Entwicklerdokumentation KryptoProjekt

Dokument Typ:	Entwicklerdokumentation
Title:	SR-Solutions - Framework für die Lehre
Document Number	*002*
Version	00.01
Status:	*Final*
Erstellungsdatum:	19.07.2010
Schutzklasse:	*For internal use only*

1 Inhaltsverzeichnis

2 Einführung

2.1 Was ist das KryptoProjekt

Das Programm wird Open-Source entwickelt und soll die Studierenden im Praktikum der Vorlesung "Kryptographie und Codierung" im Lernprozess unterstützen. Es ist möglich verschiedene Algorithmen und Basisfunktionen nach dem Baukastenprinzip zusammenzustellen, zu kombinieren und damit zu experimentieren.

2.2 Anmerkungen zu diesem Dokument

Dieses Dokument dient der Weiterentwicklung des Programms. Für die Endanwender wurde eine eigene Dokumentation erstellt, welche die Bedienung des Programms erläutert.

3 Programmstruktur

Im folgenden Abschnitt soll die Programmstruktur des KryptoProjekts näher erläutert werden. Hierbei werden nicht alle Methoden erläutert, sondern es werden nur wichtige, für die Weiterentwicklung nötige, Methoden und der gesamte Aufbau des Programms näher beschrieben. Für weiterführende Informationen über die Methoden stehen Javadocs zur Verfügung.

3.1 Formalien

3.1.1 Sprache

Die Software KryptoProjekt wird in der Programmiersprache Java entwickelt.

3.1.2 Coding Conventions

- Java-Conventions
- NetBeans-Standard
- Kommentare über die betreffende Zeile/Methode
- Methodendokumentation über die Methode
- Kommentare, Variablen, Methoden, usw. auf Englisch
- übersichtlicher/lesbarer Code (Whitespace, wenn nötig)

3.1.3 IDE

Als IDE für die Architektur ist das Programm Visual Paradigm vorgesehen, als Entwicklungsumgebung NetBeans.

3.1.4 Versionsverwaltung

Um die Arbeit im Team zu erleichtern wurde die Versionsverwaltung Git eingesetzt. Das Projekt kann später aber auch auf andere Versionsverwaltungen exportiert werden.

3.2 Aufbau

Die Software soll im Baukastenprinzip erstellt werden, sodass die Software leicht um Module erweitert und leicht gewartet werden. Hierfür wurden mit den folgenden Punkten die nötigen Voraussetzungen geschaffen.

3.2.1 Graphischer Aufbau von Algorithmen

Die Algorithmen, welche zur Lernunterstützung programmiert werden, sollen in einzelne Bausteine (JInternalFrames) zerlegt werden. Diese sollen dann über eine logische Verbindung (dargestellt durch einen Pfeil zwischen den Bausteinen) miteinander verbunden werden.

3.2.2 MVC Pattern

Das gesamte Softwaresystem basiert auf einem modifizierten MVC-Pattern. Folglich ist das System in drei Hauptpackages aufgeteilt. Im Model werden die verschiedenen Funktionalitäten implementiert, der Controller überprüft die Eingaben aus der View und leitet sie gegebenenfalls an das Model weiter. In der View werden die Ergebnisse aus dem Model angezeigt.

3.2.3 UML Diagramm

Hier wurden die wichtigen Funktionalitäten auf die entsprechenden Pakete aufgeteilt. Diese sind View, Controller, Network, Administration und Model.

Für jedes neue Themengebiet, beispielsweise Kodierung oder Verschlüsselung, wird eine neue Oberklasse/Interface erstellt, welche die grundlegenden Methoden beinhalten. An diese werden dann die verschiedenen speziellen Klassen angehängt, sodass es einfach ist, die verschiedenen Themengebiete zu erweitern.

Die Pakete Network und Administration sind bisher noch nicht implementiert und sind der Vollständigkeit halber dargestellt.

Das Controller-Paket bildet die Schnittstelle zwischen View und {Model, Network, Administration}. Er bildet die darunter liegenden Funktionen auf einer höheren Abstraktionsebene ab. Für jedes Themengebiet wird hierzu eine eigene Controllerklasse erstellt. So wird auch die Übersicht bewahrt.

Im View-Paket werden einzelne Bausteine (Kits, erbt von JInternalFrame) erstellt, die dem MainPanel (Desktop) in KryptoProjektView hinzugefügt werden müssen.

