2016•2017
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef
Ontwikkeling van een BlueJ-getinte uitbreiding voor NetBeans

Promotor :
dr. Kris AERTS
ing. Leo RUTTEN

Stijn Boutsen
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2016•2017

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterproef

Ontwikkeling van een BlueJ-getinte uitbreiding voor NetBeans

Promotor :
dr. Kris AERTS
ing. Leo RUTTEN

Stijn Boutsen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Woord vooraf

Welkom bij het eindrapport van deze masterthesis. Deze proef is afgelegd ter afsluiting van mijn opleiding Academische Master Industriële Ingenieurswetenschappen, Elektronica-ICT.

Voor een student met weinig programmeerervaring is de overstap van BlueJ naar NetBeans moeilijk door het wegvallen van BlueJ's educatieve tools. Deze plugin zorgt ervoor dat de gebruiker makkelijker de overstap kan maken naar een gevorderd IDE.

Bij deze wil ik mijn externe promotor dr. Kris Aerts bedanken om dit project mogelijk te maken. Tevens wil ik hem bedanken voor zijn technische kennis en praktische hulp die hij aanbood wanneer nodig was. Tevens wil ik mijn interne promotor ing. Leo Rutten bedanken voor zijn ondersteuning en kennis van de evaluatieklasse. Dankzij hun deskundige begeleiding kon deze proef succesvol worden voltooid.

Hiernaast wil ik graag dr. Wim Deferme bedanken voor het bijbrengen van projectplanning en de taalvaardigheden nodig voor het schrijven van dit eindrapport.

Ten slotte wil ik graag mijn familie en vrienden bedanken voor hun steun en advies.

De broncode van dit project wordt online ter beschikking gesteld op github en is terug te vinden via onderstaande link.
https://github.com/Boutsman/UHasseltEDU

Inhoudsopgave

Lijst met tabellen

Lijst met figuren

Verklarende woordenlijst

API	Application Programming interface
IDE	Integrated Development Environment Geïntegreerde ontwikkelomgeving
OO	Object georiënteerd
Structureel programmeren	Programmeerconcept waarbij men een bepaalde structuur in de code brengt ter bevordering van de leesbaarheid, kwaliteit en ontwikkeltijd van een computerprogramma [1].
Procedureel programmeren	Programmeerconcept waarbij een programma wordt opgedeeld in methodes die direct uit te voeren zijn [2].
Widget	Een Window gadget is een GUI element verantwoordelijk voor interactie met de gebruiker [3].
Nodes	Presentatieobjecten voor een onderliggend datamodel [4].
Edges	Een presentatieobject voor een verbinding tussen een source en target node [5].

Abstract

BlueJ is een open-source IDE die het aanleren van object georiënteerd programmeren in JAVA vergemakkelijkt. Dit programma wordt gebruikt in 1BA maar is niet geschikt om grote applicaties te schrijven. Door het ontbreken van de educatieve tools van BlueJ hebben studenten moeilijkheden met de overgang naar een professionele programmeeromgeving zoals de NetBeans IDE.

Deze masterproef realiseert een plugin voor NetBeans met de belangrijkste educatieve kenmerken van BlueJ. Zowel het visueel klasse-schema als de objectenbench van BlueJ werden in NetBeans binnengebracht. Hierdoor kunnen objecten dynamisch worden aangemaakt en hun methodes uitgevoerd met de Java Reflection API.

Daarnaast is onderzocht hoe de bestaande evaluatie-extensie uit een oude masterproef naar NetBeans kan geconverteerd worden. Deze gebruikt CheckStyle om fouten op te sporen die indruisen tegen de opgelegde codeerstandaard. De mate waarin het programma zich aan de codeerafspraken houdt, wordt weergeven met een percentage.

Met deze aanpassingen kunnen studenten die overschakelen van BlueJ naar NetBeans de vertrouwde didactische aanpak blijven gebruiken.

Abstract in Engels

BlueJ is an open-source IDE that makes it easier to learn object oriented programming in Java. This program is being used in 1BA but isn't suitable for creating large applications. The absence of the educational tools provided by BlueJ make it hard for students to transition to a professional development environment like the NetBeans IDE.

This thesis develops an educational plug-in for the NetBeans IDE with the educational features of BlueJ. The class diagram and objectbench from BlueJ are implemented in NetBeans. This makes it possible to dynamically load objects and execute their methods using the Java Reflection API.

An existing evaluation extension from an existing master's thesis is converted. This uses the Java CheckStyle API to find faults against a defined coding standard. How much the program complies to the coding conventions is represented by a percentage.

With these adaptations, students can transition from BlueJ to NetBeans while using the familiar pedagogic approach.

1 Inleiding

1.1 Situering

In het 1e en 2e jaar in de gemeenschappelijke opleiding industrieel ingenieur van de KU Leuven en Universiteit Hasselt wordt de BlueJ IDE gebruikt in de inleidende Java-vakken om object georiënteerd programmeren aan te leren volgens het principe van objecten-eerst. BlueJ, afgebeeld in Fig. 1, is bedoeld om het leerproces van object-georiënteerd programmeren te vergemakkelijken. Hiervoor biedt het educatieve tools aan zoals visualisatie en interactie [6]. De visuele voorstelling van de klassen maakt de structuur van het programma en de samenhang tussen de klassen duidelijk. Hierdoor kan men ook bij grotere programma's het overzicht houden.

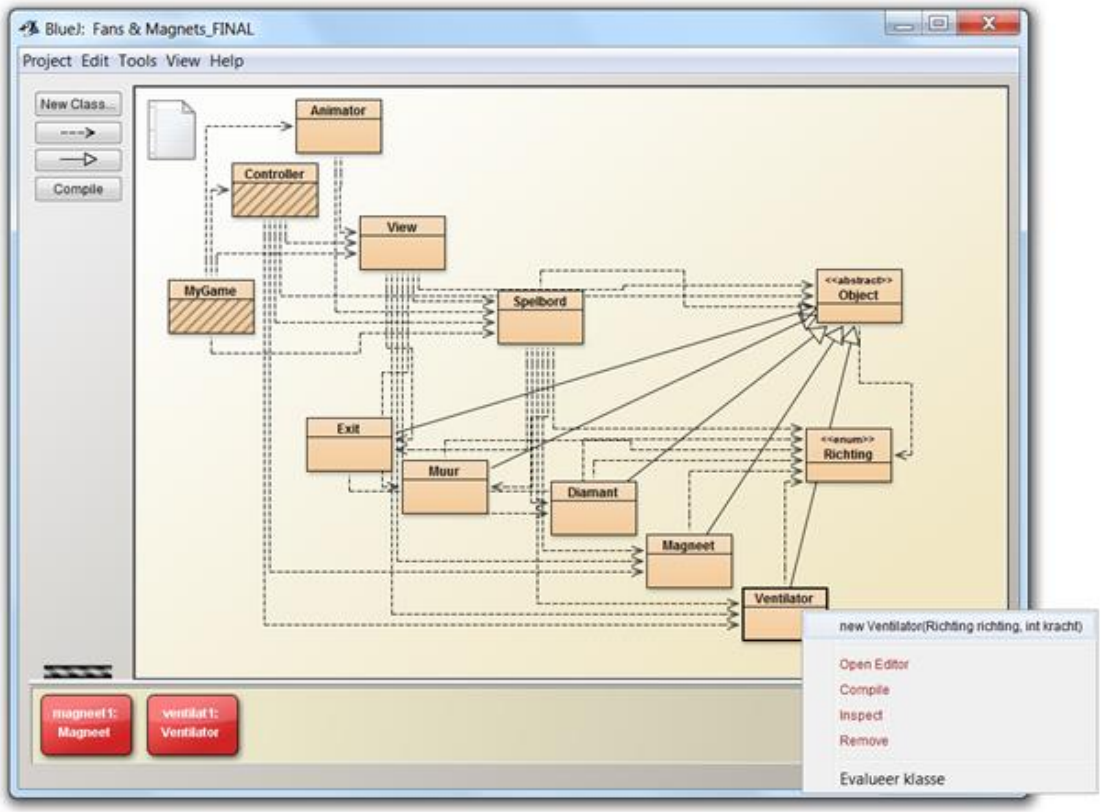

Fig. 1 - De BlueJ IDE

In latere jaren van de opleiding gaat men van deze didactische omgeving over naar de NetBeans IDE. Door het wegvallen van de visuele representatie verliezen studenten het overzicht van de structuur naarmate de programma's groter worden. Het doel van de masterproef is een plug-in te ontwikkelen die de visuele voorstelling en objectenbench van BlueJ binnenbrengt in NetBeans.

Bovendien is er in de gemeenschappelijke opleiding industrieel ingenieur van de KU Leuven en Universiteit Hasselt een BlueJ-extensie gebouwd die studenten helpt om zelf te evalueren of ze aan enkele basiscodeerregels voldoen. Door structureel te programmeren wordt een programma beter leesbaar en dus makkelijker onderhoudbaar. Hierdoor is het makkelijker voor derden om aan een programma bij te dragen. Een percentage weergeeft in welke mate aan deze codeerregels voldaan is. De opleiding wil deze zelf-evaluatieklasse ook in NetBeans gaan gebruiken.

1.2 Probleemstelling

BlueJ is een prima omgeving voor beginners, maar niet bruikbaar voor grootschaligere projecten of het werken met een GUI. Hiervoor worden professionelere programmeeromgevingen zoals NetBeans en Eclipse gebruikt die beschikken over een scene builder, code-aanvulling en gevorderde debugging tools. De overgang van een didactische tool naar een professionele programmeeromgeving is voor studenten overweldigend [7]. Dit geeft nood aan een software-extensie die deze tools in een gevorderde IDE integreert.

Grote software projecten worden niet door één enkele persoon geschreven maar door meerdere ontwikkelaars die samen werken en over het project moeten communiceren. Dit geeft nood aan een goede structuur van het ontwerp en de code. Door structureel te programmeren wordt code beter verstaanbaar. Met de herwerkte versie van deze evaluatieklasse kan de student zelf controleren of hij aan de codeerconventies voldoet.

1.3 Doelstellingen

Het doel is een module te maken voor NetBeans die de voordelen van BlueJ en de evaluatie-klasse binnenbrengt in NetBeans. De extensie moet de volgende functionaliteiten aanbieden.

- BlueJ-project importeren in NetBeans
- NetBeans-project exporteren naar BlueJ-project
- BlueJ visuele voorstelling implementeren in NetBeans
    - Pijlen trekken genereert code en omgekeerd
- Objecten-bench uit BlueJ implementeren
- Aanbieden van de evaluatieklasse in NetBeans

1.4 Materialen en methoden

De plug-in voor de NetBeans IDE zal ontwikkeld worden in Java met het NetBeans Platform. De IDE bevat alle componenten nodig voor de ontwikkeling van nieuwe modules. Door de modulaire werking van het NetBeans Platform is het gemakkelijk om nieuwe functies in het programma binnen te brengen.

Het klassediagram gebruikt de NetBeans Visual Library om bestanden en hun samenhang te visualiseren. De data nodig om het schema te tekenen zit in het package.bluej bestand. Voor een Java project dat niet in BlueJ gemaakt is zal dit bestand gegenereerd moeten worden. Het Java Reflection API reikt methodes aan om de relaties tussen de bestanden op te vragen.

De BlueJ objectenbench wordt ontwikkeld met de Reflection API. Hiermee worden objecten gemaakt van bestaande klassen, methodes uitgevoerd en parameters opgevraagd.

De evaluatie-klasse is een extensie voor BlueJ gemaakt door Tim Hermans en Raf Marcoen als masterthesis voor de opleiding academische master industriële wetenschappen aan de KHLim. Deze plug-in voert met behulp van CheckStyle controle uit op fouten tegen een opgelegde codeerstandaard. De code van dit programma zal geanalyseerd worden en herwerkt voor gebruik in NetBeans.

2 BlueJ

BlueJ is een open-source, educatief JAVA IDE ontwikkeld aan de Deakin University en University of Kent. Deze omgeving verschilt van andere IDE's daar ze de klassenstructuur grafisch weergeeft [6] en aangemaakte methodes met de objectenbench kunnen getest worden zonder gebruik te maken van een "main()"-methode.

Deze sectie bespreekt eigenschappen eigen aan BlueJ die nodig zijn om dezelfde tools in NetBeans binnen te brengen.

2.1 Didactische tools

BlueJ bied enkele didactische tools aan die het aanleren van object georiënteerd programmeren vergemakkelijken. Deze sectie geeft een overzicht van de functies die in de plug-in zullen geïmplementeerd worden.

