1. Allgemeines

1.1. Kurzcharakterisierung

GeoViewer ist eine menügesteuerte graphische Java-Applikation, mit der sich geometrische Konfigurationen visualisieren lassen, die in Form von GEO-Records vorliegen. Die algebraischen Rechnungen zur Bestimmung der Koordinaten der geometrischen Objekte erfolgt mit dem GeoProver-Paket und dem Computeralgebrasystem Maple

1.2. Systemvoraussetzungen

GeoViewer ist als eigenständige Applikation konzipiert, die über eine grafische Nutzerschnittstelle mit gängigen Fenster-Techniken bedient wird. GeoViewer benötigt als Voraussetzungen zum Starten die Java JRE 1.4., Maple (ab Version 5), das GeoProver-Paket für Maple sowie GEO-Records zur Eingabe. Die genaue Lage der einzelnen Ressourcen kann in einer Datei GeoViewer.rc spezifiziert werden.

1.3. Beschreibung der Produktumgebung

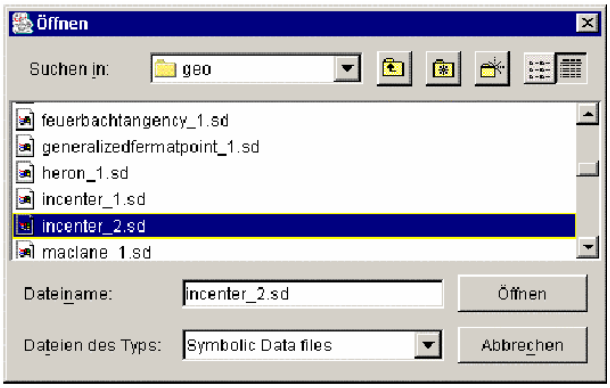
GEO-Records enthalten Beschreibungen geometrischer Beweisschemata, die in einem speziellen XML-artigen Format abgelegt sind, das auf der GeoCode-Spezifikation aufsetzt. Beides ist näher in der Dokumentation des SymbolicData-Projekts (http://www.symbolicdata.org) beschrieben. Ein GEO-Record enthält neben der Beschreibung der geometrischen Konfiguration eine Liste von unabhängigen Parametern, die für die konkrete Visualisierung mit geeigneten Zahlenwerten zu belegen sind. Die Parameterwerte haben Einfluss auf die Lage einzelner geometrischer Objekte (freier Punkte und Gleiter) und damit auch auf die Lage abgeleiteter Objekte. Verschiedene Parameterwerte ergeben also verschiedene Bilder derselben geometrischen Konfiguration. In diesem Sinne kann die Visualisierung „dynamisiert“ werden.
Zur Interpretation dieser Beweisschemata und Behandlung der algebraischen Fragen wird eine Implementierung des GeoCode-Standards im GeoProver-Paket für das Computeralgebrasystem Maple herangezogen.

1.4. Abgrenzung

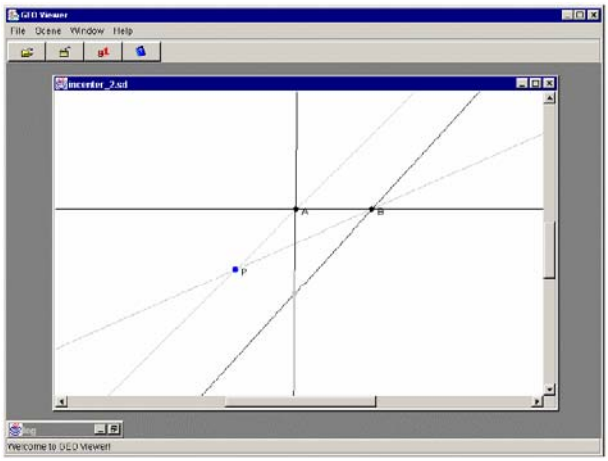
Ein GEO-Record kann auch abhängige Parameter enthalten, deren Werte sich als Lösungen eines Gleichungssystems ergeben, dessen Koeffizienten mit den unabhängigen Parametern variieren. Da die Behandlung von (nichtlinearen) Gleichungssystemen mit variierenden reellen Koeffizienten auch für Maple schwierig ist, werden derartige GEO-Records (vom Gleichungstyp) nicht visualisiert. GEORecords ohne abhängige Parameter bezeichnet man als Records vom konstruktiven Typ. Eine „Dynamisierung“ der Visualisierungen durch direkte Mausmanipulation ist nicht implementiert, da hierfür weitergehendes Event-Handling erforderlich ist. Parameterwerte einer Visualisierung können nur über ein Dialogfeld geändert werden.

