Funktionen mit Mathematik-Display

Die Darstellung komplexer mathematischer Formeln ist mit wenigen Methoden realisierbar. In vielen Systemen wird die Darstellung von der Funktionalität getrennt. Hier wird Funktion und Darstellung vereint. Dadurch wird die Bearbeitung vom etablierten Systemen zur Darstellung mathematischer Formeln erleichtert. Den ersten Schritt macht das Interface zu MathML, das ebenfalls im Paket enthalten ist.
Es folgt eine Liste mit den vorhandenen Methoden, die für den Aufbau zur Verfügung stehen.

DispObject

Die Basisklasse aller weiteren Objekte. Sie stellt zwar nur ein Zeichen dar, betrachtet dafür aber auch jedes abgeleitete und komplexere Objekt als Zeichen. Diese Klasse übernimmt alle erforderlichen Verkettungen der zusammengesetzten Objekte. Normalerweise wird ein Baum aufgebaut, der als einzelnes Zeichen in komplexere Objekte eingefügt wird. Deshalb existiert für dieses Objekt auch auch keine explizite Creator-Methode.
Um ein Zeichen (Character) als DispObject zu definieren sind mehrere Schritte erforderlich:

```
DispObject obj = new DispObject();
DispObject.characterData( obj, 'x');
```

Das Zeichen 'x' wurde als DispObject mit dem aktuellen Zeichensatz und den aktuellen Farben aufgebaut.
Werden mehrere DispObjects zu einem einzigen zusammengefasst, besteht das vereinende Objekt aus eine Liste. In diesem obersten Objekt existieren keine Daten für Zeichen. Es wird jedoch ein Leerzeichen (Space, 0x20) abgelegt.
Die wesentlichen Vorgaben für die Darstellung werden ebenfalls über diese Klasse gemacht. Dafür sind folgende Methoden verantwortlich:

```
DispObject.setDefaultColor( Color color);
DispObject.setDefaultFont( Font font);
DispObject.setDefaultSpacing();
```

Die letzte Methode setzt die Basis für

> Zeichenabstand in Strings (1/24 der Zeichenhöhe)
> Spaltenabstand zwischen Spalte in Tabellen (doppelter Zeichenabstabnd)
> Abstand zwischen Elementen in einer Disp-Row-Zeile (vierfacher Zeichenabstand)
> Faktor zur Verkleinerung der Darstellung von Indizes und Exponenten (58%)
> Faktor zur Verschiebung der Darstellung von Indizes und Exponenten (30%)

Alle Abstände sind also nur abhängig von Zeichenbreite oder -höhe. So ist eine sehr flexible Grundlage für die Entwicklung eigener Erweiterungen gelegt.

DispString

Das einfachste zusammengesetzte Objekt. Ab jetzt haben alle weiteren Objekte mindestens eine Creator-Methode.

Ein "leerer" String:

```
DispString()
```

String mit den aktuell vorhandenen Attributen:

```
DispString( String s)
```

String mit bestimmter Farbe:

```
DispString( String s, Color c)
```

String mit bestimmtem FontStyle:

```
DispString( String s, int sty)
```

String mit definierter Farbe und Style:

```
DispString( String s, Color c, int sty)
```

Dieses Objekt sollte immer Verwendung finden, wenn Bezeichner (z.B. Funktionsnamen) oder Operatoren (z.B. +) benötigt werden. Natürlich kann ein DispString eine ganze Formel enthalten, aber nur als Text, nicht als mathematischer Ausdruck.
DispString-Objekte sollten nur für Zeichenfolgen benutzt werden und nicht für Formeln.

DispRow

Das einfachste Objekt zum Aufbau von Formeln. Es dient prinzipiell zur zeilenorientierten Darstellung unterschiedlicher Objekte. Dieses Objekt hat ebenfalls mehrere Creator-Methoden.

Eine leere Zeile:

```
DispRow()
```

Zeile mit mehreren Objekten, die als Liste vorliegen:

```
DispRow( LinkedList list)
```

Zeile mit mehreren Objekten und einem besonderen Abstand zwischen ihnen:

```
DispRow( LinkedList list, double space)
```

Zeile mit mehreren Objekten, die als Array vorliegen:

```
DispRow( DispRow[] arr)
```

Zeile mit Objektarray und einem besonderen Abstand zwischen den Elementen:

```
DispRow( DispRow[] arr, double space)
```

Objekte dieser Klasse dienen zur Zusammenfassung beliebiger anderer Objekte. Besonders angenehm ist die automatische Ausrichtung von Bruch- Exponential- und Matrix-Termen.

