

# CDIO3

## Gruppe 16

10. november 2017



Mathias  
Fager  
s175182



Milishia  
Moradi  
s175193



Nicki  
Christiansen  
s170208



Semi  
Seitovski  
s175181



Simon  
Pedersen  
s175195



Thyge  
S. Steffensen  
s175176

# DTU

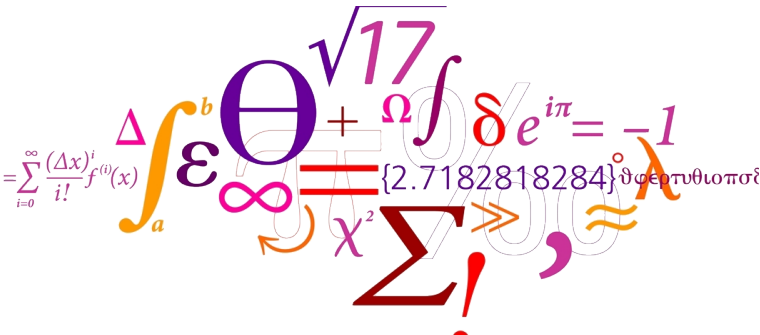


# Timeregnskab

Timeregnskabet kan ses via linket [her](#) eller i billag XX.

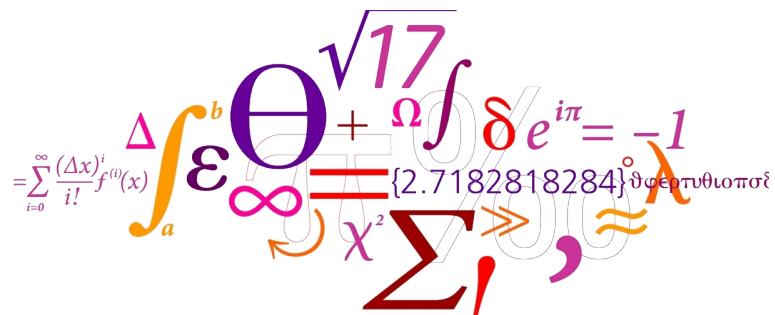
Deltager	Total timer per deltager
Mathias Fager	3
Milishia Moradi	3,5
Nicki Christiansen	10
Semi Seitovski	6
Simon Steen Pedersen	5,5
Thyge Steffensen	14,5
Total	42,5

Figur 1: Overblik over Timeregnskab den 21/11/2017



## Indhold

<b>1</b>	<b>Analyse</b>	<b>3</b>
1.1	Kravliste . . . . .	3
1.2	UseCases . . . . .	4
1.3	UseCase diagram . . . . .	7
1.4	GRASP . . . . .	7
<b>2</b>	<b>Design</b>	<b>9</b>
2.1	Klasse diagram . . . . .	9
2.2	Sekvensdiagram . . . . .	10
2.3	System sekvensdiagram . . . . .	10
2.4	Domænemodel . . . . .	10
<b>3</b>	<b>Dokumentation</b>	<b>11</b>
3.1	Forklar hvad arv er . . . . .	11
3.2	Forklar hvad abstract betyder . . . . .	11
3.3	Fortæl hvad det hedder hvis alle fieldklasserne har en landOn- Field metode der gør noget forskelligt . . . . .	11
3.4	Dokumentation for test med screenshots . . . . .	12
3.5	JUnit test . . . . .	12
3.6	Positiv negativ test . . . . .	12
3.7	Black- og Whitebox test . . . . .	12
3.8	Dokumentation for overholdt GRASP . . . . .	12
3.9	Vejledning til import af Git Repository til eclipse . . . . .	12





## 1.2 UseCases

UseCase Section: Start af spil	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	IOOuterActive
Stakeholder og interesser	IOOuterActive er interesseret i at spillerne skal kunne starte spillet.
Forudsætninger	Spillet er installeret på enheden.
Success garanti	Spillet starter og spillerne bliver sendt videre til opsætning.