2. Produktübersicht

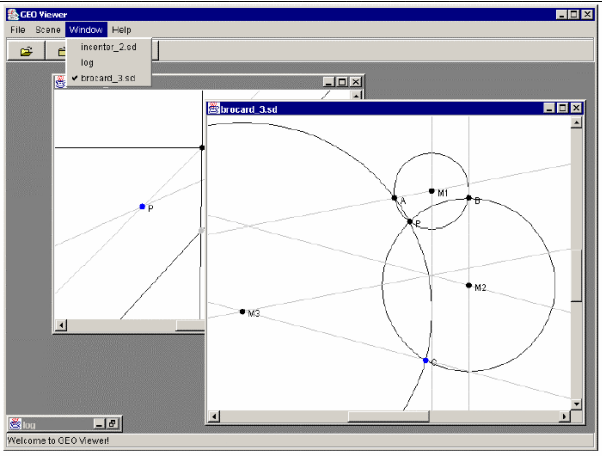
Die Applikation wird durch den Kommandozeilen-Aufruf
`java geometry.GeoViewer`
im Wurzelverzeichnis der Pakethierarchie gestartet. Während der *Initialisierung* wird die Verbindung zu Maple hergestellt und im Erfolgsfall das Hauptfenster aufgebaut. Das *Hauptfenster* enthält am oberen Rand einen *Menü- und Toolbalken*, am unteren Rand einen *Statusbalken* und in der Mitte eine *Desktop-Fläche*, in der im weiteren Verlauf die verschiedenen Zeichenflächen sowie ein spezielles Fenster für Fehlermeldungen geöffnet werden können.
Über den Menüeintrag *File* oder das linke Icon wird das *Laden eines GEO-Records* gestartet.



Handelt es sich um ein konstruktives Schema, so wird die zugehörige Visualisierung generiert und in einer neuen Zeichenfläche im Desktopbereich angezeigt und aktiviert. Die Parameterwerte werden dabei zunächst mit Zufallszahlen initialisiert.

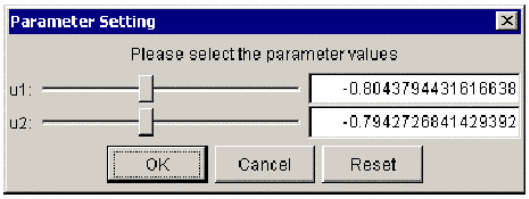


Zwischen verschiedenen Fenstern der Desktopfläche kann mit der Maus hin- und hergeschaltet werden. Alternativ kann das aktuelle Fenster über ein RadioButton- Menü eingestellt werden.



Wird als Fenster eine Zeichenfläche ausgewählt (aktive Zeichenfläche), so werden für diese die Menüpunkte und Icons „Parameterwerte ändern“ und „Zeichenfläche schließen“ aktiviert.

Die Parameterwerte können für die aktive Zeichenfläche über einen Menüeintrag geändert werden, der ein Dialogfenster mit Schiebereglern öffnet. Danach werden die Koordinaten der einzelnen Objekte von Maple neu berechnet und die Visualisierung in der aktiven Zeichenfläche neu dargestellt.



Über einen weiteren Menüpunkt, Icon oder einen speziellen Knopf am Fensterrand wird die aktive Zeichenfläche geschlossen. Die Applikation wird über den Exit- Eintrag im File -Menü beendet.

3. Grundsätzliche Designentscheidungen:

- Objekt-orientierte Konzepte
- Kommunikation mit CAS (Maple bzw. MuPad)
- Symbolische Berechnungen
- Verwenden einer eigenen Beschreibungssprache
- Dynamische Steuerung
    - Parametereingaben
- MVC
- MDI
- Modularer Aufbau, Benutzung von fertigen Bausteinen

Objekt-orientierte Konzepte
Das Design der Software ist auf objekt-orientierten Konzepten aufgebaut.
Verwandte Abstraktionskonstrukte werden in Pakete verteilt und in logisch aufgebaute Klassen bzw. Interfaces integriert. Einzelne Funktionalitäten sind in den Klassen integriert (Datenkapselung).
Als grundsätzliches Werkzeug beim Spezialisieren (Eigenschaften erweitern) bzw. Generalisieren (Eigenschaften zusammenfassen) einzelner Klassen wurde Interface-Bildung und/ oder Vererbung eingesetzt.

Kommunikation mit CAS (Maple bzw. MuPad)
Symbolische Berechnungen
Algebraische Berechnungen werden mithilfe CAS- Klassen an Maple bzw. MuPad delegiert.

Beschreibungssprache
Dynamische Steuerung
Parametereingaben
Die Anwendung liest eine Datei mit Konstruktionsbeschreibung, parst diese, berechnet sie symbolisch und zeichnet sie schließlich auf der Arbeitsfläche. Dynamik wird damit erreicht, dass der Benutzer neue Parameter für jeweilige Funktion angibt.

