

## Inhaltsverzeichnis

<b>1</b>	<b>Rahmenbedingungen und Zielstellung</b>	<b>2</b>
<b>2</b>	<b>Entwicklungsumgebung</b>	<b>2</b>
2.1	E(fx)lipse . . . . .	2
2.2	Package SmartcardIO . . . . .	2
2.3	JCIDE . . . . .	3
2.4	SceneBuilder . . . . .	4
2.5	GitHub . . . . .	5
<b>3</b>	<b>Programmcode</b>	<b>5</b>
3.0.1	Geldkarte . . . . .	5
3.0.2	Ticketsystem . . . . .	7
3.0.3	Verwaltung der Personendaten . . . . .	10
<b>4</b>	<b>Zusammenfassung &amp; Fazit</b>	<b>13</b>

## 1 Rahmenbedingungen und Zielstellung

In diesem Projekt geht um die Realisierung eines Smartcard-basierten Ticketsystems für öffentliche Verkehrsmittel. Am Beispiel der U-Bahn soll es möglich sein Tickets nicht nur auf der Karte zu speichern, sondern auch die Bezahlung vorzunehmen. Hierzu kann der Fahrgast über die integrierte Geldkarte bezahlen oder mittels gesammelter Bonuspunkte für Vielfahrer. Des Weiteren soll es möglich sein den Fahrgast auf seiner Fahrt zu kontrollieren. Wurde das Ticket bereits beim Einlass kontrolliert und bestätigt, bleibt das Risiko das die Karte entwendet wurde. Ein Kontrolleur kann so während der Fahrt die Person anhand ihrer hinterlegten Daten und einem Bild, welches aus Platzgründen als Binärbild gespeichert wird, identifizieren. Aus Sicherheitsgründen wird die Übertragung verschlüsselt, somit ist es nicht möglich durch einwirken Dritter das Verhalten der Karte zu verändern.

## 2 Entwicklungsumgebung

Nachfolgend werden alle eingesetzten Hilfsmittel zur Erstellung der On- und OffCard Anwendungen aufgeführt und erklärt. Hauptrolle spielen hierbei JCIDE für die Entwicklung der OnCard Applets und Eclipse zur Erstellung der OffCard Anwendung auf Basis von JavaFX.

### 2.1 E(fx)clipse

Um die OffCard Anwendung mit JavaFx entwickeln zu können wurde von uns Eclipse verwendet. Da in der aktuellen Version noch keine Integration von JavaFX durchgeführt wurde, kann es mit Hilfe des Plugins e(fx)clipse um die nötige Funktionalität erweitert werden.

Die Vorzüge von JavaFX liegen unter anderem im neuartigen Aufbau der Benutzeroberfläche. Diese wird als FXML-Datei aufgebaut und ist somit isoliert vom restlichen Programmcode. Zusätzlich können die UI-Elemente über CSS-Datei optisch verändert werden. Auch hier bietet e(fx)clipse eine sehr attraktiven FXML und CSS-Editor, welcher Vorschläge gibt welche Attribute angewendet werden können, um ein Element zu definieren.

### 2.2 Package SmartcardIO

Das Package SmartcardIO definiert eine Java-API für die Kommunikation mit SmartCards. Es ermöglicht den Java-Anwendungen mit Applets der SmartCards zu interagieren, um so

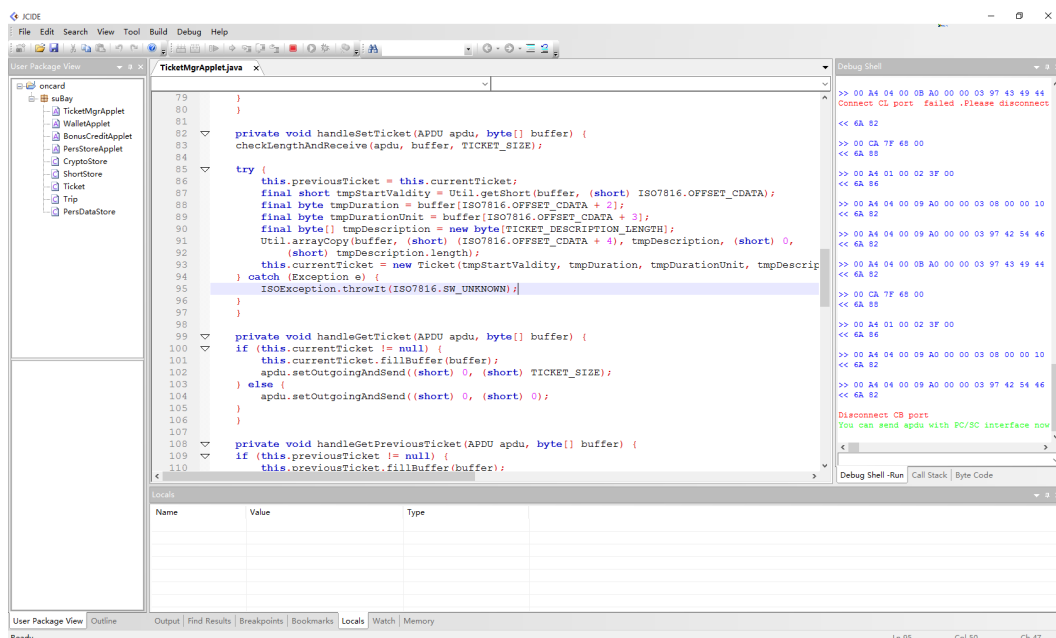
Daten auf der Karte zu speichern und abzurufen. Eine *TerminalFactory* erkennt alle angeschlossenen Terminals und stellt eine Verbindung zur Karte her.

