

CDIO - DEL 3

Kurser

02312-14 Indledende programmering

02313 Udviklingsmetoder til IT-systemer

02315 Versionsstyring og Testmetoder

Studiegruppe 33

s185029 Andreas Hyldahl

s185128 Karoline Emilie Thomsen

s185035 Peter Bach

s152716 Rasmus Strange Jakobsen

s17128 Sascha Thorsgaard Jacobsen

Rapport

Rapporten må gerne benyttes til undervisning



Sascha



Peter



Karoline



Rasmus



Andreas

Timeregnskab

	Analyse/Design	Implementering	Test	Rapport	I alt
Peter	1	10	1	3	15
Rasmus	3	3	1	5	12
Andreas	3	20	1	4	28
Sascha	3	15	1	6	25
Karoline	7	3	1	6	17

Indholdsfortegnelse □

Timeregnskab	2
1. Indledning	3
2. Problemformulering	4
3. Metode	4
4. Analyse	4
4.1 Krav:	4
4.2 Use case-diagram	6
4.3 Domænemodel:	8
4.4 Systemsekvensdiagram	8
5. Design	9
5.1 Designklassediagram	9
5.2 Sekvensdiagram	10
5.3 GRASP-mønstre	10
6. Implementering	11
6.1 Branching strategi	11
6.3 Kode	11
7. Test	12
8. Brugervejledning	12
10. Konklusion	12
Litteraturliste	12
Bøger	12
Hjemmesider	12
Opgaver	13
Bilag	13
Bilag A	13

1. Indledning

Rapporten har til formål at give et indblik i tankerne bag design, implementering, samt test af vores Monopoly-spil. Spillet går kort sagt ud på, at 2-4 personer skal kaste med en terning og derefter lande på et felt, der svarer til en ejendom, de derefter skal købe. Hvis der allerede er en anden spiller, der ejer den ejendom man er landet på, skal man betale et beløb til denne spiller. Vinder er den der har flest penge, efter den første spiller er gået bankerot.

For at danne overblik over klasserne og deres relation, benyttes et designklassediagram og sekvensdiagram i samspil med vurdering af GRASP-mønstre, mens en use-case-model er anvendt til at erklære forhold mellem spillet og dets aktører.

Der ønskes tests på om optælling af score virker som forventet.. Derudover om der bliver fundet en vinder og en taber i overensstemmelse med spillets regler.

2. Problemformulering

Opgaven lyder på at programmere et Monopoly Junior spil for IOOuterActive. Spillet skal kunne spilles på databaserne på DTU mellem 2-4 personer. Spillet går ud på, at man skal kaste med en terning, og ud fra det viste øjental flytter spillerne sig rundt på brættet. Spillerne kan lande på en ejendom, et chance felt eller et passivt felt. Når en spiller lander på en ejendom skal de købe den, medmindre den allerede er ejet af en anden spiller. Hvis der allerede er en ejer, skal spilleren betale leje til denne ejer for at lande på deres ejendom.

Hver spiller modtager en startbeholdning, der er afhængig af antallet af spiller. Derudover får man tildelt penge, når man passerer start-feltet. Spillet slutter når en spiller ikke har penge til at købe en ejendom eller betale en leje. Vinder er den spiller der har den største score på dette tidspunkt.

Der skal testes på ovenstående krav og laves mindst én brugertest. Spillet skal bygges op omkring GRASP-principper og med dokumentation for dette. Derudover ønsker IOOuterActive en Graphic User interface.

3. Metode

Projektet er startet ud med en domænemodel samt et designklassediagram for at skabe overblik over klasserne og deres relationer. Hensigten herfor, var at opnå lavest mulig kobling for bedre overblik. Dette er dog ikke overholdt til fulde, hvorfor vi er stødt ind i problemer med hvilke metoder der skal creates samt kaldes i henholdsvis spiller og account klasserne.

Spillets regler er delvist forenklet, hvilket eksempelvis fremgår under chancekortene som endnu ikke er implementeret ordentligt og derfor ikke hænger sammen med gameboardet.

Hensigten er, at gameboardet skal spille sammen med movePlayer metoden fra Player-klassen, hvorefter vi ved et polymorfisk kald skal kunne kalde den respektive landOnField-metode for de enkelte felter i gameboardet.

4. Analyse

I analysen finder vi ud af hvilke krav kunden har til systemet og hvordan de kunne tænke sig det skal interagere. Det giver et overblik over kravene, hvilke der er vigtige og hvilke der er mindre vigtige, samt ved hjælp af modeller og diagrammer, at kunne analysere hvad kunden kunne tænke sig og hvordan dette kan lade sig gøre.

