1. Allgemeines

1.1. Kurzcharakterisierung

GeoX ist eine menügesteuerte grafische Java-Applikation, mit welcher sich geometrische Objekte und Konstruktionen erstellen lassen. Diese Konstruktionen können per Maus bewegt und ergänzt werden.

1.2. Systemvorraussetzungen

GeoX ist als eigenständige Applikation konzipiert, die über eine grafische Nutzerschnittstelle mit gängigen Fenster-Techniken bedient wird. GeoX benötigt als Voraussetzungen zum Starten die Java JRE 1.4. Aufgrund der Plattformunabhängigkeit ist kein spezielles Betriebssystem nötig, es muss aber über eine grafische Benutzeroberfläche verfügen.

1.3. Beschreibung der Produktumgebung

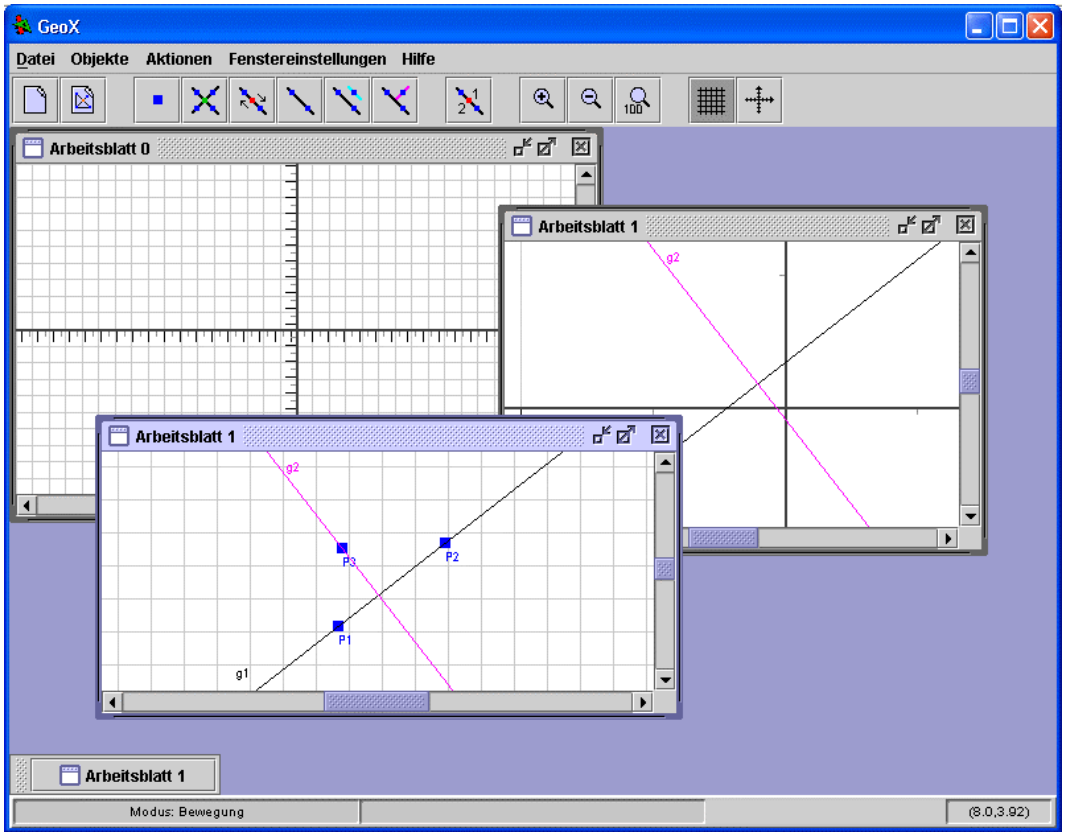
GeoX besitzt keine besondere Produktumgebung.

1.4. Abgrenzung

Es sind nur Funktionen mit Geraden und Punkten implementiert, Konstruktionen mit Kreisen oder Funktionsgraphen können nicht realisiert werden. Es existiert weiterhin keine Funktion zum Löschen von einfachen oder zusammengesetzten Objekten.

2. Produktübersicht

Nach der Initialisierung des Programms ist ein Hauptfenster zu sehen. Das Hauptfenster enthält am oberen Rand eine Menü- und Toolleiste, am unteren Rand einen Statusbalken und in der Mitte eine Desktop-Fläche, in der weitere Zeichenfenster geöffnet werden können.



Die Menüleiste hat die Einträge Datei, Objekte, Aktionen, Fenstereinstellungen und Hilfe.

Das Dateimenü enthält den Eintrag *Neues Arbeitsblatt*, welches ein neues Zeichenfenster auf dem Desktop erstellt. Weiterhin beinhaltet sie den Menüpunkt *Neue View*, welches auch ein Zeichenfenster auf dem Desktop erstellt. Dieses enthält jedoch die Konstruktion des zuletzt aktiven Zeichenfensters.
Der Menüeintrag *Schließen* schließt das aktive Zeichenfenster. Ist dieses Zeichenfenster das einzigste welches noch eine Sicht auf eine bestimmte Konstruktion hat, erscheint ein Dialog, welcher abfragt ob das Fenster wirklich geschlossen werden soll. Beim Auswählen des Menüeintrages *Beenden* wird durch einen Dialog gefragt ob das Programm wirklich beendet werden soll. Wenn dies mit „Ja“ bestätigt wird, werden alle Zeichenfenster und danach die Anwendung geschlossen.

Das Objektemenü enthält den Einträge *Punkt*, *Gerade, Gleiter, Schnittpunkt, Parallele und Senkrechte*. Nach Betätigen einer der Funktionen können nun die jeweiligen Objekte im aktuellen Zeichenfenster konstruiert werden.

Im Menü Aktionen kann der Unterpunkt Mittelpunkt angewählt werden, durch den ein Gleiter zum Mittelpunkt der beiden Spannpunkte der Geraden auf der er liegt wird.

Unter dem Menüpunkt Fenstereinstellungen befinden sich die Einträge *Zoom-In*, *Zoom-Out*, Zoom-100%, *Koordinatensystem* und *Hilfsgitterlinie*. Mit den Funtkionen *Zoom-In* und *Zoom-Out* wird der Zoomfaktor des aktuellen Zeichenfenster erhöht bzw. verringert, mit Zoom-100% wird er wieder auf 1 gesetzt. Per Menüeintrag *Koordinatensysteme* kann optional ein Koordinatensystem ein- bzw. ausgeblendet werden. Analog verhält es sich mit dem Menüeintrag *Hilfsgitterlinie*.

Das Hilfemenü beinhaltet zum einen den *Hilfe*eintrag, über den man zu einer Programmhilfe gelangt, welche in einem seperatem Fenster geöffnet wird. Weiterhin gibt es noch einen *Über*eintrag, der ein Informationsfenster öffnet.

In der Toolleiste sind wesentliche Buttons zur Steuerung des Programms integriert. Dazu gehören *Neues Arbeitsblatt*, *Neue View*, *Punkt*, *Gerade*, *Gleiter, Schnittpunk, Parallele, Senkrechte, Mittelpunkt, Zoom-In, Zoom-Out, Zoom-100%, Koordinatensystem und Hilfsgitter*. Die Buttons besitzen ein erklärendes Icon und sind äquivalent zu den oben aufgeführten gleichnamigen Menüeinträgen.

Die Statusleiste, welche sich am unteren Rand befindet, zeigt die jeweiligen Mauskoordinaten (rechts), den derzeitigen Programmzustand (links) und falls vorhanden Anweisungen an den Benutzer (mitte) an.

Der Desktop ist beim Öffnen des Programmes zunächst leer und nimmt im weiteren Verlauf alle geöffneten Zeichenfenster auf, auf denen Konstruktionen gezeichnet und dynamisch verändert werden können. Es kann durch Mausklick zwischen den einzelnen Zeichenfenstern hin- und hergeschaltet werden, dies kann ebenfalls über das Fenstermenü geschehen. Zeichenfenster können zusätzlich zur Funktion *Schließen* auch durch Drücken des x-Button geschlossen werden. Analog gilt das für das *Beenden* des Hauptfensters bzw. des Programms.

3. Grundsätzliche Design-Entscheidungen

3.1. Allgemeines

Die Paket- und Klassenstruktur ist durch das Model-View-Controller Paradigma gekennzeichnet.

Die **graphische Oberfläche** wurde mit Swing-Klassen, gängigen Fenster-Techniken und dem Ereignis-Aktions-Konzept realisiert.
Das Programm ist eine **M**ultiple **D**ocument **I**nterface Anwendung, dadurch ist es möglich mehrere Unterfenster in einem Hauptfenster zu öffnen und zu verwalten. Diese Unterfenster werden nur in dem Hauptfenster dargestellt und sind von diesem existenzabhängig.