UseCase Section: Opsætning af spil	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interesser	Spillerne er interesserede i at kunne vælge antal spillere og deres brikker.
Forudsætninger	Spillet er startet op og spillerne har nu mulighed for at vælge antal spillere og ønskede brikker.
Success garanti	Der er blevet valgt antallet af spillere og hver spiller har valgt sin brik, herefter er spillet klar til at blive spillet.

$$\begin{aligned}
 &= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \\
 &\int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1 \\
 &= \{2.7182818284\} \cdot \text{greek letters} \\
 &\chi^2 \sum \gg \approx \lambda
 \end{aligned}$$

Fully dressed UseCase:

UseCase Section: Spillerne slår med terningerne	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interesser	Spillerne er interesseret i at kunne trykke på en knap, og få et billede af en terning med tilfældige værdier.
Forudsætninger	Spillet er startet op og spillerne har valgt antallet spillere og deres ønskede brikker.
Success garanti	Der er blevet valgt antallet af spillere og hver spiller har valgt sit navn, herefter er spillet klar til at blive spillet
Hoved succes scenarie	Spillerne får vist terningens øjne og deres brik bliver rykket hen til det respektive felt.
Alternative udfald	Negative udfald: - IOOuterActive har opdateret spillet og derved opstår der en fejl når spillerne slår med terningen, der kan ende i at der ikke slået med terningen. - Systemet blokerer for en spillers tur - En spiller hopper fra/på og derved skal spillet startes om.
Specielle krav:	- Enheden som spillet kører på skal kunne køre Java. - Spillerne skal kunne interagere med GUI'en ved brug af mus eller touch. - Der skal være plads på enheden til at kunne hente spillet.
Hyppighed	Hver tur bliver der slået med en terning.

UseCase Section: En spiller køber et felt	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interesser	Spillerne er interesseret i at købe det aktuelle felt.
Forudsætninger	Spillet er i gang og en spiller har slået med terningerne
Success garanti	Spilleren køber og ejer nu feltet

$$= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \quad \Delta \int_a^b \epsilon \Theta + \Omega \int \delta e^{i\pi} = -1$$

$$\chi^2 \sum \gg \approx \lambda$$

$$\{2.7182818284\} \circ \sigma \epsilon \tau \theta \iota \omega \pi \sigma \zeta$$

UseCase Section: Spiller lander på et eget felt	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	Spillerne
Stakeholder og interesser	****
Forudsætninger	Spillet er i gang og en spiller lander på et felt, som er ejet af en anden spiller.
Success garanti	En spiller lander på et felt, der er ejet af en anden spiller og får derfor en negativ effekt på balancen.

UseCase Section: En spiller taber	Comment
Scope	Monopoly Junior spil af IOOuterActive
Level	User-goal
Primær Aktør	Terningerne
Stakeholder og interesser	Spillerne er interesseret i at deres balance ikke når 0 og derved taber.
Forudsætninger	Spillerne har slået med terningen.
Success garanti	En spillers balance når 0, taber de spillet og spillet afsluttes.

UseCase Section: Spillet afsluttes	Comment
Scope	Monopoly spil af IOOuterActive
Level	User-goal
Primær Aktør	IOOuterActive
Stakeholder og interesser	IOOuterActive er interesseret i at programmet viser en vinder og afsluttes.
Forudsætninger	En spiller har nået en balance på 0 eller derunder.
Success garanti	Spillet viser en vinder og kan derefter afsluttes.

$$= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \quad \Delta \int_a^b \Theta + \Omega \int \delta e^{i\pi} = -1$$

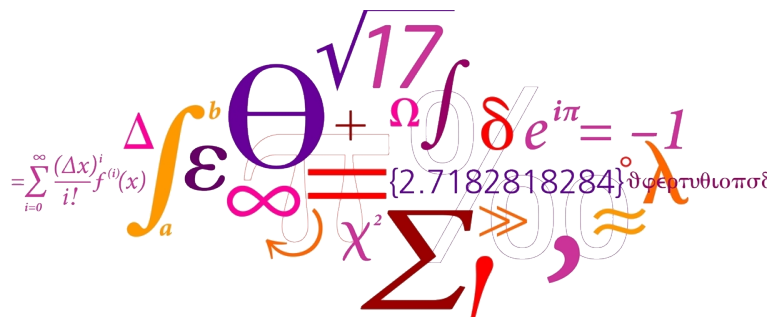
$$\infty = \{2.7182818284\} \quad \chi^2 \quad \sum \quad \gg \quad \approx \quad \lambda$$

$$\sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \int_a^b \varepsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1$$



