

CDIO3 opgave efterår 2016

02312-14 Indledende Programmering, 02313

Udviklingsmetoder til IT-Systems and 02315 Versionsstyring og testmetoder

Projekt navn: CDIO3

Gruppe #20

Afleveringsdag: Fredag 25/11-2016 23:59

Opgaven afleveres på campusnet.

Rapporten indeholder 32 sider inklusiv forsiden.



s136509, Jakobsen, Thomas
Software technology



s153374, Rasmussen, Emily Skovgaard
Software technology



s165153, Christiansen, Simon
IT & Economy

Tidsregnskab

CDIO pt. 3						
Ver. 2016-04-11						
Deltager	Design	Impl.	Test	Doc.	Andet	Total
Emily Rasmussen	3	3	2	2		10
Simon Christiansen	2	6	1	1		10
Thomas Jakobsen	2	3	3	2		10
Sum	0	0	0	0	0	30

Indholdsfortegnelse

Tidsregnskab	1
Introduktion	3
Opgavebeskrivelse	4
Risikovurdering	4
Systemkrav & konfiguration	5
Kravspecifikation FURPS+	6
Dokumentation	8
Use case model	9
Klassediagram	10
Sekvensdiagram	11
Design sekvensdiagram	12
Domænemodel	14
BCE model	15
Konklusion	16
<i>Arbejdsgang</i>	<i>16</i>
<i>Produkt</i>	<i>16</i>
Test	17
<i>JUnit test</i>	<i>17</i>
Litteraturliste	18
Referencer	18
Ordforklaring	18
Bilag	19
Bilag 1 - Feltliste	19
Bilag 2 - Use case <i>Fully dressed</i>	20
Bilag 3 - Use cases	22
Bilag 4 - Tidsplan	27
Bilag 5 - JUnit Terning	28
JUnit Hold Score	29
Bilag 6 - JUnit landPaaFelt Helle	30
Bilag 7 - JUnit landPaaFelt Flaade	31
Bilag 8 - JUnit landPaaFelt Arbejdslejr	31
Bilag 9 - JUnit landPaaFelt Grund	32
Bilag 10 - JUnit landPaaFelt Skat	32

Introduktion

Vi skal i dette projekt lave et spil, bestående af 2-6 spillere. Spillerne skal på skift slå med en terning, og rykker rundt på en spilleplade, afhængigt af øjnenes værdi. Der findes 5 forskellige felt-typer, som kan påvirke spillerens pengebeholdning. Vi har tidligere leveret programmer til kunden, som ønsker vi genbruger noget af det tidligere kode.

Vi har valgt at kalde spillet bankerot, da det gælder om at være den sidste spiller tilbage, der stadig har penge på kontoen.

Projektet er en videreudvikling af tidligere CDIO-projekter vi har arbejdet med på 1. semester med retningslinjer som henholdsvis softwareteknologi og it-økonomi. Vores gruppe består af 3 medlemmer og vi har haft en periode på 3 uger til lave projektet. Projektet skal desuden bruges til at give grundlag for feedback fra undervisere på DTU Lyngby. Målene for opgaven er at vise forståelse for test-script, unified process, GRASP og objektorienteret kode. Vores besvarelse af projektet og alle målene stillet under dette projekt bliver besvaret i konklusionen på side 16.

Opgavebeskrivelse

IOOuterActive har leveret et ønske om et nyt spil på baggrund af tidligere afleverede spil hos kunden, spillet er desuden en viderebygning for tidligere produkter. Kundens vision blev leveret som citeret nedenfor.

“Nu har vi terninger og spillere på plads, men felterne mangler stadig en del arbejde. I dette tredje spil ønsker vi derfor at forrige del bliver udbygget med forskellige typer af felter, samt en decideret spilleplade.

Spillerne skal altså kunne lande på et felt og så fortsætte derfra på næste slag. Man går i ring på brættet.

Der skal nu være 2-6 spillere.

Man starter med 30.000.

Spillet slutter når alle, på nær én spiller, er bankerot.” - ref 1.0

Derudover har vores projektleder (hjælpe lærerne) lavet et udsnit af et klassediagram han ønsker vi skal følge, og sat nogle krav i forhold til vores program og ikke mindst dokumentationen. Der ønskes desuden også en brugervejledning over spillet.

Risikovurdering

Som del af vores projektstyring har vi skulle vurdere, hvorvidt dette projekt kunne gennemføres og hvilke risiko vi forventede at møde undervejs.

Vi har lavet en risikovurdering før projektet blev påbegyndt. Vi har givet hver risiko en vurdering i form af rød, gul og grøn, hvor rød har en stor indflydelse på projektet, og kan potentielt vælte det hele og hvor grøn er en risiko vi sagtens kan arbejde udenom eller på anden vis har lille indflydelse på projektet.

Derudover har vi lavet en karakterskala for hvor sandsynligt hver hændelse er, karakterskalaen går fra 1-3 hvor 1 er meget sandsynligt og 3 er usandsynligt.

Underbemanding, mangler tid – Rød 2

Undervurdering af projektet størrelse - Gul 2

Komplikationer ved videre byg af gammel kode – Gul 3

Misforståelse af kundens vision – Grøn 3

Ingen tidligere erfaringer med GUI – Grøn 3

Risiko for yderlige underbemanding – Grøn 3

Systemkrav & konfiguration

Minimumskrav:

Processor: Pentium

Hukommelse: 128 MB

Diskplads: 30 KB

Styresystem: Windows XP eller nyere

Yderligere bemærkninger:

Java installeret

Compiler program, f.eks Eclipse

Koden kan hentes fra GitHub på følgende link: https://github.com/emilysr/CDIO3_4.git

Du kan hente koden direkte fra GitHub og importere til eclipse vha. ovenstående link som følgende.

- Åben eclipse.
- Klik på "Filer" og vælg "Import"
- Fold mappen "Git" ud og klik på "Projects from Git"
- Klik på "next"
- Vælg her "Clone URI" og klik "next" igen
- Kopier det ovenstående link og indsæt på linjen "URI"
- Klik "next" og du kommer til "Branch Selection"
- Vi lader alle felterne være hakket af og klikker "next"
- Du skal nu vælge hvor du vil gemme filerne på dit drev og klikker "next"
- Du vælger nu punktet "Import as general project" og klikker "next"
- Du kan nu navngive dit projekt.

Afslut med "finish"

Kravspecifikation FURPS+

Kravspecifikationen er udarbejdet ud fra tidligere opgaver hos kunden, og deres nye instrukser. Kravene er ikke modstridende og er alle testbare og inkluderet i koden. Opstillingen af kravene er baseret på FURPS+.

Funktionelle:

RQ101 - Spillet skal bestå af 2-6 spillere.

RQ102 - Systemet skal holde en separat pengebeholdning for hver spiller.

RQ102.1 - Hver spiller skal starte med en pengebeholdning på 30.000.

RQ102.2 - Hvis en spiller har en pengebeholdning på mindre end 0, er personen bankerot og ude af spillet.

RQ103 - Spillet skal slutte når alle på nær én er gået bankerot.

RQ104 - Spillet skal bestå af en decideret spilleplade.

RQ104.1 - Spillet skal bestå af 21 forudbestemte felter og et "Startfelt". -

Bilag 1

RQ105 - Felterne skal være inddelt i 5 typer.

RQ105.1 - Territory skal kunne købes når en spiller lander på feltet.

RQ105.1.1 - Hvis feltet allerede er ejet af en anden spiller, skal man betale feltspecifik afgift til ejeren. - Bilag 1

RQ105.2 - Refuge skal udbetale en pengesum til den spiller der lander på feltet.

RQ105.3 - Tax feltet skal give spilleren de to nedenstående muligheder. Spilleren vælger selv.

RQ105.3.1 - Spilleren skal betale 10% af sin formue.

RQ105.3.2 - Spilleren skal betale et felt specifikt beløb. - Bilag 1

RQ105.4 - Labor camp skal kunne købes når en spiller lander på feltet.

RQ105.4.1 - Hvis feltet allerede er ejet af en anden spiller, skal man betale afgift til ejeren.

RQ105.4.2 - Afgiften skal være et slag med 2 terninger ganget med 100 og yderligere ganget med antallet af Labor camp hos den samme ejer.

RQ105.5 - Fleet skal kunne købes når en spiller lander på feltet.

RQ105.5.1 - Hvis feltet allerede er ejet af en anden spiller, skal man betale en afgift, baseret på antal Fleet som ejeren har, til ejeren. - Bilag 1

RQ106 - Spillerne skal skiftes til at slå med terningerne.

RQ106.1 - Spillerne skal slå med 2 terninger.

RQ107 - Spilleren skal starte på et "Startfelt".

