Universität Paderborn

Fachbereich 17 - Mathematik/Informatik
Arbeitsgruppe Softwaretechnik
Prof. Dr. W. Schäfer
Warburger Str. 100
33098 Paderborn

Dokumentation

Guided Tour

Version 0.1alpha, Stand 20.3.02

i.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

FUJABA ist das Akronym für “From UML to Java And Back Again” und ist eine Entwicklungsumgebung, ein sogenanntes CASE-Tool (Computer Aided Software Engineering), dass den Entwurf von Software mit Hilfe der Unified Modeling Language (UML) ermöglicht. Die UML umfasst eine Reihe von verschiedenen Diagrammarten, mit denen die verschiedene Aspekte (Struktur und Verhalten) einer Software modelliert (entworfen) werden können. Um vom Entwurf, im Allgemeinen mehrere UML-Diagramme, zu einem ausführbaren Programm zu kommen beinhaltet Fujaba einen Mechanismus, der für die Diagramme des Entwurfs ausführbaren Java Quelltext generiert. Dieser Quelltext kann mit Hilfe eines Java-Compilers wie ihn zum Beispiel SUN’s JDK zur Verfügung stellt kompiliert und ausgeführt werden. Zu Fujaba gehört ebenfalls DOBS (Dynamic Object Browsing System), ein Werkzeug, dass die internen Objektstrukturen der Java Virtual Machine visualisiert und durch Methodenaufrufen auf Objekten Veränderungen in der internen Objektstruktur sichtbar machen kann. Die Erstellung von Benutzeroberflächen wird ebenfalls durch Fujaba unterstützt. Fujaba enthält eine einfache Bibliothek (FGrafik) zur Erstellung von Eingabemasken und Fenstern. Die FGrafik baut dabei direkt auf der Java Swing Bibliothek auf und erlaubt es schnell Prototypen mit Benutzerinteraktion zu erstellen.

Dieses Anleitung soll dem Leser eine erste Einführung in die Entwicklungsumgebung geben. Nach einer Installationsbeschreibung wird am Beispiel eines Roulettespiels die Verwendung der Entwicklungsumgebung demonstriert. Zuerst wird das logische Modell mit Hilfe von Klassendiagrammen, Aktivitätsdiagrammen und Kollaborationsdiagrammen erstellt. Anschließend wird das Modell mit Hilfe von Dobs getestet und mit einer Oberfläche angereichert.

1.1 Einführung

In den 70er Jahren sind verschiedene Ansätze von objektorientierten Modellierungssprachen entstanden. Die bekanntesten Vertreter sind die Sprachen OMT von James Rumbaugh, OOSE von Ivar Jacobson und Booch-Diagramme. Mitte der 90er Jahre haben die drei Entwickler gemeinsam bei der Rational Software Corporation ihre Ansätze vereinigt und die Unified Modeling Language (UML) entwickelt. Die UML unterstützt acht verschiedene Diagrammarten zur Modellierung sowohl statischer als auch dynamischer Zusammenhänge und unterstützten damit den Unified Process für die Entwicklung von Softwaresystemen. Zusätzlich zu den acht Diagrammarten enthält die UML die Object Constraint Language (OCL), die es erlaubt Konsistenzbedingungen zu formulieren. Im Gegensatz zur OCL, die eine textuelle Sprache verwendet, verwenden die verschiedenen Diagrammarten graphische Sprachelemente. Dadurch

sind die Diagramme leicht verständlich und ermöglichen es Softwaresysteme auf einem hohen Abstraktionsgrad zu spezifizieren. Implementierungsdetails treten beim Entwurf in den Hintergrund und Modellierungsaspekte mit der Diskussion um Vor- oder Nachteile treten in den Vordergrund.

Leider enthält die UML keine vollständige Semantikbeschreibung, so dass die Implementierung eines Systems per Hand erfolgen muss. Aus diesem Grund gibt es heutzutage nur sehr wenige Werkzeuge, die aus einer Spezifikation eines Softwaresystems ein lauffähiges Programm erzeugen können.

In der Entwicklungsumgebung Fujaba ist die Semantik der verwendeten Sprache durch die Verwendung von Graphgrammatiken festgelegt worden. Damit ist es möglich Quelltext aus einer Spezifikation zu erstellen. Das heißt Fujaba erzeugt für eine konsistente Spezifikation eines Softwaresystems Java Quelltexte, die mit einem herkömmlichen Kompiler übersetzt und auf einer Virtual Machine (VM) ausgeführt werden können. Anhand eines Beispiels werden im Folgenden die einzelnen Diagrammarten und ihre Zusammenhänge erläutert und mit Hilfe einer “Guided Tour” die Bedienung des Werkzeugs vorgestellt.

In diesem Dokument wird in erster Linie die Verwendung der Entwicklungsumgebung Fujaba erläutert. Somit ist diese Anleitung kein Ersatz oder eine allgemeine Einführung in die Modellierungsprache UML.

1.2 Das Roulette Spiel

Als Anwendungsbeispiel, anhand dessen die Bedienung der Entwicklungsumgebung erläutert wird, dient eine vereinfachte Form des Roulettespiels. Das allgemein bekannte Roulettespiel mit all seinen Feinheiten ist als Einstiegsbeispiel zu kompliziert, so dass hier eine stark vereinfachte Variante beschrieben werden soll.

Der Spielablauf läuft wie folgt ab: Jeder Spieler setzt auf eine Zahl zwischen 0 und 36 des Roulettes. Anschließend wird die Kugel in Bewegung gesetzt und bleibt nach einiger Zeit auf einem Feld liegen. Gewonnen haben alle Spieler, die ihren Einsatz auf das Gewinnfeld gesetzt haben. Der Gewinn berechnet sich aus dem Einsatz multipliziert mit der Anzahl der Felder (37). Damit ist eine Spielrunde beendet. Das Spiel endet, wenn alle Spieler ihr Startkapital verloren haben, oder nicht mehr weiter spielen möchten.

1.3 Oberfläche, Bedienhinweise

1.3.1 Fensteraufbau

Abbildung 1, „Benutzeroberfläche,“ auf Seite 7 zeigt die Benutzungsoberfläche der Entwicklungsumgebung Fujaba. Das Fenster ist aufgeteilt in ein Menü am oberen Rand. Jedes Menü

kann entweder Aktionen oder Untermenüs enthalten. Unterhalb der Menüs befindet sich die “Allgemeine Toolbar”, in der allgemeine Funktionen von Fujaba zu erreichen sind. Die einzelnen Funktionen sind dabei identisch mit den entsprechenden Menüs und stellen damit eine Abkürzung für häufig benutzte Aktionen dar. Unterhalb der Allgemeinen Toolbar wird abhängig von der aktuellen Diagrammart eine zweite Toolbar, die “Spezifische Toolbar”, eingeblendet. Am unteren Rand des Fenstern befindet sich die Statusleiste von Fujaba, in der eine Meldezeile, der aktuelle Speicherbedarf und die Anzahl von Inkonsistenzen im Modell aufgeführt sind.

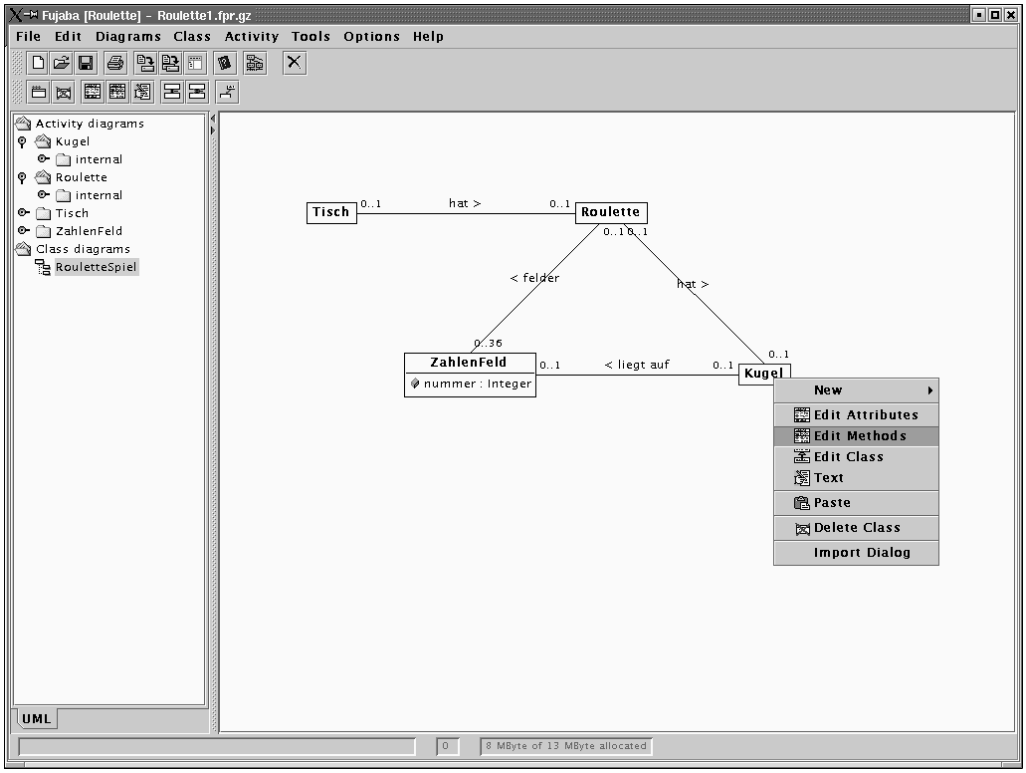


Abbildung 1: Benutzeroberfläche

1.3.2 Navigation

Das Hauptfenster der Entwicklungsumgebung Fujaba beinhaltet auf der links Seite den sogenannten Projektbaum, der es ermöglicht von einem Diagramm zu einem anderen zu springen. Die Diagramme sind gemäß ihrer Art gruppiert, so dass man zum Beispiel alle Klassendiagramme und Aktivitätsdiagramme über einen jeweils gemeinsamen Knoten im Baum erreichen kann. Unterhalb der einzelnen Diagramme können verschiedene Sichten eines Diagramms liegen.

1.3.3 Sichten

Für den Entwurf von großen Diagramme stellt die Entwicklungsumgebung Fujaba ein Sichtenkonzept zur Verfügung, Vereinfacht kann man Sichten in Fujaba mit einem Fenster in einem

Haus vergleichen. Jedes Fenster des Hauses stellt eine Sicht dar, durch den man einen Teil des Umlandes sehen kann. Durch manche Fenster kann man in unterschiedliche Himmelrichtungen sehen, durch andere sind teilweise die gleichen Teile sichtbar.

Das generische Sichtenkonzept ein Fujaba erlaubt es große Diagramme in einzelne Teile aufzuteilen. Diagrammelemente in einer Sicht sind Kopien der Elemente auf dem originalen Diagramm. Welche Elemente in einer Sicht angezeigt werden wird aufgrund von Regeln bestimmt. Der Regelsatz einer Sicht kann interaktiv zur Laufzeit erweitert werden. Ebenso ist es möglich neue Regeln zur Laufzeit zu erzeugen und zu einem Regelsatz hinzuzufügen.

Änderungsoperationen an Diagrammelementen werden allerdings nicht nur in der Sicht, in der die Änderungen eingegeben werden, ausgeführt, sondern auf dem ursprünglichen Diagramm. Dies verhindert Fehler und Inkonsistenzen im Modell.

1.3.4 Kontextabhängige Menüs

Zusätzlich zu den Menüs und Toolbars existieren für einzelne Diagrammelemente kontextabhängige Menüs (Popup-Menüs oder kurz Popups), die mit einem Klick mit der rechten Maustaste aktiviert werden. Die Popup-Menüs sind für jedes Diagrammelement individuell eingestellt und sind teilweise nicht nur vom Element selbst abhängig, sondern auch von der Position innerhalb des Elements. Es kann also durch aus sein, dass unterschiedliche Popup-Menüs erscheinen, wenn auf den Rand oder innerhalb eines Kastens geklickt wird. Im Allgemeinen beinhalten die Popup-Menüs mehr Aktionen als über die Toolbar und die Menüs erreicht werden können.

KAPITEL 2

Installation und Systemvoraussetzungen

2.1 Installation

In diesem Kapitel erfahren Sie, wie Sie Fujaba auf Ihrem Computer installieren.

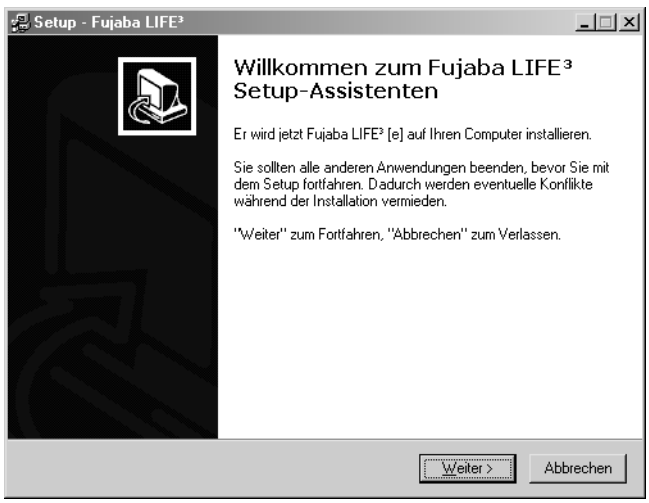
Bitte installieren Sie, falls noch nicht vorhanden, vorher das Sun Java Software Development Kit (SDK) 1.3.1 oder höher auf Ihrem Computer.

Sie erhalten es kostenlos bei Sun Microsystems Inc. unter http://java.sun.com/j2se/.

Auf der Fujaba LIFE³ Premium CD ist das SDK bereits enthalten und wird vom Fujaba Setup-Assistenten bei Bedarf automatisch installiert.

2.1.1 Installation von Fujaba LIFE³ unter Windows

Starten Sie den Fujaba Setup-Assistenten (durch Doppelklick auf die Datei **setup.exe**), der Sie durch die Installation von Fujaba führen wird.

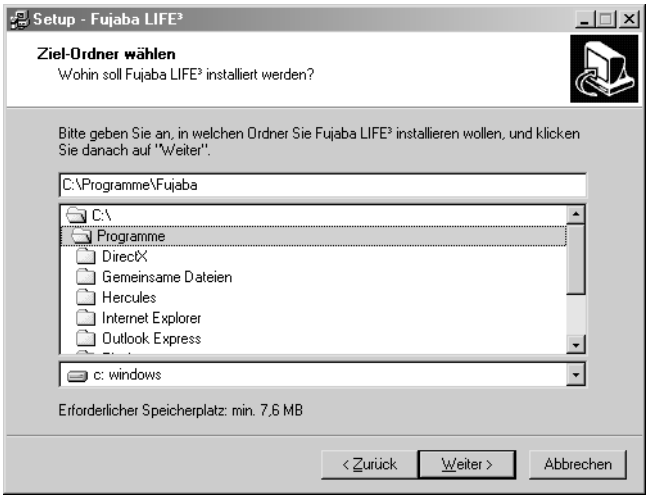


Klicken Sie dann auf die Schaltfläche *Weiter>* und lesen Sie sich die angezeigte **Lizenzvereinbarung** durch.

Wenn Sie mit den Bestimmungen der **Lizenzvereinbarung** einverstanden sind, fahren sie mit *Ja* fort, andernfalls brechen Sie den Setup-Assistenten mit der Schaltfläche *Nein* ab.

Auf der folgenden Seite erfahren Sie interessante **Informationen** über Fujaba, wenn Sie bereit sind fortzufahren, klicken Sie auf die Schaltfläche *Weiter>*.

Ziel-Ordner wählen



Wählen Sie nun Laufwerk und Ziel-Ordner, in den Fujaba installiert werden soll. Klicken Sie dann auf die Schaltfläche *Weiter>*.

Komponenten auswählen

Wählen Sie nun die zu installierenden Komponenten.

Sie haben die Wahl zwischen:

- **Vollständiger Installation**

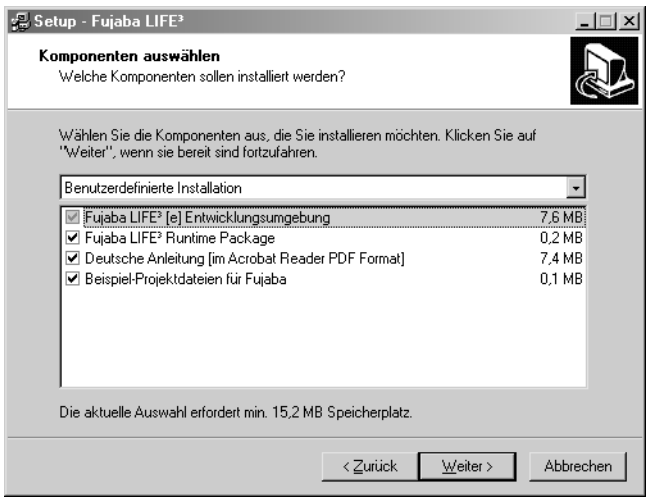
  alle Komponenten werden Installiert

- **Kompakte Installation**

  Fujaba LIFE³ Entwicklungsumgebung wird installiert

- **Benutzerdefinierte Installation**

  zu installierende Komponenten werden vom Benutzer gewählt



Dabei stehen folgende Komponenten zur Auswahl:

- **Fujaba LIFE³ Runtime Package**

  Wird benötigt, um von Fujaba erzeugten Code unabhängig von der Fujaba Entwicklungsumgebung ausführen zu können. Zu diesem Zweck kann bei Bedarf das Fujaba LIFE³ Runtime Package (fujaba-runtime.jar) in den Klassenpfad genommen werden.

- **Deutsche Anleitung**

  Installiert eine deutsche Anleitung für Fujaba, Dobs und FGrafik in den Unterordner *Documents* im Fujaba Installationsordner.

- **Beispiel-Projektdateien für Fujaba**

  Installiert Beispiel-Projektdateien, die in der Anleitung vorgestellt werden oder zu Demonstrationszwecken dienen.

Startmenü-Ordner auswählen

Wählen Sie hier einen Eintrag des Startmenüs, unter den der Fujaba Setup-Assistent die Verknüpfungen zu Fujaba anlegen soll. Klicken Sie dann zum fortfahren auf *Weiter>*.

Installation durchführen

In dieser Zusammenfassung werden noch einmal alle getroffenen Einstellungen der Installation angezeigt. Wenn Sie mit den Einstellungen zufrieden sind, klicken Sie auf *Installieren* um die Installation abzuschließen. Die gewählten Komponenten werden jetzt auf Ihrem Computer installiert und entsprechende Verknüpfungen im Startmenü angelegt.

Starten von Fujaba

Zum Starten von Fujaba klicken Sie auf die Verknüpfung *Fujaba LIFE* in Ihrem Startmenü oder doppelklicken sie auf ein Fujaba Projekt mit der Dateiendung *.fpr* .

2.1.2 Installation unter anderen Systemen mit InstallAnywhere

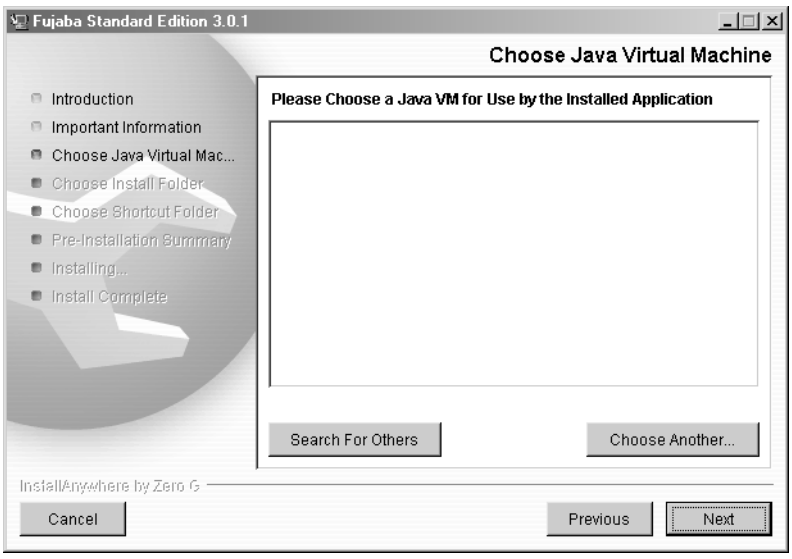
Stellen Sie zunächst sicher, das Sie eine Version des Sun Java Software Development Kit (SDK) 1.3.1 oder höher auf Ihrem Computer installiert ist.

Starten Sie dann den InstallAnywhere Assistenten, der Sie durch die Installation von Fujaba führen wird, durch das Kommando *java -jar "InstallerName"*, wobei InstallerName den Namen des InstallAnywhere Packets angibt.

z.B. : java -jar Fujaba_Developer_3.jar

Nach erscheinen der InstallAnywhere **Introduction** klicken Sie auf die Schaltfläche *Next* und lesen Sie sich die angezeigte **Information** durch. Klicken Sie dann erneut auf *Next*.

Choose Virtual Machine



Wählen Sie hier die Java Virtual Machine (VM), die zum Starten von Fujaba verwendet werden soll. Sollte keine VM angezeigt werden, klicken Sie auf die Schaltfläche *Search For Others*, um nach installierten VM auf Ihrem Computer zu suchen. Sollte das ebenfalls nicht zu befriedigen-

den Ergebnissen führen, können Sie mithilfe der Schaltfläche *Choose Another...* selbst die zu verwendende VM spezifizieren.

Es ist dringend davon abzuraten, unter Windows Betriebssystemen die Microsoft VM zu benutzen, auch wenn diese von InstallAnywhere vorgeschlagen wird, da diese nicht mit den von Sun definierten Java Spezifikationen konform läuft.

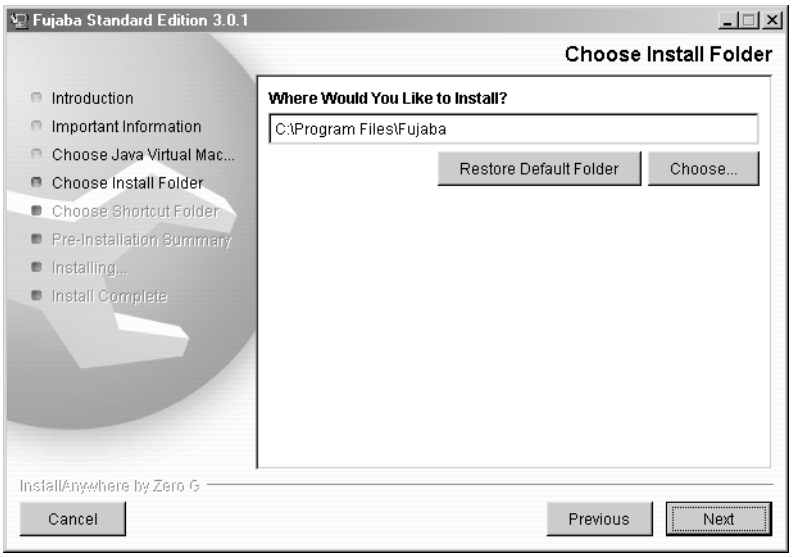
Die Microsoft VM befindet sich im Installationsordner Ihres Microsoft Betriebssystems (in der Regel C:\WINDOWS\) und umfasst folgende Executables:

jview.exe

wjview.exe

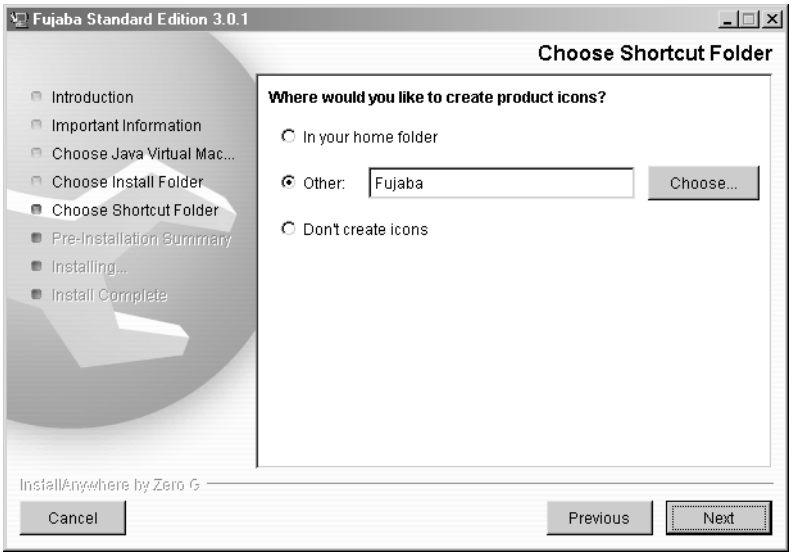
java.exe.

Choose Install Folder



Wählen Sie nun Laufwerk und Ziel-Ordner, in den Fujaba installiert werden soll, indem Sie auf die Schaltfläche *Choose...* klicken. Klicken Sie dann auf die Schaltfläche *Next*. Über die Schaltfläche *Restore Default Folder* können Sie jederzeit den Zielordner auf die Vorgabe zurücksetzen.

Choose Shortcut Folder



Wählen Sie hier die Position, unter den InstallAnywhere die Verknüpfungen zu Fujaba anlegen soll. Wenn Sie *Don't create icons* wählen, werden keine Verknüpfungen erstellt.