Über den Klasse *CommandAPDU* lassen sich dann die zu übermittelnden Daten festlegen, inklusiver aller erforderlichen Parameter (CLA, INS, ...). Das sSenden des Befehls geschieht über die verbundene Karte und der Funktion *getBasicChannel().transmit()*.

War das Senden erfolgreich, erhält man die *ResponseAPDU* und kann diese über *getData()* auswerten.

## 2.3 JCIDE

JCIDE ist eine Entwicklungsumgebung, die speziell für die Javacard-Programmiersprache entwickelt wurde. Das Entwicklungskit erlaubt es schnell und einfach JavaCard Applets mit den integrierten Tools zu erstellen. Der Grund für die Verwendung der IDE liegt in der Bereitstellung eines virtuellen Kartenlesegerätes. Nach dem Übersetzen des Applets erlaubt es die Anwendung die resultierende CAP-Datei automatisch mit dem Simulator zu verknüpfen. Das Verhalten ist gleich einer echten JavaCard. Über das virtuelle Kartenlesegerät, können alle PC/SC kompatiblen Anwendungen mit dem Smartcard-Simulator kommunizieren.



**Abbildung 1:** JCIDE mit geöffneter Shell

JCIDE wird von JAVACOS Technologies<sup>1</sup> entwickelt und auf einen aktuellen Stand gehalten. Allerdings ist die Installation nur Windows-Systemen vorbehalten.

Weitere Vorteile bietet der integrierte Debugger, mit dem es möglich ist an jeder beliebigen Stelle Breakpoints zu setzen und somit schrittweise die Belegung aller Variablen zu zeigen. Das Ausführen von Befehlen über die Debug-Shell ist ebenfalls möglich und ähnelt der Ausführung in Eclipse mit JCOP-Plugin.

Ein zusätzliches Tool (ebenfalls im Paket enthalten) ermöglicht die vereinfachte Kommunikation mit der Karte. Das als „PyApuTool“ bezeichnete Programm lädt die CAP-Datei aus dem Projektorder und zeigt alle gefundenen AID's auf. Somit entfällt die Eingabe des select-Befehls. Weiterhin existiert eine Ansicht, welche den send-Befehl übersichtlich aufteilt. Für jedes Byte existiert ein eigenes Eingabefeld. Diese Hilfe beugt Fehleingaben vor und speichert zusätzlich alle getätigten Operationen.

## 2.4 SceneBuilder

Der SceneBuilder unterstützt das Erstellen der FXML-Dateien zum Aufbau der Benutzeroberfläche. Alle unterstützten JavaFX UI-Elemente sind integriert und lassen sich per „Drag and Drop“ zur Bearbeitung in die eigene GUI einfügen. Die Darstellung wird permanent aktualisiert, somit ist jederzeit die konkrete Benutzeroberfläche ersichtlich. Ferner lassen sich alle Attribute zum Ausrichten und beeinflussen der einzelnen Elemente direkt im Editor bearbeiten. Das Hinzufügen von IDs erleichtert zusätzlich den späteren Zugriff im Programmcode. Durch Annotations (@FXML) über den jeweiligen Variablen schafft die benötigte Verbindung.

