



IMPLEMENTERING & TEST

02161 SOFTWARE ENGINEERING 1


Gruppe: 13

Rasmus Wiuff **s163977**

Mathies Henriksen **s200747**

Max-Emil Scotten **s204633**

Kasper Sylvest **s205281**

github.com/rwiuff/02161ExamProject 

8. maj 2023

INDHOLD

	Side
1 Forfatterskab	1
2 Programstruktur	1
2.1 Import af program som projekt	1
2.2 Start af programmet	1
2.3 Mappe struktur	1
3 White box test	2
3.1 createInitials()	2
3.2 generateProjectNumber()	4
3.3 createProjectActivity()	5
3.4 findWorkTimeRegistrationById()	7
4 Code coverage	8
5 Design by contract	10
5.1 generateProjectNumber	10
5.2 createInitials()	11
5.3 createProjectActivity()	11
5.4 findWorktimeRegistrationById()	12
6 Design mønstre	13
6.1 Program-lag	13
6.2 Præsentations-laget	14
7 SOLID principer	15
7.1 Single responsibility principle	16
7.2 Open/closed principle	16
7.3 Liskov substitution principle	17
7.4 Interface segregation principle	17
7.5 Dependency inversion principle	18
8 Konklusion	18
Figurer	20
Tabeller	20
Listings	20
Appendiks	21
A Klassediagram over program laget	23
B Klassediagram over præsentations-laget	24
C Klassediagram over facade-laget	25
D Sekvensdiagram: CreateProjectActivity()	27

1 FORFATTERSKAB

Følgende tabel har hensigten at indikere hvor en persons primære arbejde har været. Da projektet er udviklet iterativt og test-drevent, har vi alle været forfattere på alle filer og metoder. Projektet er udviklet som et team.

Tabel 1: Oversigt over forfatterskaber i projektet

Forfatter	Afsnit	Filer og klasser
Rasmus Wiuff 163977	Afsnit 1, Afsnit 2, Afsnit 3.2, Afsnit 5.1, Afsnit 7.5	/facades/... /domain/Activity.java /domain/Report.java /domain/ReportPDFGenerator.java
Mathies Henriksen s200747	Afsnit 3.4, Afsnit 5.4, Afsnit 7.1, Afsnit 7.2	/app/... /exceptions/... /domain/Employee.java /domain/RegularActivity.java /domain/WorktimeRegistration.java
Max-Emil Scotten s204633	Afsnit 3.1, Afsnit 4, Afsnit 5.2, Afsnit 7.3	/persistence/... /viewModels/... /domain/ConvertibleToViewModelInterface.java
Kasper Sylvest 205281	Afsnit 3.3, Afsnit 5.3, Afsnit 6, Afsnit 7.4	/cli/... /helpers/... /domain/Project.java /domain/ProjectActivity.java /**/_Interface.java

2 PROGRAMSTRUKTUR

2.1. IMPORT AF PROGRAM SOM PROJEKT

2.1.1. Eclipse

2.1.2. IntelliJ

2.1.3. VS Code

2.2. START AF PROGRAMMET

For at køre TaskFusion skal følgende fil køres: `TaskFusion/src/main/taskFusion/cli/TaskFusionCLI.java`
Ingen kodeord er nødvendige for at køre programmet. Programmet kommer med demo-data installeret. Du kan derfor logge ind med følgende initialer; kasy, rawi, mach, mash. En jar-fil kan også oprettes med mvn package, der derefter kan køres. På UNIX-systemer skal den producerede jar-fil gøres eksekverbar med `chmod +x [filnavn]`, såfremt brugeren har de rette privilegier, så den anses som en eksekverbar fil inden forsøg på kørsel.

2.3. MAPPE STRUKTUR

Figur 1 viser mappestrukturen i Java projektet.

Figur 1: Mappestrukturen i Java projektet TaskFusion

```
TaskFusion ... Rodmappe til filer som pom.xml, .project and .clas-
    |
    |__ spath
    |__ features ... Cucumber feature filer
    |__ src
        |__ main
            |__ java
                |__ taskfusion ... .java filer
                    |__ app
                    |__ domain ... Domæne-lag
                    |__ exceptions ... Exception klasser
                    |__ helpers ... Diverse hjælper klasser
                    |__ persistency ... Lagrings-lag
                    |__ viewModels ... Visningsklasser
                    |__ facades ... Facade-lag
                    |__ cli ... CLI brugergrænse-lag
                        |__ controllers ... Menu controllere
                        |__ views ... Indholdssider
                        |__ components ... Genbrugelige CLI komponenter
                        |__ TaskFusionCLI.java ... Hovedprogram ved brug af CLI brugergrænsefladen
                |__ test
                    |__ java
                        |__ taskfusion
                            |__ cucumber ... Acceptance tests
                            |__ helpers ... Hjelpeklasser til tests
                            |__ junit ... Unit tests
                    |__ resources ... cucumber.properties
```

3 WHITE BOX TEST

I dette afsnit testes fire metoder med white box test ved brug af *JUnit*. Kildekode af hver metode vises, tabeller med information vedr. input og *execution paths* er ligeledes fremme. Formidlingsmæssigt, er det blevet valgt at vise de *execution paths* i koden med det pågældende test-sæt, således at det er tydeligt at observere, hvor og under hvilke omstændigheder disse *paths* aktiveres. Dette gør det desuden nemt at se, at alle udfald bliver testet.

3.1. CREATEINITIALS()

For at garantere at `createInitials()` fungerer efter intentionen og som angivet i kommentaren over metodedefinitionen i klassen `Employee`, skal vi sikre, at initialerne genereres i den rækkefølge, der er defineret af den kommentar (se Listing 1), og at metoden kaster en undtagelse, når det forventes. JUnit-implementeringen af denne test er at finde i `EmployeeTest.java`.

Listing 1: Kommentar til createInitials() kildekode

```
45  /*
46  * Method generates unique initials based on first- and lastName.
47  * Unique initials is assured by comparing generated initials to existing
48  * initials in the employee repo
49  * The following logic is followed:
50  * First 2 letters of firstName is always used, then adding:
51  * Letter 1 and 2 from lastName is tried
52  * Then letter 1 and 3, Then 1 and 4 and so on.
53  * If last name is exhausted, then add letter 2 and 3, then 2 and 4 and so on
54  * from the last name.
55  *
56  * Solution using 3 nested for loops, is generated by ChatGPT v.4
57  */
```

Listing 2: createInitials() kildekode med execution paths

```
1  private void createInitials() throws ExhaustedOptionsException {
2      EmployeeRepository employeeRepo = EmployeeRepository.getInstance();
3
4      for (int l1 = 0; l1 < lastName.length(); l1++) { // 1
5          for (int l2 = l1 + 1; l2 < lastName.length(); l2++) { // 2
6              String init = firstName.substring(0, 2)
7                  + lastName.substring(l1, Math.min(l1 + 1, lastName.length()))
8                  + lastName.substring(l2, Math.min(l2 + 1, lastName.length())); // 3
9
10             init = init.toLowerCase(); // 4
11             if (!employeeRepo.initialsExist(init)) { // 5
12                 this.initials = init; // 6
13                 return; // 7
14             }
15         }
16     }
17
18     throw new ExhaustedOptionsException("Kunne ikke generere unikke initialer"); // 8
19 }
```

For at dække alle branches går vi igennem de execution paths, som angivet i Listing 2 og afdækker dem med test der bruger de input-sæt, der kan ses nedenunder.

Tabel 2: Execution paths i createInitials()

Execution path	Input set	Input property
1 (true), 2 (true), 3, 4, 5 (true), 6, 7	A	There are fewer than 18 employees registered with first name "Michael", last name "Laudrup"
1 (true), 2 (true), 3, 4, 5 (false), 8	B	There are exactly 18 employees registered with first name "Michael", last name "Laudrup"
1 (true), 2 (false), 8	C	There are exactly five employees registered with first name "Michael", last name "Laudrup"
1 (false), 8	B	There are exactly 18 employees registered with first name "Michael", last name "Laudrup"

Tabel 3: Input sæt i `createInitials()`

Input set	Input Data	Expected result
A	The employee repository is empty, and 18 employees named Michael Laudrup are added.	Initials are generated according to the description in the comment above <code>createInitials</code> .
B	The employee repository has 18 employees named Michael Laudrup, and another is created.	An <code>ExhaustedOptionsException</code> is thrown with the message "Kunne ikke generere unikke initialer".
C	The employee repository has five employees named Michael Laudrup, and another is created.	The sixth Michael Laudrup gets the initials miau

3.2. `GENERATEPROJECTNUMBER()`

Hvert projekt har et unikt projektnummer, der består af årstallet, hvor projektet er iværksat, samt et løbenummer. Det løbenummer vil blive genereret ud fra antallet af projekter i det pågældende år. Feks. vil første projekt i året 2023 have projektnummeret 23001, det næste 23002 osv. i stigende rækkefølge. JUnit-implementeringen af denne test kan findes i `ProjectTest.java`.

