

Agile Software Engineering

Team 23: Almo Daved, Joram Ruitenschild, Graeham Blank

Intro

In dit document is terug te vinden hoe team 23 bestaande uit: Almo Daved, Joram Ruitenschild en Graeham Blank het vak agile hebben doorlopen. De toegewezen rails projecten voor dit project zijn Rails 6 & Rails 7.

We zullen toelichten:

* Wat onze aanpak strategie is geweest
* Wat we hebben waargenomen in de Rails projecten
* De Testresultaten
* Het herschrijven van de code (welke refactorings wel en niet mogelijk waren en waarom)
* Het koppelen van de rails projecten (welke methode en waarom)

**Belangrijk**

Doordat Almo veel problemen had met zijn laptop zijn de meeste commits via Joram gedaan.

De refactorings en de Unit Test staan in een ander project om de koppeling niet in de weg te lopen.

Nadere toelichting staat in dit document

Aanpak strategie

Om het vak Agile voldoende af te ronden moet het volgende worden opgeleverd:

* Onderbouwd kwaliteitsrapport van enkele pagina’s op basis van de quality assignments 1 t/m 5 die in de les gemaakt zijn.
* Gedocumenteerde refactorings. Motiveer in enkele pagina’s waar je in het systeem, welke bad-smells.
* hebt gevonden en welke refactoring-techniek je daarop hebt toegepast. Criterium: uit elke hoofdgroep dien je 1 refactoring te hebben toegepast (dus: 6 in totaal).
* Kort reflectieverslag met daarin een oordeel over de kwaliteit van het opgeleverde systeem, en per teamlid een beschrijving van de werkzaamheden, tijdverantwoording en evaluatie.

Deze opdrachten hebben we onderling onderverdeeld: Almo is verantwoordelijk voor de documentatie en de refactorings, Joram is verantwoordelijk voor het koppelen van de systemen en Graeham is verantwoordelijk geweest voor het bouwen van de unit Test’s.

Voorheen hadden we dit anders ingedeeld.

We hadden eerst besloten, dat Joram zich bezig hield met het koppelen van de projecten. Almo en Graeham zouden zich met de Refactorings bezig houden.

Deze aanpak bleek uiteindelijk niet praktisch. We zaten elkaar continu in de weg en te vragen in welke Class er momenteel gewerkt wordt.

Nadat we besloten hadden dit anders te gaan doen heeft Joram de koppeling van rails 6 naar rails 7 werkend gekregen en had Almo al twee refactorings. Graeham heeft toen de taak van het schrijven van Unit Test’s op zich genomen.

Desondanks moesten er duidelijke afspraken worden gemaakt zoals: ”Wie, wanneer in welke Class werkt”, “ het zovaak mogelijk pushen en committen naar git” en “het continu testen van de rails projecten op fouten (handmatig).

Kwaliteitsrapport

Assignment 1 Quality Items for a software product

De kwaliteitseisen, die wij gekozen hebben zijn:

* Lines of Code
* Density of Comments
* Object Oriend Programming (Use of Classes)
* Executable statements
* Number of Exported types

Deze hebben wij gekozen, omdat deze eisen gecombineerd de basis zijn van het programmeren. Een goeie software programmeur kan: ingewikkelde codes zo kort mogelijk coderen. Dit is niet altijd handig, omdat het ook goed leesbaar moet zijn (Dus kort en leesbaar).

Ook is commentaar heel erg belangrijk. Bij ingewikkelde methodes is het erg belangrijk om uit te leggen, wat er gebeurd en zo nodig waarom dit is gedaan. Dit voornamelijk, omdat software engineers in het bedrijfsleven vrijwel nooit alleen werken.

Verder zaken als Object georiënteerd programmeren en het goed gebruik maken van de Classes zorgt voor goed gestructureerde code, die herbruikbaar is, overzichtelijk is en makkelijk uit te breiden is.

