Quick Start Guide

zum Quellcode von

Computer Science Talentscout

Autor: Reto Lamprecht

Letzte Änderung: 11.12.2007

1. Packages

Generell gilt:

- Jedes Modul besteht aus drei packages, dem Basis-package *ch.ethz.inf.csts.<moduleName>*, und zwei Sub-packages *.gui* und *.manual* sowie einer Main-Klasse im package *ch.ethz.inf.csts.modules* mit dem Namen des Moduls (z.B. *BooleanCube.java, RecursiveImages.java,* etc.).
- Packages mit der Endung *.gui* enthalten Klassen, welche mit dem GUI-Designer von NetBeans erstellt worden sind. Die Klasse mit dem Haupt-JPanel heisst dabei immer *GUI.java* und wird erweitert von einer Klasse Namens *Main.java* im Basis-package des Moduls, die den GUI-Elementen Funktionalität hinzufügt.
- Packages mit der Endung *.manual* enthalten Klassen und HTML-Dateien, die für das integrierte Manual jedes Moduls benötigt werden.

1.1 Package Liste

ch.ethz.inf.csts
Allgemein nützliche Klassen.

ch.ethz.inf.csts.consistency
Framework um verschiedenen Repräsentationen einer Grösse konsistent zu halten. Das Herzstück bildet die Klasse *ConsistencyManager*, welche Änderungen von *ConsistentFeatures* an alle registrierten *ConsistentObjects* weiter propagiert.
Namenskonvention: alle ConsisistentFeature-Variablen beginnen mit „cf_“. (z.B. *cf_paramA* oder *cf_period* in *ConsistentRndGeneratorLinCon.java* im package *ch.ethz.inf.csts.modules.randomNumbers*.)

ch.ethz.inf.csts.gui
Allgemein verwendbare GUI-Klassen (erstellt mit dem GUI-Designer von NetBeans).

ch.ethz.inf.csts.modules
Enthält die Main-Klassen aller Module sowie die dazugehörigen Sub-packages.

ch.ethz.inf.csts.modules.booleanCube
ch.ethz.inf.csts.modules.booleanCube.gui
ch.ethz.inf.csts.modules.booleanCube.manual

Die Klassen und Manual HTML-Dateien des Moduls „Boolean Cube“.

ch.ethz.inf.csts.modules.imageCompression
ch.ethz.inf.csts.modules.imageCompression.gui
ch.ethz.inf.csts.modules.imageCompression.manual
Die Klassen und Manual HTML-Dateien des Moduls „Image Compression“.

ch.ethz.inf.csts.modules.randomNumbers
ch.ethz.inf.csts.modules.randomNumbers.gui
ch.ethz.inf.csts.modules.randomNumbers.manual
Die Klassen und Manual HTML-Dateien des Moduls „Random Numbers“.

ch.ethz.inf.csts.modules.recursiveImages
ch.ethz.inf.csts.modules.recursiveImages.gui
ch.ethz.inf.csts.modules.recursiveImages.manual
Die Klassen und Manual HTML-Dateien des Moduls „Recursive Images“.

ch.ethz.inf.csts.modules.steganography
ch.ethz.inf.csts.modules.steganography.gui
ch.ethz.inf.csts.modules.steganography.manual
Die Klassen und Manual HTML-Dateien des Moduls „Steganography“.

ch.ethz.inf.csts.templates
Muster-Klassen. Diese werden nicht direkt instanziiert oder erweitert, sondern können kopiert und angepasst werden.

ch.ethz.inf.turingtools
ch.ethz.inf.turingtools.demo
Framework um rasch und einfach Endliche Automaten (Finite State Machines) und Turing Maschinen zu definieren und auszuführen. Diese eignen sich unter anderem, um Strings zu parsen und werden zu diesem Zweck konkret verwendet im Modul „Image Compression“.

2. Integration der Modul-Manuals

Die Benutzeroberfläche jedes Moduls hat im oberen Bereich auf der ganzen Fensterbreite ein Manual, das kurz die wichtigsten Funktionen erklärt. Dieses Manual ist durch eine JSplitPane von der eigentlichen Modul-Oberfläche getrennt. Die Klasse *ch.ethz.inf.csts.gui.SplitPane.java* kombiniert die beiden Komponenten folgendermassen:

- Die Klasse *Main.java*, die sich im Basis-package eines Moduls befindet, kombiniert alle Komponenten, die zur Modul-Oberfläche dazu gehören, jedoch nicht das Manual.
- Das Manual wird mit Hilfe der Klasse *ch.ethz.inf.csts.gui.SplitPane.java* in die Modul-Oberfläche integriert. Diese Klasse besitzt einen Container für das Manual Namens *manualEditorPane* und einen für die Modul-Oberfläche Namens *mainPanel*.
- In diese Container eingefügt werden Manual und Modul-Oberfläche im Konstruktor der Main-Klasse des Moduls im package *ch.ethz.inf.csts.modules*, welche die Klasse *SplitPane.java* erweitert und immer den Namen des Moduls trägt. (z.B. *BooleanCube.java*, *RecursiveImages.java*, etc.).
- Der Code im Konstruktor der Klasse *BooleanCube.java* sieht z.B. folgendermassen aus:

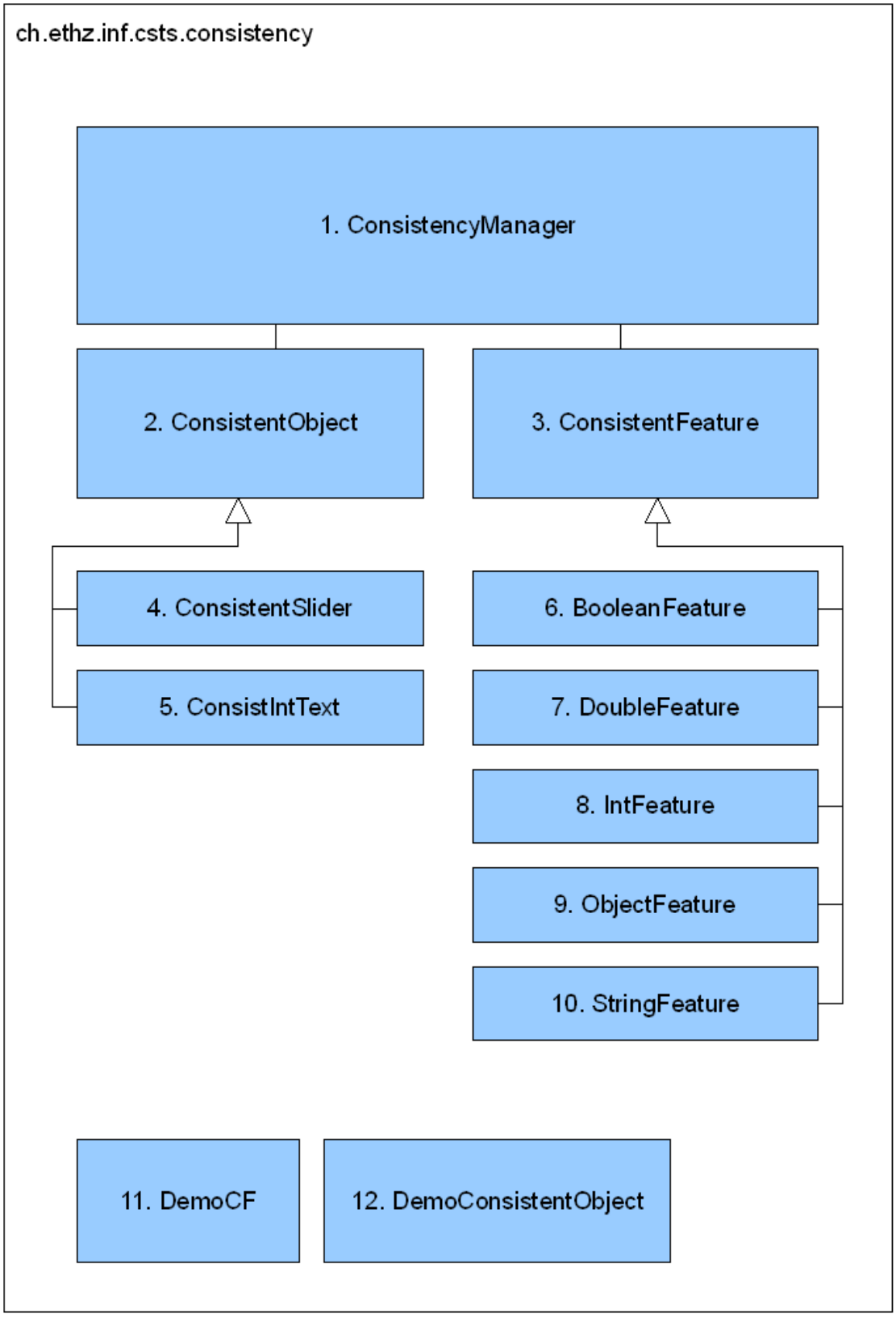
```
public BooleanCube() {
  Main main = new Main();
  mainPanel.setLayout(new BorderLayout());
  mainPanel.add(main, BorderLayout.CENTER);

  new Manual4BooleanCube(manualEditorPane,
                                main,
                                new HighlightGlassPane(this)
                               );
}
```

- Gestartet werden alle Module über diese Main-Klasse im Basis-package.

Das Manual zeigt die durch Hyperlinks verknüpften HTML-Dateien im *manual*-package. Mit speziellen Hyperlinks lassen sich zudem einzelne Kompnenten der Benutzer-oberfläche farblich hervorheben. Um eigene Hyperlinks mit selber definierter Funktionalität einzuführen, kann man einfach die Methode *highlight()* oder *execute()* der Klasse *ch.ethz.inf.csts.ManualPane* überschreiben.