Listing 3: `generateProjectNumber()` kildekode med execution paths

```
1 public static String generateProjectNumber(Calendar date) {
2
3     int year = DateHelper.twoDigitYearFromDate(date);
4     int num = 0;
5
6     // Find the highest incremental number for the current year
7     for (String projectNumber : ProjectRepository.getInstance().all().keySet()) { // 1
8         if (projectNumber.startsWith(String.format("%02d", year))) { // 2
9             num = Integer.parseInt(projectNumber.substring(2)); // 3
10        }
11    }
12
13    int nextProjectNumber = (year * 1000) + num + 1;
14
15    if (nextProjectNumber < 10000) { // 4
16        return "0" + nextProjectNumber; // 5
17    }
18
19    return "" + nextProjectNumber; // 6
20 }
```

Tabel 4: Execution paths i generateProjectNumber()

Execution path	Input set	Input property
1 (not empty), 2 (true), 3, 4 (true), 5, 6	A	There are projects from the current year, and the final two digits of the current year are less than 10
1 (not empty), 2 (true), 3, 4 (false), 6	B	There are projects from the current year, and the final two digits of the current year are greater than 10
1 (not empty), 2 (false), 4 (true), 5, 6	C	There is a project from a different year than the current, and the final two digits of the current year are less than 10
1 (not empty), 2 (false), 4 (false), 6	D	There is a project from a different year than the current, and the final two digits of the current year are greater than 10
1 (empty), 4 (true), 5, 6	E	There are no projects and the final two digits of the current year are less than 10
1 (empty), 4 (false), 6	F	There are no projects and the final two digits of the current year are greater than 10

Tabel 5: Input sæt i generateProjectNumber()

Input set	Input data	Expected result
A	The current year is 2002 and there are two projects from the same year with project numbers 02001 and 02002	A new project with project number 02003
B	The current year is 2023 and there are two projects from the same year with project numbers 23001 and 23002	A new project with project number 23003
C	The current year is 2001 and there is a project from 2022 with project number 22001	A new project with project number 01001
D	The current year is 2023 and there is a project from 2022 with project number 22001	A new project with project number 23001
E	The current year is 2001 and there are no projects	A new project with project number 01001
F	The current year is 2023 and there are no projects	A new project with project number 23001

3.3. CREATEPROJECTACTIVITY()

Der kan oprettes projektaktiviteter for et projekt objekt. For at oprette en projektaktivitet kræves det, at en medarbejder er logget ind, og at der gives en titel, start- og slutuge for projektaktivitet. Derudover er det kun projektledere, der kan oprette aktiviteter, såfremt at der er tildelt en projektleder til projektet. Herunder i 4 er et udsnit af Project.java inklusiv metoden der laves white box test af. White box testen udføres i praksis med JUnit med filen *CreateProjectActivityTest.java*.

Listing 4: createProjectActivity() kildekode med execution paths

```
1 public class Project implements ConvertibleToViewModelInterface {
2
3     private Employee projectLeader;
4     private List<ProjectActivity> activities = new ArrayList<ProjectActivity>();
5
6     public ProjectActivity createProjectActivity(String
7         ↪ title, String startWeek, String endWeek, Employee loggedInUser)
8         throws AlreadyExistsException, OperationNotAllowedException, InvalidPropertyException {
9
10         if (hasProjectLeader()) { // 1
11             if (!projectLeader.isSameAs(loggedInUser)) { // 2
12                 throw new OperationNotAllowedException("Kun
13                     ↪ projektlederen kan oprette en projekt aktivitet for dette projekt"); // 3
14             }
15         }
16
17         if (hasProjectActivity(title)) { // 4
18             throw new AlreadyExistsException("Projekt aktivitet findes allerede"); // 5
19         }
20
21         ProjectActivity activity = new ProjectActivity(title, startWeek, endWeek); // 6
22         this.activities.add(activity); // 7
23
24     return activity;
25 }
```

Tabel 6: Execution paths i createProjectActivity()

Execution path	Input set	Input property
1 (false), 4 (false), 6, 7	A	projectLeader is null, activities does not contain an activity with title title
1 (false), 4 (true), 5	B	projectLeader is null, activities contains an activity with title title
1 (true), 2 (false), 4 (false), 6, 7	C	projectLeader is the same object as loggedInUser, activities does not contain an activity with title title
1 (true), 2(false), 4 (true), 5	D	projectLeader is the same object as loggedInUser, activities contains an activity with title title
1 (true), 2(true), 3	E	projectLeader is another employee object than loggedInUser, activities does not contain an activity with title title

Tabel 7: Input sæt for `createProjectActivity()`

Input set	Input data	Expected result
A	<code>projectLeader</code> is null <code>activities</code> is empty <code>title</code> = "Planlægning" <code>startWeek</code> = "2101" <code>endWeek</code> = "2103" <code>loggedInUser</code> = {"mila"}	<code>activities</code> contains a project activity with <code>title</code> "Planlægning"
B	<code>projectLeader</code> is null <code>activities</code> = ["Planlægning"] <code>title</code> = "Planlægning" <code>startWeek</code> = "2101" <code>endWeek</code> = "2103" <code>loggedInUser</code> = {"mila"}	Exception with message "Projekt aktivitet findes allerede" is thrown
C	<code>projectLeader</code> = {"mila"} <code>activities</code> is empty <code>title</code> = "Planlægning" <code>startWeek</code> = "2101" <code>endWeek</code> = "2103" <code>loggedInUser</code> = {"mila"}	<code>activities</code> contains a project activity with <code>title</code> "Planlægning"
D	<code>projectLeader</code> = {"mila"} <code>activities</code> = ["Planlægning"] <code>title</code> = "Planlægning" <code>startWeek</code> = "2101" <code>endWeek</code> = "2103" <code>loggedInUser</code> = {"mila"}	Exception with message "Projekt aktivitet findes allerede" is thrown
E	<code>projectLeader</code> = {"brla"} <code>activities</code> is empty <code>title</code> = "Planlægning" <code>startWeek</code> = "2101" <code>endWeek</code> = "2103" <code>loggedInUser</code> = {"mila"}	Exception with message "Kun projektlederen kan oprette en projekt aktivitet for dette projekt" is thrown

3.4. `FINDWORKTImEREGISTRATIONBYID()`

For et givet projekt kan der oprettes projektaktiviteter, hvor en medarbejder kan angive mængden af arbejdstid han/hun har brugt på den pågældende aktivitet. Disse registreringer af arbejdstid bliver gemt med et id-nummer, som består af de positive naturlige tal og løber fra 1 og op. Dermed er det muligt at finde specifikke registreringer ud fra id-nummeret, og hertil benyttes metoden `findWorkTimeRegistrationById()`.

Såfremt at id nummeret man søger efter matcher det af en arbejdstidsregistrering, returneres den pågældende registrering. Hvis ikke der er et match, kastes en `NotFoundException`. Dette tester vi i filen `WorktimeRegistrationTest.java`.