De resultaten van de projecten zijn op de volgende pagina te vinden

Assignment 1 Results

Rails 6 Table with quality items and their score:

|  |  |  |
| --- | --- | --- |
| Nr | Quality item | good/average/bad |
| 1 | Lines of code | Good |
| 2 | Density of Comments | bad |
| 3 | Object Oriented Programming (Use of classes) | bad |
| 4 | Executable statements | bad |
| 5 | Number of Exported types | Good |

Rails 7 Table with quality items and their score:

|  |  |  |
| --- | --- | --- |
| Nr | Quality item | good/average/bad |
| 1 | Lines of code | Good |
| 2 | Code comments | bad |
| 3 | Object Oriented Programming (Use of classes) | bad |
| 4 | Executable statements | bad |
| 5 | Number of Exported types | Good |

Assignment 2 Size Metrics

De bedoeling van assignment 2 is het vergelijken op kwaliteit van Rails 6 vs Rails 7. Dit hebben we gedaan met SourceMonitor en RefactorIT. De resultaten en toelichtingen zijn hieronder te vinden.

|  |  |  |
| --- | --- | --- |
| Metric | SourceMonitor | RefactorIT |
| Number of files (NOF) | 24 |  |
| Number of Classes (NOC) | 54 |  |
| Lines of code (LOC) |  | 3862 |
| Comment lines (cLOC) |  | 557 |
| Number of executable lines of code (eLOC) |  |  |

Measurement results Rails 7:

|  |  |  |
| --- | --- | --- |
| Metric | SourceMonitor | RefactorIT |
| Number of files (NOF) | 80 |  |
| Number of Classes (NOC) | 122 |  |
| Lines of code (LOC) | 13947 |  |
| Comment lines (cLOC) | 4123 |  |
| Number of executable lines of code (eLOC) |  |  |

Assignment 3 Usefull or Not?

In de 3e opdracht werd onze kennis getest op het gebied van programmeren. Het is de bedoeling dat je aangeeft of het gebruiken van bijvoorbeeld commentaar met minder dan 10% lege regels handig is in een software programma of niet.

We hadden de keuze om te kiezen tussen:

* Not usefull
* Might be usefull
* en Very usefull

Bij usefull en very usefull hebben we onze mening beargumenteerd.

Asignment 3 Content Table

|  |  |
| --- | --- |
| Lay-out comments | Rating |
| 1. Physical line length > 80 characters | Not usefull |
| 1. Function/class/struct/interface white space < 10.0% | Might not be usefull |
| 1. Function comment content less than 10.0%w | Not usefull |
| 1. File white space < 10.0% | Might be usefull |
| 1. File comment content < 10.0% | Not usefull |
| 1. TAB character has been identified | Not usefull |
| 1. Class/Struct comments are < 10.0% | Might be usefull |
| 1. Function/Class Blank Line content less < 10.0% | Not usefull |
| 1. File Blank Line content < 10.0% | Might be usefull |
| 1. A line containing just a semicolon | Not usefull |
| 1. The closing brace is not on a standalone line | Very Usefull |

|  |  |
| --- | --- |
| Naming comments | Rating |
| 1. Class name not proper cased | Not usefull |
| 1. Function name length > 32 characters | Not usefull |
| 1. A specified key string is missing in a file | Not usefull |

|  |  |
| --- | --- |
| Program code comments | Rating |
| 1. Assignment "=" within "if" statement | Might be usefull |
| 1. Assignment "=" within "while" statement | Might be usefull |
| 1. Pre-decrement operator "--" identified | Might be usefull |
| 1. Pre-increment operator "++" identified | Might be usefull |
| 1. "switch" statement does not have a "default" | Might be usefull |
| 1. "case" conditions do not equal "break" | Not usefull |
| 1. if, else, for or while not bound by scope | Not usefull |
| 1. "?" ternary operator identified | Very usefull |
| 1. Number of function return points > 1 | Might be usefull |
| 1. Class specification contains public data | Might be usefull |
| 1. Class specification contains protected data | Might not be usefull |
| 1. Exception Handling "try"- "catch" has been identified | Very usefull |
| 1. Keyword "continue" has been identified | Might be usefull |
| 1. Keyword "break" identified outside a "switch" structure | Might be usefull |