RQ107.1 - Spilleren skal altid rykke frem det antal felter som terningerne viser.

RQ107.2 - Spilleren skal, udover første tur, rykke fra det felt de landede på under deres sidste kast med terningerne.

RQ107.3 - Spillerne skal gå i ring på brættet, hvilket vil sige at slår man højere, med terningerne, end der er felter tilbage, rykker man videre fra feltet start.

Brugbarhed:

RQ201 - Spillet skal være intuitivt og kunne spilles uden yderligere introduktion.

Pålidelighed:

RQ301 - Spillet skal kunnes gennemføres uden kritiske fejl og mangler.

Performance:

RQ401 - GUI'en skal være hurtigere end 333 millisekunder når den viser terningerne.

RQ402 - Spillet skal kunne køres uden mærkbare forsinkelser.

Supportering

RQ501 - Spillet skal ikke vedligeholdes efter aflevering til kunde.

RQ502 - Spillet skal let kunne oversættes til et andet sprog.

Design begrænsning

RQ601 - Spillet skal kunne køres på Windows.

RQ602 - Spillet skal kodes i Eclipse.

RQ602.1 - Koden skal have UTF-8 format.

Lovmæssige krav

RQ1001 - Spillet skal ikke i konflikt med lovmæssige krav.

Dokumentation

Vores analyse af kundens vision er set som vigtig for at kunne skabe det produkt som kunden ønsker. Det vil under normale omstændigheder foregå med flere møder mellem udviklere og købere. Til dette projekt har vi valgt at tolke kundens vision uden kunden har haft yderligere indsigelser i projektet. Denne løsning kan bruges til mindre og mere tydelige projekter for at forøge effektiviteten og undgå spild- eller dobbeltarbejde.

Da vi nu kender kravene til vores program, kan vi begynde at analysere dem og opdele dem i grupper som giver os en bedre forståelse af spillet. Dette er kaldet for *Use-cases*. Det er den første opdeling af programmets krav som vi har lavet og lægger grund for alle andre diagrammer.

Vi har i vores program valgt at benytte arv. Dette er en funktion i Java som giver os udviklere mulighed for at lave en klasse som indeholder en masse general information. Vi kan så efterfølgende lave en klasse der er en smule mere specifik og nedarbejde alle de generelle metoder og tilføje de nye specifikke metoder. I vores program er det særlig brugbart i til at holde styr på vores felter.

Vi har valgt at bruge en abstract klasse *felter*, det er en superklasse der ikke kan blive instantieret, til at definere vores generelle egenskaber. En klasse der passer perfekt med vores tidligere nævnte brug af arv. Vi har skrevet koden i Java og de understøtter kun enkelt arv, og vi kan derfor kun *extend* en abstract klasse.

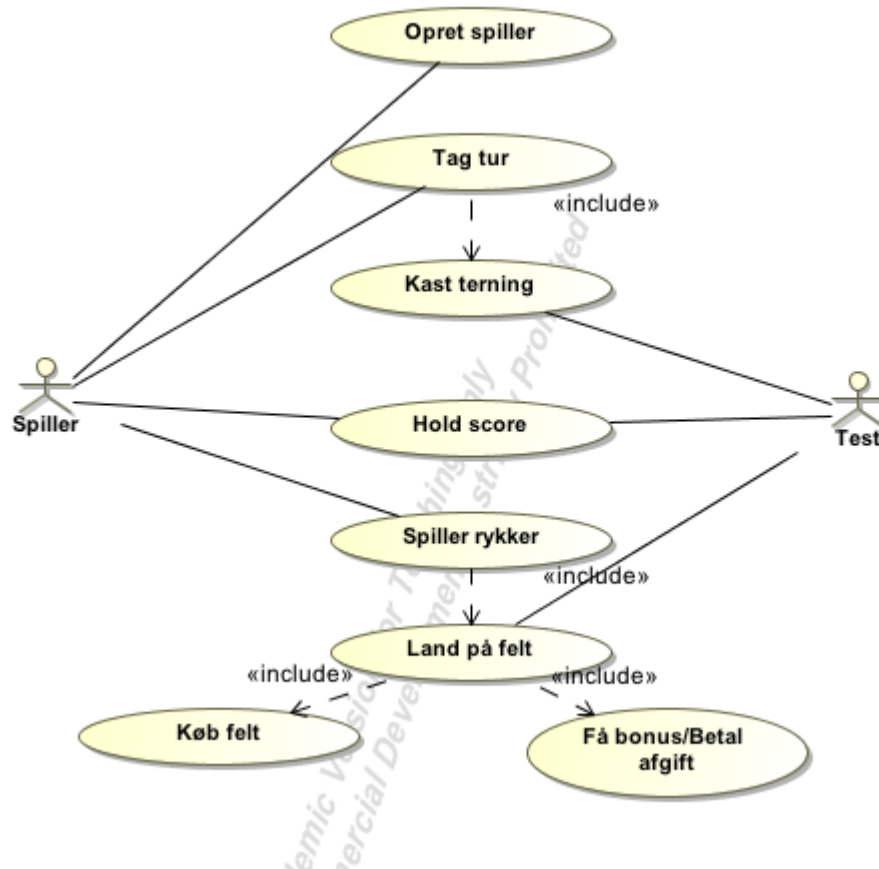
Som en del af vores nedrivning har vi en abstract metode *landPaaFelt*. Denne metode går igen i vores underklasser, og er styret af hvilken underklasse der bliver kaldt. Metoden er helt generel men forskellen ligger i hvor metoden bliver kaldt fra.

Med en solid baggrund vha. diagrammer og modeller, kunne vi begynde på vores kode som skulle laves ud fra GRASP principperne - *creator, controller, high cohesion, information expert, low coupling*. Vi har lavet vores creator klasse som består af vores spil-mekanismer.

Der er her lagt betydeligt størst vægt på har været at lavet det så objektorienteret som muligt med klasser som havde et ansvar der gav mening. Dette er *high cohesion*-delen. At opdele klasserne i en sådan grad at vi kunne give dem klare og tydelige ansvarsgrupper. Dette gør den nem at genbruge og hjælper med at kunne vedligeholde det nemmere. Disse logisk ansvarsfulde klasser skal herefter have så lidt afhængighed mellem hinanden som muligt. Dette er *low coupling*. Vores Gui har vi desværre ikke haft tid til at implementere under principperne for grasp, og dermed overholder de heller ikke *low cohesion*. Spillets controller objekt er det første objekt der bliver kørt vha brugerfladen.

Use case model

Et use case diagram er vores fortolkning af hvordan spillet er opbygget. I nedenstående diagram følges det kronologisk oppefra og ned. Dette er ikke et krav i en use case, men kan anses som en ekstra hjælp til egen forståelse. Hver use case beskriver hvad der sker i programmet vha. små generelle scenarier.



Figur 1.1

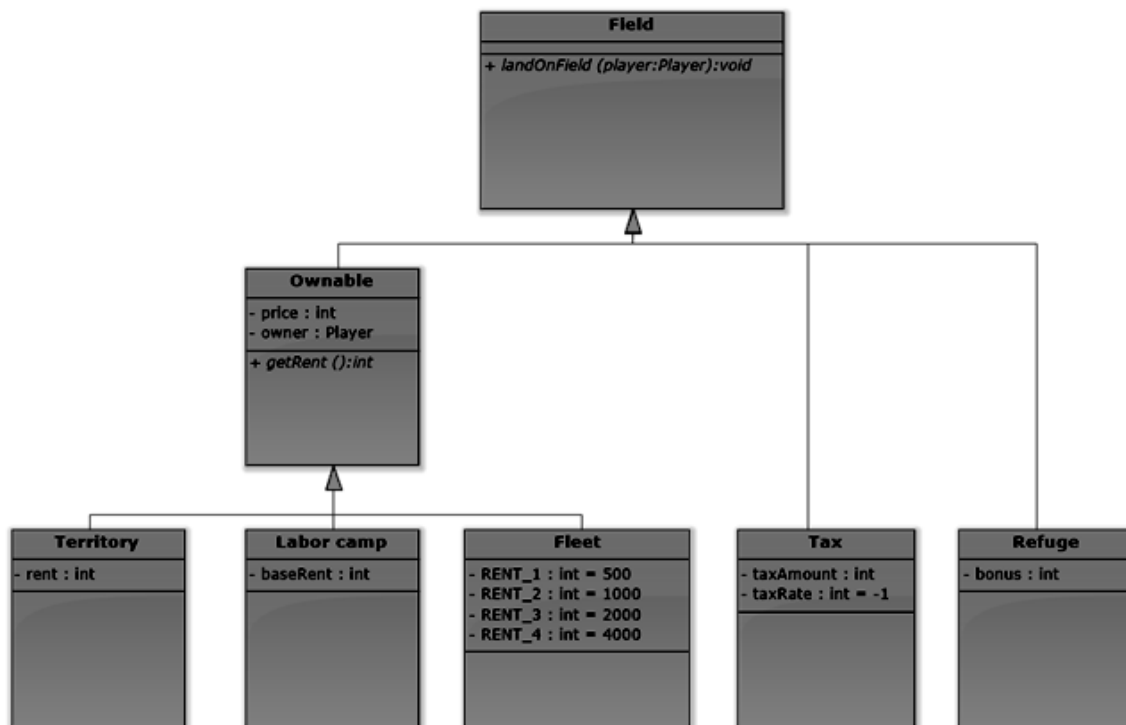
Vores use-case model over projektet, har taget inspiration fra vores tidligere use-case modeller*, hvor vores spiller aktør har adgang til de fornødne funktioner for at kunne spille spillet. Vores test aktør har mulighed for at lave Junit på terningerne, scoren og felterne.

