Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Lennart Borchert

Entwicklung eines Migrationsleitfadens zum Wechsel der Extension-API einer Lehr-IDE

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Lennart Borchert

Entwicklung eines Migrationsleitfadens zum Wechsel der Extension-API einer Lehr-IDE

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Axel Schmolitzky
Zweitgutachter: Prof. Dr. Michael Schäfers

Eingereicht am: 24. August 2018

Lennart Borchert

Thema der Arbeit
Entwicklung eines Migrationsleitfadens zum Wechsel der Extension-API einer Lehr-IDE

Stichworte
Java, BlueJ, BlueJVisualize, Swing, JavaFX, Erweiterung, Schnittstelle

Kurzzusammenfassung
Diese Arbeit befasst sich mit den Änderungen die BlueJ, mit dem Sprung zur Version 4.0.0, durchgemacht hat und wie diese sich auf dessen Erweiterungsschnittstelle auswirken. Es wird die aktuelle Entwicklung von BlueJ unter die Lupe genommen um festzustellen warum interne Änderungen Einfluss auf die Erweiterungsschnittstelle nehmen und was die Konsequenz für Erweiterungen daraus ist. Die, bisher in BlueJ eingebettete, Erweiterung BlueJVisualize wird untersucht, um festzustellen ob sie über die Erweiterungsschnittstelle anbindbar ist. Außerdem wird darauf eingegangen wie die Erweiterungsschnittstelle modifiziert werden müsste um: Sich von den Einflüssen der internen Änderungen zu befreien und die Anbindung von BlueJVisualize und ähnlichen Erweiterungen zu erlauben. Abschließend wird über die Möglichkeiten gesprochen BlueJVisualize weiter zu entwickeln und auf was hierbei geachtet werden müsste.
Lennart Borchert

Title of the paper
Development of a migratotry guide for the change of the extension-api of a teaching-ide

Keywords
Java, BlueJ, BlueJVisualize, Swing, JavaFX, Extension, Interface

Abstract
This thesis deals with the changes that BlueJ experienced with its jump to version 4.0.0 and how these affect its Extension-API. The current development of BlueJ will be examined to see why internal changes affect the extension interface and what the consequence for its Extensions is. The, previously in a modified version of BlueJ embedded, Extension BlueJVisualize is examined to see if it can be implemented via the Extension-API. In addition, it is reviewed how the Extension-API would have to be modified for: Freeing it from the influences of the internal changes and to allow connecting BlueJVisualize and similar extensions through it. Finally, possibilities to continue development of BlueJVisualize and its constraints are discussed

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeauszüge

1. Einleitung

Diese Bachelorarbeit soll sich damit befassen die Veränderungen die BlueJ mit dem Versionswechsel zur Version 4.0.0 durchgemacht hat zu untersuchen. Hierbei wird besondere Aufmerksamkeit auf Änderungen an der Extension-API, im Folgenden Erweiterungsschnittstelle genannt, gelegt um daraus resultierende Fehler sowie notwendige Änderungen an Erweiterungen aufzudecken.
Desweiteren soll überprüft werden ob die Erweiterung BlueJVisualize, welche im Rahmen von Beckert (2013) entwickelt wurde und bis jetzt direkt in den BlueJ Quellcode eingebettet ist, unter der neuen Version stattdessen über die Erweiterungsschnittstelle angebunden werden kann.
In Abschnitt 1.2 & Abschnitt 1.3 wird kurz darauf eingegangen was BlueJ und BlueJVisualize sind und wodurch sie sich auszeichnen.

1.1. Motivation

BlueJ bietet über dessen Erweiterungsschnittstelle die Möglichkeit ein für die Lehre hilfreiches Tool um weitere spezielle Funktionalitäten zu erweitern. Nach dem Versionswechsel von BlueJ auf Version 4.0.0 sind allerdings Fehler bei der Verwendung von einigen Erweiterungen zu beobachten. Die beobachteten Fehler waren hauptsächlich Darstellungsfehler, stellen aber die Frage auf ob sich mit dem Update andere Fehler, insbesondere an der Erweiterungsschnittstelle, eingeschlichen haben. Um die reibungslose Nutzung von BlueJ und dessen Erweiterungen weiterhin zu ermöglichen müssen diese Fehler aufgedeckt und beseitigt werden.
Die in Beckert (2013) entwickelte Erweiterung BlueJVisualize bietet einen deutlichen Mehrwert für BlueJ in der Programmierlehre, es kann aber zur Zeit nur beschränkt eingesetzt werden da es an den Code einer bestimmten BlueJ Version gebunden ist und nur mit dieser verteilt werden kann. Deshalb sollte geprüft werden ob es in BlueJ 4.0.0 möglich ist BlueJVisualize aus dem internen Code zu extrahieren und stattdessen über die Erweiterungsschnittstelle anzubinden.

1.2. Was ist BlueJ?

Abbildung 1.1.: Das Logo von BlueJ

Quelle: BlueJ (2018a)

BlueJ ist eine seit 1999 entwickelte integrierte Entwicklungsumgebung(IDE) für Java, die speziell mit Rücksicht auf die Programmierlehre konzipiert ist. Hierbei setzt BlueJ insbesondere auf Interaktivität mit dem Benutzer sowie Visualisierung der Abhängigkeiten zwischen Klassen und erzeugten Objekten. Unter anderem bietet BlueJ eine auf der UML basierende Ansicht der im Projekt vorhandenen Klassen und deren Beziehungen untereinander. Es können interaktiv vom Benutzer Objekte dieser Klassen erzeugt, deren Zustand untersucht und Methoden aufgerufen werden. Außerdem bietet BlueJ ein CodePad in welchem Java Ausdrücke wie in einer Konsole eingegeben und direkt ausgewertet werden können. Zusammen mit dem leicht zu verwendenden Editor und der Unterstützung für die meisten gängigen Betriebssysteme bietet BlueJ eine gute Plattform für die Programmierlehre in Java. Mehr Informationen können auf der Webseite von BlueJ Kölling und BlueJ Team (2018a) gefunden werden.

1.3. Was ist BlueJVisualize?

BlueJVisualize ist eine im Rahmen von Beckert (2013) prototypisch entwickelte Erweiterung für BlueJ. Sie dient dazu die Ausdrucksauswertung in Java kleinschrittig zu visualisieren und damit den Verständnisprozess zu fördern. Hierzu bindet BlueJVisualize an die Direkteingabe des CodePad in BlueJ an, wo ohnehin mit kontextfreien Ausdrücken experimentiert werden kann und ermöglicht über ein eigens festgelegtes Tastenkürzel den Aufruf von BlueJVisualize anstelle der normalen Auswertung des eingegebenen Ausdrucks. Anschließend kann die Auswertung des Ausdrucks Schritt für Schritt beobachtet, nachvollzogen und falls notwendig auch zurückgedreht werden. Somit ist ein Verständnis dieses sonst nicht sichtbaren Prozesses deutlich leichter zu erreichen, und bietet ein nützliches Hilfmittel in der Programmierlehre.
Allerdings ist BlueJVisualize bisher nur eingeschränkt verwendbar, da es in den Quellcode von BlueJ selbst eingebaut wurde und nur in dieser modifizerten Version verteilt werden kann. Dazu kommt, dass es keine freistehende Ausführung dieser modifzierten BlueJ Version gibt; lediglich ein in Eclipse lauffähiges Projekt mit dem modifzierten Quellcode ist vorhanden. Mit dem zusätzlichen Aufwand dieses Projekt zuerst in Eclipse zum Laufen zu bringen, ist die Verwendung durch Lernende momentan nicht empfehlenswert. BlueJVisualize kann von

Lehrenden verwendet werden für die der Einrichtungsaufwand des Projekts leichter zu bewältigen ist, aber zur effektiven Nutzung muss die einfache Verwendung für den Lernenden selbst gewährleistet werden.

1.4. Vorgehen

Die durch den Versionssprung entstandenen Veränderungen können zunächst grob an den Versionsnotizen unter Kölling und BlueJ Team (2018b) abgelesen, weiterführend an der Spezifikation der Erweiterungsschnittstelle unter Kölling und BlueJ Team (2018c) und falls notwendig am Quellcode erarbeitet werden. Hierbei soll insbesondere auf Änderungen an der Erweiterungsschnittstelle geachtet werden, da es bei Fehlern in den Erweiterungen nahe liegt, dass hier eine Änderung die beobachteten Fehleffekte hervorruft. Bestehenden Erweiterungen werden untersucht um eine eventuelle Fehlerquelle zu identifizieren und Fehler in Zukunft zu vermeiden.
Um festzustellen ob BlueJVisualize vom internen Code getrennt werden kann müssen zuerst einmal dessen Abhängigkeiten von BlueJ internen Klassen offen gelegt werden, um im Anschluss prüfen zu können ob die Erweiterungsschnittstelle die benötigten Funktionalitäten bereitstellt. Das Erarbeiten der Abhängigkeiten überschneidet sich teilweise mit dem Einblick in den BlueJ Quellcode und kann daher gleichzeitig auch dazu dienen die Änderungen des Versionswechsels zu untersuchen.

1.5. Abgrenzung

In der Arbeit soll es nicht darum gehen für jede für BlueJ vorhandene Erweiterung eine individuelle Anleitung zur Migration zu verfassen. Es werden lediglich Fehler aufgedeckt, dokumentiert und, wo möglich, Lösungsvorschläge gegeben. Ebenso sollen nur Fehler betrachtet werden, die durch den Versionswechsel entstanden sind. Sollten Fehler aus vorherigen Versionen vorhanden sein werden diese nicht thematisiert. Sollte durch die Arbeit ein Missstand am Quellcode von BlueJ festgestellt werden so ist es nicht Bestandteil der Arbeit, über Lösungsvorschläge hinaus, diesen zu beheben.

1.6. Aufbau

Zuerst werden die Veränderungen die BlueJ von Version 3.1.0 bis Version 4.1.2[1] durchlaufen hat in Kapitel 2 auf Seite 5 untersucht. Hierbei wird insbesondere auf Änderungen an der

Erweiterungsschnittstelle eingegangen und erklärt was diese für die Erweiterungen von BlueJ bedeuten.

Anschließend wird in Kapitel 3 auf Seite 8 auf den aktuellen Status der Erweiterungsschnittstelle und Erweiterungen eingegangen und der existierende Bestand an Erweiterungen für BlueJ auf Fehler untersucht die infolge des Versionswechsels entstanden sein können. Hierbei werden mögliche Lösungsvorschläge geliefert.

Danach wird BlueJVisualize in Kapitel 4 auf Seite 11 noch einmal gesondert untersucht um festzustellen, ob es möglich ist es in der neuen Version aus BlueJ zu extrahieren und über die Erweiterungsschnittstelle anzubinden. Hier wird außerdem auf den Modernisierungsprozess von BlueJVisualize eingegangen.

Abschließend wird ein Fazit über die Entwicklung von BlueJ und die Konsequenzen für dessen Erweiterungen gegeben und mögliche Folgearbeiten in Kapitel 5 auf Seite 21 beschrieben.

[1] Obwohl es in dieser Arbeit hauptsächlich um die Änderungen durch BlueJ Version 4.0.0 geht wurden zum Vergleich bewusst die Versionen 3.1.0 & 4.1.2 herangezogen. 3.1.0 da es die Version von BlueJ ist in die BlueJVisualize eingebettet wurde und es keine starken Veränderungen zwischen 3.1.0 und 3.1.7 (Letzte Version vor 4.0.0) gab. 4.1.2 wurde gewählt da es die zum Zeitpunkt dieser Arbeit aktuelle Version von BlueJ ist und es noch keinen großen Versionssprung nach 4.0.0 gab. Somit können Lösungsvorschläge gleich für den aktuellen Stand gegeben werden und es müssen weniger Versionen betrachtet werden.