|  |  |
| --- | --- |
| Program structure comments | Rating |
| 1. Function eLOC > maximum 200 eLOC | Not usefull |
| 1. Cyclomatic complexity > 10 | Might be usefull |
| 1. Number of function parameters > 6 | Not usefull |
| 1. Number of class/struct methods exceeds specified maximum 100 | Might be usefull |
| 1. Depth of the Inheritance Tree exceeds the maximum 4 | Not usefull |
| 1. Number of derived classes exceeds the specified maximum 10 | Not usefull |
| 1. Multiple inheritance has been identified | Very usefull |
| 1. Function lLOC <= 0, non-operational function | Might be usefull |
| 1. Function appears to have null or blank parameters | Very usefull |

Assignment 3: Explanation

1. Physical line length > 80 characters

Hierdoor is de code slecht leesbaar

1. Function comment content less than 10.0%w

Complexe functies kunnen door weinig comment slecht worden begrepen.

Dit is onhandig bij het upgraden of uitbreiden van een programma

6. TAB character has been identified

Hierdoor kunnen shortcuts zoals sout (system out print line) niet worden gebruikt

8. Function/Class Blank Line content less < 10.0%

Bij minder dan <10% blanke regels is het moeilijker te zien waar classes/ fucnties stoppen.

De structuur van het programma is hierdoor moeilijk af te leiden.

10. A line containing just a semicolon

Een statement etc. moet altijd worden afgesloten met een semicolon, syntax fouten binnen de

code opsporen wordt hierdoor lastig.

12. Class name not proper cased

De bedoeling van de klasse is hierdoor niet in een oog opslag duidelijk. Dit is vooral storend

Wanneer er in teams wordt gewerkt.

13. Function name length > 32 characters

Functie namen moeten werkwoorden zijn of een korte combinatie woorden. Boven 32 characters

maakt het de code slecht leesbaar. Uitleg moet in comment lines worden verwerkt.

14. A specified key string is missing in a file

19. "switch" statement does not have a "default"

Onverwachte waardes worden door de switch niet opgevangen

21. if, else, for or while not bound by scope

Een scope wordt gedefineerd door de {} , als er geen scope aanwezig is, dan zullen er errors

optreden.

29. Function eLOC > maximum 200 eLOC

Een functie neemt te veel taken opzich. Moet er iets worden aangepast of gewijzigd worden,dan

kan dit grote gevolgen hebben op de werking van het systeem.

31. Number of function parameters > 6

In de meeste gevallen zullen de parameters, dan op een andere regel komen. Hierdoor is het

slecht leesbaar en zal bij het gebruiken van dit object telkens, deze 6 parameters worden

opgegeven

33. Depth of the Inheritance Tree exceeds the maximum 4

Code wordt slecht overzichtelijk. Het is lasting te achterhalen, welke child classes wat doen.

(worden er functions overriden? Etc.)

34. Number of derived classes exceeds the specified maximum 10

Code wordt slecht overzichtelijk. Het is lasting te achterhalen, welke child classes wat doen.

(worden er functions overriden? Etc.)

Rails 6 to Rails 7 (Bridge)

Koppeling tussen rails 6 en rails 7

Om de koppeling tot stand te laten komen hebben we besloten om rails 6: spoor 1 t/m 8 Nederland te noemen en rails 7: spoor 1 t/m 8 Duitsland. Intern tellen we de sporen door van 1 tot 16. Spoor 1 in Duitsland staat gelijk aan spoor 9 in het gehele systeem.

Voor de koppeling tussen rails 6 en rails 7 gebruiken we een database. Wanneer een trein internationaal zal reizen reist hij tot spoor acht waar de passagiers zullen moeten overstappen op de trein naar het andere land. We plaatsen dan een entry in de database met het aantal passagiers en hun eindbestemming in de andere rails applicatie.