Listing 5: findWorktimeRegistrationById() kildekode med execution paths

```
1 public WorktimeRegistration findWorktimeRegistrationById(int id) throws NotFoundException {
2
3     List<WorktimeRegistration> list = allWorktimeRegistrations(); // 1
4
5     for (WorktimeRegistration worktimeRegistration : list) { // 2
6         if (worktimeRegistration.getId().equals(id)) { // 3
7
8             return worktimeRegistration; // 4
9         }
10    }
11
12    throw new NotFoundException("Ukendt tidsregistrering"); // 5
13
14 }
```

Tabel 8: Execution paths i findWorktimeRegistrationById()

Execution path	Input set	Input property
1 (empty), 2 (empty), 5	A	List containing work time registrations is empty, Arbitrary id number is set: id = 1
1 (not empty), 2 (not empty), 3 (false), 5	B	List containing work time registrations has 1 element, Input id does not match a work time registration id
1 (not empty), 2 (not empty), 3 (true), 4	C	List containing work time registrations has 5 elements, Input id matches a work time registration id

Tabel 9: Input sæt for findWorktimeRegistrationById()

Input set	Input data	Expected result
A	loggedInUser = {"mila"} id = 1 list is empty	Exception with message "Ukendt tidsregistrering" is thrown
B	title of project activity = "Test" loggedInUser = {"mila"} "mila" registers work time = 5.0 id = 2 list has 1 element	Exception with message "Ukendt tidsregistrering" is thrown
C	title of project activity = "Test" loggedInUser = {"mila"} "mila" registers work time = 5.0 "mila" registers work time = 2.5 "mila" registers work time = 4 "mila" registers work time = 12.5 "mila" registers work time = 20 id = 4 list has 5 element	list contains work time registration with matching id worktimeRegistration with work time = 12.5 is returned

4 CODE COVERAGE

Følgende tabeller er dannet fra coverage-programmets output.
Som det fremgår af Tabeller 10 til 16, er vores coverage på 100 procent, som det forventes ved god praksis af TDD.
Målet var først en kende lavere (omtrent 91 procent), hovedsageligt på grund af komplekse if-udtryk, hvor en eller flere

Tabel 10: Coverage for applikationslaget

Element	Cov (instructions)	Cov (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TaskFusion	100%	100%	0	10	0	16	0	9	0	1
DateServer	100%	n/a	0	2	0	5	0	2	0	1
Total	100%	100%	0	12	0	21	0	11	0	2

Tabel 11: Coverage for domænelaget

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ReportPDFGenerator	100%	0	14	0	164	0	5	0	1	
Project	100%	100%	0	43	0	74	0	24	0	1
Employee	100%	100%	0	28	0	53	0	15	0	1
ProjectActivity	100%	100%	0	18	0	36	0	12	0	1
Activity	100%	100%	0	12	0	21	0	4	0	1
Report	100%	100%	0	14	0	27	0	12	0	1
WorktimeRegistration	100%	n/a	0	7	0	13	0	7	0	1
RegularActivity	100%	n/a	0	3	0	5	0	3	0	1
Total	100%	100%	0	139	0	393	0	82	0	8

Tabel 12: Coverage for exceptions

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
NotFoundException	100%	n/a	0	1	0	2	0	1	0	1
AlreadyExistsException	100%	n/a	0	1	0	2	0	1	0	1
InvalidProceedException	100%	n/a	0	1	0	2	0	1	0	1
OperationNotAllowedException	100%	n/a	0	1	0	2	0	1	0	1
ExhaustedOptionsException	100%	n/a	0	1	0	2	0	1	0	1
Total	100%	n/a	0	5	0	10	0	5	0	5

Tabel 13: Coverage for facader

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProjectFacade	100%	100%	0	24	0	71	0	21	0	1
EmployeeFacade	100%	100%	0	11	0	23	0	9	0	1
Total	100%	100%	0	35	0	94	0	30	0	2

Tabel 14: Coverage for hjælperklasser

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PrintHelper	100%	100%	0	7	0	13	0	4	0	1
DateHelper	100%	100%	0	6	0	10	0	4	0	1
Total	100%	100%	0	13	0	23	0	8	0	2

Tabel 15: Coverage for persistency-laget

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Seeder	100%	100%	0	9	0	45	0	7	0	1
ProjectRepository	100%	100%	0	16	0	33	0	10	0	1
EmployeeRepository	100%	100%	0	15	0	35	0	10	0	1
Total	100%	100%	0	40	0	113	0	27	0	3

Tabel 16: Coverage for ViewModels

Element	Coverage (instructions)	Coverage (branches)	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProjectViewModel	100%	100%	0	6	0	22	0	3	0	1
ReportViewModel	100%	100%	0	3	0	17	0	1	0	1
ActivityViewModel	100%	100%	0	3	0	13	0	2	0	1
WorktimeRegistrationViewModel	100%	100%	0	3	0	12	0	2	0	1
RegularActivityViewModel	100%	100%	0	3	0	11	0	2	0	1
EmployeeViewModel	100%	100%	0	3	0	11	0	2	0	1
ViewModel	100%	n/a	0	1	0	1	0	1	0	1
Total	100%	100%	0	22	0	87	0	13	0	7

delbranches ikke var dækket af de cucumber-features, vi havde skrevet. F.eks. giver `if (jegErSulten() && jegHarSpist())` fire mulige delbranches: sand (begge er sande), falsk (den første er sand, anden ikke), falsk (omvendt af tidligere), og falsk (ingen af dem er sande). Af den mængde var vi interesseret i færre tilfælde end dem alle, hvorfor coverage faldt fra 100 procent. Dette blev løst i nogle tilfælde ved refactoring (særligt ved grelle if-udtryk) eller ved tilføjelse af nye scenarier, hvis man havde overset et relevant scenarie. De fleste blev løst ved tilføjelser af cucumber-scenarier.

En sidste nævneværdig årsag er future-proofing, særligt i `generateProjectNumber`, hvor `lastNum` sættes til `num`, hvis `num` var større end `lastNum`. Dette var gjort i starten af projektet med henblik på at gøre koden mindre afhængig af den samling, der eventuelt skulle indeholde alle projekter. I sidste ende blev projekter lagret i et `Map`, der tilfældigvis hasher projekt-numrene, så de eksisterer i mappens `keySet` i stigende rækkefølge indenfor samme år, hvorfor tjekket til sidst blev fjernet.

Hele præsentations-laget er ikke udviklet test-drevent, derfor er hele `cli` mappen ikke inkluderet i code-coverage rapporten. Ved at ekskludere hele laget fra code-coverage rapporten har det været muligt at have bedre styr på validiteten af business-laget, og netop kunne opnå 100% dækning.

5 DESIGN BY CONTRACT

I dette afsnit er fire metoder udvalgt til Design by Contract. De tilhørende assertions af preconditions og postconditions benyttet kan også ses i koden.

Grundet kombinationen mellem cucumber tests og implementering af DbC vil flere assertions kun have 1 branch testet, hvilket vil give anledning til mindre code coverage, selvom alle scenarier bliver testet. Derfor er assertions blevet udkommenteret i kildekoden og kan altså stadig ses i alle de fire nævnte metoder. Tanker og idéer til hvordan dette kunne være undgået diskuteres i konklusionskapitlet.

5.1. GENERATEPROJECTNUMBER

Precondition:

$$date \neq null \quad (5.1)$$

Postcondition:

$$result = \{s | \exists p \in projectRepository.getProjectNumbers : s \in p \wedge p = previousLargestProjectNumber + 1\} \quad (5.2)$$

Med assertions ser det ud som følger:

Listing 6: generateProjectNumber() med assertions

```
1 public static String generateProjectNumber(Calendar date) {
2     assert date != null // Precondition
3     int year = DateHelper.twoDigitYearFromDate(date);
4     int num = 0;
5
6     // Find the highest incremental number for the current year
7     for (String projectNumber : ProjectRepository.getInstance().all().keySet()) {
8         if (projectNumber.startsWith(String.format("%02d", year))) {
9             num = Integer.parseInt(projectNumber.substring(2));
10        }
11    }
12
13    int previousProjectNumber = (year * 100) + num;
14    int nextProjectNumber = (year * 1000) + num + 1;
15    assert nextProjectNumber == previousProjectNumber + 1; // Postcondition
16
17    if (nextProjectNumber < 10000) {
18        return "0" + nextProjectNumber;
19    }
20
21    return "" + nextProjectNumber;
22 }
23 }
```

5.2. CREATEINITIALS()

Precondition:

$$n\text{EmployeesWithName}(\text{employee}) < \text{maxInitials}(\text{employee}) \quad (5.3)$$

Postcondition:

$$\{s \mid \exists e \in \text{employeeRepository} : s \in e.\text{getInitials}()\}$$
 (5.4)

Hvor $\text{maxInitials}(\text{employee})$ er n vælg 2, hvor n er længden af efternavnet, og duplikater fjernes i generationen af kombinationer. I "Laudrup" (længde 7, $\binom{7}{2}=21$) går "lu", "au", "up" igen to gange, hvorfor der fås 18 og ikke de forventede 21. Der forklarer hvorfor white-box testens input er konstrueret som det er.