2. Aktuelle Entwicklung von BlueJ

BlueJ wird konstant von einem Vollzeitteam an Entwicklern weiterentwickelt, so gab es bereits mehrere weitere Versionen seit BlueJ Version 4.0.0 Anfang 2017 erschienen ist. Die aktuelle Version zur Zeit dieser Arbeit ist 4.1.2, an welcher nun die Veränderungen seit Version 3.1.0 aufgezeigt werden.

2.1. Versionssprung auf 4.0.0

Mit der Veröffentlichung von BlueJ Version 4.0.0 wurde der erste große Versionssprung in über 5 Jahren getätigt. Die Versionsnotiz[1] deckt die Veränderungen grob schon auf.

2.1.1. Hintergrund des Versionswechsel

Wohl die größte Änderung die BlueJ mit Version 4.0.0 durchgemacht hat, ist der Umstieg von Swing zu JavaFX als hauptsächlichen Werkzeugsatz für die grafische Oberfläche. Dies ist auch die auffälligste Veränderung wenn eine neue Version[2] von BlueJ verwendet wird. Alle Oberflächen wurden erneuert und deren Design modernisiert, BlueJ wirkt dadurch insgesamt moderner. Diese Veränderung hat aber vermutlich auch großen Einfluss auf die Interna von BlueJ, da in der Vergangenheit die Verbindung zu Swing im Quellcode von BlueJ[2] deutlich zu sehen war.
Eine weitere große Änderung, welche aber nur einige Teile von BlueJ betrifft, ist die Möglichkeit neuerdings auch Klassen in Stride[3]zu schreiben. Somit können jetzt nicht mehr nur reine Java Projekte in BlueJ verwaltet werden, sondern auch reine Stride Projekte oder sogar eine Mischung. BlueJ bietet auch die Möglichkeit jederzeit von einer Sprache in die andere zu übersetzten, da Stride stark an Java angelehnt ist.
BlueJ unterstützt schon seit langer Zeit Subversion als Möglichkeit für Versionsverwaltung

[1] Zu finden unter Kölling und BlueJ Team (2018b)
[2] Alle Versionen ab 4.0.0
[3] Stride ist eine Rahmen-basierte Sprache welche zuerst in Greenfoot zum Einsatz kam und nun ihren Sprung zu BlueJ getan hat. Wer mehr erfahren möchte findet Information hier: Für StrideKölling u. a. (2018) oder das Thema der Rahmen-basierten Bearbeitung in Programmiereditoren allgemein Kölling u. a. (2017)

und kollaborative Arbeit. Mit Version 4.0.0 ist zusätzlich Unterstützung für Git dazugekommen, was den Nutzern somit mehr Auswahl bietet. Die Unterstützung von Git sollte sonst allerdings keinen weiteren Einfluss auf andere Teile von BlueJ nehmen.
Weiter gibt es noch einige Funktionserweiterungen wie z. B. die Möglichkeit mehrere Reiter in einem Editorfenster zu haben und die Anzeige mehrerer Fehler gleichzeitig. Diese Änderungen nehmen aber vermutlich so wie die Git-Unterstützung keinen weiteren Einfluss auf andere Teile von BlueJ.

2.1.2. Welche Probleme sind entstanden

Offensichtlich ist zumindest ein Teil der Veränderung den bestehenden Erweiterungen nicht bekommen. Fehler in der Darstellung deuten darauf hin, dass sich entweder etwas an der Erweiterungsschnittstelle geändert hat, oder aber interne Änderungen sich auf die Erweiterungen niederschlagen ohne das dies gewollt war. Dies scheint den BlueJ Entwicklern allerdings auch bewusst zu sein, da in der Versionsnotiz zu 4.0.0[4] darauf hingewiesen wird, dass einige Erweiterungen eventuell nicht mehr funktionieren könnten, allerdings wird der Hintergrund dieser Aussage nicht weiter erläutert. Nach weiterer Untersuchung ergibt sich, dass durch den Wechsel zu JavaFX einige Teile der Erweiterungsschnittstelle gebrochen sind, da diese so entworfen wurde, dass sie auf Swing Klassen angewiesen ist. So ist es verständlich, dass nun wo der Großteil von BlueJ zu JavaFX migriert wurde es Schwierigkeiten bei der Abbildung zwischen BlueJ und der Erweiterungsschnittstelle gibt, da die Erweiterungsschnittstelle darauf aufbaut das Swing Elemente verwendet werden wo jetzt stattdessen JavaFX Elemente existieren.

2.1.3. Was hat sich nicht verändert

Interessanterweise hat sich kaum etwas an der Erweiterungsschnittstelle verändert. Die Methode BPackage#getFrame wurde veraltet, ist allerdings noch vorhanden und die Methode BProject#openWebViewTab wurde hinzugefügt. Basierend auf diesen Veränderungen wäre keine spürbare Veränderung bei den Erweiterungen zu erwarten. Die Anwendung von BlueJ ist größtenteils unverändert geblieben. Fast alle Elemente die vorher in der Oberfläche zu finden waren, sind weiterhin vorhanden, wenn auch leicht verändert.

[4] Unter Kölling und BlueJ Team (2018b) zu finden

2.2. Erweiterungen für BlueJ

Für BlueJ existiert eine ganze Reihe an Erweiterungen[5], welche es um diverse Funktionen bereichern. Viele davon wurden von Nutzern entworfen und entwickelt, was zeigt, dass auch von dieser Seite aus ein Interesse daran besteht BlueJ zu erweitern und weiter zu entwickeln. Allerdings findet sich neben den veröffentlichten Erweiterungen auch der Hinweis, dass mit dem Versionswechsel zu BlueJ 4.0.0 und dem Umbau der Oberfläche auch an den internen Schnittstellen viel verändert wurde und die Erweiterungsschnittstelle aus diesem Grund nicht mehr zuverlässig funktioniert. Es wird in Aussicht gestellt, dass die Erweiterungsschnittstelle neu entworfen werden muss und Erweiterungen anschließend zu der neuen Erweiterungsschnittstelle migriert werden müssen.
Somit ist zumindest klar warum Darstellungsfehler in der Verwendung von Erweiterungen auftreten und es ist sogar überraschend das die Funktion teilweise noch gegeben ist.

[5] Die Sammlung der veröffentlichen Erweiterungen ist unter BlueJ (2018b) zu finden.

3. Stand der Erweiterungsschnittstelle und ihrer Erweiterungen

Wie im vorherigen Kapitel festgestellt wurde sieht es von Entwicklerseite schlecht für den aktuellen Zustand der Erweiterungsschnittstelle aus. Es wird geraten damit zu rechnen das Erweiterungen Fehler produzieren, oder gar nicht funktionieren. Unter diesen Umständen kann vermutlich keine Änderung von Seiten der Erweiterungen Abhilfe schaffen, es kann nur versucht werden den defekten Teilen der Erweiterungsschnittstelle auszuweichen und stattdessen mit der verringerten Funktionalität auszukommen. Im Folgenden soll festgestellt werden zu welchem Grad die Erweiterungen von BlueJ unter diesen Bedingungen nutzbar sind.

3.1. Untersuchen der Erweiterungen

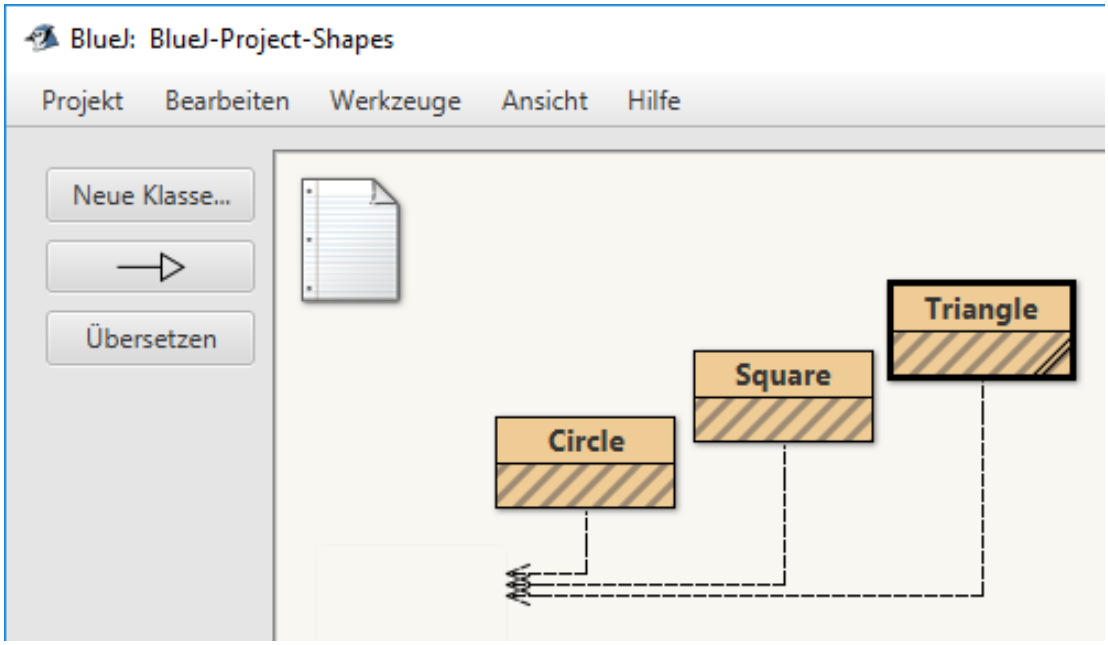
Es wurde eine Reihe an Erweiterungen auf ihre Funktion überprüft. Hierfür wurden diese sowohl in BlueJ Version 3.1.0 als auch Version 4.1.2 unter ansonsten gleichen Bedingungen auf ihre fehlerfreie Funktion überprüft. Hierbei wurde differenziert darauf geachtete ob die beobachteten Fehler rein in der Darstellung liegen, oder ob sie auch die Funktion der Erweiterung einschränken.

3.1.1. Beobachtungsergebnisse

Die Ergebnisse sind überraschen, da sie nicht mit den Aussagen der BlueJ Entwickler einher gehen: Von den getesteten Erweiterungen hatte nur eine einzige Fehler und selbst diese beschränkten sich nur auf die Darstellung. Hier wurde entweder bereits vieles behoben oder die Auswirkungen der internen Änderungen auf die Erweiterungsschnittstelle wurden weit überschätzt.

Abbildung 3.1.: Darstellungsfehler bei der Verwendung von BlueJVisualize

Quelle: Eigene Darstellung



3.1.2. Aufgedeckte Fehler

Ein Fehler konnte in der Oberfläche von BlueJ selbst ausgemacht werden: Das Erweiterungsmenü[1] kann nur schlecht mit Größenveränderungen des Fensters umgehen. Wird die Größe des Fensters verändert so verschwindet der Inhalt innerhalb des Reiters Erweiterungen. Übergangslösungen sind es entweder die Größe des Fensters nicht zu verändern oder, oder anschließend das Fenster an eine andere Position zu ziehen, womit die neuzeichnung des Inhalts ausgelöst wird. Dieser Fehler entsteht vermutlich, da die entsprechende Stelle der Erweiterungsschnittstelle so aufgebaut ist das sie Swing Elemente von den Erweiterungen fordert, welche hier in die JavaFX Oberfläche eingebunden werden müssen.