Klicken Sie dann zum Fortfahren auf *Weiter>*.

Pre-Installation Summary

In dieser Zusammenfassung werden noch einmal alle getroffenen Einstellungen der Installation angezeigt. Wenn Sie mit den Einstellungen zufrieden sind, klicken Sie auf *Install* um die Installation abzuschließen. Die gewählten Komponenten werden jetzt auf Ihrem Computer installiert und entsprechende Verknüpfungen im Startmenü angelegt.

Starten von Fujaba

Achten Sie darauf, das sich das Package tools.jar aus dem SDK (/lib/tools.jar) in der CLASSPATH Umgebungsvariablen Ihres Betriebssystems befindet. Fujaba benötigt dieses Package zwingend zur Ausführung.

Zum Starten von Fujaba klicken Sie auf die entsprechende *Fujaba* Verknüpfung in Ihrem Startmenü.

Sollten Sie sich entschlossen haben unter **Choose Shortcut Folder** keine Verknüpfungen mit Fujaba zu erstellen, wechseln sie auf einer Konsole in den unter **Choose Install Folder** gewählten Installationsordner und starten Sie Fujaba mit dem Kommando

java -mx128m -jar fujaba.jar

2.1.3 Manuelle Installation von Fujaba

Stellen Sie auch hier zunächst sicher, das Sie eine Version des Sun Java Software Development Kit (SDK) 1.3.1 oder höher auf Ihrem Computer installiert ist.

Achten Sie ferner darauf, das sich das Package tools.jar aus dem SDK (/lib/tools.jar) in der CLASSPATH Umgebungsvariablen Ihres Betriebssystems befindet. Fujaba benötigt dieses Package zwingend zur Ausführung. Alternativ können Sie es auch beim Starten von Fujaba spezifizieren (siehe Beispiel unten).

Entpacken Sie dann das Fujaba Installations-ZIP-Archiv mit Hilfe eines ZIP-Tools im gewünschten Installationsordner. Wechseln Sie in diesen Ordner und starten Sie Fujaba durch das Kommando

java -mx128m -jar fujaba.jar

oder, wenn Sie tools.jar nicht in der CLASSPATH Umbebungsvariable aufgenommen haben,

java -cp <jdkdir>\bin\tools.jar -mx128m -jar fujaba.jar

z.B. *java -cp C:\JDK\bin\tools.jar -mx128m -jar fujaba.jar* für ein Windows System

2.2 Systemvoraussetzungen

Fujaba Benötigt:

Java kompatibles Betriebssystem (Windows, Linux, UNIX)

Java Software Development Kit (SDK) 1.3.1 oder höher

AMD Duron 600MHz oder vergleichbar schnelle CPU

64MB RAM

20MB Festplattenplatz

Empfohlen:

Java Software Development Kit (SDK) 1.4

AMD Athlon 1GHz oder vergleichbar schnelle CPU

512MB RAM

KAPITEL 3

Konfiguration und Systemarchitektur

KAPITEL 4

Klassendiagramme

Ein zentraler Bestandteil der UML sind die Klassendiagramme. Mittels Klasssendiagrammen wird die statische Struktur eines Softwaresystems beschrieben, indem Klassen und deren Beziehungen untereinander grafisch dargestellt werden. Eine Klasse beschreibt hierbei eine Menge von Objekten mit gemeinsamen Merkmalen und einem identischen Verhalten.

Der Entwurf eines Softwaresystems beginnt in der Regel mit der Erstellung eines oder mehrerer Klassendiagramme, welche im Laufe des Entwicklungsprozesses immer wieder angepaßt werden.

Bevor mit der Erstellung eines UML Klassendiagramms begonnen werden kann, müssen zunächst die Klassen identifiziert werden. Der Prozeß der Klassenfindung ist allerdings kein Bestandteil der UML. Eine Möglichkeit besteht jedoch darin, alle Substantive aus dem Anforderungsdefinitionsdokument auszuwählen. Ausgehend von diesen Worten werden nun doppelte Vorkommen, verschieden Substantive für das selbe Objekt, sowie Worte, die offensichtlich nichts mit dem zu realisierenden System zu tun haben, gestrichen. Alle nun verbleibenden Substantive sind „Kandidaten“ für eine Klasse.

Diese Klassen können nun in einem UML Klassendiagramm näher spezifiziert und in Beziehung zueinander gesetzt werden. Die Elemente, die von der UML hierfür zur Verfügung gestellt werden, sollen nun im folgenden Abschnitt detailliert beschrieben werden.

4.1 Syntax und Semantik

Klassen

Der Hauptbestandteil eines UML Klassendiagramms ist die Klasse. In einer Klasse werden Attribute, Methoden und die Semantik definiert, die alle Objekte dieses Typs zur Laufzeit des Programms besitzen.

In der UML wird eine Klasse als ein einfaches Rechteck dargestellt, welches in drei Zeilen unterteilt wird. Die erste Reihe enthält den Namen der Klasse. Die zweite Zeile listet die definierten Attribute und deren Typ sowie der Sichtbarkeit dieser Variablen auf. In der untersten Reihe stehen die vorhandenen Methoden mit deren Parametern und Rückgabewerten, sowie ebenfalls der jeweiligen Sichtbarkeit.

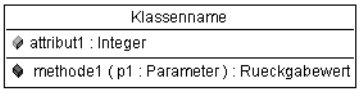


Abbildung 2: Darstellung einer Klasse in Fujaba

Sichtbarkeiten

In der UML sind verschiedene Arten von Sichtbarkeit für Attribute und Methoden definiert. In FUJABA wird die Sichtbarkeit durch das Symbol vor dem jeweiligen Attribut- oder Methodennamen gekennzeichnet.

- Public 

Durch *public* werden Attribute oder Methoden als öffentlich definiert. Somit kann von überall auf sie zugegriffen werden.

- Private 

Werden Methoden oder Attribute mit *private* deklariert, so darf auf diese nur von Methoden innerhalb der Klasse zugegriffen werden, in der die Definition erfolgte. Auch in möglichen Unterklassen ist die Definition nicht sichtbar.

- Protected 

Protected ermöglicht einen Zugriff aus der eigenen Klasse heraus, aus möglichen Unterklassen, sowie aus anderen Klassen, die im selben Package liegen.

- Package 

Die Sichtbarkeit *package* ist eine Einschränkung der *protected* Sichtbarkeit. Hierbei sind Methoden und Attribute aus Unterklassen heraus nicht sichtbar.

Beziehungen zwischen Klassen

In einem Klassendiagramm werden nicht nur die Eigenschaften einzelner Klassen beschrieben. Vielmehr kann auch dargestellt werden, wie die einzelnen Klassen In Beziehung zu einander stehen.

In UML Klassendiagrammen werden verschiedene Beziehungstypen definiert, welche im Folgenden beschrieben werden.

Vererbung

Vererbung wird häufig dann eingesetzt, wenn eine Klasse eine speziellere Ausprägung einer anderen Klasse ist. Durch Vererbung werden die Eigenschaften, wie zum Beispiel Attribute und Methoden, der so genannten Oberklasse, an die erbende Unterklasse weitergegeben.

Eine Vererbung wird durch einen Pfeil mit geschlossener Spitze von der Unterklasse auf die Oberklasse dargestellt.

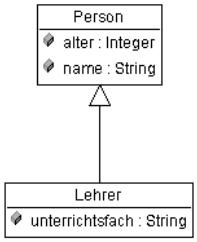


Abbildung 3: Vererbung

Assoziation

Eine Assoziation wird verwendet, um eine Beziehung zwischen zwei Klassen darzustellen. Sie wird durch eine einfache Linie zwischen den beiden Klassen dargestellt.

Die Informationen über die Art der Beziehung wird durch den Namen der Assoziation gegeben. Welche Rollen die beteiligten Objekte in dieser Beziehung spielen, wird durch ihre Rollennamen, die am jeweiligen Enden einer Assoziation angezeigt werden, erläutert.

Des weiteren kann durch so genannte Kardinalitäten beschrieben werden, wie viele Objekte an der Assoziation beteiligt sind. Kann ein Objekt zum Beispiel zwischen 0 und 5 Referenzen auf Objekte eines bestimmten Typs haben, so wird dies durch *0..5* dargestellt. Ist die Anzahl der Beziehungen beliebig, so wird dies in der Regel durch *0..** oder *0..n* dargestellt werden.

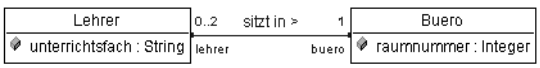


Abbildung 4: Einfache Assoziation

Aggregation

Eine Aggregation wird verwendet, wenn eine Klasse eine andere Klasse enthält. So besteht ein Buch zum Beispiel aus mehreren Kapiteln. Die Klasse Kapitel wäre in diesem Fall die aggregierte Klasse.

Aggregationen sind so zu sagen Assoziationen, welchen den Namen *hat* tragen.

Dargestellt werden Aggregationen durch eine Linie zwischen den Klassen, wobei an der Seite des aggregierenden Objekts eine Raute gezeichnet wird.

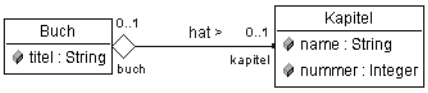


Abbildung 5: Aggregation

Komposition

Die Kompostion ist ein Spezialfall der Aggregation. Sie wird eingesetzt, wenn eine Klasse Bestandteil einer anderen Klasse ist und für deren Existenz zwingend erforderlich ist. Zum Beispiel kann ein Verzeichnis mehrere Dateien enthalten, wobei die Existenz eines Verzeichnisses für die Datei zwingend notwendig ist.

Eine Komposition kann immer dann eingesetzt werden, wenn man eine Assoziation mit dem Namen *besteht aus* erstellen würde.

Die Komposition wird ähnlich wie die Aggregation dargestellt. Nur die Raute wird in diesem Fall nicht leer, sondern ausgefüllt gezeichnet

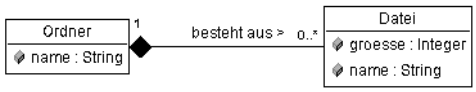


Abbildung 6: Komposition

Qualifizierende Assoziationen

Stehen zwei Objekt in Beziehung zu einander und erfolgt der Zugriff von einem der Objekt auf das andere stets über ein bestimmtes Attribut, so nennt man dies eine qualifizierende Assoziation. Das entsprechende Attribut wird als Qualifizierer bezeichnet.

Um eine Assoziation dieser Art in UML darzustellen, wird auf Seite der zugreifenden Klasse die Assoziation um ein Rechteck mit dem Namen des Qualifizierers ergänzt.

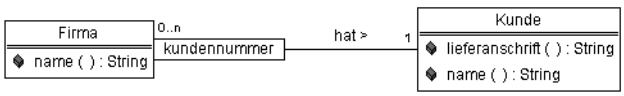


Abbildung 7: qualifizierte Assoziation

4.2 Guided Tour

Im Folgenden soll anhand eines einfachen Beispiels die Erstellung eines UML Klassendiagramms mit Hilfe von Fujaba beschrieben werden. Hierfür soll das *Roulette Beispiel*, welches auch in *fujaba/Examples/Roulette.fpr.gz* zu finden ist, genutzt werden. Zu modellieren ist ein Tisch, auf dem sich ein Roulette-Spiel befindet. Dieses Roulette-Spiel besteht aus 37 Zahlenfeldern sowie einer Kugel.

Wie in der Einleitung bereits erwähnt, können die verschiedenen Befehle mit Hilfe von Popup Menüs, der Menüleiste oder der Toolbar ausgeführt werden. Im Rahmen dieser Guided Tour wird jeweils nur eine dieser Methoden genutzt. Die anderen Vorgehensweisen sind dabei meist ebenfalls möglich und funktionieren äquivalent zur vorgestellten Variante.

Bevor mit dem Erstellen des Klassendiagramms begonnen werden kann, muß zunächst ein neues Fujaba-Projekt erstellt werden. Dies geschieht über den Menüpunkt »*File*« - »*New Project*«

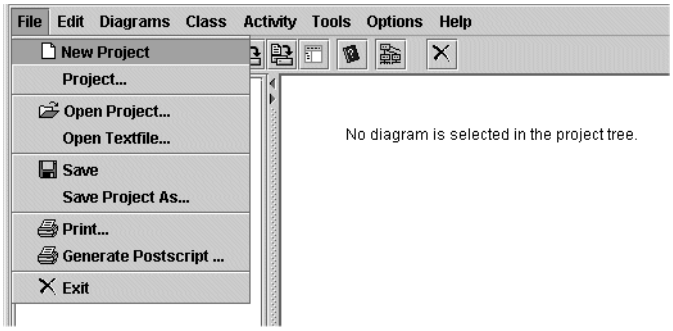


Abbildung 8: Erstellen eines neuen Projekts

Nach Ausführen dieses Befehls öffnet sich ein Dialog, in dem dem Projekt sowohl ein Name, als auch ein Pfad, in dem es gespeichert werden soll, zugewiesen wird.

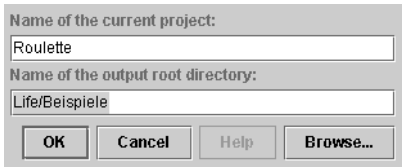


Abbildung 9: Der „New Project“ Dialog

Nun kann unter dem Menüpunkt »*Diagrams*« - »*New Class Diagram*« dem Projekt ein neues Klassendiagramm hinzufügt werden.

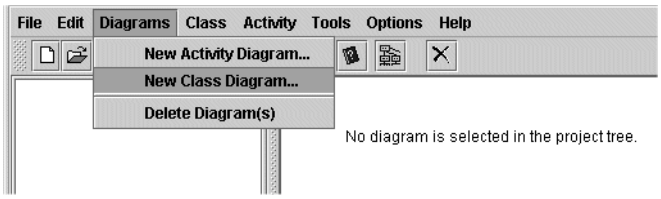


Abbildung 10: Erstellen eines Klassendiagramms

Im Folgedialog wird der Name des Diagramms eingegeben und mit »*OK*« bestätigt.

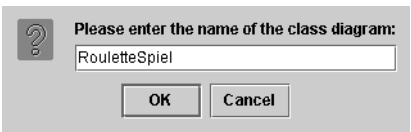


Abbildung 11: Benennen eines Klassendiagramms

Am Projektbaum (links) ist nun zu erkennen, dass das neue Klassendiagramm erstellt wurde. Er beinhaltet nun einen Ordner *Class Diagrams*, der das soeben erstellte Diagramm *RouletteSpiel* beinhaltet

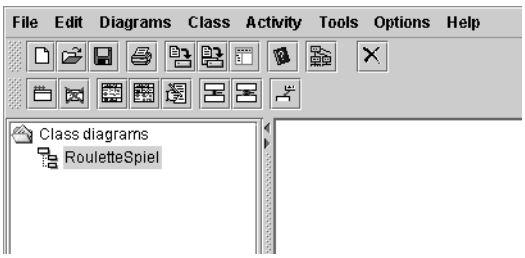


Abbildung 12: Der Projektbaum

Werden innerhalb eines Projekts mehrere Diagramme definiert, so werden diese ebenfalls im Projektbaum angezeigt. Durch anklicken der Diagrammnamen im Baum kann zwischen den einzelnen Diagrammen gewechselt werden.

Zusätzlich hat sich die Toolbar verändert. Sie wurde um spezifische Klassendiagramm Befehle erweitert.

Nachdem ein Projekt und die Umgebung für ein Klassendiagramm geschaffen wurde, kann nun mit der eigentlichen Erstellung des Klassendiagramms begonnen werden.

Im vorgestellten Roulette Beispiel soll nun zu aller erst eine Klasse „Tisch“ erstellt werden. Dies geschieht über den Menübefehl »*Class*« - »*New / Edit Class...*«

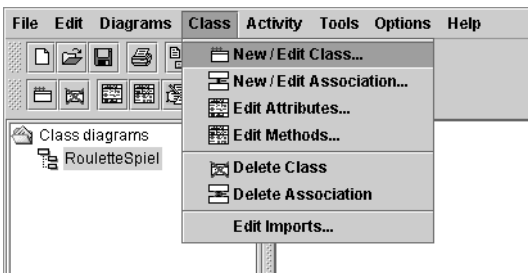


Abbildung 13: Erstellen einer neuen Klasse

Der sich nun öffnende Dialog dient zur Einstellung einiger Eigenschaften der zu erstellenden Klasse. Ein großer Teil dieser Einstellung ist wegen der Einfachheit des hier besprochenen Beispiel an dieser Stelle nicht von Relevanz. Dennoch sollen die wichtigsten Einstellungsmöglichkeiten kurz beschrieben werden.

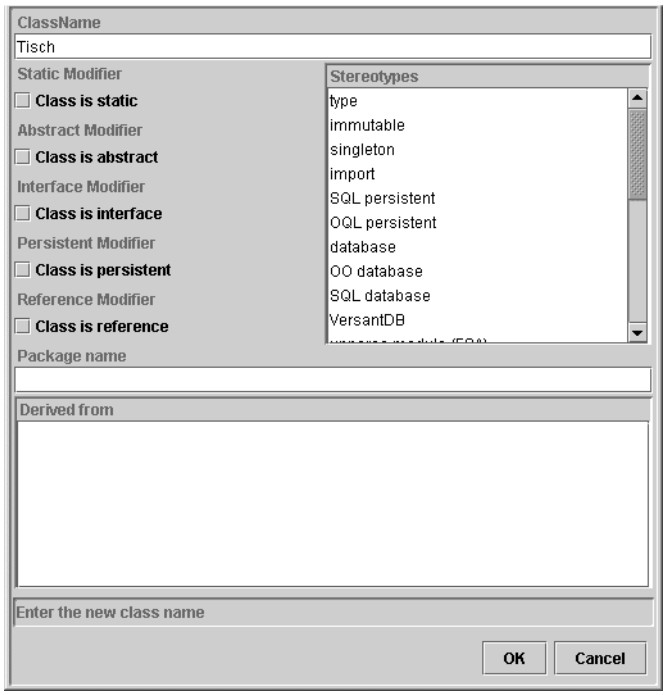


Abbildung 14: Der „New Class“ Dialog

Im obersten Textfeld wird der Name der neuen Klasse festgelegt. Links darunter können einige Eigenschaften wie *static* oder *abstract* festgelegt werden. Eine Abstrakte Klasse zum Beispiel ist eine Oberklasse in einer Vererbungshierarchie, von der keine Instanzen erzeugt werden können. Neben diesen Eigenschaften können noch mehrere Stereotypen für die Klasse ausgewählt werden.

Durch das *Package name* Textfeld kann die Klasse einem bestimmten Package zugeordnet werden. Packages sind eine Sammlung von JAVA Dateien, die gemeinsam einen gewissen Teil des Softwaresystems implementieren. Packages werden in JAVA durch Ablegen der Datein in verschiedene Verzeichnisse realisiert. In grossen Softwaresystemen werden Packages zur Steigerung der Übersichtlichkeit eingesetzt.

In dem darunterliegenden Teil können Vererbungsbeziehungen festgelegt werden. In dem Textfeld werden alle bisher erstellten Klassen aufgelistet. Zur Modellierung einer Vererbung wird hier lediglich die entsprechende Oberklasse selektiert. In diesem Fall ist die Liste jedoch leer, da gerade erst die erste Klasse erstellt wird und somit keine Oberklassen existieren können.

Nach Bestätigung des Dialogs mittels »*OK*« sollte die Klasse „Tisch“ als kleines Rechteck in der Diagrammansicht zu sehen sein.

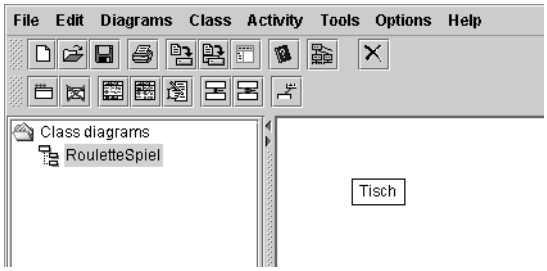


Abbildung 15: Eine neue Klasse in Fujaba

Was auffällt, ist dass die Klasse nicht wie oben beschrieben als ein Rechteck mit drei Zeilen erscheint. In FUJABA werden die zwei weiteren Zeile erst eingeblendet, wenn bereits Attribute oder Methoden für die Klasse definiert wurden.

Auf die gleiche Art und Weise können nun auch die anderen notwendigen Klassen „Roulette“, „Kugel“ und „Zahlenfeld“ erstellt werden. Dabei kann man durch ziehen der Maus mit durchgedrückter linker Maustaste die einzelnen Klassen in der Diagrammansicht beliebig positionieren, um eine sinnvolle Anordnung der Klassen zu finden.

Achtung: Beim Erstellen weiterer Klassen sollte darauf geachtet werden, dass der Menübefehl zum Erstellen einer Klasse ebenfalls zum ändern einer bereits vorhandenen Klasse dient. (»*New / Edit Class*«). Hat man also eine Klasse in der Diagrammansicht selektiert, so wird mit diesem Befehl keine neue Klasse erstellt, sondern die selektierten Klasse modifiziert. Es sollte also darauf geachtet werden, dass vor dem Erzeugen einer neuen Klasse keine andere Klasse selektiert ist. Durch einen Klick auf eine freie Fläche in der Diagrammansicht werden alle Selektionen aufgehoben.

Nachdem alle Klassen erstellt worden sind, sollte die Diagrammansicht etwa so aussehen

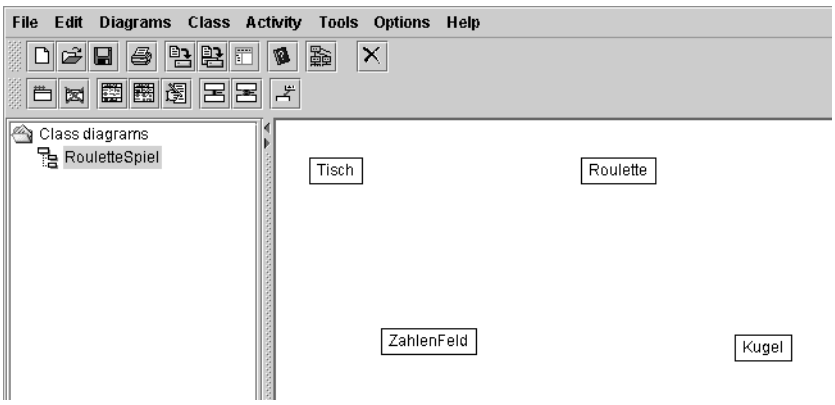


Abbildung 16: Roulette Beispiel nach dem Erstellen aller Klassen

Als nächstes müssen die Beziehungen zwischen den einzelnen Klassen festgelegt werden. An dieser Stelle soll mit der Beziehung zwischen den Klassen „Tisch“ und „Roulette“ begonnen werden. Vor dem eigentlichen Einfügen der Assoziation müssen die beteiligten Klassen selek-

tiert werden. Die erste Klasse „Tisch“ kann zunächst wie gewohnt selektiert werden. Dann wird die Klasse „Roulette“ mit gehaltener *STRG* Taste ebenfalls angeklickt. Die Klassen werden nun blau umrandet dargestellt, was die Selektion der Objekte anzeigt. Über »*Class*« - »*New / Edit Association*« kann der Assoziationsdialog aufgerufen werden.

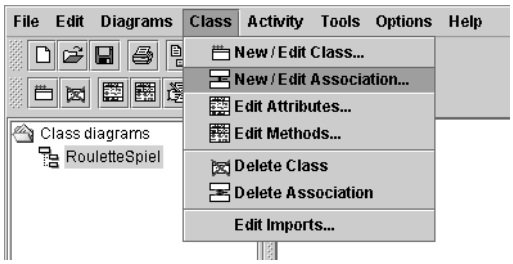


Abbildung 17: Erstellen einer neuen Assoziation

Der Dialog zur Erstellung neuer Assoziation sieht wie folgt aus:

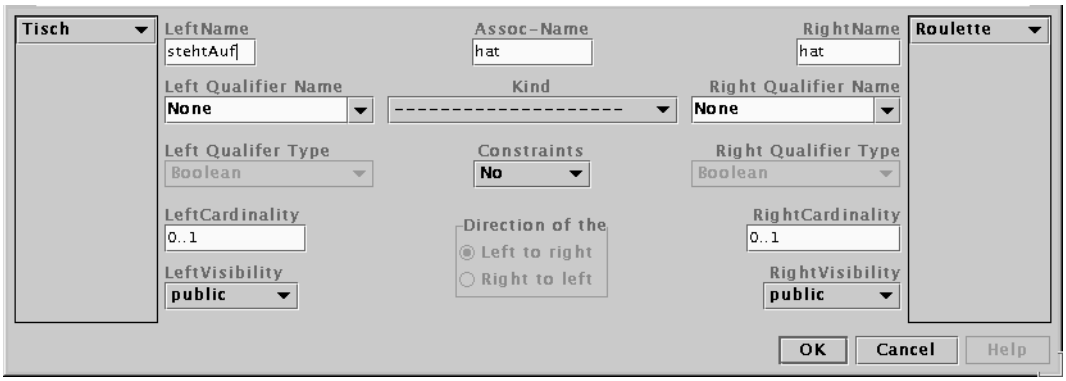


Abbildung 18: Der „New Association“ Dialog

Die Spalten am rechten beziehungsweise linken Rand des Dialogs zeigen die an der Assoziation beteiligten Klassen an. Durch die Selektion im letzten Schritt werden hier bereits die richtigen Klassen angezeigt. Die Auswahl der Klassen kann an dieser Stelle natürlich noch geändert werden.