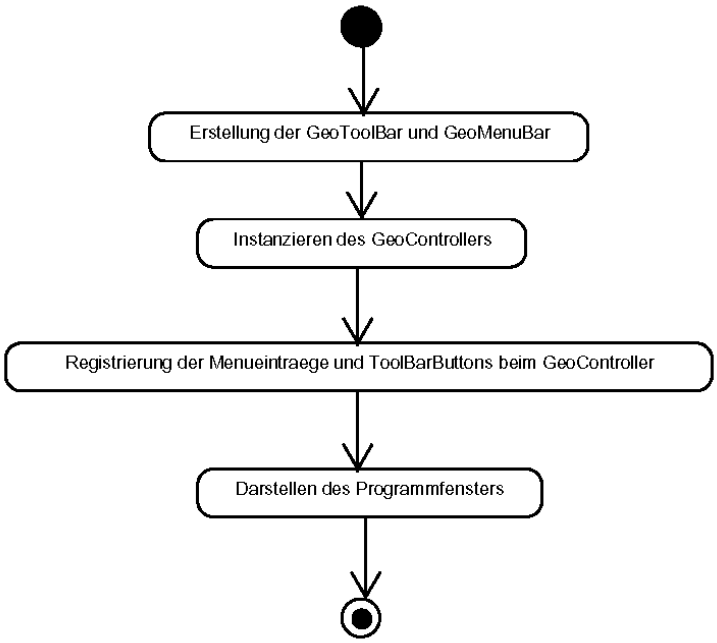
3.2. Darstellung geometrischer Objekte – die Modellsicht

Alle geometrischen Objekte sind durch eine ID eindeutig gekennzeichnet, welche beim Instanzieren des Objektes vergeben wird. Die ID ermöglicht ein leichteres Ansprechen der geometrischen Objekte.
Jedem geometrischen Objekt sind Abhängigkeiten sowohl nach „unten“, d.h. alle Objekte, die von diesem abhängen, als auch nach „oben“, d.h. alle Objekte, von denen das Objekt abhängt, zugeordnet. Für die Abhängigkeit nach „unten“ steht stellvertretend, die Verschiebung eines Punktes. Dabei müssen alle Objekte die von diesen Punkt abhängen, informiert und aktualisiert werden. Ein Beispiel für die Abhängigkeit nach „oben“ ist die Bewegung eines Gleiters. Da dieser sich nur auf der Gerade bewegen darf, von der er abhängt, muss er ihren Anstieg kennen.
Jeder Punkt besitzt Weltkoordinaten, die seine Lage im kartesischen Weltkoordinatensystem bestimmen.
Jede Gerade ist durch ihren Anstieg und ihre Schnittstelle mit der Y-Achse bestimmt.
Alle erzeugten geometrischen Objekte werden in einer Arrayliste verwaltet.

3.3. Darstellung geometrischer Objekte – die Visualisierung

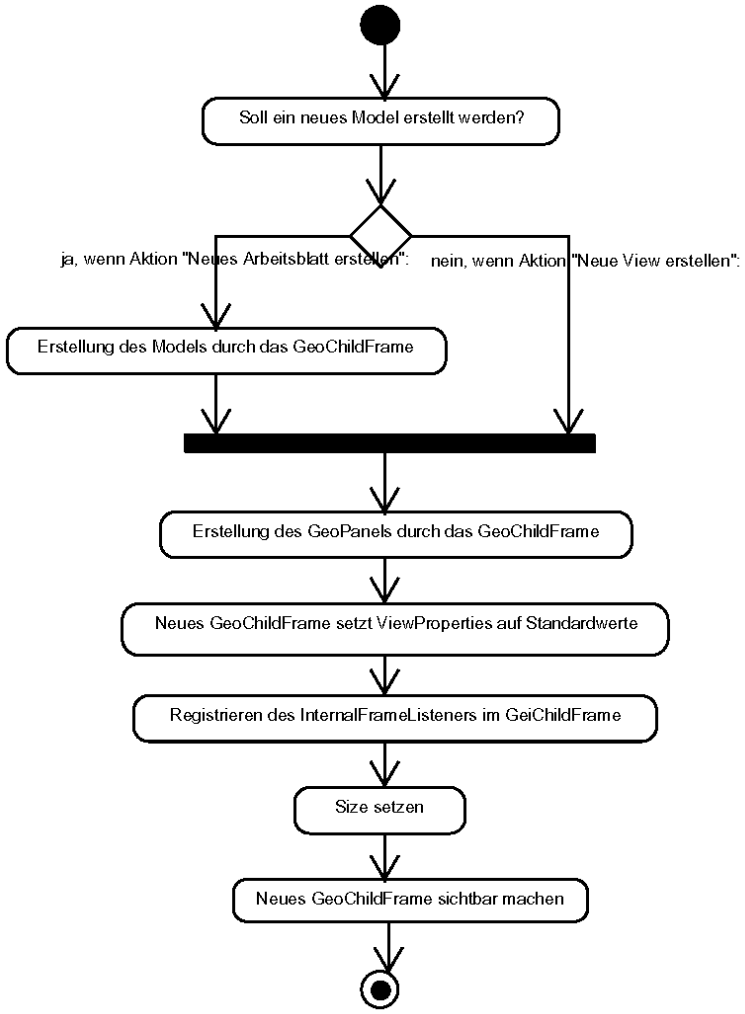
Zur Visualisierung muss einerseits zwischen den Weltkoordinaten des Models und den Pixelkoordinaten der Zeichenfläche umgerechnet werden. Anderseits müssen bei Lageänderung der geometrischen Objekte die Pixelkoordinaten der Zeichenfläche in Weltkoordinaten des Modells umgerechnet werden. Die Umrechnung der Koordinaten wird in der Klasse Affine Transform vorgenommen.
In der Klasse GeoPainter wird ermittelt von welchem Typ das geometrische Objekt ist und die entsprechende Paint-Methode wird aufgerufen.

3.4. Initialisierungsphase



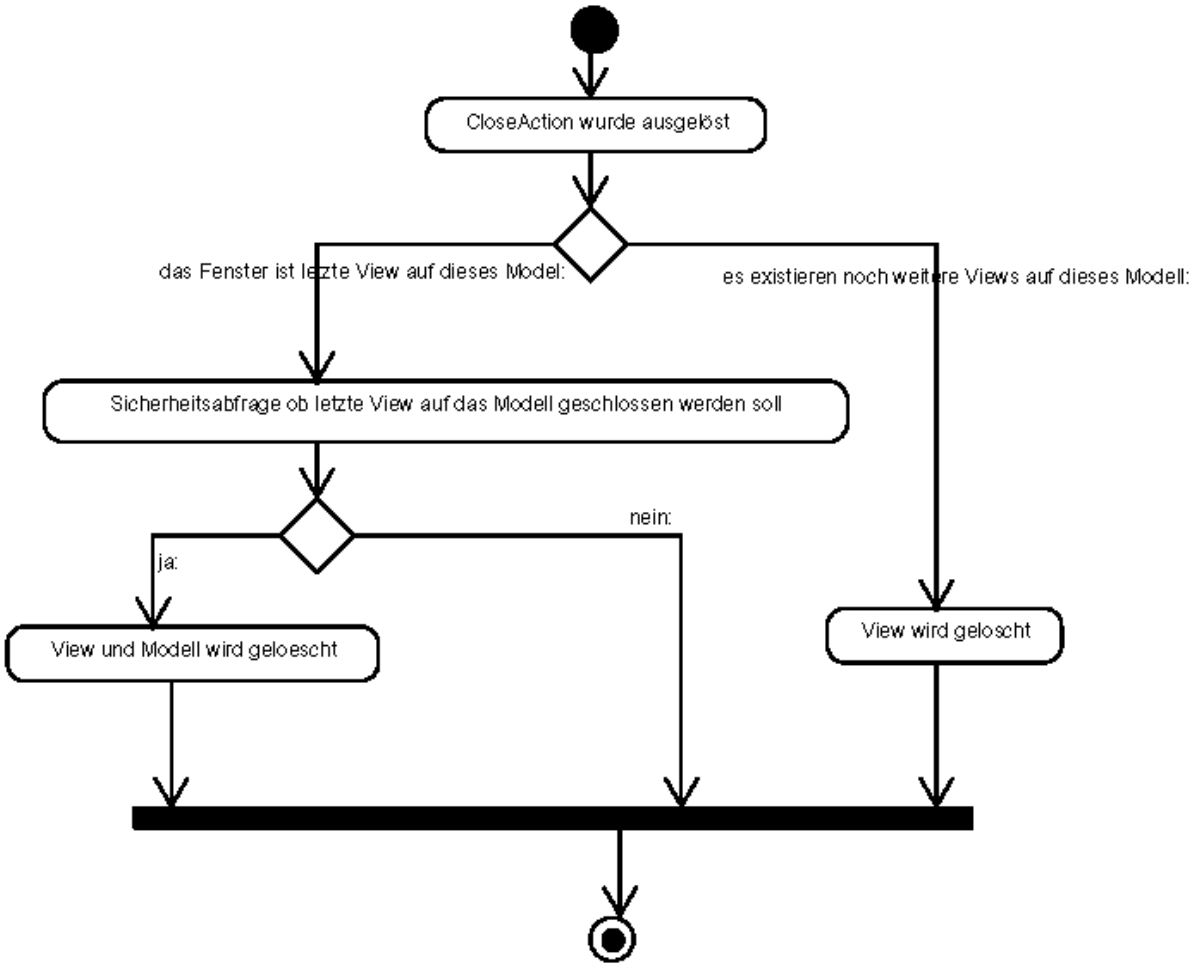
Bei der Initialisierung des Programms wird zunächst je ein Objekt von GeoToolBar, GeoMenuBar und GeoController instanziert. Anschließend werden die Menü- und Toolbareinträge beim GeoController registriert und das Programmfenster wird dargestellt.