DispFence

Klammern sind etwas schwieriger darzustellen. Bereits die Klammerung von Brüchen scheitert oft in der Darstellung, wenn im Nenner wieder ein Bruch steht. Die Ausrichtung innerhalb der Zeile an der bestehenden Grundlinie ist das Problem. Deshalb wurde für diese Aufgabe eine eigene Klasse erstellt.
Die Bezeichnung "Fence" verdeutlicht, dass ncht nur die Zeichen '(' und ')' Verwendung finden, sondern jedes Zeichen für die "Einzäunung" benutzt werden kann.

Ein leerer Klammerausdruck:

```
DispFence()
```

Klammerausdruck allgemein:

```
DispFence( char open, DispObject obj, char close)
```

Es ist möglich die Klammezeichen einzeln zu benutzen. Soll die öffnende und/oder schließende Klammer fehlen, wird einfach das Leerzeichen (' ' 0x20) angegeben.
Die Klammern werden automatisch gedehnt und an der Grundlinie ausgerichtet. So ist die Klammerung von Objekten mit beliebiger Komplexität über beliebig viele Ebenen möglich.

DispFract

Zähler und Nenner sind beliebige DispObjects. Der Bruchstrich hat die Länge des breiteren Ausdrucks und wird links und rechts um eine halbe Zeichenbreite velängert.

Leerer Bruch:

```
DispFract()
```

Bruch allgemein:

```
DispFract( DispObject num, DispObject den)
```

Die vertikale Ausrichtung erfolgt weitgehend anhand des Bruchstrichs. Wenn Nenner oder Zähler selbst wieder Brüche sind, ist trotzdem die Position des *aktuellen* Bruchstrichs relevant.

DispSqrt

Ein beliebiges DispObject wird mit einem Wurzelzeichen versehen.

Leerer Bruch:

```
DispSqrt()
```

Bruch allgemein:

```
DispSqrt( DispObject obj)
```

Die erforderliche Dehnung der Wurzelzeichens erfolgt wie bei den Klammerzeichen in der Methode DispFract.

Hier wird die Grundlage für ein System vorgestellt, mit dem zunächst nur die Darstellung mathematischer Ausdrücke möglich ist. Eine Übersetzung nach DispML, LATEX und OpenOffice ist enthalten. Um auch die Ausgabe selbst unabhängig von Programmiersprachen zu halten, wird die Ausgabe auch über reine Glyphen im SVG möglich sein.

Der Kernel

Die elementaren Methoden (Funktionen) des Systems werden hier beschrieben. Für eine Arbeit im Open-Source Bereich ist die Vorstellung der einzelnen Verhaltensweisen sehr wichtig. So wird

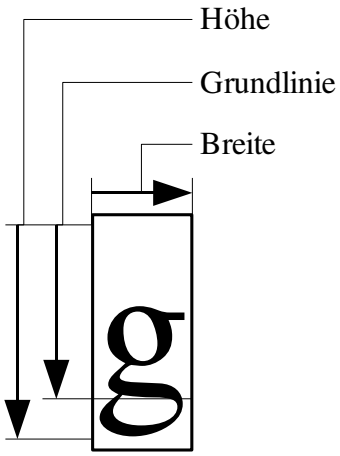
doppelte Arbeit vermieden und der Einstieg in das System erleichtert.

DispObject

Diese Klasse ist die Basisklasse aller Disp-Objekte des Systems. Hier werden Zeichen (Character) und Darstellung koordiniert. Weil jedes Disp-Objekt von dieser Klasse abgeleitet ist, betrachtet das System auch jedes Objekt zunächst als einzelnes Zeichen. Die differenzierte Betrachtung der Interna dieses „Zeichens“ erfolgt rekursiv und endet bei den Linien der grafischen Ausgabe (Shape).

Eine Instanz von DispObject hat entweder ein Zeichen zur Darstellung oder eine Liste mit DispObject-Instanzen aus denen ein entsprechendes Objekt gebildet wird. Es ist wichtig, diese „*entweder-oder*“-Bedingung einzuhalten.

In jedem Fall besitzt das Objekt einen Rahmen (Bounds) und eine Grundlinie (Baseline). Diese beiden Eigenschaften hängen eng zusammen.



Die Pfeile geben die Zählrichtung an. Aus dieser wird deutlich, dass der Bezugspunkt jedes DispObjects die linke, obere Ecke des umgebenden Rechtecks ist. In allen Textsystemen bildet die Grundlinie diesen Bezug.

