetsdiagram for spillflyt

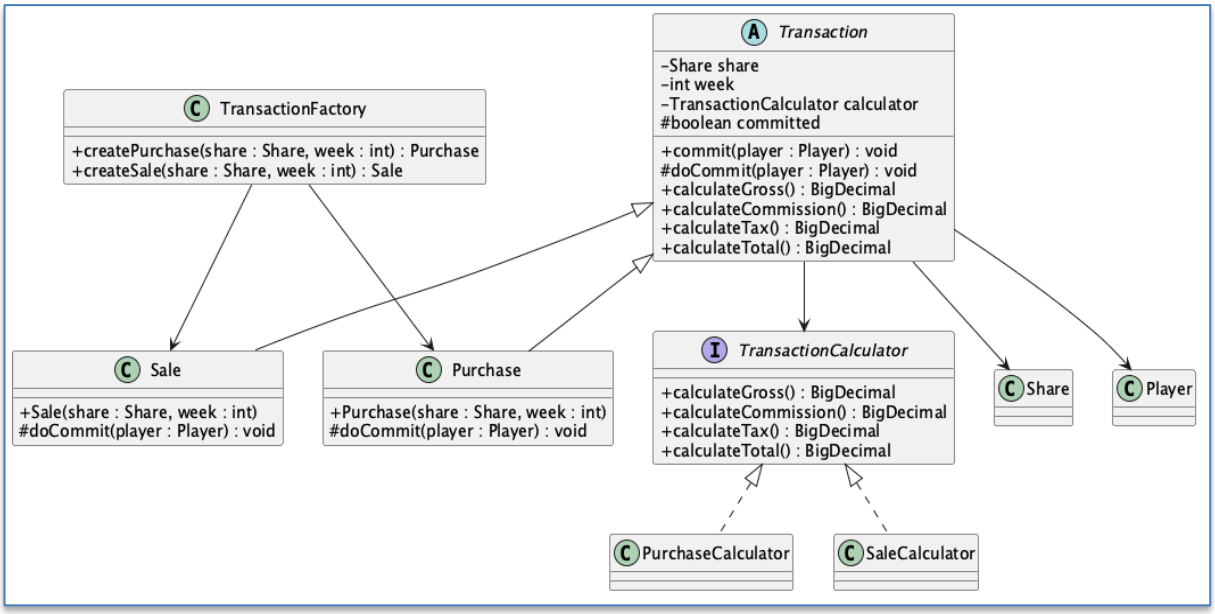
Løsningen vil validere filformat og innhold ved lasting for å redusere risikoen for ugyldig data.

Gjennom hele implementasjonen er det tatt i bruk prinsipper fra:

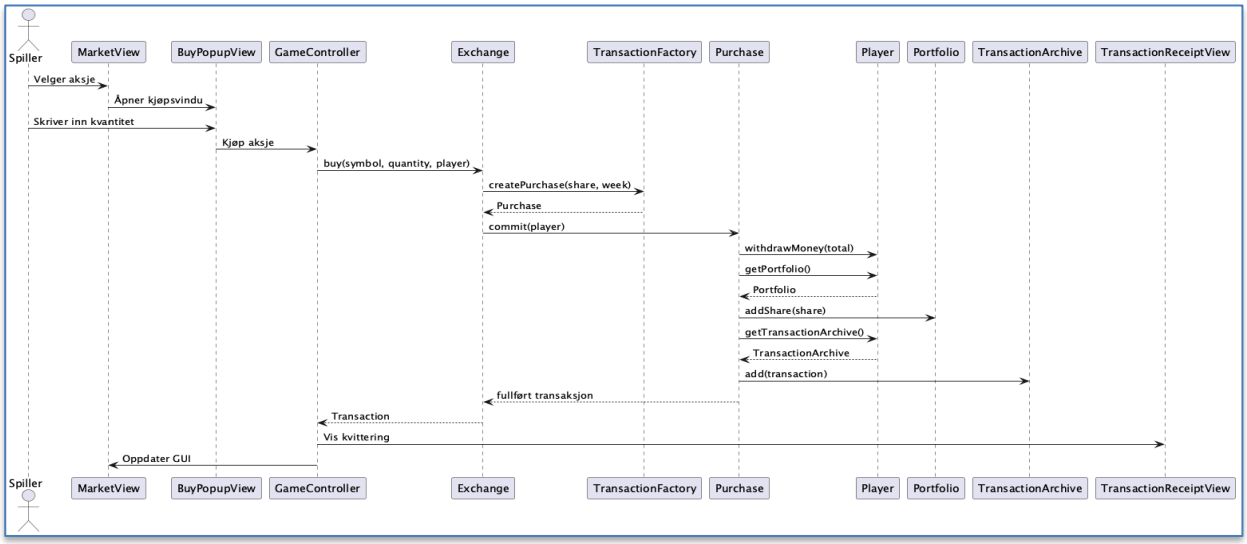
- Høy kohesjon gjennom modell-, view-, og Controller-klasser
- Lav kobling mellom MVC lagene
- Innkapsling
- Arv og polymorfi
- Observer pattern og Factory pattern
- SOLID-prinsipper.
- Robusthet gjennom validering av input, exceptions og feilmeldinger.
- Modulær filhåndtering
- Responsiv og oversiktlig GUI
- Maven strukturering av prosjektet