MVC
Interaktiver, dialogbasierter Betrieb wird mithilfe der GUI realisiert. Dabei verwendet man das übliche in Java- Swing MVC- Konzept. Die wichtigste, von der Steuerung her, Klasse der Anwendung, SceneController [Paket: controller], übernimmt die Steuerung (Botschaften, Informationsfluss) innerhalb der Anwendung.

Modulare Aufbau, Benutzung von fertigen Bausteinen
Der Aufbau der Anwendung ist modular.
Einzelne Module bilden:

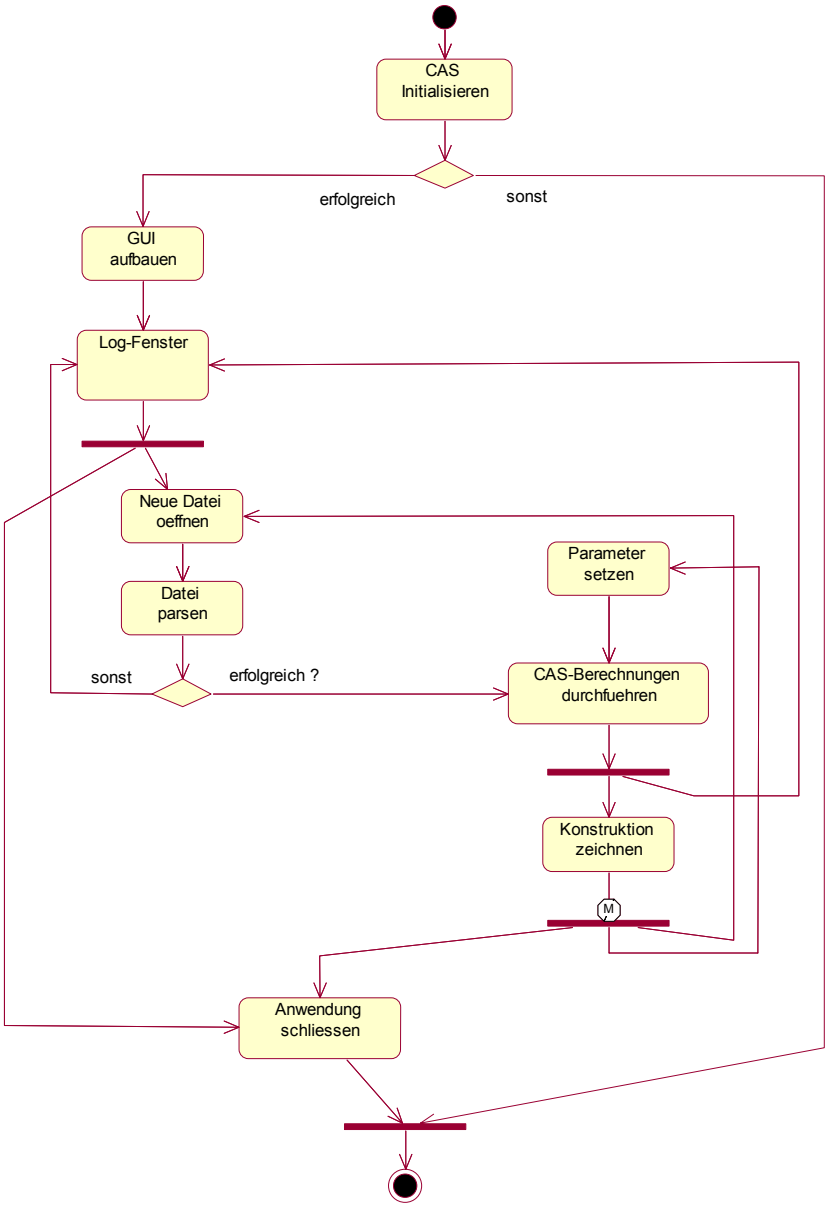
- Parser, CUP (http://www.cs.princeton.edu/~appel/modern/java/CUP/)
- noch nicht vollständig realisiertes Testsystem

Modulare Designplanung erlaubt es schon gefertigte Java- Bausteine zu verwenden und diese mit eigenen Funktionalitäten erweitern. Dies ist im Falle Parsergenerierung sehr spürbar.

MDI
Gleichzeitig können mehrere Fenster geöffnet werden, was bedeutet, dass es innerhalb der Anwendung ständig überprüft wird, welches Fenster aktiv ist bzw. welches Modell von SceneController betreut wird.