Unter *Assoc Name* wird zunächst der Name der Assoziation eingetragen. Die Eingabefelder *Left* und *Right Name* sind zur Vergabe der Rollenamen gedacht. Unter *Kind* kann einer der Beziehungstypen (siehe Kapitel 4.1) ausgewählt werden. In diesem Beispiel wird eine gewöhnliche Assoziation gewählt. Weiterhin soll bei der Modellierung davon aus gegangen werden, dass auf einem Tisch ein oder kein Roulettespiel steht und dass ein Roulette entweder auf einem oder auf keinem Tisch steht. Deswegen bleiben die Kardinaliäten unverändert.

Nach dem der Dialog mit OK bestätigt wurde ist nun die Assoziation in der Diagrammansicht zu sehen.

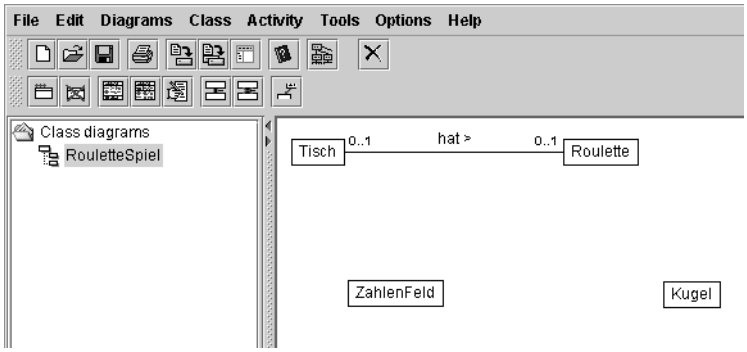


Abbildung 19: Anzeige einer Assoziation in FUJABA

Die Assoziation trägt wie angegeben den Namen „hat“. Der Pfeil „>“ hinter dem Assoziationsnamen gibt die Leserichtung an. Im vorangegangen Dialog wird implizit eine Leserichtung von links nach rechts angenommen. Weiterhin werden die Kardinalitäten angezeigt. Zur Anzeige der Rollennamen, muß die Assoziation selektiert werden.

Auf die eben beschriebene Weise können nun auch die anderen Assoziationen erstellt werden. Als Beispiel für eine Assoziation ohne 0..1 Kardinalität sei die „Roulette“ - „Zahlenfeld“ Assoziation genannt. Hier würde man auf Seite des Zahlenfeldes eine 0..37 Kardinalität eintragen.

Wurden alle Assoziationen erstellt, ergibt sich folgendes Bild in der Diagrammansicht.

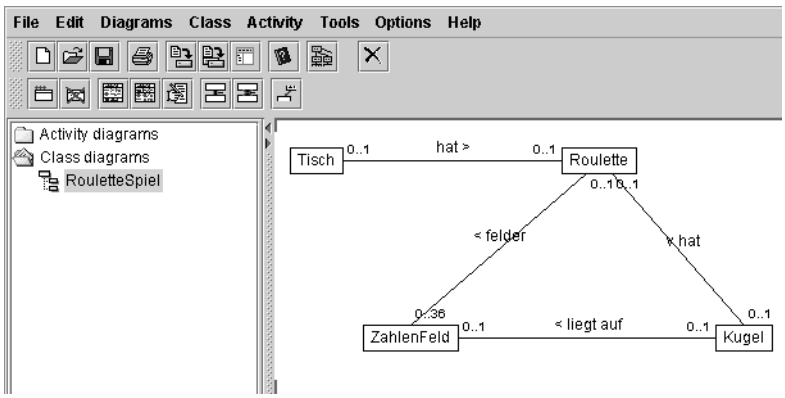


Abbildung 20: Anzeige aller Klassen und Assoziationen

Zum Schluß müssen noch die Klassen genauer spezifiziert werden. Hierzu werden ihnen die wichtigsten Attribute und Methoden zugewiesen. So sollte ein Zahlenfeld zum Beispiel eine Nummer besitzen.

Um der Klasse dieses Attribut hinzuzufügen wird mit der rechten Maustaste auf die Klasse „Zahlenfeld“ geklickt. Darauf hin öffnet sich ein Popup Menü, indem sich mehrere Befehle zum

editieren der Klasse finden. Die Attribute können über den Befehl »*Edit Attributes*« editiert werden.

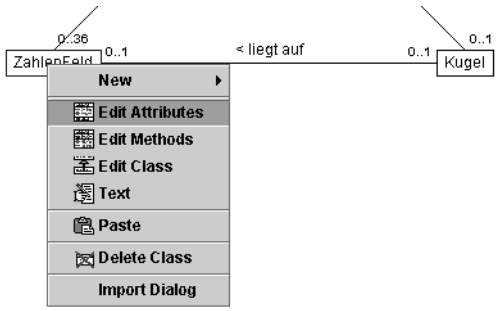


Abbildung 21: Anlegen von Attributen

Der nun erscheinende Dialog dient sowohl zum Anlegen neuer Attribute, als auch zum Ändern beziehungsweise Löschen bereits bestehender Attribute..

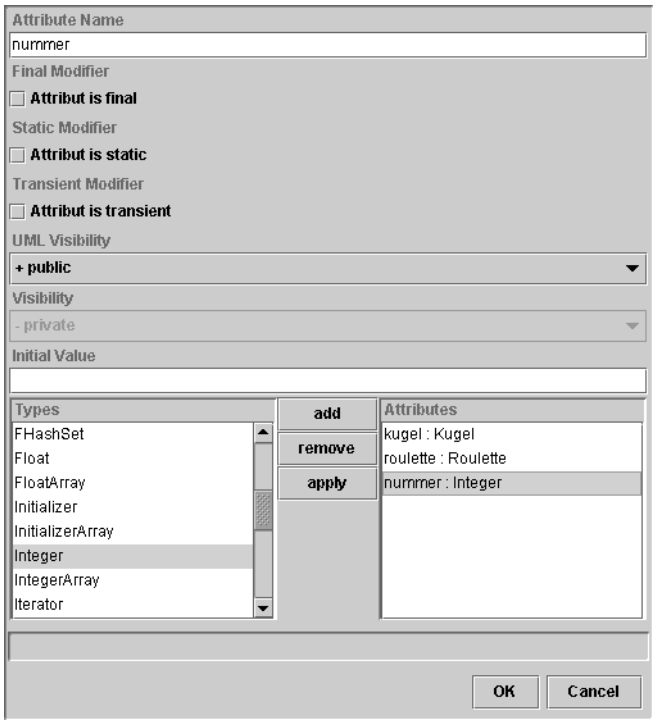


Abbildung 22: Der „Edit Attributes“ Dialog

Um ein neues Attribut zu erstellen, wird zunächst dessen Name, hier also „nummer“, in der obersten Textzeile eingegeben. Weitere Eigenschaften wie *final*, *static* oder *transparent* können direkt darunter zugewiesen werden. Diese Eigenschaften sind für das hier besprochene Beispiel jedoch unwichtig. Die Sichtbarkeit der Variable (siehe Kapitel 4.1) kann unter dem Punkt *Visibility* eingestellt werden (hier nicht notwendig).

Unter *Initial Value* kann ein Wert für die Variable festgelegt werden, der diesem Attribut gleich bei der Initialisierung zugewiesen wird.

Unter *Types* muss der Typ des Attributes ausgewählt werden. In diesem Fall ist das Attribut „nummer“ ein Integer. Der unter Initial Value angegeben Wert sollte dabei natürlich dem hier angegebenen Typ entsprechen.

Sind all diese Einstellungen vorgenommen worden, so kann das neu definierte Attribut mit einem Klick auf den »*Add*« Button erstellt werden. Das Attribut erscheint nun in der Attribut Liste im unteren rechten Teil des Dialogs. Hier werden auch die bereits definierten Attribute aufgelistet. Wird eines dieser Attribute selektiert, so nehmen alle Einstellungsfelder im Dialog die entsprechenden Werte dieses Attributs an. Diese können nach Belieben geändert werden. Ein Klick auf den »*apply*« Knopf weist dem gewählten Attribut dann die veränderten Werte zu.

Über dem »*remove*« Knopf kann ein, in der Attribut Liste selektiertes Attribut, gelöscht werden.

In der Diagrammansicht trägt die Klasse „Zahlenfeld“ nun eine Beschriftung *Collapsed*. Aus Übersichtsgründen werden in FUJABA Attribute und Methoden standardmäßig ausgeblendet, da diese bei komplexeren Systemen sehr zahlreich werden können und somit das Diagramm unübersichtlich machen.

Durch einen Doppelklick auf die entsprechende Klasse können Attribute und Methoden wieder ein- und ausgeblendet werden.

Abbildung 23: Eine Klasse „Collapsed“ und „Expanded“

Nun soll noch exemplarisch eine Methode erzeugt werden. In dem Beispiel soll davon ausgegangen werden, dass die Klasse „Tisch“ eine Methode „erzeuge“ besitzt, durch deren Aufruf das Szenario später initialisiert werden soll.

Nach einem Rechtsklick auf die Klasse Tisch erscheint erneut ein Popup Menü. Hier wird diesmal der Befehl »*Edit Methods*« ausgewählt. Der daraufhin erscheinende Dialog hat große Ähnlichkeit mit dem bereits bekannten *Edit Attributes* Dialog.

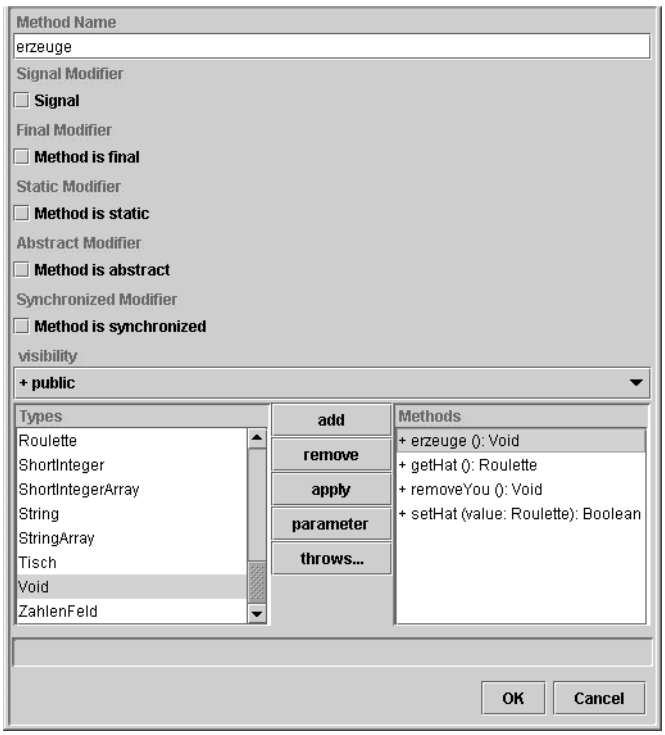


Abbildung 24: Der „Edit Methods“ Dialog

Auch die Benutzung dieses Dialogs ähnelt der, des *Edit Attributes* Dialogs. Die Liste unterhalb von „*Types*“ legt diesmal jedoch den Rückgabewert der Methode fest. Um der Funktion Parameter zuzuweisen, genügt ein Klick auf den »*parameter*« Knopf. In dem daraufhin erscheinenden Dialog können nun einfach die Parameter der Methode definiert werden.

In diesem Beispiel soll lediglich eine Methode erzeugt werden, die weder Parameter noch Rückgabewerte besitzt. Deshalb wird lediglich der Name der Methode, zum Beispiel „erzeuge“, eingetragen. Durch einen Klick auf den »*add*« Knopf wird die Methode der Klasse hinzugefügt. Daraufhin kann der Dialog mit »*OK*« bestätigt werden.

Es sei darauf hingewiesen, daß an dieser Stelle lediglich die Signatur einer Methode und nicht deren Rumpf erstellt wird.

Auf die selbe Art und Weise kann so noch die Methode „einFeldWeiter“ der Klasse Kugel erstellt werden, an welcher im nächsten Kapitel das Erstellen eines Storydiagramms beschrieben werden soll.

Das fertige Klassendiagramm für das beschriebene Beispiel könnte wie folgt aussehen:

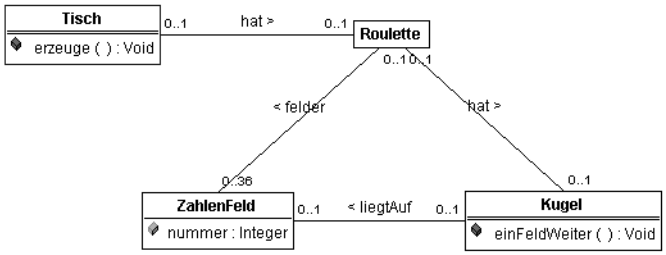


Abbildung 25: Das fertige Klassendiagramm

KAPITEL 5

Storydiagramme

In diesem Kapitel sollen die FUJABA Storydiagramme näher erläutert werden. Storydiagramme sind kein Bestandteil der UML, sondern eine Kombination aus UML Aktivitäten- und UML Kollaborarionsdiagrammen.

In Aktivitätendiagrammen wir der (zeitliche) Ablauf eines Programmteils dargestellt. Hierzu wird dieser Programmteil in einzelne Aktivitäten zerlegt, welche dann in ein eine feste Ablaufreihenfolge gebracht werden. Ein Beispiel für eine Aktivität ist zum Beispiel eine einzelne Anweisung oder ein Block von Anweisungen im Quellcode.

Kollaborationsdiagramme sollen die Interaktion oder Zusammenarbeit (englisch: *Collaboration*) zwischen mehreren (verschiedenen) Objekten darstellen.

FUJABA Storydiagramme sind Aktivitätendiagramme, die um einen eigenen Aktivitäts-Typ, den so genannten *Story-Aktivitäten*, erweitert wurden. Hierdurch wird es möglich, die Interaktion von Objekten im zeitlichen Ablauf eines Programmteils darzustellen.

Eine kurze Einführung in Aktivitätendiagramme und die Beschreibung der Syntax und Semantik von Story-Aktivitäten, wird im folgenden Abschnitt gegeben.

5.1 Syntax und Semantik

5.1.1 Aktivitätendiagramme

Wie bereits erwähnt, bauen Storydiagramme unter anderem auf UML Aktivitätendiagrammen auf. Aktivitätendiagramme bestehen aus verschiedenen Aktivitäten, welche durch Transitionen miteinander verbunden sind. Dadurch wird der sequenzielle Ablauf eines Programmteils, wie zum Beispiel eine Methode, beschrieben.

Zunächst sollen kurz die verschiedenen Aktivitätstypen aus FUJABA vorgestellt werden.

Start- und End Activities

Jedes Aktivitätendiagramm besitzt einen eindeutigen Anfangs- und Endpunkt. Diese Punkte repräsentieren den Moment, in dem die Methode aufgerufen beziehungsweiseverlassen wird. Der Startknoten wird durch einen ausgefüllten Kreis dargestellt. Über diesem Kreis werden die Klasse, der Methodenname mit den entsprechenden Parametern und Rückgabewerten angege-

ben. Der Endknoten wird, wie auch die meißten anderen Aktivitäten durch ein Rechteck mit abgerundeten Ecken dargestellt. Innerhalb dieses Rechtecks wird die Beschriftung „STOP“, sowie gegebenenfalls ein Rückgabewert angegeben.

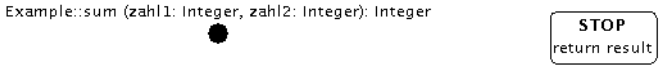


Abbildung 26: Start- und Stop Activity in FUJABA

Statement- und NOP Activities

Mittels Statement-Aktivitäten werden Anweisungen innerhalb des Programms dargestellt. Sie werden ebenfalls als Rechtecke mit abgerundeten Kanten gezeichnet. Innerhalb der Rechtecke befindet sich (JAVA) Quellcode.

Verzweigungen im Programm können mit Hilfe von NOP-Aktivitäten modelliert werden. NOP steht hier für No Operation. An solchen Stellen werden keine Operationen ausgeführt. Sie dienen der Darstellung von Verzweigungen im Ablauf des Programms. Diese Aktivitäten werden durch eine Raute dargestellt.

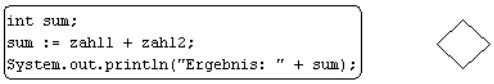


Abbildung 27: Statement- und NOP Activities in FUJABA

Transitionen

Transitionen beschreiben den Ablauf des Programms in dem sie die einzelnen Aktivitäten in eine feste Reihenfolge bringen. Sie werden als Pfeile vom Rand einer Aktivität zur nächsten dargestellt. Sollen bestimmte Bedingungen gelten, um eine Transition zu benutzen, also um in einen gewissen Programmabschnitt zugelangen, so werden diese Bedingungen in (JAVA) Quellcode formuliert und umgeben von eckigen Klammeren an die Transition geschrieben.

Als einfaches Beispiel für ein Aktivitätendiagramm sei das folgende Diagramm gegeben, welches ein Beispiel für eine Methode darstellt, die entscheidet, ob eine gegeben Zahl gerade oder ungerade ist.

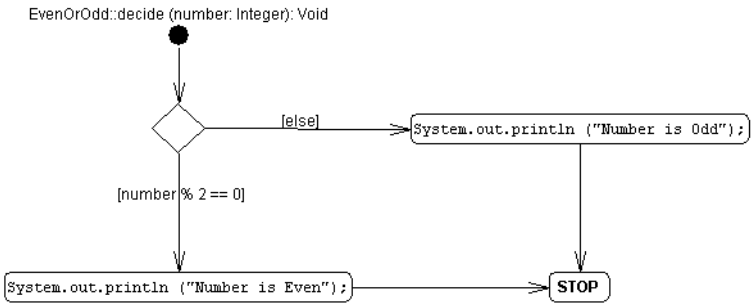


Abbildung 28: Ein einfaches Activity Diagram

5.1.2 Storyaktivitäten

Die Kollaboration zwischen einzelnen Objekten wird mittels Storyaktivitäten in ein Aktivitätendiagramm eingebracht. Diese Aktivitäten sind somit Diagramme innerhalb eines anderen Diagramms. Wie zum Beispiel Statement-Aktivitäten werden Story-Aktivitäten von einem Rechteck mit abgerundeten Ecken umgeben.

Die verschiedenen Elemente von Storyaktivitäten, sowie deren Bedeutung soll nun beschrieben werden.

Objekte

Anders als bei Klassendiagrammen, stehen hier nicht die Klassen, sonder die Objekte im Vordergrund. Objekte sind einzelne spezielle Ausprägungen einer Klasse. Man spricht auch von Instanzen einer Klasse.

Die Darstellung geschieht analog zur Klasse. Zusätzlich zum Klassennamen wird bei Objekten noch der Name des Objekts angegeben. Die UML Notation sieht dabei vor, dass zunächst der Name des Objekts, gefolgt von einem Doppelpunkt und dem Klassennamen, angegeben wird. Zur weiteren Abhebnung von einer Klasse wird der Komplette Name noch unterstrichen.

Einzige Ausnahme dieser Regel ist das so genannte *this* Objekt. Das *this* Objekt ist kontextabhängig und stellt das Objekt dar, auf dem die Methode aufgerufen wurde, die mit dem Diagramm modelliert wird. Dieses Objekt trägt nur den Namen „this“ ohne die Angabe einer Klasse,

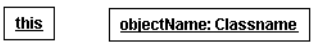


Abbildung 29: Darstellung von Objekten

In Storyaktivitäten können Änderungen, die auf Objektebene auftreten, dargestellt werden. Hier tritt Hauptsächlich das Erstellen neuer Objekte auf. Neu erstellte Objekte werden innerhalb von Storyaktivitäten in grün gezeichnet. Zusätzlich tragen sie in der oberen rechten Ecke die Beschriftung »*create*«. Zu löschende Objekte werden weiterhin in schwarz gezeichnet. Sie tragen lediglich in riter Schrift die zusatzbeschriftung »*destroy*« in der rechten oberen Ecke.

Abbildung 30: Neue und zu löschende Objekte

Das Löschen von Objekten macht nur aus konzeptioneller Sicht sinn, da ein echtes Löschen von Objekten in JAVA nicht möglich ist. Ein Objekt wir erst dann (automatisch) gelöscht, wenn keine Assoziationen mehr zu diesem Objekt bestehen.

Links

Links repräsentieren Assoziationen zwischen Objekten. Genau wie bei den Objekten selbst, wird hier in der Darstellung zwischen bereits bestehenden und unveränderten Links, sowie neuen und zu löschenden Links unterschieden.

Links werden als Linie zwischen den beteiligten Objekten dargestellt. An dieser Linie steht zusätzlich der Name der Aossziation. Die Hauptleserichtung wird wie bei Assoziationen in Klassendiagrammen durch ein „>“ dargestellt. Die Unterscheidung zwischen neuen und zu löschenden Links geschieht analog zu den Objekten. Neue Links werden in grün, alte Links in schwarz mir roter Beschriftung dargestellt.

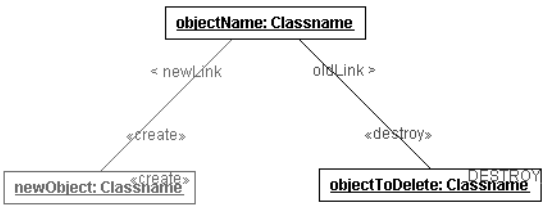


Abbildung 31: Darstellung von Links

Paths

Durch einen Pfad zwischen zwei Objekte wird angezeigt, dass es eine Verbindung von dem einen Objekt zum anderen gibt. Im Gegensatz zum Link wird durch einen Pfad jedoch keine Assoziation repräsentiert. Vielmehr kann dieser Weg über mehrere Objekte verlaufen. Durch das Zeichnen von Pfaden kann hauptsächlich die Übersichtlichkeit einer Aktivität gesteigert werden.

Ein Pfad wird als ein Pfeil mit doppelline dargestellt, an dem, wie auch bei Links, ein Name und die Leserichtung angegeben wird.

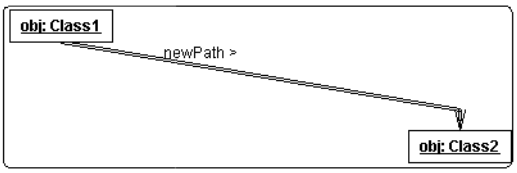


Abbildung 32: Darstellung eines Paths

Das folgenden Diagramm ist ein einfaches Beispiel für ein FUJABA Storydiagramm. In dem Beispiel wird auf einem Objekt vom Typ Example die Methode „sum“ mit zwei Integern als Parameter übergeben. Diese Methode erstellt ein neues Objekt „result“ vom Typ „Summe“ und

setzt „value“ dieses Objekts auf die Summe der beiden Integer. Daraufhin wird das Objekt „result“ als Ergebnis der Methode zurückgegeben.

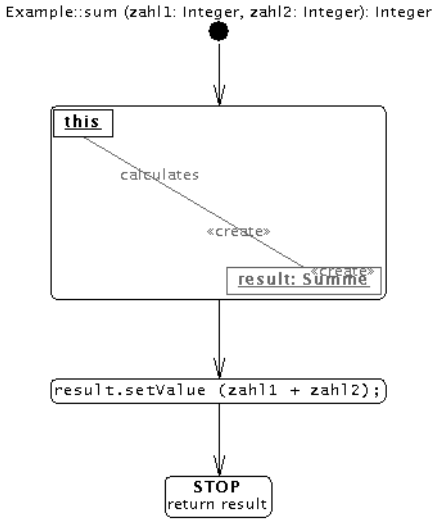


Abbildung 33: Ein einfaches Beispiel für ein Storydiagramm

5.2 Guided Tour

Nun soll am bereits bekannten Roulette Beispiel demonstriert werden, wie ein Storydiagramm in FUJABA erstellt werden kann. Im Speziellen wollen wir das Verhalten der Methode „einFeldWeiter“ aus der Klasse „Kugel“ modellieren. Da dieses Beispiel auf dem Klassendiagramm, welches im letzten Kapitel erzeugt wurde, aufbaut, müssen die folgenden Schritte im selben Projekt ausgeführt werden.

Bevor mit der eigentlichen Modellierung begonnen werden kann, muß auch hier zunächst ein leeres Storydiagramm erzeugt werden. Dies geschieht über den Menüpunkt »*Diagrams*« - »*New Activity Diagram*«.

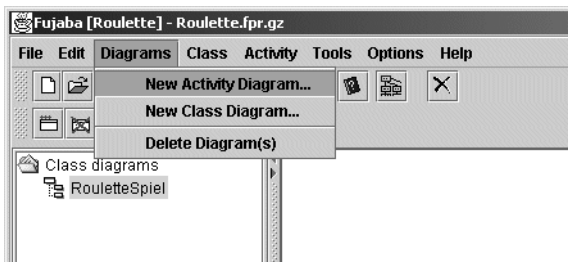


Abbildung 34: Erstellen eines neuen Storydiagramms

In dem daraufhin erscheinenden Dialog muß festgelegt werden, welche Methode durch das neue Diagramm beschrieben werden soll. Da wir das Verhalten der Methode „einFeldWeiter“ der Klasse „Kugel“ beschreiben wollen, muss zunächst die entsprechende Klasse ausgewählt

werden. Daraufhin erscheint eine Liste aller in dieser Klasse definierten Methoden, in der nun die Methode „einFeldWeiter“ selektiert wird.