2.1.1 Visualisatie

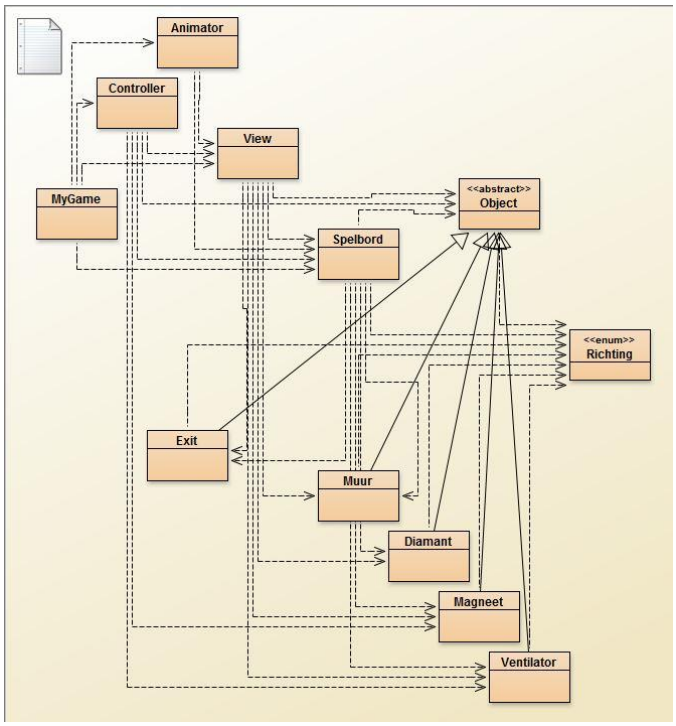

Fig. 2 - BlueJ grafisch venster

Het visueel voorstellen van klassen en hun onderlinge relaties, weergeven in Fig. 2, dwingt studenten om na te denken over de structuur van een OO-Java project [8]. Voor docenten is dit interessant om over te brengen hoe men een software-architectuur ontwikkelt. De student leert zo hoe hij een software project opbouwt door eerst het visueel schema te tekenen en pas daarna de functionaliteit te coderen.

De compositie (has-a relatie) wordt voorgesteld met onderbroken pijlen en erving (is-a relatie) met volle pijlen. Indien men in de code een klasse oproept wordt automatisch de relatie getekend. Dit is ook zo voor erving maar hier genereert het trekken van pijlen ook code in tegenstelling tot de has-a relatie.

2.1.2 Objectenbench

Fig. 3 - BlueJ objecten bench

Een java programma heeft een methode “main(String[] args)” die het programma start. De objectenbench in BlueJ, geïllustreerd in Fig. 3, maakt het mogelijk om de constructor van een individuele klasse uit te voeren zonder dat de gebruiker een main-functie moet schrijven. Hierdoor wordt er een object van de klasse gemaakt waarvan de methodes kunnen uitgevoerd worden, zoals in Fig. 4. Indien de methode of constructor parameters nodig heeft wordt er een JFrame getoond zoals in Fig. 5. Returnwaarden worden ook in een JFrame afgebeeld zoals in Fig. 6. De user kan zo zijn klasse uittesten zonder dat hij hier code voor moet schrijven of het hele programma moet compileren.

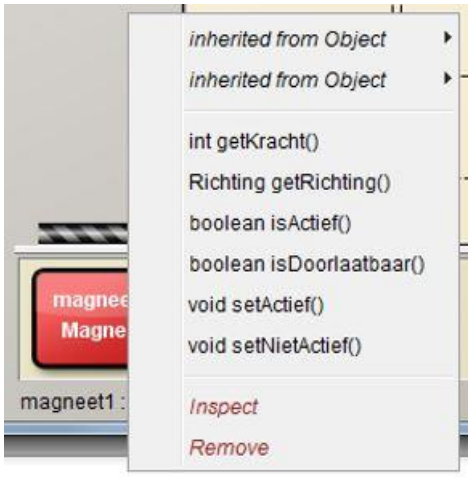

Fig. 4 - BlueJ functionaliteit objectenbench

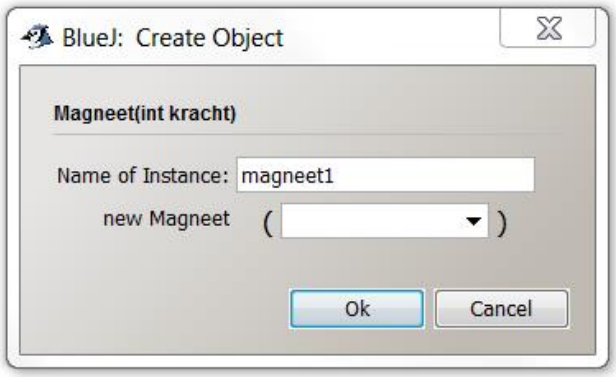

Fig. 5 - JFrame om parameters mee te geven in BlueJ

Fig. 6 – Tonen van de returnwaarde van voorbeeldmethode isActief() in BlueJ

2.2 BlueJ projecten

Om BlueJ projecten te kunnen importeren wordt eerst de mappenstructuur geanalyseerd. Deze verschilt van de lay-out van een standaard Java project in NetBeans. Een BlueJ project bestaat uit 1 hoofdmap die alle bestanden bevat zoals getoond in Fig. 7. In tegenstelling tot een NetBeans project bezit een BlueJ project geen ANT script om het programma te compileren.

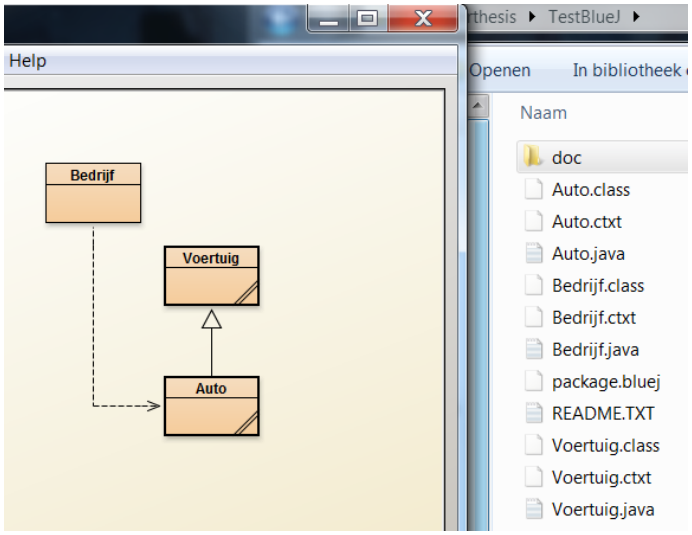

Fig. 7 - Inhoud van een BlueJ projectfolder

Een BlueJ project bevat steeds een package.bluej bestand. Door naar zulke bestanden te zoeken kunnen mappen herkend worden als BlueJ projecten. Dit bestand bevat de parameters van het BlueJ venster en de informatie nodig om het klassediagram te tekenen. De data van een “is a”-relatie ontbreekt en zal zelf opgevraagd moeten worden.

BlueJ slaat de gemaakte .class bestanden op in de hoofdmap naast de corresponderende .java bestanden. Dit zal problemen geven in NetBeans daar dit de map van de oude build steeds verwijdert. Hierdoor kunnen ook de orginele .java bestanden verwijderd worden.

Elk project heeft .ctxt bestanden die een copy van de commentaar in de code bevatten. Deze bestanden zijn niet nodig in een Java project in NetBeans en kunnen weg gelaten worden. Dezelfde commentaar staat ook vervat in de .java files.

Indien de gebruiker de javadoc opvraagt wordt er een map genaamd “doc” gecreëerd die de documentatie bevat.

Fig. 8 toont een diagram met de structuur van een BlueJ project.

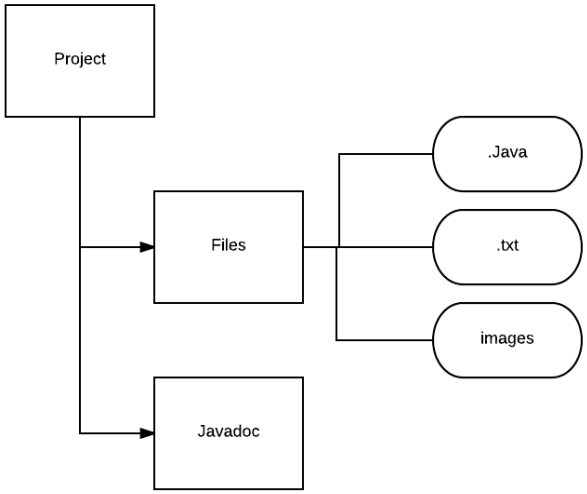

Fig. 8 - BlueJ Projectstructuur

2.3 Package.bluej

Elk BlueJ project bezit een package.bluej bestand waarin de projectinformatie, nodig voor het construeren van het klassediagram, wordt bijgehouden. Deze data bevat de compositie van het programma en de afmetingen van de BlueJ-vensters en objecten. De informatie nodig om erving te visualiseren is niet aanwezig, informatie over instantiëren wel.

Elke Java-klasse wordt in package.bluej voorgesteld als een “target”. De informatie die wordt bijgehouden is beschreven in Tabel 1. Een target kan een klasse, een enum of een interface zijn.

Tabel 1 - Opgeslagen klasse informatie

target1.name=Muur	De naam van de klasse
target1.type=ClassTarget	Het type van de klasse (class, enum, abstract)
target1.x=250	De x-coördinaat in het klassediagram
target1.y=350	De y-coördinaat in het klassediagram

De “has-a” relaties worden bijgehouden door de regels code in Tabel 2. Hierin wordt zowel de oproepende klasse als de opgeroepen klasse bijgehouden. Elke afhankelijkheid heeft een integer als index.

Tabel 2 - Opgeslagen relatie informatie

dependency1.from=MyGame	De oproepende klasse

dependency1.to=Spelbord	De opgeroepen klasse
dependency1.type=UsesDependency	Het relatietype

Het aantal klassen en relaties wordt apart bijgehouden volgens Tabel 3.

Tabel 3 - Aantal relaties en klassen volgens Package.bluej

package.numDependencies=31	Het aantal relaties
package.numTargets=12	Het aantal klassen

De overige informatie houdt informatie bij over de vormgeving van BlueJ. Dit is niet relevant voor NetBeans en wordt dus niet verwerkt. Bij het genereren van een package.bluej bestand zullen deze parameters ingevuld worden met standaard waarden. De parameters worden weergeven in Tabel 4.

Tabel 4 - Parameters in package.bluej irrelevant voor NetBeans

package.editor.height=848	Hoogte van het BlueJ venster
package.editor.width=824	Breedte van het BlueJ venster
package.editor.x=0	X-coördinaat van het BlueJ venster op het bureaublad.
package.editor.y=0	Y-coördinaat van het BlueJ venster op het bureaublad.
package.showExtends=true	Bepaalt of “Is a” relaties worden getoond.
package.showUses=true	Bepaalt of “has a” relaties worden getoond.
project.charset=UTF-8	Stelt de standaard karakter codering in.
readme.editor.height=1036	Hoogte van het readme venster
readme.editor.width=1292	Breedte van het readme venster
readme.editor.x=-9	x-coördinaat van het readme venster op het bureaublad
readme.editor.y=-9	Y-coördinaat van het readme venster op het bureaublad

3 NetBeans

NetBeans begon in 1996 als Xelfi. Dit was een studentenproject voor een Java IDE geleid door de faculteit voor wiskunde en fysica aan de Charles University in Praag. In 1997 werd het commercieel tot Sun Microsystems het kocht in 1999 en de IDE open-source aanbood. Sun werd in 2010 overgenomen door Oracle Corporation [9].

NetBeans is een software-ontwikkelingsplatform geschreven in JAVA. Men maakt hierbij onderscheid tussen de IDE en het NetBeans platform.

Het NetBeans Platform is een framework dat de ontwikkeling van grote JAVA Swing desktop applicaties vergemakkelijkt. Het laat toe om programma's te ontwikkelen met modulaire componenten genaamd modules. Deze modules kunnen dynamisch geïnstalleerd worden. Ook kunnen nieuwe modules gecreëerd worden om zo extra functionaliteit aan te bieden.

NetBeans IDE is een open-source ontwikkelomgeving gebouwd op het ANT build systeem. Oorspronkelijk was het bedoeld voor JAVA maar tegenwoordig worden ook andere programmeertalen ondersteund. Het programma is ontwikkeld met het NetBeans Platform, dat reeds alle modules bevat die nodig zijn voor JAVA ontwikkeling.

3.1 Modules

Het NetBeans platform biedt standaard meerdere APIs aan waarmee applicaties gebouwd kunnen worden. Er zijn ook modules die specifiek voor de IDE zijn.