Med asserts ser det ud som følger

Listing 7: createInitials() med assertions

```
1 private void createInitials() throws ExhaustedOptionsException {
2     EmployeeRepository employeeRepo = EmployeeRepository.getInstance();
3     assert true; // Precondition
4
5     for (int l1 = 0; l1 < lastName.length(); l1++) {
6         for (int l2 = l1 + 1; l2 < lastName.length(); l2++) {
7             String init = firstName.substring(0, 2)
8                 + lastName.substring(l1, Math.min(l1 + 1, lastName.length()))
9                 + lastName.substring(l2, Math.min(l2 + 1, lastName.length()));
10
11             init = init.toLowerCase();
12             if (!employeeRepo.initialsExist(init)) {
13                 this.initials = init;
14                 assert this.initials == init; // Postcondition
15                 return;
16             }
17         }
18     }
19
20     throw new ExhaustedOptionsException("Kunne ikke generere unikke initialer");
21 }
```

Implicit garanteres det, der står i createInitials' precondition gennem funktionens kørsel, hvorfor der assertes true som precondition. I virkeligheden er der to postconditions: I tilfælde af at preconditionen holder, sættes Employee-instansens initials-felt til init, og i tilfælde af at den overtrædes, kastes en ExhaustedOptionsException.

5.3. CREATEPROJECTACTIVITY()

Precondition:

$$\text{title} \neq \text{null} \wedge \text{startWeek} \neq \text{null} \wedge \text{endWeek} \neq \text{null} \wedge \text{loggedInUser} \neq \text{null} \wedge \text{activities} \neq \text{null} \quad (5.5)$$

Postcondition:

$$\begin{aligned} & \text{activity} \in \text{activities} \wedge \\ & \neg \text{project@pre.hasProjectActivity}(\text{title}) \wedge \\ & (\text{hasProjectLeader}() \rightarrow \text{projectLeader.isSameAs}(\text{loggedInUser})) \end{aligned} \quad (5.6)$$

Listing 8: `createProjectActivity()` kildekode med assertions

```
1 public ProjectActivity createProjectActivity(String
   ↳ title, String startWeek, String endWeek, Employee loggedInUser)
2     throws AlreadyExistsException, OperationNotAllowedException, InvalidPropertyException {
3
4     assert title != null && startWeek !=
       ↳ null && endWeek != null && loggedInUser != null && activities != null; // Precondition
5
6     if (hasProjectLeader()) {
7         if (!projectLeader.isSameAs(loggedInUser)) {
8             throw new OperationNotAllowedException("Kun
               ↳ projektlederen kan oprette en projekt aktivitet for dette projekt");
9         }
10    }
11
12    if (hasProjectActivity(title)) {
13        throw new AlreadyExistsException("Projekt aktivitet findes allerede");
14    }
15
16    ProjectActivity activity = new ProjectActivity(title, startWeek, endWeek);
17    this.activities.add(activity);
18
19    /**
20     * NOTE: This post condition does not check for
21     * !project@pre.hasProjectActivity(title)
22     */
23    assert (
24        activities.contains(activity) &&
25        (!hasProjectLeader() || projectLeader.isSameAs(loggedInUser) )
26    ); // Postcondition
27
28    return activity;
29 }
```

5.4. FINDWORKTImEREGISTRATIONBYID()

Precondition:

I dette tilfælde er vores precondition `true`, da vores postconditions korrigerer for fejl, så alle tilfælde af inputs fejlhåndteres.

Postcondition:

$$\exists i : (result = i \iff \exists worktimeRegistration.getId() = i \vee result = NotFoundException) \quad (5.7)$$

Det skal nævnes, at postconditionen vil være exceptionen `NotFoundException`, i tilfælde af at leddet før \vee i ligningen over ikke er sand.

Listing 9: `findWorktimeRegistrationById()` kildekode med assertions

```
1 public WorktimeRegistration findWorktimeRegistrationById(int id) throws NotFoundException {
2
3     assert true // Precondition
4
5     List<WorktimeRegistration> list = allWorktimeRegistrations();
6
7     for (WorktimeRegistration worktimeRegistration : list) {
8         if (worktimeRegistration.getId().equals(id)) {
9             assert list.stream().anyMatch(x -> x.getId() == id) // Postcondition
10
11             return worktimeRegistration;
12         }
13     }
14
15     throw new NotFoundException("Ukendt tidsregistrering");
16
17 }
```

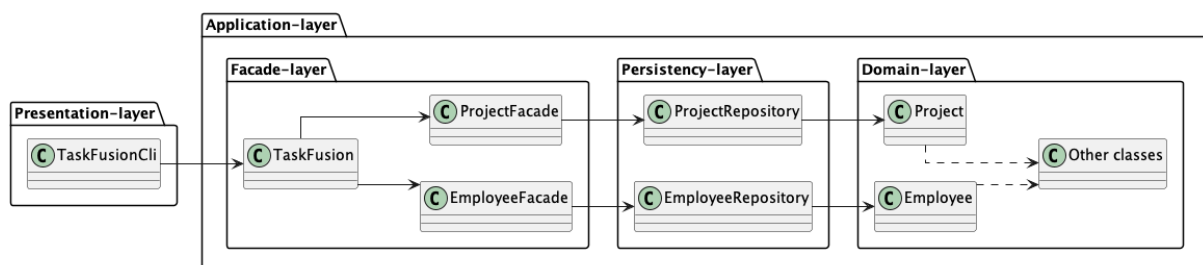
6 DESIGN MØNSTRE

Der har været fokus på at adskille præsentationslag, businesslag, og persistence-lag og flere design mønstre er anvendt for at opnå et let læseligt, overskueligt og lavt koblet system.

6.1. PROGRAM-LAG

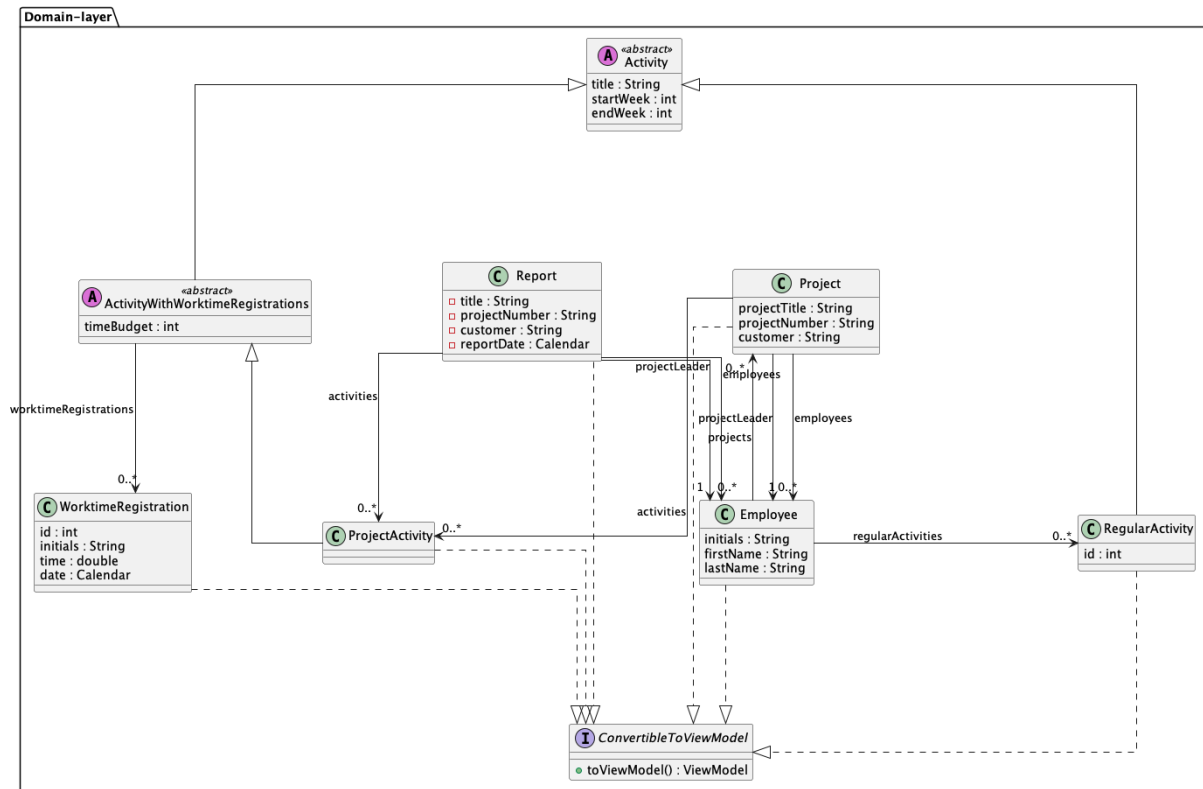
Et fuldt klassediagram over program-laget kan ses i appendix A. Et simplificeret klassediagram herunder i 2 viser relationer mellem klasserne i TaskFusion programmet. TaskFusion fungerer som hovedklasse og er primært ansvarlig for brugergodkendelser. Et facade mønster (eng: Facade pattern) er implementeret med `EmployeeFacade.java` og `ProjectFacade.java` for at opnå lav kobling. F.eks. bliver systemet, der behandler employees, tilgået gennem `EmployeeFacade.java` således at klasserne, bestående af bl.a. `EmployeeRepository.java`, `Employee.java`, `RegularActivity.java`, ikke skal kaldes af klienten på forskellig vis. I stedet kan klienten tilgå alle de nødvendige egenskaber igennem facaden kun. Sammen med `EmployeeFacade` og `ProjectFacade` eksponere `TaskFusion` de offentlige metoder der skal kunne tilgås i programmet. På den måde kan vi frihed til at ændre alle metoder i de resterende program-lag, uden det vil påvirke eventuelle brugergrænseflader. Alle metoder i facade klasserne kan i virkeligheden ligge i `TaskFusion` klassen, men ved at opdele metoderne i passende separate klasser, kan vi bedre vedligeholde og udbygge programmet. Programmets domæne lag består af instantierbare objektklasser og er ansvarlig for *Business-logic*. Persistency-laget indeholder *Repositories*, der er ansvarlige for kommunikation med en lagringsløsning. Selvom programmet ikke har et database lag på nuværende tidspunkt, vil det være let at bygge flere lagringsløsninger på senere, ved kun at skulle modificere persistency klasserne.