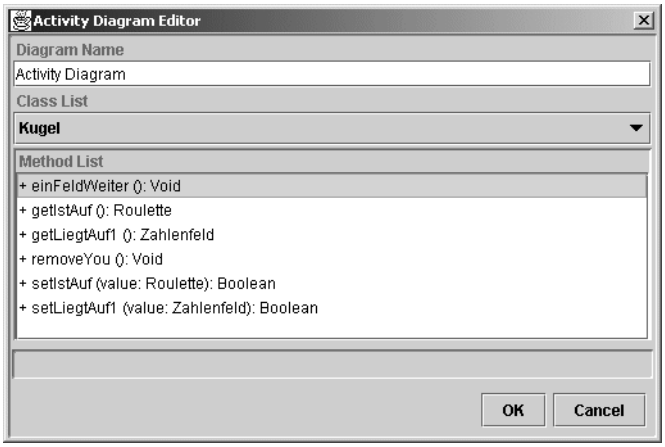


Abbildung 35: Der „New Activity Diagram“ Dialog

Nach der Bestätigung des Dialogs wurde im Projektbaum das soeben erstellte Diagramm angelegt und bereits ausgewählt.

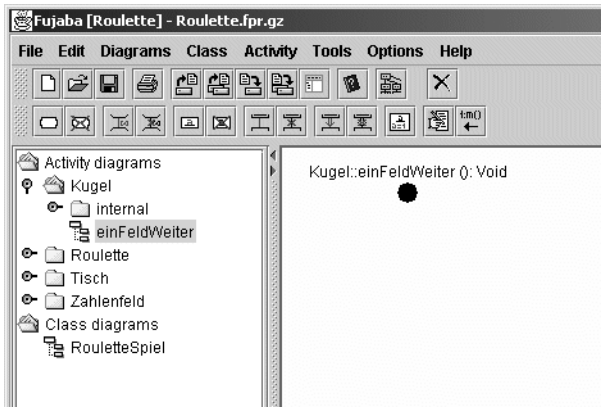


Abbildung 36: Die Fujaba Umgebung nach dem Erstellen eines Story Diagramms

Der Projektbaum wurde um einen zusätzlichen Ordner mit dem Titel »*Activity diagrams*« erweitert. Für jede Klasse, für die ein Aktivitätendiagramm erstellt wird, wird hier ein Unterordner mit dem Namen der Klasse angelegt. Die Diagramme zu den einzelnen Methoden sind in dem Ordner der entsprechenden Klasse zu finden. Die Diagramme tragen den Namen der Methode, die sie beschreiben.

Es kann passieren, dass FUJABA nach dem erstellen eines Aktivitätendiagramms bereits die Ordner für sämtliche Klassen und angelegt hat und in diesen bereits leere Diagramme für die einzelnen Methoden. In diesem Fall, brauchen diese Diagramme nicht mehr erstellt zu werden. Es können die von FUJABA erstellten Diagramme verwendet werden. Der gleiche Effekt kann auch nach dem Abspeichern eines Projekts auftreten...

Außerdem ist zu erkennen, dass sich die spezifische Toolbar verändert hat. Sie enthält nun die wichtigsten Befehle für die Erstellung von Story Diagrammen. In der Diagrammansicht wird bereits das neue Diagramm, dessen Startknoten automatisch generiert wurde, angezeigt.

Nun kann mit der Erstellung des Storydiagramms begonnen werden. Zu aller erst muß eine neue Aktivität angelegt werden, in der das Verhalten der Methode beschrieben werden soll. Diese kann über den Menübefehl »*Activity*« - »*New / Edit Activity*« erstellt werden.

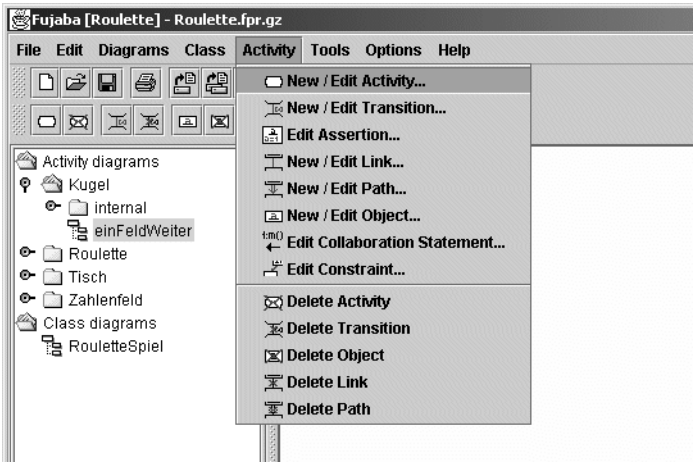


Abbildung 37: Erstellen einer neune Aktivität

Auch hier erscheint wieder ein Dialogfenster, in dem zunächst einige Eigenschaften der Aktivität festgelegt werden müssen. Auf der linken Seite des Dialogs muß die Art der Aktivität gewählt werden. Je nach Auswahl ändert sich der rechte Teil des Dialogs. In diesem Beispiel soll eine Story Aktivität erstellt werden. Hierzu wird also der Aktivitätstyp »*Story*« ausgewählt, und die Aktivität zum Beispiel mit „einFeldWeiter“ benannt.

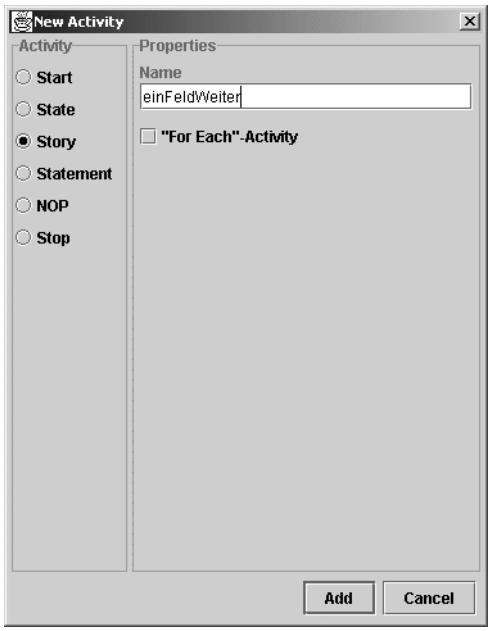


Abbildung 38: Der „New Activity“ Dialog bei Auswahl von „Story“

Durch ein »*Add*« wird die zunächst leere Aktivität erzeugt, die in der Diagrammansicht als ein kleines Rechteck erscheint.

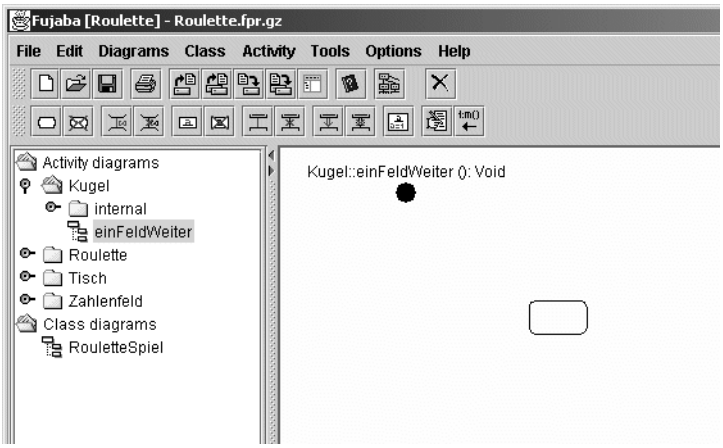


Abbildung 39: Eine neue Aktivität in Fujaba

Innerhalb dieser Story-Aktivität kann nun das Verhalten des Programms zur Laufzeit aus Objektsicht beschrieben werden.

Die Methode „einFeldWeiter“ soll das Weiterspringen der Kugel von einem Zahlenfeld auf das nächste repräsentieren. Wie aus dem Klassendiagramm bereits ersichtlich ist, hat ein Kugelobjekt genau eine Referenz auf ein Zahlenfeld, nämlich auf das, auf dem die Kugel gerade liegt. In unserer Aktivität muss also modelliert werden, dass die Referenz auf das alte Feld gelöscht und auf das nächste Feld erstellt wird.

Bei der Modellierung soll von dem aktuellen Kugelobjekt ausgegangen werden, das als erstes in die neue Aktivität eintragen wird. Ein Objekt kann über einen Klick auf die rechte Maustaste auf die entsprechende Aktivität und Auswahl des Befehls »*New*« - »*Object*« erstellt werden.

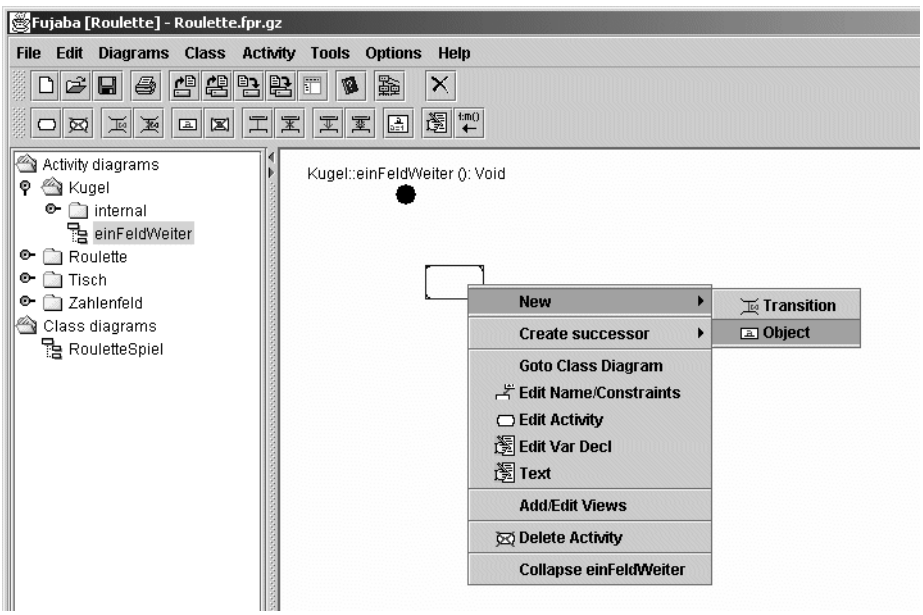


Abbildung 40: Anlegen eines neuen Objekts

Im Folgedialog muss das neue Objekt benannt und dessen Typ festgelegt werden.

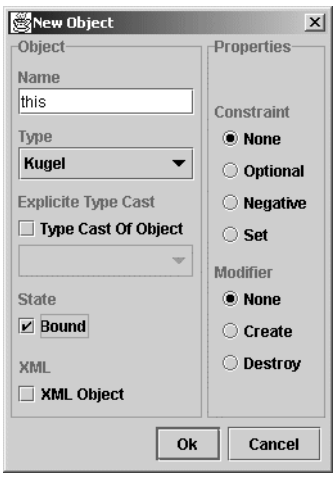


Abbildung 41: Der „Edit Object“ Dialog

Da wir uns auf das Objekt beziehen, auf dem die Methode aufgerufen wurde, benennen wir das Objekt „this“ und setzen den Typ selbstverständlich auf den Wert „Kugel“. Zusätzlich muss bei this-Objekten unter *State* die Option *Bound* angewählt werden.

Möchte man zum Beispiel einen Initialisierungsprozess darstellen, wobei ein Objekt neu erzeugt wird, so wäre der Modifier in diesem Dialog auf »*Create*« zu stellen.

Durch ein »*OK*« wird das neue Objekt der Aktivität zugewiesen. Objekte werden ebenfalls als Rechtecke dargestellt. Der Name des Objekts wird dabei innerhalb des Rechtecks angegeben.

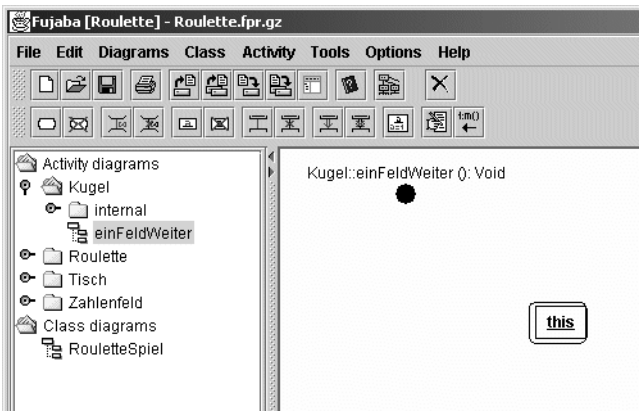


Abbildung 42: Objekt innerhalb einer Aktivität

Um nun darzustellen, dass eine Referenz auf ein altes Zahlenfeld Objekt gelöscht wird, muß dieses Objekt ebenfalls in die Aktivität eingetragen werden. Die geschieht analog zur Erstellung des „this“ Objekts. Hier wählen wir den Typ dieses Objekts als „Zahlenfeld“ und benennen es zum Beispiel mit „zAlt“.

Auch diese Objekt erscheint daraufhin innerhalb der Aktivität.

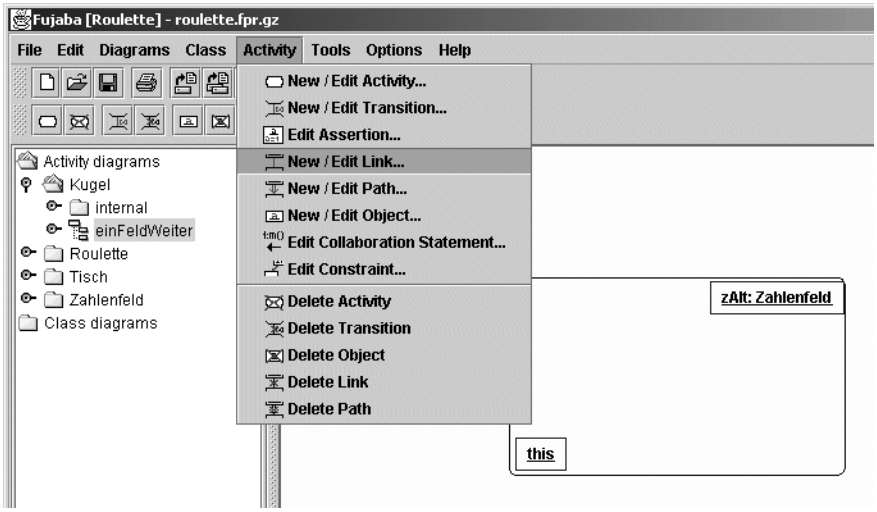


Abbildung 43: Erstellen eines Links zwischen zwei Objekten

Das Löschen der Assoziation zwischen den beiden Objekten kann durch einen so genannten Link zwischen diesen Objekten dargestellt werden. Das Erstellen eines Links ist vergleichbar mit dem Erstellen einer Assoziation in Klassendiagrammen. Zunächst werden beide Klassen, zwischen denen wir eine Verbindung erstellen werden soll, selektiert. Dann kann über den Menübefehl »*Activity*« - »*New / Edit Link...*« ein Link zwischen den beiden Objekten erstellt werden.

In dem folgenden Dialog muss der Link näher spezifiziert werden.

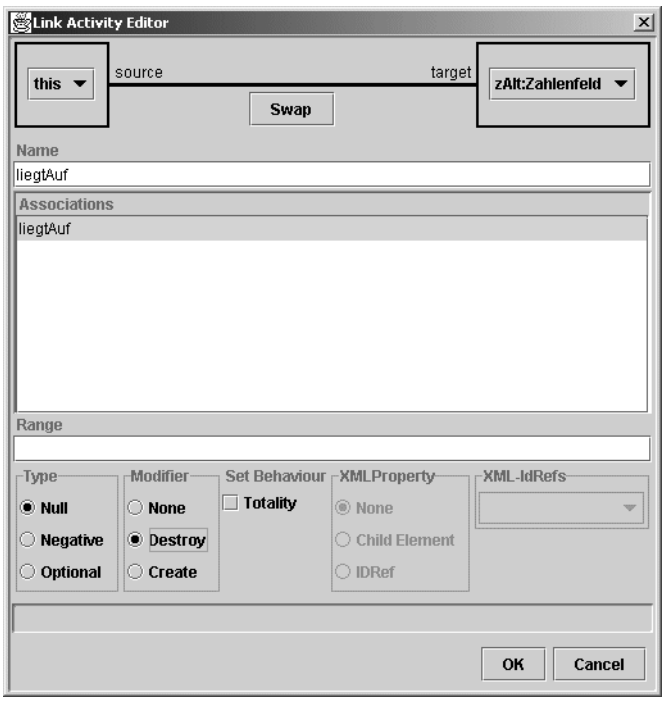


Abbildung 44: Der „Link Activity“ Dialog

Der obere Teil des Dialogs dient zur Auswahl der richtigen Objekte. Dies geschieht ebenfalls analog zur Auswahl von Klassen beim Erzeugen von Assoziationen in Klassendiagrammen. Auch hier sind durch unsere Selektion im letzten Schritt die richtigen Objekte ausgewählt.

Achtung: An dieser Stelle können nicht beliebige Objekte durch einen Link miteinander verbunden werden. Es können nur Links zwischen Objekten erstellt werden, die durch eine Assoziation im entsprechenden Klassendiagramm verbunden sind. Nach Auswahl eines Objekts zeigt der Dialog in der Auswahl für das zweite Objekt nur noch mögliche Partnerobjekte an. Sind zum Beispiel im Klassendiagramm keine Assoziationen definiert, so bleiben beide Auswahlfelder, ungeachtet der Selektion vor dem Aufruf des Dialogs, leer.

Neben den Objekten wird unter Associations bereits der Name dieser Assoziation, so wie er im Klassendiagramm spezifiziert wurde, angezeigt. Gibt es mehrere Assoziationen zwischen den beteiligten Klassen, so ist die Richtige aus der Liste auszuwählen.

Um ein Löschen der Referenz zu modellieren, wird also unter Modifier der Punkt *Destroy* selektiert. Eine Bestätigung über den »*OK*« Knopf bringt uns zurück zur Hauptansicht.

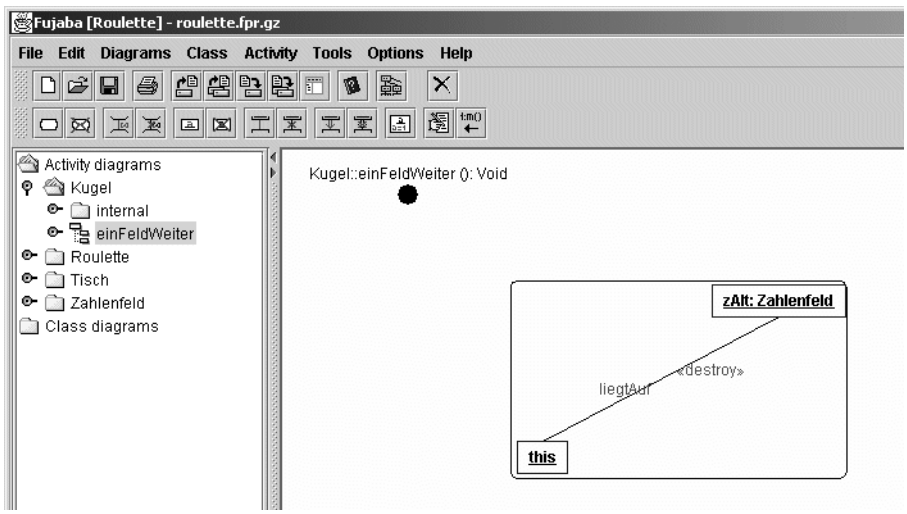


Abbildung 45: Ein „Destroy“ Link innerhalb einer Aktivität

Zwischen den Objekten ist nun ein Link eingetragen, der in roter Schrift zum einen mit dem Wort »*Destroy*« und zum anderen mit dem Namen der Assoziation versehen ist. Die rote Schrift soll dabei verdeutlichen, dass diese Referenz zwischen den beiden Objekten während der Ausführng der Aktivität gelöscht wird.

Analog wird nun mit dem neuen Zahlenfeld verfahren, auf dem die Kugel nun liegt. Es wird ein neues Objekt „zNeu“ vom Typ Zahlenfeld erzeugt. Auch hier wird ein Link zwischen Kugel und Zahlenfeld angelegt, wobei der Modifier des Links dieses mal auf *create* setzen, da die Referenz zwischen den Objekten neu erstellt wird. Dieser Link wird nun ebenfalls in der Diagrammansicht dargestellt. Die Beschriftung von *create*-Links wird in grün dargestellt.

In einem der folgenden Schritte wird zusätzlich noch eine Methode der Klasse Roulette benötigt, welche die Gesamtanzahl der Zahlenfelder liefert. Deswegen muss ebenfalls ein Roulette

Objekt in das Diagramm aufgenommen werden. Das Erzeugen des „Roulette“ Objekts und der dazu notwendigen Links geschieht analog zu den letzten Schritten.

Wurde das „Roulette“ Objekt und dessen Links angelegt, so ist die Struktur der Aktivität fertig. Die Aktivität sollte nun wie folgt aussehen.

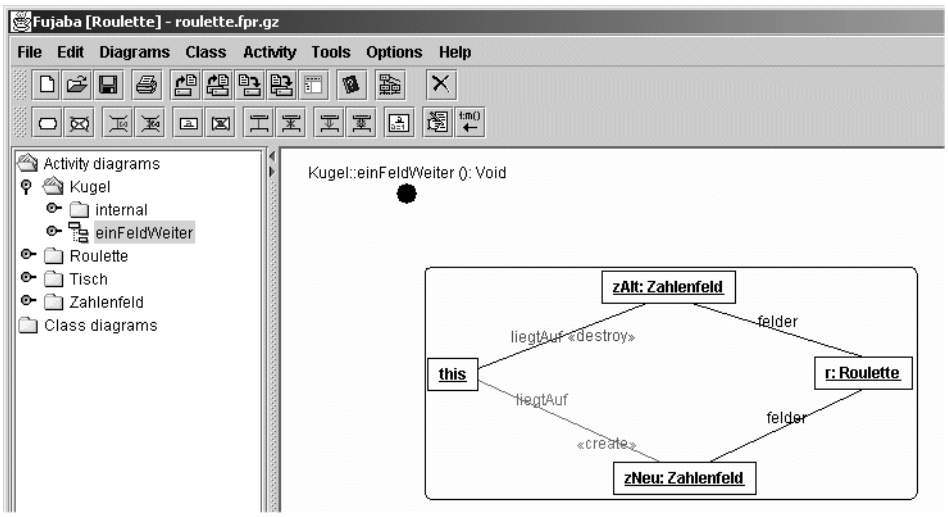


Abbildung 46: Die fertige Story Aktivität

Allerdings beschreibt die bisher erstellte Aktivität noch nicht, zu welchem speziellen Zahlenfeld eine Assoziation aufgebaut werden soll. Dafür müssen wir eine so genannte *Assertion* oder Bedingung zu diesem Objekt angeben, um das richtige „Zahlenfeld“ Objekt zu referenzieren.

Durch einen Rechtsklick auf das „zNeu“ Objekt kann nun im Popupmenü über den Befehl »*Edit Assertions...*« diese Bedingungen festgelegt werden.

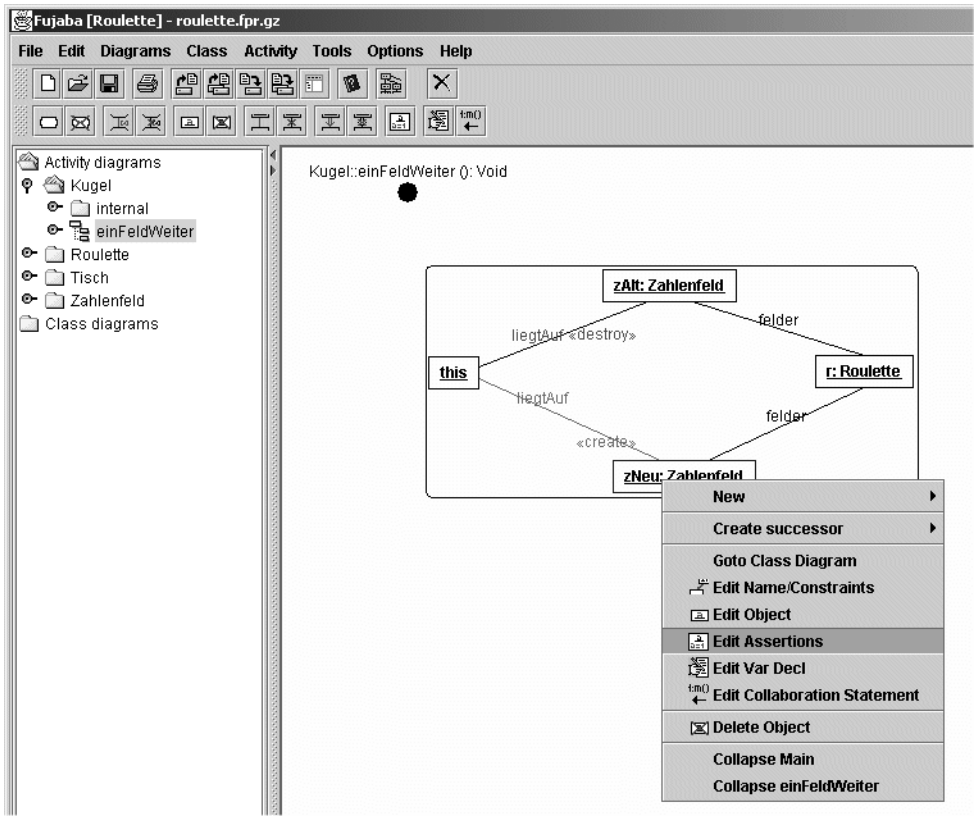


Abbildung 47: Festlegen von Objektbedingungen

In dem Dialog werden in dem *Source* Feld alle Attribute des gewählten Objekts aufgelistet, für die mit Hilfe der oben angegebenen Operatoren Bedingungen formuliert werden können.