Een NetBeans module is een Java-archief (.jar) dat de klassen en andere bestanden bevat die de module vormen [10]. Klassen in een module kunnen niet gezien worden door andere modules. Om ze te importeren moet deze module eerst publiek gemaakt worden zodat een oproepende module er zijn afhankelijkheid van kan verklaren.

De functionaliteit van de NetBeans IDE wordt dus aangebracht door bestaande APIs. Deze modules hebben hun eigen methoden en zorgen voor de werking van de IDE. Met de "@override" annotatie kan de methode van een superklasse overschreven worden. Er wordt hierdoor een methode overschreven i.p.v. toegevoegd.

Er is in NetBeans geen onderscheid tussen een plug-in en een module. Voor deze thesis is de term "plug-in" gekozen omdat het eindproduct zal bestaan uit meerdere modules die samen nieuwe functies zullen bieden en dus samen een plug-in vormen.

3.2 NetBeans Projecten

Een standaard JAVA project in NetBeans bevat verschillende mappen voor zijn bron bestanden en de bestanden voor de build. Fig. 9 laat zien dat er bij het aanmaken van het project nog geen map bestaat voor de gecompileerde bestanden. Na compilatie wordt er een nieuwe map genaamd "build" gecreëerd waarin de .class files worden opgeslagen zoals getoond in Fig. 10.

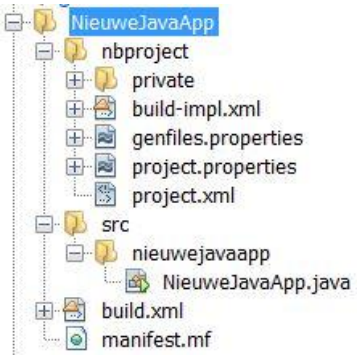

Fig. 9 - NetBeans project voor compilatie

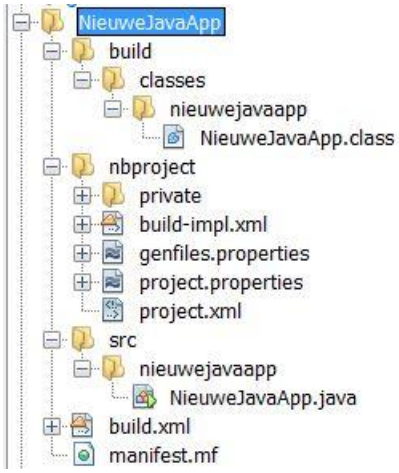

Fig. 10 - NetBeans project na compilatie

Het Build.xml bestand bevat een ANT script voor het compileren en runnen van het project.

Het manifest geeft een beschrijving van het project en de omgeving. Dit is het eerste bestand dat NetBeans bekijkt.

De nbproject-map bevat de configuratie bestanden van de applicatie.

De src-map bevat de bronbestanden van het Java-project.

Fig. 11 geeft een visueel schema van deze projectstructuur.

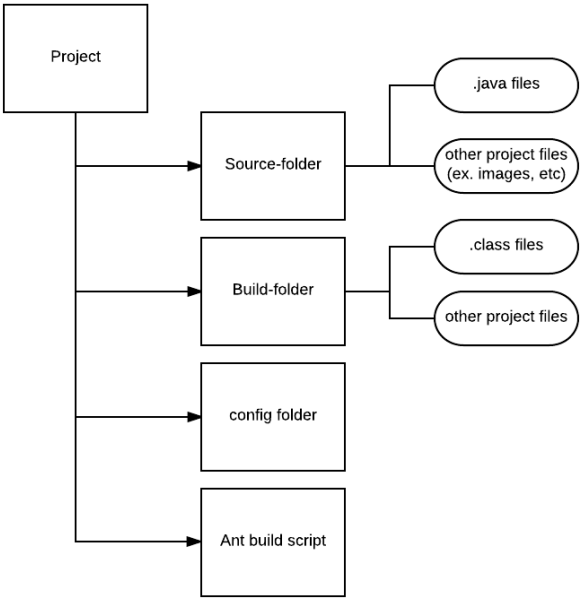

Fig. 11 - NetBeans project structuur

4 APIs

4.1 Java Reflection API

Met deze API kan men het runtime gedrag van applicaties in Java bekijken. Fig. 12 toont dat de functionaliteit in 2 groepen verdeeld is. De interface kan met introspectie het type en de eigenschappen van objecten analyseren. Met reflectie kan de structuur en het gedrag van een object bekeken en veranderd worden. Hiermee kunnen objecten dynamisch worden aangemaakt en hun methodes uitgevoerd.

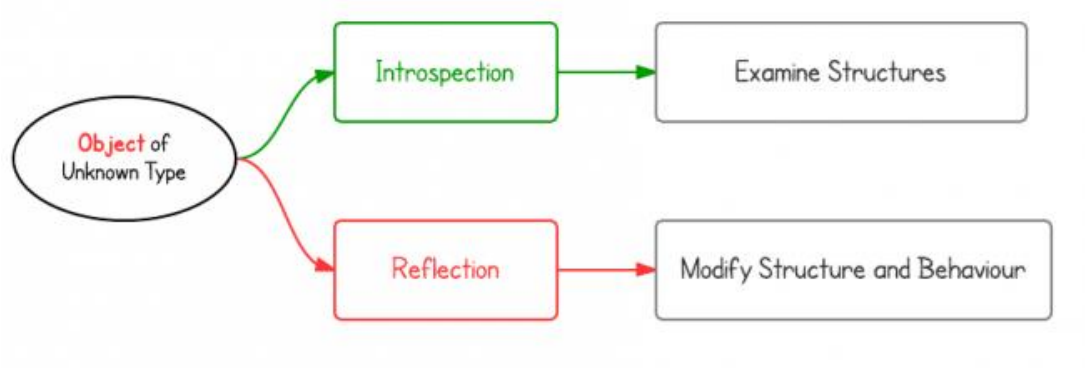

Fig. 12 - Introspectie vs. Reflectie [11]

Het klassediagram gebruikt deze API om informatie te vergaren over de samenhang van de bestanden. De objectenbench gebruikt de interface om informatie over constructors en methodes op te vragen en deze functies uit te voeren. Tabel 5 geeft een overzicht van de gebruikte methodes.

Tabel 5 - Gebruikte methodes van de Java Reflection API

Object.getClass()	Geeft de klasse terug van een aangemaakt Object.
Class.getName()	Geeft een string terug met de naam inclusief package van een klasse terug.
Class.getSimpleName()	Geeft een string terug met de naam zonder package van een klasse terug.
Class.getModifiers()	Hiermee kan men de klasse modifiers opvragen. (vb. isPublic(), isFinal(), isAbstract(), …)
Class.getDeclaredFields()	Geeft alle aangemaakte globale datamembers terug.
Class.getDeclaredConstructors()	Geeft alle aangemaakte constructors terug.
Constructor.getParameters()	Geeft de parameters van de constructor.
Constructor.getParameterTypes()	Geeft de types van de parameters van de constructor.

Constructor.newInstance(Object… params)	Voert de constructor uit van een klasse en geeft de parameters mee. Dit geeft een object terug.
Class.getDeclaredMethods()	Geeft alle aangemaakte methodes van een klasse terug.
Class.getMethod("methodName", Object… params)	Geeft 1 methode terug met bepaalde naam en parameters.
Method.getParameters()	Geeft de parameters terug van een methode.
Method.getParameterTypes()	Geeft de parametertypes terug van een methode.
Method.getReturnType()	Geeft het returntype terug van een methode.
Method.invoke(Object doel, Object… params)	Voer de methode uit van een Object.

4.2 Checkstyle API

CheckStyle is een tool die in softwareontwikkeling gebruikt wordt om statische code te analyseren [12]. De API controleert automatisch of Java broncode volgens de juiste codeerstandaard geschreven wordt. Hiervoor worden standaard checks aangeboden maar er kunnen ook nieuwe checks geschreven worden.

Welke controles moeten gebeuren, is configureerbaar via een xml-bestand. Dit bevat steeds een root-module *checker* met alle uit te voeren checks als submodules zoals in Fig. 13.

```
<module name="Checker">
    <module name="JavadocPackage"/>
    <module name="TreeWalker">
        <module name="AvoidStarImport"/>
        <module name="ConstantName"/>
        <module name="EmptyBlock"/>
    </module>
</module>
```

Fig. 13 - CheckStyle configuratie

De configuratiefile maakt gebruik van een Module TreeWalker. Deze ondersteunt submodules die zijn afgeleid van de AbstractCheck klasse. De TreeWalker zet Java bestanden om in een abstract syntax tree, zoals in Fig. 14, en geeft deze door aan elk van de check submodules.

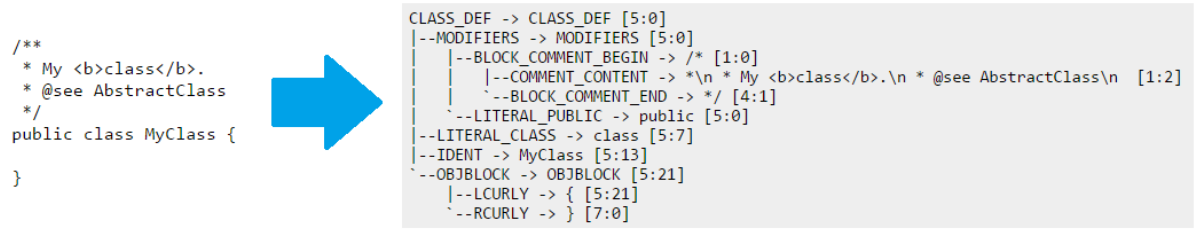

Fig. 14 - Java codeblok en bijhorende abstract syntax tree

4.3 NetBeans Visual Library

De Visual library 2.0 API is de opvolger van de Graph library 1.0 [13]. De bibliotheek bevat een aantal herbruikbare, voorgedefineerde widgets. Deze widgets zijn primitieve visuele elementen, vergelijkbaar met een JComponent in Swing [13], en hebben ingebouwde features zoals actions, layouts en borders [14]. Dit maakt het mogelijk om snel en makkelijk data te visualiseren.

De programmeerstijl is gelijkaardig aan Swing. In een javafx.scene.Scene klasse wordt een boomstructuur van visuele elementen, de widgets, opgebouwd. De methode *Scene.createView()* geeft een JComponent terug die de scene weergeeft [13].

Een GraphScene<myNode,myEdge> geeft data grafisch weer met nodes en edges. Een node beeldt in dit geval een instantie van de klasse JavaBestand af. Een edge is een verbinding tussen 2 nodes en heeft zowel een oorsprong als bestemming [5]. Fig. 15 toont 2 label widgets die door middel van een edge verbonden zijn. De GraphScene biedt methodes om deze widgets aan te maken methodes, hiervan is een overzicht geplaatst in Tabel 6.

Fig. 15 - een edge tussen 2 nodes

Tabel 6 – Methodes van de klasse GraphScene [13]

Methode	Modifier en type	Beschrijving
attachNodeWidget(*N node)*	protected abstract Widget	Maakt een widget aan die de node in de scene representeert.
attachEdgeWidget(*E edge)*	protected abstract Widget	Maakt een widget aan die de edge in de scene representeert.
attachEdgeSourceAnchor(*E edge,* *N oldSourceNode,* *N sourceNode)*	protected abstract void	Bepaalt de oorsprong van de edge.
attachEdgeTargetAnchor(*E edge,* *N oldTargetNode,* *N targetNode)*	protected abstract void	Bepaalt de bestemming van de edge.

5 Structuur van de plug-in

5.1 4 C's

Het C4-model wordt gebruikt bij de ontwikkeling en communicatie van software architectuur. Visualisatie is een belangrijk hulpmiddel bij het ontwikkelen van programma's. Met behulp van simpele diagrammen wordt getoond hoe de verschillende onderdelen met elkaar werken. Fig. 16 toont hoe de 4 lagen van het model verdeeld zijn.

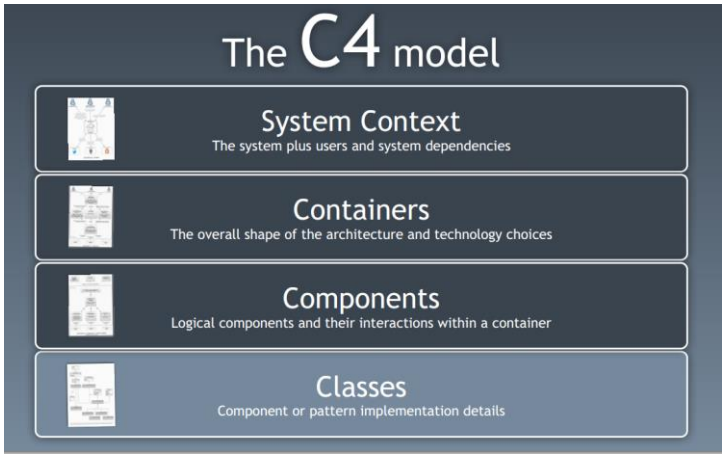

Fig. 16 - Overzicht van het C4-model [15]

5.1.1 Context diagram

Dit is het startpunt in de ontwikkeling van het programma. Het systeem zelf wordt centraal geplaatst, omringd door de users en systemen waarmee het in contact komt. Hier wordt niet in detail gegaan maar resulteert in een globaal overzicht dat makkelijk te begrijpen is voor niet-technisch opgeleiden.