Beide rails applicaties maken een connectie met de mysql database op het moment dat hij gestart wordt. Iedere seconde kijken ze in de database of er nieuwe entries op hun van toepassing zijn. Wanneer dat het geval is verwijderen we deze entries en plaatsen we ze in de applicatie. Dit werkt beide kanten op.

De volgende classes zijn bijna het zelfde voor beide applicaties:

Listen.java – De classe die wordt aangeroepen bij het initialiseren van de applicatie. Hij extend een Thread. In deze thread wordt iedere seconde een SELECT query uitgevoerd (door Queries.java) om de entries op te halen van toepassing voor deze rails applicatie. Als er nieuwe entries zijn toegevoegd

Unit Tests

Inleiding

De unit test die gedocumenteerd staan, zijn voortgekomen uit de rails 6 applicatie

Hier wordt getest op het functioneren van de applicatie. Dit hebben we niet specifiek voor de refactorings gedaan, omdat er vrij complexe refactorings zijn toegepast. We hebben er dus voor gekozen om rond de applicatie kritieke methoden te testen.

* TestAddReis   
  *Test op de functie AddReis, deze bepaald of een reis daadwerkelijk wordt toegevoegd.*

*Door te controleren of de lijst reizen een reis van het Type Reis bevat is direct te zien of dit goed is gebeurd.*

* Test getAlleActReizen

*Test of het lokaliseren van de rails cab goed verloopt.*

* Test TreinPositie (Boven positive)

*Het was hier de bedoeling om het start getal (dit moet de test verwachten) in te voeren van de Modulos en het rest getal echter konden wij deze berekening niet vinden op het internet. Dit is gedaan om int rxoff een waarde tussen 177 en 183 te geven om ook de If-statement te testen.*

*Wel hebben we een opzet voor deze test gemaakt een waarde van rx=800 ingevuld die als modulo getal 100 moet verwachten.*

# TestAddReis

@Test

public void testAddReis(){

LinkedList<Reis> reizen = new LinkedList<Reis>();

Halte start = new Halte(id);

Halte eind = new Halte(id);

start.id = 4;

eind.id = 6;

Reis reis = new Reis(5,start, eind,1,tijd);

reizen.addFirst(reis);

Boolean expects = true;

Boolean result = reizen.contains(reis);

assertEquals(expects, result);

}

Test Boven positieTest Get Alle Actieve Reizen

@Test

public void testPositieBoven(){

CabAnimation cabAnime = new CabAnimation(31, rb, tb);

rx = 800;

rxoff = (int) (Math.round(rx) % 350);

int tempRxOff = rxoff;

// checkWissel op wisselId is 178 t/m 182 wordt nooit uitgevoerd + wisselId bestaat niet..

if (rxoff > 177 && rxoff < 183) {

rxnr = (int) Math.round((rx - 180) / 350);

cabAnime.checkWissel(tempRxOff);

}

cabAnime.wOffset(1);

rx = (int) (81 + ((1092 / 37.5) \* positie));

ry = 26 + (int)pl;

int expects = 100;

int result = rxoff;

System.out.println(rxoff);

assertEquals(expects, result);

}

@Test

public static ArrayList<Reis> TestGetActReizen(){

LinkedList<Reis> reizen = new LinkedList<Reis>();

ArrayList<Reis> ar =new ArrayList<Reis>();

Halte start = new Halte();

Halte eind = new Halte();

eind.id = 6;

start.id = 2;

boolean actReizen = true;

for(Reis r:reizen){

if(r.getStatus()>0&&r.getStatus()<3)

ar.add(r);

Boolean expects = true;

Boolean result = ar.contains(r);

System.out.println("printje " + r);

assertEquals(expects, result);

}

return ar;

}

Refactoring

Inleiding

Voordat we een refactoring strategie hebben toegepast, zijn we eerste de code gaan analyseren. We hebben toen besloten dat de code eerst voor een deel opgeruimd moest worden. Aangezien we nog geen koppeling tussen Rails 6 en 7 hebben gemaakt en geen functionaliteiten wilde breken zijn we eerste begonnen met het renamen van een aantal Classes. Vervolgens zijn we naar bad smells gaan zoeken die overeen komen met de refactorings uit de lijst. Zoals eerder aangegeven zijn gedurende dit vak de taken binnen ons team veranderd. Aangezien Almo al bezig was met de refactorings in Rails 6 hebben we besloten alleen Rails 6 te refactoren.