4 Programmtechnischer Aufbau

4.1 GUI

4.1.1 Erstellen von neuen Bausteinen

Der Aufbau eines neuen Bausteins funktioniert folgendermaßen:

Zuerst muss im betreffenden Paket eine neue `JFrame`-Form erstellt werden. Auf dieser wird links oben ein `JLabel` angebracht, welches als Namensbezeichnung des Bausteins dient. Auf das restliche Frame wird ein `JPanel` gezogen, um die logischen Komponenten `DropTextField` und `DragList` platzieren zu können.

Es kann alles mit dem GUI Builder von Netbeans modelliert werden, außer die Textboxen für Drag-and-Drop, da diese speziell überschrieben wurden. Diese müssen extra im Code hinzugefügt werden.

Im Folgenden am Beispiel des `AdditionFrames` erläutert, wie ein neuer Baustein erstellt wird. Hierfür müssen einige Änderungen am Code durchgeführt werden. Wichtige Änderungen im Gegensatz zum normalen `JFrame` sind fett markiert.

```
public class AdditionFrame extends Kit {
    private DropTextField textField1 = getDropTextField();
    private DropTextField textField2 = getDropTextField();
    //...
    public AdditionFrame(ConnectionHandler handler) {
        super(handler);
        initComponents();
        initLogicComponents();
    }
    //...
}
```

Die automatisch von NetBeans generierte `main()`-Methode muss gelöscht werden.

Um `DropTextFields` zu verwenden, muss man lediglich ein neues Attribut `DropTextField` deklarieren und mit der Methode `getDropTextField` initialisieren. In der Methode `initLogicComponents()` werden die Felder auf dem `JPanel` platziert.

```
private void initLogicComponents() {

        textField1.addKeyListener(new KeyListener() {
            public void keyReleased(KeyEvent e) {
                if (LogicValidator.isInteger(textField1.getText())){
                    textField1.setForeground(Color.black);
                } else {
                    textField1.setForeground(Color.red);
                }
            }
        });

        jPanel1.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.weightx = 0.495;
        c.fill = GridBagConstraints.BOTH;
```

```
        c.gridx = 0;
        c.gridy = 0;
        jPanel1.add(textField1, c);

        //...

        c.weightx = 1;
        c.fill = GridBagConstraints.BOTH;
        c.gridwidth = 3;
        c.gridx = 0;
        c.gridy = 1;
        jPanel1.add(getDragList(new Object[]{getTitle() + "_sum"}),
                    c);

        this.setSize(160, 120);
    }
```

Um die Felder auf dem `JPanel` zu platzieren muss zunächst das Layout des `JPanels` gesetzt werden. Danach können die einzelnen Komponenten mit `GridBackConstraints` platziert werden. Schlussendlich wird mit dem Hinzufügen der `getDragList` das Objekt so gespeichert, dass es in den nächsten Baukasten gezogen werden kann. Der an die `DragList` übergebene String muss mit dem `KeyValue` des an die `Hashmap results` übergebenen Objekts übereinstimmen. Mit dem `ActionListener` für das Textfeld lässt sich realisieren, dass falsche Eingaben rot markiert werden.

```
public String execute() {
        KryptoType value1, value2;
        if(textField1.getResult() != null)
            value1 = (KryptoType)textField1.getResult();
        else
            value1 = new Z(textField1.getText());
        if(textField2.getResult() != null)
            value2 = (KryptoType)textField2.getResult();
        else
            value2 = new Z(textField2.getText());
        KryptoType result = BasicController.addition(value1,
                                                     value2);
        results.put(getTitle() + "_sum", result);
        return  value1 + " + " + value2 + " = " + result.toString();
    }
```

Zum Schluss muss noch die `execute()`-Methode überschrieben werden. In ihr werden die Werte für die Berechnung dem `Controller` übergeben und das berechnete Ergebnis zurückgegeben.

4.1.2 Hinzufügen der erstellten Frames zum Menü

Hierzu muss in der Klasse `KryptoProjektView` dem Menü ein neues `Item` hinzugefügt werden. Mit Rechtsklick auf das neu hinzugefügte `Item` öffnet sich das Kontextmenü, in welchem man auf `Events -> Action-> actionPerformed` geht.