Aktivitätsdiagramm:

Grundsätzliche Arbeitsweise der Anwendung kann man von dem folgenden Diagramm ablesen:



Bemerkung 1:

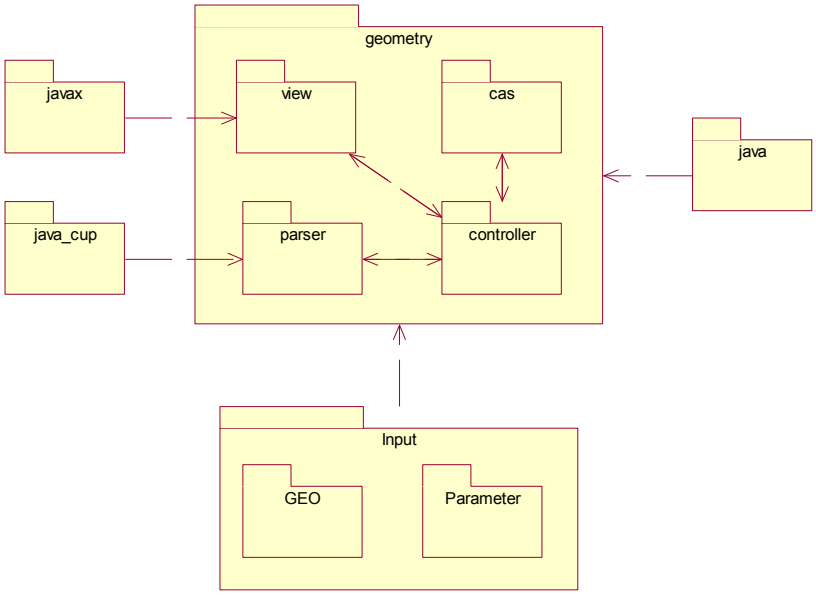
Im 3. Punkt der Designbeschreibung verzichten wir auf genaue Beschreibungen der einzelne Klasse und Pakete. Diese finden im 4. Punkt der Designbeschreibung statt.

Bemerkung 2:

Alle Diagramme, die in der Beschreibung verwendet wurden kann man in der Rose- Datei „Geo09_Diagramme“ finden

4 Paket- und Klassenstruktur

Paketdiagramm:

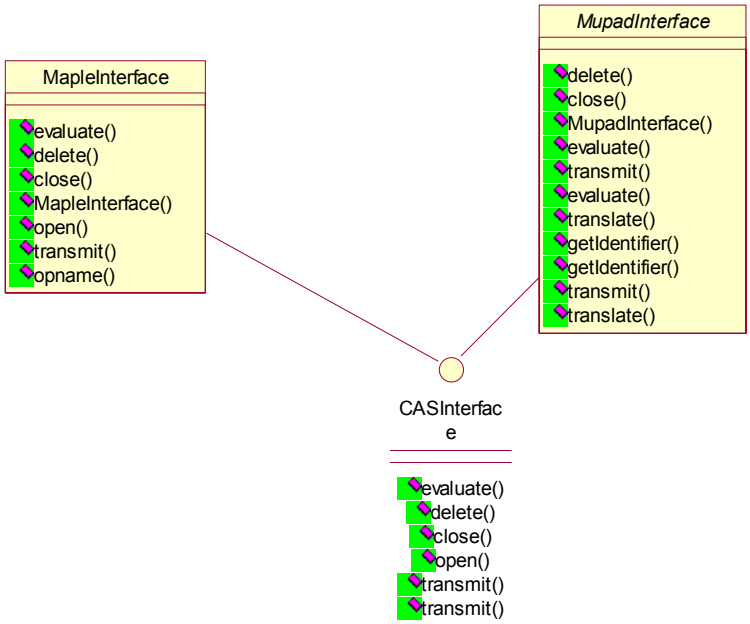


Nach dem Reverse Engineering haben wir festgestellt, dass es 4 Hauptpakete gibt. Das sind javax, java_cup, java und geometry. Das Paket geometry enthält wiederum 4 Pakete: view, cas, parser und controller.
Das Paket **parser** benutzt das Paket **java_cup** als Runtime und Parsergenerator und arbeitet mit **controller**.
Das Paket **view** benutzt das Paket **javax** für die GUI und kommuniziert mit **controller**.
Das Paket **controller** kommuniziert mit **parser**, **view** und **cas**.
Das Paket **Input** enthält Pakete GEO und Parameter und stellt die Kommunikation mit der Außenwelt (Benutzer, Eingabe, Parameter) dar.
Im Paket GEO liegen die Konfigurationsdateien mit der Erweiterung .sd, die die Beschreibung der zu zeichnenden Konstruktion in einer XML-mäßigen Sprache enthalten.
Das Paket Parameter stellt die Konstruktionen dar, die in den Hauptspeicher geladen werden. Die Parametereingabe des Benutzers wird während des Programmablaufs in diesem Paket abgelegt, aber sie dürfen nicht als .sd-Datei gespeichert werden.