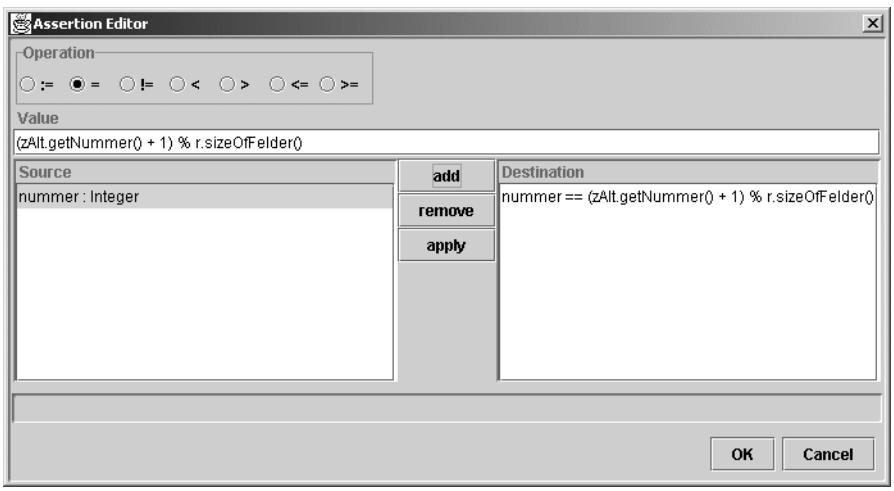


Abbildung 48: Der Assertion Dialog

Wir wollen davon ausgehen, dass die Zahlenfelder sortiert auf dem Roulette befinden, oder die Nummer eine interene Verwaltungsnummer der Zahlenfelder ist. Die Kugel springt also immer

von einem Feld n auf ein Feld n+1. Lediglich beim letzten Zahlenfeld, springt sie wieder auf das erste zurück. Dies Bedingung kann in JAVA formuliert so aussehen:

```
nummer == (zAlt.getNummer () + 1) % r.sizeOfFelder ()
```

In dem Dialog muss nun zunächst das Attribut und der Operator ausgewählt werden. Die rechte Seite der Bedingung wird nun in dem *Value* Feld eingetragen. Durch »*Add*« wird die neue Bedingung in die *Destination* Liste aufgenommen.

Durch ein »*OK*« wird die Änderung auch am Diagramm sichtbar.

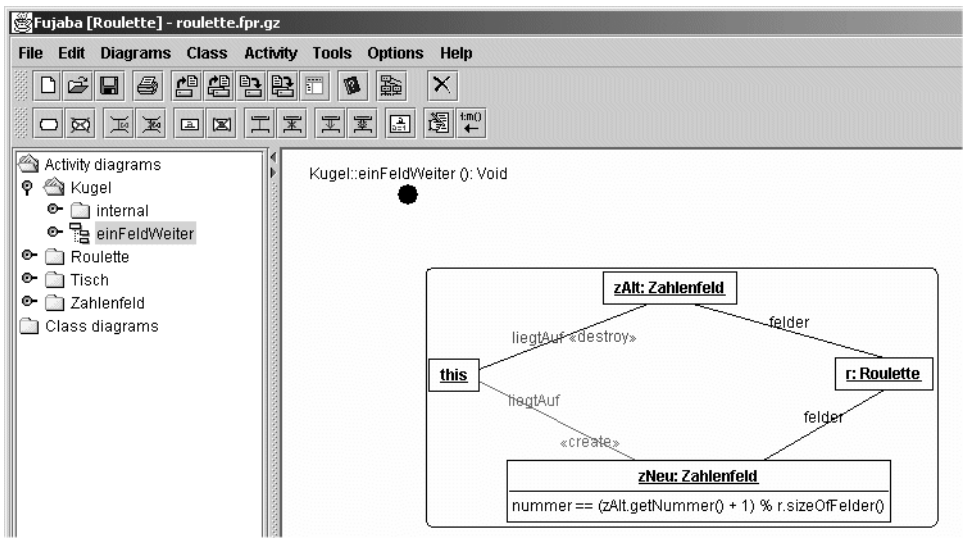


Abbildung 49: Objekt mit Bedingung

Die Erstellung der Aktivität ist in diesem Beispiel hiermit beendet. Nun muß diese Aktivität aber noch in den Zeitlichen Ablauf beim Methodenaufruf der Methode „einFeldWeiter“ eingebracht werden.

Da diese Aktivität die Einzige in dieser Methode ist, ist die Modellierung hierfür sehr einfach. Wir müssen sie lediglich zwischen Start- und Endknoten einreihen.

Der Startknoten ist bereits vorhanden. Um nun den Programmablauf darzustellen muss eine Transition zwischen Startknoten und Aktivität eingeführt werden. Hierfür werden die zu verbin-

deneden Elemente mit Hilfe der STRG Taste selektiert und in der Menüleiste der Befehl »*Activity*« - »*New / Edit Transition*« ausgeführt.

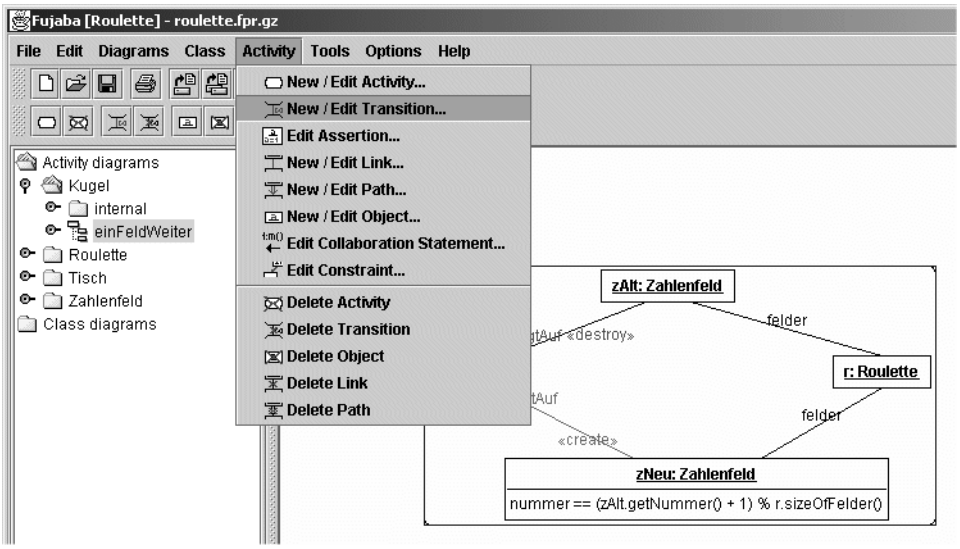


Abbildung 50: Erstellen einer Transition

Auch hier erscheint wiederum ein kleiner Dialog.

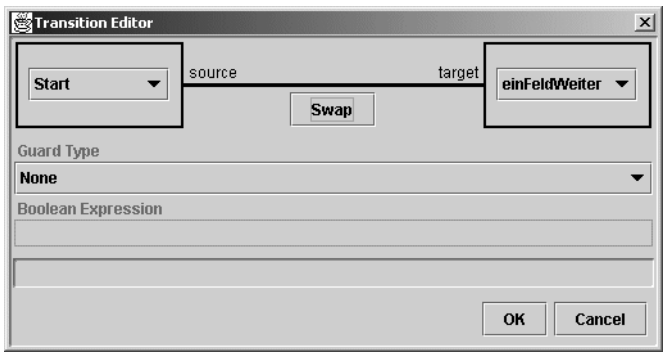


Abbildung 51: Der Tranistions Dialog

In diesem Dialog können gegebenenfalls die beiden ausgewählten Objekte noch einmal verändert werden. Unter »*Guard Type*« können Bedingungen angegeben werden, die gelten müssen, um diesen Transition zu benutzen. Hierdurch können zum Beispiel if-Abfragen modelliert werden, was in diesem Beispiel jedoch nicht notwendig ist.

Nach einem »*OK*« erscheint die Transition als Pfeil vom Startknoten zu unserer Aktivität. Angegeben Bedingungen würden umgeben von eckigen Klammern an die Transition geschrieben.

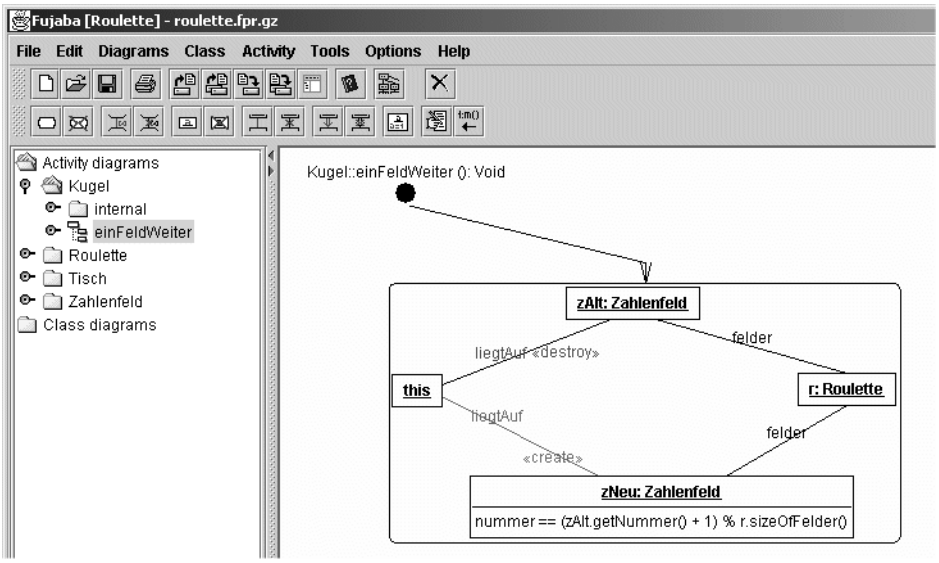


Abbildung 52: Eine Transition von Startknoten zur Aktivität

Nun muss noch der Endknoten der Methode erstellt werden, was ebenfalls über den »*New / Edit Activity*« Dialog geschieht.

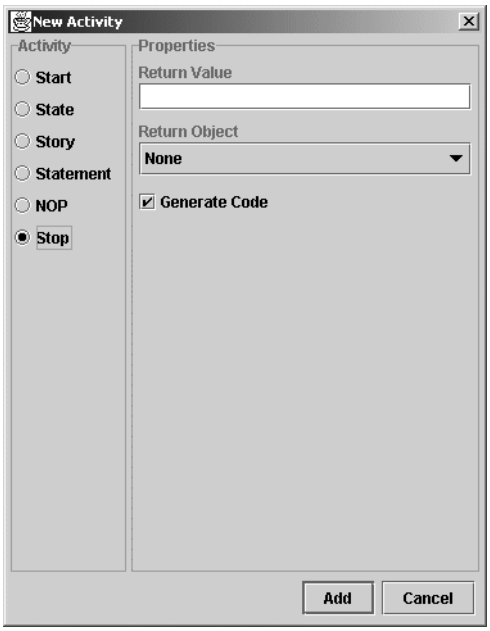


Abbildung 53: Erstellen einer Stop-Aktivität

Als Aktivitätstyp wählen wir nun den Typ „Stop“. In dem *Properties* Rahmen erscheinen nun Steuerelemente, mit denen der Stop-Activität ein Rückgabewert zugewiesen werden kann. Da unsere Methode „einFeldWeiter“ jedoch keinen Rückgabewert besitzt, kann die Einstellung auf *None* belassen werden.

Nach dem Erzeugen dieser letzten Aktivität muss diese nur noch mit unserer Story-Aktivität durch eine Transition verbunden werden und die Erstellung unseres Story Diagramms ist beendet.

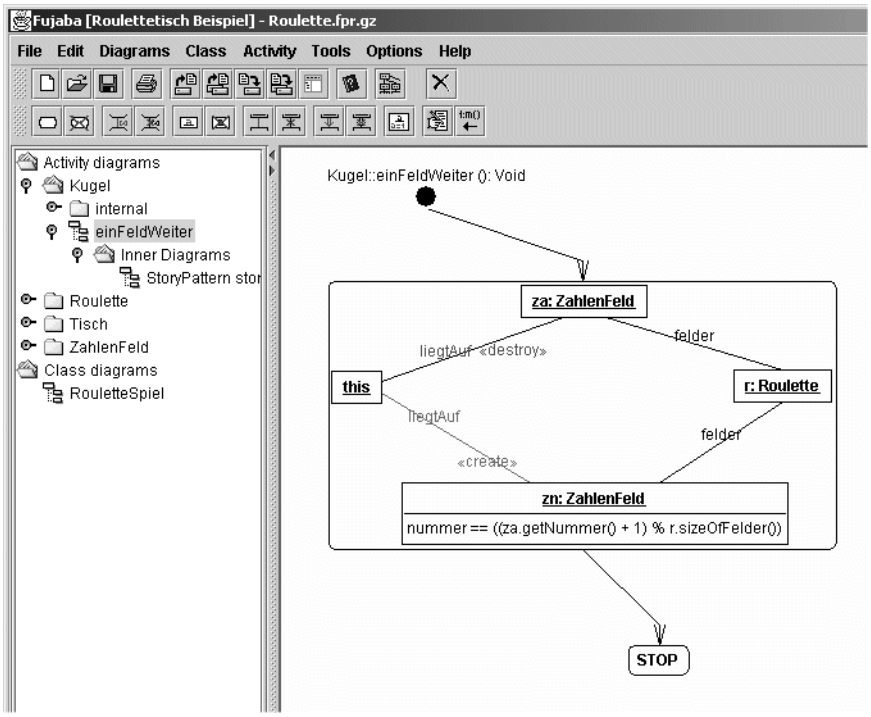


Abbildung 54: Das fertige Story Diagramm

Im Projektbaum kann man noch erkennen, daß unser Diagramm zur Methode „einFeldWeiter“ einen Unterordner mit dem Namen *Inner Diagrams* besitzt. In diesem Ordner finden wir jede Story-Aktivität der Methode noch einmal als eigenes Diagramm.

KAPITEL 6

Codegenerierung

6.1 Guided Tour

6.2 Generelle Konzepte

Assoziationen, Attribute, Storydiagramme(?)

KAPITEL 7

Dynamic Object Browsing System

In den vorangegangenen Kapiteln ist die Erstellung eines UML-Modells in FUJABA und die Übersetzung dieses Modells in ein ausführbares Programm erläutert worden. Nachdem das Modell erstellt worden ist, muß es in einem nächsten Schritt auf korrekte Funktion getestet werden, um Fehler im Modell zu finden und zu beseitigen. Das Testen geschieht auf Basis des ausführbaren Java-Programms.

Üblicherweise wird hierzu eine Testsoftware, ein sogenannter *Debugger*, eingesetzt, die auf *Quelltext-Ebene* ein schrittweise kontrolliertes Ausführen des Programms und die Überwachung der Laufzeit-Umgebung der Virtual Machine ermöglicht. Um mit einem solchen Debugger arbeiten zu können, ist die genaue Kenntnis des Java-Quelltextes und des Zusammenhangs zwischen Modell-Elementen und Quelltext-Passagen erforderlich. Die Vorteile der anschaulichen grafischen Darstellung und der Konzentration auf Modellierungsaspekte statt Implementierungsdetails, der durch die Modellierung in UML gewonnen worden ist, gehen durch eine solche herkömmliche Testsoftware verloren.

Um auch in der Testphase die Vorteile der UML-Darstellung zu erhalten, stellt Fujaba das *Dynamic Object Browsing System* - kurz DOBS - zur Verfügung. DOBS ist eine Testumgebung, in der die Objektwelt eines laufenden Programms grafisch in UML-Notation dargestellt wird und die dem Tester die kontrollierte Simulation des Programmablaufes durch Interaktion mit dieser Objektwelt ermöglicht.

7.1 Überblick

DOBS visualisiert die Objektwelt innerhalb der *Java Virtual Machine* in Form eines *Objektdiagramms*, das Nachbarschaftsbeziehungen zwischen den Objekten darstellt und die aktuellen Attributwerte der Objekte zeigt. Zur Manipulation dieser Objektwelt stellt DOBS eine Liste der verfügbaren Methoden der Objekte zur Verfügung, die der Benutzer aufrufen kann. Zudem ist es natürlich möglich, neue Objekte zu instanziieren und existierende Objekte zu „löschen“ (siehe unten, Abschnitt 7.1.3).

Bevor weiter auf die Benutzung von DOBS eingegangen werden kann, sollen zunächst die von DOBS zur Visualisierung genutzten Objektdiagramme erläutert werden.

7.1.1 Objekt-Diagramme

Objektdiagramme stellen konkrete Objekte und deren Beziehungen zueinander dar. Sie sind in ähnlicher Form bereits im Rahmen von Storydiagrammen in Kapitel 5 vorgestellt worden. Abbildung 55 zeigt ein Beispiel eines Objektdiagramms.

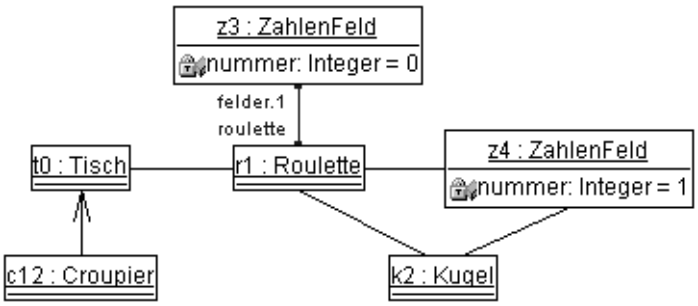


Abbildung 55: Objektdiagramm

Objekte werden durch Rechtecke dargestellt. Der *Typ* (also der Name der instanziierten Klasse) des jeweiligen Objektes sowie ein für DOBS *eindeutiger Bezeichner* sind zentriert im oberen Bereich dieser Rechtecke als unterstrichener Text zu finden. In diesem Beispiel sind also je ein Objekt vom Typ *Tisch*, *Croupier*, *Roulette* und *Kugel*, sowie zwei Objekte vom Typ *ZahlenFeld* dargestellt.

Im unteren Teil des Rechteckes sind die *Attribute* des Objektes mit *Sichtbarkeit* (siehe auch Kapitel 4), *Attributtyp* und *aktuellem Wert* aufgelistet. Klassenattribute werden unterstrichen dargestellt. Im Beispiel haben die Objekte vom Typ *ZahlenFeld* jeweils ein öffentliches Attribut vom Typ *Integer* mit Namen »nummer«.

Beziehungen zwischen den Objekten werden durch Kanten im Diagramm dargestellt. Einseitige *Referenzen* werden, ähnlich wie in Klassendiagrammen, durch gerichtete Kanten mit Pfeilspitze an einer Seite dargestellt. Im Beispiel hat das Objekt *c12* eine Referenz zu dem Objekt *t0*. Beidseitige *Assoziationen* werden analog durch ungerichtete Kanten dargestellt. Die *Rollennamen* unter denen ein Objekt jeweils referenziert wird, sind an den Kantenenden dargestellt. Standardmäßig werden Rollennamen in DOBS nur für selektierte Kanten dargestellt, wie im Beispiel an der Kante zwischen *t0* und *z3* zu sehen ist.

7.1.2 Grafische Benutzeroberfläche

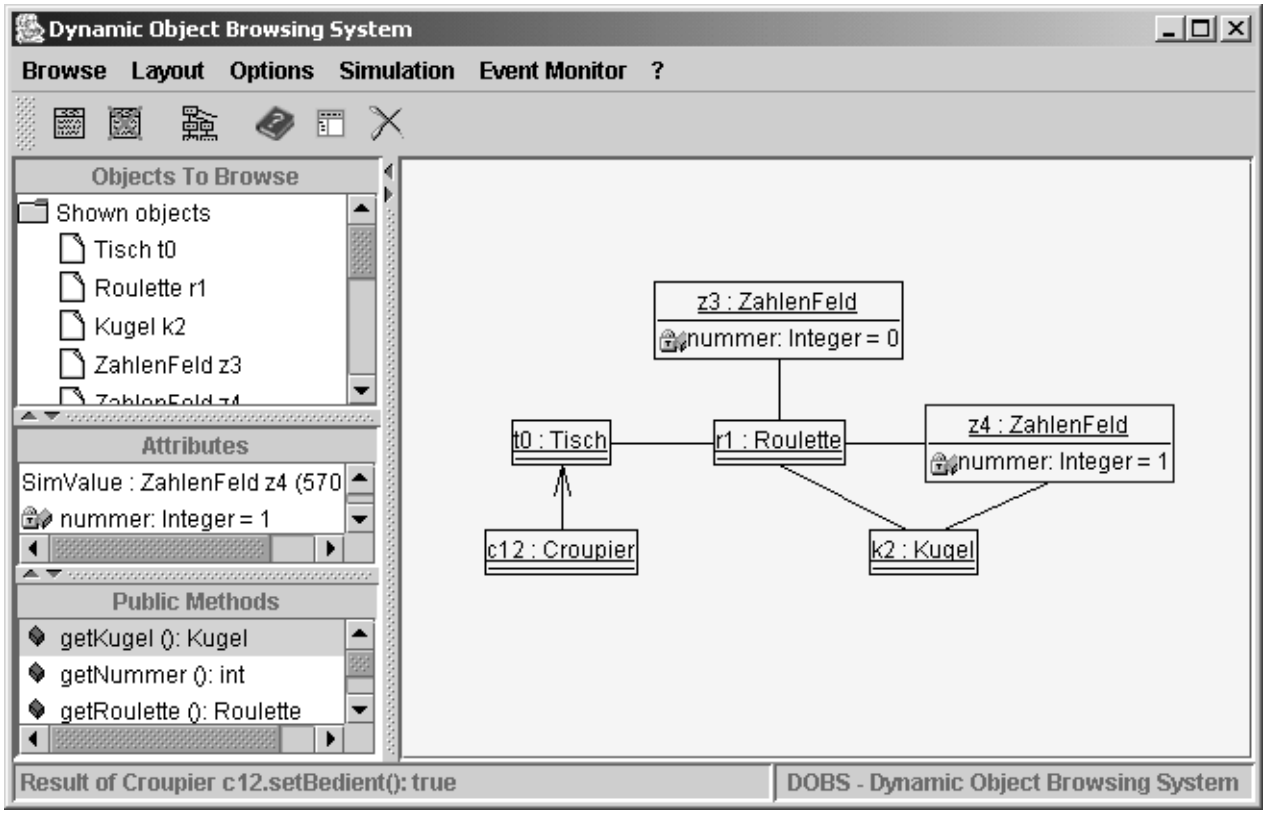


Abbildung 56: Grafische Benutzeroberfläche von DOBS

Die grafische Benutzeroberfläche von DOBS ist in sechs Bereiche unterteilt. Den größten Teil der Oberfläche nimmt der *Diagrammbereich* ein. Hier angezeigte Objekte können durch einen Mausklick selektiert und durch Klicken und Ziehen verschoben werden. Durch einen Doppelklick auf ein Objekt kann zudem zwischen einer kompakten und einer detaillierten Darstellung des Objektes gewechselt werden. Durch einen Rechtsklick auf den Diagrammhintergrund oder ein Objekt wird ein *Kontextmenü* geöffnet.

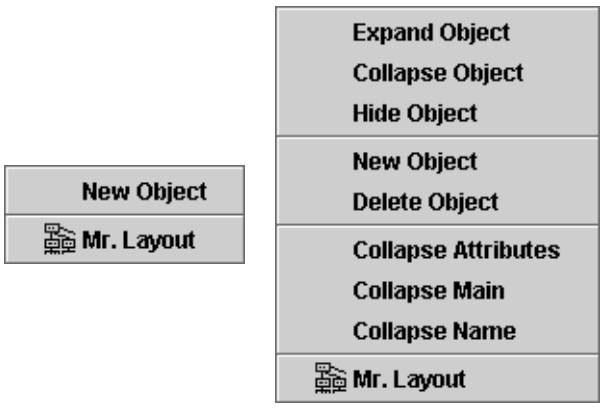


Abbildung 57: Objektdiagramm Kontextmenüs

Mit den Menüpunkten »*Expand Object*«, »*Collapse Object*« und »*Hide Object*« kann die Anzeige des Objektes und seiner Nachbarobjekte kontrolliert werden. Mit den Menüpunkten »*New Object*« und »*Delete Object*« können Objekte angelegt und gelöscht werden. Für detaillierte Informationen siehe Abschnitt 7.1.3 »Hinzufügen und Entfernen von Objekten« und Abschnitt 7.3 »Referenz der Menüeinträge«

Neben dem Diagrammbereich befinden sich im linken Teil der Oberfläche drei Bereiche mit Detailinformationen über die Objektwelt. Der oberste Bereich »*Objects To Browse*« enthält eine Auflistung aller DOBS zur Zeit bekannten Objekte, unterteilt in zur Zeit sichtbare und versteckte Objekte. Auch hier steht ein Kontextmenü mit Funktionen zum Anzeigen und Verstecken von Objekten zur Verfügung. Unter diesem Bereich befinden sich die Bereiche »*Attributes*« und »*Public Methods*«, in denen die Attribute und öffentlichen Methoden des zur Zeit im Diagrammbereich selektierten Objektes angezeigt werden. Die aufgelisteten Methoden können mit Hilfe des Kontextmenüs ausgeführt werden. Siehe hierzu Abschnitt 7.1.4 »Ausführen von Methoden«. Die Kontextmenüs für diese Bereiche sind in Abbildung 58 zu sehen.

