The JavaView Guide

March 16, 2015

Contents

1 Installation

1.1 Software Version List

Since the required software is frequently replaced by newer versions we define path variables that are used in the installation instructions below.

[JAVAJDK] Latest version (March 2015) is Java Development Kit 8. You can download it on `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`

[JAVAJDKVERS] The jdk version identifier. For JDK 8 this will be *jdk1.8.xxx* where the *xxx* are numbers corresponding to minor subversions.

[JAVAJDKPATH] Depending on the current Java JDK version, this should be something like `C:/ProgramFiles/Java/jdk1.8.xxx`

[JAVAJRELIBPATH] The *lib/ext* directory of the Java runtime (JRE) that comes with the JDK. This is [JAVAJDKPATH]`/jre/lib/ext`, for example `C:/ProgramFiles/Java/jdk1.8.xxx/jre/lib/ext`

[JAVAVIEW] Go to `http://www.javaview.de/download/beta` to download the latest version of JavaView or to `http://www.javaview.de/download` for the latest stable release. If you want to use the installer for Windows(exe)/Linux(rpm), simply execute it and install JavaView. Otherwise, download *javaviewFull.zip* resp. *javaview_[version_number].zip* and extract it to any folder (named `%javaview%`).

[ECLIPSE] Latest version (March 2015) is Eclipse Luna, available here: `http://www.eclipse.org/downloads/`. Download *Eclipse IDE for Java Developers* and install/extract it. Note: Do not download Eclipse Standard but Eclipse IDE for Java Developers, currently at `http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr2`

[ECLIPSEVERS] This is the name of the current Eclipse release, usually the name of a moon or some other astronomical name. For example if you use Eclipse Luna, then [ECLIPSEVERS]is *Luna*

[JAVAHL] The SVN connector to connect the Eclipse IDE with the subversion repository. Use *Native JavaHL 1.8.10 (64-bit)*.

[TORTOISE] Latest version (March 2015) is Tortoise SVN v1.8.10
Download from `http://tortoisesvn.net/`

1.2 How to install JavaView within Eclipse IDE w/o Sources

JavaView_eclipse_quickins

1. Install Java JDK from [JAVAJDK]or make sure it is already installed. You can download it on `http://www.java.com/de/download/`

2. Download JavaView: Install JavaView from [JAVAVIEW].

3. Download and install Eclipse: Install Eclipse from [ECLIPSE]. At first start you are asked to specify a workspace directory (your workspace directory is not the one where your source code or JavaView is located). Make sure that your Eclipse version is for the same architecture as your installed Java version. That means if you have downloaded Eclipse for a 64bit machine (e.g. Win7 64bit) then Java needs to be installed as a 64bit version as well. Similarly if one of them is for 32bit the other one needs be a 32bit version, too.

4. Create a new project: Create a new directory on your disk for your source code (named `%sources%`). Create a subfolder named "Base" in your `%sources%` directory. In Eclipse, select `File -> New Project -> Java Project`. Enter "JavaView" as Project name. Under Project Layout, make sure that `Use project folder as root...` is enabled. Click next. On the Source tab, click `Link additional source` Hit `Browse` and select "`%sources%`/Base" as a folder Ensure that `Replace existing project source folder...` is selected, then click "Finish". Back in the Java Settings dialog, click the Libraries tab. Click `Add External Jars...` and add the files `%javaview%`/jars/javaview.jar, jvx.jar and vgpapp.jar. Click Finish.

   Hint: If you have any problems during this step, e.g. if some options are disabled or not possible to activate then it is likely that you still have some leftovers from a previous try and therefore Eclipse automatically tries to set up something based on the existing folders or files. If this is the case make sure you delete the project folder "JavaView" (not the %javaview% folder of your JavaView installation!) in your workspace using a file explorer from your operating system (e.g. Windows Explorer, Dolphin,...). Alternatively you can delete the project folder directly in Eclipse in which case you are asked whether the content on disk should also be deleted. You need to select this option. Then try step 4 again.