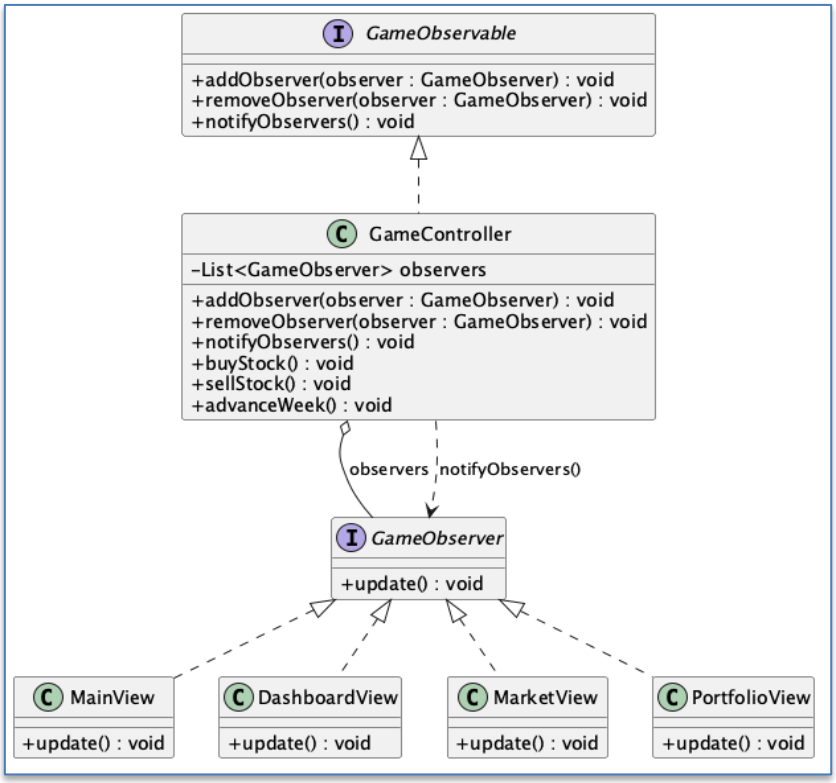
Figur 4: Klassediagram som vider modell laget



Figur 5: Klassediagram som viser transaksjonssystemet



Figur 6: Sekvensdiagram som viser kjøp av askjer



Figur 7: Observer diagram

4.3 Testing

Løsningen ble testet gjennom enhetstesting og brukertesting. Enhetstestene ble utviklet med JUnit og fokuserte på forretningslogikk som transaksjoner, portfeøljehåndtering, statistikk og filhåndtering. Det var både positive og negative tester for å valider forventet oppførsel og håndtering av ugyldige situasjoner.