3. Low coupling
4. Controller
5. High cohesion
6. Indirection
7. Polymorphism
8. Protected variations
9. Pure fabrication

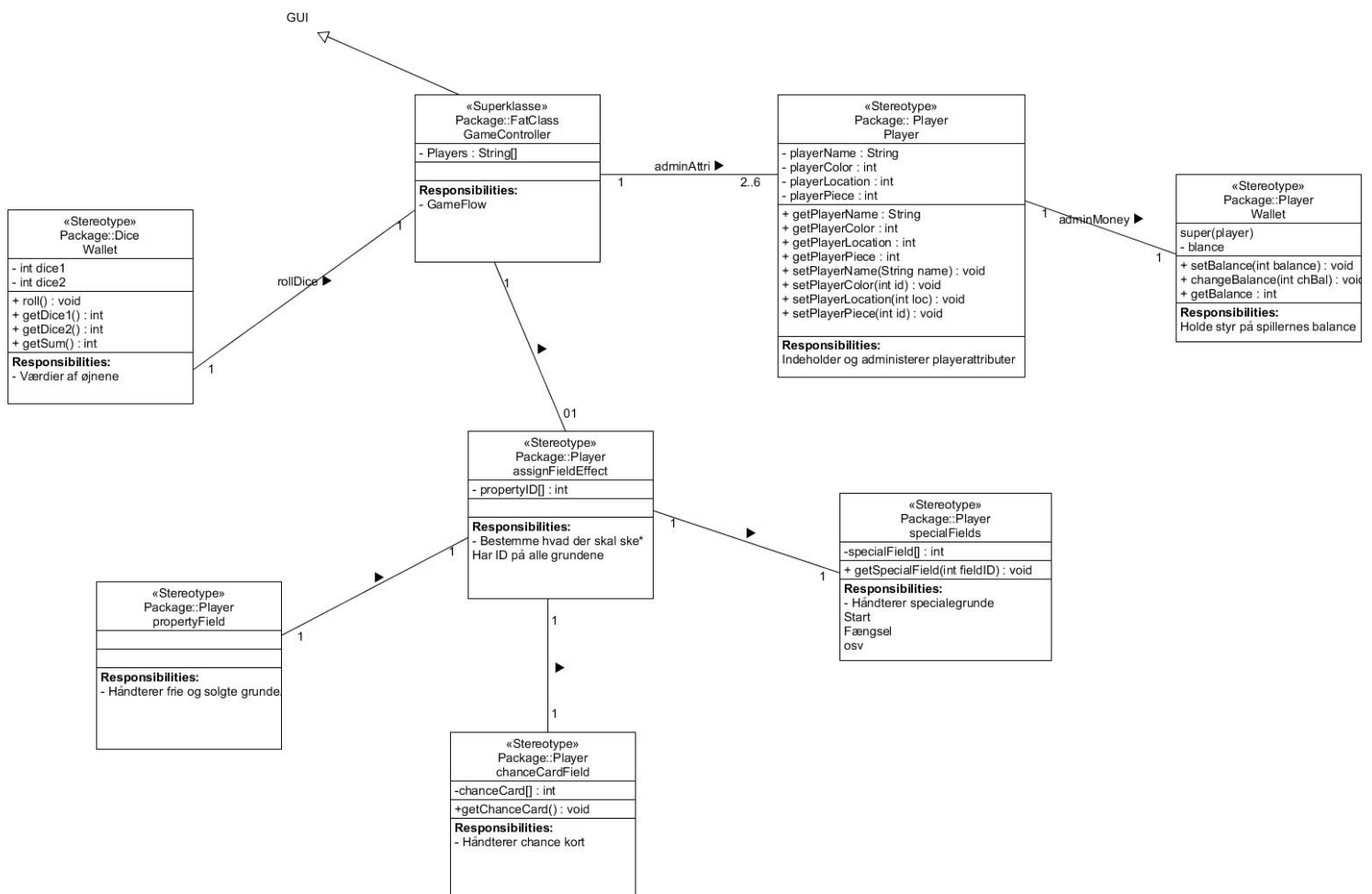
(Der skrives mere når vi er nået længere i projektet)



## 2 Design

Herunder ses en række 'design steps', som skal hjælpe os med at lave Monopoly Junior spillet.