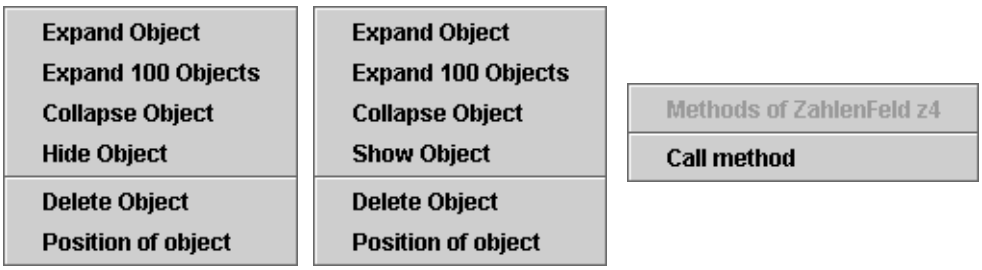


Abbildung 58: Kontextmenüs »Objects To Browse« und »Public Methods«

Im oberen Teil der Oberfläche sind eine *Menü*- und eine *Werkzeugleiste* angesiedelt. Eine Beschreibung der einzelnen Menüpunkte ist in Abschnitt 7.3 »Referenz der Menüeinträge« zu finden.

Am unteren Rand der Oberfläche ist schließlich noch eine Zeile mit aktuellen *Statusinformationen* über erzeugte Objekte und aufgerufene Methoden zu finden.

7.1.3 Hinzufügen und Entfernen von Objekten

In DOBS können Objekte auf unterschiedliche Weise zum Objektdiagramm hinzugefügt und wieder daraus entfernt werden.

Erzeugen von Objekten

Eine Möglichkeit zur Darstellung eines Objektes ist dessen Neuerzeugung. Hierzu steht der Menüpunkt »*Create New Object*« im »*Browse*«-Menü, der Menüpunkt »*New Object*« im Kontextmenü des Diagramms oder die Schaltfläche »*Create new Object*«  in der Werkzeugleiste zur Verfügung. Diese Befehle öffnen einen Dialog, in dem die zu instanziierende Klasse in einer Auswahlliste von verfügbaren Klassen ausgewählt werden kann. Ist die Klasse hier nicht auf-

gelistet, kann sie vom Benutzer durch Betätigen der Schaltfläche »*Other...*« in einem weiteren Dialog mit dem *vollreferenzierten Klassennamen* in der Java-üblichen Punkt-Notation (also etwa javax.swing.JComponent) explizit angegeben werden. Hat die angegebene Klasse mehr als einen Konstruktor oder benötigt der Konstruktor der Klasse zusätzliche Parameter, so öffnet sich ein weiterer Dialog, in dem zwischen den Konstruktoren ausgewählt werden kann und die Parameter angegeben werden können. Ist die Erstellung des Objektes erfolgreich, so ist es anschließend in dem Objektdiagramm zu sehen.

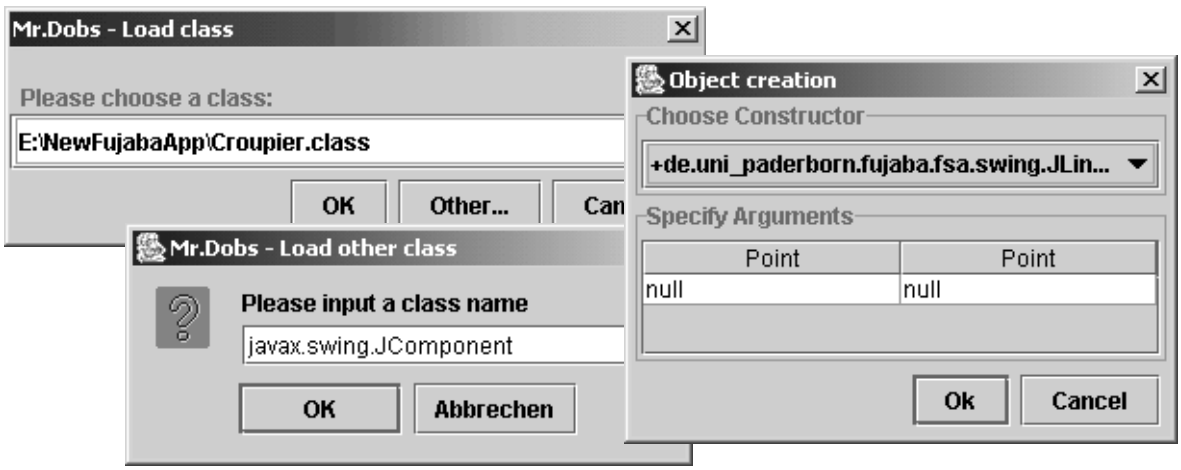


Abbildung 59: Instanziierung einer Klasse in DOBS

Anzeigen existierender Objekte

Objekte, die im Objektdiagramm dargestellt sind, können Assoziationen zu weiteren Objekten haben, die DOBS bislang nicht bekannt sind. Das Finden und Anzeigen dieser unbekannten Objekte ausgehend von einem bekannten Objekt wird in DOBS als *Expansion* bezeichnet. Dabei wird das zu expandierende Objekt nach Assoziationen zu weiteren Objekten durchsucht. Die gefundenen Objekte werden der Analyseumgebung von DOBS hinzugefügt und im Objektdiagramm angezeigt. Die Expansion kann über den Menüpunkt »*Expand Object*« im Kontextmenü des zu expandierenden Objektes entweder im *Objektdiagramm* oder im Bereich »*Objects To Browse*« ausgelöst werden. Um die Expansion größerer Objektmengen zu vereinfachen, steht zudem im Kontextmenü des Bereiches »*Objects To Browse*« der Menüpunkt »*Expand 100 Objects*« zur Verfügung, der eine wiederholte (rekursive) Expansion auf den neu gefundenen Objekten bis zu einer Tiefe von 100 auslöst.

Objekte, die in DOBS bereits bekannt, jedoch durch den Benutzer ausgeblendet worden sind (siehe unten), finden sich in der Liste »*Hidden Objects*« im Bereich »*Objects To Browse*«. Sie werden beim Expandieren eines Objektes nicht wieder sichtbar. Stattdessen können sie über das Kontextmenü dieser Liste wieder sichtbar gemacht werden. Hierzu dient der Menüpunkt »*Show Object*«.

Ausblenden von Objekten

Da die Menge der Objekte, die im Laufe der Testphase erzeugt und betrachtet werden, häufig sehr groß ist und das Objektdiagramm bei Anzeige aller Objekte sehr unübersichtlich wird, ermöglicht DOBS das *selektive Ausblenden* einzelner Objekte. Hierzu dienen die Menüpunkte »*Hide Object*« und »*Collapse Object*« im Kontextmenü der Objekte. Durch »*Hide Object*« wird das ausgewählte Objekt ausgeblendet. »*Collapse Object*« blendet alle Objekte aus, auf die das ausgewählte Objekt verweist. Ausgeblendete Objekte werden im Bereich »*Objects To Browse*« in der Liste »*Hidden Objects*« aufgelistet, die durch einen Doppelklick auf das Ordnersymbol geöffnet und geschlossen werden kann. Um ein ausgeblendetes Objekt wieder anzuzeigen, muß es in dieser Liste ausgewählt und über den Kontextmenü-Eintrag »*Show Object*« wieder sichtbar gemacht werden.

Löschen von Objekten

Um ein oder mehrere Objekte vollständig zu löschen, müssen diese zunächst selektiert werden. Anschließend kann im Kontextmenü eines der Objekte oder in der Werkzeugleiste der Punkt »*Delete Object*«  benutzt werden, um die selektierten Objekte aus DOBS zu löschen.

Achtung: *Java erlaubt kein echtes, manuelles Löschen von Objekten. Stattdessen werden Objekte, die nicht mehr über Referenzen erreichbar sind, automatisch durch die sogenannte Garbage Collection der JVM gelöscht. Daher werden durch die Löschfunktion von* DOBS *lediglich alle* DOBS *bekannten Referenzen auf das zu löschende Objekt entfernt. Da* DOBS *unter Umständen nicht alle Referenzen auf das zu löschende Objekt kennt, ist nicht sichergestellt, daß das Objekt tatsächlich durch die Garbage Collection entfernt wird. Es ist in diesem Fall sogar möglich, daß das Objekt zu einem späteren Zeitpunkt durch Expansion eines anderen Objektes wieder in* DOBS *erscheint.*

7.1.4 Ausführen von Methoden

Neben dem Erstellen und Löschen von Objekten ist die wichtigste Methode zur Manipulation der Objektwelt in DOBS der *Aufruf von Methoden* auf den Objekten.
Um auf einem Objekt eine Methode aufrufen zu können, muß dieses Objekt zunächst im Objektdiagramm selektiert werden. Für das selektierte Objekt sind dann die verfügbaren Methoden im Bereich »*Public Methods*« im linken Teil der Oberfläche aufgelistet. Die gewünschte Methode kann dann ausgewählt und über den Menüpunkt »*Call Method*« des Kontextmenüs ausgeführt werden. Besitzt die Methode Parameter, so wird ein Dialog geöffnet, in dem diese angegeben werden können. Parameter mit *primitiven Datentypen* wie Zahlen und Text können in Textfeldern eingetragen werden. Für *objektwertige* Parameter stehen Auswahllisten mit allen

bekannten kompatiblen Objekten zur Verfügung. Abbildung 60 zeigt ein Beispiel für einen solchen Parameterdialog.

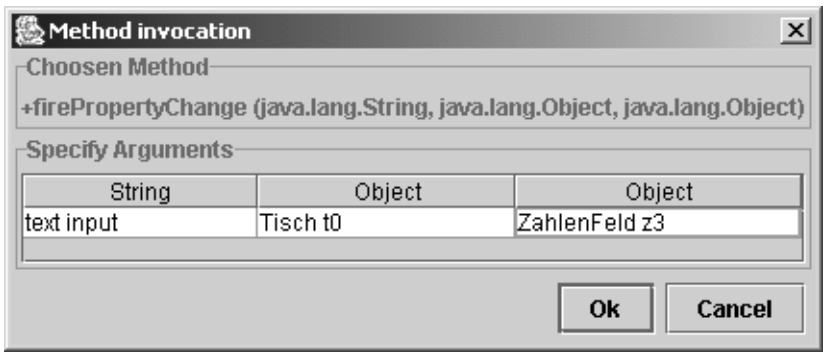


Abbildung 60: Parameterdialog

Achtung: *Bei der Angabe von primitiven Parametern in Textfeldern ist unbedingt zu beachten, daß die Eingabe mit der Return-Taste abzuschließen ist, da ansonsten die Eingabe nicht übernommen wird!*

Das *Ergebnis* des Methodenaufrufes ist im unteren Bereich von DOBS in der *Statuszeile* zu sehen. Hier ist zu sehen, ob die Methode erfolgreich ausgeführt werden konnte und welchen Rückgabewert sie geliefert hat.

Da ein Methodenaufruf auf einem Objekt gegebenenfalls massive Änderungen an der Objektwelt zur Folge haben kann, ist es möglich, daß DOBS nicht alle von einem Methodenaufruf gemachten Änderungen sofort registriert und das Objektdiagramm entsprechend aktualisiert. Aus diesem Grund ist es sinnvoll, nach der Ausführung *komplexerer Methoden* eine solche Aktualisierung manuell zu veranlassen. Dazu steht im Menü »*Browse*« der Menüpunkt »*Update Diagram*« und in der Werkzeugleiste die Schaltfläche »*Refresh Diagram*«  zur Verfügung, der die Darstellung aller bekannten Objekte und ihrer Beziehungen aktualisiert.

7.2 Guided Tour

Der erste Schritt zur Benutzung von DOBS ist natürlich das Starten des Programms. Hierzu gibt es zwei Alternativen. DOBS kann entweder allein gestartet werden (*stand-alone*), oder aus FUJABA heraus. Beim direkten Starten von DOBS ist darauf zu achten, daß alle Klassen, mit denen in DOBS gearbeitet werden soll, im *Klassenpfad* der JVM enthalten sein müssen.

Für diese Tour soll DOBS aus FUJABA heraus gestartet werden. Dazu sollte zunächst das Roulette-Projekt in FUJABA geladen sein. Ist dies geschehen, kann DOBS über den Menüpunkt »*DOBS...*« im »*Tools*«-Menü gestartet werden.

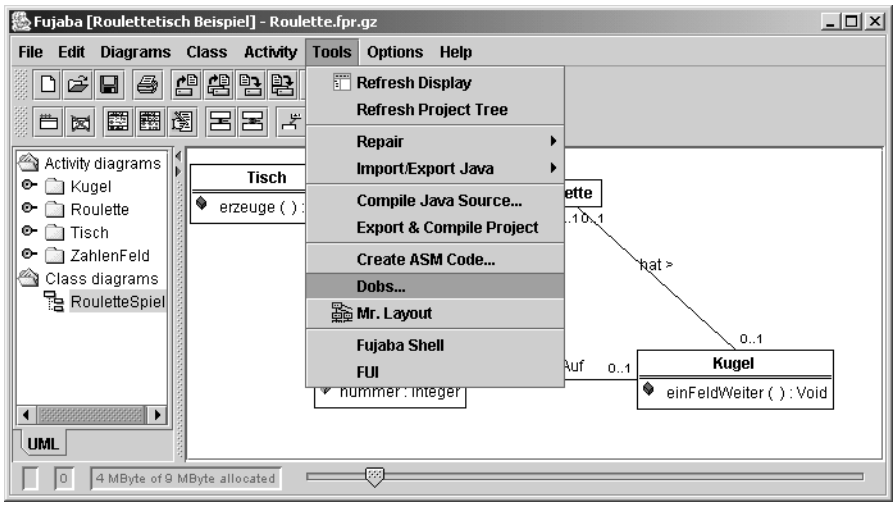


Abbildung 61: Starten von DOBS

Dabei werden automatisch die Klassen des aktuellen Modells in DOBS zur Verfügung gestellt. Hierzu wird das Modell zunächst automatisch als Java-Quelltext exportiert. Falls in dem Export-Verzeichnis bereits Java-Dateien existieren, wird der Benutzer in einem Dialog gefragt, ob diese gelöscht werden sollen, um Konflikte bei der Übersetzung zu vermeiden. Dieser Dialog sollte mit „Yes“ beantwortet werden.

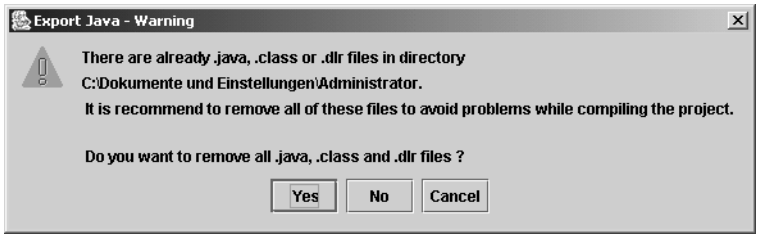


Abbildung 62: Exportkonflikt-Dialog

Nach erfolgreicher Exportierung werden die neuen Java-Dateien automatisch mit Hilfe des Java-Compilers in ausführbare Java-Klassendateien übersetzt. Zur Kontrolle öffnet sich ein »*Process Output Viewer*«-Fenster, das die Ausgabe des Compilers zeigt. Der Kompiliervorgang kann jederzeit über die Schaltfläche »*Abort*« abgebrochen werden. Ist die Übersetzung fehler-

frei abgeschlossen, wechselt die *Statusnachricht* im oberen linken Bereich des Fensters von »*running...*« auf »*finish*«. Das Fenster kann nun über die Schaltfläche »*Exit*« verlassen werden.

Abbildung 63: Process Output Viewer

Daraufhin wird DOBS gestartet. Hierbei öffnet sich ein weiterer *Process Output Viewer*, in dem Konsolen-Meldungen von DOBS angezeigt werden.

Erzeugen von Objekten

Beim Starten von DOBS wird ein Dialog zum Erzeugen eines initialen Objektes angezeigt.

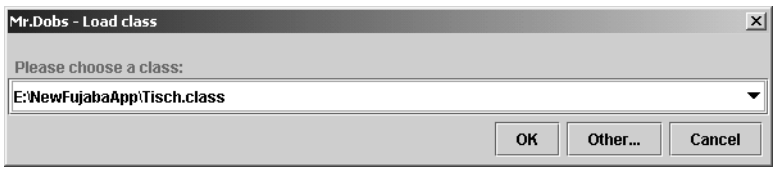


Abbildung 64: Load Class

Da in dem Roulette-Beispiel die Klasse *Tisch* für das Anlegen der Objektstrukturen verantwortlich ist, soll hier ein *Tisch*-Objekt erzeugt werden. Dazu wird in der Auswahlliste der Eintrag „*Tisch.class*“ ausgewählt und die Auswahl mit »*Ok*« bestätigt. Das neue Objekt erhält den eindeutigen Namen *t0* und wird im Objektdiagramm von DOBS angezeigt. Zusätzlich wird es in dem Bereich »*Objects to Browse*« im linken Teil der Oberfläche in einer Liste in dem Ordner »*Shown Objects*« aufgeführt. Dieser Ordner ist initial geschlossen und kann durch einen Doppelklick geöffnet werden. Nach dem Öffnen des Ordners sollte sich eine Situation wie in Abbildung 65 ergeben.

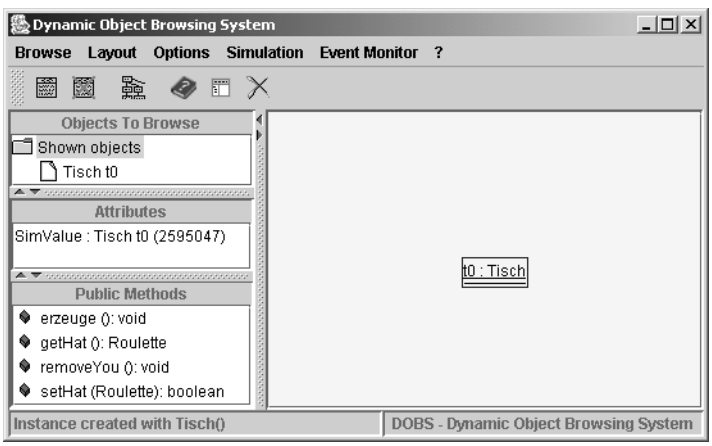


Abbildung 65: DOBS *mit Tisch-Objekt*

Um weitere Objekte anzulegen, kann jederzeit in der *Werkzeugleiste* im oberen Teil der Oberfläche die Schaltfläche »*Create new Object*«  betätigt oder im *Kontextmenü* des Objektdiagramms der Menüpunkt »*New Object*« angewählt werden.

Anzeigen und Ausblenden von Objekten

Die weiteren Objekte des Spiels können nun mit Hilfe der Methode *erzeuge()* der *Tisch*-Instanz *t0* angelegt werden. Dazu muß *t0* zunächst durch einen Mausklick im Diagramm selektiert werden, so daß das Objekt blau umrandet dargestellt ist. Nun sind in dem Bereich »*Public Methods*« im linken, unteren Teil der Benutzeroberfläche alle verfügbaren Methoden der Klasse *Tisch* aufgelistet. Durch einen Rechtsklick auf die Methode *erzeuge()* wird ein Kontextmenü geöffnet, in dem mit dem Menüpunkt »*Call Method*« die Methode aufgerufen wird, wie in Abbildung 66 dargestellt.

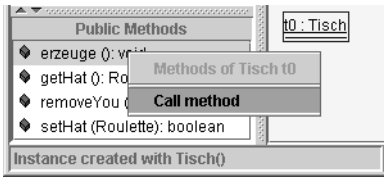


Abbildung 66: Methodenaufruf von erzeuge()

In der Statuszeile im unteren Teil der Oberfläche sollte anschließend der Text "*Result of Tisch t0.erzeuge(): null or void*" erscheinen, der bestätigt, daß die Methode erfolgreich ausgeführt worden ist und - da die Methode den Rückgabetyp *void* hat - keinen Rückgabewert zurückgeliefert hat.

Das Ausführen der Methode *erzeuge()* sollte eigentlich eine Reihe weiterer Objekte angelegt haben. Diese sind aber bisher nicht in DOBS zu sehen. Um sie sichtbar zu machen, muß das *Tisch*-Objekt *t0* expandiert werden. Dabei wird es auf Referenzen auf bisher nicht in DOBS enthaltenen Objekten durchsucht, die dann der Analyseumgebung hinzugefügt werden. Die Expansion von *t0* erfolgt über den Menüeintrag »*Expand Object*« im Kontextmenü von *t0* im Objektdiagramm oder in der Objektliste links. Anschließend sollte ein Objekt vom Typ *Roulette*

mit Namen *r0* in der oberen linken Diagrammecke angezeigt werden, das eine Assoziation zu *t0* hat, so daß sich die in Abbildung 67 dargestellte Situation ergibt

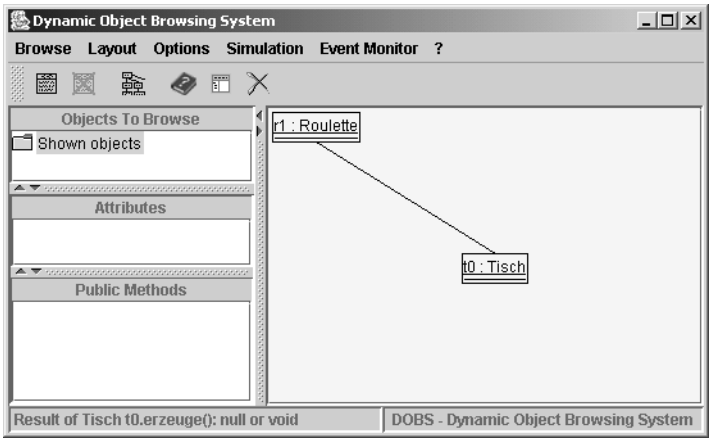


Abbildung 67: Expansion von Tisch t0

Da die übrigen von *erzeuge()* angelegten Objekte keine direkte Assoziation zu *t0* haben, sind sie jetzt noch nicht sichtbar. Sie werden analog durch Expansion des *Roulette*-Objektes der Analyseumgebung hinzugefügt.
Nach der Expansion befinden sich alle neu angezeigten Objekte in der linken oberen Diagrammecke. Eine übersichtlichere Anordnung der Objekte kann manuell durch Klicken und Ziehen mit der Maus erfolgen. Alternativ steht eine (rudimentäre) automatische Layoutfunktion zur Verfügung, die über den Menüeintrag »*Mr. Layout*« im Kontextmenü und im »*Layout*«-Menü, sowie über die entsprechende Schaltfläche  in der Werkzeugleiste gestartet werden kann. Nach der Neuanordnung der Objekte sollte sich die Situation entsprechend Abbildung 68 darstellen

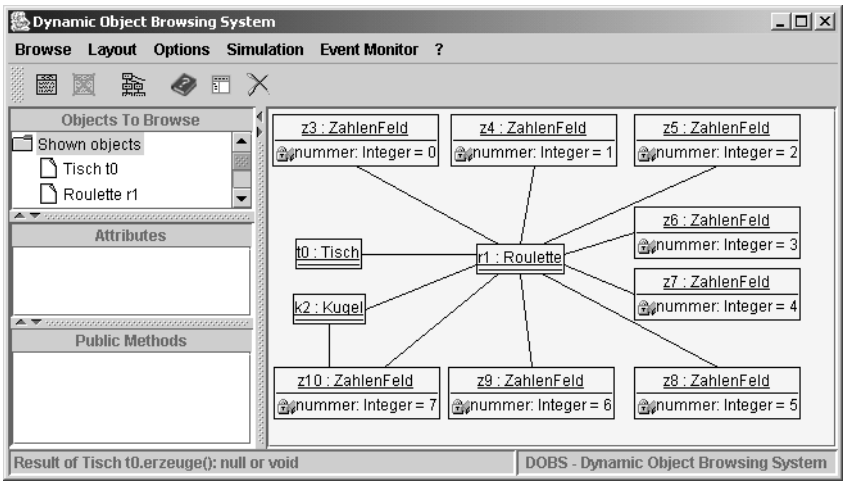


Abbildung 68: Alle Objekte in DOBS

Um im weiteren Verlauf mit einem besser überschaubaren Diagramm arbeiten zu können, sollen nun die soeben angezeigten Objekte vom Typ *ZahlenFeld* mit den Nummern 1 bis 6 (die Bezeichnungen sind üblicherweise z4...z9) wieder ausgeblendet werden, da sie für die nächsten

Schritte nicht relevant sind. Das Ausblenden geschieht über den Menüpunkt »*Hide Object*« im Kontextmenü des auszublendenden Objektes. Die ausgeblendeten Objekte bleiben in der Objektwelt unverändert bestehen, sie werden lediglich aus der Diagramm-Darstellung herausgenommen. Die ausgeblendeten Objekte sind als Liste im Ordner »*Hidden Objects*« im linken Teil der Oberfläche aufgelistet. Von hier können sie bei Bedarf über den Kontextmenü-Eintrag »*Show Object*« wieder sichtbar gemacht werden. Das Objektdiagramm nach dem Ausblenden der Objekte ist in Abbildung 69 dargestellt.