Brukertesting ble gjennomført med en ekstern bruker med erfaring innen programvareutvikling. Testpersonen sin oppgave var å navigere gjennom spillet og gi tilbakemelding på brukeropplevelsen og GUI design. Tilbakemeldingen var at spillet var lett å navigere og forstå, men at enkelte pop-up vinduer burde hatt et mer konsistent design med resten av GUI. Det ble også foreslått bedre filtrering og større visning av aksjer i Marketplace delen av applikasjonen.

USERS			**Experience/User Profile**	
Anonym			*Born in 1993. Is a senior software engineer at Telenor, Trondheim.*	
Date	Candidate	Testing type	Instructions	Conclutions
28.03.2026	Anonym	GUI	Prøv å navigere deg gjennom spillet. Er det forståelig, hva ville du endret?	Veldig bra spill til å være et av deres første GUI prosjekter. Lett å navigere og forstå. Ville endret alle ruter som kvitteringer etc. til å inneholde samme layout som resten av spillet. Det er vanskelig å se alle stocks i kortet på Marketplace, denne burde vært større. Fint om man kan filtrere i kortene på pris, change, navn etc. slik at man kan få den største verdien først av den valgte attributten.

Figur 8: Brukertesting

4.4 Utrulling til sluttbruker (deployment)

Millions er utviklet som en desktop applikasjon i Java. Selve prosjektet er strukturert som et Maven prosjekt og kan bygges, testes og kjøres gjennom maven kommando på terminal eller gjennom eget utviklingsmiljø (IDE).

Applikasjonen kjøres ved bruk av kommandoen mvn javafx:run eller Play knappen på egen IDE. Som vil starte JavaFX applikasjonen og dermed laste inn brukergrensesnittet. Maven vil håndtere dependencies, bygging og kjøring.

Løsningen er utviklet for å kunne kjøres på systemer som har Java og Maven installert. aksjedata håndteres lokalt gjennom CSV filer som brukeren velger selv ved oppstart av spillet.

5 DRØFTING

5.1 Drøfting av løsning/design

Etter kravspesifikasjonen ser man at løsningen vår oppfyller de fleste kravene. Systemet støtter bl.a. kjøp og salg av aksjer, visning av portefølje, statistikk, transaksjonshistorikk og dynamisk oppdatering av GUI. Det ble også implementert ekstra funksjonalitet som pop-up baserte transaksjoner og forbedret søk og filtrering.

Under utvikling ble lagt vekt på ansvarsfordeling mellom klassene med RDD og prinsippene om høy kohesjon. Som gjør klassene lettere å vedlikeholde og teste.

MVC arkitekturen bidro til lavere kobling mellom GUI og forretningslogikk. GUI komponentene kommuniserer med modellen gjennom controller-laget istedenfor direkte koblinger. Observer ble brukt for å oppdatere GUI-et dynamisk når data endrer seg.

Arv og polymorfi ble brukt i transaksjonssystemet gjennom den abstrakte Transaction og subklassene Purchase og Sale. Som gjør det mulig å samle felles funksjonalitet samtidig som salg kunne implementere ulik logikk. Denne løsningen følger spesielt Single Responsiblity Principle.

Robusthet var et viktig fokus. Systemet validerer bl.a. om spilleren har nok penger ved kjøp av aksjer, og at spilleren eier aksjer før salg. Ved filhåndtering for vi validert filformat og innhold som reduserer risikoen for ugyldig data. Exceptions blir brukt for å håndtere feilsituasjoner og gi brukeren tydelige feilmeldinger.

Enhetstesting var jevnlig brukt for å kvalitetssikre deler av forretingslogikken. Forskjellige prøvesvar gjorde det mulig å validere både forventet oppførsel og håndtering av ugyldige situasjoner.

Løsningen er modulær og fleksibel, dette gjør at det er områder som kunne ha vært forbedret videre. Enkelte controller-klasser har fortsatt mange ansvarsområder, og løsningen kunne ha brukt interfaces mer for å redusere koblingen mer.

5.2 Drøfting av prosess

Prosessen fulgte en iterativ prosess. Siden mappevurderingen var delt it re deler med tilbakemeldingssamtaler underveis, fungerte denne arbeidsformen godt. Løsningen ble stegvis utviklet samtidig som tidligere kode ble forbedret etter hver del.