5. Run JavaView: Open `Run -> Run Configurations...`. Choose `Java Application -> New configuration` and name it JavaView. As main class, choose `javaview`. Click `Run`. JavaView should start.

6. Import tutorial code: In the Package Explorer, right-click the Base folder and click `Import` Choose `General -> File System` and hit `Next`. Browse to %javaview%/vgp and click `OK`. A folder "vgp" should appear in the list on the left. Mark the checkbox. Select the checkbox `Create top-level folder`. Click `Finish`. In your package explorer you should now see a bunch of subfolders starting with "vgp.tutor.xxx"

7. Running a tutorial project: Select any tutorial project of your choice. For instance, browse to "vgp.tutor.parm". This folder should contain (among others) the source file "PaParmSurface.java" Right-click this file and choose `Run as Java Application`.

8. Adding your own source to the project: It might be convenient to add a new folder for your own sources to the project. To do this, in the package explorer right-click the top project folder "JavaView" and select `New ->`

`Source Folder`. Give it a name: In the `Folder Name` text field enter "MySource" and click `Finish`.

You can now add own source code to this project. Remember to change the run configuration to your own class containing a Java main()-method.

9. Adding JavaView-JavaDoc to the project: You may want to add the generated JavaDoc documentation for JavaView methods to your project to have automatic help on what a method does available when typing. To do so, go to your project folder "JavaView" in Eclipse and unfold the "Referenced Libraries" subfolder where you should find your three external libraries javaview.jar, jvx.jar and vgpapp.jar you have added in the beginning. Right-click javaview.jar and select `Propertiers`. Select `Javadoc Location` from the list on the left and browse and enter the path to the JavaView doc which is %javaview%/doc/reference, then hit `OK`.

1.3 Install JavaView within Eclipse IDE with Sources

setup_eclipse_javaview.doc

1. Required Software:

   - Java JDK
     Download from [JAVAJDK]
   - Tortoise SVN
     Download from [TORTOISE]
   - Eclipse IDE for Java Developers
     Download from [ECLIPSE]
   - Eclipse - SVN connector
     - Subversive: `Eclipse -> Help -> Install New Software`. Work with (use dropdown selector):
       * [ECLIPSEVERS]- http://download.eclipse.org/releases/[ECLIPSEVERS]
       * Collaboration and select and install
         · Subversive Revision Graph (optional)
         · Subversive Team Provider
         · Subversive Team Provider Source
       * Restart Eclipse
     - Connector:
       You should be promoted to load a connector DLL. IF NOT, then open `Window -> Preference -> Team -> SVN` which now should open install dialog. Select: [JAVAHL]and install.

2. Set up Java Software:

   - Copy sjpp_modified.jar to [JAVAJRELIBPATH].
   - - (optionally) Lower security level to medium in order to run applets on local html files: `System -> Java -> Security -> Security Level`

3. Create JavaView Folders in File System

- Create folder structure:

```
.../JavaView-SVN/
            JavaView/
            JavaViewDev/
```

4. Setup of SVN

   - Checkout SVN repository JavaView
     - Right-mouse click on folder JavaView
     - Choose Checkout
     - then: `https://scherk.imp.fu-berlin.de/svn/JavaView/trunk`
       Checkout directory: JavaView
   - Checkout SVN repository JavaViewDev
     - Right-mouse click on folder JavaViewDev
     - Choose Checkout
     - then: `https://scherk.imp.fu-berlin.de/svn/JavaViewDev/trunk`
       Checkout directory: JavaViewDev