Når vores spiller lander på et felt, vil en af to use-cases ske. Enten kan spilleren købe det felt spilleren lander på eller passere turen videre. Den anden use-case ændre pengebeholdningen, dette sker ved at lande på felter allerede ejet af andre spillere, eller ved at modtage penge af banken. Det hele er inkluderet i vores use-case om at spilleren rykker sin brik.

*- ref 2.0.

Klassediagram

Vores projektleder har lavet os et klassediagram over felter og dets underklasser med et ønske om at vi inkorporerer det i vores kode. Kode skal laves ved hjælp af nedarvning fra klasser mere generelle. Dette er kun et udsnit af det samlede klassediagram og reflektere derfor ikke hele projektet men kun en mindre specifik del. Klassediagrammet er oversat til dansk i koden og vises her med projektlederens attributter. Vi har bestræbt os efter at overholde de ønskede attributter og mener selv de er inkorporeret i vores klasser, muligvis under andet navn da vores kode er på dansk. Alle vores klasser i dette diagram er underlagt *Land på felt* use-casen og dens under cases.



Figur 1.2

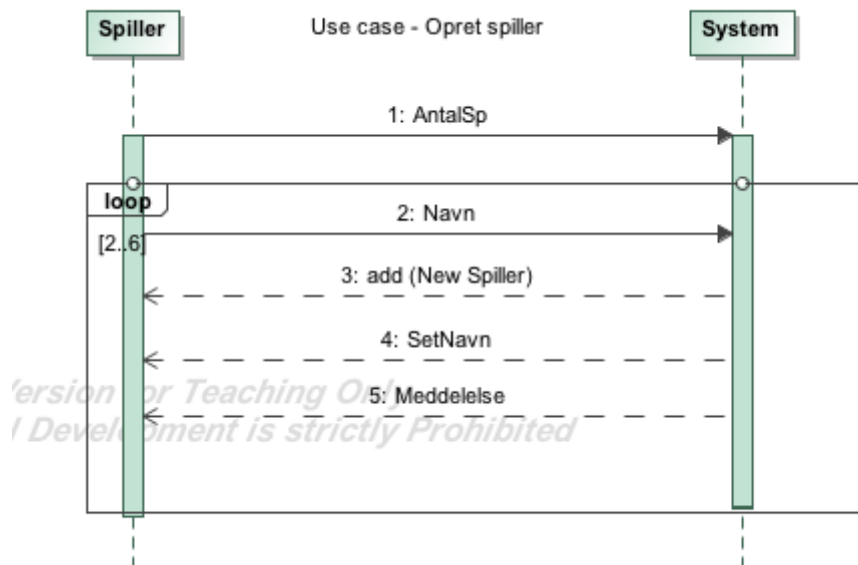
På klassediagrammet kan vi se hvordan opdelingen af programmets felter er opdelt og dermed få en bedre forståelse for hvornår man er underlagt en klasse specifik klasse.

Klassen som bliver nedarvet er felter, den bliver nedarvet i tre underklasser som hver hedder *ejerskab*, *skat* og *Helle*. Mens skat og helle ikke har nogle yderligere underklasser så har ejerskab. Denne klasse styrer, som navnet forhåbentlig indikere, hvilke felter der kan ejes, og da der er tre forskellige typer felter der kan blive købt af spillere, virker det kun logisk at der skulle være de tre underklasser

Sekvensdiagram

Vi har lavet et sekvensdiagram over vores use case *opret spiller* for at vise et lille eksempel i rapporten.

Opret spiller bliver startet af vores controller, som kører hele vores use case med et enkelt metodekald. Vi laver system sekvensdiagrammet med henblik på at vise et scenarie af vores program, og vise hvilken data der bliver sendt frem og tilbage mellem spillet og aktører.



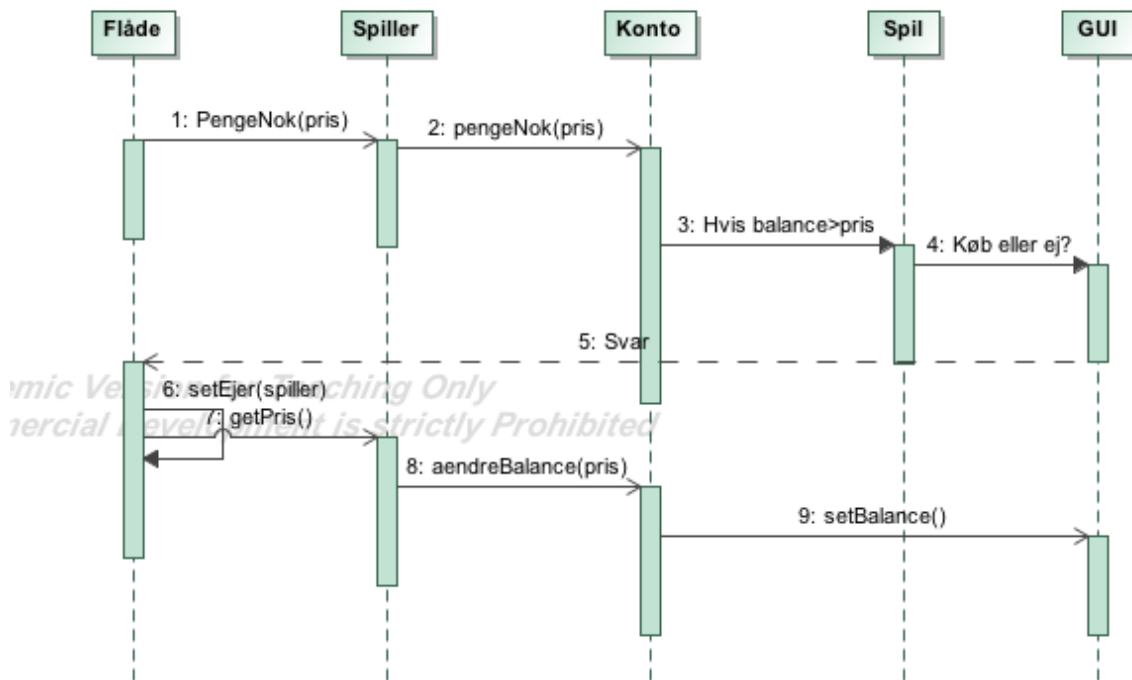
figur 1.3

Sekvens diagrammet viser at metoden spørger spilleren for input og bruger dette input i en loop med minimum grænse på 2 og maks på 6. Spillerne indtaster herefter sine navne og for hvert navn, opretter programmet en konto til spilleren og sætter en række variabler for spilleren til videre brug i koden.

Design sekvensdiagram

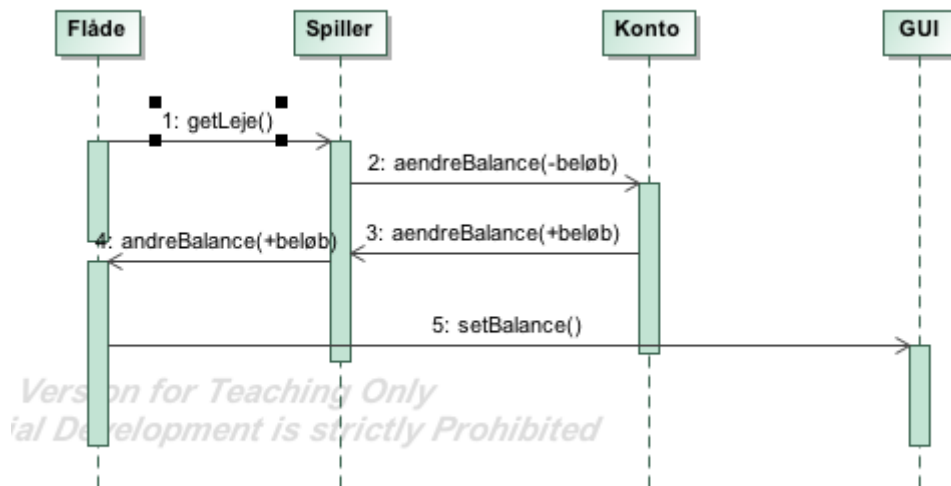
Vi laver et design sekvens diagram der viser hvordan klasserne snakker sammen, i landPaaFelt i Flåde klassen. Der er to forskellige scenarier det er spændende at kigge nærmere på. Når den er ejet og man dermed skal betale leje, og når den ikke er ejet og man har muligheden for at købe den.