### 2.1 Klasse diagram



Figur 3: Klasse diagram tegnet i UMLet

Klasse diagrammet bygger på vores umiddelbare overvejelser, såvel som vores use case's. Dette er for at illustrere sammenspillet mellem vores klasser og deres associationer. Dog skal det siges, at dette er en skitse og den aktuelle programmering kan variere heraf.

$$= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \quad \int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1$$

$$\chi^2 \sum \gg \approx \lambda$$

## 2.2 Sekvensdiagram

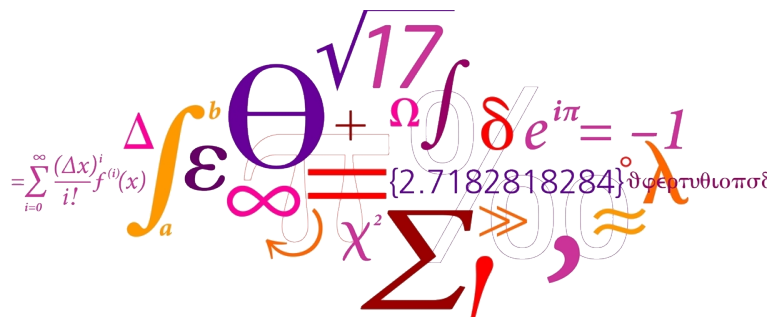
Vi har her lavet et sekvensdiagram, der skal skabe et overblik over hvordan aktøren, her spilleren, kommunikerer med spillet.

## 2.3 System sekvensdiagram

Vi har her lavet et systemsekvensdiagram, for at forhøje gennemsigtigheden ved bruge af 'chanceCard' klassen.

## 2.4 Domænemodel

Ved hjælp af domænemodellen vil vi trække paralleller mellem den virkelige verden og programmeringen. Domænemodellen er en visuel repræsentation af konceptklasser og 'objekter fra den virkelige verden'. Ved hjælp af denne kan vi også oplyse kunden om, hvad vi vil lave.



## 3 Dokumentation

### 3.1 Forklar hvad arv er

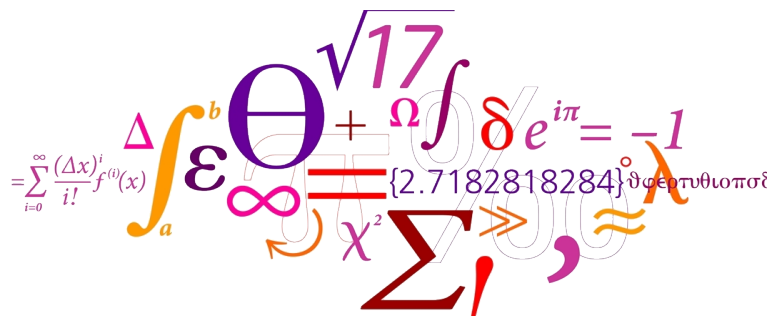
Et af de vigtigste elementer i objektorienteret programmering er nedarvning, men hvad betyder nedarvning? At arve betyder i bund og grund at et "forældre-objekt videregiver ressourcer til efterkommere. Denne naturligt forekommende handling anvendes på ens hvis i programmerings-verdenen. Her fungerer arv konkret ved at en eller flere egenskaber overføres fra en generation til en anden generation, sagt på en anden måde, en klasse får overført attributter og metoder fra en superklasse. Den klasse der nedarves fra kaldes for en Superklasse (parent), hvor klassen der nedarver eller får egenskaber fra en Superklasse, kaldes for en Subklasse (child). (Kressner, Marts 2014). Ved benyttelse af denne proces vil oplysningerne om de forskellige klasser ende op i en rækkefølge struktureret hierarkisk. På figuren ses forskellige eksempler på typer af nedarvning, der findes, men dog skal man være opmærksom på, at Java ikke understøtter flere nedarvninger. I Java kan en klasse kun have én Superklasse, dvs. at hver klasse kun kan nedarve fra én klasse. (Kressner, Marts 2014).