5. Setup of Eclipse

   - Create workspace in folder JavaView-SVN
     - Launch eclipse
     - Type workspace: ./JavaView-SVN
       This will create a subfolder .metadata in folder JavaView-SVN
   - Configure Package Explorer via arrow-down button:
     - Select Top Level Elements: Working Sets
     - Select Package Presentation: Hierarchical
     - Disable "Show 'Referenced libraries' node"
     - Filters: select only:
       .*resources, Empty library containers, Libraries from external, Libraries in project, Non-Java Elements
   - Create working sets:
     - Right-click inside Package Explorer
     - Select `New -> Java Working Sets`
     - Type working set name: JavaView
     - Same way create working set: JavaViewDev
   - Import projects:
     - Into working set JavaView:
       * Select: `File -> Import -> General -> ‘‘Existing Projects into Workspace’’`
       * Find projects in folder: ./JavaView-SVN/JavaView
       * Select projects: _eclipse, srcBase, srcDoc
     - Into working set JavaViewDev:

    - Select: `file -> import -> General -> ‘‘Existing Projects into Workspace’’`
    - Find projects in folder: ./JavaView-SVN/JavaViewDev
    - Select projects: Develop, DevelopContrib, OpenGL, OpenGL_LWJGL, ThirdLibs

6. Configure of Eclipse

    - Open Window-Preferences
    - `General -> Editors -> Text Editors -> Spelling`:
      User defined dictionary:
      `${workspace_loc:_eclipse/rsrc/javaview_userDictionary.txt}`

      Encoding: Assure PREPROC selected as Default
    - `General -> Workspace`:
      Set “Text File Encoding” and type “PREPROC” (must be typed by hand), press APPLY
    - `Java -> Build Path`
      Set “Output folder name” and type “class”
    - Load Java Preferences:
      `File -> Import -> General -> Preferences`
      Preference file: `JavaView-SVN/JavaView/_eclipse/rsrc/javaview_javaCodeStylePrefs.epf`
    - `Java -> Editor -> Folding`:
      Deselect “Imports”
    - `Java -> Installed JREs` and add your jdk version here: [JAVAJDKVERS]: `Add -> Standard VM -> Directory` = [JAVAJDKPATH].
      Select JDK as Default
    - `Run/Debug -> Launching -> Launch Configurations`:
      Deselect: Filter configurations in closed projects
      Deselect: Filter configurations in deleted or missing projects
      Filter checked launch configurations:
      Select: all other EXCEPT Java Applet and Java Application
    - Connect with SVN
      Right-click on Working Set `JavaView -> Team -> Share Projects`
      Select: srcBase, srcDoc, _eclipse
      Right-click on Working Set `JavaViewDev -> Team -> Share Projects`
      Select: Develop, DevelopContrib, ThirdLibs, OpenGL, OpenGL_LWJGL

      Reconnect: URL: `https://scherk.imp.fu-berlin.de/svn/JavaView/`
      Must increase prepending number = 2
      User Authentication: insert your SVN credentials, enable SAVE
    - Procedure with JavaViewDev, connect all five projects
    - Select number of warnings:
      `Eclipse -> Problems Panel -> Dropdown Selector -> Configure Contents`. Select: Number of items visible per group = 2000
      Note: in the “common” tab, set “save as” to shared file “_eclipse\launch”

1.4 Installation Pitfalls

This section lists some questions that came up in the tutorials regarding the installation and configuration of JavaView, Eclipse and Java.

1. Q: Under Mac OSX Lion (10.7.x): On JavaView launch several error messages are printed in the console of the type

   ```
   java[31185] <Error>: CGContextGetCTM: invalid context 0x0
   java[31185] <Error>: CGContextSetBaseCTM: invalid context 0x0
   java[31185] <Error>: CGContextGetCTM: invalid context 0x0
   java[31185] <Error>: CGContextSetBaseCTM: invalid context 0x0
   ```

   A: This bug seems to appear on OSX Lion and newer, but apparently it does not have any serious impact, so you may ignore it.