Vi kigger først på betingelsen at den er ikke er ejet.



Figur 1.4

Design sekvens diagram over “land på flåde”. Flåden tjekker først om spilleren har råd til at købe feltet, spilleren tjekker videre om der er nok penge på kontoen. Hvis spillerens konto er større end prisen på feltet, får spilleren mulighed for at købe det, ved en ja/nej mulighed i GUI'en. Hvis spilleren svarer at de gerne vil købe feltet, bliver spiller sat som ejer, og prisen på feltet bliver trukket fra kontoen. Til sidst sættes den nye balance i GUI'en.

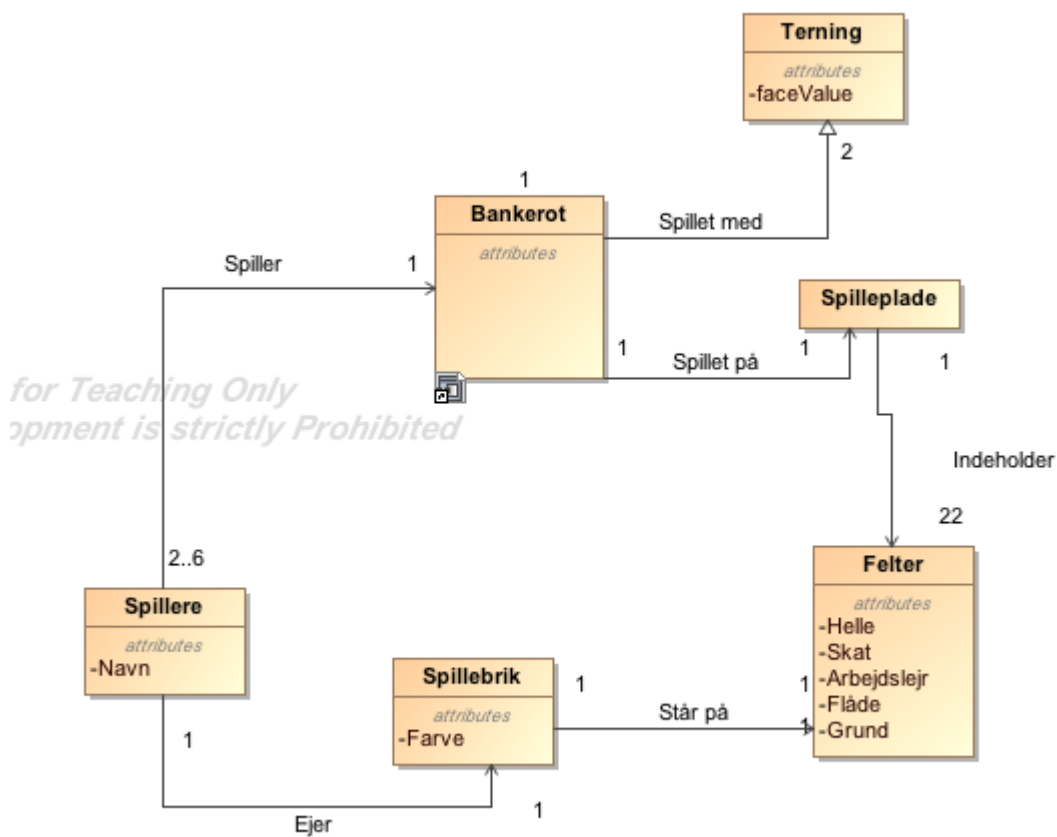


Figur 1.5

Figur 1.5 viser et design sekvensdiagram over “land på flåde” hvor det er opfyldt at feltet bliver ejet af en anden spiller, end den der lander på det. Spilleren lander på feltet, som oplyser hvor meget lejen er på. Dette afhænger af hvor mange andre felter ejeren er i besiddelse af. Derefter trækkes lejen fra kontoen ved *spiller.ændreBalance(-beløb)*. Når det er sket, lægges det samme beløb til ejerens konto ved *ejer.ændreBalance(beløb)*. Disse to værdier sættes i GUI'en.

Domænenemodel

Domænenemodellen viser relationen mellem de forskellige klasser i programmet og kan bruges til nemmere at forstå hvordan spillet hænger sammen. Det er en beskrivelse af hvordan spillet ville være i fysisk forstand. Dette er den helt grundlæggende del af programmet.



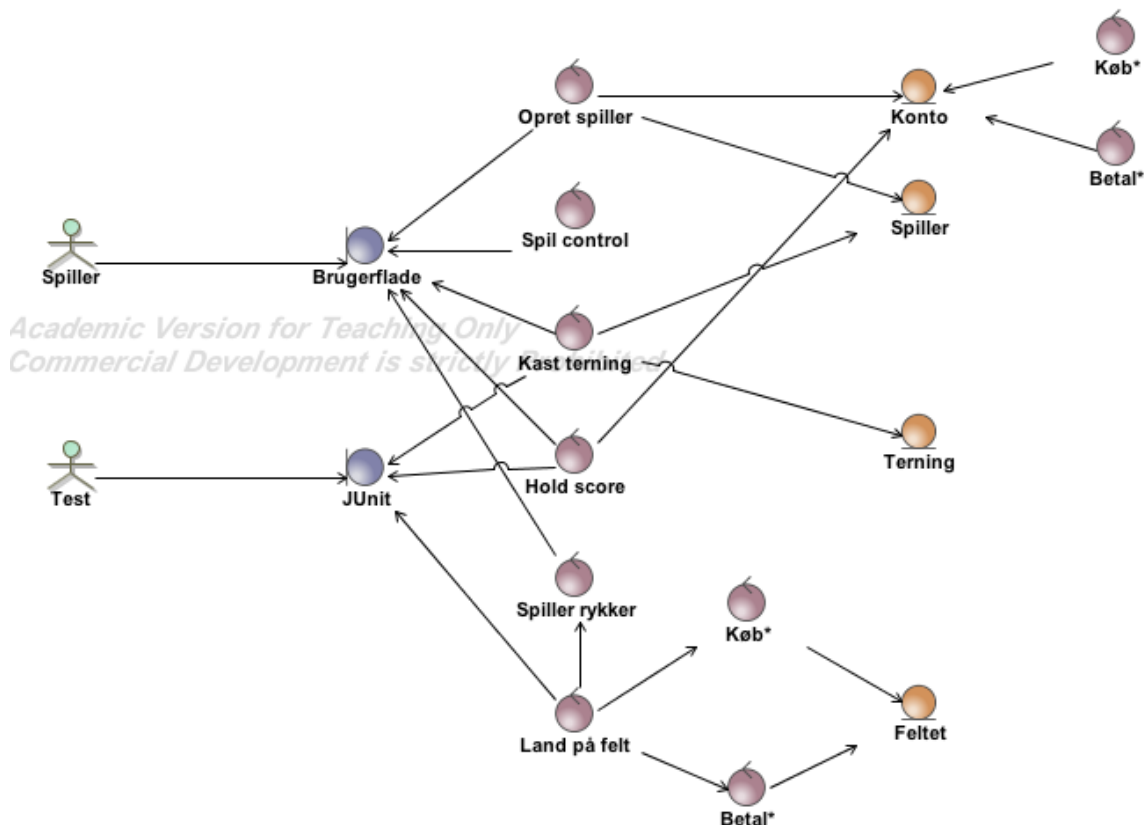
Figur 1.6

Selve spillet er kaldt Bankerot, og det bliver spillet med 2 terninger, hvis værdi bliver lagt sammen. Spillet bliver spillet på én spilleplade, som indeholder 22 felter med 5 forskellige egenskaber. Spillet kan spilles af mellem 2 og 6 spillere, som ved spillets begyndelse kan vælge et navn. Hver af de spillere får tildelt en brik, der har en farve. Dette forløb kan ses på figur 1.5.

BCE model

Vi har lavet vores BCE model med henblik på at vise hvordan brugerfladen bliver berørt af de enkelte controllere og samtidig vise hvordan spillet er opbygget.