Bij het opstellen van het diagram in Fig. 17 is het duidelijk dat het project uit 3 onderdelen bestaat. Er zullen 3 modules gemaakt worden die de functionaliteit opsplitsen in visualisatie, simulatie en codeerstandaardisering.

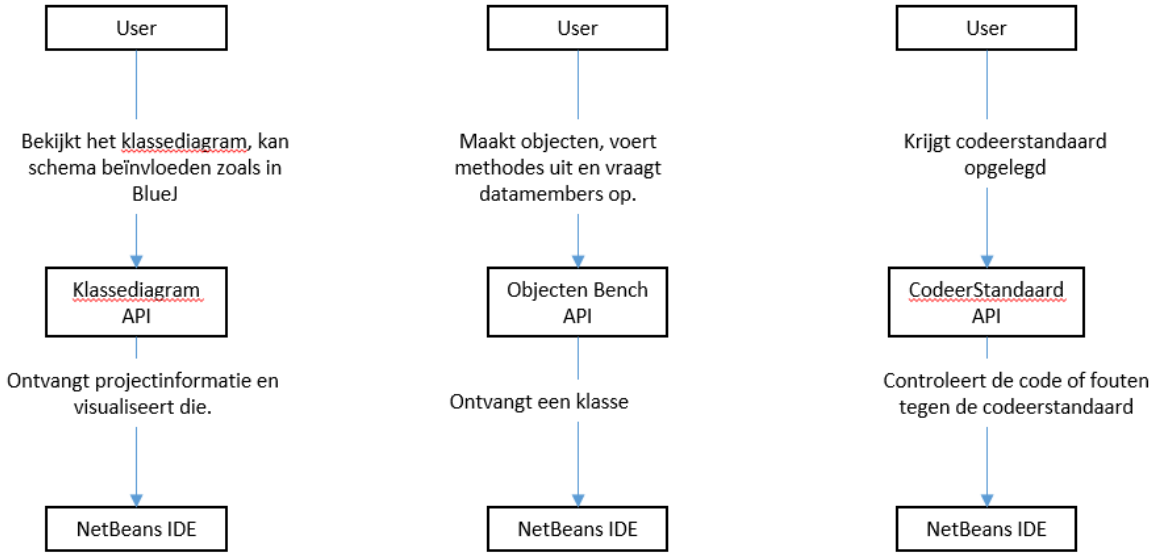

Fig. 17 - Context diagram

5.1.2 Container diagram

Fig. 18 geeft een inzicht in de externe APIs die worden aangesproken. Het klassediagram en de objectenbench zullen dezelfde interfaces gebruiken maar andere methodes aanroepen. De module voor codeerstandaardisering zal de CheckStyle API gebruiken.

De user gebruikt de NetBeans IDE. Dit maakt aanspraak op de ontwikkelde modules en bevat de projectinformatie. Het klassediagram maakt gebruik van introspectie om de relaties tussen bestanden op te roepen. De objectenbench gebruikt reflectie om objecten aan te maken en hun methodes op te roepen. Deze 2 modules maken ook gebruik van de NetBeans Visual Library om hun informatie grafisch weer te geven.

De evaluatieklasse maakt enkel gebruik van het CheckStyle API. Informatie van fouten tegen de codeerstandaard worden weergeven in de code-editor van NetBeans. Alle fouten worden bij elkaar opgeteld en in een JFrame weergeven als score voor de student.

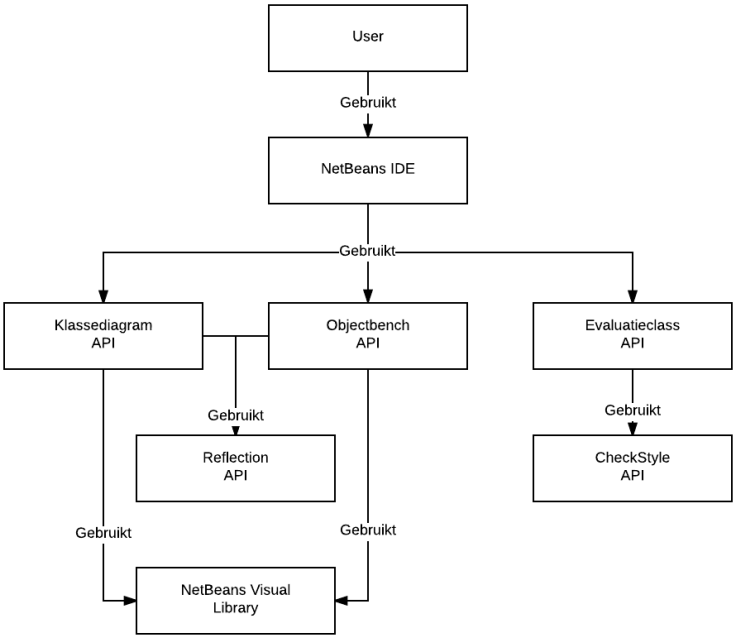

Fig. 18 - Container diagram

5.1.3 Component diagram

Het componentendiagram in Fig. 19 beschrijft de componenten die nodig zijn om de functionaliteit te verkrijgen. Het beschrijft de fysieke delen van een systeem zoals bestanden, bibliotheken, enz. [16].

Interactie tussen de user en de plugin gaat altijd via de NetBeans IDE. Hiervoor is een interface uitgewerkt die de projectinformatie opvraagt en doorgeeft aan de plugin. Een hulpklasse werd ontwikkeld om de Reflection API aan te spreken.

De hulpklasse verwerkt de methodes van Java Reflection in een meer bruikbare vorm. Het aantal parameters dat moet meegegeven worden hangt af van de constructor of methode. Deze klasse zorgt dat er een Array met objecten kan meegegeven worden om zo de functie correct uit te voeren.

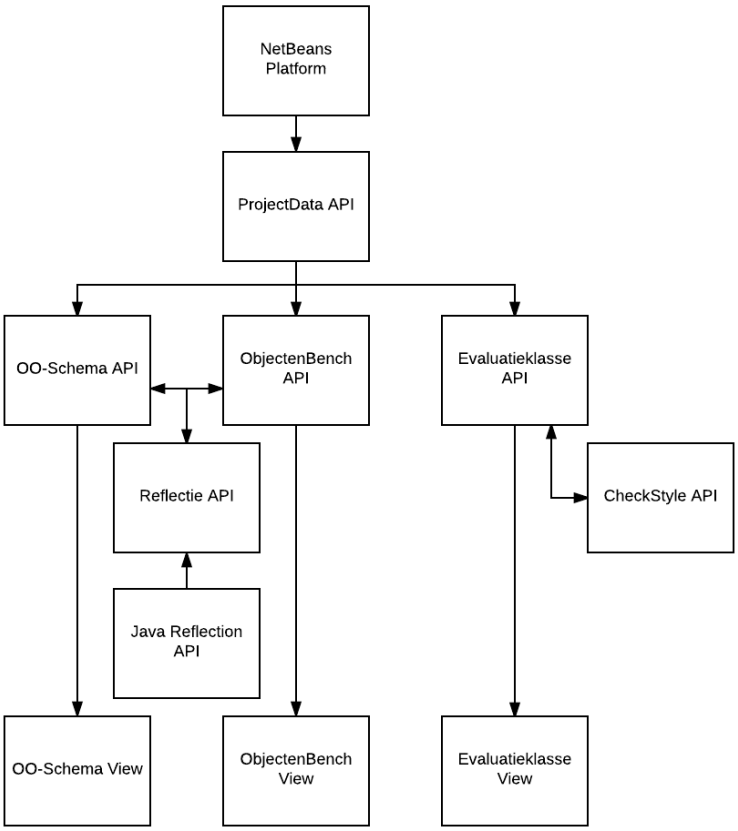

Fig. 19- Component diagram

5.1.4 Class diagram

Het klassediagram, afgebeeld in Fig. 20, beschrijft de structuur van een systeem door de aanwezige klassen en hun samenhang te tonen. Dit diagram is de belangrijkste bouwblok bij object georiënteerd programmeren.

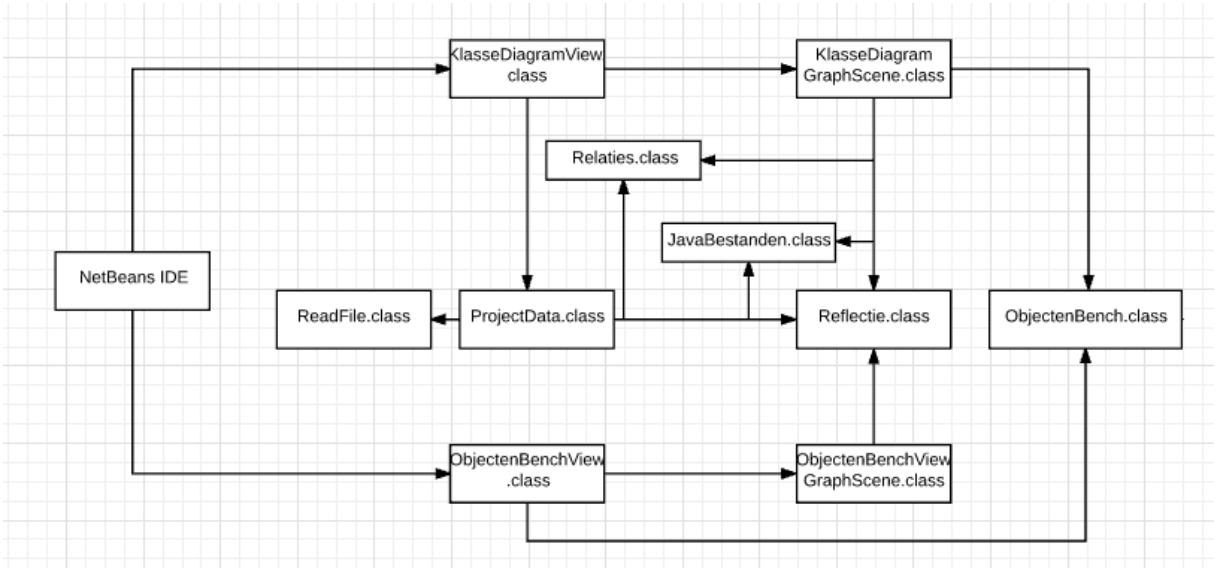

Fig. 20 - Klassediagram plugin

De GUIs voor zowel het klassediagram als de objectenbench worden opgeroepen via het NetBeans IDE. Indien de user het klassediagram opent, wordt de projectdata klasse aangemaakt. Deze gebruikt de Reflectie-hulpklasse om de JavaBestanden en Relaties op te slaan. Het ReadFile.class bestand wordt gebruikt om broncode en het package.bluej bestand in te lezen.

Wanneer de projectdata gegenereerd is wordt deze in het KlasseDiagramView getekend met een GraphScene. In deze scene kunnen via een rechtermuisknop menu JavaBestanden in objecten worden omgezet die in ObjectenBench.class opgeslaan.

Als de gebruiker de ObjectenBenchView opent worden de objecten uit de ObjectenBench klasse geladen en afgebeeld in een GraphScene. Deze GraphScene gebruikt de reflectie hulpklasse om de methodes van de objecten uit te voeren.

5.2 Model-View-Controller

Het Model-View-Controller principe is een design pattern waarbij men een toepassing opsplitst in 3 eenheden. Dit wordt geïllustreerd in Fig. 21.

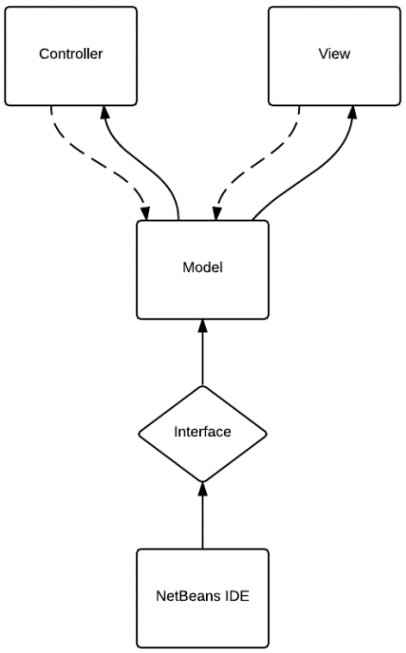

Fig. 21- Software architecture volgens MVC model

5.2.1 Het model

Dit concept omvat alle data en methodes nodig om de extensie te laten functioneren. Deze zitten vervat in de ProjectData module. Via een interface ontvangt het model de datasource van de NetBeans IDE. Hierna worden de bestanden geanalyseerd om de BlueJ-visuele voorstelling te tekenen.