Es gibt natürlich noch viele weitere Eigenschaften, wie Zeichensatz, Farbe usw. Sie sind aber in der API-Dokumentation hinreichend beschrieben. Es soll aber auf einige Methoden eingegangen werden, deren Anwesenheit nicht unmittelbar gegeben ist.

characterData

Diese Methode muss auf Objekte angewendet werden, die nur ein Zeichen verkörpern. Hier werden die wesentlichen Daten eines Zeichens für die Darstellung ermittelt. Zu diesen Daten gehört auch die Position, die sich auf die Bounds (umgebendes Rechteck) bezieht. Eine weitere Abweisung zu *normaler* Textverarbeitung besteht im fehlenden „Nachspann“. Die Breite des Bounds entspricht genau der (sichtbaren) Zeichenbreite.

compress

Bei manchen (zusammengesetzten) DispObjekten müssen einzelne Zeichen nur in eigenen Ausdehnung vorhanden sein. Die Zeichenhöhe, die ja normalerweise vom Zeichensatz abhängig ist, wird also entfernt. Bei diesen (komprimierten) Zeichen ist die Grundlinie 0 (Null).

scale

Manchmal müssen Zeichen über einen Bereich ausgedehnt werden. Das ist oft bei Klammern und Brüchen der Fall. Für diesen Zweck ist diese Methode vorhanden. Sie wird normalerweise auf

komprimierte Zeichen angewendet, die dann in horizontaler und/oder vertikaler Richtung über einen bestimmten Bereich gedehnt werden.

print

Die textuelle Ausgabe ist mit dieser Methode möglich. Ein DispObject wird strukturiert als Text ausgegeben. Weil jedes abgeleitete DispObjekt eine spezielle Ausgabe verlangt, ist in derartigen Objekten auch die Methode „*printSpecial*“ vorhanden. Weil aber auch diese Methode statisch ist, muss sie über die Klasse ermittelt werden. Deshalb darf auch in der Klasse DispObject keine Methode namens „*printSpecial*“ vorhanden sein.

DispCol

Klasse zum Anordnen von DispObjects in einer Spalten. Diese Klasse dient auch dem Aufbau von Vektoren und/oder Matrizen.

Die Ausrichtung der enthaltenen Elemente muss stets angegeben werden. Die nötigen Konstanten sind in der Klasse „*Align*“ und tragen die Bezeichnungen LEFT, CENTER, RIGHT.

Die Elemente der Spalte werden als Array übergeben. Damit wird ein erster Schritt in Richtung Dispematik gemacht, denn Arrays sind in ihrer Größe normalerweise nicht änderbar; eine Spalte in Matrizen schon gar nicht.

DispFence

Klasse zur Bereitstellung eines geklammerten Ausdrucks. Als Klammer-Zeichen ist jedes darstellbare Zeichen erlaubt. Die Ausdehnung der Klammern auf die erforderliche Höhe erfolgt automatisch. Wenn eine Klammer nicht erwünscht ist, muss ein Leerzeichen verwendet werden. Dieses Leerzeichen ist wichtig für die Eindeutigkeit der Methode „*printSpecial*“, bzw. für die Programme, die das Ergebnis dieser Methode weiter auswerten.

DispFract

Klasse zum Aufbau von Bruchermen. Als Zähler und Nenner wird jedes DispObject akzeptiert. Sollte es sich bei einem der beiden wieder um einen Bruch handeln, erfolgt *keine* automatische Verkleinerung des Zeichensatzes. Für diese veraltete Darstellung muss das betroffene Objekt individuell angepasst werden.

DispMatr

Klasse zum Aufbau von Matrix-Objekten. Es handelt sich eigentlich um eine sekundäre Klasse. Das Attribut wurde gewählt, weil sowohl Aufbau als auch textuelle Ausgabe der Klasse „*DispFence*“ entsprechen.

Der Aufbau erfolgt tabellenorientiert. Es wird also ein Objekt der Klasse „*DispTab*“ übergeben. So kann eine Matrix als eine geklammerte Tabelle angesehen werden. Damit sind Erweiterungen auf Determinanten und Tensoren stark vereinfacht.

DispMultiScript

Klasse zur Bereitstellung von Ausdrücken mit Skripten. Links- und rechtsseitige Indizierung und "Potenzierung" kann mit dieser Klasse erfolgen. Als Ausdruck und Skriptum ist jedes DispObject erlaubt oder *null* erlaubt. Die Verkleinerung des Zeichensatzes der Skripte erfolgt automatisch, entsprechend der Einstellungen.