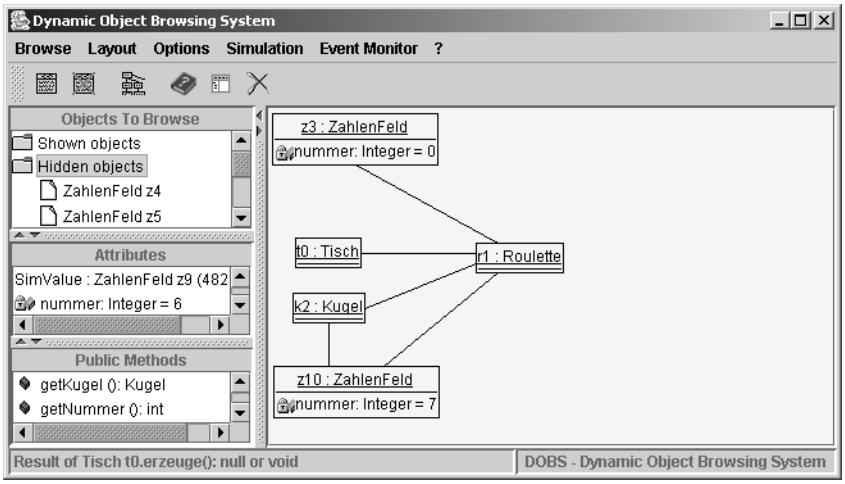


Abbildung 69: Objektdiagramm mit ausgeblendeten Objekten

Methodenaufrufe

Zum Abschluß der Guided Tour soll noch einmal genauer auf den Aufruf von Methoden eingegangen werden. Ein einfacher Methodenaufruf ist ja schon mit der Methode *erzeuge()* zu Beginn der Tour durchgeführt worden. Nun sollen weitere Methodenaufrufe mit *Parameterangabe* und *Rückgabewerten* behandelt werden. Zu diesem Zweck wird zunächst ein neues Zahlenfeld angelegt. Hierzu wird der Menüeintrag »*New Object*« im Kontextmenü aufgerufen und in dem bereits bekannten Dialog die Klasse *ZahlenFeld* ausgewählt.

Das frisch erzeugte ZahlenFeld benötigt nun zunächst eine passende Nummer. Hierzu wird das Objekt selektiert und die Methode "*setNummer (int):void*" aufgerufen. Da diese Methode einen Zahlenwert als *Parameter* benötigt, öffnet sich nun ein Dialog zur Eingabe des Parameterwertes. Das Textfeld für den int-Wert wird durch einen Doppelklick aktiviert. Anschließend kann der gewünschte Zahlenwert eingetragen werden. Das ZahlenFeld soll die Nummer 8 erhalten. Die Eingabe muß *unbedingt mit der Return-Taste abgeschlossen werden.* Anschließend kann

der Dialog über die Schaltfläche »*Ok*« verlassen werden. Abbildung 70 zeigt noch einmal den korrekt ausgefüllten Dialog.

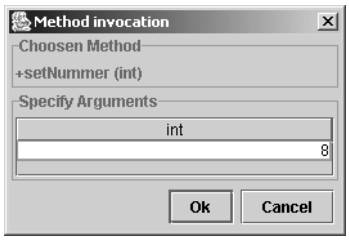


Abbildung 70: Dialog zur Eingabe von Methodenparametern

Das Attribut "*Nummer*" des ZahlenFeldes sollte nun den Wert 8 besitzen. Als nächstes muß das neue Zahlenfeld dem Roulette hinzugefügt werden. Hierzu wird die Methode "*setRoulette (Roulette):boolean*" des ZahlenFeldes aufgerufen. Wieder öffnet sich ein Parameter-Dialog für den Parameter vom Typ *Roulette*. Da es sich bei dem Typ dieses Parameters nicht um einen primitiven Datentyp (Zahl, Wahrheitswert oder Text) sondern um eine Klasse handelt, steht statt eines Textfeldes eine Auswahlliste mit allen bekannten kompatiblen Objekten zur Verfügung. In diesem Fall enthält die Liste nur den null-Wert und eine Instanz der Klasse *Roulette*. Nach Auswahl des *Roulette*-Objektes kann der Dialog wieder über »*Ok*« verlassen werden. Das erfolgreiche Anlegen der Assoziation zu dem *Roulette*-Objekt wird von der Methode durch den *booleschen Rückgabewert* `true` signalisiert, der in der Statuszeile am unteren Rand des Fensters ausgegeben wird. Außerdem ist im Objektdiagramm nun die neue Assoziation zu dem *Roulette*-Objekt zu sehen, wie in Abbildung 71dargestellt.

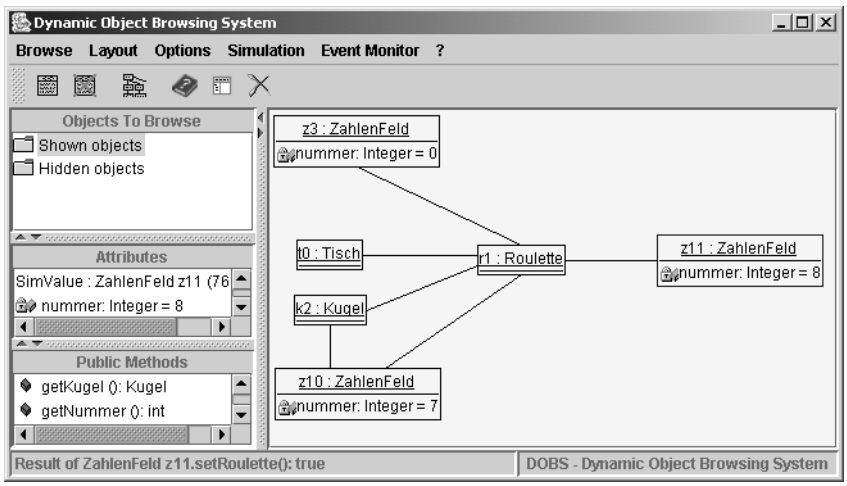


Abbildung 71: zusätzliches ZahlenFeld-Objekt

Auf dem so komplettierten Roulette-Spiel kann man nun die Kugel "rollen lassen". Dazu muß lediglich auf dem *Kugel*-Objekt die Methode "*einFeldWeiter (): void*" ausgeführt werden. Die Assoziation zwischen *Kugel* und *ZahlenFeld*, die initial auf das Feld mit der Nummer 7 weist, wandert dabei zu dem jeweils nächsten *Zahlenfeld* weiter. Nach dem ersten Aufruf also auf das neue *ZahlenFeld* mit der Nummer 8, danach auf das Feld mit der Nummer 0. Wird die Kugel noch weiter geschickt, so verschwindet die Assoziation zwischen Kugel und ZahlenFeld schein-

bar. Das liegt daran, daß sie nun auf ein ausgeblendetes ZahlenFeld verweist. Die Assoziation wird wieder sichtbar, sobald dieses Zahlenfeld eingeblendet oder die Assoziation auf ein sichtbares Feld gesetzt wird.

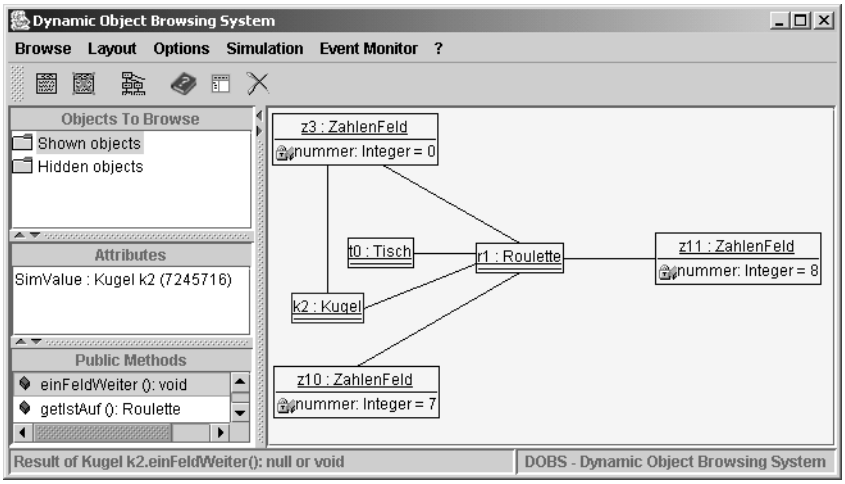


Abbildung 72: Kugel auf Feld 0

Für eine weitere Einarbeitung in DOBS und weitere Simulationen kann die Analyseumgebung in den (leeren) Anfangszustand zurückgesetzt werden. Dies geschieht über den Menüeintrag »*Reset* DOBS« im »*Browse*«-Menü.

7.3 Referenz der Menüeinträge

Die Menüleiste

...

Die Werkzeugleiste

...

Die Kontextmenüs

...

KAPITEL 8

GUI-Bibliothek FGrafik

8.1 Konzepte

8.1.1 Einführung

FGrafik stellt eine intuitiv zu verwendende Grafikbibliothek für Fujaba zur verfügung, die es mit einfachen Mitteln ermöglicht, Fujaba Projekte mit grafischen Oberflächen zu versehen.

Dabei wurde im Rahmen des LIFE³ Projekts Wert auf einfache Bedienbarkeit und schnelle Erlernbarkeit gelegt. FGrafik stellt dabei einen Satz von Klassen bereit, die sowohl Text Ein- und Ausgabe, Schaltflächen, grafische Komponenten und Ereignisbehandlung ermöglichen.

Dabei ist keinerlei Vorwissen über Java-Swingkomponenten notwendig, auch wenn FGrafik auf Swing aufbaut wurden alle verwendeten Swingkomponenten so gekapselt, das die Anwendung auch einem Anfänger keine Schwierigkeiten bereitet.

8.1.2 Grundlagen

FGrafik stellt eine Reihe von Komponenten zur Verfügung, um einfache grafische Umsetzungen unter Fujaba zu realisieren.

Die Grundlage stellt hierbei ein Fenster dar, das sich mit verschiedenen weiteren Komponenten (Ein- und Ausgabefelder, Knöpfe und Zeichenflächen, auf denen grundlegende grafische Elemente wie Kreise, Rechtecke und Linien liegen können) füllen lässt. Die Position der Komponenten wird für die Ein- und Ausgabefelder, Knöpfe und Zeichenflächen über Koordinaten bestimmt, die eine Position für das Layout angeben. Dabei wird automatisch so viel Platz reserviert, wie für die jeweiligen Komponenten notwendig ist. Sollte es erforderlich sein, vergrößert sich die Fenstergröße ebenfalls automatisch.

Alternativ kann auch ein Schachbrettlayout verwendet werden, bei dem das Fenster in Bereiche mit vordefinierter Größe aufgeteilt wird und die Komponenten jeweils in diesen Feldern zentriert werden. Dies ist besonders praktisch, wenn man z.B. ein Memory Spiel realisieren will, bei dem es auf gleichmäßige Anordnung der Komponenten ankommt und ein automatisches Layout eher hinderlich wäre.

Zeichenobjekte, sogenannte Symbole (dazu gehören Kreise, Rechtecke und Linien) können auf Zeichenflächen eines Fensters Komponenten positioniert werden, wobei hierbei ein Pixelkoordinatensystem verwand wird.

Alle Koordinatensysteme beginnen im Punkte 0, 0 an der linken oberen Ecke eines Fensters oder Zeichenfläche und erstrecken sich dann abwärts bzw. nach rechts im positiven Bereich.

Die Bibliothek selbst liegt im Fujaba Package unter de.uni_paderborn.tools.fgrafik und kann im Bedarfsfalle selbst erweitert werden.

8.1.2.1 Hinzufügen des Fgrafik Template

Um FGrafik nutzen zu können fügt man zunächst zu dem selbsterstellten Fujaba-Projekt das FGrafik Template hinzu. Es Enthält Referenzen auf alle FGrafik Klassen sowie die notwendigen Assoziationen und Attribute.

Um das Template hinzuzufügen, wählen Sie im Menü **Tools > Create FGrafik Template**.

Nach kurzer Zeit erscheint nun im Projektbaum ein neues Klassendiagramm „FGrafik“. Sollte bereits ein solches Klassendiagramm existieren, warnt Fujaba Sie vor dem Überschreiben. Bitte vergewissern Sie sich, daß ein eventuell schon existierendes Diagramm nicht in anderen Klassendiagrammen bereits verwendet wird, da sonst alle Referenzen auf FGrafik verloren gehen.

Alle FGrafik Komponenten werden allein durch ändern ihrer Attribute an den jeweiligen Einsatzbedarf angepasst, Methodenaufrufe auf FGrafik sind deshalb nur in wenigen Fällen notwendig.

8.1.3 Fgrafik Template

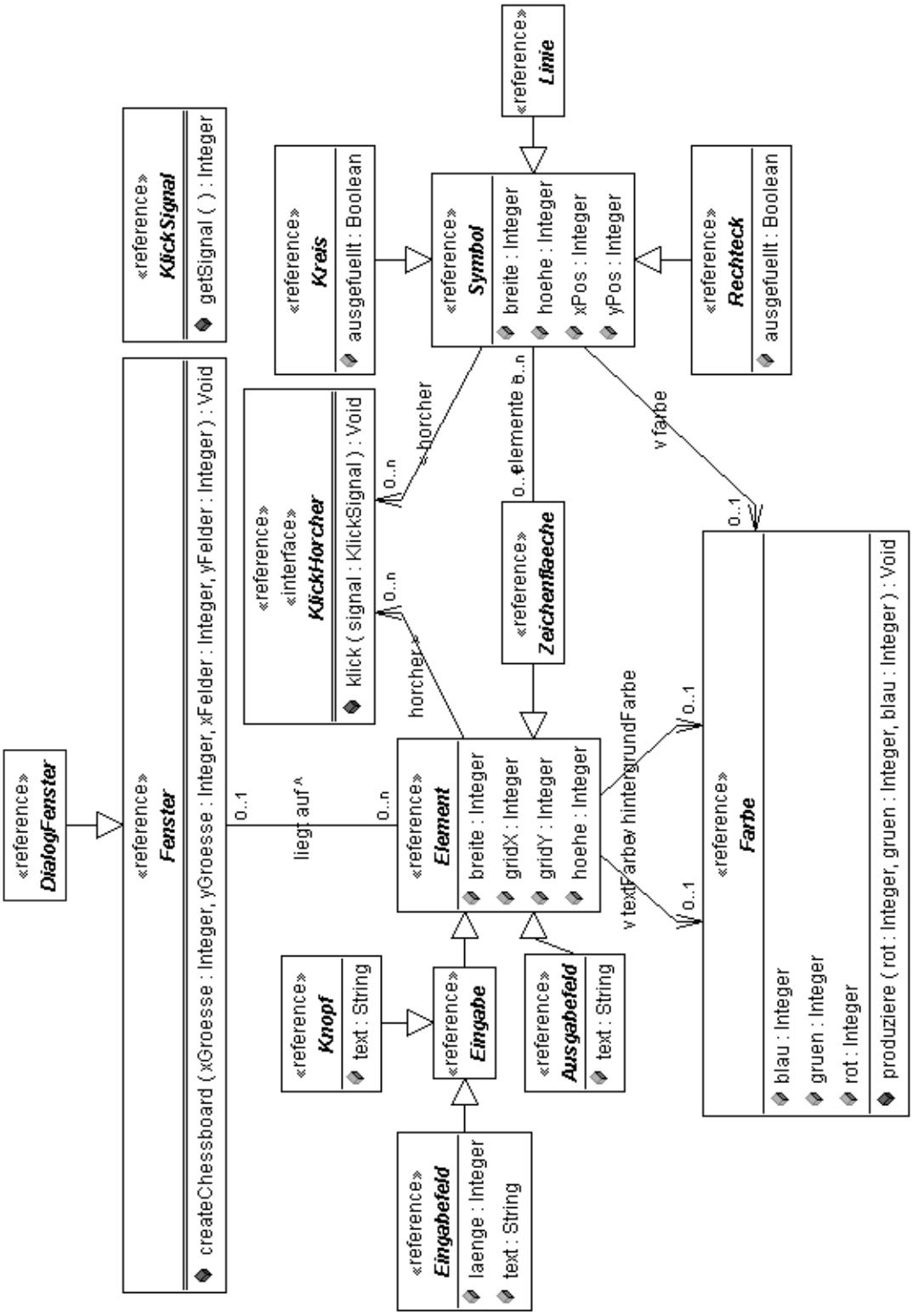


Abbildung 73: FGrafik Template

8.1.4 FGrafik Klassen

8.1.4.1 Fenster

Die Klasse *Fenster* stellt ein Fenster zur Darstellung verschiedener Unterklassen von *Element* zur Verfügung. Es verfügt über einen automatischen Layoutmechanismus, der diese Komponenten in einem Raster anordnet, wobei die Größe des Rasters abhängig von der Größe der darin enthaltenen Komponenten gewählt wird.

Sollte dieses Verhalten nicht erwünscht sein, kann über die Methode *createChessboard* ein Layout mit festen Rasterfeldgrößen erzeugt werden.

createChessboard(xGroesse : Integer, yGroesse : Integer, xFelder : Integer, yFelder : Integer)

xGroesse: Breite eines Feldes

yGroesse: Höhe eines Feldes

xFelder: Anzahl der Felder entlang der x-Achse

yFelder: Anzahl der Felder entlang der y-Achse

Alle Elemente, die sich dann noch außerhalb dieses Bereiches befinden werden automatisch gelayoutet.

8.1.4.2 DialogFenster

Die Klasse *DialogFenster* erbt von *Fenster* und unterscheidet sich von dieser Klasse nur insofern, das beim Schließen eines *DialogFensters* nicht System.exit() aufgerufen wird, d.h. Dobs als auch andere Objekte in der JavaVirtualMachine erhalten bleiben.

8.1.4.3 Element

Alle Elemente, die auf einem Fenster liegen dürfen, erben von der Klasse *Element*.

Über die Attribute *hoehe* und *breite* kann einem *Element* eine spezifische Größe zugeordnet werden, ansonsten wählt es die passende Größe selbst. Dies gilt nicht für *Zeichenflaeche*, bei der die Größe immer über die Attribute *hoehe* und *breite* zu bestimmen ist.

Mit *gridX* und *gridY* kann das Rasterfeld im *Fenster* bestimmt werden, in welchem das *Element* liegen soll.

Über die Assoziation *liegt auf* wird bestimmt, auf welchem Fenster das *Element* liegt, über *textFarbe* und *hintergrundFarbe* kann die Farbe des Hintergrunds oder des Textes eines *Elements* geändert werden. Mit der Assoziation *horcher* können Klassen, die das Interface *KlickHorcher* implementieren, an ein *Element* gebunden werden, so das es auf Mausereignisse reagiert.

8.1.4.4 Eingabe

Eingabe ist Oberklasse aller *Elemente*, die Eingabefunktionen zur Verfügung stellen.

Dient für spätere Erweiterungen von Fgrafik.

8.1.4.5 Knopf

Die Klasse *Knopf* stellt einen Knopf dar, der auf Mausklicks reagiert. Über das Attribut *text* kann die Beschriftung des Knopfes definiert werden.

8.1.4.6 Eingabefeld

Über ein *Eingabefeld* kann der Benutzer Text eingeben, der dann ¸ber das Attribut *text* abgerufen werden kann. ‹ber das Attribut *laenge* wird die maximale L‰nge des Textes definiert, den das *Eingabefeld* aufnehmen kann. Au?erdem kann es zum festlegen der Breite des *Eingabefeldes* genutzt werden.

8.1.4.7 Ausgabefeld

Ausgabefeld stellt eine einfache Möglichkeit zur Ausgabe von Text dar. Auch hier wird der gewünschte Text über das Attribut *text* gesetzt.

8.1.4.8 Zeichenflaeche

Das Element *Zeichenflaeche* dient zur Darstellung von grafischen *Symbolen* wie *Kreis, Linie* und *Rechteck*. Über die Assoziation *elemente* diese *Symbole* zu einer *Zeichenflaeche* hinzugefügt werden.Dabei überdecken zuletzt hinzugefügte *Symbole* ihre Vorgänger.

Die Positionierung geschieht über ein Pixelkoordinatensystem, das in der oberen linken Ecke einer *Zeichenflaeche* seinen Ursprung hat und sich dann im positiven Bereich nach unten und rechts erstreckt.

Die Angabe der Attribute *hoehe* und *breite*, die *Zeichenflaeche* von *Element* erbt, ist dabei zwingend erforderlich, da die *Zeichenflaeche* selbst kein automatisches Layout für Symbole durchführt und sonst die *Zeichenflaeche* nicht die notwendige Größe hat, um die gewünschten *Symbole* anzuzeigen. Mit der Assoziation *hintergrundFarbe* von Elemente kann eine beliebige Hintergrundfarbe für eine Zeichenfläche gewählt werden.

8.1.4.9 Symbol

Die Klasse Symbol stellt die Oberklasse aller grafischen Symbolkomponenten (*Linie, Kreis* und *Rechteck*) für *Zeichenflaechen* dar. Über die Assoziation *elemente* werden diese Symbole an eine *Zeichenflaeche* gebunden, die diese dann darstellt.

Über die Assoziation *farbe* kann einem Symbol eine *Farbe* zugewiesen werden und mit der Assoziation *horcher* können Klassen, die das Interface KlickHorcher implementieren, an ein *Symbol* gebunden werden, so daß es auf Mausereignisse reagiert. Dabei werden Mausereignisse an alle Symbole weitergereicht, die sich an der entsprechenden Position befinden, auch wenn sie durch andere Symbole verdeckt sein sollten.

Die Attribute *hoehe* und *breite* bestimmen die Größe eines Symbols, gemessen an einem das Symbol umschließenden Rechteck (siehe Grafik). Die Positionierung des Symbols geschieht über die Attribute *xPos* und *yPos*, die sich auf die obere linke Ecke eines das Symbol umschließenden Rechtecks beziehen.

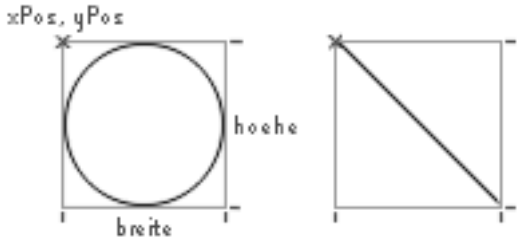


Abbildung 74: Symbol

8.1.4.10 Rechteck

Rechteck erbt von *Symbol* und stellt ein Rechteck dar. Setzt man das Attribut *ausgefuellt* auf true, erhält man ein ausgefülltes Rechteck. Das *Rechteck* reagiert auf seiner Fläche auf Mausklicks, wenn über die Assoziation *horcher* der Oberklasse *Symbol* ein *KlickHorcher* zugewiesen wurde.

8.1.4.11 Kreis

Kreis erbt von *Symbol* und stellt einen Kreis (oder auch Ellipse bei ungleichen *hoehe* und *breite* Werten) dar. Setzt man das Attribut *ausgefuellt* auf true, erhält man einen ausgefüllten Kreis.Ein *Kreis* reagiert auf Mausklicks, die auf der gedachten Fläche eines den Kreis umschließenden Rechtecks gemacht werden, wenn über die Assoziation *horcher* der Oberklasse *Symbol* ein *KlickHorcher* zugewiesen wurde.

8.1.4.12 Linie

Linie erbt von *Symbol* und stellt eine Linie dar, deren Anfangspunkt durch die Attribute *xPos* und *yPos* und deren Endpunkte durch *xPos + breite* und *yPos + hoehe* der Oberklasse *Symbol* bestimmt werden.

8.1.4.13 Farbe

Farbe dient zur Bestimmung der *hintergrundFarbe*, *textFarbe* eines *Elements* oder der *farbe* eines *Symbols*. Der Farbwert wird entweder mit Hilfe der Methode

produziere (rot : Integer, gruen : Integer, blau : Integer)

> *rot* Rotwert im RGB-System
>
> *gruen* Grünwert im RGB-System
>
> *blau* Blauwert im RGB-System

über das RGB-System bestimmt, wobei die Werte für RGB zwischen 0 und 255 liegen müssen, oder über den Konstruktor

Farbe(farbe : String) : Void

> *farbe* Name der Farbe, wobei *farbe* = {weiss, schwarz, rot, grün, blau, magenta, cyan, gelb}

oder über die Attribute *rot*, *gruen* und *blau*, deren Wert ebenfalls im Bereich 0-255 liegen muß.