4.1 Krav:

Følgende viser en kravspecifikationsliste over kravene, som kunden har til, hvad systemet og spillet skal kunne.

Funktionelle krav

Vi har lavet en prioritering af kravene i forhold til hvad systemet “must have”, “should have”, “could have” og “want to have”, hvor det vigtigste er “must have” og “should have”.

Must have:

- 2-4 spillere i spillet. Systemet skal kunne kende forskel på de 2-4 spillere.
- Systemet skal indeholde og kaste med en ægte terning.
- Spillet slutter når en spiller lander på et felt og ikke har råd til at betale for konsekvensen.
- Vinderen er den, der har flest penge, når en modspiller går fallit.
- Efter hvert kast går turen videre til den næste spiller. Turen går med uret.

Should have:

- Spillets regler overholdes (se bilag A)
- Hver spillers pengebeholdningen starter på 16, 18 eller 20 afhængig af antal spillere.
- Man rykker antal felter frem tilsvarende det antal øjne terningen viser.
- Er feltet en grund, som skal købes såfremt den er ledig. Ellers betales grundens værdi til ejeren.
- Ejendomme forekommer i par. Ejers begge ejendomme af samme spiller, fordobles betalingen.
- Hvis man selv ejer grunden sker der ingenting.
- Alle spillere starter på felter “START”.

Could have:

- Felterne “på besøg” og “gratis parkering” har samme udfald i form af ingenting.
- Feltet “gå i fængsel” sætter spillere på fængselsfeltet, og der betales M1 for at komme ud næste runde. Man kan godt modtage leje mens man er i fængsel.
- Chancefelter udløser et chancekort, som varierer i instrukser.

Want to have:

- Den yngste spiller starter

Ikke-funktionelle krav

- Skal kunne spilles på DTU’s computere.
- Ingen forsinkelser.
- Der skal laves en klasse GameBoard, der kan indeholde alle felterne i et array. Tilføj en toString metode, der udskriver alle felterne i arrayet.
- Kunden ønsker dokumentation for test og overholdt GRASP mønstre.
- Der skal laves mindst tre testcases med tilhørende testscripts og testrapporter.

- Der skal være mindst en brugertest.
- Benyt GUI.

Usability

- Brugsanvisning - Brugerne skal kunne se hvordan spillet spilles
- Spilleets regelsæt.

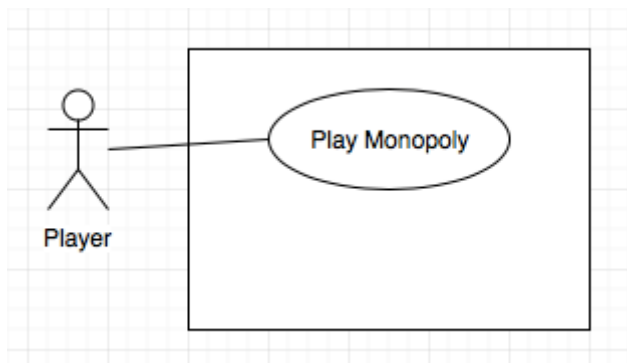
Ufuldstændige beskrivelser:

(undladelse af regler? / væsentligste elementer?)

Vi får at vide, at vi gerne må udelade nogle regler, samt prioritere dem. Dette er ufuldstændigt i forhold til at vi i teorien kunne udelade mange regler men som måske ville være bedre hvis de var med. Det ville have været bedre hvis vi fik oplyst en procentdel af hvor mange regler der i hvert fald skal med, og stadig selv prioritere hvilke, der er vigtigst for at spillet kan køre.

Derudover beder de os også om at implementere de vigtigste elementer for at spillet kan spilles. Dette måtte også gerne være uddybet mere, i forhold til hvad kunden synes, der er vigtige elementer, og kunden havde måske selv en ide til hvilke elementer, der er vigtige for at spillet kan spilles.

4.2 Use case-diagram



Use case diagrammet viser, at systemet består af en use case, som er forbundet med en aktør. Ud fra kravene er det bestemt, at spillet skal kunne spilles af 2-4 spillere på samme tid. Hver spiller får tildelt deres egen figur til spillet. Aktøren "Player" kan altså fremgår op til 4 gange.

Nedenfor ses en brief use case beskrivelse.

Play Monopoly

Et Monopoly-spil med 2-4 spillere, som slår med en terning.