Spilleren har adgang til brugerfladen, som har 5 controllere tilknyttet. Alle disse controllere styrer hver deres funktion i spillet og kan beskrives med en Use-case af tilsvarende navn. De fleste af vores controllere er desuden forbundet med en entity, der holder styr på alt lige fra spillerantal til hvilket felt de står på.



Figur 1.7

Test skal kunne påvise at spillet virker efter hensigten, det har vi gjort ved at lave en JUnit test på henholdsvis terningen, spillernes score og felterne. "Land på felt" controlleren styrer hvor spillerne står, hvad der sker når en spiller lander på et felt og sker der det der skal ske, på alle felterne. Det vil blive udviklernes opgave at sørge for at alle JUnits er godkendt.

* - henviser til at det er en dublet og ikke en ny controller, med intentionen om at give et bedre overblik i diagrammet.

Konklusion

Vi har som gruppe leveret det ønskede produkt til kunden og har opfyldt alle dets krav. Vores mål som gruppe beskrevet i introduktionen er opnået og vi har som gruppe levet op til vores egne høje forventninger.

Koden har vi lavet over flere omgange for at overholde kravene om GRASP. Vores endelige produkt er objektorienteret kode med klasser der har hver sit ansvarsområde med så lav afhængighed mellem hinanden som muligt.

Principperne for GRASP skulle fremstå tydeligt i vores kode.

Vi har under dette projekt ikke lavet en navneords analyse da vi byggede videre på klasser vi allerede havde, og yderligere fik udleveret klassenavne og attributter af vores projektleder til en stor del af det nye i programmet. Vi har dog fulgt unified process på andre punkter og har kørt vores projekt efter vandfaldsmodellen. En model der normalt får en del kritik men som vi mente ville passe til projektets størrelse og det egenskaber, dette kan også ses i vores tidsplan som er vedlagt - se bilag 4.

Arbejdsgang

Vores projekt tog fart ret hurtigt i forløbet og vi havde lavet det meste af vores analyse og UML inden projektet rigtigt var startet. Vi har arbejdet tæt sammen, med en jævn arbejdsfordeling hen over projektet. Derudover har vi haft planlagte møder hvor vi har kunnet drøfte vores individuelle opgaver såvel som projektet som helhed. Projektet har desuden båret præg af at vi ikke har en team leader men en mere demokratisk tilgang.

Vi lavede en risikovurdering og vidste hvilke risiko vi skulle holde øje med under projektet. Grundet vores lave gruppeantal var tidspres en meget reel risiko, som vi blev ramt af igen under vores forløb. Vi blev desværre også ramt af en risiko der ikke var direkte i vores risikovurdering, vi manglede at inkorporere GRASP principperne i vores kode og var ude af stand til at rydde op i alt den kode vi havde og besluttede at prøve forfra.

Produkt

Vores produkt lever op til vore forventninger med alle de funktioner kunden ønsket. Alle kundens krav er opfyldt inklusiv de krav vi har anbefalet kunden at inkludere.

Test

Vi har som del af kvalitetssikringen omkring koden, lavet en række test hvilket inkludere positiv- og negativ test og systematiske test som JUnits. Testene skal bekræfte at vi som udviklere har levet op til kravene stillet af kunden.

Vi har desuden lavet en del black box test i form af at kører programmet og holde øje med inputs og det tilsvarende resultat. Der er i denne del ikke holdt øje med hvordan programmet har opnået resultatet, her har det været mest vigtigt at resultatet var rigtigt.

Vi som udviklere har løbende kørt vores program for at tjekke for eventuelle syntax fejl der måtte kunne opstå. Da den endelige kode stod færdig galt det for os om at teste spillet som det oprindeligt er tænkt af os at det skulle kunne køres

Efterfølgende testede vi spillet som det ikke oprindeligt var tænkt at skulle køres. Dette kaldes en negativ test og inkludere at man som bruger af den pågældende software skal prøve at ødelægge programmet og få det til at fejle. Dette kan gøres ved at give inputs i programmet som det ikke kan håndtere eller lignende. Mange af disse fejl er blevet fjernet for at optimere brugervenligheden.

JUnit test

JUnit tests er en vigtig del for udviklingen af software, det giver os mulighed for at teste programstumper systematisk så man derfor kan fjerne de menneskelige fejl ved kodning og samtidig spare tid.

Vores første projekt til kunden blev leveret med en JUnit på vores terninger. Testen slog 100.000 og måtte ikke afvige med mere end 4%, for hvert værdi. Testen gik igennem upåklageligt og ligger inkluderet i GitHub.

Hold scoren har vi lavet en test over i vores tidligere projekt om hvorvidt scoren ikke måtte kunne blive negativ. Da vi har et krav om at man taber spillet hvis man får en negativ score, fandt vi det passende at vi kørte testen igennem over vores nye kode. Testen gik igennem og spillere går bankerot når deres konto bliver negativ. Testen ligger på GitHub.

Vi har lavet en JUnit test for hver af felt typernes *landPaaFelt* metode. Vores projektleder havde vedlagt en JUnit han mente kunne bruges til *helle* felterne. Vi har dog valgt en lidt anderledes kode end vores projektleder oprindeligt havde tænkt og har derfor kun brugt den som en skitse for vores andre JUnits. Testen er lavet uden om GUI'en og vil derfor ikke kunne vises direkte ved at hente fra Github.

Litteraturliste

APPLYING UML AND PATTERNS - Craig Larman, third edition
Objektorienteret programmering i java - Jacob Nordfalk, 2. udgave.

Referencer

Ref 1.0 - CDIO del 3.docx - version 7. nov 2014. Kopieret udsnit af opgavebeskrivelsen, udleveret på campusnet.dtu.dk til studerende.

Ref 2.0 - CDIO 2 - Afleveret 4/11-2016

Ordforklaring

FURPS+ - Functionality Usability Reliability Performance Supportability +

FURPS+ er en model for hvordan man kan sortere sin kravspecifikation for nemmere forståelse, den er oprindeligt lavet som bare FURPS men der kom hurtigt flere grupper af krav til og er siden blevet til FURPS+

GRASP - General Responsibility Assignment Software Patterns

GRASP er en række principper inden for kodning som sikrer at det er nemt at fejlfinde og videre bygge på. Det handler meget om ansvarsfordeling og uafhængighed mellem klasserne.

UML - Unified Modelling Language

UML er en standard for diagrammer og strukturer i objektorienteret kode.

GUI - Graphical User Interface

GUI er det visuelle en bruger møder når han/hun bruger vores produkt. Oversættes til grafisk brugerflade.

Bilag

Bilag 1 - Feltliste

Vedlagt af kunden som det af kravene. Felterne er oversat til dansk i vores opgave.

Feltliste:

1. Tribe Encampment	Territory	Rent 100	Price 1000
2. Crater	Territory	Rent 300	Price 1500
3. Mountain	Territory	Rent 500	Price 2000
4. Cold Desert	Territory	Rent 700	Price 3000
5. Black cave	Territory	Rent 1000	Price 4000
6. The Werewall	Territory	Rent 1300	Price 4300
7. Mountain village	Territory	Rent 1600	Price 4750
8. South Citadel	Territory	Rent 2000	Price 5000
9. Palace gates	Territory	Rent 2600	Price 5500
10. Tower	Territory	Rent 3200	Price 6000
11. Castle	Territory	Rent 4000	Price 8000
12. Walled city	Refuge	Receive 5000	
13. Monastery	Refuge	Receive 500	
14. Huts in the mountain	Labor camp	Pay 100 x dice	Price 2500
15. The pit	Labor camp	Pay 100 x dice	Price 2500
16. Goldmine	Tax	Pay 2000	
17. Caravan	Tax	Pay 4000 or 10% of total assets	
18. Second Sail	Fleet	Pay 500-4000	Price 4000
19. Sea Grover	Fleet	Pay 500-4000	Price 4000
20. The Buccaneers	Fleet	Pay 500-4000	Price 4000
21. Privateer armade	Fleet	Pay 500-4000	Price 4000

Bilag 2 - Use case *Fully dressed*

Vi har fået som opgave at lave en *fully dressed* use case af *land på flaade* metoden og dens egenskaber. Vi har valgt at lave et skema som ved vores andre use cases, men følger vejledningen beskrevet af Craig Larman i hans bog - se *litteraturliste*.