Danach kann man im Quellcode seinen neuen `Frame` dem Menü hinzufügen. Dies sollte folgendermaßen aussehen:

```
private void
additionMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
        Kit kit = new AdditionFrame(handler);
        kit.setVisible(true);
        desktop.add(kit);
}
```

4.1.3 Sprachfiles verwenden

In diesem Programm ist es möglich mehrere Sprachen einzubinden, welche in einem XML-File definiert wurden. Diese finden sich im `languageFiles`-Ordner des Projekts. Näheres zum Aufbau des Files findet man in der Klasse `XMLReader`.

Dass ein neu hinzugefügtes `Item` die Sprachunterstützung benutzt, muss lediglich in der Klasse `KryptoProjektView` in der Methode `initializeControlsLanguage()` der Text des Items neu gesetzt werden. Dies geschieht mit:

```
additionMenuItem.setText(xml.getTagElement("KryptoView",
"additionMenuItem"));
```

Die beiden an `getTagElement()` übergebenen Strings repräsentieren die Nodes innerhalb des XML-Files, über die die Übersetzung des Objekts ausgelesen werden kann.

Selbiges gilt für die Sprachunterstützung bei den `Frames/Komponenten` und `Exceptions` für/in innerhalb der Baukästen. Lediglich der Aufruf sieht etwas anders aus.

```
return Kit.xmlReader.getTagElement("EncodeHammingCodeFrame",
"EncodedWord") + result.getEncodedWord();
```

Zusätzlich muss die XML-Datei um den Eintrag für das neue `MenuItem` erweitert werden.

4.2 Controller

Für jede neue Funktionalität muss eine spezifische `Controller`-Klasse geschrieben werden. Sie ist dafür verantwortlich, dass die Eingaben aus der `View` korrekt an das `Model` weitergegeben werden. Weiterhin bildet der `Controller` die Schnittstelle mit der die berechneten Ergebnisse aus dem `Model` zurück an die `View` gegeben werden.

4.2.1 Erstellen einer Controller Klasse

Die neue Controllerklasse soll im Paket `Controller` abgelegt werden. In ihr werden statische Methoden definiert, welche die Weiterleitung übernehmen. Hier ein kleines Beispiel für den `HammingCode`:

```
public class CoderController {

    //...
    public static HammingCode calculateHammingSyndrom(HammingCode
                  hc) {
        hc.calculateSyndrom();
```

```
        return hc;
    }
}
```

4.2.2 Verwenden des Controllers

In nachfolgenden Abschnitt wird veranschaulicht wie der Controller in der `execute()`-Methode eines `Kits` eingebunden wird.

```
KryptoType result = BasicController.addition(value1, value2);
results.put(getTitle() + "_sum", result);
return value1 + " + " + value2 + " = " + result.toString();
```

In diesem Code-Abschnitt wird über den BasicController die addition()-Methode aufgerufen, welche ein KryptoType result zurückgibt. Dieses Ergebnis muss in die Hashmap results eingetragen werden. Anschließend wird das berechnete Ergebnis zurückgegeben.

4.3 Exception-Handling

Im Programm werden `ClassCastExceptions` und `NullpointerExceptions`, sowie die allgemeinen `Exceptions` im `Executor` abgefangen. Baukasten- oder algorithmusspezifische `Exceptions` warden bis zur `execute()`-Methode des einzelnen Bausteins durchgereicht, dort abgefangen und die Fehlermeldung zurückgegeben, somit in dem `Result`fenster ausgegeben:

```
try {
     if (textField1.getResult() != null) {
         HammingCode result = CoderController.encodeHammingCode(
                             (HammingCode) textField1.getResult());
         results.put(getTitle() + "_hcElem", result);

         return Kit.xmlReader.getTagElement(
                "EncodeHammingCodeFrame", "EncodedWord") +
                result.getEncodedWord();
     } else {
         return Kit.xmlReader.getTagElement("HammingFrames",
                "NoHammingCodeElement");
     }
} catch (RuntimeException e) {
     return Kit.xmlReader.getTagElement("HammingCode",
            e.getMessage());
}
```

5 Zukünftige Aufgaben

Im Folgenden werden die Aufgaben aufgelistet, die in den nächsten Semestern zu erledigen sind. Näheres zu den einzelnen Punkten ist in der Anforderungsspezifikation zu finden oder beim Auftraggeber zu erfragen:

- Netzwerkunterstützung
- administrative Aufgaben
- Erweiterung der Kodierung- und Verschlüsselungsalgorithmen
- Übersetzung der Sprachdatei in weitere Sprachen
- Einbindung des Sprachfiles durch z.B. Optionsfenster (bis jetzt ist der Pfad zum XML-File hardcodiert)