Außerdem gibt es Darstellungsfehler in Verbindung mit der Erweiterung "Foldable Class Diagram"(FCD)[2]. Es versteckt auf Wunsch die Klassen von denen eine Klasse abhängig ist (Schnittstellensicht) aus der Oberfläche. In der aktuellen Version ist dies zwar möglich, allerdings werden die Abhängigkeitspfeile zu den ausgeblendeten Klassen weiterhin angezeigt. Diese sollten zusammen mit den Klassen zu denen sie gehören ausgeblendet werden. Ebenso sollte eine Verfärbung die Klassen kennzeichnen deren Abhängigkeiten im Moment ausge-

[1] Zu finden in der Oberfläche von BlueJ unter Werkzeuge → Einstellungen → Erweiterungen

blendet werden, dies geschieht aktuell ebenso nicht. Weitere Fehler sind wider Erwarten nicht aufgetreten.

3.1.3. Umgang mit fehlerhafter Erweiterungsschnittstelle

Wie zuvor festgestellt gehen die beobachteten Fehler vermutlich von der Erweiterungsschnittstelle aus, weshalb innerhalb der Erweiterungen selbst leider keine hilfreichen Änderungen vorgenommen werden können. Hier kann auf den Missstand, solange er besteht, hingewiesen werden, um die Benutzer aufzuklären; weshalb in der Verwendung der Erweiterungen Fehler auftreten können. Die Ergebnisse deuten allerdings darauf hin, dass die Fehler zusammenhängen mit grafischen Elementen welche über die Erweiterungsschnittstelle manipuliert werden sollen; Erweiterungen die diese nicht verwenden sollten ohne Schwierigkeiten weiter verwendbar sein.

3.2. Aktueller Stand der Erweiterungsschnittstelle

Die Erweiterungsschnittstelle ist momentan in einer unsicheren Position: Anscheinend konnten die meisten Fehler die in Verbindung mit dem Wechsel von Swing zu JavaFX entstanden sind behoben werden, aber Teile der Erweiterungsschnittstelle sind so geschrieben, dass sie auf Elemente von Swing angewiesen sind.[3] Außerdem ist ein Neuentwurf der Erweiterungsschnittstelle im Zusammenhang mit dieser Problematik bereits von den BlueJ Entwicklern in Betracht gezogen worden.[4] Für Erweiterungen und deren Entwicklung bedeutet dies, dass vermutlich über kurz oder lang eine Migration zu einer angepassten Erweiterungsschnittstelle notwendig ist, sofern der Wechsel von Swing zu JavaFX vollständig durchzogen werden soll.

[2] Nicht in der BlueJ Erweiterungssammlung zu finden, wurde im Rahmen von Gerlach (2012) entwickelt.
[3] Zu sehen in Anhang A
[4] BlueJ Team (2018)

4. BlueJVisualize

Im Folgenden werden die Abhängigkeiten die BlueJVisualize zum BlueJ Quellcode hat offen gelegt und geklärt ob diese auch über die Erweiterungsschnittstelle abgedeckt werden können. Offensichtlich wird hier auf BlueJ internen Code eingegangen, es wird aber auf eine vollständige Erläuterung der einzelnen Klassen verzichtet und lediglich auf die relevanten Bestandteile eingegangen. Falls ein weiterführendes Interesse am BlueJ Quellcode besteht ist dieser über die BlueJ Versionsseite Kölling und BlueJ Team (2018b) zu finden.

4.1. Abhängigkeiten von internem Code

Die Erweiterung BlueJVisualize ist in der prototypischen Version wie sie in Beckert (2013) entworfen wurde von mehreren BlueJ internen Klassen abhängig. Die Abhängigkeiten zu den einzelnen Klassen sind wie folgt:

NamedValue Ein Interface das nur benötigt wird um die von BlueJ hinterlegten Daten, zu den im Ausdruck verwendeten Variablen, entgegenzunehmen.

ValueCollcetion Ein weiteres Interface das wie der Name suggeriert eine Sammlung an Objekten des zuvor genannten Typen NamedValue darstellt.

Invoker Der Invoker kümmert sich um Aufrufe die in BlueJ getätigt werden, im Fall von BlueJVisualize wird er dazu verwendet die Werte zu den im Ausdruck vorhandenen Variablen aufzulösen. Somit bietet der Invoker eine Funktionalität die zwingend erforderlich ist für die Funktion von BlueJVisualize.

TextAnalyser Wird dazu verwendet zu einem gegebenen Variablennamen den Variablentyp in Erfahrung zu bringen. Auch dies ist für die Funktion von BlueJVisualize zwingend erforderlich.

CodePad(TextEvalPane) [1,2] Ist BlueJVisualize zu dem Zweck bekannt um Rückmeldung über den Erfolgszustand oder bei Fehlern eine individualisierten Fehlermeldung zu liefern.

PkgMgrFrame Der PkgMgrFrame, obwohl zentraler Teil von BlueJ wird nur benötigt um eine Instanz des Invoker und TextAnalyzer zu erzeugen.

DebuggerObject, ExceptionDescription, InvokerRecord, ResultWatcher Typen die überwiegend aus syntaktischen Gründen vorhanden sein müssen um die Ergebnisse des Invoker zu erhalten und in BlueJVisualize verarbeiten zu können. Sie werden als mit dem Invoker zusammenhängend behandelt und deshalb nicht weiter einzeln erwähnt.

Nun kann unterschieden werden zwischen den Klassen deren Funktionalität, für eine Extraktion, über die Erweiterungsschnittstelle abgebildet werden müssten und den Klassen die intern nur als Sammlungen für Werte verwendet oder lediglich zur Verwendung Ersterer benötigt werden. So können die kritischen Typen von den ersetzbaren getrennt werden und deutlich gemacht werden welche Anforderungen die Erweiterungsschnittstelle erfüllen muss.
Dies ist zu sehen in Tabelle 4.1

Tabelle 4.1.: Aufteilung der Klassen von denen BlueJVisualize abhängig ist

Quelle: Eigene Darstellung

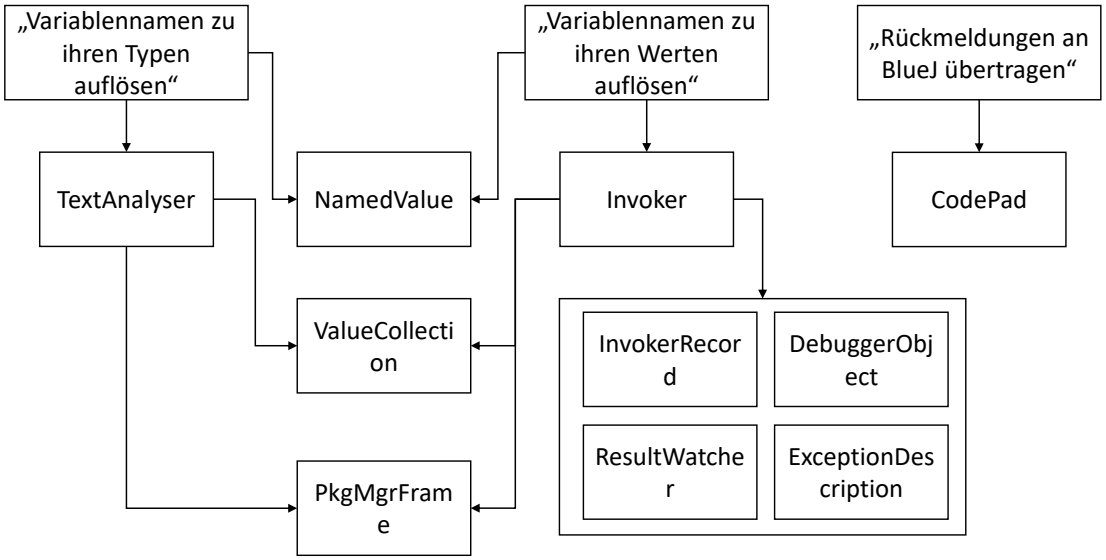
Kritisch	Ersetzbar
Invoker	NamedValue
TextAnalyser	PkgMgrFrame
	ValueCollection

Leider bietet die aktuelle Erweiterungsschnittstelle nicht die notwendige Funktionalität an, welche benötigt wird um BlueJVisualize darüber anzubinden. Es besteht aktuell keine Möglichkeit für Erweiterungen mit der Direkteingabe im CodePad zu interagieren oder auf Variablen und Objekte zuzugreifen die interaktiv vom Benutzer erzeugt wurden. Hier wäre eine Erweiterung der Erweiterungsschnittstelle notwendig auf die in 4.3 eingegangen wird.
Nichtsdestotrotz kann BlueJVisualize auf die Extraktion vorbereitet werden, indem es aktualisiert und an die neue BlueJ Version angepasst wird.

[1] Das CodePad ist ohnehin mit BlueJVisualize verbunden, da hier die Eingabe der Ausdrücke und der, die Visualisierung startende, Aufruf erzeugt wird. Aber momentan muss BlueJVisualize auch eine Referenz zu dem es erzeugenden CodePad halten um darüber Fehlermeldungen an den Nutzer zu übertragen.
[2] In BlueJ Version 3.1.0 hieß das CodePad intern noch TextEvalPane, weshalb auch die Abhängigkeit eigentlich so benannt sein müsste; der Verständnis halber habe ich mich dagegen entschieden.

Abbildung 4.1.: Abhänigigkeiten von BlueJVisualize

Quelle: Eigene Darstellung



4.2. Modernisieren von BlueJVisualize

Um BlueJVisualize von BlueJ Version 3.1.0 zu Version 4.1.2 zu migrieren sind einige Schritte notwendig:
Zuerst werden die Abhängigkeiten von BlueJVisualize gegenüber dem BlueJ Quellcode minimiert um den Aufwand beim Einbau in die neue Version und einer eventuell späteren Extraktion gering zu halten.
Anschließend werden parallel zu den Verbindungspunkten zwischen BlueJVisualize und BlueJ V3.1.0 Stellen im Code von BlueJ V4.1.2 aufgezeigt an denen BlueJVisualize anknüpfen kann.
Zuletzt wird die Oberfläche von BlueJVisualize angepasst und modernisiert um mehr dem aktuellen Oberflächendesign von BlueJ zu entsprechen.

4.2.1. Minimieren der Abhängigkeiten

Wie in Abschnitt 4.1 zu sehen, gibt es viele Typen von denen BlueJVisualize abhängig ist, ohne das sie tatsächlich zur Funktion beitragen.
Wovon BlueJVisualize aus welchen Gründen abhängig ist ist in Abbildung 4.1 noch einmal verdeutlicht.
Der TextAnalyser der dazu verwendet wird die Variablennamen zu ihren Typen aufzulösen kann problemlos entfernt werden, da NamedValue bereits eine Möglichkeit bietet den Typen

der Variable zu bestimmen. Da NamedValue auch für andere Teile von BlueJVisualize gebraucht wird kann man es hier also genauso gut anstelle von TextAnalyser verwenden. Dies spart eine Abhängigkeit und die unnötige Erzeugung einer Instanz von TextAnalyser.
Das aktuelle Vorgehen, Fehlermeldungen über Methodenaufrufe am CodePad weiterzugeben, ist umständlich und erfüllt keinen weiteren Zweck; daher kann dies ebenso gut anhand von Exceptions erledigt werden. Es werden anstelle von Methodenaufrufen eine Exception eines eigenen Exceptiontypen geworfen und im CodePad verarbeitet. Hierzu ist eine Änderung am CodePad nötig um das entsprechende Verhalten zu implementieren, aber das war bei der alten Variante ebenso, bloß an anderer Stelle, notwendig. Somit benötigt BlueJVisualize keine Referenz mehr zu dem CodePad von dem aus es gestartet wurde und es konnte eine weitere Abhängigkeit eingespart werden.
Anstatt NamedValue & ValueCollection zu verwenden können die benötigten Daten auf einfache Strings heruntergebrochen werden, da diese nur zum Übermitteln der Variablennamen und Typen verwendet werden. Dies macht keinen Sinn solange BlueJVisualize noch in BlueJ eingebettet ist, wäre aber bei einer Anbindung über die Erweiterungsschnittstelle eine Möglichkeit die Übertragung umzusetzen ohne auf BlueJ interne Klassen angewiesen zu sein. Eine andere Möglichkeit wäre die Erweiterungsschnittstelle um eine ähnliche Klasse zu erweitern. Die erforderlichen Abhängigkeiten konnten so auf die in Abbildung 4.2 zu sehenden Anteile reduziert werden.