Use case - Land på Flaade
Level : User-goal
Primær aktører: Spiller
Interessenter og interesser: Spilleren der lander på feltet - Vil gerne kunne købe feltet Spilleren der ejer feltet - Vil gerne have leje for feltet Spillere der ejer samme type felt - Kan forøge deres leje vha. flere af feltypen
Forudsætninger: Spilleren lander på feltet af typen <i>flaade</i>
Slutresultat: Spilleren har enten købt- , betalt leje for- eller afslået køb af feltet
Succesfuldt scenarie: Spilleren lander på feltet af typen <i>flaade</i> Spilleren køber eller betaler leje af feltet Spillerens tur slutter
Alternative flows: Spilleren ejer selv feltet Spilleren går bankerot
Specielle krav: Lejen for feltet fordobles for hver felt af typen <i>flaade</i> som ejeren af feltet har
Teknologi og variable data:

Feltets ejer hentes og ejerens antal af denne type felt hentes

Feltets lejepris

Feltets pris

Spilleren pengebeholdning

Hyppighed af hændelser: Kan ske næsten hele tiden, under spillets brug.

Bilag 3 - Use cases

Use case skema for hver use case som er indgået i vores use case model - *figur 1.0*

Use case
Navn: Opret spiller
ID: 1
Primær aktører: Spiller
Forudsætninger: Programmet er startet
Beskrivelse: System spørg om antallet af spillere der skal oprettet til spillet og deres navn. antallet af spillere der må oprettes er 2-6.
Main flow: 1. System spørg om antallet af spiller 1.1 Spillerne indtaster deres navne
Slutresultat: Mellem 2 - 6 spillere oprettes Hver spiller har et selvvalgt navn
Alternative flows: Ugyldigt spillerantal

Use case
Navn: Tag tur
ID: 2
Primær aktører: Spiller
Forudsætninger: Spillere er oprettet
Beskrivelse: Spilleren får sin tur til at kaste terningerne
Main flow:

1. Spilleren tager sin tur 1.1 Spilleren kaster terningerne
Slutresultat: Næste spillers tur
Alternative flows: Ikke andre spillere tilbage

Use case
Navn: Kast terning
ID: 3
Primær aktører: Spiller
Sekundær aktører: Test
Forudsætninger: Det spillerens tur
Beskrivelse: Spilleren kaster med to, 6 sided, terninger.
Main flow: 1. Spilleren kaster med to terninger 1.1 summen af de to terninger lægges sammen
Slutresultat: Summen af de to terninger gives videre til use case <i>Spiller rykker</i>
Alternative flows:

Use case
Navn: Hold scoren
ID: 4
Primær aktører: Spiller
Sekundær aktører: Test
Forudsætninger: Spilleren er landet på et felt som ikke er et neutral felt Der skal enten: betales en afgift, modtagelse af afgift, købes et felt eller modtagelse af bonus
Beskrivelse: Spillerens kontos pengebeholdning bliver ændret ud fra en forudsætning
Main flow: 1. Penge differencen bliver lagt til spillerens pengebeholdning
Slutresultat: Ændret en til to af spillernes pengebeholdning Hvis pengebeholdningen kommer under nul, går spilleren konkurs
Alternative flows:

Use case
Navn: Spiller rykker
ID: 5
Primære aktør: Spiller
Sekundær aktør: Test
Forudsætninger: Spilleren har slået med terningerne
Beskrivelse: Efter at have kastet med terningerne rykker spilleren summen af de to terninger, felter frem på brættet. Hvis det sidste felt passerer rykker spilleren tilbage til start og rykker de resterende felter frem derfra.

Main flow: <ol style="list-style-type: none"> 1. Spilleren flyttes summen af terningerne felter frem på brættet. 1.1 Spillerens pengebeholdning kan blive ændret ud fra handlinger sket på feltet
Slutresultat: Kan ændre pengebeholdningen Kan medføre spilleren går konkurs
Alternative flows:

Use case
Navn: Land på felt
ID: 6
Primær aktører: Spiller
Forudsætninger: Spilleren har rykket sig til et nyt felt
Beskrivelse: Spilleren lander på et felt og ud fra hvilken type felt der landes på og om feltet ejes eller ej, er der forskellige aktiviteter
Main flow: <ol style="list-style-type: none"> 1. Spilleren lander på et felt 2. System ser på hvilken type felt spilleren er landet på <ol style="list-style-type: none"> 2.1 Tjekker om feltet kan have en ejer
Slutresultat: Spilleren står på et nyt felt
Alternative flows:

Use case
Navn: Køb felt
ID: 7
Primær aktører: Spiller
Forudsætninger: Feltet kan have en ejer Spilleren har råd til, at købe feltet Feltet er ikke tidligere blevet købt
Beskrivelse: Spillere har mulighed for at købe felter de lander på, hvis feltet ikke tidligere er blevet købt. Feltet koster et forudbestemt beløb som bliver trukket fra spilleren konto.
Main flow: <ol style="list-style-type: none"> 1. Spillet ser om feltet er til salg, hvis ja giver spilleren muligheden for at købe feltet. 2. Spilleren køber feltet og prisen bliver trukket fra spillerens konto.
Slutresultat: Spilleren ejer nu et nyt felt og vil modtage betalings afgifterne fra feltet hvis andre spiller lander på feltet.
Alternative flows: Feltet bliver ikke købt Feltet ejes i forvejen

Use case
Navn: Få bonus/Betal afgift
ID: 8
Primær aktører: Spiller
Forudsætninger: Spilleren er landet på et felt der ikke kan ejes Spilleren er ikke gået bankerot
Beskrivelse: Hvis spilleren lander på et felt der er ejet af en anden spiller, eller et felt med en form for afgift, skal beløbet trækkes fra spillerens konto. Ligeledes får spilleren et beløb sat ind på sin konto, hvis spilleren lander på et bonusfelt.

Main flow:

1. Spilleren er landet på et felt
2. Får eller betaler beløb

Slutresultat:

Ændring i spillerens pengebeholdning

Alternative flows:

Bilag 4 - Tidsplan

Vores tidsplan lagt på første dag af projekt start.

	Uge 1			Uge 2			Uge 3		
Kravspecifikation									
Risikovurdering									
UML									
Kode									
Gui									
Test									
Rapport									

Bilag 5 - JUnit Terning

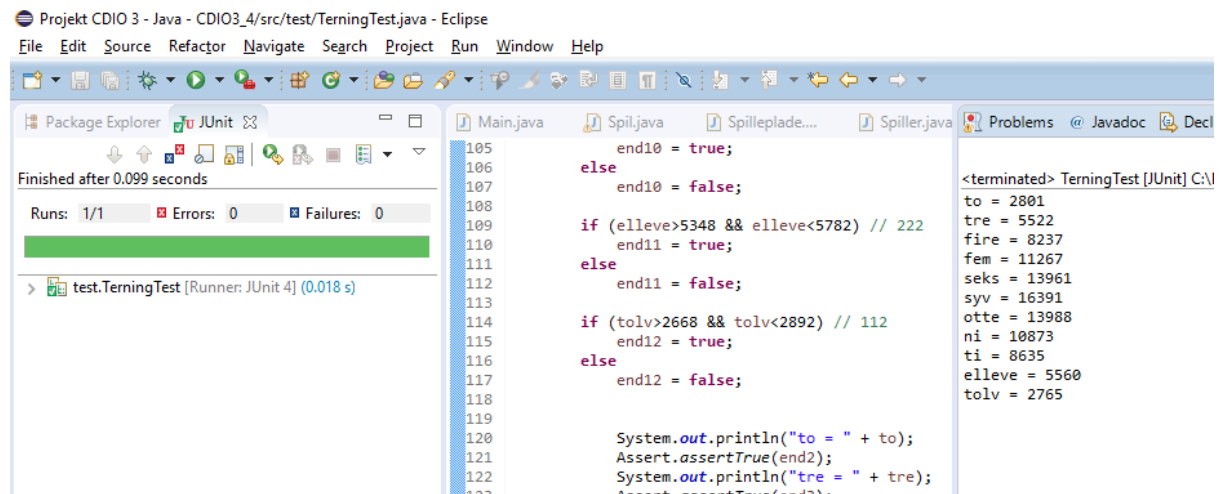
Billeddokumentation for JUnit test terninger med vedlagt skema over det statistisk sandsynlige tilfældighed for to terninger.