3.5. Erstellen eines Zeichenfensters



Dieses Aktivitätsdiagramm beschreibt den Ablauf beim Erstellen eines neuen Zeichenfensters. Wichtig dabei ist zu unterscheiden ob ein Arbeitsblatt angelegt wird, d.h. Zeichenfenster und Model oder ob nur ein View, d.h. ein Zeichenfenster mit Referenz auf ein schon vorhandenes Model, angelegt wird.

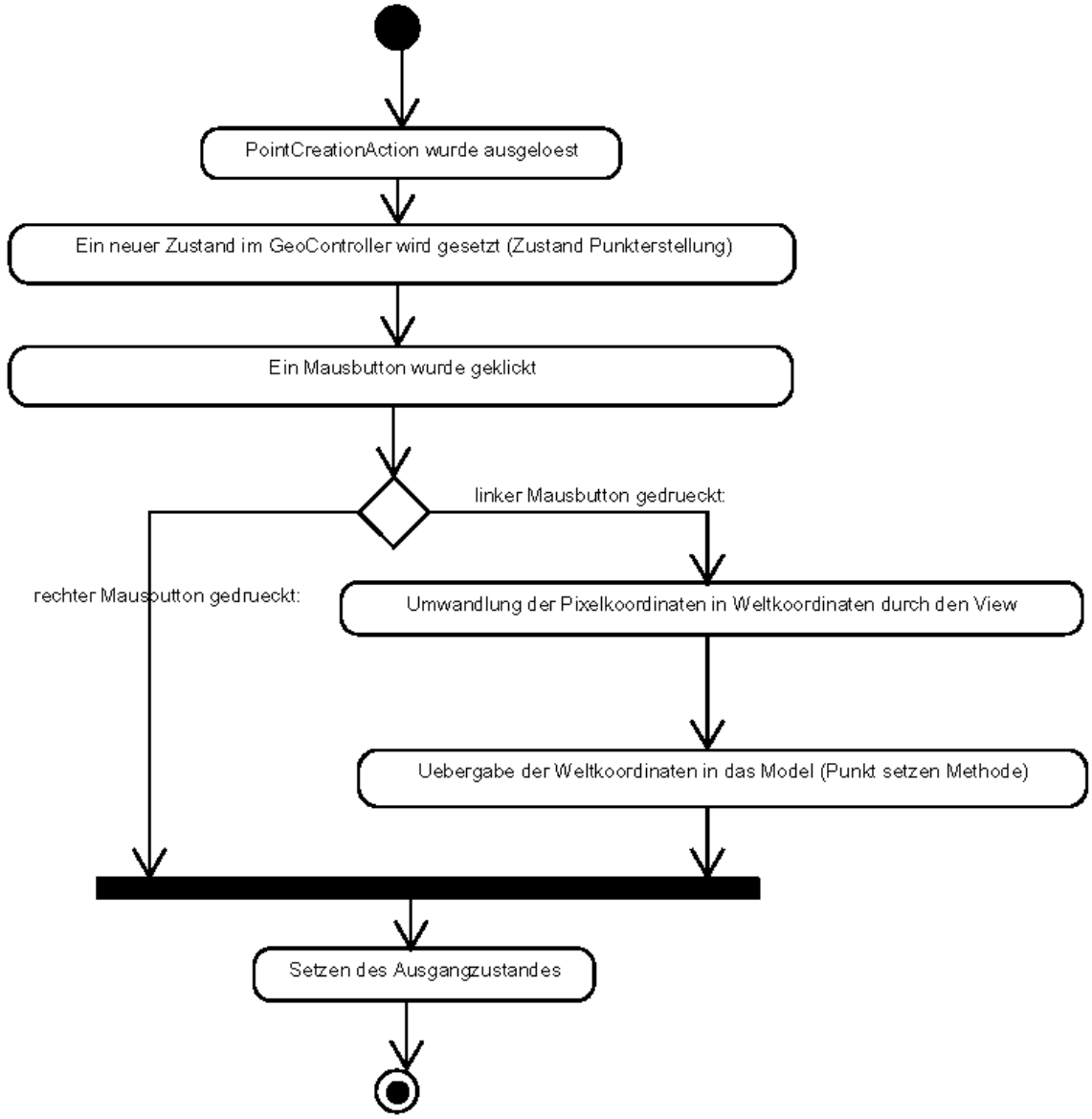
3.6. Schließen eines Zeichenfensters



Beim Schließen eines Zeichenfensters ist zu beachten, dass ein Modell mehrere Views besitzen kann. Jede View besitzt eine Referenz auf genau ein Modell. Soll nun ein Zeichenfenster geschlossen werden, welches als einziges noch eine Referenz auf das dessen Modell hat so wird zur Sicherheit der User über die Korrektheit dieses Vorgangs gefragt. Existieren jedoch weitere Views dieses Modells, d.h. gibt es noch Zeichenfenster die die gleiche Konstruktion nutzen, dann wird das Zeichenfenster sofort geschlossen.

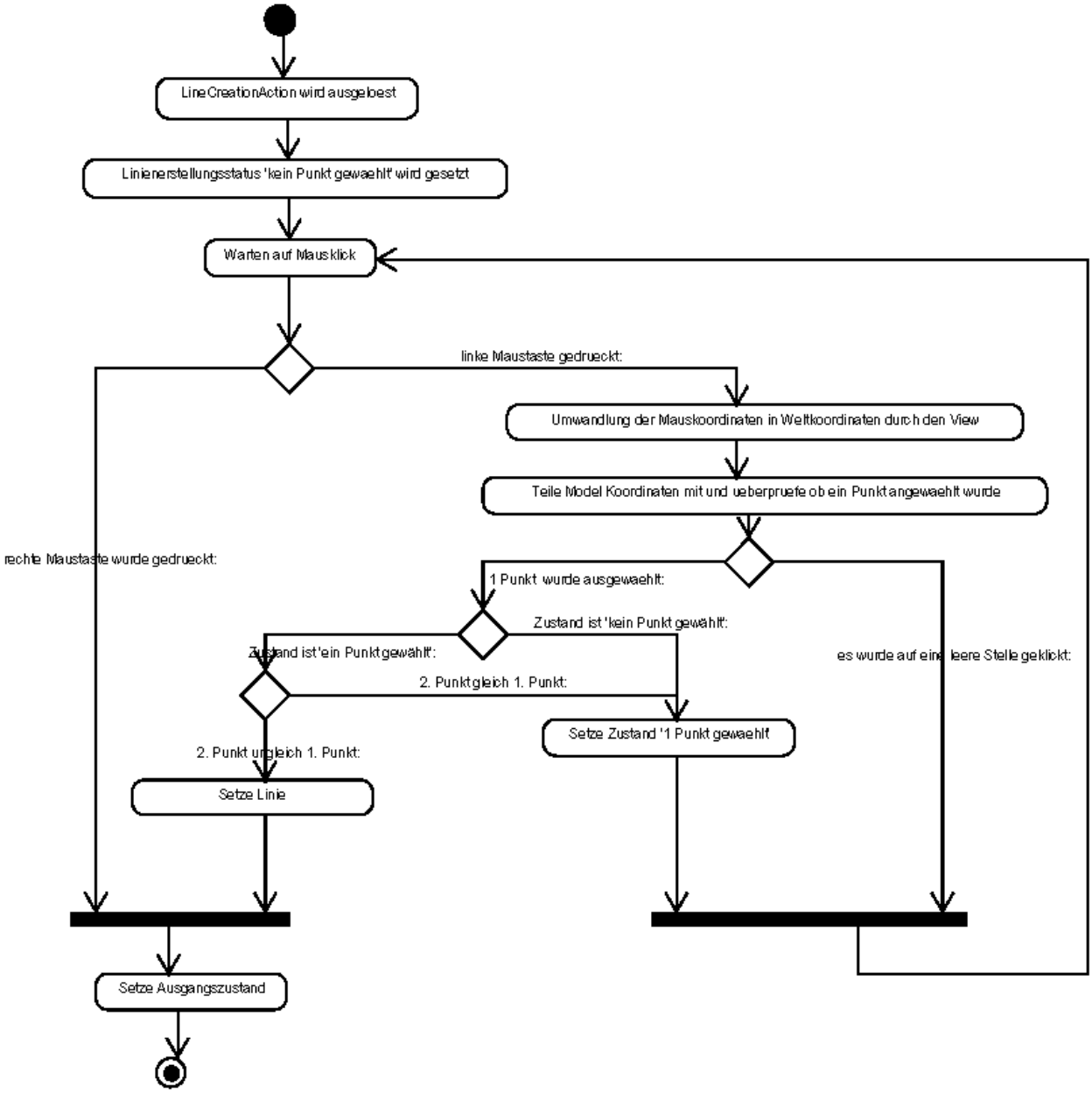
3.7. Konstruktion von geometrischen Objekten