4.2.2. Veränderungen an den Verbindungsstellen

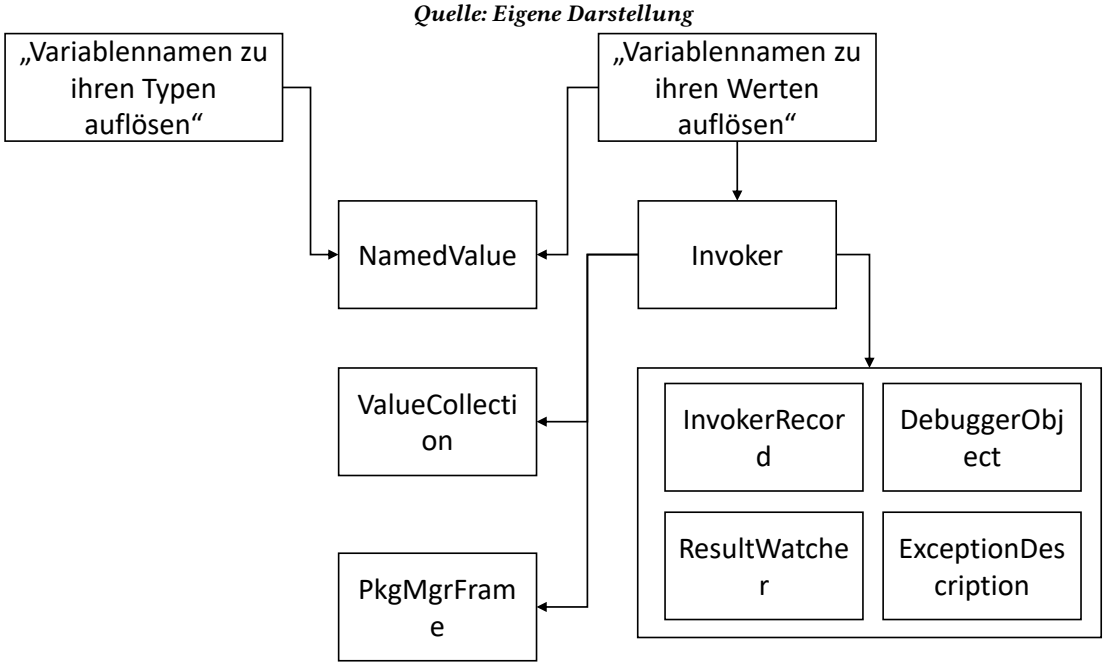
Mit dem Versionssprung von BlueJ Version 3.1.0 zu Version 4.1.2 besteht natürlich auch die Möglichkeit das sich die Klassen an denen BlueJVisualize anknüpft verändert haben. Deshalb werden diese nochmal darauf untersucht ob BlueJVisualize problemlos an die neue Version angeknüpft werden kann:

NamedValue & ValueCollection Es gab keine relevanten Änderungen, beide können ohne jedweden Aufwand weiter verwendet werden wie zuvor.

Invoker Die Invoker-Klasse ist nach wie vor vorhanden, es gab einige Änderungen im Zuge des Versionswechsels, aber die Benutzung hat sich für BlueJVisualize nicht verändert.

CodePad Trotz starker Veränderung am CodePad ist die grundsätzliche Struktur gleich geblieben. So hat sich zwar einiges an den Funktionen und dem Aufbau verändert, aber es ist der vorherigen Version immer noch ähnlich genug um keinen großen Aufwand bei der Migration zu erzeugen.

Abbildung 4.2.: Abhänigigkeiten von BlueJVisualize nach Reduktion



PkgMgrFrame Der PkgMgrFrame hat sich sehr stark verändert, allerdings ändert sich dadurch nichts für BlueJVisualize da er nur zur Erzeugung des Invoker verwendet wird und sich daran nichts geändert hat.

Insgesamt hat sich zwar viel an BlueJ intern und der Oberfläche verändert, aber die erneute Anbindung von BlueJVisualize ist nahezu ohne weiteren Aufwand möglich.

4.2.3. Migration zu BlueJ Version 4.1.2

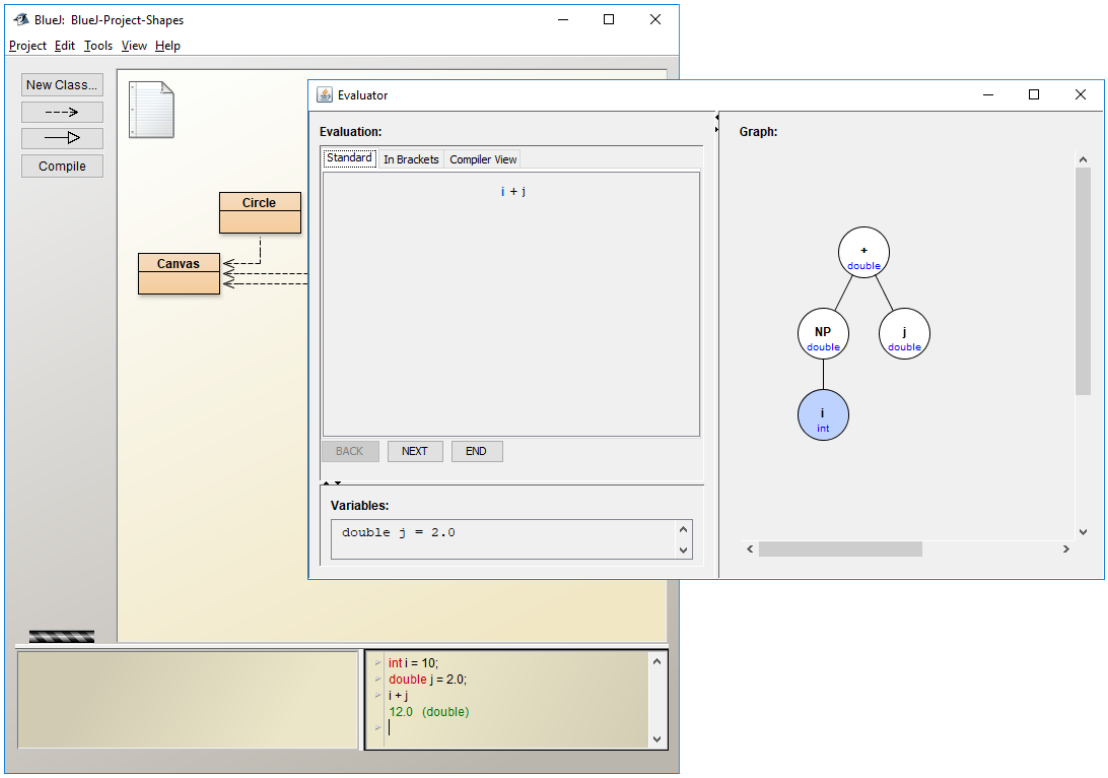
Die Trennung vom alten BlueJ ist kein Problem, da bei dem Entwurf von BlueJVisualize darauf geachtet wurde das es möglichst frei steht von dem Rest von BlueJ. Deshalb sind alle erforderlichen Klassen die nicht zu BlueJ selbst gehören in einem eigenen Paket zusammengefasst. Somit kann der Code von BlueJVisualize leicht vom Quellcode der alten BlueJ Version in den Quellcode der neuen Version übertragen werden.
Wie zuvor festgestellt muss sich auch an der Verwendung der einzelnen Typen nichts ändern. Lediglich die Anbindung an das CodePad benötigt hier Aufwand, da es stark verändert wurde. Ähnlich wie zuvor kann ein eigens im Code festgelegtes Tastenkürzel die Oberfläche von BlueJVisualize starten. Hierzu sind wie zuvor einige Änderungen im Code des CodePad not-

wendig. Die Anbindung gestaltet sich in der neuen Version jedoch nicht schwieriger als zuvor, sondern eher sogar angenehmer; es sind weniger schwere Änderungen notwendig als vorher. Wo es zuvor notwendig war einen Subtypen des Typen Action aus dem java.swing Paket zu implementieren, welcher zusammen mit einer Tastenbelegung hinterlegt wurde, genügt jetzt ein einfacher Methodenaufruf welcher über einen Filter aufgerufen wird der bei Eingabe in das CodePad überprüft wird. Die relevanten Quellcodeauszüge sind im Anhang B auf den Seiten 28 bis 31 zu finden.
Die Einbindung von BlueJVisualize in BlueJ nimmt keinen Einfluss auf die sonstige Anwendung von BlueJ und die modifizierte Version von BlueJ weist, wenn BlueJVisualize nicht verwendet wird, keine für den Benutzer spürbaren Unterschiede auf.