Skema over det sandsynlige resultat fra vores raflebæger

	Terning 1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							</
--	-----------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

Billede 1 ud af 3

Skærmpoint fra vores JUnit



Billede 2 ud af 3

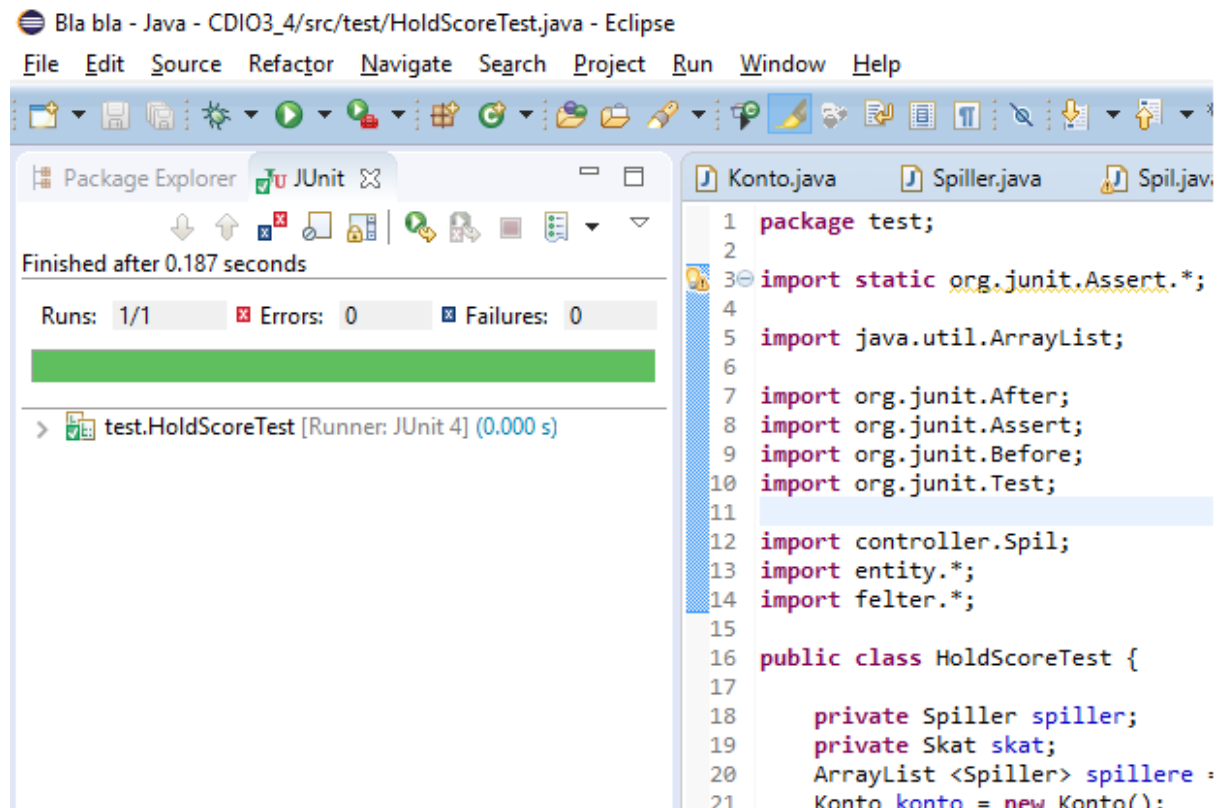
Oversigt over hvor mange gange hver værdi blev slå med vores raflebæger.

```
<terminated> TerningTest [JUnit] C:\
to = 2801
tre = 5522
fire = 8237
fem = 11267
seks = 13961
syv = 16391
otte = 13988
ni = 10873
ti = 8635
elleve = 5560
tolv = 2765
```

Billede 3 ud af 3

JUnit Hold Score

Vi har lavet en JUnit test case på at scoren ikke skulle kunne blive negativ i vores kode. Vi kørte i en for loop at vores spiller lander på et felt med negativ indflydelse på spillerens pengebeholdning, spillere blev sat til at gå bankerot og og vi forventede at spillerens pengebeholdning blev 0. Det blev den også réelt og testen gik igennem.

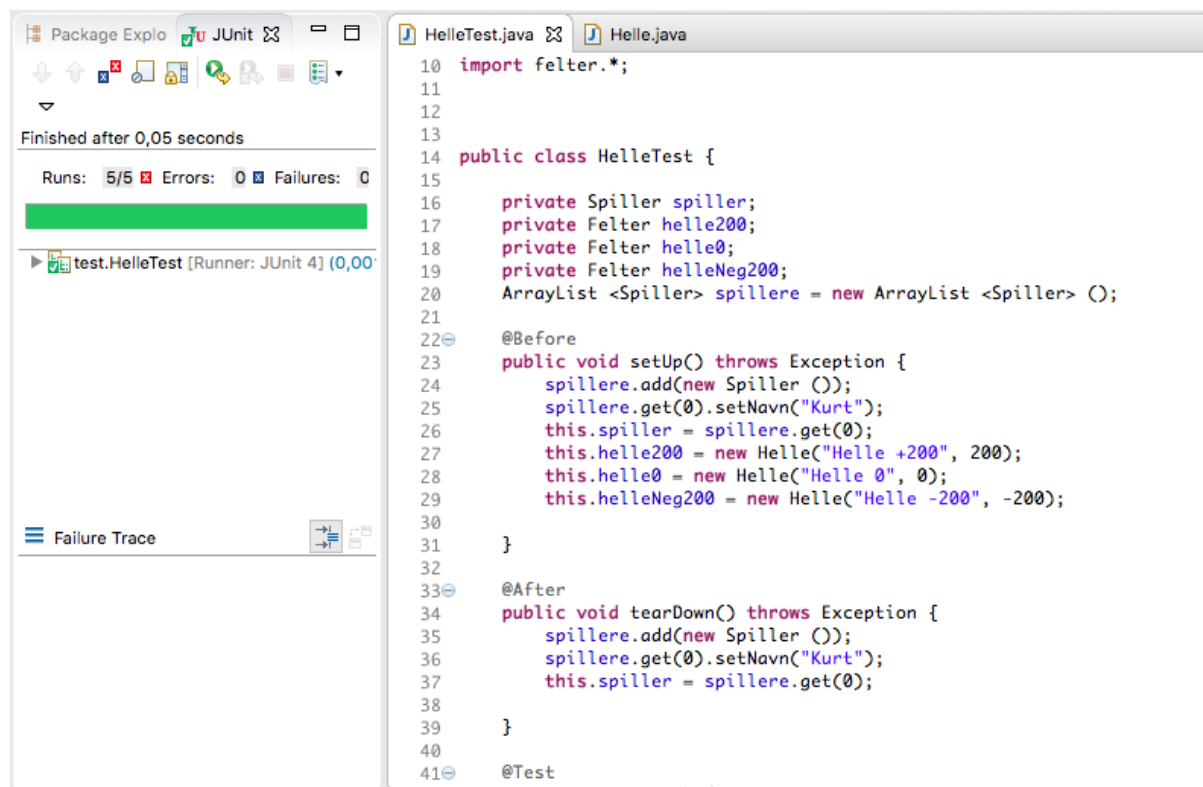


The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. The Package Explorer on the left shows a project named 'Bla bla' with a sub-package 'test'. The JUnit runner window shows 'Finished after 0.187 seconds' and 'Runs: 1/1', 'Errors: 0', 'Failures: 0'. The main editor window displays the source code for 'HoldScoreTest.java'.

```
1 package test;
2
3 import static org.junit.Assert.*;
4
5 import java.util.ArrayList;
6
7 import org.junit.After;
8 import org.junit.Assert;
9 import org.junit.Before;
10 import org.junit.Test;
11
12 import controller.Spil;
13 import entity.*;
14 import felter.*;
15
16 public class HoldScoreTest {
17
18     private Spiller spiller;
19     private Skat skat;
20     ArrayList <Spiller> spillere =
21     Konto konto = new Konto();
```

Bilag 6 - JUnit landPaaFelt Helle

Billede dokumentation over vores JUnit af *landPaaFelt* i *helle* klassen. Vores projekt leder havde udleveret en skitse som vi har lavet minimale ændringer i for at tilpasse vores ønskede kode.