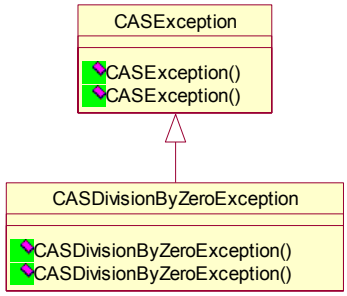
Paket CAS:

Dieses Paket enthält Klassen, die für die Kommunikation mit dem CAS (wahlweise Maple oder Mupad) zuständig sind.

- Die Klasse **CASInterface** ist eine abstrakte Klasse, die von den Klassen **MapleInterface** und **MupadInterface** implementiert wird.
- **MupadInterface** implementiert das Interface zu Mupad zur numerischen Auswertung von Ausdrücken.
- **MapleInterface** implementiert das Interface zu Maple zur numerischen Auswertung von Ausdrücken.



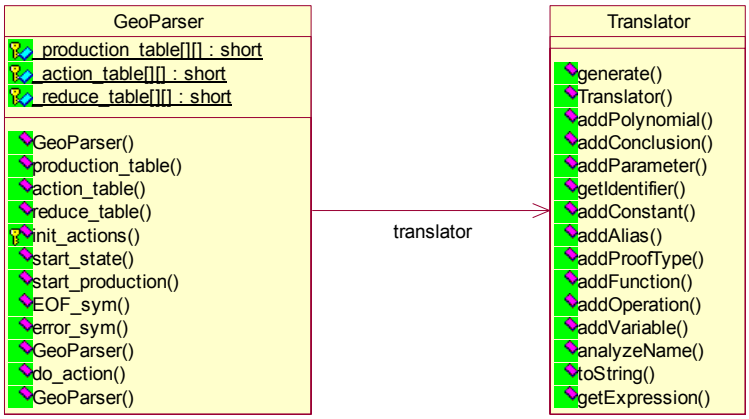
- Die Klasse **CASException** ist abgeleitet von IOException und behandelt Ausnahmen, die im CAS auftreten.
- Die Klasse **CASDivisionByZeroException** ist von CASException abgeleitet, sie behandelt die Ausnahmen bei der Division durch Null im CAS.



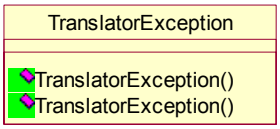
Paket PARSER:

Dieses Paket enthält Klassen, die zum Parsen der Eingabedatei verantwortlich sind.

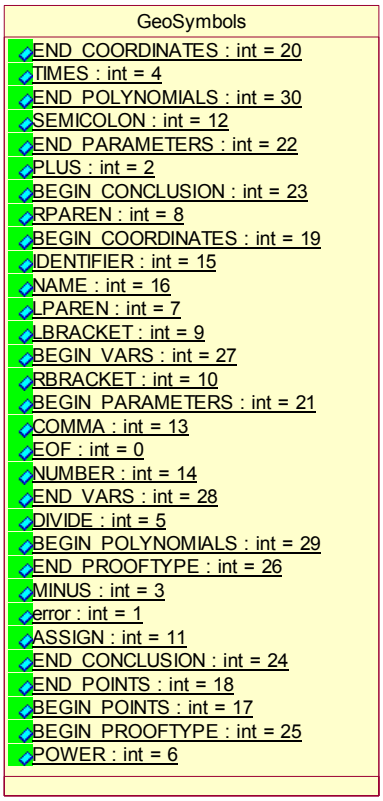
- Die Klasse **GeoParser** dient zum Parsen der Eingabedatei. Dieser Parser wurde mit Hilfe von CUP generiert.
- Die Klasse **Translator** stellt die Funktionen zur Verfügung, die vom GeoParser zur Konstruktion der Szene benutzt werden.



- Die Klasse **TranslatorException** ist abgeleitet von der Klasse Exception und behandelt die Ausnahmen, die beim Übersetzungsprozess auftreten.



- Die Klasse **GeoSymbols** enthält Konstanten, die im Parsing-Prozess benutzt werden.