Det som fungerte best, var oppbygning av prosjektet. Ved å teste og implementere underveid, ble det lettere å oppdage feil tidlig og videreutvikle. Versjonskontroll med GitHub gjorde det enklere å holde oversikt over endringer gjennom prosjektet.

Samtidig oppsto det enkelte utfordringer under utviklingen bl.a. knyttet til JavaFX, MVC arkitekturen, Merge conflicts. Etter hvert som prosjektet ble større, måtte flere deler av controller- og view-strukturen oppdateres for å redusere kompleksitet og forbedre oversiktligheten.

En svakhet ved koden vår er at GameController ha flere ansvarsområder. En forbedring ville vært å dele controller laget i mindre kontrollere, f.eks. TradingController og FileController, slik at RDD følges bedre.

Dersom Prosjektet skulle blitt utviklet på nytt, ville vi ha planlagt GUI strukturen tidligere i prosessen og brukt mer tid på diagrammer og arkitektur før implementasjonen startet. Hvis vi også hadde flere tester av GUI komponenter kunne vi ha forbedret kvalitetssikringen ytterligere.

5.3 Drøfting av bruken av KI-verktøy

Med begrenset erfaring innenfor områder som JavaFX og brukergrensesnitt valgte vi å bruke KI-verktøy for å støtte utviklingsprosessen og utarbeide prosjektet mer effektivt. Formålet med å benytte verktøyet var å sikre hjelp til feilsøking, forstå konsepter og forslag til løsninger.

KI-verktøyet ble brukt som et hjelpemiddel for å kunne forstå hvordan ulike deler av systemet kunne implementeres. Målet med å bruke KI var derfor å støtte læring og utvikling, fremfor å generere ferdig kode. Det ble også oppdaget at KI genererte forslag enkelte ganger var for generelle eller ikke tilpasset nødvendig som gjorde manuelt kvalitetssikring nødvendig. Koden som ble brukt i systemet er derfor gjennomgått, tilpasset og integrert av oss selv.

6 KONKLUSJON - ERFARING

Gjennom utvikling av Millions ble de fleste kravene i kravspesifikasjonen oppnådd. Prosjektet ga oss erfaring med objektorientert design, designmønstre, filhåndtering og testing av programvare.

Dersom prosjektet skulle bli gjort på nytt, ville vi ha planlagt GUI og systemarkitektur tidligere. Løsningen er også begrenset lokal CAV fil og mangler støtte for database eller lagring av spill tilstand.

Ved videreutvikling kunne prosjektet blitt utvidet med mer avansert persistens, bedre GUI testing og flere funksjoner for visualisering av aksjedata.

REFERANSER

[1] «Responsibility-Driven.pdf». Åpnet: 24. mai 2026. [Online]. Tilgjengelig på: https://www.wirfs-brock.com/PDFs/Responsibility-Driven.pdf
[2] «MVC Design Pattern», GeeksforGeeks. Åpnet: 7. desember 2025. [Online]. Tilgjengelig på: https://www.geeksforgeeks.org/system-design/mvc-design-pattern/

[3] «Inheritance and polymorphism in object-oriented programming», Unity Learn. Åpnet: 24. mai 2026. [Online]. Tilgjengelig på: https://learn.unity.com
[4] «Topic: Exception Handling». Åpnet: 24. mai 2026. [Online]. Tilgjengelig på: https://users.ece.cmu.edu/~koopman/des_s99/exceptions/
[5] «File Handling in Java», GeeksforGeeks. Åpnet: 24. mai 2026. [Online]. Tilgjengelig på: https://www.geeksforgeeks.org/java/file-handling-in-java/
[6] «SOLID Principles with Real Life Examples», GeeksforGeeks. Åpnet: 6. desember 2025. [Online]. Tilgjengelig på: https://www.geeksforgeeks.org/system-design/solid-principle-in-programming-understand-with-real-life-examples/
[7] «Coupling and Cohesion - Software Engineering», GeeksforGeeks. Åpnet: 6. desember 2025. [Online]. Tilgjengelig på: https://www.geeksforgeeks.org/software-engineering/software-engineering-coupling-and-cohesion/