En spiller starter med en pengebeholdning på 20, hvis man er to spillere, 18 hvis man er tre spillere og 16 hvis man er fire spillere. Hvis en spiller ikke har råd til at betale husleje eller købe en ejendom, som spilleren lander på, er spillet slut. Spillet slutter ligeledes hvis en spiller ikke kan betale afgift fra et chancekort. Spilleren med flest penge vinder.

Nedenfor ses en fully-dressed beskrivelse af use casen.

Use case: PlayMonopoly
ID: 1
<p>Kort beskrivelse:</p> <p>Et Monopoly-spil med 2-4 spillere, som slår med en terning.</p> <p>En spiller starter med en pengebeholdning på 20, hvis man er to spillere, 18 hvis man er tre spillere og 16 hvis man er fire spillere. Hvis en spiller ikke har råd til at betale husleje eller købe en ejendom, som spilleren lander på er spillet slut. Spillet slutter ligeledes hvis en spiller ikke kan betale afgift fra et chancekort. Spilleren med flest penge vinder.</p>
<p>Primær aktør:</p> <p>En spiller.</p>
<p>Andre interessenter:</p> <p>IOOuterActive.</p>
<p>Preconditions:</p> <p>Starter spillet op i en af DTU's databaser, og spillerne indtaster deres alder og vælger en figur.</p>
<p>Main flow:</p> <ol style="list-style-type: none"> 1. Spillet starter når en af spillerne sætter systemet til at kaste terningerne. 2. Systemet viser antallet af øjne. 3. Hertil lander spilleren på et felt, hvis konsekvens varierer. De har næsten alle en påvirkning på spillerens pengebeholdning. 4. Systemet lægger det nye over-/underskud til spillerens pengebeholdning. 5. Hvis en spiller lander på et felt hvis konsekvens betyder, at han/hun ikke har flere penge, slutter spillet. <ol style="list-style-type: none"> 5.1 Vinderen er den af modstanderne, der nu har flest penge. 5.2 Ellers (else) skifter systemet spiller og starter forfra i main flow.
<p>Postconditions:</p> <p>None.</p>
<p>Alternative flows:</p> <ol style="list-style-type: none"> 1. Hvis man lander på felterne "på besøg" eller "gratis parkering" sker der ingenting 2. Hvis man lander på feltet "gå i fængsel", skal spilleren betale M1 for at komme ud næste runde. Man kan godt modtage leje mens man er i fængsel.

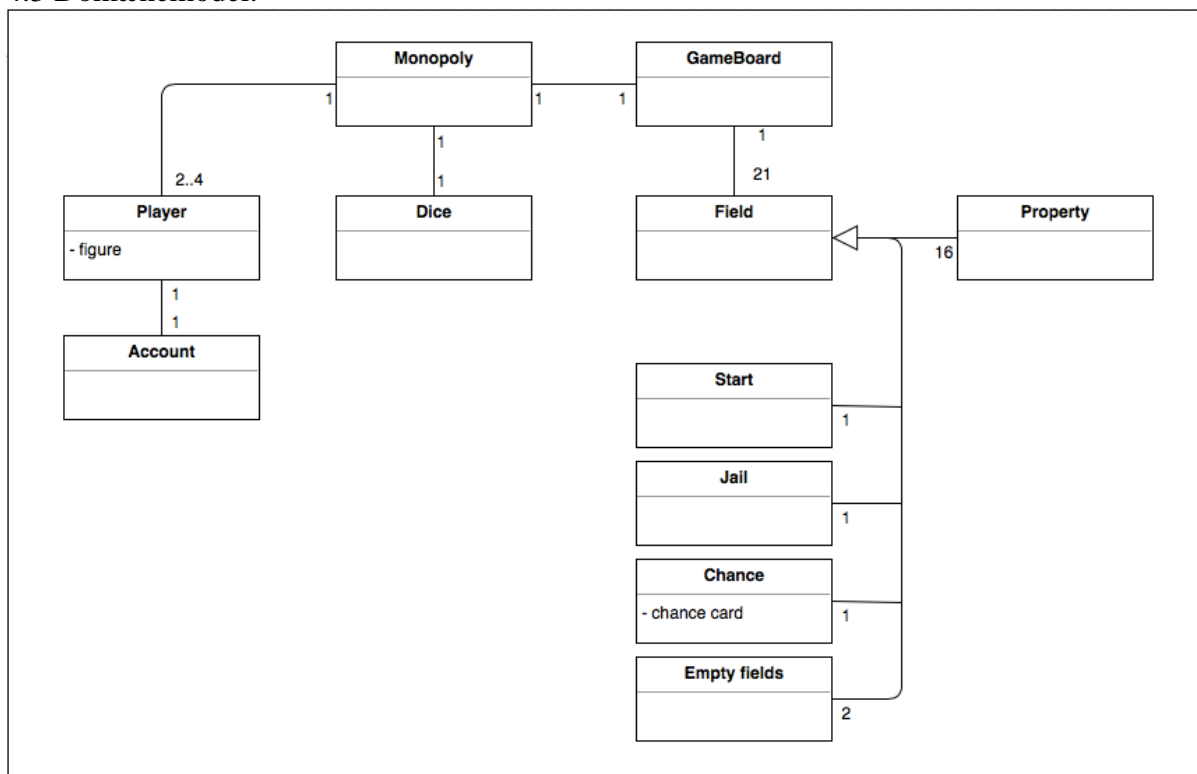
3. Hvis man lander på et chance-felt udløses et chancekort, som varierer i instrukser.