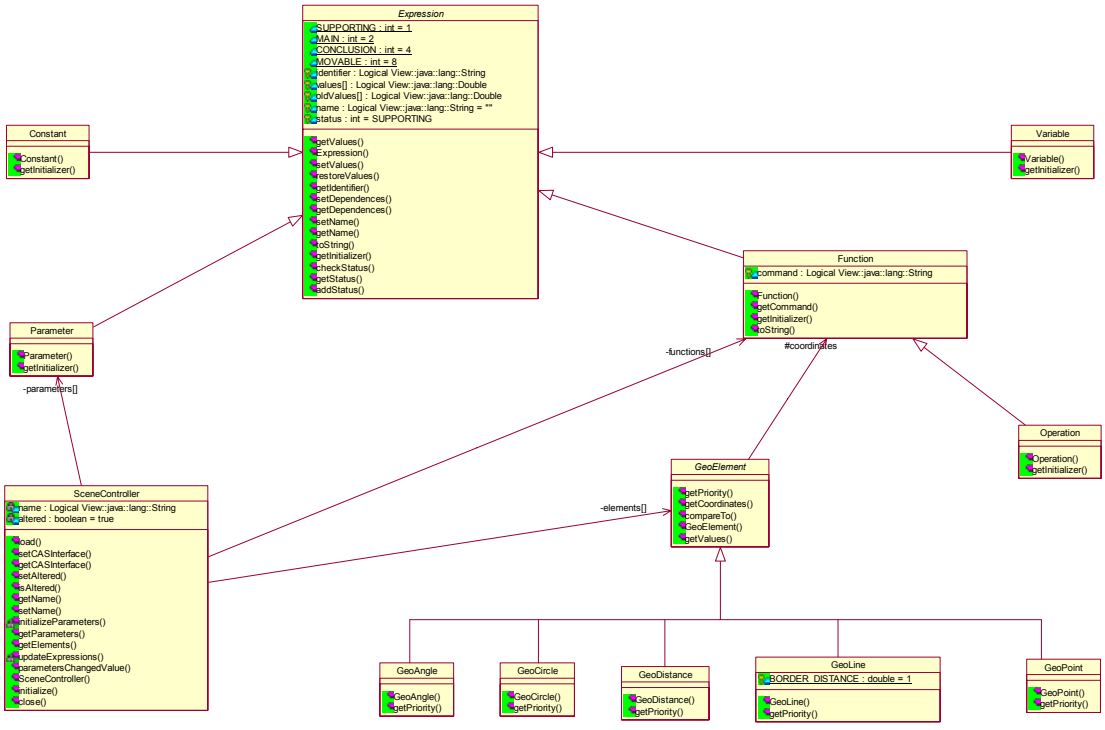


Das Paket controller

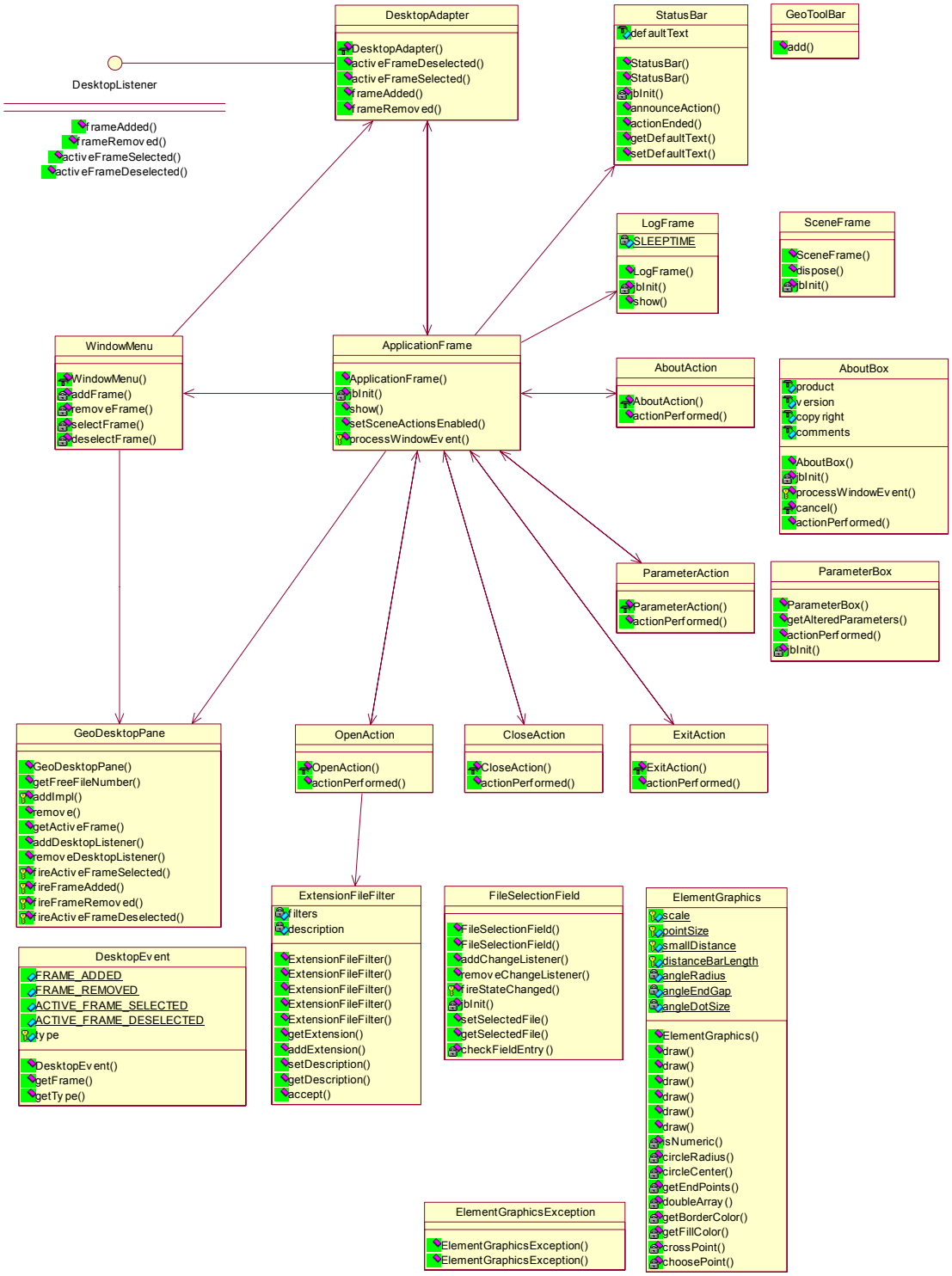
Klassenübersicht	
Constant	Klasse die konstante Ausdrücke repräsentiert.
Expression	Klasse die GEO Ausdrücke modelliert.
Function	Klasse die GEO Ausdrücke und allgemeine algebraische Funktionen modelliert.
GeoAngle	Klasse die Winkel Objekte repräsentiert.
GeoCircle	Repräsentiert Kreiselementen.
GeoDistance	Repräsentiert Entfernung.
GeoElement	Superklasse für geometrische Elemente.
GeoLine	Linienrepräsentation.
GeoPoint	Punkte.
Operation	Klasse zur Modellierung allgemeiner algebraischen Operationen.
Parameter	Klasse zur Modellierung von Parametern.
SceneController	Klasse die Methoden zur Kontrolle einer Ansicht bereitstellt.