---

<sup>1</sup><http://www.javacos.com/developmentkit.php>

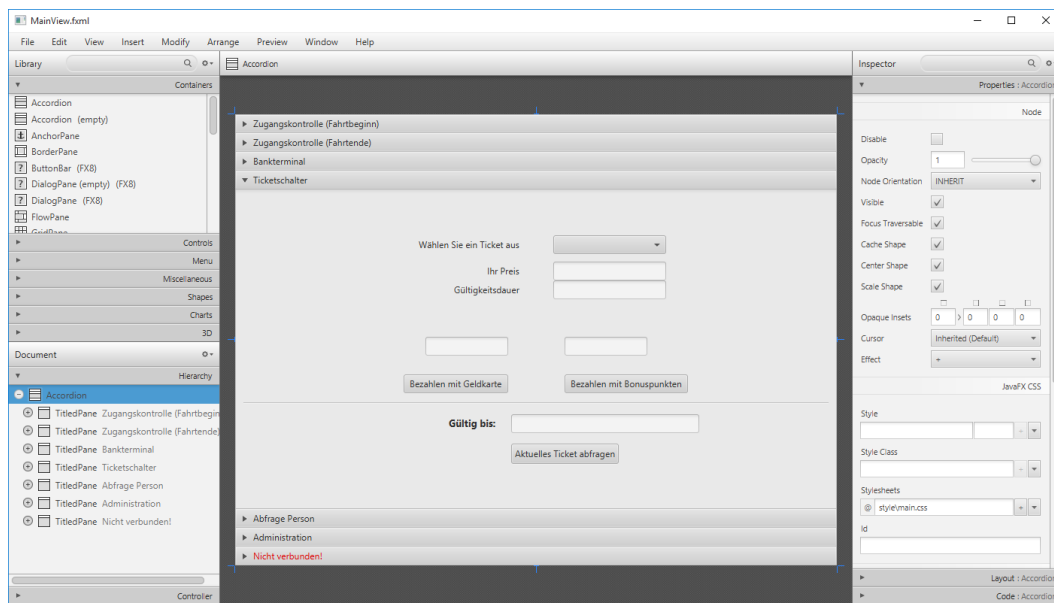


Abbildung 2: SceneBuilder

## 2.5 GitHub

Alle Programmteile sind öffentlich auf GitHub<sup>2</sup> zu finden. Neben den On- und OffCard Anwendungen ist auch diese Dokumentation zu finden. Somit ist das Projekt für jeden zugänglich und verständlich.

## 3 Programmcode

In diesem Punkt geht es um die Erklärung der einzelnen Bestandteile des Projektes. Unterteilt in einen OnCard und einen OffCard Bereich werden die nötigen Schnittstellen erläutert, sowie die Funktionsweise der Programme erklärt.

### 3.0.1 Geldkarte

Die Geldkarte dient dem Bezahlen der Tickets und musste somit vor dem Ticketsystem realisiert werden. In der ersten Phase genügte es die grundlegenden Funktionen zu implementieren, hierzu gehört das Aufladen, Abbuchen sowie das Abfragen des Kontostandes. Im

---

<sup>2</sup><https://github.com/philippsied/smartcard-course>

späteren Verlauf gehört die Integration einer Verschlüsselungsmethode, um die Aufladung durch Unbefugte zu vermeiden.

**OnCard** Zum Speichern des Geldbetrages wurde ein short-Wert definiert. Somit ist es möglich maximal 65536 Cent zu speichern. Dieser Betrag reicht für die Geldkarte vollkommen aus, da ein Verlust der Karte, auch den Verlust des Geldbetrages bedeuten würde.

**Tabelle 1:** Klassenbyte und AID

<b>CLA</b>	E0
<b>AID</b>	FD 75 42 61 79 57 61 6C 6C 65 74

**Tabelle 2:** Funktionsübersicht

<b>Daten</b>	<b>Größe</b>	<b>Funktion</b>	<b>INS</b>
Geldbetrag	short	aufladen	0x10
		abbuchen	0x20
		prüfen	0x30

Bei dem Abbuchen eines Geldbetrages wird geprüft, ob der angeforderte Betrag mindestens dem aufgeladenen Guthabens entspricht. Bei einem zu hohen Geldbetrag wird die Aktion mit einer Exception (0x6A83) beendet. Auch das Überschreiten des maximalen Guthabens wird durch eine Exception (0x6A84) abgefangen.

**OffCard** Die OffCard-Anwendung zum Aufladen der Geldkarte fällt entsprechend der wenigen Funktionen des Applets sehr klein aus. Über eine ComboBox lässt sich ein entsprechender Betrag zum Aufladen schnell auswählen und über die Schaltfläche *Aufladen* zur Karte übertragen.