Figur 2: Klassediagram over program-lag



Vi ønsker desuden aldrig at eksponere programmets klasser udenfor program-laget. Derfor implementere alle instantierbare klasser i domæne-laget *ConvertibleToViewModel*-interfacet, vist herunder i 3. Dette interface kræver at klasserne kan eksporteres til en tilsvarende visningsklasse til brug i præsentations-lag. På den måde kan vi undgå at brugere kan kalde metoder fra domæne-laget, og dermed sikre de forudsætninger de enkelte metoder måtte have.

Figur 3: Domæne-laget



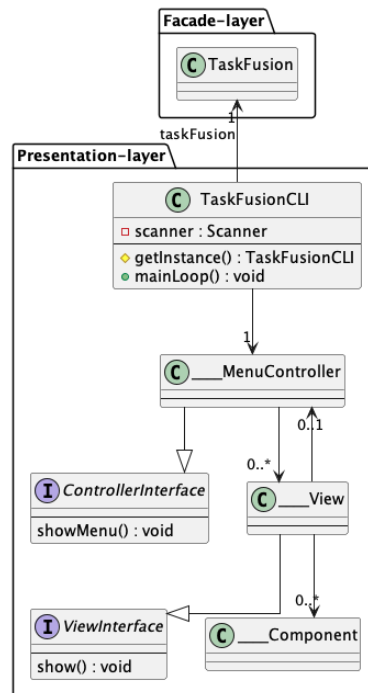
Et andet design mønster der er taget i brug, er singleton design mønstret. Feks. haves et opbevaringssted for alle medarbejdere kaldet *EmployeeRepository*. Der ønskes kun én instans af dette objekt, da idéen er at tilgå og opbevare medarbejderne via. ét objekt. Hvis flere instanser af dette objekt skulle forekomme, er det ikke sikret, at brugeren kan tilgå alle medarbejderne fra den ene instans, da medarbejdere kan eksistere i de andre instanser, hvorfor objektet skal være en singleton. Af samme grunde som *EmployeeRepository* er en singleton, er *ProjectRepository* ligeså.

6.2. PRÆSENTATIONS-LAGET

Som det blev nævnt i indledningen af dette kapitel, har der været fokus på adskillelse af de forskellige lag i arkitekturen af programmet. Måden hvorpå business-logikken er blevet separeret fra præsentationslaget i programmet er ved brug af Model-View-Controller design mønstret. Dette er gjort ved at have 'controller' klasser og 'view' klasser, som har funktionen at modtage forespørgsler fra brugeren og fremvise det efterspurgte data uden at have noget business logik i sig. Disse kan ses under mapperne *controllers* og *views*.

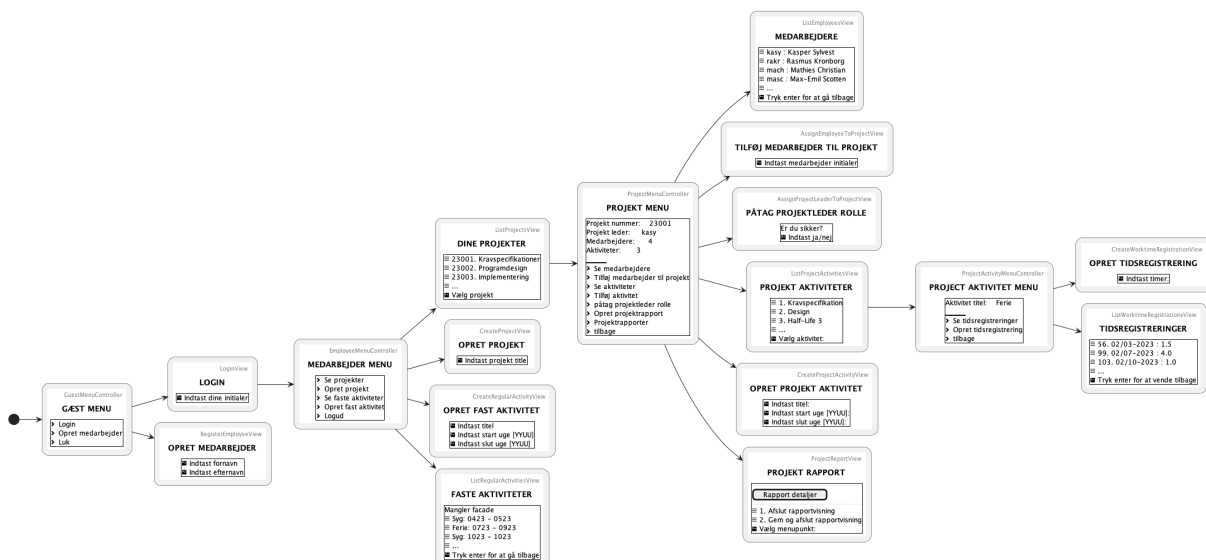
6.2.1. CLI klassediagram *TaskFusionCLI* er hovedklassen når *TaskFusion* programmet skal benyttes igennem en CLI brugergrænseflade. Når grænsefladen skal interagere med *TaskFusion* programmet, foregår al kommunikation imellem *Facade*-laget. CLI'en er fundamentalt opbygget med tanke på genbrugelighed, og simplicitet. Det er opnået ved at tage udgangspunkt i en *MenuController*, hvorfra *View*'s bruges til at skrive information brugeren efterspørger til konsollen. Hvordan et *View* ser ud for brugeren, afhænger af hvilke variabler og objekter der gives ved konstruktion af *View*'et. *Component*-klasser er en form for hjælpe klasser, der igennem `public static` metoder, tilbyder universelle komponenter til brug i grænsefladen.

Figur 4: Klassediagram over præsentations-laget



6.2.2. CLI brugergrænsefladen Da CLI grænsefladen til TaskFusion er opbygget af *MenuController*'re og *View*'s, ender vi da også med et netværk af mulige veje brugeren kan gå. For at få et overblik over TaskFusion, er herunder i 5 et *flow*-diagram over menuer og sider i CLI grænsefladen. Diagrammet er ikke tiltænkt at være noteringsmæssig korrekt, men har fungeret effektivt som mockup i udviklingsfasen, og stadig til at skabe et overblik over programmet. En større version af figuren kan ses i appendix B.

Figur 5: Flowdiagram over CLI brugergrænsefladen



7 SOLID PRINCIPER

Nogle SOLID principper er blevet benyttet og vil herunder blive gennemgået med kodeeksempler.

7.1. SINGLE RESPONSIBILITY PRINCIPLE

I forbindelse med at designe programmet efter mønstre som nævnt i forrige kapitel, har det også været naturligt at opdele klasser til kun af have et enkelt formål. Det ses som eksempel på *Persistency* klasserne `ProjectRepository.java` og `EmployeeRepository`, der henholdsvis har ansvaret for at håndtere de primære hovedobjekt-klasser; *Employee's* og *Project's*.

Derudover er alle klassers metoder som udgangspunkt lavet til at kun at have et enkelt formål. Herunder i 10 er to eksempler på metoder fra `Employee.java`, hvis eneste formål er at validere det givne argument.