Waarom Refactoren?

De eerste bad smells die naar voren kwamen in Rails 6 zijn:

* Big Class
* Excessive use of literals

Een veelgebruikte structuur in de programmeer wereld is het ModelVieldController (MVC) model.

Hiermee wordt voornamelijk voorkomen dat een Class meerdere verschillende verantwoordelijkheden krijgt.

Een voorbeeld hier van is de Class: “Trein” op regel 195 wordt een void method aangemaakt die zorgt voor het toevoegen van een reis, terwijl de verantwoordelijk van de Klasse gaat over het besturen van de trein.

Excessive use of literals (EUL) heeft dus meerdere gevolgen. Zodra een naam van een klasse slecht is aangegeven kan het ten koste gaan van de structuur van de applicatie.

EUL is in Rails 6 in de gehele applicatie te vinden, waardoor functies, parameters en fields slecht zijn af te leiden. **Refactoring is nodig om code een richtlijn te geven waaraan ze moeten voldoen**

De refactorings die we uiteindelijk hebben toegepast zijn Extract Class, deze refacotring is niet helemaal gelukt. Dit wordt in dit hoofdstuk nader uitgelegd. Voor een Extract Class Refactoring, zijn in dit geval echter meerdere refactorings nodig o.a. : Introduce method, rename, Inline Method, move Method etc.

Extract Class I

Extract class is een refactoring waarbij een nieuwe class wordt aangemaakt door delen van een al bestaande klasse over te brengen naar een nieuwe class. Bij deze refactoring kan de bad smell: Large Class, Long Method, EUL en complex conditionals voorkomen.

De grootste Klasse met de meeste verwijzingen in Rails 6 is CabAnimation.java

Deze klasse heeft als hoofdverantwoordelijkheid het animeren van de railcab.

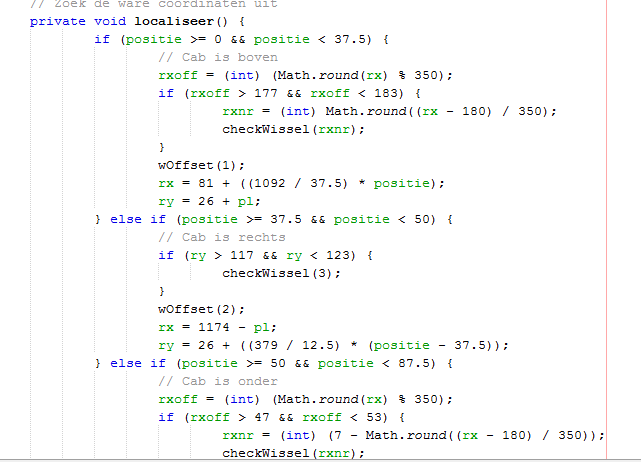
Hiernaast heeft deze klasse andere verantwoordelijkheden. Om alle inhoud in kaart te brengen hebben we besloten om de refactoring Introduce method toe te passen.

Voor het animeren worden complexe berekeningen uitgevoerd (het rond rijden van het treintje). Hierin is vrij veel logica voor nodig.