Eens de constructoren van de verschillende klassen bekend zijn kunnen met de Java Reflection API objecten worden aangemaakt en hun methodes uitgevoerd. Hiervoor moet het .java-bestand eerst gecompileerd worden zodat Java Reflection een .class-bestand ter beschikking heeft dat de bytecode [17] bevat.

De evaluatie-klasse bekijkt enkel de Java broncode en kan dus op elk moment gestart worden indien er een .java bestand als datasource beschikbaar is.

5.2.2 De view

Hier wordt het model gevisualiseerd. Het NetBeans Platform reikt hiervoor verschillende modules aan. De basis display eenheid is een TopComponent. Dit is een JComponent subklasse die in het NetBeans window system is verwerkt [18]. Hierdoor is het mogelijk om de JComponent los te maken van het IDE venster en te manipuleren, zonder dat de programmeur hier zelf code voor moet schrijven.

Het editorview en outputview krijgen nieuwe TopComponent-vensters waarin een GraphScene wordt aangemaakt. In deze scene kunnen met behulp van de NetBeans Visual API widgets getekend worden. Dit kunnen nodes of edges zijn.

Om de constructors en methodes uit te voeren wordt er een JPopupMenu gegenereerd wanneer er met de rechtermuisknop op een node geklikt wordt. Dit is mogelijk met een methode van de ActionFactory, de factory klasse voor alle ingebouwde functies van de NetBeans Visual Library.

5.2.3 De controller

De controller bepaalt hoe het model moet reageren op input van de gebruiker. Hiernaast kan er geluisterd worden naar veranderingen in de lookup van een bepaalde component en om zo een handeling te activeren.

Indien het klassediagram wordt opgeroepen wordt de projectdata opgevraagd. Indien het om een herkend projecttype gaat wordt het diagram getekend. Hier kunnen dan Objecten aangemaakt worden door uit een JPopupMenu een constructor te kiezen. Dit menu is beschikbaar via de rechtermuisknop.

Indien de objectenbench wordt opgeroepen beeldt deze de objecten af die via het klassediagram in de bench zijn geplaatst. Voor het uitvoeren van methodes wordt ook een JPopupMenu voorzien via de rechtermuisknop.

De evaluatieklasse kan gestart worden in het klassediagram via het JPopupMenu op te roepen met de rechtermuisknop.

6 ProjectData module

De projectData module is het model van de applicatie. Dit bevat alle klassen en methoden voor de BlueJ-functionaliteit van de plugin. Deze sectie bespreekt de functies die deze module aanbrengt.

6.1 Projectdata opvragen

Om de informatie van een project te verwerken moet het eerst ingeladen worden. Het kan zowel om een NetBeans Java als een BlueJ project gaan. Daarom wordt gekeken naar de mappenstructuur en of een package.bluej bestand aanwezig is. Fig. 22 geeft hier een overzicht van.

Methodes zijn uitgewerkt om compositie en erving van de projectbestanden op te vragen. Hiermee kan het package.bluej bestand gegenereerd worden indien dit nog niet aanwezig is. Wanneer dit bestand bestaat, wordt het geparsed om de relaties tussen de bestanden op te vragen.

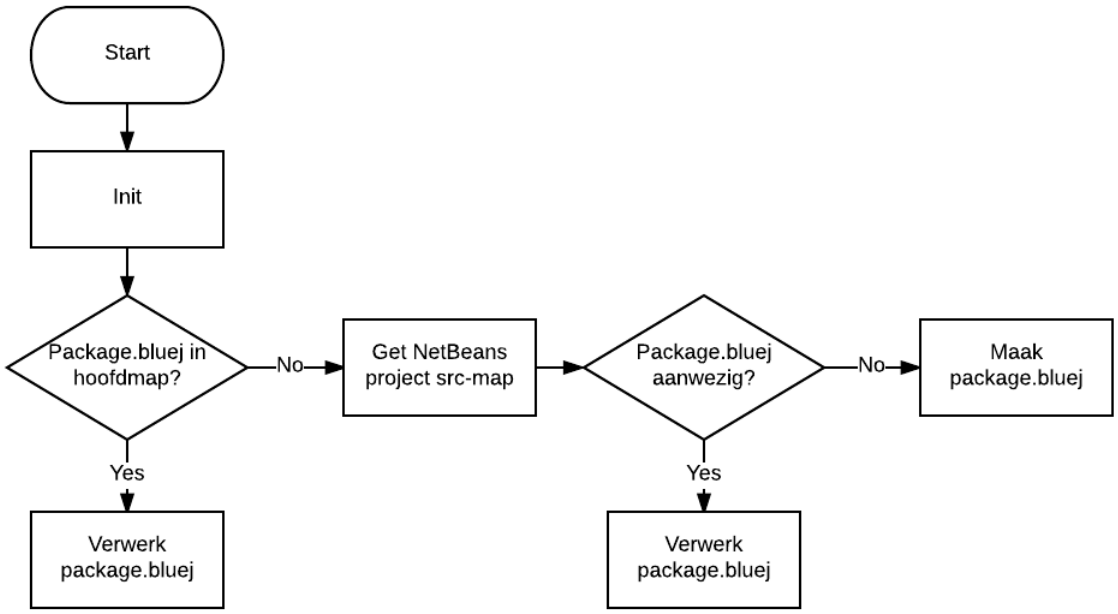

Fig. 22 - Flowchart voor laden en verwerken van package.bluej

6.1.1 Het project opvragen

Om de locatie van een open project op te vragen worden enkele methodes aangeboden door de NetBeans Project API. Alle bestanden in een NetBeans Java programma zitten in een projectmap. De projecten die in de IDE geopend zijn kunnen opgevraagd worden met *OpenProjects.getDefault().getOpenProjects()*.

De locatie van het project wordt verkregen met *Project.getProjectDirectory()* en geeft de projectmap terug als FileObject. Dit is een abstracte klasse die de basis vormt voor alle implementaties van bestandsobjecten in een bestandsysteem [19].

De nodes in het klassediagram stellen de .java bestanden voor in de projectmap. Voor de werking van de Reflection IDE moeten de .class bestanden gebruikt worden. Deze moeten eerst ingeladen worden met behulp van een ClassLoader. In BlueJ zitten de broncode en de gecompileerde bestanden in dezelfde map. NetBeans splitst deze op in een “src” en een “class” map. Nu de locaties

van de bestanden bekend is kan de informatie gegenereerd worden die nodig is om het klassediagram te tekenen.

6.1.2 Compositie opvragen

Java reflection laat toe om “Has a”-relaties op te vragen met de methode getFields(). Deze methode geeft alleen de globale datamembers terug. Indien een variabele genest zit in een methode dan wordt deze niet herkend. De variabele bestaat namelijk enkel in de methode maar niet daarbuiten [20].

Een .java bestand bevat de broncode die leesbaar is voor de gebruiker. Een gecompileerd .class bestand bevat de bytecode die de Java virtuele machine gebruikt om een applicatie uit te voeren. De Java Reflection API gebruikt deze laatste en kan dus de lokale variabelen niet opvragen, omdat de compiler ze niet opslaat nadat de methode is uitgevoerd.

Om lokale datamembers toch uit de bytecode te kunnen opvragen moet de klasse gecompileerd zijn met “lokale variabele debug informatie” door middel van *javac –g … .* De lokale variabelen kunnen dan opgevraagd worden met externe debugger APIs indien de JVM gepauzeerd wordt door de debugger agent [21]. Alternatief kan er gebruik gemaakt worden van een bytecode bibliotheek zoals ASM (http://asm.ow2.org/).

Voor dit project werd gekozen om lokale datamembers te zoeken in de broncode. Het .java bestand wordt geparsed en nagegaan op aanwezigheid van woorden die overeen komen met namen van de projectbestanden. Dit kan problemen geven indien er bestanden zijn die dezelfde naam hebben als klassen van externe bibliotheken. Door na te gaan welke imports er gedaan worden kunnen deze externe klassen geïdentificeerd worden. Het proces om de compositie op te vragen wordt geïllustreerd in Fig. 23.

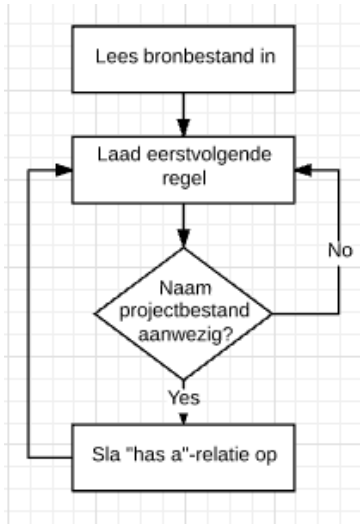

Fig. 23 - Flowchart compositie opvragen

6.1.3 Erving opvragen

Een klasse is altijd afkomstig van een andere klasse en erft diens methodes. De Object klasse, van het java.lang package, staat bovenaan in de klasse hiërarchie. Iedere klasse is direct of indirect afkomstig van de Object klasse [22].

Java Reflection laat toe om de superklasse op te vragen van een Java klasse met *Class.getSuperclass()*. Indien deze superklasse voorkomt in de projectbestanden, en niet afkomstig is van een andere package zoals java.lang.object, wordt deze klasse opgeslagen als bestemming van de "is a"-relatie. Fig. 24 illustreert dit proces.

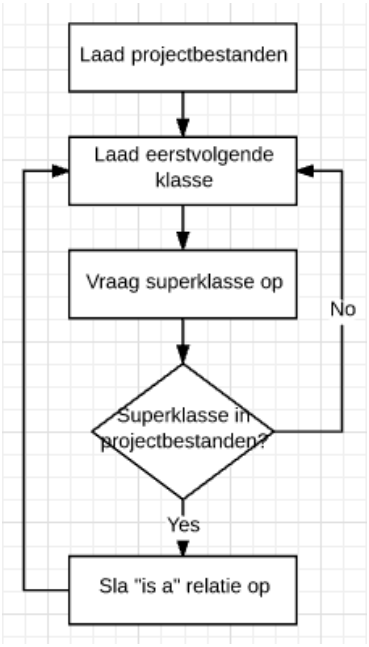

Fig. 24 - Flowchart erving opvragen

6.1.4 Package.blueJ parsen

Door package.bluej te parsen kunnen de klassen en relaties opgevraagd worden. Elke regel wordt ingelezen als een string met behulp van een ReadFile klasse. De data wordt hieruit gehaald en opgeslagen in ArrayLists.

De eerste regel moet altijd "#BlueJ package file" bevatten, anders is het een foutief bestand. Bij een correct bestand wordt regel per regel verwerkt. Deze Strings hebben de structuur onderdeelNaam.parameterNaam=waarde en worden zo gesplitst.

De klassen en relaties worden gevonden door de *onderdeelNaam* te filteren. Elke relatie bezit een index, oorsprong, bestemming en type. Elke klasse bezit een naam, x- en y-coördinaat. Deze parameters worden gebruikt om het klassediagram te tekenen.

6.1.5 Package.bluej aanmaken

Om het bestand te genereren moeten de klassen en hun relaties opgevraagd worden. Hiervoor wordt de Reflection API gebruikt. De informatie wordt gegenereerd en opgeslagen in de projectData module. Hierna wordt een package.bluej bestand aangemaakt in de src-map van het project en de data hier naartoe geschreven met de klasse PrintWriter.

De writer wordt geïnstantieerd met PrintWriter(String fileName, String charSet) waardoor het nieuwe bestand wordt aangemaakt. Aan dit bestand worden nieuwe regels toegevoegd met PrintWriter.println(String tekstRegel). De eerste regel van het bestand moet altijd "#BlueJ package file" bevatten. Hierna worden de relaties en de bestanden opgeslagen zoals in Tabel 1 en Tabel 2.

Wanneer alle info naar het package.bluej bestand geschreven is wordt de writer afgesloten met PrintWriter.close().

6.2 Javaklassen en hun relaties

Er zijn 2 klassen aangemaakt om de informatie van Javabestanden en hun onderlinge relaties bij te houden. Deze data wordt gebruikt om het klassediagram te tekenen, externe klassen te laden voor gebruik in de objectenbench en om het package.bluej bestand te genereren.

6.2.1 Javabestanden

Met deze klasse wordt de informatie van elke Java-klasse bijgehouden. Elk bestand krijgt naast zijn naam en type een uniek index-nummer. De coördinaten in het klassediagram van de node die het bestand voorstelt worden hier bijgehouden. Fig. 25 geef hier een overzicht van. De ProjectData klasse slaat al deze Javabestand-objecten op in een array.