Listing 10: Single responsibility metoder

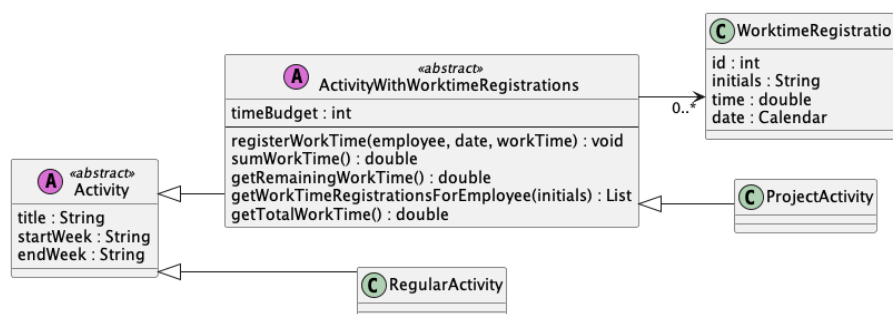
```
1 private String validateFirstName(String firstName) throws InvalidPropertyException {
2     if (firstName.length() < 2) {
3         throw new InvalidPropertyException("Fornavn mangler");
4     }
5     return firstName;
6 }
7
8 private String validateLastName(String lastName) throws InvalidPropertyException {
9     if (lastName.length() < 2) {
10        throw new InvalidPropertyException("Efternavn mangler");
11    }
12    return lastName;
13 }
14
```

Der er dog enkelte metoder, der kunne optimeres i forhold til *Single responsibility* princippet. Her kan nævnes *createProjectActivity()* som beskrevet i *White box test* 3.3 kapitlet. Denne metode er både ansvarlig for at acceptere om en medarbejder kan oprette en projekt aktivitet, og derefter også at oprette den. Her kunne man refaktorere til, at tjekket om medarbejderen må oprette en aktivitet, lå udenfor metoden.

7.2. OPEN/CLOSED PRINCIPLE

Det lyder fra opgaveformuleringen, at der skal være mulighed for at opdele projekter op i aktiviteter, og at der skal være mulighed for at lave faste aktiviteter til registrering af bl.a. ferie og kurser, som ikke er knyttet til projekter. Et simpelt eksempel hvor *Open/Closed*-princippet er blevet benyttet til at imødekomme førnævnte er at lave en abstrakt klasse kaldet *activity*, som vist herunder i figur 6.

Figur 6: Klassediagram af activity klasserne



Vi skelner i programmet imellem en fast aktivitet og en projekt aktivitet, henholdsvis `RegularActivity` og `ProjectActivity`. Begge typer har behov for egenskaberne *title*, *startWeek* og *endWeek*. For at sikre at kunne overholde de forventninger der er til disse egenskaber, er disse i en abstrakt klasse *Activity*, som underklasserne nedarver fra. På den måde er *Activity* lukket for ændringer men åben for udvidelser.

Der er desuden også det argument, at *single responsibility princippet* bliver fulgt, da *Activity* klassen kun har én grund til at blive ændret. Den grund vil indbefatte en ændring af definitionen, af hvad en aktivitet er, f.eks. en ændring af dets felter.

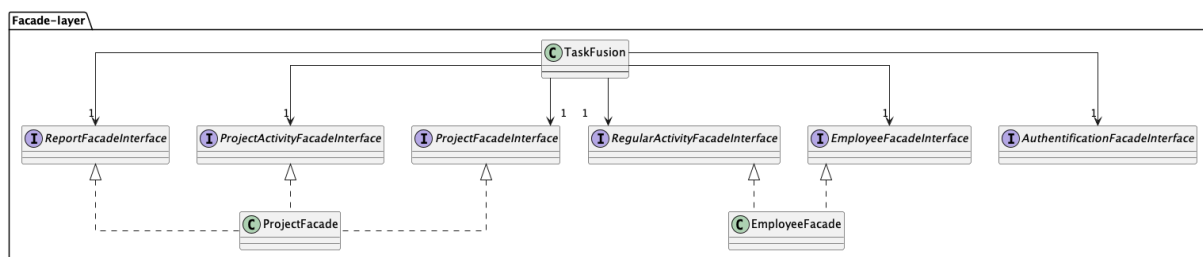
7.3. LISKOV SUBSTITUTION PRINCIPLE

Derudover skal projekt aktiviteter kunne håndtere tidsregistreringer. Istedet for at fylde `ProjectActivity`-klassen med metoder, nedarves igen fra en abstrakt klasse `ActivityWithWorktimeRegistration`, se igen klassediagram 6. På denne måde kan metoder relateret til tidsregistrering ikke kaldes fra `RegularActivity`, og det er samtidig muligt at udvide senere med aktiviteter der kan have tidsregistreringer eller ej.

7.4. INTERFACE SEGREGATION PRINCIPLE

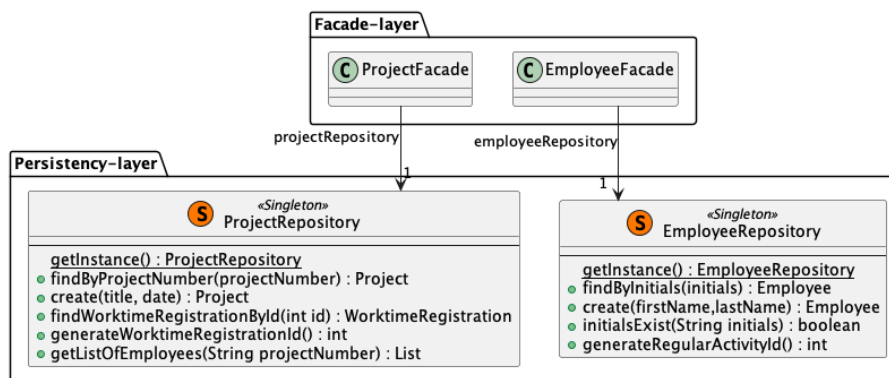
Facade-laget bruges som mellemlag mellem præsentations- og program-lag, og på den måde afkobler forskellige dele af programmet. Ser vi nærmere på facade-laget, har vi også her forsøgt at reducere sammenkoblingen mellem høj- og lavniveaus komponenter, ved at afhænge af abstraktioner istedet for konkrete implementeringer. Det er gjort ved brug af *interfaces* der afkobler `TaskFusion` fra henholdsvis `EmployeeFacade` og `ProjectFacade`, som vist herunder i figur 14. Et udvidet klassediagram over facade-laget kan ses i appendix C.

Figur 7: Klassediagram af facade-laget



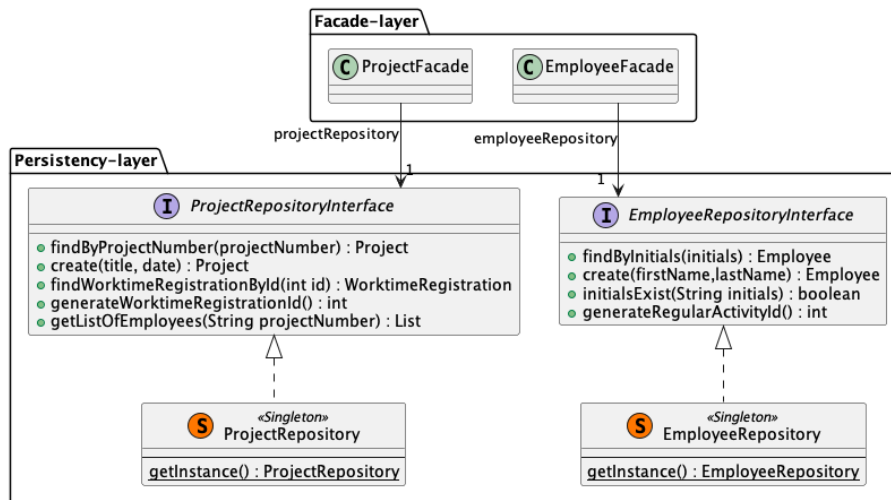
Ser vi istedet på koblingen mellem *facade-* og *persistency-laget*, kan vi her forbedre programmet i forhold til *Interface segregation principle*. Den nuværende implementation ser ud som vist herunder i figur 8

Figur 8: Klassediagram af nuværende forbindelse mellem facade- og persistency-laget



Som programmet er nu, afhænger `EmployeeFacade` og `ProjectFacade` direkte af implementerede metoder i henholdsvis `EmployeeRepository` og `ProjectRepository`. Istedet kunne vi indføre et `EmployeeRepositoryInterface` og et `ProjectRepositoryInterface`, hvilke facade klasserne så kunne afhænge af. En sådan implementering vil så se ud som illustreret herunder i figur 9