4.2.4. Oberflächenerneuerung von BlueJVisualize

Abbildung 4.3.: Alte Swing Oberfläche von BlueJVisualize in BlueJ 3.1.0

Quelle: Eigene Darstellung



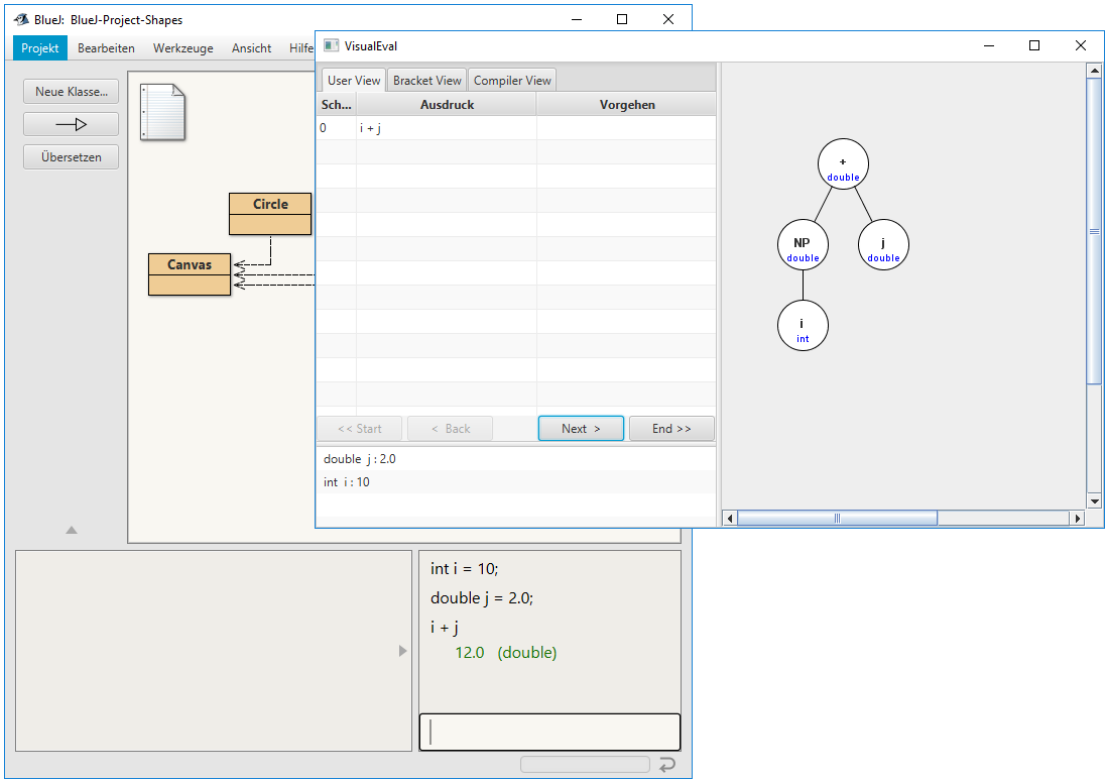
Da BlueJ mit Version 4.0.0 von Swing auf JavaFX umgestiegen ist und auch BlueJVisualize Swing als Werkzeugsatz für die grafische Oberfläche verwendet bietet sich hier die Gelegenheit BlueJVisualize ebenfalls auf JavaFX zu aktualisieren um ein mit BlueJ einheitliches Bild darzustellen. Hierzu wird die vorhandene Swing Oberfläche von BlueJVisualize entfernt und durch eine neue JavaFX Oberfläche ersetzt; wobei einige Gesichtspunkte beachtet werden sollen. Der Code der Oberfläche soll möglichst von der Funktionalität von BlueJVisualize getrennt sein um hier eventuelle Änderungen an einem der beiden leicht zu ermöglichen ohne Änderungen am Anderen zu benötigen. Hierzu wird die Möglichkeit von JavaFX genutzt die Oberfläche in einer FXML Datei zu beschreiben und über einen Controller mit dem Modell von BlueJVisualize zu verknüpfen. So entsteht eine MVC[3]-ähnliche Struktur und die Trennung der Komponenten ist weitestgehend gegeben. Der unterliegende Aufbau von BlueJVisualize kann aufgrund der bisherigen Trennung überwiegend ohne Änderung erhalten bleiben.
Nicht der Fall ist dies bei sowohl der Klammer- als auch der Compiler-Ansicht. Beide konnten nicht in die aktuelle Version übernommen worden, da diese den eigens dafür geschriebenen LabelVisitor[4]benötigen. Dieser ist aber leider fest für Interaktionen mit Swing Oberflächen geschrieben worden; an dieser Stelle wurde nicht sauber getrennt.
Im Zuge dieser Erneuerung ist auch das Oberflächendesign noch einmal überdacht worden mit dem Ergebnis das einige Änderungen vorgenommen wurden:[5]

- Entfernen von selbsterklärenden Beschriftungen für eine übersichtlichere Oberfläche und effektivere Nutzung des vorhandenen Platz.
- Bedienelemente vergrößert um die Bedienung zu erleichtern und um Schaltfläche erweitert um an den Anfang der Auswertung zurück zu kommen was beim Untersuchen großer Ausdrücke benötigt wird.
- Erklärungen zu den einzelnen Schritten der Auswertung aus dem Hinweis beim Schritt in die Auswertungstabelle verschoben, um übersichtlicher zu machen was in den einzelnen Schritten passiert ist.

[3] Eckstein (2007)
[4] Der LabelVisitor ist eine in Beckert (2013) nach dem Visitor PatternGamma u. a. (1994) entworfene Klasse, welche den intern von BlueJVisualize erzeugten Baum, der den aktuellen Stand in der Auswertung eines Ausdrucks beschreibt, durchläuft und für die Klammer- & Compiler-Ansicht jeweils Label generiert. Genau da liegt auch das Problem; die Produkte der Traversierung sind Klassen aus dem javax.swing Paket und können nicht sauber in eine JavaFX Oberfläche eingebaut werden.
[5] Die hier beschriebenen Veränderungen können in Anhang C auf Seite 32 betrachtet werden.

Abbildung 4.4.: Neue JavaFX Oberfläche von aktualisiertem BlueJVisualize in BlueJ 4.1.2

Quelle: Eigene Darstellung



Die an der Oberfläche vorgenommenen Veränderungen dienen der Steigerung der Benutzerfreundlichkeit da Defizite hier ,besonders bei Einsteigern, welche die Zielgruppe der Erweiterung darstellen, leicht von der Verwendung abschrecken können.
Problematisch bei dem Umstieg auf JavaFX ist die Darstellung des Auswertungsbaums. Diese basiert auf dem Graphen Framework JUNGJUNG (2018), welches auf Swing aufbaut und ebenso zur Verwendung in einer Swing Oberfläche gedacht ist. Nur mit zusätzlichem Aufwand und unter Vorbehalt kann JUNG unter JavaFX verwendet werden. Die Stabilität der Erweiterung ist unter diesen Umständen nicht garantiert, es sollte daher in Betracht gezogen werden in Zukunft auf ein anderes Framework zur Darstellung des Auswertungsgraphen umzusteigen.
Die aktualisierte JavaFX Oberfläche von BlueJVisualize ist in Abb. 4.4 zu sehen.

4.2.5. Stand von BlueJVisualize nach dieser Arbeit

Die an BlueJ 4.1.2 angepasste Version von BlueJVisualize besitzt nicht die Fülle der gewünschten Funktionen und könnte in einer zukünftigen Arbeit noch einmal um diese erweitert werden. Mehr dazu in Abschnitt 5.2.3 & Abschnitt 5.2.4. In ihrer jetzigen Form ist die aktualisierte Version von BlueJVisualize sowohl als Projekt innerhalb von Eclipse lauffähig[6], sowie über eine generische BlueJ-Installation[6] verfügbar mit welcher es leichter zu verteilen, installieren und benutzen ist. Es ist leider in dieser Form immer noch an eine bestimmte Version von BlueJ gebunden, und deshalb nach wie vor nicht für die allgemeine Nutzung geeignet; aber es kommt dem näher.

4.3. Für BlueJVisualize notwendige Änderungen an der Erweiterungsschnittstelle

Nachdem die Abhängigkeiten von BlueJVisualize reduziert wurden und es an die aktuelle BlueJ Version angepasst wurde ist das nächste Ziel die Anbindung über die Erweiterungsschnittstelle. BlueJVisualize ist mit einem Teil von BlueJ verbunden, welcher bis jetzt nicht über die Erweiterungsschnittstelle abgebildet wird: Dem CodePad. Um eine Verbindung mit diesem für Erweiterungen zu ermöglichen ist es notwendig die Erweiterungsschnittstelle weiter auszubauen. Wie dies aussehen könnte wird im Folgenden Abschnitt betrachtet.

4.3.1. Abbildung der Abhängigkeiten über die Erweiterungsschnittstelle

Für eine Anbindung von BlueJVisualize an die Erweiterungsschnittstelle müssen die in Abschnitt 4.2.1 verbliebenen Abhängigkeiten über die Erweiterungsschnittstelle abgebildet werden:

- Zu einem gegebenen Variablennamen den Typen der Variable festzustellen.
- Zu einem gegebenen Variablennamen die Belegung der Variable festzustellen.

Diese Abhängigkeiten lassen sich leicht zusammenfassen: BlueJVisualize benötigt die Möglichkeit die Informationen zu einer Variable die im auszuwertenden Ausdruck vorkommt in Erfahrung zu bringen. Das ließe sich leicht zu einer Operation verbinden in der für einen gegebenen Variablennamen die hinterlegten Information übertragen werden. Allerdings ist

[6] Der Quellcode und die modifizierte generische BlueJ-Installation befindet sich auf der dieser Arbeit beiliegenden CD

das nicht alles, hinzu kommen die Funktionen die bei BlueJVisualize im Moment über den modifizierten Teil des BlueJ Quellcode übernommen werden:

- Aufruf der Erweiterung über das CodePad ermöglichen
- Übertragen der Namen von im CodePad erzeugten Variablen

Auch hier lassen sich die Anforderungen wieder zusammenfassen: Es muss eine Möglichkeit geben die Erweiterung über das CodePad aufzurufen bei welchem eine Kopie der erzeugten Variablen zum Zeitpunkt des Aufrufs mit übertragen werden.
Insgesamt kann gesagt werden, dass die Erweiterungsschnittstelle für Erweiterungen die Möglichkeit bieten muss sich zu registrieren, um Aufrufe über das CodePad zu erhalten, und Zugriff auf die Daten(Name, Typ, Belegung) der im CodePad erzeugten Variablen zu erlangen.[7]
Da bis jetzt kein Teil der Erweiterungsschnittstelle mit dem CodePad verknüpft ist wären diese neuen Funktionen vermutlich am besten in einer neuen Komponente der Erweiterungsschnittstelle angebracht, welche mit Rücksicht auf das bisherige Schema, der Abbildung auf interne Klassen von BlueJ, ebenfalls als CodePad benannt werden sollte, da es sich hier um Proxy-Objekt handeln würde.

[7] Hier wäre zu beachten bloß eine Kopie der Variablen o. Ä. herauszugeben um die normale Funktionen von BlueJ zu sichern. Hierauf wurde beim Entwurf der Erweiterungsschnittstelle Wert gelegt und ist vermutlich weiterhin ein Anliegen der BlueJ Entwickler

5. Fazit & Ausblick

5.1. Fazit

Die Version 4.0.0 hat viele Veränderungen für BlueJ mit sich gebracht: Die Oberfläche wurde fast komplett modernisiert und mit Stride ist Unterstützung für eine neue Sprache und eine neue einsteigerfreundliche Art Programme zu schreiben, für mehr Möglichkeiten in der Lehre, dazugekommen. Im Zuge dieser Änderungen ist die Langlebigkeit der Erweiterungsschnittstelle nicht mehr gewiss und sorgt für Unsicherheit bei der (Weiter-) Entwicklung von Erweiterungen. Allerdings besteht hier auch die Chance Wünsche für Änderungen der Erweiterungsschnittstelle einzubringen; diese könnten dann direkt in die Umschreibung/Neukonzipierung einfließen anstatt im Nachhinein hinzugefügt zu werden.

5.2. Ausblick

Im Anschluss an diese Arbeit biete sich viele Anschlusspunkte an denen für eine Folgearbeit angeknüpft werden kann. Es sind Größtenteils praxislastige Arbeiten und die meisten beschäftigen sich in irgendeiner Form damit BlueJVisualize näher an das Ziel zu bringen vollständig und leicht einsetzbar in der Lehre verfügbar zu machen.

5.2.1. Bereinigen der Erweiterungsschnittstelle von durch die Migration entstandenen Seiteneffekten

Die Bereinigung der Erweiterungsschnittstelle würde mindestens eine Anpassung der Teile der Erweiterungsschnittstelle benötigen, welche noch auf Swing angewiesen sind. Entsprechend müssten aber auch die Stellen im BlueJ Quellcode angepasst werden, welche momentan noch so aufgebaut sind die Swing Elemente in die, hauptsächlich auf JavaFX umgestellte Oberfläche, zu integrieren. Eventuell wäre sogar eine komplette Neukonzipierung der Erweiterungsschnittstelle möglich, allerdings sollte hier klare Absprache mit den BlueJ Entwicklern herrschen und die Anforderungen und Erwartungen an die neue Erweiterungsschnittstelle möglichst einvernehmlich festlegen.

5.2.2. Erweiterung der Erweiterungsschnittstelle um notwendige Methoden um BlueJVisualize zu extrahieren

Ob die Erweiterung der Erweiterungsschnittstelle in ihrem aktuellen Zustand ratsam ist, ist fragwürdig. Hier sollte zuerst Rücksprache mit den BlueJ Entwicklern gehalten werden um deren Pläne bezüglich Veränderungen der Erweiterungsschnittstelle in Erfahrung zu bringen. Eventuell könnten dann diese vorgeschlagenen Änderungen sogar in einen neuen Entwurf der Erweiterungsschnittstelle integriert werden.