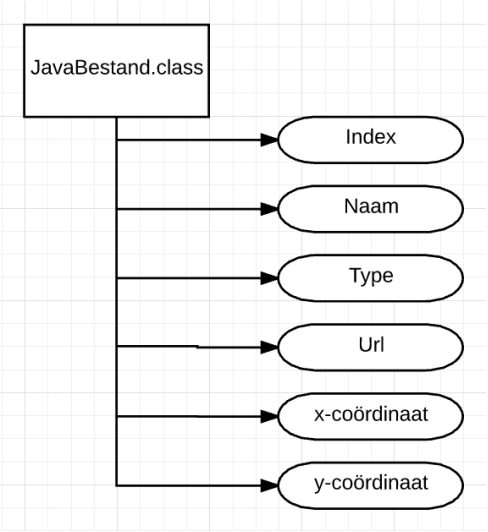

Fig. 25 - Globale datamembers van een JavaBestand.class

Het klassediagram visualiseert de .java-bestanden. De datamember Url verwijst naar de locatie van het bijhorende .class bestand, deze is verschillend voor BlueJ en NetBeans projecten. Indien dit .class bestand aanwezig is kan de klasse worden ingeladen als een externe afhankelijkheid. Daarom wordt in JavaBestand.class de methode *laadKlasse()* aangereikt. Hierna kan de klasse gebruikt worden met de Java Reflection API.

6.2.2 Relaties

Een edge is een relatie tussen 2 bestanden zoals besproken in 4.3. De klasse Relatie.class is aangemaakt om voor elke edge de index, het type relatie, de oorsprongs- en de bestemmingsnode op te slaan. Compositie wordt opgeslagen als “has a”-relatie, erving als “is a”-relatie. Dit onderscheid wordt in het klassediagram visueel aangegeven. Fig. 26 geeft een overzicht van de data die deze klasse opslaat.

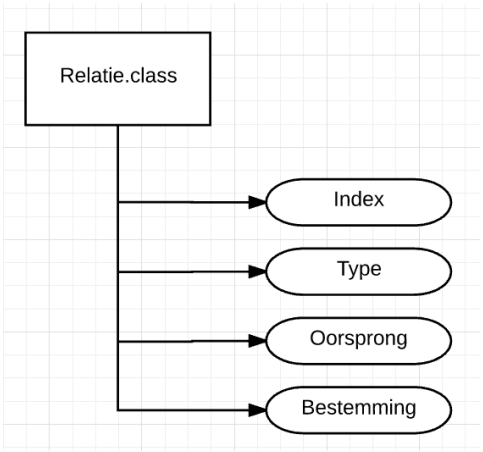

Fig. 26 - Globale datamembers van Relatie.class

Alle relaties worden bijgehouden in een array in de ProjectData klasse.

6.3 Objecten in de bench

De objectenBench houdt een array bij met alle aangemaakte Java-objecten. Eenmaal een object is aangemaakt kunnen de methodes uitgevoerd worden met de Java Reflection API.

Wanneer er vanuit het klassediagram een Java-object wordt aangemaakt, controleert het programma of de objectenbench al is aangemaakt. Indien dat het geval is worden de objecten toegevoegd aan de bestaande objecten en wordt het objectenbench-venster geüpdatet. Anders wordt er een nieuwe bench aangemaakt en het object erin opgeslagen. Dit is geïllustreerd in Fig. 27.

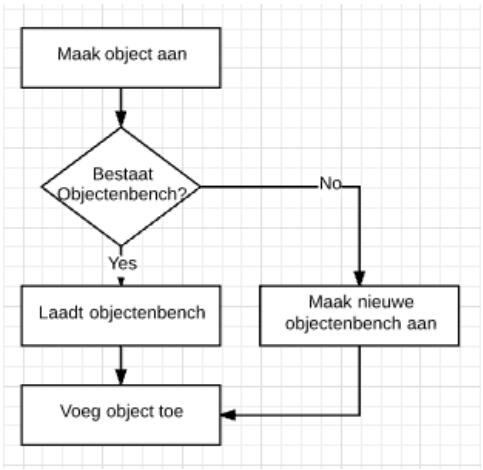

Fig. 27 - flow-chart voor objecten toe te voegen aan de objectenbench

6.4 Java Reflectie hulpklasse

De hulpklasse hervormt methodes van de Reflection API zodat ze gemakkelijker te gebruiken zijn door het klassediagram en de objectenbench. Deze methodes zijn opgedeeld in functies uit te voeren op klassen en functie voor objecten.

Afhankelijk van de constructor of methode moet er al dan niet een aantal parameters worden meegegeven van bepaalde types. De parameters worden uit een JFrame ingelezen als Strings. Deze worden dan geconverteerd naar het nodige type.

6.4.1 Dynamisch klassen laden

Om de Java Reflection API te gebruiken moet een klasse worden ingeladen met de java.lang.ClassLoader. Dit creëert een externe afhankelijkheid op de volgende manier [23]:

```
URL[] urls = new URL[]{url};
ClassLoader myLoader = new URLClassLoader(urls);
Class myClass = myLoader.loadClass(klasseNaam);
```

De URLClassLoader wordt aangemaakt met een array die de locaties van de klassebestanden bevat. Deze laadt de klasse en andere bestanden waar deze naar refereert. Java's ingebouwde klasse laders controleren altijd of een klasse reeds geladen is. Daarom wordt de klasse telkens opnieuw geladen indien de methodes van een object worden uitgevoerd. Op deze manier is er steeds maar 1 klasse ingeladen wanneer Java Reflection gebruikt wordt. De array *URL[] urls* bevat dus maar 1 element, de locatie van de te laden klasse.

Eens aangemaakt, wordt met de ClassLoader de klasse geladen met de methode *ClassLoader.loadClass(String klasseNaam)*. Java Reflection kan deze klasse dan gebruiken om objecten aan te maken en hun methodes uit te voeren.

6.4.2 Objecten aanmaken

Om een Javaobject aan te maken moet er een klasse ter beschikking zijn. De constructors van deze klasse worden dan opgevraagd en opgeslagen in een array met de functie *Class.getDeclaredConstructors()*. Een integer wordt meegegeven als parameter om de uit te voeren constructor op te roepen.

De parameters die aan een methode moeten meegegeven worden kunnen van verschillende types zijn en verschillen voor elke constructor. Hierom wordt er gebruik gemaakt van het type Object… varArgs. Dit slaat de parametertypes op als een sequentie van argumenten.

Voordat het Object wordt aangemaakt met *Constructor.newInstance(Object… inintargs)* wordt er een JFrame gegenereerd waarin de user de parameters kan ingeven. Deze worden ingelezen als Strings en omgezet naar het vereiste datatype. De gewenste types kunnen opgevraagd worden met de functie *constructor.getParameterTypes()*.

Deze feature maakt veelvuldig gebruik van de Reflection API. De hulpmethode *createInstance(Class deKlasse, int constructorNr, Object… params)* verzamelt deze onder 1 methode om zo makkelijker objecten te kunnen aanmaken. Deze objecten worden in de objectenbench opgeslaan.

De flowchart in Fig. 28 toont hoe een object wordt aangemaakt.

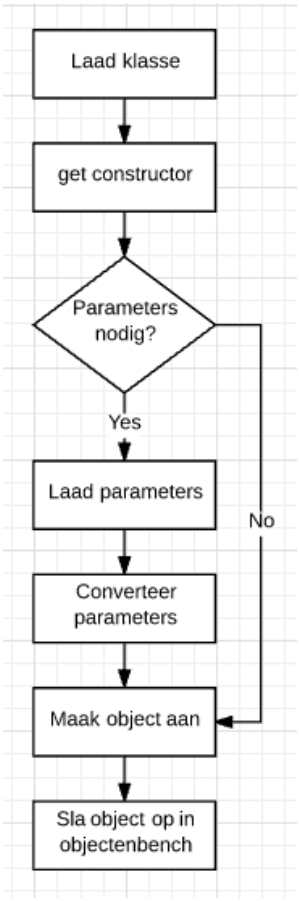

Fig. 28 - Flowchart objecten aanmaken

6.4.3 Methodes uitvoeren

Net zoals bij het aanmaken van objecten wordt voor het uitvoeren van methodes de Java Reflection API gebruikt. Ook hier moet informatie van de functie opgeroepen worden voordat ze kan uitgevoerd worden. Dit wordt gedaan met de nieuwe hulpfunctie *invokeMethod(Object obj, String methodName, Object… varArgs)*. De flowchart in Fig. 29 geeft een overzicht van de werking.

Om informatie te verkrijgen over de methodes moet eerst de klasse van het object opgevraagd worden met *Object.getClass()*. De klasse geeft toegang tot de methodes met *Class.getDeclaredMethods()*. Deze worden opgeslagen in een array. Hier is de constructor te kiezen met een String die zijn naam bevat.

De gewenste parametertypes worden opgevraagd met *Method.getParameterTypes()*. De parameters worden weer ingelezen d.m.v. een JFrame met JTextFields. Ook hier zullen de Stringwaardes moeten omgezet worden naar de vereiste datatypes om ze dan mee te geven als Object… varArgs.

De methode wordt opgeroepen vanuit de klasse maar wordt uitgevoerd door een object. Dit object wordt meegegeven wanneer de methode wordt uitgevoerd met *Method.invoke(Object obj, Object… args)*.

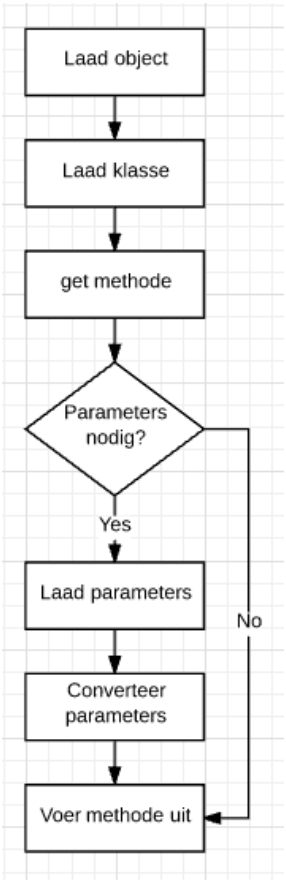

Fig. 29 - Flowchart methodes uitvoeren

6.4.4 String naar bepaald type converteren

Parameters die aan een constructor of methode moeten meegegeven worden, worden via een JFrame ingelezen. De JTextFields, waarin de user de waarden ingeeft, geven deze als Strings terug. Deze Stringwaarde zal indien nodig naar een ander type geconverteerd worden. Dit type kan variëren voor elke parameter. De types van de parameters kan opgevraagd worden met de Java Reflection functie *constructor.getParameterTypes()*.

Op StackOverflow.com werd onderstaande functie gevonden om de Strings naar de juiste types te converteren [24]. Deze maakt gebruik van de PropertyEditorManager en laadt de PropertyEditor van een bepaald type. De PropertyEditor laat toe om een waarde als Tekst mee te geven. Daar deze editor weet welk type er verwerkt wordt geeft deze de waarde terug als juiste type. De volgende alinea toont de helpmethode die de parameters converteert.

```
private static Object convert(Class<?> targetType, String textWaarde) {
    PropertyEditor editor = PropertyEditorManager.findEditor(targetType);
    editor.setAsText(textWaarde);
    return editor.getValue();
}
```

De PropertyEditor en PropertyEditorManager zijn klassen uit het java.bean package in de standaard Java bibliotheken [24].

7 KlasseDiagram

Het klassediagram visualiseert de structuur van het programma. De constructie gebeurt aan de hand van het package.bluej bestand. Dit zit in elk BlueJ project of wordt voor een standaard Java project gegenereerd door de ProjectData module. Indien het bestand beschikbaar is wordt de grafische voorstelling getekend met nodes en edges. Fig. 30 geeft de hiërarchie van de visuele componenten.

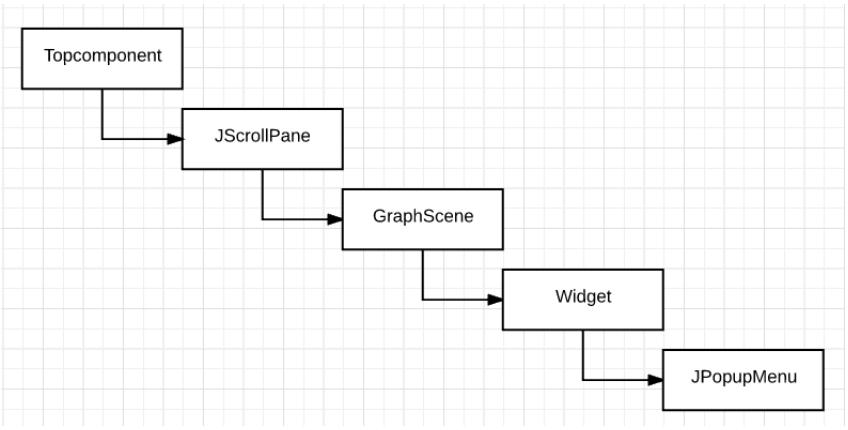

Fig. 30 - Hiërarchie van de visuele componenten

De methodes voor de werking van het klassediagram worden aangereikt door de NetBeans Visual library en de ProjectData module.