- **Expression:** Sie ist eine abstrakte Superklasse für die verschiedenen Ausdrücke.
  Mit Hilfe des Konstruktors der Klasse wird ein Expression - Objekt erzeugt. Dieses wird mit dem Stringparameter, das dem Konstruktor übergeben wird, innerhalb von CAS identifiziert.
- **Expression und ihre Unterklassen** enthalten Felder zur Speicherung von Parameterwerten des Ausdrucks (**values[]**, **oldValues[]**), ein Feld mit den Ausdrücken von denen der Ausdruck selbst abhängt (Expression **dependencies[]**), statische Attribute zur Identifikation des Status des Ausdrucks. (ob es ein Hauptausdruck ist oder ehe eine Hilfsausdruck ist usw. **MAIN**, **SUPPORTING**, **CONCLUSION**, **MOVABLE**) und die entsprechenden Methoden sie zu manipulieren. Es geht um in Expression implementierte Methoden.
- **Abstrakte Methode(n):** **public abstract** String getInitializer()
  Diese Methode wird benutzt, um das Stringparameter zu erhalten, mit denen der Ausdruck initialisiert wurde. Diese Zeichenkette wird von dem CAS bewertet. Jede Unterklasse bringt ihre eigene Implementierung.
- **Function:**
  Konstruktor nimmt als Parameter zwei Zeichenketten, ein Feld von Expression Objekte.
  Erste Zeichenkette –Die Zeichenkette mit der, der Ausdruck in CAS identifiziert wird, Zweite Zeichenkette - Operationszeichen oder ein Funktionskopf
  Feld von Expression Objekten – Feld von Objekten die selbst Ausdrücke sind. Das Function - Objekt hängt von denen ab.
- **Zusätzliche Attribute (spezifisch für Function)**
  commands eine Attribut die den Funktionskopf oder Operationszeichen enthält.
  GeoElement: Objekte der Unterklassen von GeoElement werden mit einem Konstruktor erzeugt, dessen Parameter ein Objekt der Klasse Funktion ist. Dieses Parameter wird mit dem Namen **coordinates** bezeichnet was vermuten lässt, dass die Koordinaten wo das Objekt auf dem Zeichenfeld erscheinen muss, mit einem Ausdruck ermittelt wird.
- Die abstrakte Klasse GeoElement implementiert die Schnittstelle Comparable um den Geometrischen Elementen, die von ihren Unterklassen repräsentiert wird, Priorität zuzuweisen. Die Priorität von Objekten wird benutzt um die Reihenfolge zu ermitteln in denen die Objekte gezeichnet werden. Jede Unterklasse muss eine Methode zum Abfragen der Priorität des Objekts implementieren.