### 3.2 Forklar hvad abstract betyder

Den abstrakte klasse kører ingen metoder, og indeholder kun attributter. Klassen bruges som en slags "over"-klasse, hvor andre klasser nedarver attributterne. Det anvendes i en situation, hvor man f.eks har to klasser, der har mange af de samme attributter. Så vil alle de attributter, som de har tilfælles stå i den abstrakte klasse. Hvis man f.eks har klasserne lærer og elev, kunne den abstrakte klasse til disse, hedde "personer", da både lærer og elever har en fødselsdato, køn, navn osv.

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

Hvis alle fieldklasserne gør brug af den samme landOnField metode, er det fordi, denne metode er en super metode. En super metode er nedarvet til de forskellige subklasser fra super klassen, dette gør at alle klasserne kan gøre brug af samme metode. Eksempeltvis, hvis vi har en masse dyr som klasser, kat, hund, kanin etc. så kan de alle nedarve super metoden eat() fra superklassen som hedder SurvivalRequirements, eftersom disse er metoder alle dyrene får brug for, så vil det give mening at lave det til en superklasse


$$\begin{aligned} &= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \\ &\int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1 \\ &= \{2.7182818284\} \Delta \Theta \epsilon \tau \theta \iota \omega \pi \sigma \zeta \end{aligned}$$

med supermetoder, så der holdes lav kobling og høj kohæsion, samt undgås kopiering af kode og høj mulighed for genbrug.

### 3.4 Dokumentation for test med screenshots

### 3.5 JUnit test

JUnit test er en autonomiseret testmetode. Her skriver/koder man selv en test, som tester java kode. Oftest opbygger man JUnit test ud fra Java klasser. Der er mange måder, hvorpå man kan bruge JUnit testen. Man kan både skrive testen inden, man kan skrive den efter, man kan lave den på baggrund af indsigt i koden eller uden nogen form til kendskab af programkoden. Det to sidst nævnte kaldes Black- og Whitebox test. Her er et eksempel på et stykke udført JUnit test fra vores spil: Det ser på Figur 3.5 at koden ikke bestod testen, hvor vi også får vist den rette, selv skrevet, fejl besked.

### 3.6 Positiv negativ test

Denne test bruges for at teste om systemet kan håndtere 'ugyldige' input, dette kan både være negative som positive værdier, såvel som bogstaver og andre tegn. Denne test forbindes oftest med en eller flere ækvivalensklasse test.

### 3.7 Black- og Whitebox test

**Blackbox test:** Her har man intet kendskab til koden, ud over hvad denne skal kunne gøre. Det er derfor lige til højrebænet, at teste det faktiske output mod det forventede output og at teste ækvivalensklasserne.

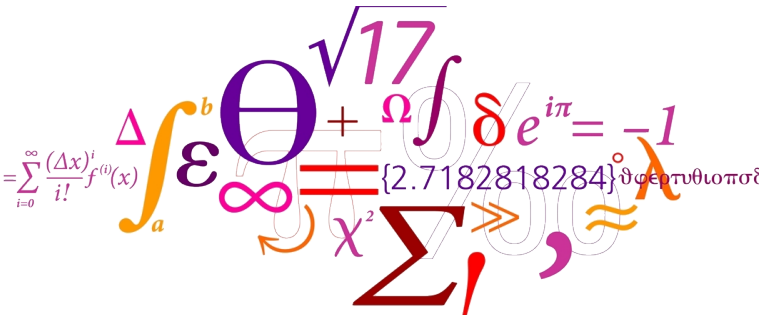
**Whitebox test:** Her opstiller man tests på baggrund af koden og dens opbygning. Man vil med disse tests teste hele koden, altså alle instruktioner, forgreninger og stier, som koden kan blive udført i. Dette er ikke altid muligt med *BlackBox testen*, da man her, ikke har et indblik i koden.