7.1 Klassediagram view

Het klassediagram wordt getekend als GraphScene in een JScrollpane. Dit wordt geplaatst in een TopComponent venster dat op de plaats van de editor verschijnt. De user stelt namelijk zijn architectuur op of codeert een klasse, maar nooit beiden samen. Dit zijn 2 verschillende processen in softwareontwikkeling. De getekende items worden voorgesteld als widgets zoals in Fig. 31.

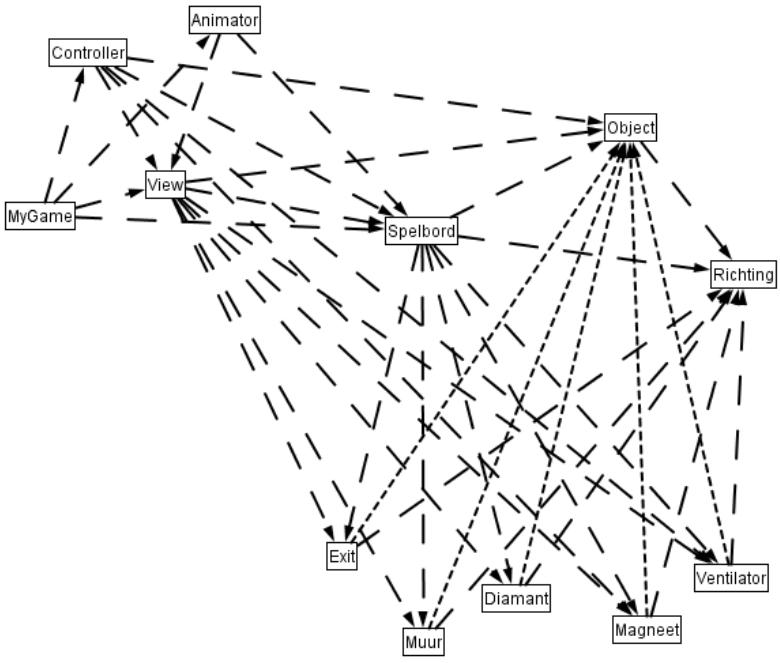

Fig. 31 - Klassediagram view

Wanneer de user het klassediagram opent vraagt dit de nodige informatie op uit de ProjectData module. De view gebruikt deze data om een GraphScene te genereren en hierin nodes en edges te plaatsen. Nodes worden afgebeeld met een LabelWidget en edges met een ConnectionWidget. Deze gadgets worden voorzien door de NetBeans Visual Library.

7.1.1 Bronbestanden

Bronbestanden worden voorgesteld door middel van nodes. Met het commando *Widget.addNode(JavaBestand bestand)* wordt een node aan de scene toegevoegd. De positie van elke node kan ingesteld worden met *Widget.setPreferredLocation(new Point(bestand.getX(), bestand.getY())).* Anderzijds kan deze positie ook worden opgevraagd om op te slaan waar de user de nodes plaatst. De NetBeans Visual Library reikt hiervoor de volgende methodes aan.

```
String xCor = Widget.getLocation().getX();
String yCor = Widget.getLocation().getY();
```

Een node wordt hier gebruikt om een object van de klasse JavaBestand weer te geven. Op deze manier kan de klasse van het Java object worden opgevraagd aan de node. Deze klasse wordt dan geladen om te gebruiken met de Reflection API voor de functionaliteit van de objectenbench.

7.1.2 Relaties

Relaties worden voorgesteld als edges. Dit zijn ConnectionWidgets die een verbinding vormen tussen 2 nodes, eventueel met een pijl op het uiteinde. De relaties worden gegenereerd door de ProjectData module en gebruikt om zowel compositie als erving op te slaan. Dit onderscheid wordt visueel weergeven door de lijnstijl of *stroke* in te stellen voor elk type relatie.

De widget klasse biedt de mogelijkheid om acties aan een node te koppelen. De ActionFactory laat toe om een edge te tekenen met de muis indien de ctrl-toets is indrukt. Hiervoor wordt een custom ConnectionProvider geschreven. Deze wordt aan de widget toegevoegd met de methode *ActionFactory.createExtendedConnectAction()*. De bron- en bestemmingsnode van de edge worden dan opgevraagd uit de view met *edge.getSource()* en *edge.getTarget()* waarna ze worden opgeslagen in de ProjectData module.

8 Objecten bench

De objecten bench gebruik de NetBeans Visual Library als view om objecten en hun methodes te visualiseren. Anderzijds fungeert de bibliotheek als controller om de objecten aan te maken en de methodes uit te voeren. Dit laatste wordt bereikt met de Java Reflection API.

De taken van de Java Reflection API kunnen opgedeeld worden in functies voor klassen en functies voor objecten. Deze sectie geeft hier een overzicht van.

8.1 Klasse

Een klasse is een blauwdruk voor een object. Het object moet gecreëerd worden voordat de methodes gebruikt kunnen worden voor de functionaliteit van de objectenbench. Met een popup menu kan de gebruiker, in het klassediagram, de constructors oproepen en uitvoeren. De bewerkingen die op klassen worden uitgevoerd zijn samengevat in Fig. 32

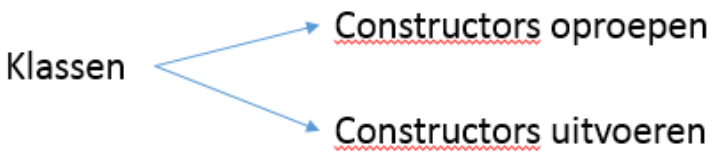

Fig. 32 - Eigenschappen van Java klassen

Constructors kunnen al dan niet parameters bevatten. Dit aantal kan variëren en van diverse types zijn. De parameters worden via een JFrame ingelezen als Strings. Het type van de parameter wordt opgevraagd en de stringwaarde wordt hiernaar geconverteerd.

De constructors die de user heeft aangemaakt worden opgevraagd als een array met *Class.getDeclaredConstructors()*. De nodige constructor wordt dan geselecteerd en, indien aanwezig, worden de types van de parameters opgevraagd met *Constructor.getParameterTypes()*. Het object wordt dan aangemaakt met *Constructor.newInstance(Object… varArgs)*. De parameters worden meegegeven via een variabele van het type varArgs. Dit is een datatype waarin een variabel aantal waarden kan worden opgeslagen van verschillende types.

8.2 Object

Voor de objecten zijn de constructors niet relevant meer. De methodes van het object moeten getoond worden en uitvoerbaar zijn. Op deze manier kan de functionaliteit van een klasse getest worden. De bewerkingen die op objecten worden uitgevoerd zijn samengevat in Fig. 33

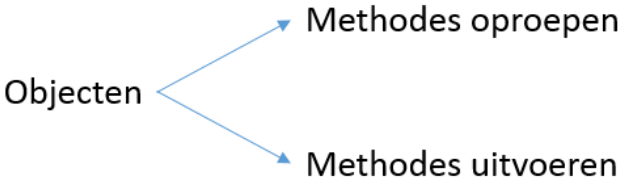

Fig. 33 - Eigenschappen van Java objecten

Om een object te analyseren wordt eerst zijn klasse opgevraagd met *Object.getClass()*. De functie *Class.getDeclaredMethods()* levert de methodes van dit object. Daarna worden de parameters ingelezen en omgezet op dezelfde manier als bij de constructors.

In de objectenbench kunnen meerdere objecten zitten van dezelfde klasse. Bij het uitvoeren van een methode moet daarom worden aangegeven welk object de handeling doet. Dit gebeurt met de functie *Method.invoke(Object doel, Object… params)*.

8.3 Objecten bench view

De view visualiseert alle objecten die zijn opgeslagen in de objecten bench. Deze worden getekend als LabelWidgets in een GraphScene, zoals in het klassediagram.

8.3.1 Rechtermuisknop contextmenu

Om een object te maken van een klasse werd een JPopupMenu gekoppeld aan de nodeWidget in het klassediagram. Dit toont alle constructors die de klasse bezit en hun parametertypes zoals in Fig. 34. Elk van deze constructors wordt voorafgegaan door een index die hun plaats weergeeft in de array van constructoren.

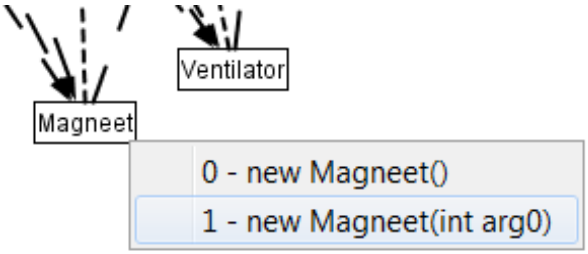

Fig. 34 - Klassediagram popup menu

Indien er geen parameters nodig zijn wordt het object aangemaakt. Wanneer er wel parameters moeten meegegeven worden wordt er een JFrame geopend waarin de gebruiker de waarden kan ingeven. Alle aangemaakte objecten worden opgeslagen in ObjectenBench.class.

8.3.2 JFrame voor constructors en methodes

Indien de gebruiker een methode wil testen, waaraan parameters moeten worden meegegeven, wordt er een JFrame gegenereerd. In dit venster wordt voor elke parameter een JTextField geplaatst waarin de user de waarden kan ingeven zoals in Fig. 35. Deze feature wordt zowel gebruikt om objecten aan te maken als methodes uit te voeren.

Om de verschillende widgets mooi op de frame te zetten moet een Layout Manager gebruikt worden. Gebruik van deze managers wordt verkozen in plaats van absolute positionering omdat men Java platformonafhankelijk wil houden [25]. Daar het aantal parameters op voorhand niet bekend is wordt er gebruik gemaakt van een SpringLayout.

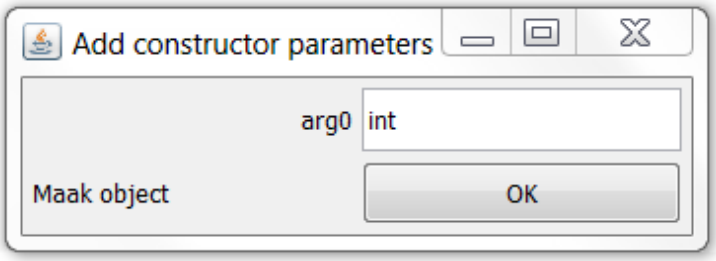

Fig. 35 - JFame om parameters toe te voegen

8.3.3 SpringLayout

Bij het genereren van de JFrame is op voorhand niet bekend hoeveel parameters er moeten ingegeven worden. Daarom moet de grootte van het venster ofwel groot genoeg zijn of veranderen naargelang het aantal waarden.

De SpringLayout is gelijkaardig aan absolute positionering maar reageert beter op veranderingen van de container [26]. Deze layout definieert directionele relaties of constraints tussen de randen van componenten [27]. De positie van een component is dus afhankelijk van de grenzen van andere componenten.

Een tweede voordeel van deze layout is dat met behulp van SpringUtilities de componenten in een grid kunnen geplaatst worden. Voor dit project worden 2 kolommen aangemaakt, 1 voor een label met het type van de parameter en een tweede kolom waar de user de parameters kan invullen in JTextFields. Het aantal rijen wordt bepaald door het aantal parameters die de constructor nodig heeft.

9 Evaluatie klasse

De evaluatieklasse controleert of de broncode conform is met een opgelegde codeerstandaard. Dit wordt mogelijk gemaakt met de CheckStyle API. Deze sectie bespreekt reeds bestaande applicaties en de mogelijkheid om hiermee de evaluatieklasse in NetBeans binnen te brengen.

Oorspronkelijk zou de evaluatieklasse herschreven worden voor het NetBeans IDE. Hierbij zou een nieuwe plugin geschreven moeten worden die de checks overneemt van de bestaande BlueJ extensie. De CheckStyle website verwijst naar een bestaande NetBeans plugin die volledig is geïntegreerd in het IDE en waaraan aangepaste checks kunnen worden toegevoegd.