Mögliche Vorteile

Die hier vorgeschlagenen Änderungen sind hauptsächlich zu dem Zweck entworfen BlueJVisualize über die Erweiterungsschnittstelle als reguläre Erweiterung verfügbar zu machen. Dennoch können durch die Eröffnung neuer Möglichkeiten, zur Interaktion mit dem Benutzer, hoffentlich auch andere Erweiterungen profitieren. Das CodePad findet häufig Verwendung in der Lehre um einfache Thesen zu testen und erwartete Ergebnisse zu überprüfen. Erweiterungen hier einen Zugang zu ermöglichen kann dabei helfen dieses interaktive Mittel in der Lehre, unter Zuhilfenahme von Erweiterungen wie BlueJVisualize, noch effektiver zu machen.

Nötige Änderungen an der Erweiterungsschnittstelle

Um effektiv mit dem CodePad interagieren zu können müssten mehrere Anforderungen erfüllt werden:

- Erweiterungen müssen in der Lage sein die Eingabe aus dem CodePad auszulesen oder übertragen zu bekommen - Hierfür gibt es mehrere Möglichkeiten:
  1. Einen dauerhaften Beobachter[1] am CodePad anmelden und über alle Eingaben informiert werden.
  2. Ebenfalls Registrierung als Beobachter, aber wie in BlueJVisualize mit Aufruf und Übertragung der aktuellen Eingabe bei Betätigung eines zuvor angegebenen Tasten-Kürzel.[2]
  3. Beides anbieten um die Verwendung des CodePad in unterschiedlichen Erweiterungen zu ermöglichen.

[1] Nach dem Beobachter-Muster von Gamma u. a. (1994)
[2] Dieses sollte möglichst schon bei der Registrierung des Beobachters mitgeliefert werden um Überschneidungen so früh wie möglich zu erkennen.

- Es muss Erweiterungen möglich sein zu den, im CodePad erzeugten, Variablen mindestens den Namen, Typ und die Belegung in Erfahrung zu bringen. Mögliche Umsetzungen wären:
  1. Eine Methode bereitstellen die bei Aufruf die Sammlung der zu diesem Zeitpunkt im CodePad existierenden Variablen liefert.
  2. Beim Aufruf der Erweiterung, durch den Benutzer, die Sammlung der zu diesem Zeitpunkt im CodePad existierenden Variablen übertragen[3]

Des Weiteren muss eine Form festgelegt werden in der die Daten einer Variable übertragen werden. Hierzu könnten für eine einfache Lösung mehrere bloße Strings verwendet werden. Eine andere Möglichkeit wäre es in einer Klasse die relevanten Informationen zu sammeln und diese so gebündelt zu übertragen. Hier bietet sich eine Form ähnlich der interne Klasse NamedValue an, um die Umwandlung leicht zu gestalten. Wenn alle diese Anforderungen von der Erweiterungsschnittstelle erfüllt werden müsste der Extraktion und anschließenden Entwicklung von BlueJVisualize als reguläre Erweiterung nichts mehr im Weg stehen.

Implementationsvorschlag der Erweiterungen an der Erweiterungsschnittstelle

Dies ist ein Vorschlag für eine mögliche Implementation der notwendigen Änderungen um, basierend auf den zuvor erarbeiteten Möglichkeiten, die Erweiterungsschnittstelle so zu erweitern, dass es Erweiterungen wie BlueJVisualize möglich wäre an das CodePad anzuknüpfen und so neue Funktionen für die Benutzer zu ermöglichen. Es werden 3 neue Klassen an der Erweiterungsschnittstelle benötigt:

CodePad Ein Proxy-Objekt[4], welches Erweiterungen erlaubt mit dem CodePad zu interagieren und wie in 2 auf 22 beschrieben registrieren. Aufrufe des Benutzers werden wie in 2 auf Seite 23 beschrieben an die entsprechenden Erweiterungen weitergereicht.

BVariable Wie zuvor beschrieben wäre dies die Repräsentation der im CodePad existierenden Variablen, wobei jeweils der Name, Typ und die Belegung der Variable vorhanden sein sollte. Für BlueJVisualize reicht es hier den Typ nur als String darzustellen, für mehr mögliche Funktionen könnte aber ein komplexerer Typ verwendet werden[5]

CodePadListener Eine simple Schnittstelle die den implementierenden Erweiterungen erlaubt sich an der neuen Klasse CodePad zu registrieren.

[3] In Verbindung mit 2
[4] In Anlehnung an die Klasse BlueJ der Erweiterungsschnittstelle.
[5] Wie z. B. die momentan BlueJ interne Klasse JavaType

5.2.3. Extrahieren von BlueJVisualize

Die erhoffte Entfernung von BlueJVisualize aus dem BlueJ Code ist in der aktuellen Version von BlueJ leider noch nicht möglich; es müsste erst eine Erweiterung der Erweiterungsschnittstelle wie in 5.2.2 beschrieben stattfinden.

Vorteile

Durch die Umsetzung von BlueJVisualize als reguläre Erweiterung wäre es nicht mehr Notwendig es an jede neue BlueJ Versionen anzupassen um es in Verbindung mit dieser lauffähig zu machen. Außerdem wäre der Aufwand der Ersteinrichtung deutlich verringert und somit auch eine weitere Verbreitung leichter zu erreichen. Zusätzlich kann so die zukünftige Weiterentwicklung von BlueJVisualize erleichtert werden da dann keine Kenntnis des BlueJ Quellcode mehr notwendig wäre.

Bereits minimierte Abhängigkeit

Wie in Abschnitt 4.2.1 beschrieben konnte BlueJVisualize bereits von einigen Abhängigkeiten zum BlueJ Quellcode befreit. Somit sollte eine Extraktion nun leichter als bisher möglich sein. Die verbleibenden Abhängigkeiten sind vergleichsweise einfach und werden auch in Abschnitt 4.2.1 noch einmal erklärt und sollten anschließend leicht zu lösen sein.

Eventuelle Komplikationen

Das CodePad existiert momentan in BlueJ nicht dauerhaft, sondern ist an ein geöffnetes Projekt gebunden. Wird dieses geschlosen wird auch das CodePad verworfen. Das heißt auch das kein CodePad existiert, solange kein Projekt geöffnet ist. Außerdem kann das CodePad ausgeblendet werden was intern dazu führt das dieses verworfen und erst beim erneuten öffnen neu erzeugt wird. Deshalb müssten registrierte Erweiterungen jedes mal wenn das CodePad neu aufgebaut wird sich erneut registrieren. Ein möglicher Lösungsansatz wäre es sich bei einer Zwischenklasse[6] zu registrieren welche sich beim Initialisieren des CodePad auch darum kümmert das alle registrierten Erweiterungen wieder auf entsprechende Eingaben reagieren können. Wenn Tastenkürzel programmatisch von den Erweiterungen festgelegt werden könnte es zu Überschneidungen kommen welche von BlueJ erkannt werden müssten. Es bietet sich an als Reaktion auf eine solche Überschneidung dem Benutzer die Möglichkeit zu geben das Problem zu beheben indem er selber Tastenkürzel vergibt. Grundsätzlich würde es sich lohnen

[6] Wie beschrieben in Abschnitt 5.2.2

dem Nutzer die Möglichkeit zu geben diese Kürzel jederzeit selber über das Optionsmenü festzulegen und die Vorgaben der Erweiterungen nur als Standardwerte zu behandeln.

5.2.4. Weiterentwickeln von BlueJVisualize

BlueJVisualize hat in der Form wie es jetzt in BlueJ Version 4.1.2 eingebettet ist nicht seinen vollen Funktionsumfang. Dieser kann in Beckert (2013) nachgelesen werden und umfasst z. B. die Möglichkeit die Ansicht der Auswertung auf eine geklammerte Sicht zu wechseln welche die Ordnung der Auswertungsschritte verdeutlicht und die Verwendung von Methodenaufrufen in den auszuwertenden Ausdrücken.
Es wäre allerdings von Vorteil BlueJVisualize erst nach einer Extraktion aus BlueJ weiter zu entwickeln, da sonst weitere Abhängigkeiten entstehen können welche bei einer Extraktion wieder aufgearbeitet und mühsam aufgelöst werden müssten.

Erweiteren der Funktionalität

Weggefallene Funktionen können wiederhergestellt werden: Die Klammeransicht sowie die Compileransicht müssten für Verwendung unter JavaFX neu implementiert werden, der nötige Platz dafür ist in der Oberfläche schon vorhanden. Wenn BlueJVisualize als reguläre BlueJ Erweiterung implementiert ist kann das Menü der Erweiterung in BlueJ um eine Sammlung an beispielhaften hilfreichen Ausdrücken, die bestimmte Sachverhalte demonstrieren, und als Einstiegspunkt in die Erweiterung dienen erweitert werden.

Reevaluierung der Architektur

Im selben Zug würde es sich lohnen die Architekturentscheidungen die in BlueJVisualize eingeflossen sind noch einmal zu untersuchen, da es zuerst nur als Prototyp implementiert und anschließend weiterentwickelt wurde. Hierbei konnte aufgrund der Rahmenbeschränkungen nicht immer so viel Sorgfalt angewendet werden wie es eigentlich gewünscht wäre. Dadurch entstehen Altlasten welche refaktorisiert werden sollten um die Qualität des Quellcode zu steigern. Insbesondere bei der Visualisierung des Auswertungsbaums sollte geprüft werden ob ein anderes Framework als JUNG (2018) verwendet werden kann, da es hier durch JUNGs Abhängigkeit von Swing zu Konflikten mit BlueJ kommt und auf unschöne Zwischenlösungen ausgewichen werden muss.

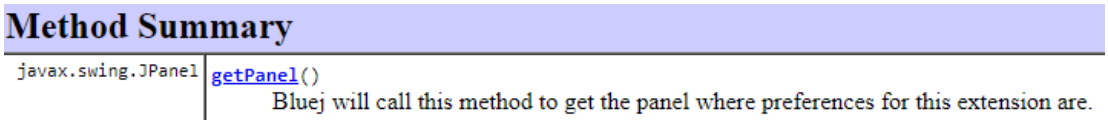
A. Auszüge aus der Erweiterungsschnittstelle

Method Summary

javax.swing.JMenuItem	getClassMenuItem(BClass bc) Returns the JMenuItem to be added to the BlueJ Class menu Extensions should not retain references to the menu items created.
javax.swing.JMenuItem	getMenuItem() **Deprecated.** *As of BlueJ 1.3.5, replaced by getToolsMenuItem(BPackage bp)*
javax.swing.JMenuItem	getObjectMenuItem(BObject bo) Returns the JMenuItem to be added to the BlueJ Object menu Extensions should not retain references to the menu items created.
javax.swing.JMenuItem	getToolsMenuItem(BPackage bp) Returns the JMenuItem to be added to the BlueJ Tools menu.

Abbildung A.1.: Abhängigkeit von Swing in der Implementation von MenuGenerator

Quelle: Kölling und BlueJ Team (2018c)



Method Summary

javax.swing.JPanel	getPanel() Bluej will call this method to get the panel where preferences for this extension are.