3.7.1. Erstellen von Punkten



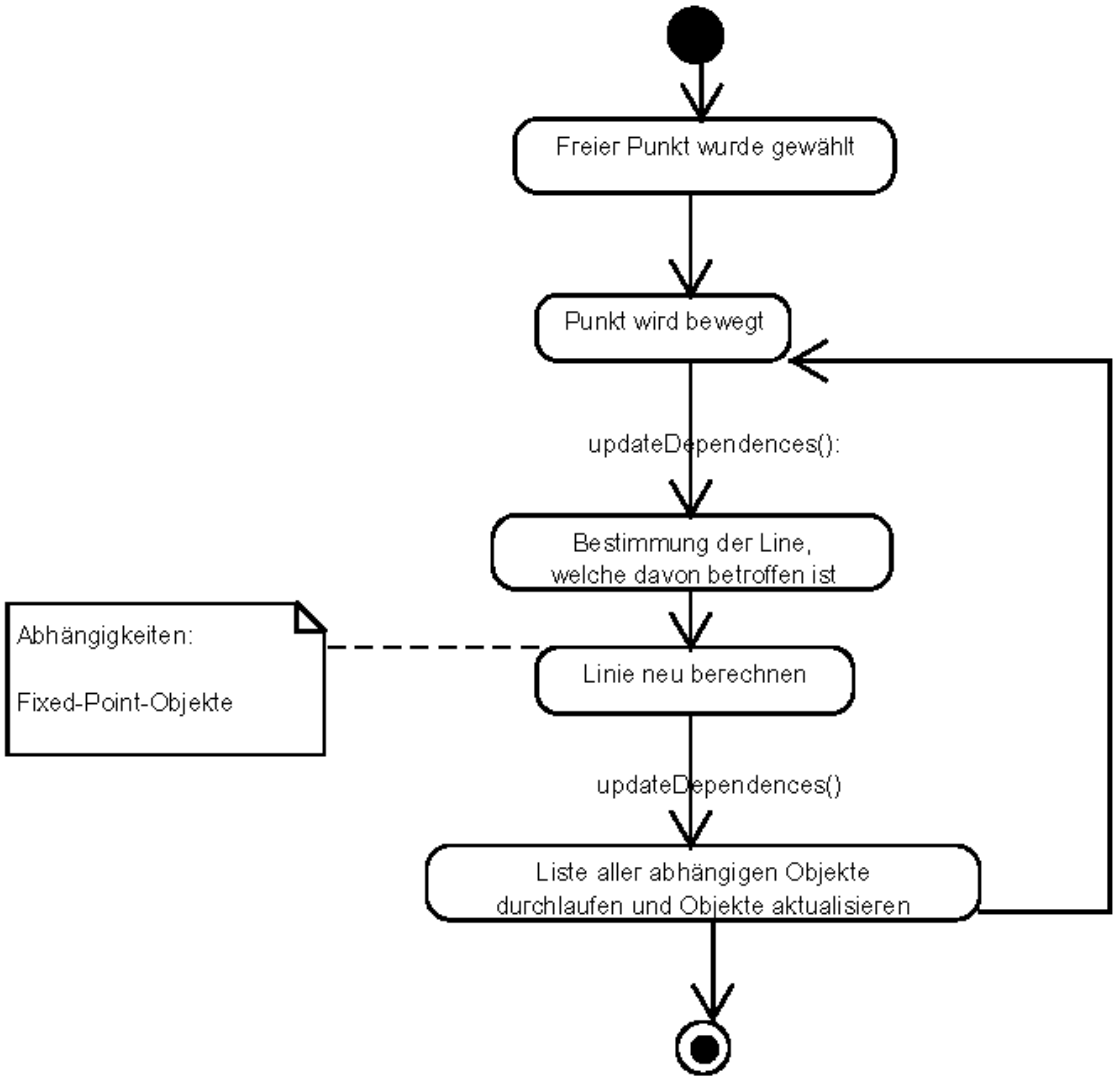
Durch Auswählen des Menüeintrags *Punkt erstellen*, wechselt der GeoController in den Zustand Punkterstellung. Wird die linke Maustaste auf dem Zeichenfenster gedrückt, wird ein Punkt auf dem Zeichenfenster gezeichnet. Die Punkterstellung kann durch Drücken der rechten Maustaste abgebrochen werden. Nach der Punkterstellung bzw. dem Abbruch wechselt der GeoController wieder in den Ausgangszustand.

3.7.2. Erstellen von Geraden



Durch Auswählen des Menüeintrags *Gerade erstellen*, wechselt der GeoController in den Zustand Geradenerstellung - kein Punkt. Nach Auswählen des ersten Punktes wechselt der GeoController in den Zustand Geradenerstellung – ein Punkt. Wurde nun ein zweiter Punkt ausgewählt, wird die Gerade konstruiert und in die Zeichenfläche gezeichnet.

3.7.3. Verschieben von geometrischen Objekten



Der zu verschiebende Punkt muss per Maus markiert werden. Durch Festhalten der linken Maustaste, wird der angewählte Punkt entsprechend der Mausbewegung verschoben. Dabei wird ein „Verschieben"-Mauscursor angezeigt. Wird die linke Maustaste losgelassen, so wird der Punkt an der aktuellen Mausposition gezeichnet. Abhängige Geraden, mit allen von ihr abhängigen Objekte müssen nun auch verschoben werden.

4. Paket- und Klassenstruktur

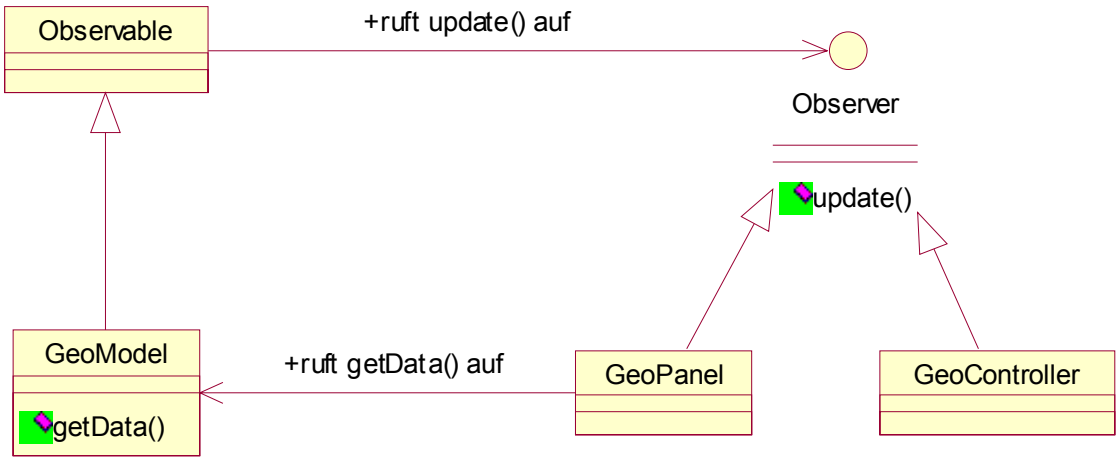
Die Paketstruktur von GeoX ist durch das Konzept von Model-View-Controller geprägt. Es ergibt sich eine Gliederung in drei Pakete.

model: Enthält alle Klassen zur Beschreibung von geometrischen Objekten.

view: Besteht aus Klassen die das GUI realisieren und aus der Klasse GeoPainter, welche das Aussehen der verschiedenen geometrischen Objekte beschreibt.

controller: Setzt sich aus den Klassen für die Ereignisbehandlung zusammen.