8.1.4.14 KlickHorcher

Klassen, die das Interface KlickHorcher implementieren, können über die Assoziation *horcher Elementen* und *Symbolen* zugewiesen werden, die dann auf Mausklicks mit dem Aufruf der Methode *klick(signal : KlickSignal)* reagieren.

8.1.4.15 KlickSignal

Über ein *KlickSignal* kann innerhalb eines *KlickHorchers* besimmt werden, welche Maustaste den Aufruf eines *KlickHorchers* verursachte. Die drei in *KlickSignal* definierten Integerkon-

stanten *LINKS, MITTE, RECHTS* beziehen sich dabei auf entsprechende Maustasten, über die Methode *getSignal()* wird der Integerwert der gedrückten Maustaste wiedergegeben.

8.2 Benutzung von Bibliotheken

Für die Verwendung von Fgrafik ist es zunächst erforderlich, das FGrafik Template zu Ihrem bereits existierenden Fujaba Projekt hinzuzuladen.

8.2.1 Erstellen des FGrafik Templates

Wählen Sie dazu im Menü *Tools* den Menüpunkt **Create FGrafik Template**. Bitte Gedulden Sie sich einen Moment, besonders bei langsameren Rechnern kann das Erzeugen des Templates etwas Zeit in Anspruch nehmen. Im Projektbaum auf der linken Seite der Fujaba Oberfläche erscheint unter **Class Diagrams** nun ein neuer Eintrag *FGrafik*. Sollte das Template in Ihrem Projekt bereits existieren, erscheint eine Warnung, die Sie daran erinnert.

Bitte stellen Sie sicher, das Sie das Template nicht unbeabsichtigt überschreiben, da eventuell vorhandene Referenzen auf das Template aus Ihrem Projekt in diesem Falle zerstört werden.

8.2.2 Zufügen von FGrafik Komponenten

Nachdem das Template erstellt worden ist, können Sie beginnen, benötigte FGrafik Komponenten in Ihr selbsterstelltes Klassendiagram aufzunehmen. Wechseln sie hierzu mit Hilfe des Projektbaumes auf das gewünschte Klassendiagram und klicken sie mit der **rechten Maustaste** auf eine freie Fläche im Klassendiagram.

Das folgende Popupmenü erscheint:

Abbildung 75: Edit ClassDiagram

Wählen Sie den Menüpunkt **Edit ClassDiagram**, mit dem Sie Klassen aus dem FGrafik Template in Ihr eigenes Klassendiagram aufnehmen können um darauf zu Referenzieren.

Es erscheint folgender Dialog:

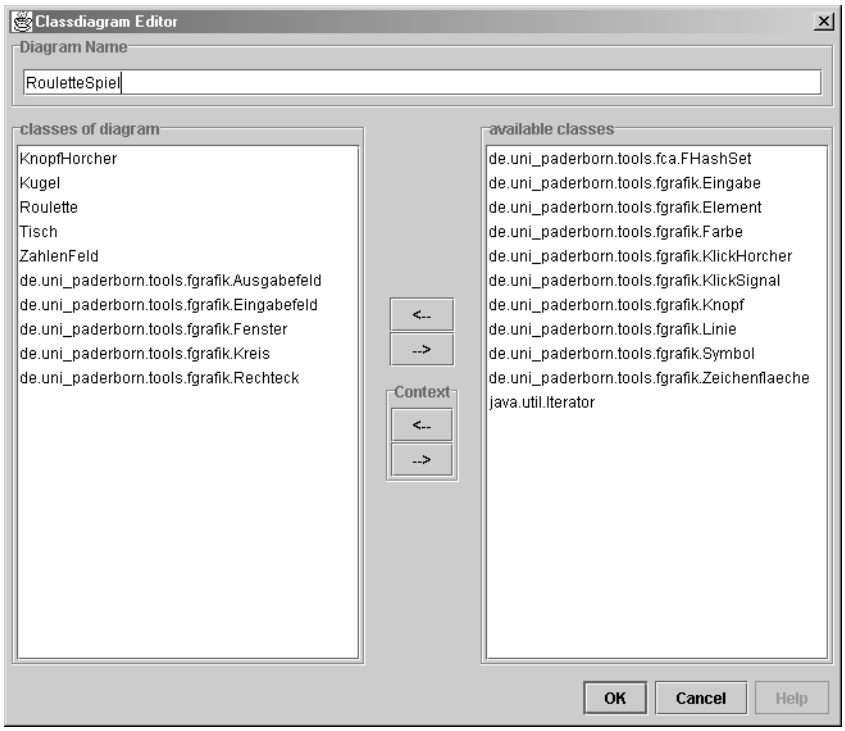


Abbildung 76: Classdiagram Editor

Wählen Sie auf der rechten Seite unter **available classes** mit der **linken Maustaste** die Komponenten von FGrafik aus, die Sie verwenden möchten und übertragen Sie diese durch drücken von <–– in Ihr Diagram. Alle Klassen, die in Ihrem Klassendiagram vorhanden sind, werden auf der linken Seite unter **classes of diagram** angezeigt. Durch halten von **Shift (Umschalttaste)** können Sie auch mehrere Klassen gleichzeitig selektieren. Im obigen Beispiel wurden die FGrafik Klassen *Ausgabefeld, Eingabefeld, Fenster,* Kreis und Rechteck in das Klassendiagram Flaschendrehen übernommen.

Möchten Sie Klassen von FGrafik zu einem späteren Zeitpunkt wieder aus Ihrem Diagram entfernen, öffnen Sie erneut den **Edit ClassDiagram** Dialog, selektieren die entsprechenden Klassen auf der linken Seite unter **classes of diagram** und entfernen Sie diese durch druck auf ––>.

Auf keinen Fall sollten Sie Klassen von FGrafik mit **Delete Class** oder der Taste **Entf**ernen aus dem Klassendiagram löschen. Diese Option entfernt Klassen nicht nur aus Diagrammen sondern **LÖSCHT** sie dauerhaft, auch aus dem FGrafik Template, so dass Ihr Projekt eventuell unbrauchbar wird. Ein Dialog warnt Sie vor diesem Schritt.

Nun können Sie die übertragenen Klassen in Ihrem eigenen Klassendiagram verwenden, indem Sie ihre eigenen Klassen über Assoziationen mit denen von FGrafik verknüpfen.

8.3 Guided Tour

In diesem Kapitel wird das Roulette Beispiel um eine grafische Benuzerschnittstelle mit Hilfe der FGrafik Bibliothek erweitert.

Das fertige Roulette mit FGrafik wird dann wie folgt aussehen:

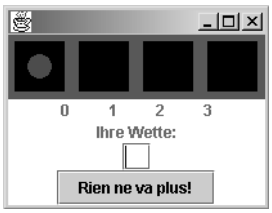


Abbildung 77: Roulette mit FGrafik

In einem Fenster befindet sich ganz oben eine *Zeichenfläche* mit grüner Hintergrundfarbe, darauf liegen vier schwarze *Rechtecke*, die Felder für eine rote Kugel in Form eines *Kreises* darstellen. Darunter befindet sich ein *Ausgabefeld*, das einen Zahlenstrahl zeigt um den Feldern Nummern zuzuweisen. Ein weiteres *Ausgabefeld* beschriftet ein *Eingabefeld*, indem man eine Wette auf ein Feld (0 - 3) eingeben kann. Schließlich kann man mit Hilfe des *Knopfes* „Rien ne va plus“ das Roulette starten.

Ein Ausgabefeld unter dem Knopf zeigt dann an, ob man gewonnen oder verloren hat.

Abbildung 78: Roulette mit FGrafik

8.3.1 Klassendiagramm Roulette mit FGrafik

Die neu hinzugekommenen Klassen für das Roulette mit FGrafik sind in der folgenden Abbildung blau markiert.

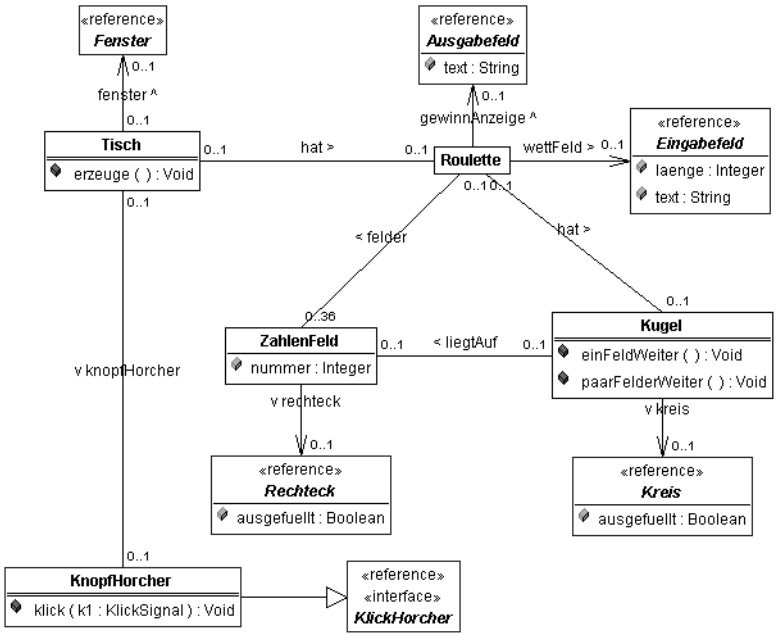


Abbildung 79: Roulette mit FGrafik Klassendiagramm

Tisch wurde um eine Assoziation auf ein *Fenster* erweitert um die Elemente von FGrafik darzustellem. Das Roulette hält eine Assoziation auf ein *Eingabefeld*, in das der Benutzer seine Wette eingeben kann, als auch auf ein *Ausgabefeld*, um anzuzeigen, ob er gewonnen oder verloren hat. Jedes *ZahlenFeld* wird durch ein *Rechteck* repräsentiert, auf einem davon liegt die *Kugel*, dargestellt durch einen *Kreis*.

Eine Sonderrolle nimmt die Klasse *KnopfHorcher* an, die später für die Ereignisbehandlung des *Knopfs* zuständig ist. Sie wurde nicht wie die anderen blau markierten Klassen aus dem FGrafik Klassendiagramm importiert, sondern im Roulette Klassendiagramm neu erstellt und importiert das Interface *KlickHorcher* aus FGrafik.

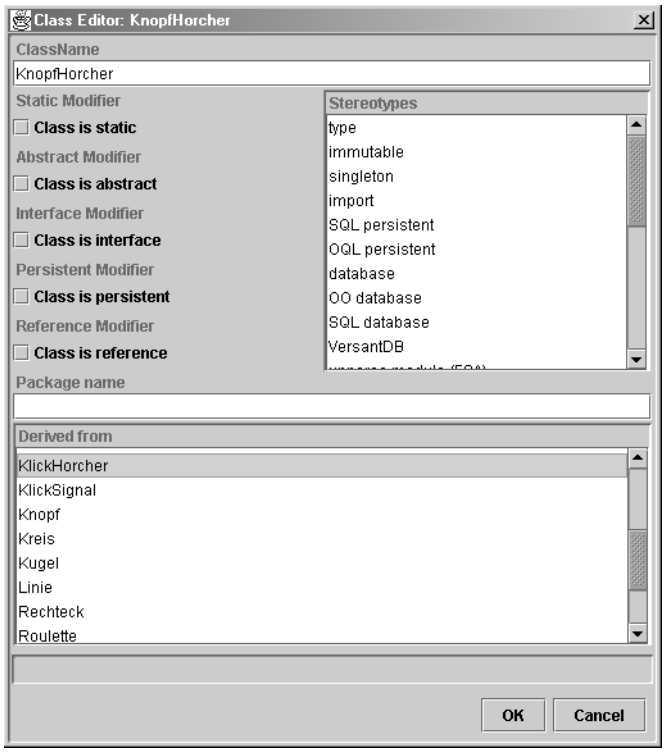


Abbildung 80: Class Editor - KnopfHorcher

Dieses Interface definiert die Methode *klick*, die bei Ereignissen auf Elementen und Symbolen aufgerufen wird, wenn der *KlickHorcher* über die Assoziation *horcher* diesen zugeordnet wurde.

8.3.2 Storydiagramm der Methode Tisch.erzeuge()

Nun müssen noch zur Laufzeit passende Objekte für die Assoziationen erzeugt werden. Dafür verwendet man das Storydiagramm der Methode *erzeuge* der Klasse Tisch, mit denen das Verhalten dieser Methode beschrieben wird.

Nähere Details und den genauen Umgang mit Storydiagrammen erlernen Sie im ***Kapitel 5 Storrydiagramme****.*

Da für die grafischen Elemente einige Objekte mehr erzeugt werden müssen als in den vorangegangenen Guided Tours, wird in dem folgenden Roulettbeispiel ein eleganterer Weg zur erzeugung der ZahlenFelder und den damit assoziierten Rechtecken verwand. Folglich weicht das Storydiagram der Methode Tisch.erzeuge() von der vorangegangenen Guided Tour in einigen Details ab.

Zunächst einmal werden die verwendeten Farben festgelegt; *gruen* für die *hintergrundFarbe* der *Zeichenflaeche*, *schwarz* für die *Rechtecke* der *ZahlenFelder* und *rot* für den *Kreis*, der die Roulettekugel darstellt.

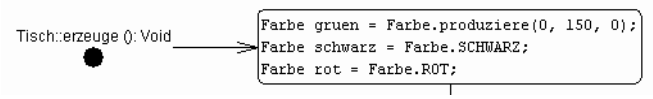


Abbildung 81: Tisch.erzeuge() Storydiagramm 1

Dies übernimmt bei dem Roulette Beispiel ein Statement, wobei die *Farbe gruen* über die statische Methode produziere erzeugt wird, während *schwarz* und *rot* Konstanten aus *Farbe* zugewiesen werden.

Danach erzeugt ein Storypattern alle weiteren benötigten Objekte bis auf die *ZahlenFelder* und ihre *Rechtecke*.

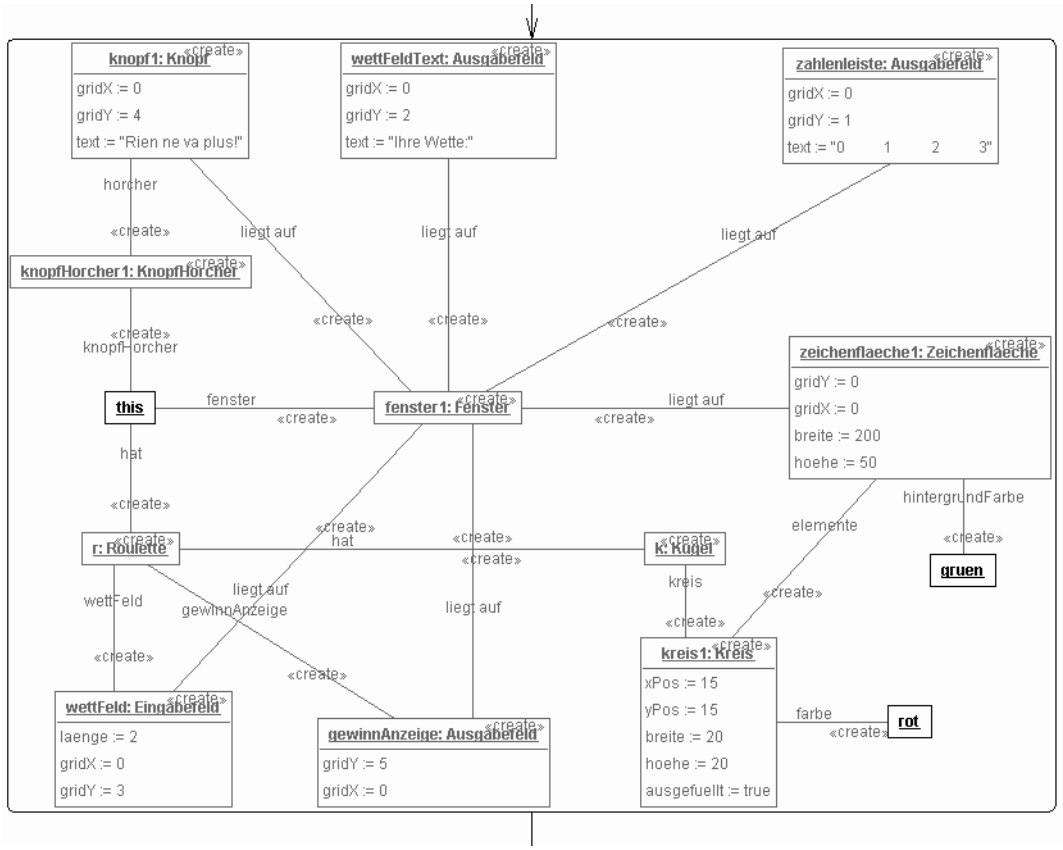


Abbildung 82: Tisch.erzeuge() Storydiagramm 2

Dem *Tisch*, im Storypattern durch das *this* Objekt verkörpert, wird über den Link *fenster* an seine Assoziation *fenster* ein *Fenster* zugewiesen, das dann alle weiteren Objekte von FGrafik über die Assoziation *liegt auf* hält.

Hiervon ausgenommen sind alle Objekte, die von *Symbol* erben, bei dem Roulette Beispiel *Kreis* und später die *Rechtecke*. Grafische *Symbol* Objekte müssen sich stets über die Assoziation *elemente* auf einer *Zeichenflaeche* befinden, *Zeichenflaeche* lässt dann eine pixelweise Positionierung über die Attribute (*xPos, yPos, hoehe, breite*) der *Symbole* zu, die hier über Assertions gesetzt werden. Über den Link *farbe* und *hintergrundFarbe* bekommen *Kreis* und *Zeichenflaeche* die *Farben rot* und *gruen*, das Attribut *ausgefuellt* lässt den *Kreis* später ausgefüllt erscheinen.

Alle anderen Objekte werden dem automatischen Layout von Fenster überlassen und erhalten über die Attribute *gridX* und *gridY* Positionen zugewiesen. Die Ausgabefelder *wettFeldText* und *zahlenleiste* sowie der *Knopf knopf1* erhalten über das Attribut *text* eine Beschriftung. Das Ausgabefeld *gewinnAnzeige* bleibt zunächst leer und wird erst im Zuge der Ereignisbehandlung von *KnopfHorcher* relevant.

Damit der *Knopf knopf1* auf einen Mausklick reagieren kann, weist ein Link ihm an seine Assoziation *horcher* ein Objekt der Klasse *KnopfHorcher* zu, die das Interface *KlickHorcher* aus FGrafik implementiert (siehe Klassendiagramm). Sobald der *Knopf* angeklickt wird, ruft er auf allen über *horcher* gebundenen *KlickHorchern* die Methode *klick* auf, die für *KnopfHorcher* in einem weiteren Storydiagram beschrieben wird. Ferner hat *KnopfHocher* eine Assoziation an den *Tisch*, über die seine *klick* Methode Zugriff auf *Kugel* und deren Methode *paarFelderweiter* hat, um auf ein Klicken mit einer Kugelbewegung zu reagieren. Auch *einFeldWeiter* und *paarFelderWeiter* werden über Storydiagramme modelliert, auf diese Methoden soll hier jedoch nicht weiter eingegangen werden, siehe dazu das **Kapitel 5: Storydiagramme**.

Der anschließende Teil des Storydiagramms der Methode *erzeuge* besteht aus einer Schleife mit der Laufvariablen i, die zur Erzeugung der vier *ZahlenFelder* und deren *Rechtecke* dient.

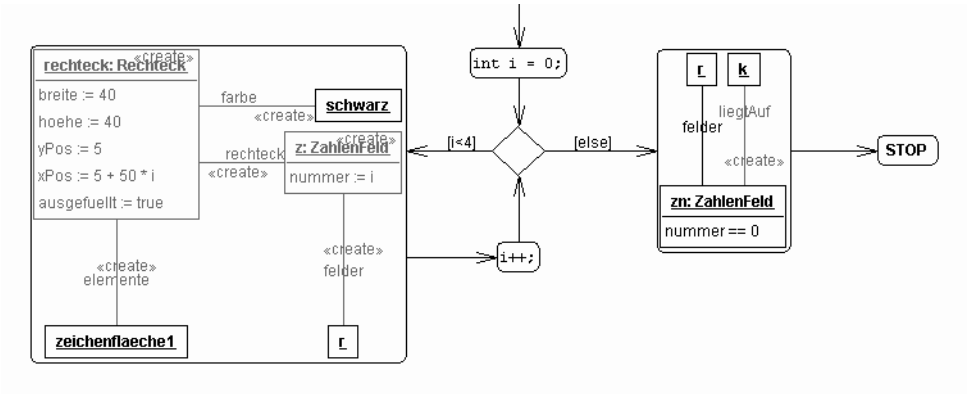


Abbildung 83: Tisch.erzeuge() Storydiagramm 3

Die Laufvariable i weist jedem der vier *Rechtecke* über die Assertion *xPos* eine neue Position zu, mit der es dann auf der *Zeichenflaeche zeichenfläche1* dargestellt wird. Über die Assoziation *rechteck* bekommt jedes neu erzeugte *ZahlenFeld z* sein *Rechteck* zugeordnet, das *ZahlenFeld* selbst erhält sein Attribut *nummer* über die Laufvariable i. Jedes *ZahlenFeld* wiederum wird über die Assoziation *felder* dem *Roulette r* zugewiesen, das hier, wie auch die *Farbe schwarz* und die *Zeichenflaeche zeichenflaeche1*, als schon im vorhergehenden Storypattern gebundenes (Bound) Objekt auftritt.

Nach Beendigung der Schleife wird die *Kugel* über die Assoziation *liegtAuf* auf das *ZahlenFeld* mit dem Attribut nummer := 0 plaziert.

8.3.3 Storydiagramm der Methode KnopfHorcher.klick()

Neu hinzugekommen ist ebenfals das Storydiagram der Methode *klick* von *KnopfHorcher,* die das Verhalten beim klicken des „Rien ne va plus!“ Knopfes modelliert.

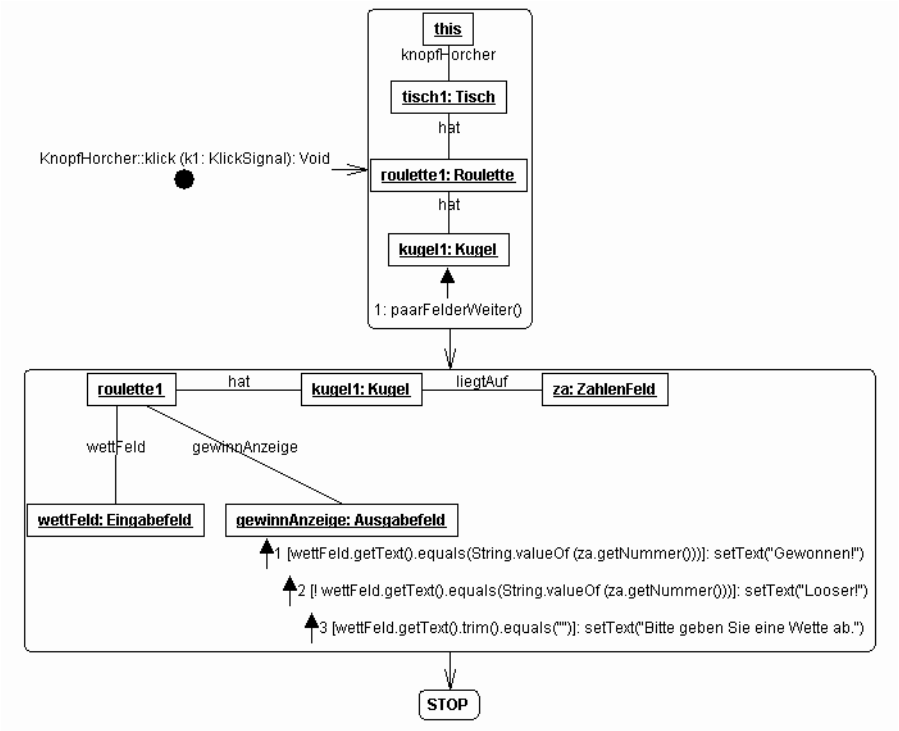


Abbildung 84: KnopfHorcher.klick() Storydiagramm

Das erste Storypattern läuft vom *KnopfHorcher,* der durch das *this* Objekt repräsentiert wird, an den Assoziationen entlang bis zur *Kugel*, um über die Methode *paarFelderWeiter* die *Kugel* auf ein zufälliges *ZahlenFeld* zu setzen.

Das darauffolgende Storypattern kümmert sich um die Auswertung der Wette, die notendigen Vergleiche werden über Constraints realisiert.

Dazu werden zunächst die zur Auswertung erforderlichen Objekte *wettFeld* vom Typ *Eingabefeld* und *za* vom Typ *ZahlenFeld* zugänglich gemacht. Drei Constraints vergleichen dann den im *wettFeld* eingegebenen String auf Übereinstimmung mit dem *nummer* Attribut des *ZahlenFeldes* auf dem die Kugel aktuell liegt. Stimmen Wette und ZahlenFeld Nummer überein, wird der Text des *Ausgabefelds gewinnanzeige* auf „Gewonnen!“ gesetzt, andernfalls auf „Looser!“. Sollte das *wettFeld* gar kein Text enthalten, fordert *gewinnAnzeige* auf, für das nächste Spiel eine Wette einzugeben.

ANHANG A

Abbildungsverzeichnis

ANHANG B

Index