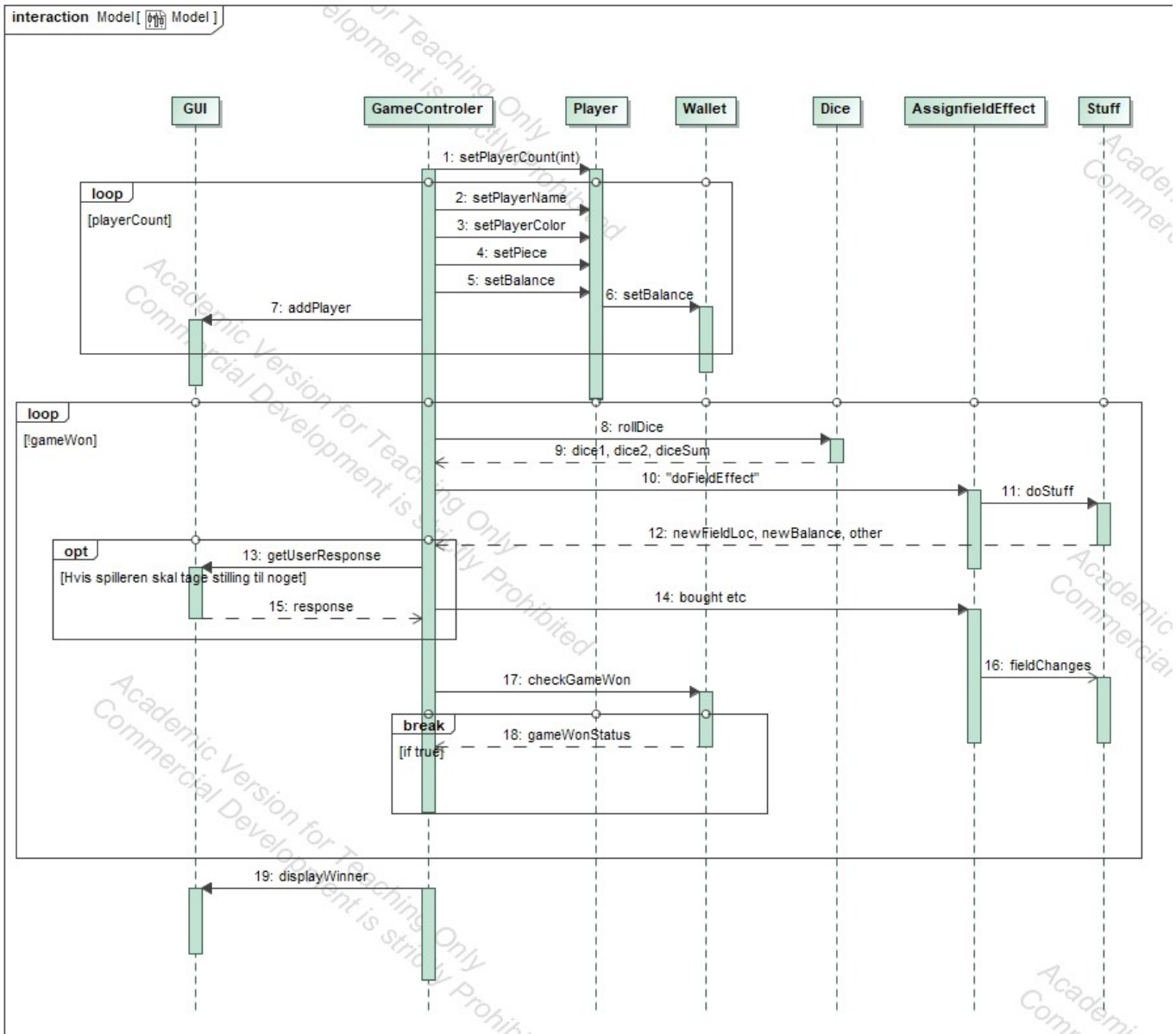
### 3.8 Dokumentation for overholdt GRASP