Um das Model-View-Controller Konzept umzusetzen, müssen das Model, die View und der Controller untereinander kommunizieren können. Dies wird durch das Verwenden des Interface Observer, der „Überwachende“, und der Klasse Observable, das zu „Überwachende“, erreicht. GeoPanel und GeoController überwachen das GeoModel, d.h. wird das GeoModel verändert werden die entsprechenden update() Methoden der registrierten Observer aufgerufen. In der update()-Methode von GeoPanel werden die Daten durch die getData() Methode vom GeoModel übermittelt und es werden entsprechende Änderungen am GeoPanel vorgenommen. Der GeoController wird durch den Aufruf der update()-Methode nur aktualisiert, er braucht die Daten des Models nicht.



4.1. Die Hauptklasse GeoX

Die Hauptklasse GeoX ist verantwortlich für das Starten der Applikation. Es erstellt in ihrer main-Methode das Hauptfenster, d.h. es wird ein Objekt von der Klasse GeoMainFrame instanziert.

4.2. Das Paket model

Die zentrale Klasse des Paketes model ist GeoElement. Aus ihr werden alle anderen Klassen abgeleitet, welche die geometrischen Objekte beschreiben.
Die Klasse besitzt als Attribut eine ID durch die jedes geometrische Objekt programmintern eindeutig identifiziert werden kann. Als weiteres Attribut enthält sie den Namen, welches die Objekte im Zeichenfenster für den Benutzer unterscheidbar machen.
Damit die einzelnen Objekte ihre Abhängigkeiten zu anderen Objekten kennen, werden die abhängigen Objekte in Form von den ID’s in zwei Arrays gespeichert. Wobei das erste Array die Abhängigkeiten nach „oben“, d.h. dieses Objekt hängt von anderen Objekten ab und das zweite Array die Abhängigkeiten nach „unten“, d.h. die Objekte hängen von diesem ab, speichert.
Die von GeoElement abgeleitete Unterklasse GeoPoint beschreibt freie Punkte, welche zusätzlich durch die Attribute X- und Y-Koordinaten gekennzeichnet sind. Aus GeoPoint ergeben sich die Unterklassen

GeoIntersectionPoint und GeoSlider, wobei ersteres ein Schnittpunkt zweier Geraden und letzteres einen Gleiter entlang einer Geraden darstellt.
Eine weitere Unterklasse von GeoElement ist GeoLine, welche Geraden definiert und über zusätzliche Attribute wie Anstieg m und Schnittstelle n mit der Y-Achse verfügt. Es existiert weiterhin die Klasse GeoParallel die eine Parallele einer Geraden durch einen Punkt repräsentiert und die Klasse GeoPerpendicular die eine Parallele einer Geraden durch einen Punkt darstellt. Beide sind Unterklassen von GeoLine.
Die Klasse GeoModel, welche von der Klasse Observable abgeleitet ist, repräsentiert je ein Model. Dieses enthält geometrische Objekte eines Zeichenfensters, welche in Form einer ArrayList gespeichert werden. Durch das Ableiten der Klasse Observable wird ein „Überwachen“ des Model vom Observer erreicht.

Klassendiagramm siehe Abbildung 1

4.3. Das Paket view

Das Hauptfenster wird durch die Klasse GeoMainFrame, abgeleitet von JFrame, erzeugt. Im Konstruktor werden Objekte der Klassen GeoDesktopPane, GeoController, GeoToolBar, GeoMenuBar und GeoStatusBar erstellt und zum Hauptfenster hinzugefügt.
Damit die Zeichenfenster verwaltet werden können, wird ein Objekt der Klasse GeoDesktopPane, abgeleitet von JDesktopPane, erstellt. Diesem wird eine Instanz der Klasse GeoDesktopManager, abgeleitet von DefaultDesktopManager, hinzugefügt. Der GeoDesktopManager stellt spezielle Methoden zur Behandlung, z.B. Aktivierung oder Schließen, der Zeichenfenster zur Verfügung.
Die Klasse GeoChildFrame, abgeleitet von JInternalFrame, definiert das Zeichenfenster. Es besteht aus einer Zeichenfläche, dem GeoPanel, welches von JPanel abgeleitet ist. Das Zeichenfenster bezieht die zu zeichnenden geometrischen Objekte aus seinem GeoModel. Das GeoPanel ist seinerseits in ein JScrollPane eingebettet, dieses ermöglicht einen Ausschnitt des GeoPanels anzuzeigen und diesen mit einer vertikalen und einer horizontalen ScrollBar auf dem GeoPanel entlang zu verschieben.
Die GeoMenuBar, welche abgeleitet von JMenuBar ist, enthält mehrere Menüeinträge des Typs JMenu.
Die Klasse GeoPainter stellt Methoden zum Zeichnen der verschiedenen geometrischen Objekte zur Verfügung, die in der Paint-Methode von GeoPanel aufgerufen werden.
Änderung vom GeoModel werden über das Observer-Interface überwacht und über die update()-Methode eine Nachricht an das GeoPanel gesendet.
In der Klasse AffineTransform werden die Mousekoordinaten in Weltkoordinaten und andersherum umgewandelt.

Klassendiagramm siehe Abbildung 2

4.4. Das Paket controller

Die zentrale Einheit des controller-Paketes ist die Klasse GeoController. Es existiert genau eine Instanz dieser Klasse, welches dem GeoMainFrame zugefügt wird. Der GeoController vereinigt alle Action-Klassen, abgeleitet von GeoAction und alle benötigten Listener, zur Überwachung der Maus und der Tastatur. Dazu gehören die Klassen GeoMouseAdapter, abgeleitet MouseInputAdapter, GeoKeyAdapter, abgeleitet von KeyAdapter, und die Klasse GeoChildAdapter, welche von InternalFrameAdapter abgeleitet ist und das Schließen bzw. Minimieren und Maximieren der GeoChildFrames ermöglicht. Analog zum GeoChildAdapter existiert der GeoWindowAdapter, abgeleitet von WindowAdapter, für das Hauptfenster.
Die verschiedenen Action-Klassen werden vom GeoController aufgerufen um die jeweiligen Anwendungsfälle auszulösen. Bei Jedem Anwendungsfall befindet sich der GeoController in einem anderen Zustand, es kann in einem Anwendungsfall auch mehrere Unterzustände für den Controller geben.
Der GeoController implementiert das Observer-Interface um die Models des Programms zu überwachen, ändert sich ein Modell wird der GeoContoller per update() aktualisiert.