Abbildung A.2.: Abhängigkeit von Swing in der Implementation von PreferenceGenerator

Quelle: Kölling und BlueJ Team (2018c)

B. Quellcodeauszüge aus BlueJVisualize

Quellcodeauszug B.1: Anbindung von BlueJVisualize in BlueJ 3.1.0

```
1 // OpenVisualizerAction
2 action = new OpenVisualizerAction();
3 newmap.addActionForKeyStroke(KeyStroke.getKeyStroke(KeyEvent.
    VK_ENTER, Event.CTRL_MASK), action);
```

Quellcodeauszug B.2: Aufruf von BlueJVisualize in BlueJ 3.1.0

```
1  /* -------------------- HERE TODO INSERT ---------------------- */
2  final class OpenVisualizerAction extends AbstractAction
3  {
4      private static final long serialVersionUID = 42L;
5
6      /**
7       * Create a new action object. This action opens the
           visualization
8       * of the current command.
9       */
10     public OpenVisualizerAction()
11     {
12         super("OpenVisualizerAction");
13     }
14
15     /**
16      * Visualizes the current line. If successful, the line will
           also
17      * be executed in the text area.
18      */
19     @Override
20     final public void actionPerformed(ActionEvent arg0)
21     {
```

```
22          // Pretty much doing the same thing as in
                ExecuteCommandAction,
23          // but we start the visualization instead of executing the
                expression.
24          // For some reason the output is prettier if we don't call
                markCurrentAs().
25
26          if (busy)
27          {
28              return;
29          }
30
31          // get current line
32          String line = getCurrentLine();
33
34          // we do not add the line to currentCommand, because
                ExecuteCommandAction
35          // (which we invoke later) already does it and we don't want
                 it added twice
36          String command = (currentCommand + line).trim();
37
38          if (command.length() != 0)
39          {
40              setEditable(false); // don't allow input while we're
                    thinking
41              busy = true;
42
43              // get declared variables
44              List<NamedValue> vars = new ArrayList<NamedValue>();
45              vars.addAll(localVars);
46
47              // start visualizer
48              EvalVisualizationStarter visualizer = new
                    EvalVisualizationStarter();
49              visualizer.visualize(command, vars, frame, TextEvalPane.
                    this);
50          }
51      }
52  }
53
```

```
54 /**
55  * Visualization was successful.
56  * Invokes the execution in the text area as well.
57  */
58 public void visualizationSuccessful()
59 {
60     busy = false;
61     executeCommandAction.actionPerformed(null);
62 }
63 
64 /**
65  * Visualization failed.
66  * Adds the error message to the text area.
67  *
68  * @param error error message
69  */
70 public void visualizationFailed(String error)
71 {
72     // add line to the history
73     history.add(getCurrentLine());
74 
75     // show error message
76     append("\n");
77     showErrorMsg(error);
78 }
79 /* -------------------- HERE END INSERT ----------------------- */
```

Quellcodeauszug B.3: Anbindung von BlueJVisualize in BlueJ 4.1.2

```
1 }// TODO --- Hacked Code starts here ---
2 else if (e.getCode() == KeyCode.ENTER && e.isControlDown())
3 {
4         visualize();
5         //Do not consume the event to enable normal execution in the
              background
6         //e.consume();
7 }// --- Hacked Code ends here ---
```

Quellcodeauszug B.4: Aufruf von BlueJVisualize in BlueJ 4.1.2

```
1 // TODO --- Hacked Code starts here ---
```

```
2 private void visualize() {
3         if(busy) {
4                 return;
5         }
6
7         String line = inputField.getText();
8         if (line.trim().isEmpty())
9                 return; // Don't allow entry of blank lines
10        inputField.setEditable(false);    // don't allow input while
              we're thinking
11        String statement = (currentCommand + line).trim();
12        if(statement.length() != 0) {
13
14                history.add(line);
15                command(line, true);
16                inputField.setText("");
17                executeCommand(statement);
18
19                EvalVisualizationStarter visualizer = new
                     EvalVisualizationStarter();
20                try {
21                        visualizer.visualize(statement, new
                             ArrayList<>(localVars), frame);
22                } catch (EvalVisualizationException e) {
23                        System.out.println(e.getMessage());
24                        error(e.getMessage());
25                }
26        }
27        inputField.setEditable(true);
28 }
29 // --- Hacked Code ends here ---
```

C. Abbildungen zu den Oberflächenänderungen an BlueJVisualize

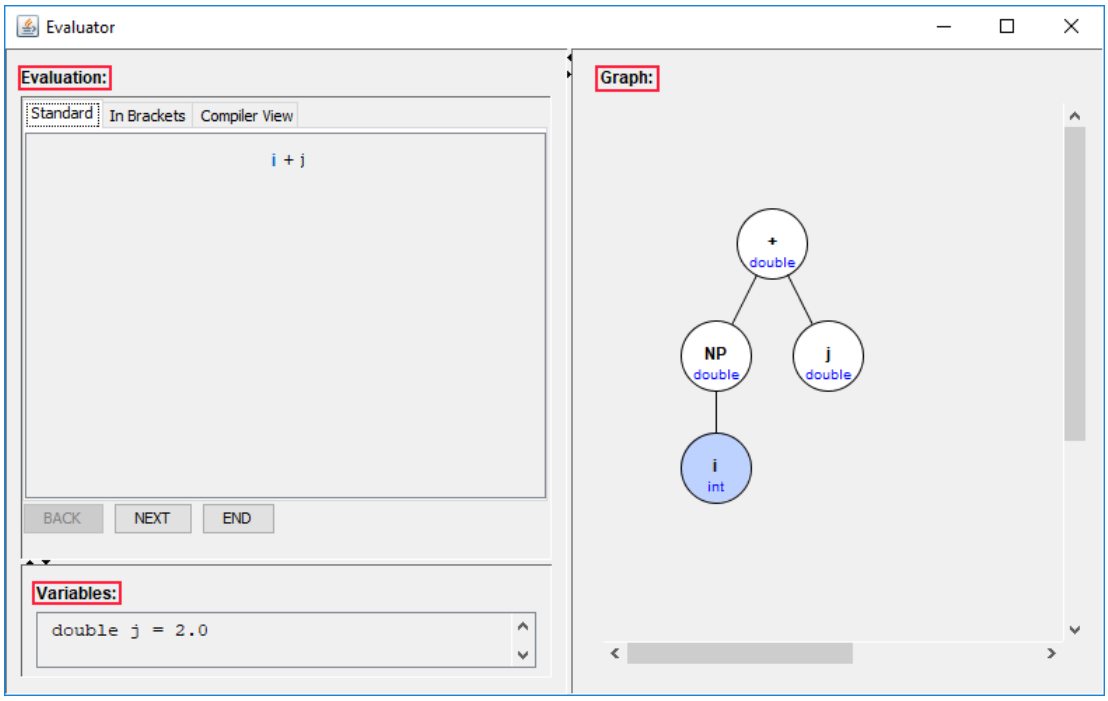


Abbildung C.1.: Selbsterklärende Beschriftungen in der BlueJ 3.1.0 Version von BlueJVisualize

Quelle: Eigene Darstellung

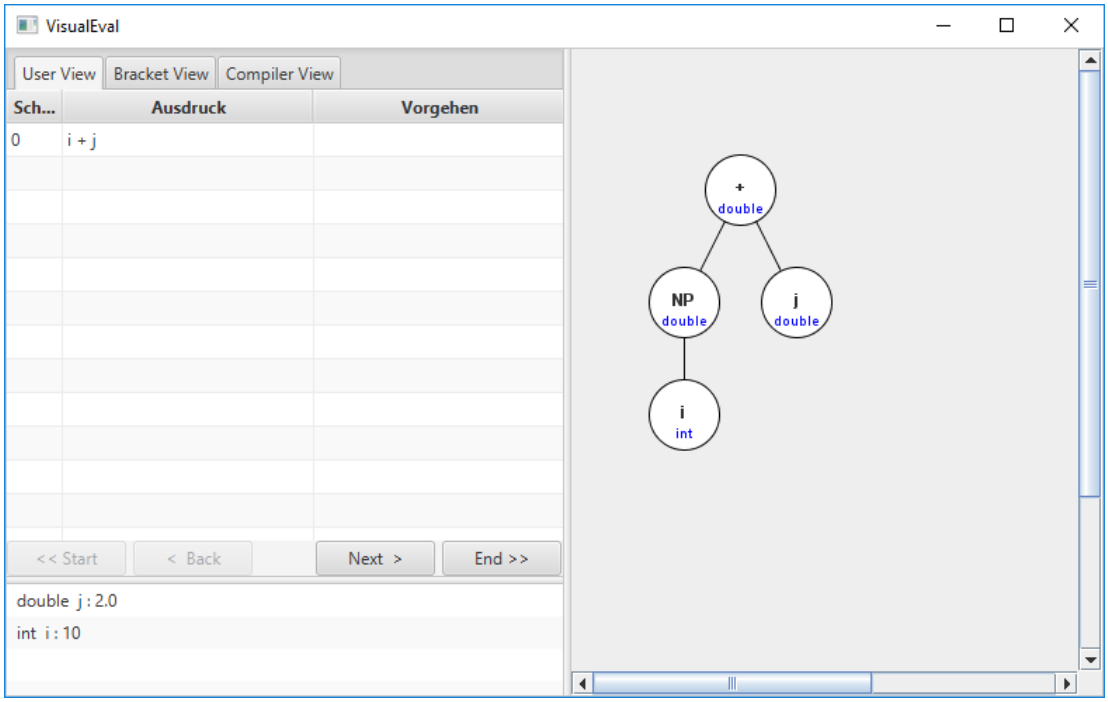


Abbildung C.2.: Keine Beschriftungen in der BlueJ 4.1.2 Version von BlueJVisualize

Quelle: Eigene Darstellung

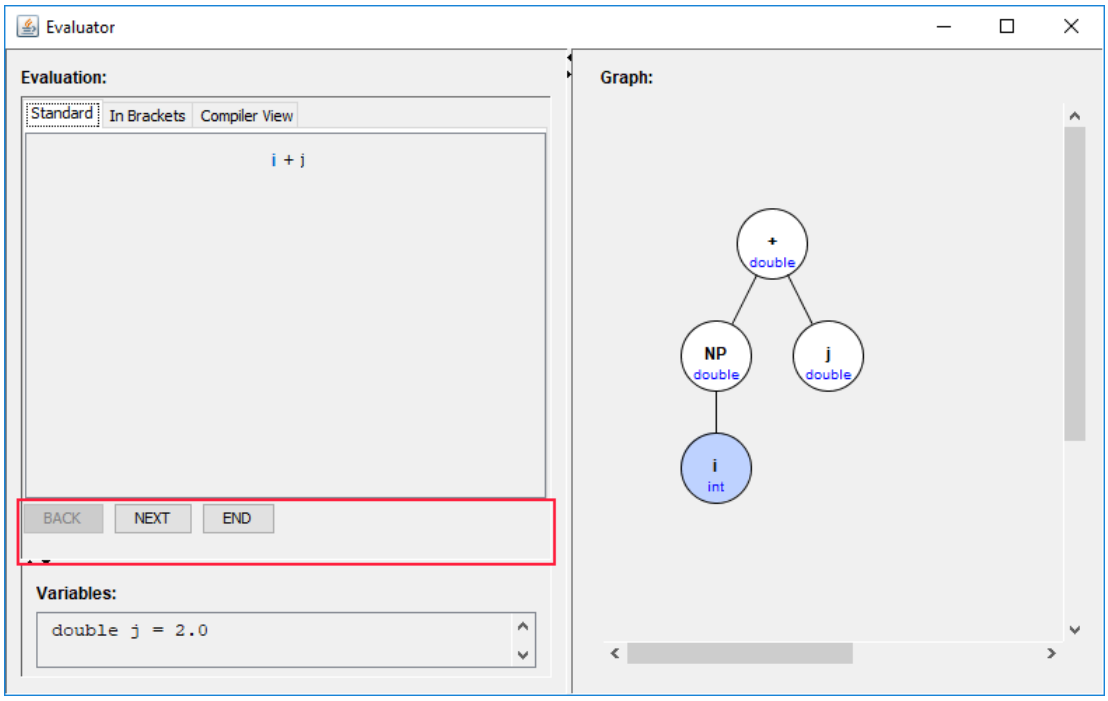


Abbildung C.3.: Ungünstige Schaltflächen in der BlueJ 3.1.0 Version von BlueJVisualize