2. Q: I have installed Eclipse and Java but everytime I try to run Eclipse it says "A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run eclipse. No Java virtual machine was found after searching the following locations: `C:\eclipse \jre \bin\javaw.exe`. javaw.exe in your current path"

   A: Usually the Java installation automatically sets the Java installation path in a global environment variable. What is most likely is that you have downloaded Eclipse and Java for different architectures, e.g. Eclipse for 64bit Windows and Java for 32bit Windows. Under Windows 7, 64bit software is typically installed under Program Files whereas 32bit software is installed und Program Files (x86). Make sure that your versions are for the same architecture, i.e. : If Java 64bit is installed then use Eclipse 64bit and vice versa. If Java 32bit is installed then use Eclipse 32bit and vice versa.

   Note that you can have both versions installed at the same time, but the default Java runtime environment is probably set to only one of them.

1.5 Exporting a runnable .jar

This tutorial will explain how to export your written projects as runnable jar archives

1. Make sure you know the name of the run configuration to use for your project.
2. Right-click the main java class of your project, i.e. the class which contains the static main- method and is launched by your run configuration. In JavaView these classes usually start with Pa..., e.g. PaMyProject
3. Hit "export" and select `Java -> Runnable JAR file`. Hit "next"
4. Under "Launch Configuration" select the run configuration you used to use for starting your project. Under "Export destination" select a path and file name your jar file will be saved to. Under "Library handling" make sure that the option `Extract required libraries into...` is selected. Hit "Finish".

5. Most likely you can just ignore the warnings.

6. Make sure that your program runs by double-clicking the generated jar-file.

Remember possible naming convention your tutor might have told you for electronic submission.

2 Essential JavaView Classes

We start with a representation of basic geometry shapes as Java classes. The following code is intended for educational purpose and may not reflect the current implementation.

2.1 jv.vecmath

The vecmath package provides classes and methods for linear algebra and simple vector calculations.

2.1.1 Classes

```
jv.vecmath.PdVector    // Class for an array of double values.
jv.vecmath.PiVector    // Class for an array of integer values.
jv.vecmath.PdMatrix    // Class for a small m*n matrix of double values
jv.number.PdColor      // Class for a color with red, green and blue components
```

2.1.2 Details of jv.vecmath.PdVector

A point or a vector in 3D respectively nD space can be represented with an instance of PdVector. A vector also provides several methods for linear algebra calculations.

```
class PdVector {
  double [] m_data;    // array of double values

  // constructs a vector with given size
  PdVector(int dim) {
    m_data = new double[dim];
  }
  // constructs a vector with two entries
  PdVector(double x, double y) {
    m_data = new double[] {x, y};
  }
  // retrieve a single array component at given index
  double getEntry(int ind) {
    return m_data[ind];
  }
}
```

Sample usage:

```
PdVector v   = new PdVector(1., 0.);
PdVector w   = new PdVector(0., 1.);
v.add(w);
double norm  = v.length();
```


```
PdVector v   = new PdVector(1., 0.);
PdVector w   = new PdVector(0., 1.);
v.add(w);
double norm  = v.length();
```


This type also serves as a list for doubles. It can have several flags set. For instance if p is a PdVector which is a point of geom, then p.hasTag(IS_SELECTED) is true iff p is currently selected/marked on the display. In addition it comes

with many methods for common vector calculations such as dot and cross product, length, distance, orthogonal projections and so on. Also check the static methods!

2.1.3 Details of jv.vecmath.PiVector

Similar to PdVector, but for integers. May be used as a list/stack for integer values. Use instead of List<Integer>. New entries can be added to list by calling aPiVector.addEntry(n). This will append the number n to the list. Don't confuse with aPiVector.add(x) which adds the value x to each entry of the vector. An interesting method is removeSuccessiveDuplicates() which does what its name says.

2.2 jv.geom

The geom package provides classes and methods for widely used geometry containers such as sets of points, polygons and faces.