### 3.9 Vejledning til import af Git Repository til eclipse

Man skal i første omgang have et URL af det pågældende repository, som du kopierer. Derefter åbner du eclipse og åbner menuen Window  $\Rightarrow$  View  $\Rightarrow$  søg på "git"  $\Rightarrow$  git repositories. Derefter har du vinduet "Git Repositories", hvor der er en knap; "Clone a git repository and add to this view". Ikonet

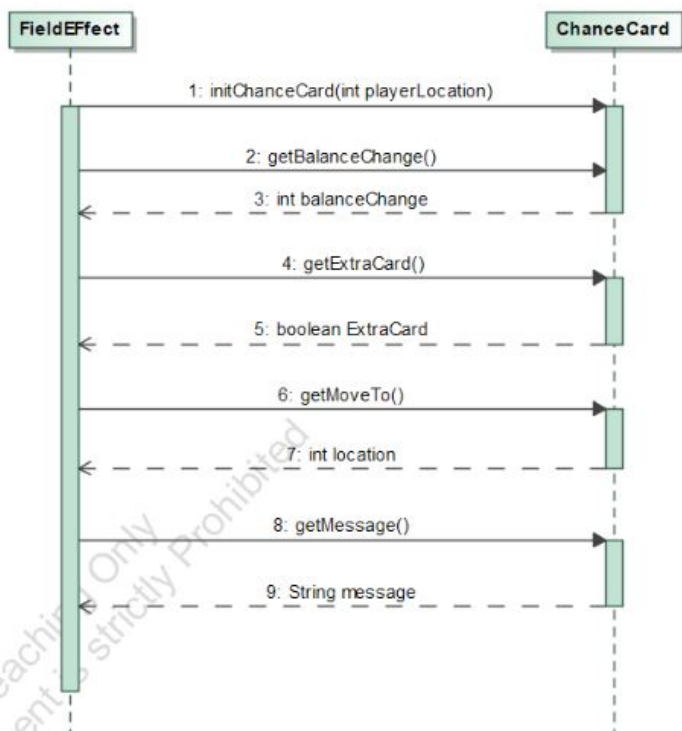
er mappen i midten med en blå pil på, her trykker du. Hvis du huskede at kopiere URL’et trykker du nu ”Next”, hvorefter at alle branches bliver loaded. Når den har loaded trykker du next igen. Nu kommer du til det sidste vindue, hvor du trykker ”Finish, og nu har du importeret(clonet) git repositoriet.





Figur 4: Sekvensdiagram tegnet i MagicDraw

$$\begin{aligned}
 &= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \\
 &\int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1 \\
 &= \{2.7182818284\} \lambda \\
 &\chi^2 \sum_i \gg \approx
 \end{aligned}$$



**initChanceCard**  
 - initialisere et chancekort: Vælger et kort nummer (1ud af 18) og skal benytte den hele vejen igennem denne session.

**balanceChange**  
 - Både en positiv og negativ værdi, der skal lægges til balancen

**extraCard**  
 - Hvis spilleren skal trække et chance kort til, er denne positiv

**moveTo**  
 - Giver en int som skal lægges til spillerens lokation  
 - BEMÆRK: Denne tager ikke højde for antal felter på pladen, dette skal gøres i AssignField