DispOver

Diese Klasse dient dazu, einen Ausdruck zu „überschreiben“. Der klassische Fall ist wohl ein Binomialkoeffizient (ohne Klammern). Es könnte zwar auch die Klasse „DispCol“ verwendet werden, aber in DispML ist dieses Objekt eben vorhanden. Ob eine Umstellung auf „DispCol“ erfolgt steht noch nicht fest, ist aber unwahrscheinlich. Der zu betreibende Aufwand zur Bestimmung der Grundlinie wäre zu groß.

DispOverline

Klasse zum Überstreiche eines Ausdrucks. Ein beliebiger Ausdruck wird mit einem beliebigen, darstellbaren Zeichen überstrichen. Das Zeichen wird vorher um 90° gedreht und erst dann in seiner komprimierten Form aus die erforderliche Länge gedehnt.

DispRow

Diese Klasse ist das am häufigsten eingesetzte Mittel zur Zusammenfassung von DispObjects. Die einzelnen Elemente werden entsprechend ihrer Grundlinie auf gleiche Höhe gebracht und erst dann von einem neuen Rechteck umgeben. In LATEX und OpenOffice entsprechen die geschweiften Klammern dieser Klasse.

DispSqrt

Hier wird ein Ausdruck unter eine Wurzel gestellt. Das Problem bei dieser Klasse ist die Vermeidung von „zeichnenden“ Methoden. Der lange Strich am Wurzelzeichen, über dem Ausdruck muss durch ein entsprechend transformiertes Zeichen gebildet werden. Das ist notwendig, um für alle DispObjects die Glyphen aus dem Zeichensatz zu bilden. Es könnte ein Shape erzeugt werden, aber der Aufwand ist nicht zu unterschätzen.

DispString

Mit dieser Klasse werden zusammenhängende Zeichenketten aufgebaut. Objekte dieser Klasse bestehen aus einzelnen DispObject-Instanzen, die jeweils ein Zeichen enthalten. Im Gegensatz zu allen anderen Klassen hat diese die Möglichkeit Zeichensatz und/oder -farbe für alle enthaltenen Objekte festzulegen. Diese Möglichkeit besteht bereits bei der Instanzierung mit dem entsprechenden Creator.

Als Abstand zwischen den einzelnen Zeichen wird die Breite eines komprimierten Punktes „.“ gewählt. Der Vorteil bei dieser simplen Bestimmung ist, dass immer ein Abstand vorhanden ist (punkt ist darstellbares Zeichen) und der Platz für einen Cursor reicht.

DispTab

Tabellen werden aus einem zweidimensionalen Array aufgebaut. Der ursprüngliche „Verband“ aus Spalten und Zeilen wird aufgelöst und in eine lineare Liste überführt. Damit die Organisation aus Spalten und Zeilen nicht total zerstört wird, sind entsprechende Variablen als Properties vorhanden.

Die Ausrichtung der Zellen erfolgt über die Konstanten der Klasse *Align*. Die Abstände zwischen den Zellen gelten erst nach erfolgter Ausrichtung und beziehen sich dann auf die jeweiligen Maximalwerte von Breite und Höhe.

DispUnder

Diese Klasse dient dazu, einen Ausdruck zu „unterschreiben“. Der klassische Fall ist wohl wieder

ein Binomialkoeffizient (ohne Klammern). Es könnte zwar auch die Klasse „DispCol“ verwendet werden, aber in DispML ist dieses Objekt eben vorhanden. Ob eine Umstellung auf „DispCol“ erfolgt steht noch nicht fest, ist aber unwahrscheinlich. Der zu betreibende Aufwand zur Bestimmung der Grundlinie wäre zu groß.

DispUnderline

Klasse zum Unterstreiche eines Ausdrucks. Ein beliebiger Ausdruck wird mit einem beliebigen, darstellbaren Zeichen unterstrichen. Das Zeichen wird vorher um 90° gedreht und erst dann in seiner komprimierten Form aus die erforderliche Länge gedehnt.

DispUnderover

Diese Klasse dient dazu, einen Ausdruck zu „unterschreiben“ und „überschreiben“. Normalerweise wird diese Klasse benutzt, um Objekte mit „Laufanweisungen“ (*DispLoopOp*) bereitzustellen.

Wieder scheint sich eine MatCol-Konstruktion anzubieten, aber es wäre viel Aufwand nötig, die Grundlinie zu bestimmen.