- **Abstrakte Methode(n):** public abstract int getPriority() Für diese Methode bringt jede Unterklasse ihre eigene Implementierung.
  - **SceneContoller:**
    Stellt die Geometrische Struktur dar, das auf dem Zeichenfeld erscheint. Die Werte und Parameter werden in den unten beschriebenen Attribute gespeichert.
  - Wichtige Attribute:
    ```
    private static CASInterface cas  - Referenz auf die CAS Schnittstelle
    private Parameter parameters[] – Feld von Szene - Parametern
    private GeoElement elements[] -  Feld von allen GEO Elementen
    private Function functions[] - Feld von „Function“ - en – Ausdrücke von denen die Geometrische Objekte abhängen.
    private boolean altered – „true“ wenn die Parameter kürzlich geändert worden sind. Die Objekte sind noch nicht gezeichnet.
    ```
- **Konstruktor:** Nimmt als Parameter Feld von Parameter Objekten, Function Objekten, GeoElement Objekten und eine boolean Variable namens initializeParameters. Konstruktor prüft erst ob eine CAS Schnittstelle gesetzt worden ist. Wenn ja dann speichert er die Parameter in die entsprechende Instanzattribute andernfalls wirft ein Error Objekt (Fehler).
- **Wichtige Methoden:**
  public static SceneController load(File file, **int** fileNumber).
  Diese Methode lädt die GEO Datei in die Applikation . (die Datei muss SymbolicData Format haben) Die Datei wird mit fileNumber nummeriert. Die Datei wird vermutlich von innerhalb der Methode erzeugten Objekten Translator und GeoParser (die Klassen aus cas Paket) in brauchbare Form umgewandelt und mit Hilfe der nützlichen Informationen wie Parameterwerte und benötigten Ausdrucken wird ein Objekt der Klasse SceneController erzeugt die eine Gruppe von Geometrischen Elementen – eine Konstruktion – verwaltet.



Das Paket geometry.view



- **AboutAction:** Öffnet ein Fenster mit den Eigenschaften des Programms (Name der Applikation, Version, Autor etc).
- **AboutBox:** Implementiert die Option „about“, die später in „Menu/Help“ erscheint.
- **ApplicationFrame:** Erstellt das „Main Frame“ für die Applikation. Implementiert die wichtigsten Funktionen.

- **CloseAction:** Schließt das aktive Konstruktionsfenster.
- **DesktopEvent:** Merkt sich, falls eine Aktion in dem aktiven Frame ausgeführt war.
- **DesktopListener:** Interface. „Listener“ für das Geschehen auf dem Desktop.
- **ElementGraphics:** Implementiert die geometrische Darstellung geometrischer Objekte.
- **ElementGraphicsException:** Implementiert das Abfangen der „Exceptions“, falls ein geometrisches Objekt auf irgendwelchen Grund nicht dargestellt werden kann.
- **ExitAction:** Schließt die Anwendung.
- **ExtensionFileFilter:** Klasse fürs filtern der Dateiextensionen. Es besteht die Möglichkeit Dateien bestimmtes Typs zu sehen (alle oder .sd Typ).
- **FileSelectionField:** Klasse fürs Wählen der zu öffnenden Datei. Ermöglicht das Wählen einer Datei aus der (vom ExtentionFileFilter) gefilterten Liste oder durch Tastatureingabe.
- **GeoDesktopPane:** Erweitert javax.swing.JDesktopPane
- **GeoToolBar:** Implementiert die „Toolbar“ in dem Applikationsfenster. Ermöglicht Abkürzungen zu den wichtigsten Funktionen über Buttons.
- **LogFrame:** Implementiert das Log – Fenster. Beschäftigt sich mit dem Ausgeben der aufgetauchten Fehler in dem Log – Fenster.
- **OpenAction:** Öffnet das Fenster zur Auswahl der zu öffnenden Datei.
- **ParameterAction:** Öffnet das Fenster mit den Parametern der Elemente der Konstruktion.
- **ParameterBox:** Implementiert die Option „parameter“, die später in „Menu/Scene“ zu sehen ist.
- **SceneFrame:** Implementiert das Frame für die Applikation.
- **StatusBar:** Implementiert einen Statusbalken, wo man den Status einer Aktion sowie eine kurze Beschreibung dazu ablesen kann.
- **WindowMenu:** Implementiert das Menu in dem Applikationsfenster.