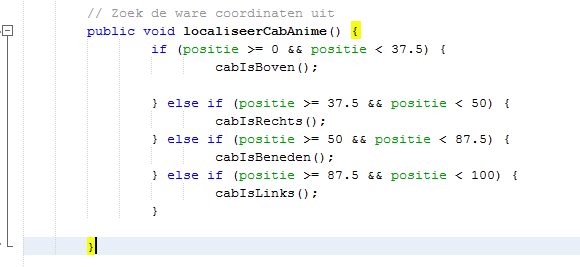
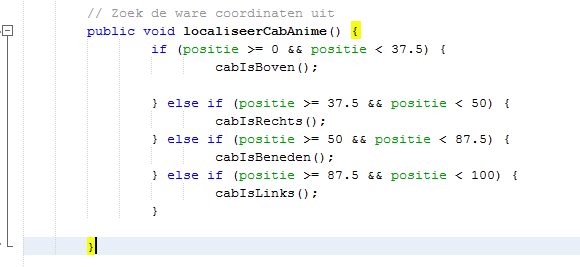
Door de slechte benaming in de klasse en onnodig commentaar te verhelpen is er een Introduce method toegepast binnen de lokaliseer methode.

Binnen deze methode stonden lange berekening om de trein coördinaten te berekenen.

**Before**



**After**

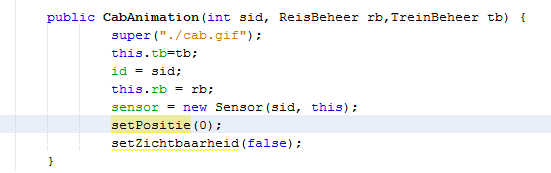


# Om te kijken hoe sterk de objecten aan elkaar gekoppeld waren hebben we een move method refactoring toegepast. Dit hebben we gedaan, door de methode lokaliseerCabAnime aan te roepen vanuit een andere klasse. Aller eerste zijn de nodige fields uit de Class CabAnimation.Java gekopieerd naar de nieuwe Class.

Na het schrijven van de constructor “new CabAnimation(sid, tb,rb)” van het object

“CabAnimation(int sid,TreinBeheer tb, ReisBeheer rb)” is al snel opgevallen, dat er meerdere objecten nodig zijn om de methode LocaliseerCabAnime() aan te roepen.

Deze conclusie is afgeleid uit o.a. het “Sensor” Obect en de “setPositie(int)” methode



Desondanks hebben we door het gebruiken van fields te methode vanuit een andere klasse werkend te krijgen, helaas is dit niet gelukt. Met een default constructor werkt de rails applicatie zonder errors, maar is het treintje niet zichtbaar.

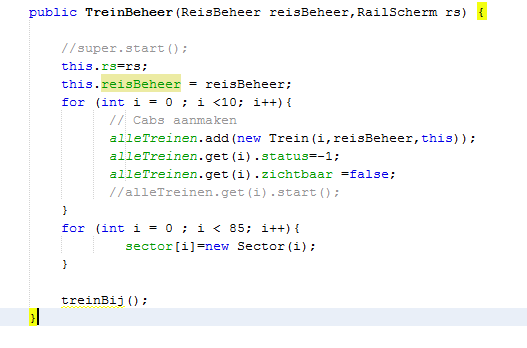
Nadat we een poging hebben gedaan voor een move method in cabAnimation is er gelijk voor gekozen om een poging om de Extract Class method toe te passen aangezien er meerdere objecten een sterk verband met elkaar hebben in dezelfde Klasse.

Om te zien waar de verbanden tussen de objecten liggen hebben er niet voor gekozen om de klasse in 1x te kopiëren en de errors van de compiler 1 voor 1 weg te werken. In plaats daarvan hebben we een nieuwe Klasse “CabAnimationTemp” gemaakt.

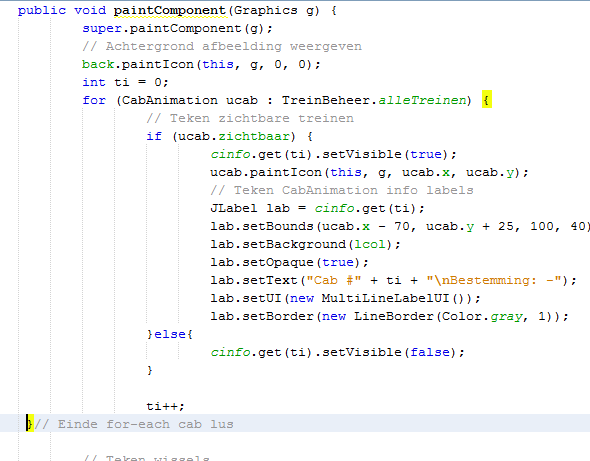
Voor elke methode is op volgorde:

1. De methode zelf gekopieerd naar CabAnimationTemp
2. Afhankelijke objecten aangemaakt
3. De nodige velds toegevoegt.

Door deze strategie toe te passen kwamen we er al vrij snel achter dat de Klasse TreinBeheer (voorheen Trein) het object TreinBeheer bevat als parameter ook een object van Railscherm.



Voor het weergeven van de trein op het scherm is de onderstaande for loop in de Jpanel van de Railscherm.java Class erg belangrijk.

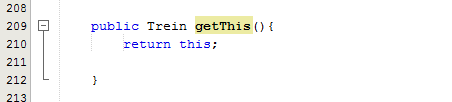


Doordat Railscherm sterk afhankelijk is van het CabAnimation object was het voor ons lastig uit te vogelen, waar de verbanden liggen. Een oplossing zou zijn een nieuwe Jpanel aan te maken en door de treinen heen te loopen. Wij hebben ervoor gekozen om dit niet te doen, omdat we dan technisch gezien functionaliteiten aan het na bouwen zijn in plaats van de code te refactoren.

# Extract Class II

Nadat een volledige Extract Class method niet mogelijk lijkt te zijn op CabAnimation.Java, is er gezocht naar andere potentiëlen Classes.

De Klasse “TimerHandler” in Klasse Trein.Java viel op, omdat er referenties aanwezig waren naar de methode getThis() in de Klasse Trein.Java



Het sleutelwoord “this” refereert naar de fields in Trein.Java.

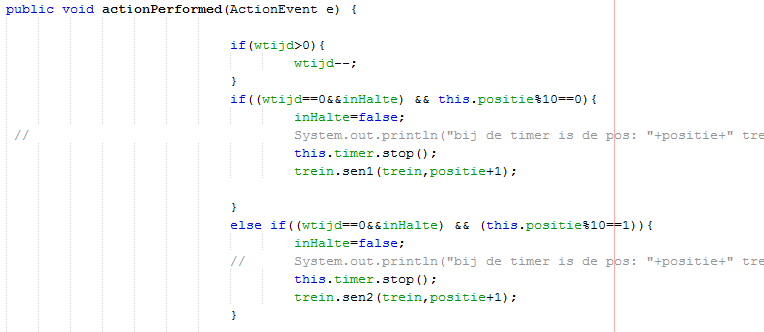
Voor de class Extract Class refactoring zijn de volgende stappen ondernomen:

1. Er is een new Class “TimerHandlerTemp.Java” aangemaakt
2. ActionListener is geimplementeerd in de nieuwe Klasse
3. De nodige velden zijn toegevoegd
4. Voeg een trein constructor Trein toe aan de nieuwe klasse om het sleutelwoord “this” te vervangen (Onderdeel van Inline Method)
5. Kopieer TimeHandler Class
6. InlineMethod Refactor getThis()
7. Vind alle referenties naar de oude TimeHandler Class en verwijs ze naar de nieuwe

**Before**



**After**



Conclusie

In Rails 6 zijn objecten sterk aan elkaar verbonden met zeer slechte naamgevingen, waardoor lastig te zien is welke verantwoordelijkheden elk object/functie heeft.

Door een Extract Class op de CabAnimation uit te voeren wisten we toch te achterhalen wat de meeste objecten en methodes doen.

De Bad smells die zich in Rails 6 bevinden zijn: Excessive use of Literals, large class, long method en duplicate code.

Deze hebben we verholpen door de volgende refactorings toe te passen:

* Extract Class
* Inline Method
* Introduce Method
* Move Method
* Rename

Het was lastig om de applicatie werkend te houden met het toepassen van de refactorings, daarom hebben we elke poging zo duidelijk mogelijk proberen te documenteren.