Specielle krav:

Du skal bruge en Windows computer fra DTU's databarer.

Forsinkelse skal være målbart.

4.3 Domænemodel:

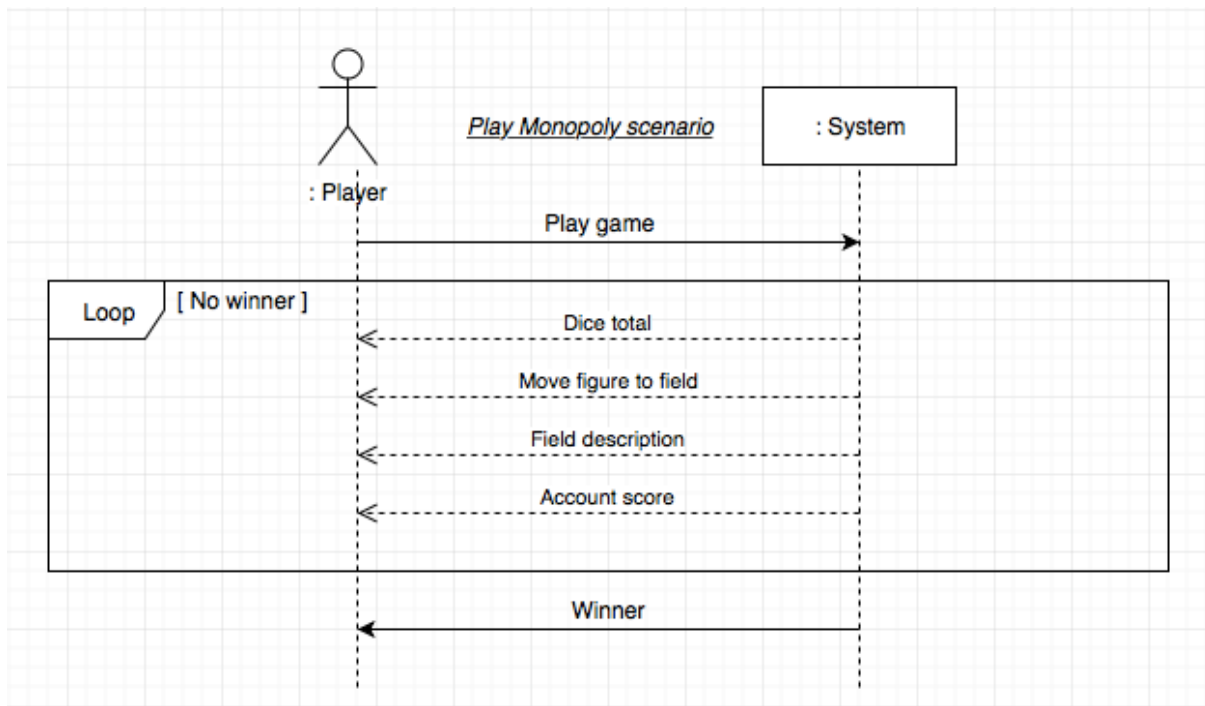


Domænemodellen giver en idé til, hvilke klasser der skal være i spillet, og hvordan de samarbejder med hinanden. Vi har tilføjet vigtige attributter til nogle af klasserne.

I domænemodellen ses det hvordan klasserne samarbejder med hinanden, blandt andet ved de to nedarvninger vi har. Selve spillepladen har nogle forskellige felter som deler visse metoder og attributter. Disse samles i en superklasse, som deler nogle metoder og attributter med sine subklasser.

4.4 Systemsekvensdiagram

I systemsekvensdiagrammet ses samspillet mellem aktøren (Player) og systemet. Det ses i diagrammet, hvad spilleren kan få systemet til at gøre, og derefter hvad systemet returnerer til aktøren (Player).