3. Consistency Package



3.1 Javadoc Comments

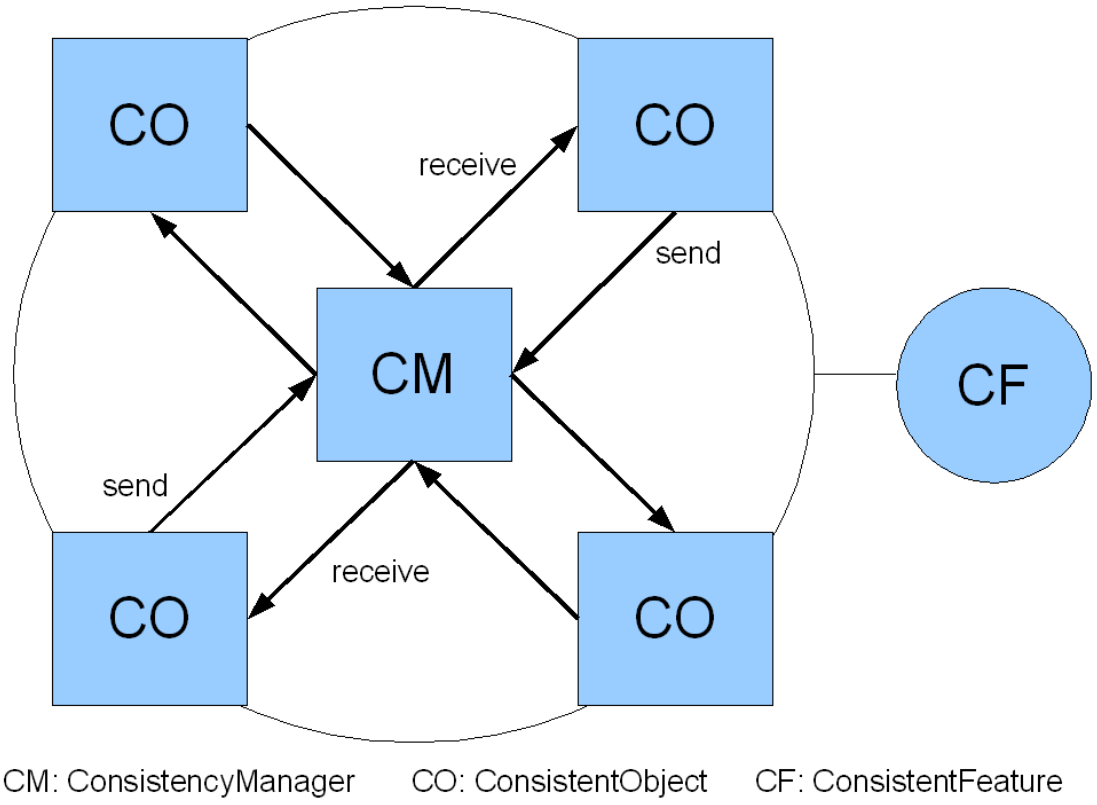
1. *ConsistencyManager*

This class manages consistency between different representations of data features. It does what a model in an MVC architecture normally takes care of, but since it showed that some modules contained many complex MVC constellations with models which may be distributed in several classes, this part has been factored out.

Instances of the class *ConsistentObject* can notify the ConsistencyManager of changes and receive notifications from it.

Instances of the class *ConsistentFeature* contain the data as name/value-pairs.

Through the method *addBond(...)* a ConsistentFeature and a ConsistentObject can be linked. After that, the ConsistentObject instance will be notified by the ConsistencyManager of any feature changes through the method *receiveStateChanged()*, and if your ConsistentObject notifies the ConsistencyManager of its own feature changes through the method *sendStateChanged()*, the notification will be forwarded.



2. *ConsistentObject*

This class is able to notify the *ConsistencyManager* and get notifications from it. To distinguish incoming and outgoing events it has two methods:

- *receiveStateChanged()*
- *sendStateChanged()*

It implements various Listener methods to receive GUI-events, so you can add the ConsistentObject as a ChangeListener, FocusListener or KeyListener to a GUI component and notify the ConsistencyManager of changes through *sendStateChanged()* when needed. Your own objects can either be extensions of ResponsiveObject or have a delegate to one and receive events from the ConsistencyManager through *receiveStateChanged()* and react on them as needed.

3. *ConsistentFeature*

This is the superclass of all ConsistentFeature with different types. It just has a name and a toString method.

4. *ConsistentSlider*

This is a ConsistentObject associated with a JSlider. It will set the slider value to the feature value upon *receiveStateChanged()* and forward any change of the slider position to the ConsistencyManager.

5. *ConsistIntText*

This is a ConsistentObject associated with a JTextField. It will set the JTextField value to the feature value upon *receiveStateChanged()* and notify the ConsistencyManager if ENTER is pressed or the JTextField loses the focus.

6. *BooleanFeature*

A ConsistentFeature with a boolean value.

7. *DoubleFeature*

A ConsistentFeature with a value of type double.

8. *IntFeature*

A ConsistentFeature with an int value.

9. *ObjectFeature*

A ConsistentFeature with a value of type Object.

10. *StringFeature*

A ConsistentFeature with a String value.

11. *DemoCF*

This class just contains the features for *DemoConsistentObject*.

12. *DemoConsistentObject*

A sample appication of ConsistentObject.

You need to override ConsistentObject for each of your own objects since each of your own objects will send and react differently upon feature changes.