Figur 9: Klassediagram af forbindelse mellem facade- og persistency-laget ved brug af interfaces

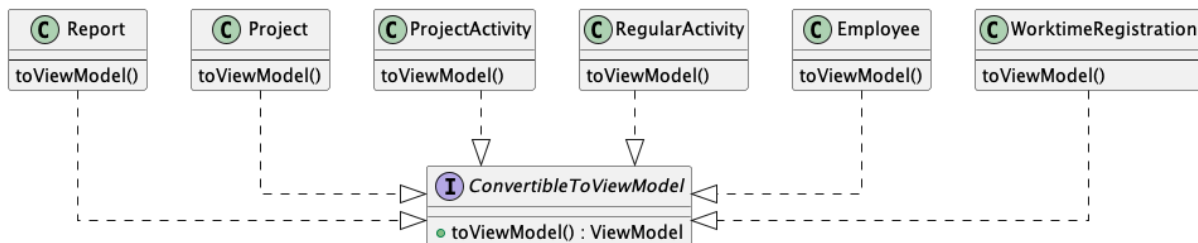


7.5. DEPENDENCY INVERSION PRINCIPLE

Med et fortsat fokus på facade-laget og figur 14, implementere `EmployeeFacade` og `ProjectFacade` flere interfaces hver. På den måde kan mindre og fokuserede interfaces gøre det nemmere for facade klasserne at implementere de metoder, de har behov for.

Vi kan også fremhæve et andet eksempel på opfyldelse af *Dependency inversion principle*. `ConvertibleToViewModelInterface` er et interface der adskiller konverteringslogikken fra domæneobjekterne, se klassediagram herunder i figur 10. Klasser som `Project` og `Employee` implementerer dette interface og kan dermed konverteres til en `ViewModel` til brug i præsentationslaget. På denne måde har vi en simpel og fokuseret kontrakt for domæneobjekter, der kan udvide deres funktionalitet uden at ændre deres kerneadfærd.

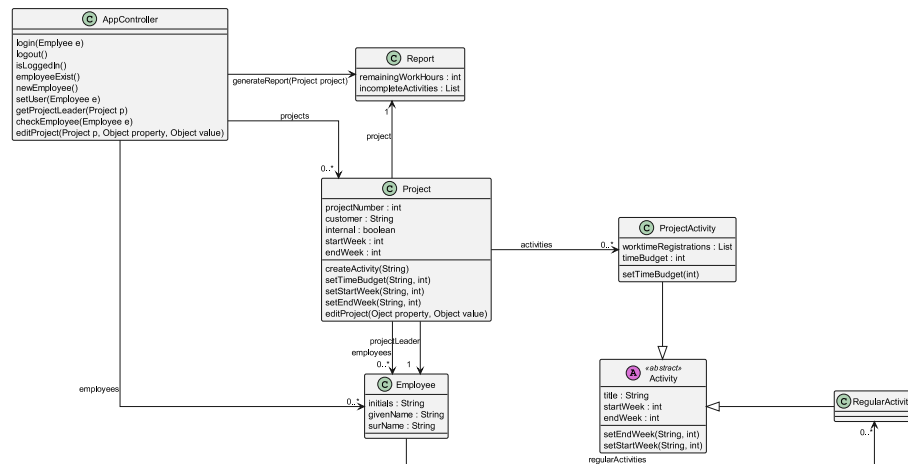
Figur 10: Klassediagram af `ConvertibleToViewModelInterface` implementeringer



8 KONKLUSION

I første version af rapport 1, lagde vi ud med et design af programmet som vist herunder i figur 11. Her var en `AppController` ansvarlig for stort set alt, vi ikke kunne pakke ind i de øvrige modeller.

Figur 11: Klassesdiagram fra rapport 1



I takt med introduktion til *Good Design*, *Design patterns* og *SOLID-principles* i undervisningen, har vi refaktoreret programmet for at forbedre designet. Det endelige design kan ses i appendix A. Hovedsageligt er det `AppController`, der er trukket ud af *domain*-laget, og opdelt til et *facade*- og *persistence*-lag. Med den indsigt vi har nu omkring program design, vil vi i projekter fremover være i bedre stand til at tænke god design ind fra starten.

Ser vi på funktionen `createRegularActivity()` i `Project` klassen, som vist i appendix D. Her kan vi se metoden er ansvarlig for at verificere de argumenter der bliver givet, istedet for at gå ud fra, de allerede er verificeret. Det resulterer i, at en hel del kode eksekveres, inden de check nås, og *Exceptions* da skal sendes hele vejen tilbage i forløbet. Men væsentligt er også, at metoden dermed har ansvar for mange forhold, hvilket kunne sepereres smartere. I forbindelse med introduktion af *Design-By-Contract* begyndte vi blive opmærksomme på dette. Hvorfra mange af vores metoder er refaktoreret, til bedre at have styr på hvad de kan forvente og hvad der kan forventes af dem. Udover programlaget, er tilføjet et præsentationslag til håndtering af en brugergrænseflade. I den forbindelse oprettede vi *ViewModels* for alle klasser i domæne-laget. For at opnå et bedre design, kan vi opdele disse domæne-klasser yderligere med en tilhørende *Controller*-klasse, der kan have ansvar for disse argument valideringer, inden de bliver givet til selve modellen. Det vil desuden fuldende et *MVC*-design mønster i domæne-laget, og gøre udbygning og vedligehold nemmere.

FIGURER

1	Mappestrukturen i Java projektet TaskFusion	2
2	Klassediagram over program-laget	13
3	Domæne-laget	14
4	Klassediagram over præsenterings-laget	15
5	Flowdiagram over CLI brugergrænsefladen	15
6	Klassediagram af <i>aktivitet</i> klasserne	16
7	Klassediagram af facade-laget	17
8	Klassediagram af nuværende forbindelse mellem facade- og persistency-laget	17
9	Klassediagram af forbindelse mellem facade- og persistency-laget ved brug af interfaces	18
10	Klassediagram af <i>ConvertibleToViewModelInterface</i> implementeringer	18
11	Klassediagram fra rapport 1	19
12	Klassediagram over program laget	23
13	Flowdiagram over CLI brugergrænsefladen	24
14	Detaljeret klassediagram af facade-laget	25
15	Sekvensdiagram: Opret projektaktivitet	27

TABELLER

1	Oversigt over forfatterskaber i projektet	1
2	Execution paths i createInitials()	3
3	Input sæt i createInitials()	4
4	Execution paths i generateProjectNumber()	5
5	Input sæt i generateProjectNumber()	5
6	Execution paths i createProjectActivity()	6
7	Input sæt for createProjectActivity()	7
8	Execution paths i findWorktimeRegistrationById()	8
9	Input sæt for findWorktimeRegistrationById()	8
10	Coverage for applikationslaget	9
11	Coverage for domænelaget	9
12	Coverage for exceptions	9
13	Coverage for facader	9
14	Coverage for hjælperklasser	9
15	Coverage for persistency-laget	9
16	Coverage for ViewModels	9