I systemsekvensdiagrammet fremgår det, at aktøren (Player) sætter spillet i gang ved at få systemet til at kaste terningerne. Det er altså aktøren (Player) der initialiserer spillet. Derefter viser systemet hvor meget man har slået, flytter spillerens figur så mange felter som man slog med terningen, hvad man skal når man lander på dette felt og viser hvad ens saldo er, altså hvor mange penge man har. Dette kører i et loop for alle spillerne, indtil en af spillerne ikke har flere penge. Når en spiller går fallit, giver systemet en vinder tilbage til aktøren eller i dette tilfælde aktørerne, da der kan være op til 4 af den samme aktør.

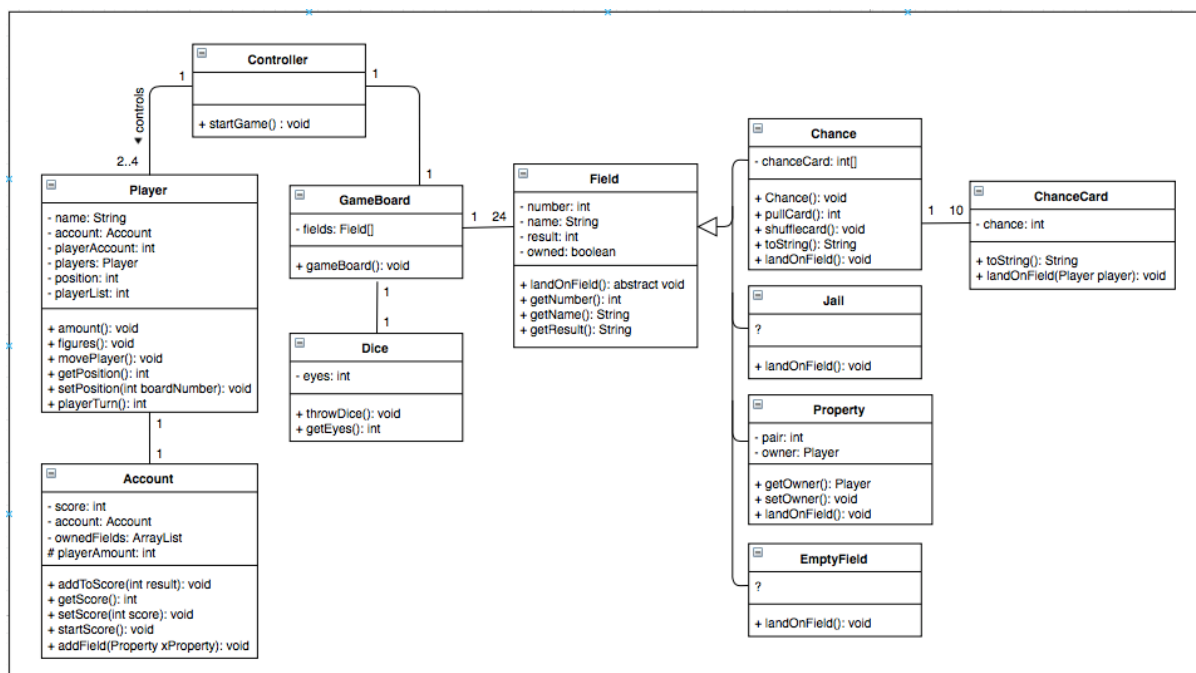
5. Design

Systemet designes ud fra hvilke oplysninger man har fået fra kunden i analysen. I design præciseres hvordan systemet skal se ud og hvad det skal kunne. Der kigges på hvad der er vigtigt at have med og hvad der er mindre vigtigt, og fortolker på hvad vi i samarbejde med kunden er kommet frem til i analysen.

5.1 Designklassediagram

Designklassediagrammet hjælper med at give en idé til, hvilke klasser programmet skal have, samt deres attributter og metoder. Diagrammet giver et bedre overblik til, hvad der skal laves, og en idé til, hvilke relationer der skal være.