Quelle: Eigene Darstellung

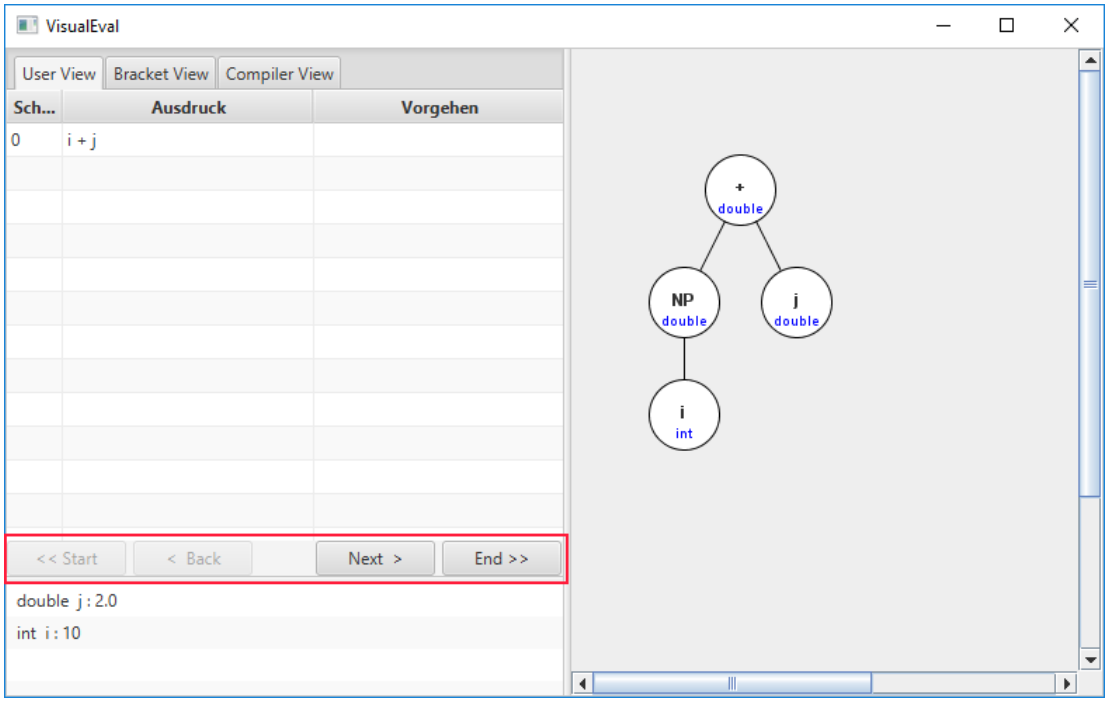


Abbildung C.4.: Optimierte Schaltflächen und neue Bedienelemente in der BlueJ 4.1.2 Version von BlueJVisualize

Quelle: Eigene Darstellung

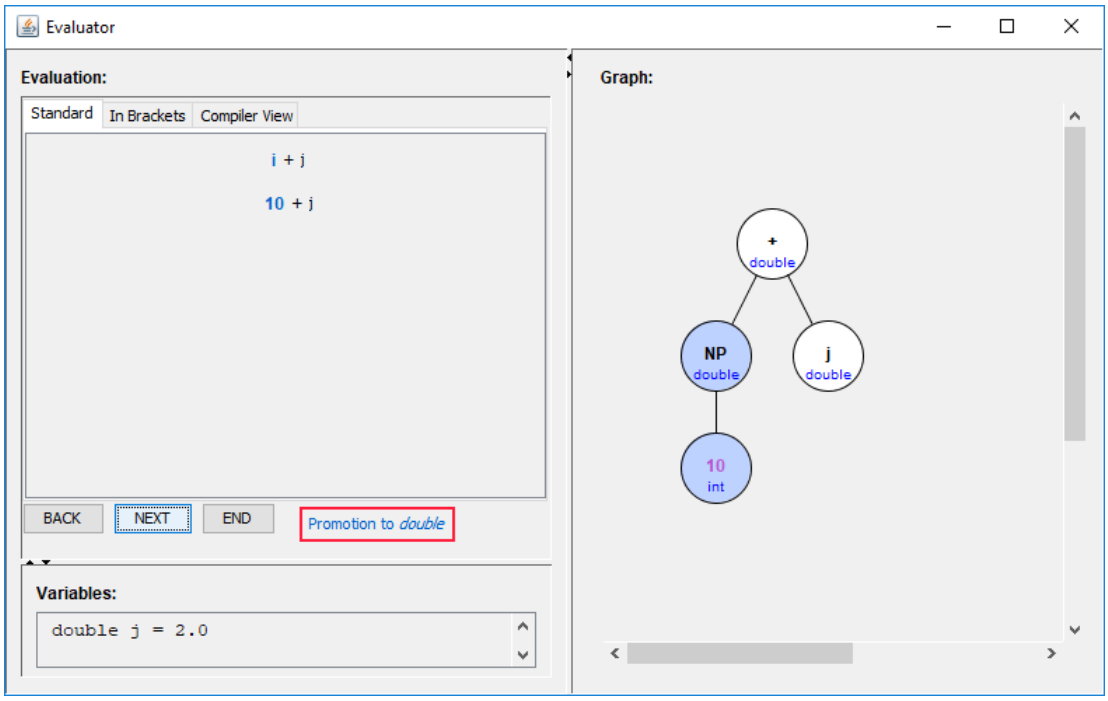


Abbildung C.5.: Beim Schritt sichtbare Hinweise in der BlueJ 3.1.0 Version von BlueJVisualize

Quelle: Eigene Darstellung

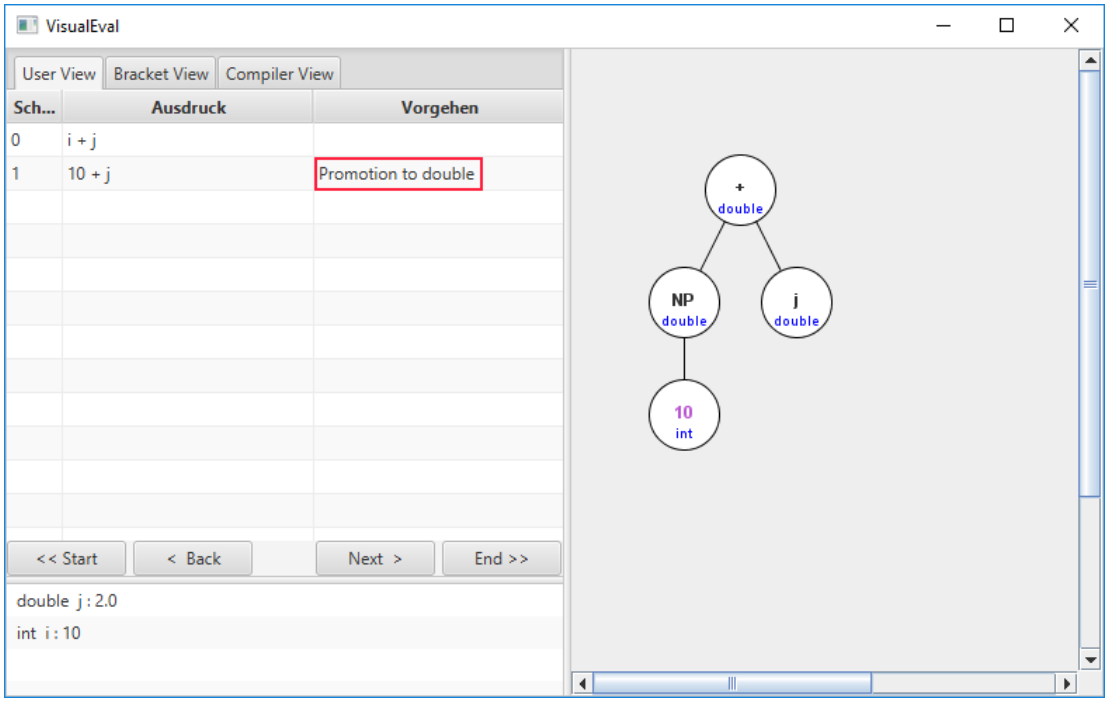


Abbildung C.6.: In der Auswertungstabelle sichtbare Hinweise in der BlueJ 4.1.2 Version von BlueJVisualize

Quelle: Eigene Darstellung

Literaturverzeichnis

[Beckert 2013] Beckert, Manuela: *Anschauliche Visualisierung von Ausdrucksauswertung in BlueJ*. 2013. – Universität Hamburg

[BlueJ 2018a] BlueJ: *Das BlueJ Logo*. 2018. – URL https://de.wikipedia.org/wiki/BlueJ#/media/File:BlueJ_Logo.png. – Zugriffsdatum: 23.08.2018

[BlueJ 2018b] BlueJ: *BlueJ Webseite - Erweiterungen*. 2018. – URL https://www.bluej.org/extensions/extensions.html. – Zugriffsdatum: 23.08.2018

[BlueJ Team 2018] BlueJ Team: *BlueJ Entwicklung - Bugtracker*. 2018. – URL http://bugs.bluej.org/browse/BLUEJ-932. – Zugriffsdatum: 23.08.2018

[Eckstein 2007] Eckstein, Robert: Java SE Application Design With MVC. (2007). – URL https://www.oracle.com/technetwork/articles/javase/mvc-136693.html. – Zugriffsdatum: 23.08.2018

[Gamma u. a. 1994] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. – ISBN 0201633612

[Gerlach 2012] Gerlach, Simon: *Die Erweiterbarkeit von MVC-Systemen durch Plug-in-Schnittstellen am Beispiel der Entwicklungsumgebung BlueJ*. 2012. – Universität Hamburg

[JUNG 2018] JUNG: *JUNG - Java Universal Network/Graph Framework*. 2018. – URL http://jung.sourceforge.net/. – Zugriffsdatum: 23.08.2018

[Kölling und BlueJ Team 2018a] Kölling, Michael ; BlueJ Team: *BlueJ Webseite*. 2018. – URL https://www.bluej.org/. – Zugriffsdatum: 23.08.2018

[Kölling und BlueJ Team 2018b] Kölling, Michael ; BlueJ Team: *BlueJ Webseite - Versionshistorie*. 2018. – URL https://bluej.org/versions.html. – Zugriffsdatum: 23.08.2018

[Kölling und BlueJ Team 2018c] Kölling, Michael ; BlueJ Team: *BlueJ Webseite - Übersicht Erweiterungsschnittstelle*. 2018. – URL https://www.bluej.org/doc/extensionsAPI/. – Zugriffsdatum: 23.08.2018

[Kölling u. a. 2018] Kölling, Michael ; Brown, Neil ; Altadmri, Amjad ; McKay, Fraser: *Frame-Based Editing*. 2018. – URL https://www.greenfoot.org/frames/. – Zugriffsdatum: 23.08.2018

[Kölling u. a. 2017] Kölling, Michael ; Brown, Neil C. C. ; Altadmri, Amjad: Frame-Based Editing. In: *Journal of Visual Languages and Sentient Systems* 3 (2017), 6

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 24. August 2018 Lennart Borchert