Klassendiagramm siehe Abbildung 3 - 5

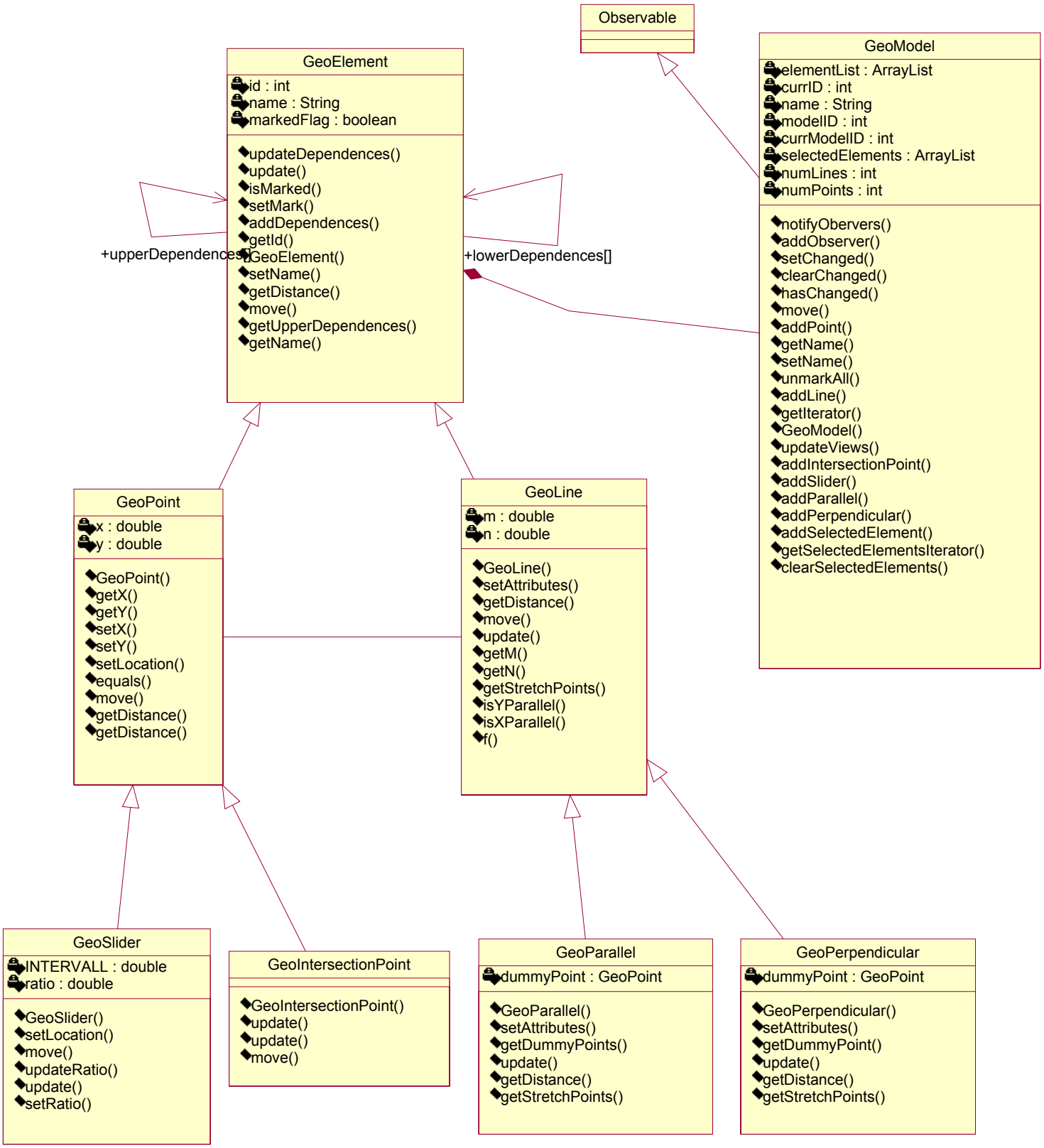

Observable
GeoElement
id : int
name : String
markedFlag : boolean
updateDependences()
update()
isMarked()
setMark()
addDependences()
getId()
GeoElement()
setName()
getDistance()
move()
getUpperDependences()
getName()
+upperDependences[]
+lowerDependences[]
GeoModel
elementList : ArrayList
currID : int
name : String
modelID : int
currModelID : int
selectedElements : ArrayList
numLines : int
numPoints : int
notifyObervers()
addObserver()
setChanged()
clearChanged()
hasChanged()
move()
addPoint()
getName()
setName()
unmarkAll()
addLine()
getIterator()
GeoModel()
updateViews()
addIntersectionPoint()
addSlider()
addParallel()
addPerpendicular()
addSelectedElement()
getSelectedElementsIterator()
clearSelectedElements()
GeoPoint
x : double
y : double
GeoPoint()
getX()
getY()
setX()
setY()
setLocation()
equals()
move()
getDistance()
getDistance()
GeoLine
m : double
n : double
GeoLine()
setAttributes()
getDistance()
move()
update()
getM()
getN()
getStretchPoints()
isYParallel()
isXParallel()
f()
GeoSlider
INTERVALL : double
ratio : double
GeoSlider()
setLocation()
move()
updateRatio()
update()
setRatio()
GeoIntersectionPoint
GeoIntersectionPoint()
update()
update()
move()
GeoParallel
dummyPoint : GeoPoint
GeoParallel()
setAttributes()
getDummyPoints()
update()
getDistance()
getStretchPoints()
GeoPerpendicular
dummyPoint : GeoPoint
GeoPerpendicular()
setAttributes()
getDummyPoint()
update()
getDistance()
getStretchPoints()

Abbildung 1

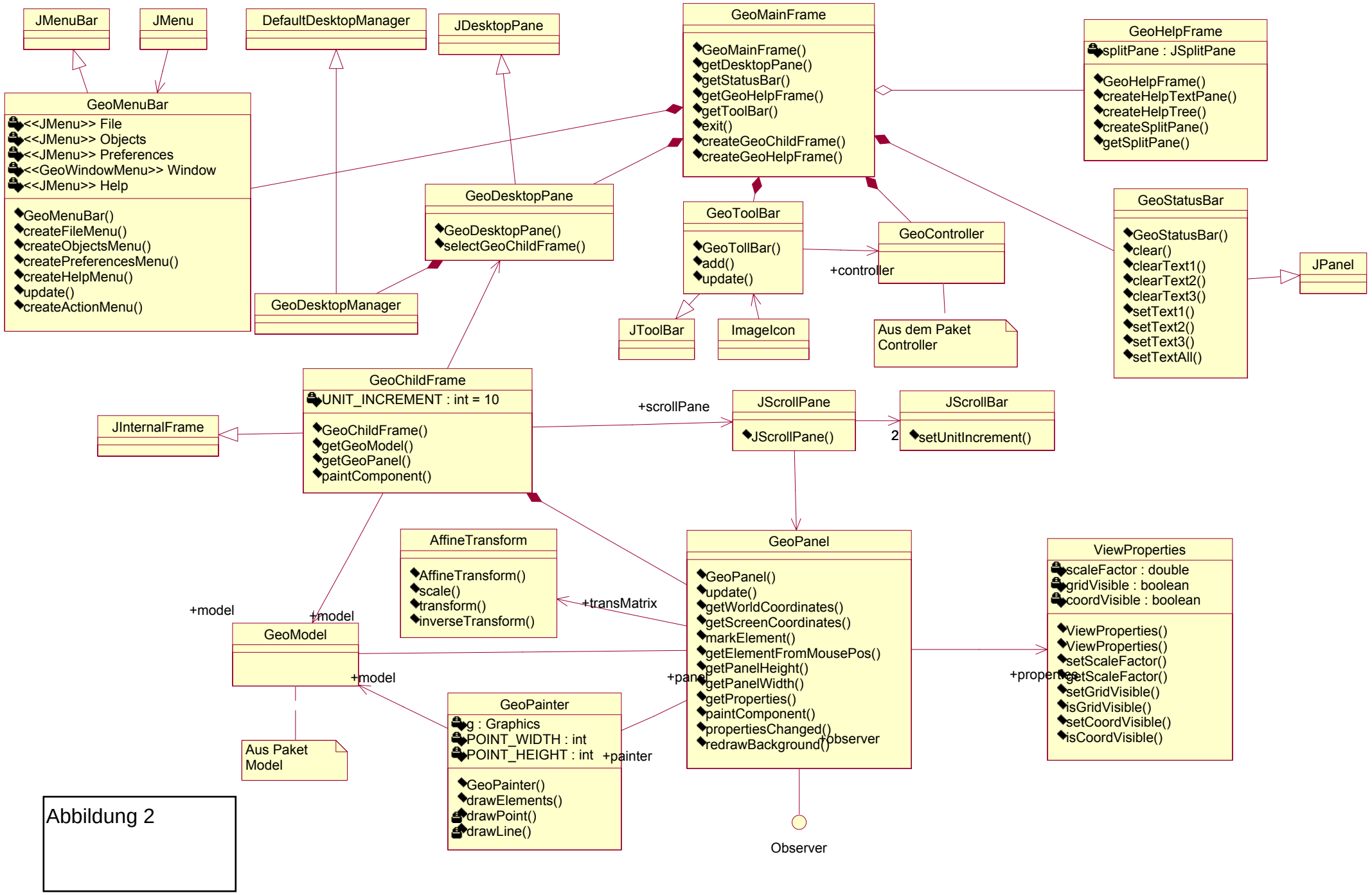

JMenuBar
JMenu
DefaultDesktopManager
JDesktopPane
GeoMainFrame
GeoMainFrame()
getDesktopPane()
getStatusBar()
getGeoHelpFrame()
getToolBar()
exit()
createGeoChildFrame()
createGeoHelpFrame()
GeoHelpFrame
splitPane : JSplitPane
GeoHelpFrame()
createHelpTextPane()
createHelpTree()
createSplitPane()
getSplitPane()
GeoMenuBar
<<JMenu>> File
<<JMenu>> Objects
<<JMenu>> Preferences
<<GeoWindowMenu>> Window
<<JMenu>> Help
GeoMenuBar()
createFileMenu()
createObjectsMenu()
createPreferencesMenu()
createHelpMenu()
update()
createActionMenu()
GeoDesktopPane
GeoDesktopPane()
selectGeoChildFrame()
GeoToolBar
GeoTollBar()
add()
update()
GeoController
+controller
GeoStatusBar
GeoStatusBar()
clear()
clearText1()
clearText2()
clearText3()
setText1()
setText2()
setText3()
setTextAll()
JPanel
GeoDesktopManager
JToolBar
ImageIcon
Aus dem Paket Controller
GeoChildFrame
UNIT_INCREMENT : int = 10
GeoChildFrame()
getGeoModel()
getGeoPanel()
paintComponent()
JInternalFrame
+scrollPane
JScrollPane
JScrollPane()
JScrollBar
2
setUnitIncrement()
AffineTransform
AffineTransform()
scale()
transform()
inverseTransform()
+transMatrix
GeoPanel
GeoPanel()
update()
getWorldCoordinates()
getScreenCoordinates()
markElement()
getElementFromMousePos()
getPanelHeight()
getPanelWidth()
getProperties()
paintComponent()
propertiesChanged()
redrawBackground()
+observer
ViewProperties
scaleFactor : double
gridVisible : boolean
coordVisible : boolean
ViewProperties()
ViewProperties()
setScaleFactor()
getScaleFactor()
setGridVisible()
isGridVisible()
setCoordVisible()
isCoordVisible()
+properties
+model
+model
GeoModel
+model
+panel
Aus Paket Model
GeoPainter
g : Graphics
POINT_WIDTH : int
POINT_HEIGHT : int
+painter
GeoPainter()
drawElements()
drawPoint()
drawLine()
Observer