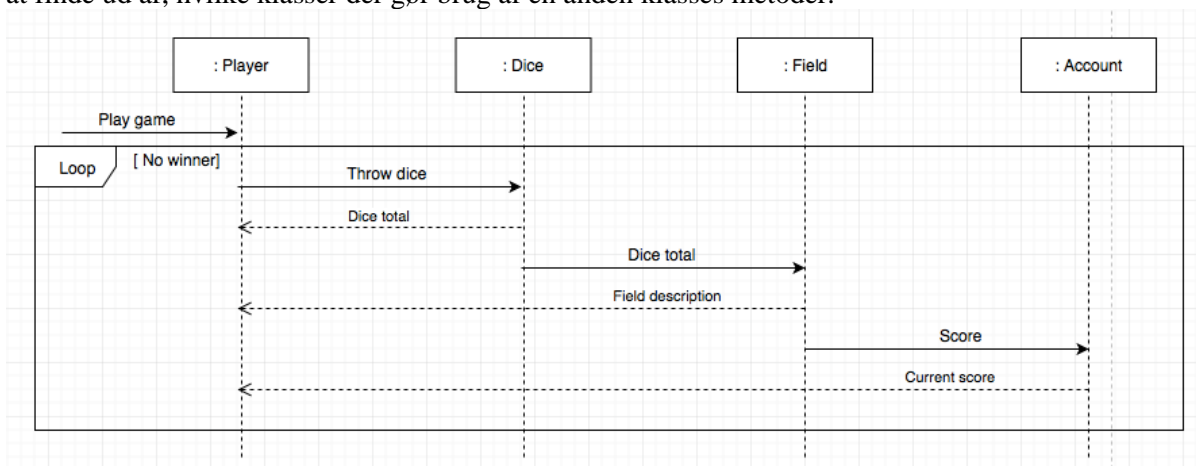
Ud fra diagrammet kan man forsøge at gøre koblingerne så minimale som muligt.



Designklassediagrammet giver et overblik over hvilke attributter og metoder, vi kunne tænke os til hver af klasserne. Det giver også et overblik over hvordan vi gerne vil have klasserne til at associere til hinanden. Da vi ikke blev helt færdige med programmet er dette stadig vores ide til hvordan vi gerne vil have det færdige produkts klasser til at interagere med hinanden. Dog vil dette diagram ændre sig hvis nogle af klassernes associeringer ændre sig under implementeringen.

5.2 Sekvensdiagram

Her ses hvilket samspil der er mellem nogle udvalgte klasser i programmet. Sekvensdiagrammet hjælper til at finde ud af, hvilke klasser der gør brug af en anden classes metoder.



En aktør sætter spillet i gang, hvorefter Player-klassen kaster terningerne. Dice-klassen både viser antal øjne, og giver dette videre til Field-klassen for at kunne rykke figurerne. Field-klassen viser hvad der sker på det gældende felt (field description) og giver en score som bruges i Account-klassen. Denne score bliver enten trukket fra eller lagt til den nuværende saldo, som så bliver vist til spilleren. Dette fortsætter i et loop for alle spillere indtil en vinder er fundet.

5.3 GRASP-mønstre

Vores projekt gør brug af nogle GRASP principper, som creator, information expert og controller. Vi har en controller klasse som sørger for at uddele opgaver fra brugergrænsefladen til de andre klasser.

Vi har en creator, Player klassen, som er ansvarlig for at oprette objekter fra bl.a. Account klassen.

Både Dice klassen og Account klassen er information experts til Field klassen, hvilket skaber en høj binding for Field klassen. Derudover er Account klassen information expert til Player klassen, hvilket igen skaber høj binding.

Vores projekt bærer præg af høj binding og høj kobling, dog vil vi helst have at koblingen skal være så lav som muligt. Da der er høj kobling, gør dette at vores klasser afhænger for meget af hinanden, og der er derfor højere risiko for fejlkommunikation mellem klasserne.

6. Implementering

Vi startede ud med at bruge meget lang tid på at importere GUI, uden at dette lykkedes. Dette resulterede i, at vi kom for sent i gang med programmering af programmet. Derudover lavede vi klasserne med en alt for høj kobling, hvilket resulterede i, at vi ikke kunne teste om de enkelte klasse virkede, før den næste classes metoder var færdige.

Hensigten er at have en super-klasse - Field - som nedarves af alle egentlige felter. Ved et polymorfisk kald bør metoden landOnField kaldes i controlleren, og afhængig af feltet spilleren står på, kaldes den respektive overskrevne metode. Fejlen ligger i, at vi endnu mangler at koble klassen GameBoard til movePlayer-metoden. Der registreres dermed ikke hvilket felt spilleren er på og dermed kaldes den rigtige landOnField metode ikke som det ser ud nu. Eksemplet under viser landOnField metoden for Property-klassen:

```
public void landOnField() {  
    / hvis feltet er ledigt kaldes addfield metoden, der tildeler spilleren  
    if(owner == null) {  
        player.getAccount().addField( xProperty: this);  
        player.getAccount().addToScore(- getResult());  
    } else {  
        System.out.println("Ejendommen tilhører" + owner);  
        //trækker resultat fra nuværende spillers konto  
        account.setScore(account.getScore() - (getResult()));  
        // lægger resultat til ejers konto.  
    }  
}
```

Der er lavet en arrayList over hver spillers ejede felter. Det er desværre ikke til at teste på endnu grundet ovenstående. Hensigten er dog, at der i landOnField metoden i Property-klassen, skal kunne kendes forskel på om feltet er ejet eller ej. Står det tomt, sættes feltet i ejerens array over ejede fields, og prisens trækkes fra account. Hermed står dette felt ikke længere som Null, så næste gang en spiller lander på feltet, står det ikke længere som ledigt. Nu kan ejeren af feltet kaldes, så der kan betales.