2.2.1 Classes

```
jv.geom.PgPointSet    // Class for a set of points in 2D, 3D, or nD space.
jv.geom.PgPolygon     // Class for a single polygon.
jv.geom.PgPolygonSet  // Class for a set of polygons sharing points
jv.geom.PgElementSet  // Class for a surface of planar polygonal faces
jv.geom.PgVectorField // Class for a vector field associated to a base geometry
jv.geom.PgTexture     // Class for a texture image
```

2.2.2 Details of jv.geom.PgPointSet

A set of points in either 2d, 3d, or any n-dimensional ambient space. All points in a point set must have the same dimension.

```
class PgPointSet {
  int           m_dim;            // uniform dimension of each point
  PdVector []   m_vertex;         // array of all points
  PdVector []   m_vertexNormal;   // optional, a normal vector at each point
  PdColor  []   m_vertexColor;    // optional, a color of each point
  PdVector []   m_vertexTexture;  // optional, a texture position of each point

  // constructor specifies dimension of ambient space
  PgPointSet(int dim) {
    m_dim    = dim;
  }
  // allocate a set of vertices (and, if exist, vertex normals and colors)
  void setNumVertices(int num) {
    m_vertex = new PdVector[num];
    for (int i=0; i<num; i++)
      m_vertex[i] = new PdVector(m_dim);
  }
}
```

insert pic

Sample usage:

```
PgPointSet ps = new PgPointSet(2);
ps.setNumVertices(3);
ps.setVertex(0, 1., 0.);
```

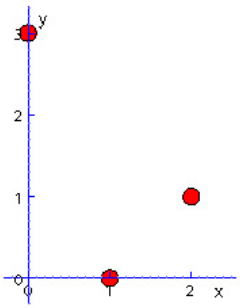

Figure 1: A point set geometry.

```
ps.setVertex(1, 2., 1.);
ps.setVertex(2, 0., 3.);
```

2.2.3 Details of jv.geom.PgElementSet

Polyhedral surface in n-dimensional space is based on a point set.

```
class PgElementSet extends PgPointSet {
  PiVector [] m_element;          // array of faces
  PdVector [] m_elementNormal;  // optional, a normal of each face
  PdColor  [] m_elementColor;   // optional, a color of each face
  PdVector [] m_elementTexture; // optional, a texture position of each point

  // constructor specifies dimension of ambient space
  PgElementSet(int dim) {
    super(dim);
  }
  // allocate a set of facees (and, if exist, element normals and colors)
  void setNumElements(int num) {
    m_element = new PiVector[num];
    // note: size of individual of elements not known yet
  }
}
```

Sample usage:

```
PgElementSet es = new PgElementSet(2);
// Use points of previous point set
es.copy(ps);
es.setVertices(3);
es.setVertex(3, 2., 3.);
// Define two triangles
es.setNumElements(2);
es.setElement(0, 0, 1, 2);    // vertex indices of first face
es.setElement(1, 2, 1, 3);    // vertex indices of second face
double area = es.getArea();
```

Additional useful member methods:

- geom.getVertices(): Returns a list of the vertex indices

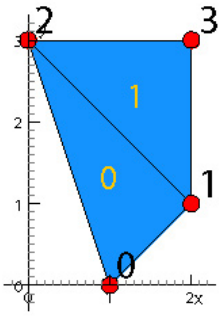

Figure 2: An element set geometry.

- geom.showElementColors(true/false): Enables the display of individual element colors in contrast to the global element color which is shown by default. The same story with geom.showVertexColors(true/false).
- geom.showVertices(true/false): Displays the vertices of the geometry.
- geom.setVertexColor/Size/...: Sets individual vertex properties such as color or size. Make sure you have enabled display of individual vertex colors by calling geom.showVertexColors(true) before.
- geom.getElement(i): Returns a PiVector of integers of the vertex indices forming the element with index i.

2.3 jv.viewer

Provides functionality for the display and related managing tasks.