**message**  
 - indeholder besked, som skal vises til brugeren.

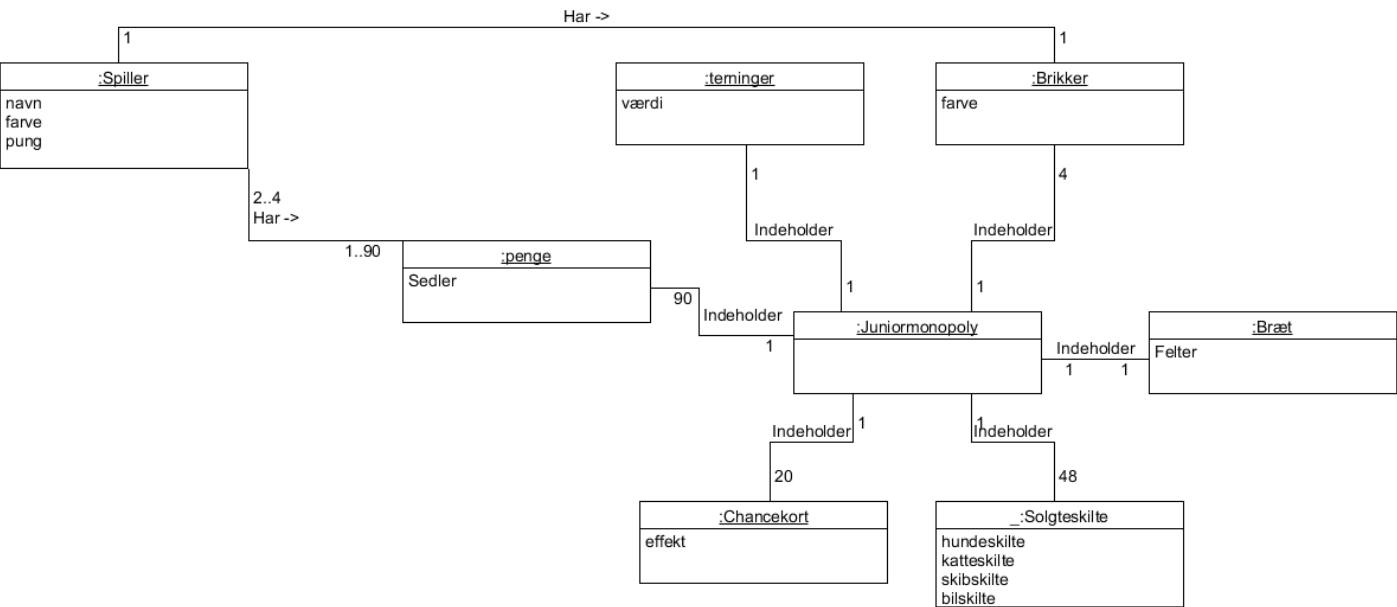
**Init**  
 - set message  
 - set moveTo  
 - set extraCard  
 - set balance

Figur 5: Systemsekvensdiagram tegnet i MagicDraw

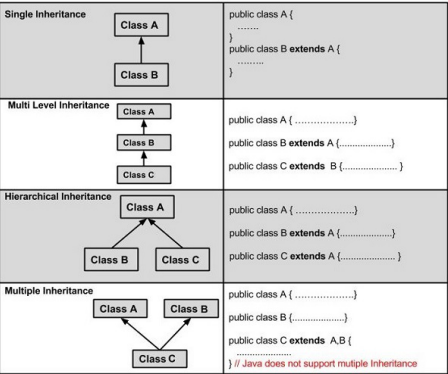
$$= \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x) \quad \Delta \int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1$$

$$\infty = \{2.7182818284\} \quad \chi^2 \sum \gg \approx \lambda$$





Figur 6: Domænemodel tegnet i UMLet



Figur 7: eksempel på typer af arv

Complex mathematical symbols and formulas including:

- $\sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$
- $\int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = -1$
- $\{2.7182818284\}$
- $\chi^2$
- $\sum$
- $\approx$
- $\gg$
- $\infty$
- $\Delta$
- $\Omega$
- $\epsilon$
- $\Theta$
- $\delta$
- $e^{i\pi}$
- $\int$
- $\int_a^b$
- $\sum_{i=0}^{\infty}$
- $\frac{(\Delta x)^i}{i!}$
- $f^{(i)}(x)$
- $\chi^2$
- $\approx$
- $\gg$
- $\sum$

fig/JUnitTestDice.jpg

Figur 8: JUnit test udført i Eclipse den 23/11-2017

A complex, colorful collage of mathematical symbols and formulas, including integrals, summations, and various Greek letters, arranged in a chaotic, overlapping manner.