6.1 Branching strategi

Vi har som udgangspunkt arbejdet med fire branches - Master, Development, GUI og Test. Master er kun blevet brugt til at merge ind i, således at Master-branchen kun består af kode, som er testet og kører. Development er brugt til at udvikle og forbedre koden, mens Test er brugt til at teste og skrive tests til programmet. GUI er brugt til at eksperimentere med at få GUIen til at virke.



6.3 Kode

Arv

Arv er, hvis man vha. en superklasse samler attributter og metoder, som flere subklasser har brug for, sådan at attributterne og metoderne kan nedarves fra superklassen uden at de skal indeholdes i hver enkelt subklasse.

Abstract

En abstrakt klasse er en superklasse som indeholder attributter der er fælles for de subklasser der nedarver fra den. Der bliver aldrig oprettet et objekt af klassen, da den udelukkende bliver brugt til nedarvning.

Override og overload

Fortæl hvad det hedder hvis alle fieldklasserne har en landOnField metode der gør noget forskelligt:

Override (overskrive)





I vores projekt har vi en field-klasse (superklasse) som nedarver til andre field-klasser (sub-klasser), dog med andet navn. Når disse klasser, både superklassen og subklasserne, alle har en landOnField metode, kaldes dette en overskrivning eller override.

Overload

Man kan også have kun én klasse som har flere metoder med samme navn, dog med forskellig struktur. Altså at disse metoder hedder det samme, men de gør ikke nødvendigvis det samme, og modtager forskellige parametre.

7. Test

Tests blev startet sent i forløbet. I bagklogskabens lys kunne vi have benyttet os mere af test driven implementering, men da implementeringen aldrig faldt helt på plads, blev der ikke meget tid til tests. Vi har dog en række test cases som kunne være oplagt. De er dog ikke gennemført grundet presset tid og følgende fejl, som vi ikke kan komme udenom:

-  Information: java: Errors occurred while compiling module '33_del3'
-  Information: javac 10.0.2 was used to compile java sources
-  Information: 30-11-2018 12:31 - Compilation completed with 1 error and 0 warnings in 946 ms
-  Error: java: release version 5 not supported

Test case ID	TC01
Summary	Test af playerTurn
Requirements	Viser at det er næste spillers tur
Preconditions	Der er 4 spillere, der spiller spillet
Postconditions	Næste spillers tur
Test procedure	Der oprettes en Junit test case Der oprettes en stubklasse, hvor antal spillere og hvilken spiller der lige har haft sin tur bestemmes på forhånd Næste spillers tur bestemmes Resultatet sammenlignes med det faktisk resultat
Test data	Test1 - 4 spillere, spiller 3 har lige haft tur Test2 - 4 spillere, spiller 2 har lige haft tur Test3 - 4 spillere, spiller 1 har lige haft tur Test4 - 4 spillere, spiller 4 har lige haft tur
Expected result	Test1 - spiller 4's tur Test2 - spiller 3's tur Test3 - spiller 2's tur Test4 - spiller 1's tur
Actual result	

Status	
Tested by	
Date	30.11.2018
Test environment	IntelliJ
Appendix	

8. Brugervejledning

1. Spillet kræver 2-4 spillere
2. Start spil
3. Vælg hvor mange spillere I er - så får I hver tildelt en figur og et pengebeløb
4. Spillerne skiftes til at slå med en terning - terningens udfald afgør, hvor man rykker hen på spillepladen
5. Lander man på en ejendom, køber man den automatisk, hvis den ikke er ejet
 - a. Er ejendommen allerede ejet, skal man betale husleje til den spiller, der ejer ejendommen
6. Lander man på et chancekort, trækker man det øverste i bunken og følger dets instruktioner
7. Når en spiller løber tør for penge, skal de andre tælle deres
8. Den spiller med flest penge vinder

8.1 Konfiguration

Importeret fra git til IntelliJ samt start af spil:

- Følg linket https://github.com/Studiegruppe33/33_del3
- Vælg 'clone or download'-knappen og kopier linket
- Åbn IntelliJ og vælg checkout from version control
- Indsæt linket og tryk ok.
- Find og åbn Main-klassen i src-mappen gennem stisystemet i venstre side af IntelliJ
- Højreklik i vinduet og vælg Run

Platform:

Operativsystem: Microsoft Windows Version 1803, build 17134.407 & MacOS Mojave 10.14.1

Java: 10.0.2

IntelliJ: IntelliJ IDEA 2018.2.2 (Ultimate Edition)

Build #IU-182.4129.33, built on August 21, 2018

matadorgui.jar: 3.1.5

10. Konklusion

Efter at have arbejdet med et projekt med væsentligt større krav end tidligere, har vi erfaret, at nedprioritering af mindre vigtige krav kan være en nødvendighed for at overholde den givne deadline. Kravene er prioriteret efter MoSCoW-princippet for at afgøre, hvilke features der er mest nødvendige at have med i spillet.

Endvidere har vi blandt andet udarbejdet et designklassediagram for at danne et overblik over, hvilke klasser programmet skal bestå af, samt hvilke relationer der skal være mellem de forskellige klasser. I sammenhæng med designklassediagrammet er hensigten, at der understøttes GRASP-mønstre, for eksempel i form af lav kobling og høj binding.

Det har dog forvoldt os problemer i selve implementeringen, at vi står tilbage med for høj kobling i visse klasser, hvilket ikke hjælper på overskueligheden. Dette er et resultat af en spillerklasse med for lav binding. For mange klasser er afhængige af denne klasse.

Litteraturliste

Matador Junior Regler.pdf

Bøger

Larman, Craig: *“Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development”*, 3rd edition, 2005, Pearson Education, ISBN: 0-13-148906-2

Lewis, John, William Loftus & Addison Wesley: *“Java Software Solutions”*, 9th edition, 2018, Pearson Education, ISBN: 9781292221724

Hjemmesider

Hjælp til Player-klassen:

<https://github.com/Niemeijeren/Matador/blob/master/Matador/src/Player.java>

Opgaver

Studiegruppe 33, DTU 2018: CDIO del 1

Studiegruppe 33, DTU 2018: CDIO del 2

Bilag

Bilag A

Spillets formål:

- Sus rundt på brættet, køb alle de ejendomme, du lander på, saml kontanter, og tag chancekort.
- Når en spiller løber tør for penge, skal de andre tælle deres.
- Spilleren med flest penge vinder.

Spillets regelsæt:

1. Den yngste spiller starter, og spillet fortsætter mod venstre.
2. Kast terningen, og systemet rykker brikken.
 - Ryk altid frem, aldrig tilbage. (Hvis man skal til et felt der ligger bag sig, rykkes der frem, forbi start, til man er på feltet)
 - Hver gang START passeres eller landes på modtager spilleren 2M.
3. Hvor landede du? Læs om de forskellige felter:
Hvis du lander på:
Ejendomme
 - Et ledigt felt
Hvis ingen ejer det skal du købe det.
 - Et ejet felt
Hvis en anden spiller ejer feltet, skal du betale husleje til den spiller. Huslejen er det beløb, der står på feltet.
Hvis du selv ejer feltet, sker der ikke noget.
 - Et par = dobbelt husleje

Hvis en spiller ejer begge ejendomme i samme farve, er huslejen det dobbelte af hvad der står på feltet.

Brættets felter

- **START**
Modtag 2M hver gang du passerer eller lander på START.
- **Chance**
Tag det øverste chancekort i bunken, læs og følg instruktionerne.
Læg kortet tilbage nederst i bunken.
- **Gå i fængsel**
Gå lige i fængsel. Man passerer ikke START og modtager ikke 2M. Ved din næste tur skal du betale 1M, og derefter kaster du terningen og ryk som normalt.
Du kan godt modtage husleje, selvom du er i fængslet.
- **På besøg**
Her er du bare på besøg i fængslet, du skal altså ikke betale noget ved din næste tur.
- **Gratis parkering**
Her sker der ikke noget, bare tag en pause.

4. Når terningen er kastet og du har fulgt instruktionerne til det felt du landede på, går turen videre til næste spiller (som sidder til venstre for dig).

5. For at vinde:

- a. Hvis du ikke har penge til at betale husleje, købe ejendom, eller betale afgiften fra et chancekort, er du gået fallit. Spillet er slut.
- b. De andre spillere tæller deres penge, den der har flest vinder.
- c. Hvis det bliver uafgjort, tæller du hvor meget dine ejendomme er værd og lægger det til dine penge.