2.3.1 Classes

```
jv.viewer.PvDisplay    // Class for a single display for viewing.
jv.viewer.PvControl    // Class is a panel to hold several inspectors.
jv.viewer.PvViewer     // Manages several displays and a control panel.
jv.viewer.PgAxes       // Utility class for the coordinate axis and labels.
jv.viewer.PvCamera      // Utility class for a single camera.
jv.viewer.PvLight      // Utility class for a single light.
```

2.3.2 Details of jv.viewer.PvDisplay

A display is a drawing canvas which shows a collection of geometries:

```
class PvDisplay extends Canvas {
  // add a geometry to the scene
  void addGeometry(PgGeometryIf geom) { ... }
  // adjust the scene
  void showAxes(boolean flag) { ... }
  void setCamera(int type) { ... }
}
```

Sample usage:

```
PvDisplay disp = new PvDisplay();
disp.addGeometry(es);
disp.setBackground(Color.red);
// insert display to applet (java.awt.Applet or java.awt.Frame)
applet.add(disp);
```

You can get the currently selected (i.e. active) geometry by
PgElementSet activeGeom = (PgElementSet) m_display.getSelectedGeometry();

2.4 jv.number

Utility classes for inspectable numbers and colors. A double or integer may have a slider, and a color may have an color inspector.

2.4.1 Classes

```
jv.number.PuDouble     // Class for a single double with slider.
jv.number.PuInteger    // Class for a single integer with slider.
jv.number.PdColor      // Class for a color with red, green and blue components
```

These classes provide wrapper for the corresponding native types int, double and so on. They can register action listeners to react on events and ship with a default inspector panel which can be obtained and added to the project IP using the getInfoPanel() member method. Make sure that this is done in the IP class' setParent() method, since you need to access the project class instance most likely.

PdColor: This class is the JavaView implementation representing a color. Apart from the methods already provided by java.awt.Color it has many additional functions e.g. for conversion, creation and blending between different color formats, both as member methods and static functions.

2.5 jv.anim

Animation dialog and containers classes for keyframe animation.

2.5.1 Classes

```
jv.anim.PsAnimation    // Class with a play control triggers animated geometries.
jv.anim.PsKeyframe     // Container class for a keyframe animation.
```

2.6 jv.loader

Import and export classes for reading and writing of geometry files. More loader are provided in extension packages jvx.loader and elsewhere.

2.6.1 Classes

```
jv.loader.PgLoader      // Optional loader manager determines the correct loader.
jv.loader.PgJvxLoader   // Class for reading/writing a JavaView JVX file.
jv.loader.PgObjLoader   // Class for reading/writing a Wavefront OBJ file.
jv.loader.PgMathLoader  // Class for reading/writing a Mathematica file.
```

3 Functional Geometric Shapes

3.1 jvx.surface

```
jvx.surface.PgDomain            // Class is rectangle in the (x,y) plane.
jvx.surface.PgGraph             // Graph surface of one scalar function on a planar
    rectangle.
jvx.surface.PgParmSurface       // Surface of three scalar functions (u(x,y),v(x,y),w(x,y
    )).
jvx.surface.PgSurfaceOfRotation // Parametrized surface of rotation
```

3.2 jvx.geom

```
jvx.geom.PgBezierCurve          // Polygon determined by control points
jvx.geom.PgTube                 // Tube surface around a associated polygon
jvx.geom.PgPolygonOnElementSet  // Polygon which lies on an associated element set
```

3.3 dev.primitive

Contains many more functional geometry classes which support the JVF format. Will be discuss later.

4 Algorithms and Workshops

The advanced algorithms are kept separately from the geometry classes in so-called workshop classes. Workshops will take a, usually a single, geometry and perform some operations on the geometry. When done the workshop is closed.