Abbildung 2

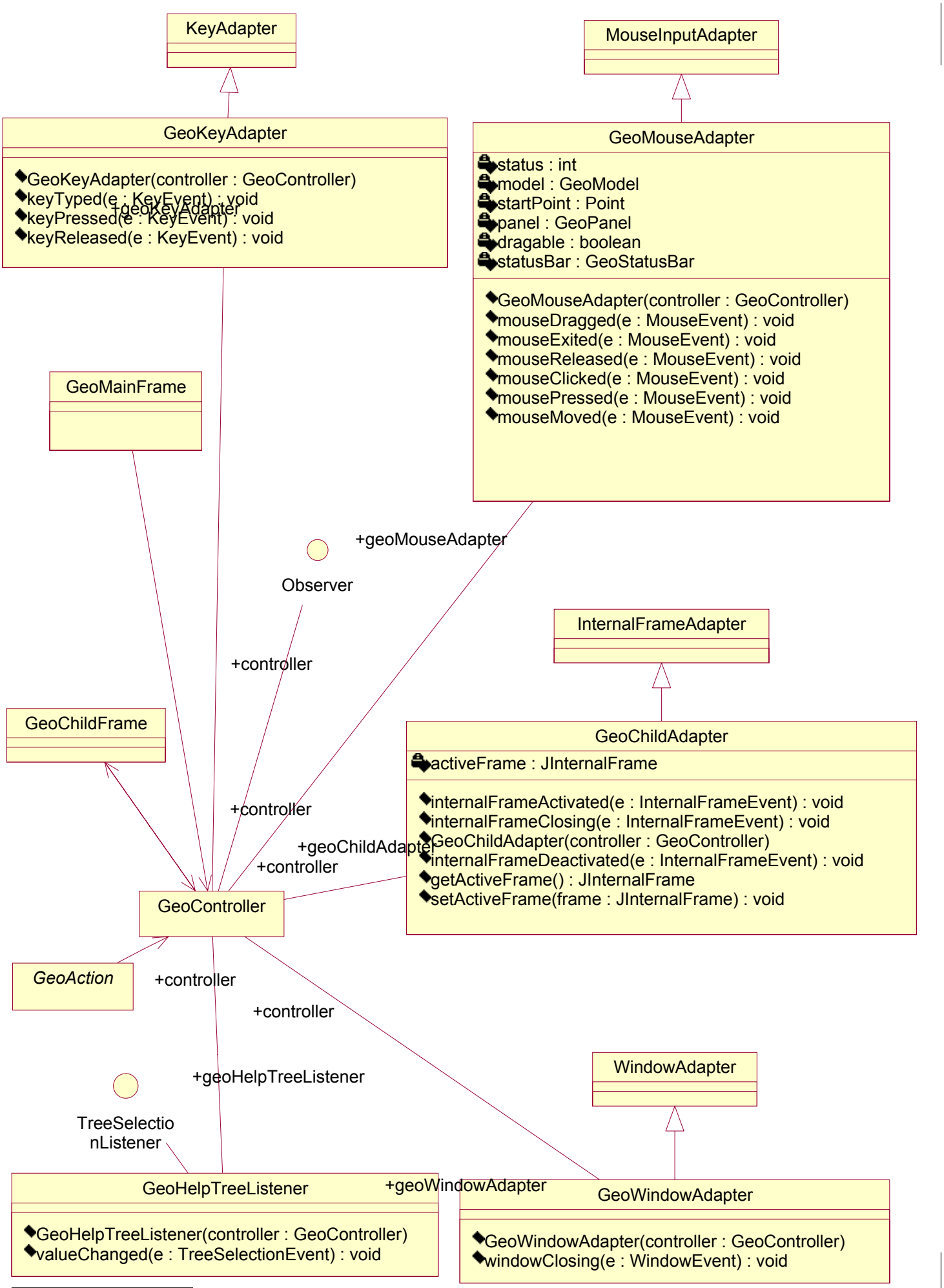

KeyAdapter
MouseInputAdapter
GeoKeyAdapter
GeoKeyAdapter(controller : GeoController)
keyTyped(e : KeyEvent) : void
keyPressed(e : KeyEvent) : void
keyReleased(e : KeyEvent) : void
+geoKeyAdapter
GeoMouseAdapter
status : int
model : GeoModel
startPoint : Point
panel : GeoPanel
dragable : boolean
statusBar : GeoStatusBar
GeoMouseAdapter(controller : GeoController)
mouseDragged(e : MouseEvent) : void
mouseExited(e : MouseEvent) : void
mouseReleased(e : MouseEvent) : void
mouseClicked(e : MouseEvent) : void
mousePressed(e : MouseEvent) : void
mouseMoved(e : MouseEvent) : void
GeoMainFrame
+geoMouseAdapter
Observer
+controller
InternalFrameAdapter
GeoChildFrame
GeoChildAdapter
activeFrame : JInternalFrame
internalFrameActivated(e : InternalFrameEvent) : void
internalFrameClosing(e : InternalFrameEvent) : void
GeoChildAdapter(controller : GeoController)
internalFrameDeactivated(e : InternalFrameEvent) : void
getActiveFrame() : JInternalFrame
setActiveFrame(frame : JInternalFrame) : void
+controller
+geoChildAdapter
+controller
GeoController
GeoAction
+controller
+controller
+geoHelpTreeListener
WindowAdapter
TreeSelectio
nListener
GeoHelpTreeListener
GeoHelpTreeListener(controller : GeoController)
valueChanged(e : TreeSelectionEvent) : void
+geoWindowAdapter
GeoWindowAdapter
GeoWindowAdapter(controller : GeoController)
windowClosing(e : WindowEvent) : void