9.1 BlueJ Plugin

Deze uitbreiding bouwt verder op de masterthesis van Tim Hermans en Raf Marcoen uit 2013 [28]. De plugin gebruikte eerst reguliere expressies om patronen te beschrijven die in code opduiken. Sindsdien is de software aangepast. De extensie gebruikt nu CheckStyle om te controleren op een opgelegde codeerstandaard. Dit zal ook gebruikt worden om deze klasse in NetBeans te implementeren.

Het onderzoek van Hermans en Marcoen legt de fouten die studenten maken tegen codeerconventies vast. Hun resultaten zullen ook gebruikt worden voor de NetBeans versie van de evaluatie-klasse.

Hier wordt een overzicht gegeven van de controles die door deze extensie worden uitgevoerd.

- Aanwezigheid van getters en setters.
- Afscherming van variabelen.
- Aanwezigheid van javadoc.
- Gebruik van CamelCase bij naamgeving van variabelen.
- Fouten tegen if-statements.
    - Variabele toekennen i.p.v. vergelijken (“=” i.p.v. “==”).
    - Boolean vergelijken met Boolean (vb. isLeef==true).
    - String controleren met “==” i.p.v. “.equals()”.

De broncode van dit project werd ter beschikking gesteld en geanalyseerd. De plugin maakt gebruik van resources specifiek voor BlueJ om een extensie in het IDE te integreren. Deze resources zijn niet beschikbaar in NetBeans.

Alternatief verwijst CheckStyle naar de CheckStyle Beans plugin voor het NetBeans IDE. Deze uitbreiding laat toe om aangepaste checks toe te voegen. Er wordt nagegaan of de checks naar deze extensie kunnen overgezet worden en hier een scoresysteem aan gekoppeld kan worden.

9.2 CheckStyle Beans Plugin

De website van CheckStyle verwijst naar een bestaande extensie die CheckStyle integreert in het NetBeans IDE. Deze plugin is gemaakt door Petr Hejl onder de GNU Lesser General Public License versie 2.1. Problemen met de broncode worden weergeven als annotaties in de editor [29] indien het bestand wordt opgeslaan. Dit wordt geïllustreerd in Fig. 36.

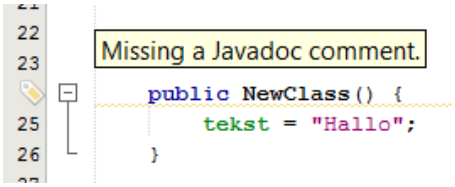

Fig. 36 - CheckStyle Beans Plugin annotatie

Een optie menu is ingebouwd in het IDE. Dit maakt het mogelijk om aan te passen welke checks er worden uitgevoerd door een aangepast configuratiebestand te laden.

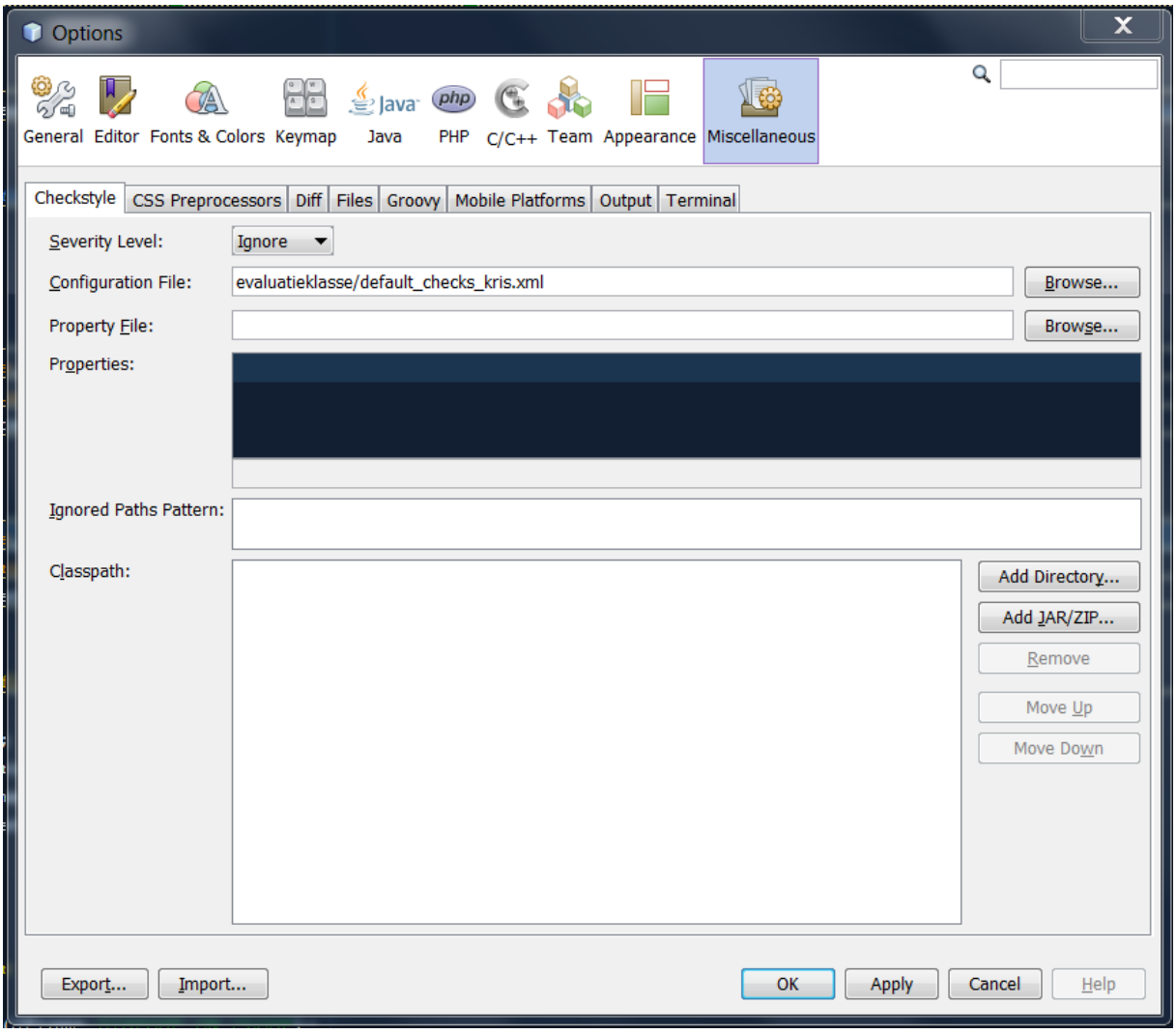

Fig. 37 - CheckStyle Beans Plugin optie menu

10 Besluit

Met deze masterthesis werd een extensie gerealiseerd die een beginnend Java programmeur helpt om de overstap te maken van BlueJ naar het NetBeans IDE.

Deze plugin analyseert de projectstructuur met de Java Reflection API. De NetBeans Visual Library tekent dan het klassediagram met de verkregen informatie. De objecten bench gebruikt de Java Reflection API om objecten aan te maken en hun methodes uit te voeren.

Deze extensie werd opgebouwd volgens het model-view-controller principe. De functionaliteit van de plugin zit in een aparte module. De NetBeans Visual Library construeert de GUI die zowel de view als de controller bevat. Tabel 7 geeft een overzicht van de APIs die elke feature gebruikt.

Tabel 7 - Kruistabel met features en APIs

	Java Reflection API	NetBeans Visual Library	CheckStyle API
Projectdata analyseren	X		
Klassediagram tekenen		X	
Objectenbench uitvoeren	X	X	
Evaluatieklasse uitvoeren		X	X

Met deze uitbreiding willen we de overgang naar het NetBeans IDE vergemakkelijken door de vertrouwde didactische tools aan te bieden. Door het integreren van het klassediagram wordt de structuur van een project gevisualiseerd. Dit doet de student nadenken over de opbouw van zijn programma. De objecten bench laat toe om individuele methodes te testen zonder dat het hele programma gecompileerd moet worden. Hierdoor kan de student zich richten op het opstellen van de klassen zonder dat hij code moet schrijven om deze uit te proberen.

Bestuderen van de evaluatieklasse en de plugin van P.Hejl geeft aan dat het mogelijk is om de checks van de evaluatieklasse in de CheckStyle Beans plugin te gebruiken. Op deze manier moet er enkel een scoresysteem aan de checks gekoppeld worden.

We hopen dat deze plugin in gebruik wordt genomen door studenten en in ontwikkeling blijft. De BlueJ functionaliteit draagt bij tot ontwikkeling van een betere software architectuur. De codeerstandaardisering bevorderd de leesbaarheid van de code zodat deze beter te onderhouden is en makkelijker verstaanbaar.

Literatuurlijst

[1] „Structured programming,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Structured_programming. [Geopend 15 11 2016].

[2] „Paradigms,” wikibooks.org, [Online]. Available: https://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages/Paradigms. [Geopend 15 11 2016].

[3] M. W. a. S. Northover, „Java Widget Fundamentals,” InformIT.com, 28 01 2005. [Online]. Available: http://www.informit.com/articles/article.aspx?p=354574. [Geopend 15 11 2016].

[4] „NetBeans Nodes API Tutorial,” NetBeans.org, [Online]. Available: https://platform.netbeans.org/tutorials/nbm-nodesapi2.html. [Geopend 15 11 2016].

[5] „Class GraphScene<N,E>,” NetBeans, [Online]. Available: http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/graph/GraphScene.html.

[6] „BlueJ,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/BlueJ.

[7] „About BlueJ,” BlueJ, [Online]. Available: https://www.bluej.org/about.html.

[8] J. M.Kölling, „BlueJ - The Hitch Hiker's Guide to Object Orientation,” Kent Academic Repository, [Online]. Available: https://kar.kent.ac.uk/13735/1/BlueJMike.pdf.

[9] „NetBeans,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/NetBeans#NetBeans_Platform.

[10] W. N. J. Wexbridge, NetBeans Platform for beginners, Leanpub, 2014.

[11] P. Creek, „Java Reflection Tutorial,” Program Creek, [Online]. Available: http://www.programcreek.com/2013/09/java-reflection-tutorial/.

[12] „CheckStyle,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Checkstyle.

[13] „Visual Library 2.0 - Documentation,” NetBeans, [Online]. Available: http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html#Introduction.

[14] „NetBeans Visual Library,” Netbeans, [Online]. Available: https://platform.netbeans.org/graph/.

[15] A. Avram, „How to Do Just Enough Up-front Design,” InfoQ, 09 05 2014. [Online]. Available: https://www.infoq.com/news/2014/05/minimal-architecture-design.

[16] „UML - Component Diagrams," tutorialspoint, [Online]. Available: https://www.tutorialspoint.com/uml/uml_component_diagram.htm.

[17] Mark, „.class vs .java," Stack Overflow, [Online]. Available: https://stackoverflow.com/questions/1015340/class-vs-java.

[18] „WindowsTopComponent," NetBeans, [Online]. Available: http://wiki.netbeans.org/DevFaqWindowsTopComponent.

[19] „FileObject," NetBeans, [Online]. Available: https://netbeans.org/download/5_5/org-openide-filesystems/org/openide/filesystems/FileObject.html.

[20] D. Lowe, „Local variables in Java," Dummies, [Online]. Available: http://www.dummies.com/programming/java/local-variables-in-java/.

[21] S. C., „Can I get information about the local variables using Java reflection?," Stackoverflow, [Online]. Available: https://stackoverflow.com/questions/6816951/can-i-get-information-about-the-local-variables-using-java-reflection.

[22] „ObjectClass," Oracle, [Online]. Available: https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html.

[23] „Dynamic Class Loading using Java Reflection API," viralpatel, [Online]. Available: http://viralpatel.net/blogs/java-dynamic-class-loading-java-reflection-api/.

[24] L. R. Viggiano, „How to convert from String to a primitive type or standard java Wrapper types," StackOverflow, [Online]. Available: https://stackoverflow.com/a/13949291/4874173.

[25] D. K. Aerts, Informatica 2: Grafische applicaties in Java (GaJa).

[26] „How to Use SpringLayout," Department of computer science, [Online]. Available: http://www.comp.hkbu.edu.hk/docs/j/tutorial/uiswing/layout/spring.html.

[27] „How to Use SpringLayout," Oracle, [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html.

[28] R. T.Hermans, Realisatie van een BlueJ-extensie voor de zelfevaluatie van een klasse, 2013.

[29] P. Hejl, „CheckStyle," SickBoy.cz, [Online]. Available: http://www.sickboy.cz/checkstyle.

[30] „Manifest file," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Manifest_file.

[31] „Nodes API javadoc," NetBeans, 23 October 2015. [Online]. Available: http://bits.netbeans.org/8.1/javadoc/org-openide-nodes/org/openide/nodes/doc-files/api.html. [Geopend 23 Juli 2016].

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Ontwikkeling van een BlueJ-getinte uitbreiding voor NetBeans

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2017**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Boutsen, Stijn

Datum: **6/06/2017**