Workshops derive from a base class which optionally stores a backup geometry and which simplifies the interaction with a user dialog.

```
jvx.project.PjWorkshop  // Base class for workshops, handles CANCEL and RESET
```

4.1 jvx.geom

```
jvx.geom.PwClip     // Clips a surface at a user defined level function
jvx.geom.PwGeodesic // Computes geodesic curves on a surface
jvx.geom.PwLIC      // Visualizes a vector field on a surface using textures
jvx.geom.PwNoise    // Adds random noise to vertices of a surface
jvx.geom.PwExplode  // Explodes a surface into parts
jvx.geom.PwSeeds    // Investigates a vector field using particles
jvx.geom.PwUnfold   // Unfolds a surface to a planar region
```

5 Projects

A collection of advanced applications. Each application is given by

```
jv.project.PjProject  // Base class for projects with support for pick events
```

5.1 vgp package

Contains more than 50 elaborated research applications as well as tutorials including

```
vgp.tutor.PjFractalImage          // Computes Mandelbrot and Julia fractals
vgp.tutor.PjTorusKnot             // Curves winding around a torus
vgp.minimal.weier.PjWeierstrass   // Classic minimal surfaces given by Weierstrass
    equations
vgp.game.life.PjLife              // Games of life on a surface in 3D.
vgp.discrete.harmonic.PjHarmonic  // Discrete harmonic maps
```

6 Usage

JavaView-SVN - doc - userManual

6.1 The Update-Mechanism

This section needs translation

Problem: Die update(Object) Methode von Objekten sollte nur von wenigen, ausgewaehlten Stellen heraus aufgerufen werden. Insbesondere sollte sie NICHT in Methoden aufgerufen werden, die Berechnungen ausfuehren, also nicht in compute-Methoden.

Begruendung: Wenn eine compute-Methode selber ein update() aufruft, kann man sie nicht in eine Folge von Befehlen einbauen, an deren Ende das Objekt dann genau einmal upgedated wuerde. Wenn man eine Folge compute1 (); compute2();... etc. haette, dann wuerde jede einzelne compute() Methode ein update() initiieren.

Loesung: Update() im wesentlichen (es gibt Ausnahmen) in der eigentlichen public boolean update(Object object) aufrufen, oder nach user interaction vom Panel, oder nach anderen Events.

Allgemeine Strategie:

1. Verwende geom.update(null), um den Parent von geom zu informieren, dass sich geom geaendert. Dieser Call initiiert in PsObject einen Aufruf m_parent .update(geom) und springt deshalb in die update() Methode vom Parent. Ausserdem verwende diesen Befehl, um die Geometrie im Viewer upzudaten. Dieser Call setzt voraus, dass geom und seine Panels alle auf dem neuesten Stand sind. Der folgende Call macht nicht diese Voraussetzung:

2. Verwende geom.update(geom), um zuerst die Info/Material/... Panels nach einer Aenderung in geom wieder auf den neuesten Stand zu bringen. Danach wird automatisch geom.update(null) aufgerufen. Der 2. Fall beinhaltet also den ersten Fall. Wenn die Panels nicht sichtbar sind, werden sie auch nicht geupdated (erst wenn sie wieder sichtbar werden. Wenn aber zum Beispiel das InfoPanel sichtbar ist, dann wird es immer geupdated.

3. Verwende geom.update(other), wenn sich object veraendert hat und geom darauf reagieren soll. In der Update Methode von geom MUSS dann abgefragt werden, ob das event==object ist. Nachdem geom dann auf den neuesten Stand gebracht worden ist, kann das update an die Oberklasse von geom weitergereicht werden. Normalerweise mit Argument null oder this, da geom ja auf dem neuesten Stand ist:

```
Geom {
    update(Object object) {
    if (object==other) {
        // bringe geom auf den neuesten Stand
        return super.update(null); // oder super.update(this)
    } else if (object== ...
```

Ein Verstaendnis dieses Mechnismus ist wichtig, wenn Ihr Code schreibt, auf den andere aufbauen.

6.2 Misc

- In each applet the author must specify topic and author in method Applet.getAppletInfo(). This information is displayed in the ABOUT APPLET panel of each applet to be invoked with the menu Help -¿ About Applet.
- Comment from JBuilder: Use frame.pack() for frames die Infos über die bevorzugte Größe besitzen, z.B. aus ihrem Layout. Use frame.validate() for frames, die eine voreingestellte Größe besitzen.