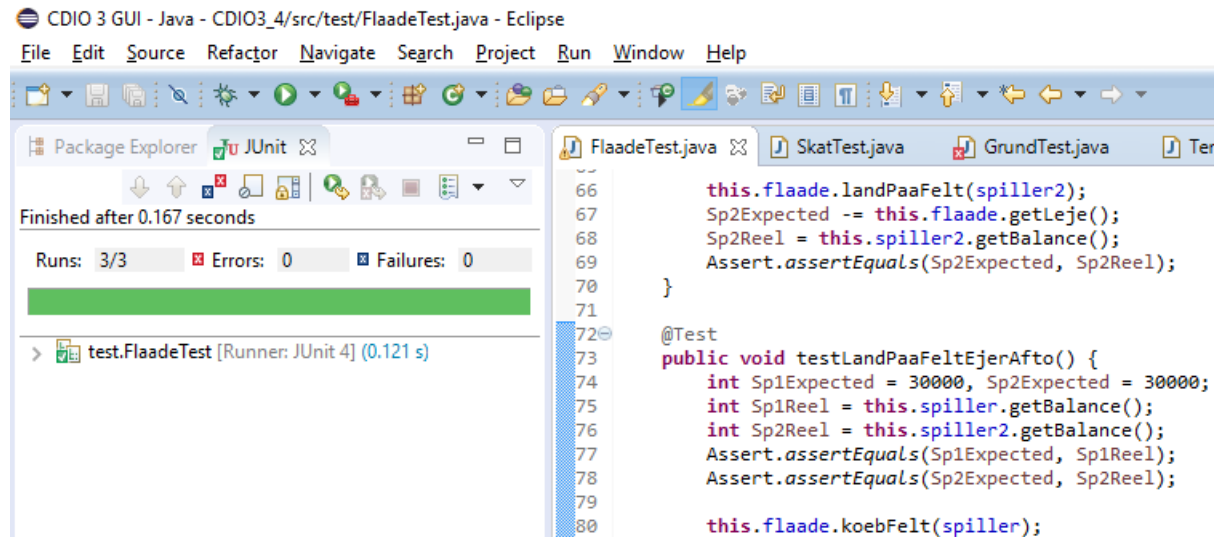


The screenshot shows an IDE with two main panels. The left panel displays the JUnit test results for `test.HelleTest`. It indicates that the tests finished after 0,05 seconds, with 5/5 runs, 0 errors, and 0 failures. A green progress bar is visible. The right panel shows the source code for `HelleTest.java`. The code includes imports for `felter.*`, a `public class HelleTest` with private fields for `Spiller` and `Felter`, and an `ArrayList` for `spillere`. It also features `@Before` and `@After` methods for setup and teardown, and a `@Test` method.

```
10 import felter.*;
11
12
13
14 public class HelleTest {
15
16     private Spiller spiller;
17     private Felter helle200;
18     private Felter helle0;
19     private Felter helleNeg200;
20     ArrayList<Spiller> spillere = new ArrayList<Spiller> ();
21
22     @Before
23     public void setUp() throws Exception {
24         spillere.add(new Spiller ());
25         spillere.get(0).setNavn("Kurt");
26         this.spiller = spillere.get(0);
27         this.helle200 = new Helle("Helle +200", 200);
28         this.helle0 = new Helle("Helle 0", 0);
29         this.helleNeg200 = new Helle("Helle -200", -200);
30     }
31
32
33     @After
34     public void tearDown() throws Exception {
35         spillere.add(new Spiller ());
36         spillere.get(0).setNavn("Kurt");
37         this.spiller = spillere.get(0);
38     }
39
40
41     @Test
```

Bilag 7 - JUnit landPaaFelt Flaade

Vores JUnit over Flaade klasse.



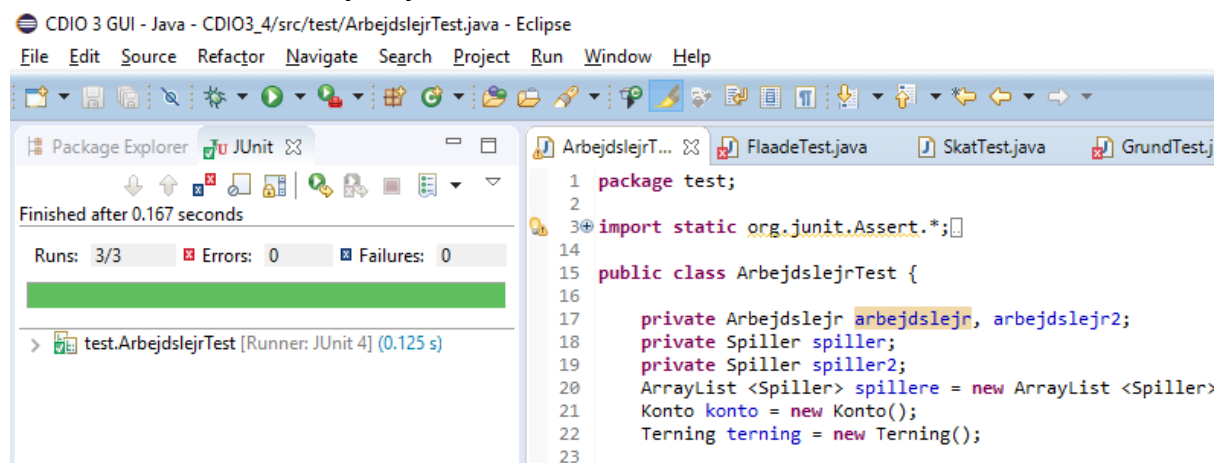
```
CDIO 3 GUI - Java - CDIO3_4/src/test/FlaadeTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit
Finished after 0.167 seconds
Runs: 3/3 Errors: 0 Failures: 0
> test.FlaadeTest [Runner: JUnit 4] (0.121 s)

FlaadeTest.java SkatTest.java GrundTest.java Ter
66 this.flaade.landPaaFelt(spiller2);
67 Sp2Expected -= this.flaade.getLeje();
68 Sp2Reel = this.spiller2.getBalance();
69 Assert.assertEquals(Sp2Expected, Sp2Reel);
70 }
71
72 @Test
73 public void testLandPaaFeltEjerAfto() {
74     int Sp1Expected = 30000, Sp2Expected = 30000;
75     int Sp1Reel = this.spiller.getBalance();
76     int Sp2Reel = this.spiller2.getBalance();
77     Assert.assertEquals(Sp1Expected, Sp1Reel);
78     Assert.assertEquals(Sp2Expected, Sp2Reel);
79
80     this.flaade.koebFelt(spiller);
```

Bilag 8 - JUnit landPaaFelt Arbejdslejr

JUnit test over vores Arbejdslejr klasse



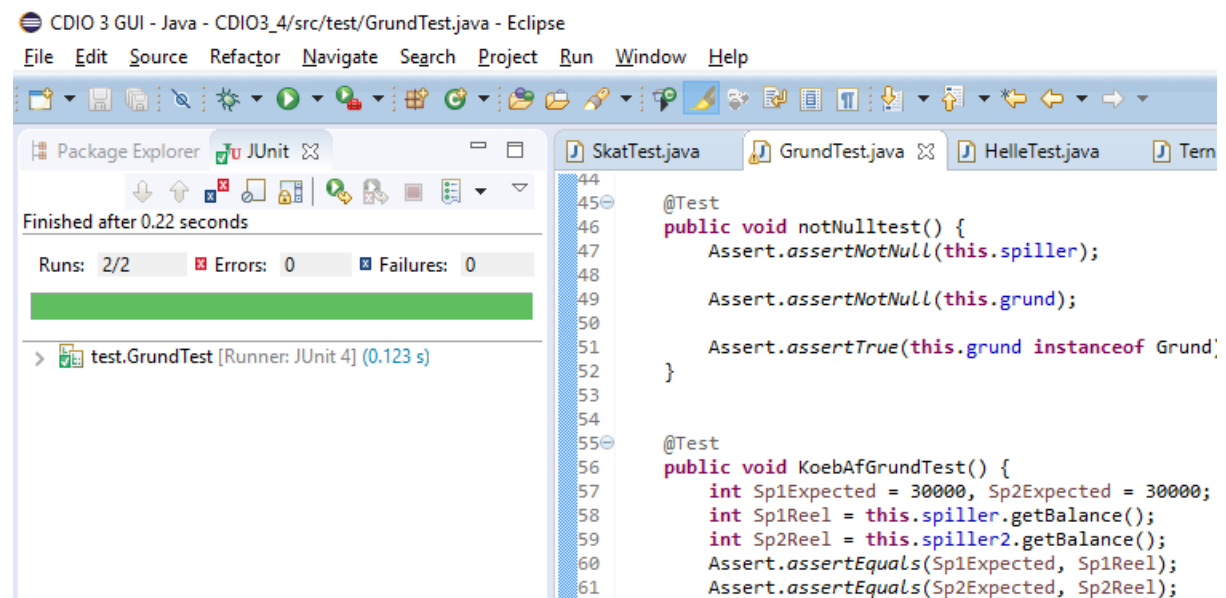
```
CDIO 3 GUI - Java - CDIO3_4/src/test/ArbejdslejrTest.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit
Finished after 0.167 seconds
Runs: 3/3 Errors: 0 Failures: 0
> test.ArbejdslejrTest [Runner: JUnit 4] (0.125 s)

ArbejdslejrT... FlaadeTest.java SkatTest.java GrundTest.j
1 package test;
2
3 import static org.junit.Assert.*;
4
14
15 public class ArbejdslejrTest {
16
17     private Arbejdslejr arbejdslejr, arbejdslejr2;
18     private Spiller spiller;
19     private Spiller spiller2;
20     ArrayList<Spiller> spillere = new ArrayList<Spiller>
21     Konto konto = new Konto();
22     Terning terning = new Terning();
23
```


Bilag 9 - JUnit landPaaFelt Grund

JUnit test for klassen Grund



Bilag 10 - JUnit landPaaFelt Skat

JUnit for klassen Skat