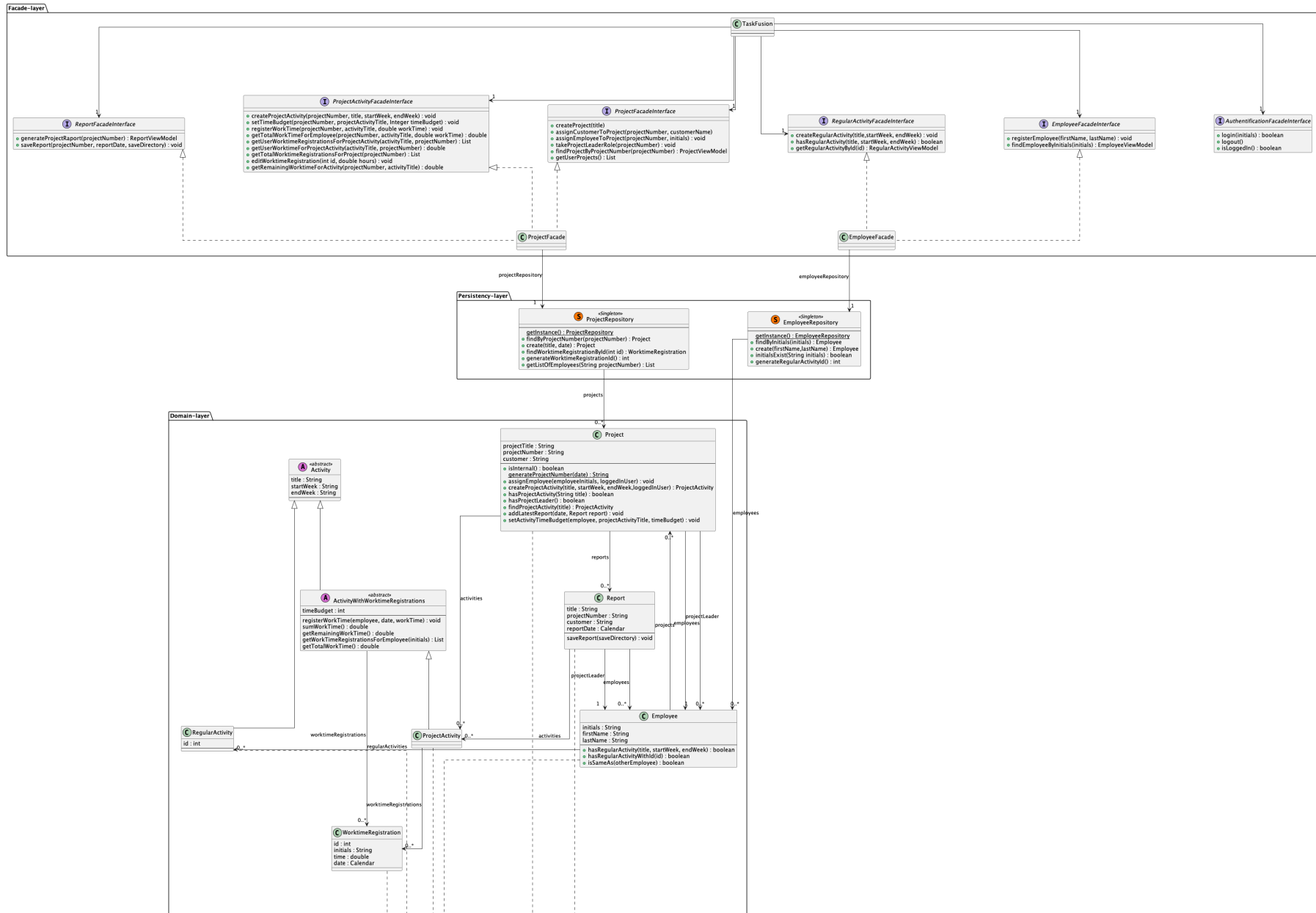
LISTINGS

1	Kommentar til createInitials() kildekode	3
2	createInitials() kildekode med execution paths	3
3	generateProjectNumber() kildekode med execution paths	4
4	createProjectActivity() kildekode med execution paths	6
5	findWorktimeRegistrationById() kildekode med execution paths	8
6	generateProjectNumber() med assertions	10
7	createInitials() med assertions	11
8	createProjectActivity() kildekode med assertions	12
9	findWorktimeRegistrationById() kildekode med assertions	13
10	Single responsibility metoder	16

Appendiks

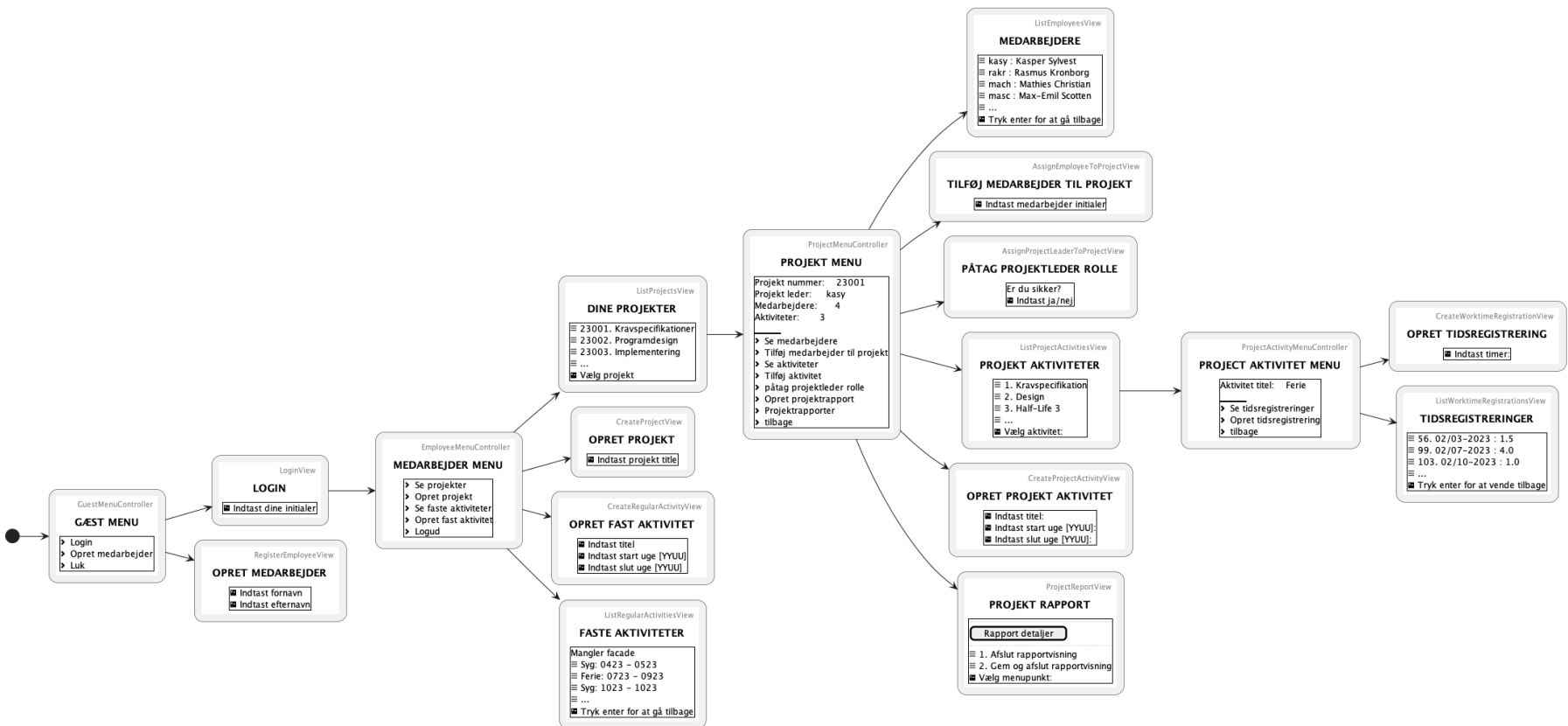
A KLASSEDIAGRAM OVER PROGRAM LAGET

Figur 12: Klassediagram over program laget



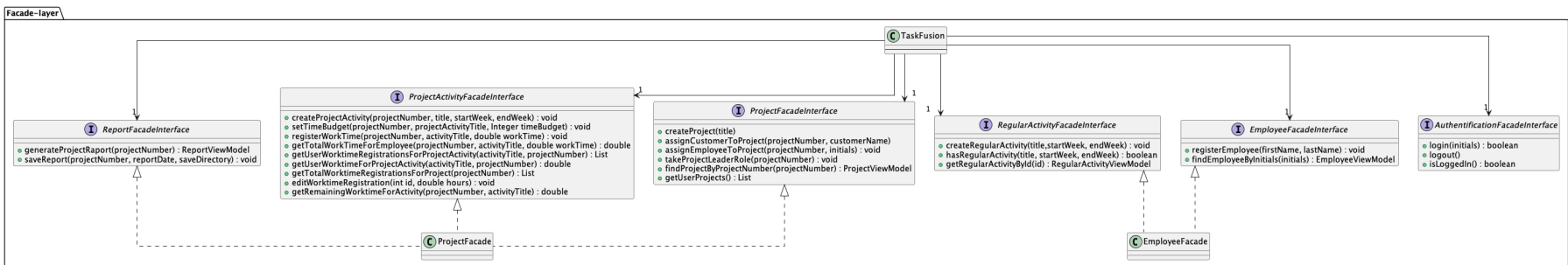
B KLASSEDIAGRAM OVER PRÆSENTATIONS-LAGET

Figur 13: Flowdiagram over CLI brugergrænsefladen



C KLASSEDIAGRAM OVER FACADE-LAGET

Figur 14: Detaljeret klassediagram af facade-laget



D SEKVENSDIAGRAM: CREATEPROJECTACTIVITY()

Figur 15: Sekvensdiagram: Opret projektaktivitet