Abbildung 3

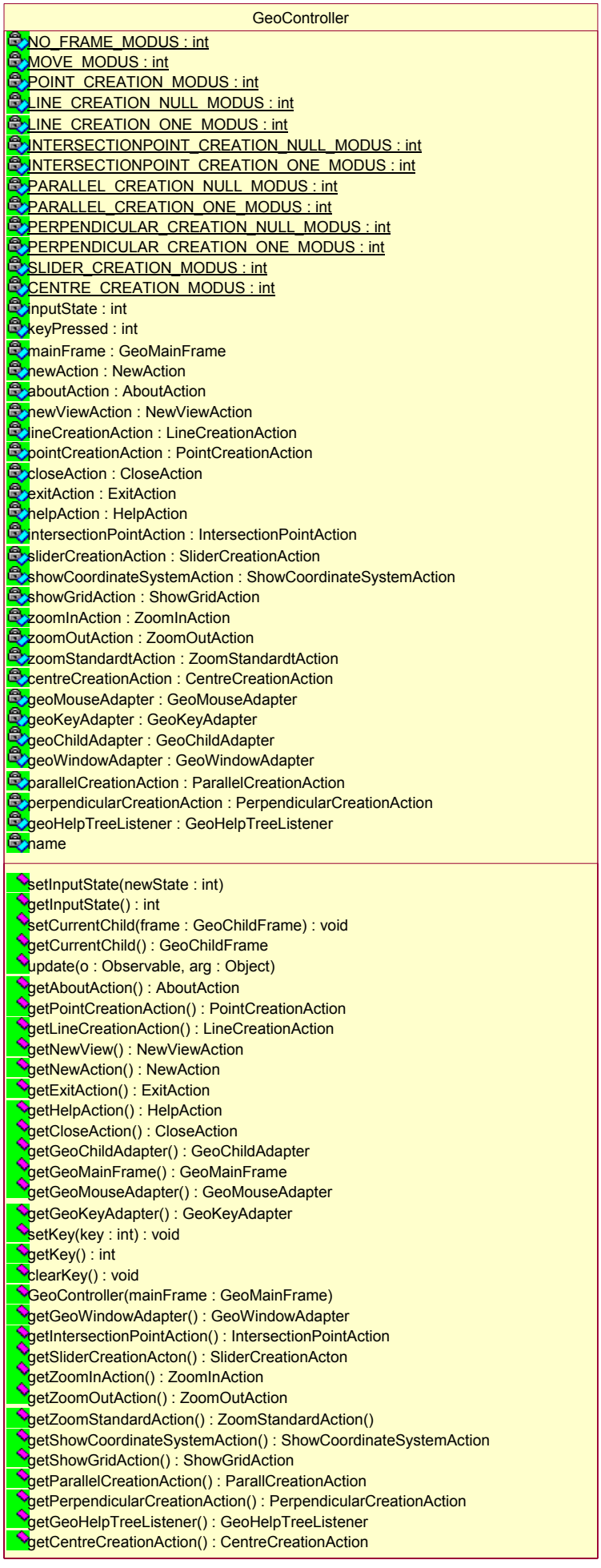

GeoController
NO_FRAME_MODUS : int
MOVE_MODUS : int
POINT_CREATION_MODUS : int
LINE_CREATION_NULL_MODUS : int
LINE_CREATION_ONE_MODUS : int
INTERSECTIONPOINT_CREATION_NULL_MODUS : int
INTERSECTIONPOINT_CREATION_ONE_MODUS : int
PARALLEL_CREATION_NULL_MODUS : int
PARALLEL_CREATION_ONE_MODUS : int
PERPENDICULAR_CREATION_NULL_MODUS : int
PERPENDICULAR_CREATION_ONE_MODUS : int
SLIDER_CREATION_MODUS : int
CENTRE_CREATION_MODUS : int
inputState : int
keyPressed : int
mainFrame : GeoMainFrame
newAction : NewAction
aboutAction : AboutAction
newViewAction : NewViewAction
lineCreationAction : LineCreationAction
pointCreationAction : PointCreationAction
closeAction : CloseAction
exitAction : ExitAction
helpAction : HelpAction
intersectionPointAction : IntersectionPointAction
sliderCreationAction : SliderCreationAction
showCoordinateSystemAction : ShowCoordinateSystemAction
showGridAction : ShowGridAction
zoomInAction : ZoomInAction
zoomOutAction : ZoomOutAction
zoomStandardtAction : ZoomStandardtAction
centreCreationAction : CentreCreationAction
geoMouseAdapter : GeoMouseAdapter
geoKeyAdapter : GeoKeyAdapter
geoChildAdapter : GeoChildAdapter
geoWindowAdapter : GeoWindowAdapter
parallelCreationAction : ParallelCreationAction
perpendicularCreationAction : PerpendicularCreationAction
geoHelpTreeListener : GeoHelpTreeListener
name
setInputState(newState : int)
getInputState() : int
setCurrentChild(frame : GeoChildFrame) : void
getCurrentChild() : GeoChildFrame
update(o : Observable, arg : Object)
getAboutAction() : AboutAction
getPointCreationAction() : PointCreationAction
getLineCreationAction() : LineCreationAction
getNewView() : NewViewAction
getNewAction() : NewAction
getExitAction() : ExitAction
getHelpAction() : HelpAction
getCloseAction() : CloseAction
getGeoChildAdapter() : GeoChildAdapter
getGeoMainFrame() : GeoMainFrame
getGeoMouseAdapter() : GeoMouseAdapter
getGeoKeyAdapter() : GeoKeyAdapter
setKey(key : int) : void
getKey() : int
clearKey() : void
GeoController(mainFrame : GeoMainFrame)
getGeoWindowAdapter() : GeoWindowAdapter
getIntersectionPointAction() : IntersectionPointAction
getSliderCreationActon() : SliderCreationActon
getZoomInAction() : ZoomInAction
getZoomOutAction() : ZoomOutAction
getZoomStandardAction() : ZoomStandardAction()
getShowCoordinateSystemAction() : ShowCoordinateSystemAction
getShowGridAction() : ShowGridAction
getParallelCreationAction() : ParallCreationAction
getPerpendicularCreationAction() : PerpendicularCreationAction
getGeoHelpTreeListener() : GeoHelpTreeListener
getCentreCreationAction() : CentreCreationAction

Abbildung 4

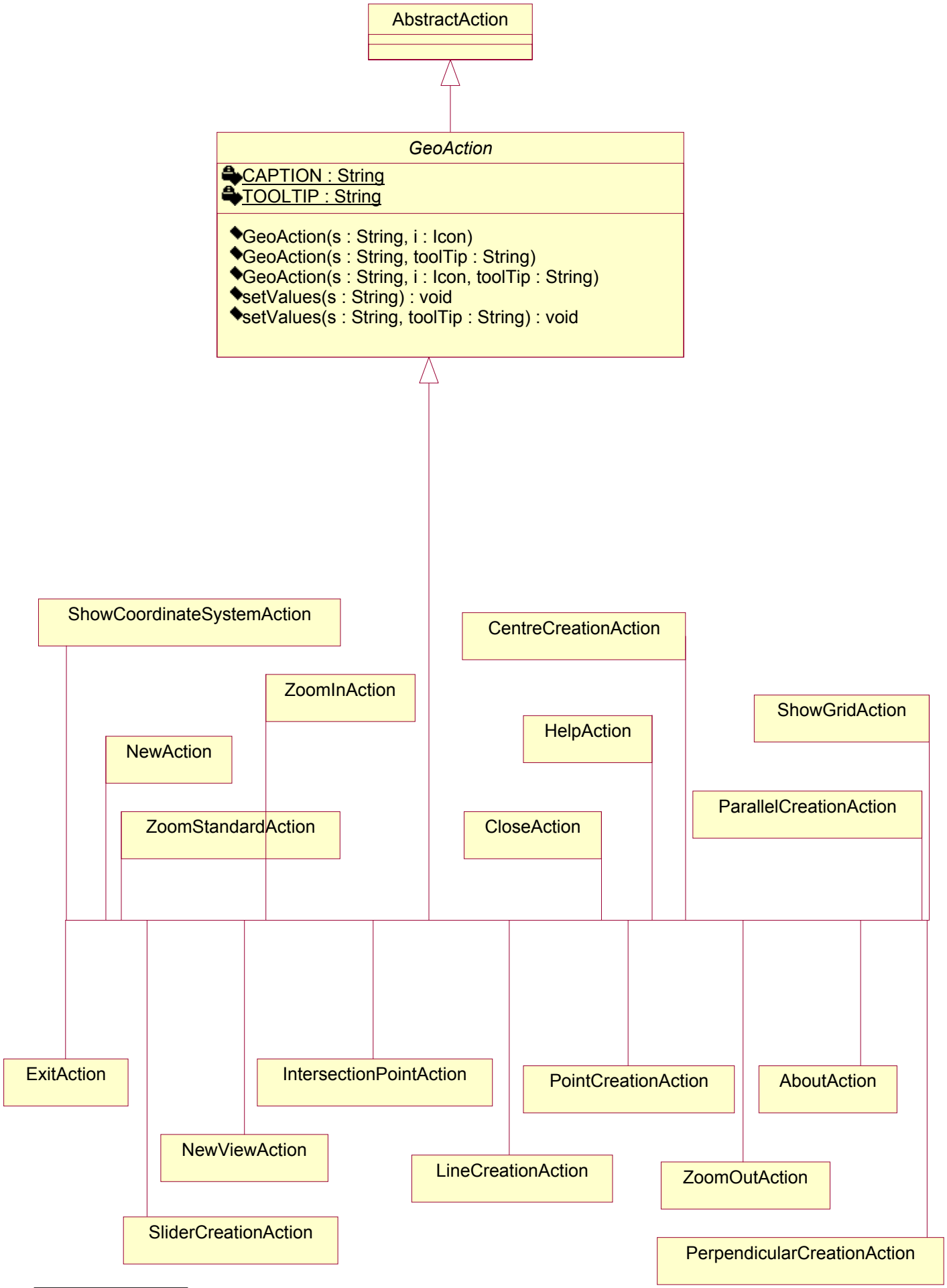

AbstractAction
GeoAction
CAPTION : String
TOOLTIP : String
GeoAction(s : String, i : Icon)
GeoAction(s : String, toolTip : String)
GeoAction(s : String, i : Icon, toolTip : String)
setValues(s : String) : void
setValues(s : String, toolTip : String) : void
ShowCoordinateSystemAction
CentreCreationAction
ZoomInAction
ShowGridAction
HelpAction
NewAction
ParallelCreationAction
ZoomStandardAction
CloseAction
ExitAction
IntersectionPointAction
PointCreationAction
AboutAction
NewViewAction
LineCreationAction
ZoomOutAction
SliderCreationAction
PerpendicularCreationAction

Abbildung 5