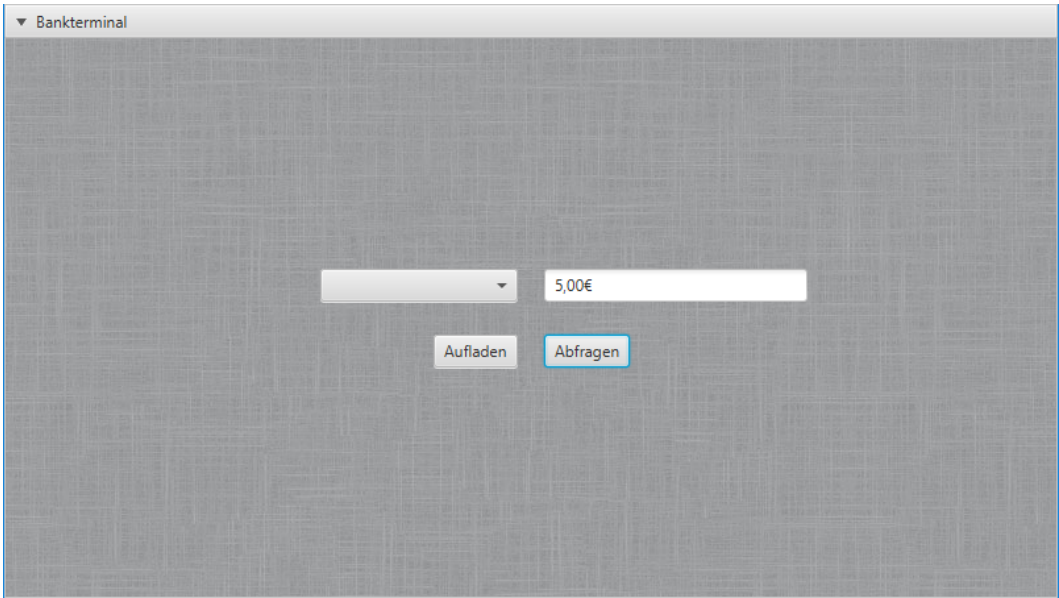


Abbildung 3: Administration der Personendaten

Um den short-Wert in eine Byte-Folge zu konvertieren wurde mit der Klasse *ByteBuffer* gearbeitet. Sie bietet die Möglichkeit beliebige Datentypen in eine Art Stapelspeicher abzulegen und als eine Byte-Folge bereitzustellen. Gleiches gilt für das entgegennehmen der Daten von der Karte. Die gelesenen Byte werden in den *ByteBuffer* abgelegt und entsprechend der Datentypen herausgenommen. Zu achten ist auf die korrekte Reihenfolge der Informationen.

3.0.2 Ticketsystem

Das Ticketsystem ist fundamentale Grundlage für dieses Projekt. Nach der Implementierung der Geldkarte stehen der Entwicklung des Systems keine Hürden mehr im Wege. Des Weiteren soll durch die spätere Integration eines Bonussystems ein zweiter Zahlungsweg genutzt werden können.

**OnCard** Auf der Karte wird hierzu eine Beschreibung des Tickets, sowie die nötigen Zeiten für den Gültigkeitszeitraum des Tickets gespeichert.

Tabelle 3: Klassenbyte und AID

CLA	E0
AID	FD 75 42 61 79 54 69 63 6B 65 74 4D 67 72

Zum speichern einer Fahrt wird eine weitere Struktur benötigt. Diese hält alle Daten zum Antritt der Fahrt, wie den Zeitpunkt und den Namen der Station. Alle gespeicherten Zeiten werden bedingt durch ihre Größe (Unix-Zeitstempel) in zwei short-Variablen gespeichert. Die Aufteilung der int-Variabel findet allerdings in der OffCard-Anwendung statt.

Tabelle 4: Add caption

Daten	Größe	Funktion	INS
<b>Ticket</b>		Aktuelles Ticket setzen	0x10
Beginn_h	short	Aktuelles Ticket abfragen	0x20
Beginn_l	short	Vorheriges Ticket abfragen	0x40
Ende_h	short		
Ende_l	short		
Beschreibung	30 Byte		
<b>Fahrt</b>		Fahr - Startzeit setzen	0xA0
Beginn_h	short	Fahrt - Endzeit setzen	0xB0
Beginn_l	short	Aktuelle Fahrt anzeigen	0xC0
Station	30 Byte		

**OffCard** Die OffCard-Anwendung fällt bei diesem System größer aus. Wie in Abbildung 4 zu sehen, wird anhand einer *ComboBox* das gewünschte Ticket ausgewählt. Die restlichen Felder geben zahlreiche Informationen zum Ticket selbst, bzw. die Zahlungsmöglichkeiten mit dem jeweiligen vorhandenen Guthabens. Ist genügend Guthaben auf der Geldkarte vorhanden, so wird dem Fahrgast zugleich das neue Guthaben (nach dem Kauf) angezeigt. Gleiches gilt für die Bezahlung mit Punkten.

Bei Auswahl der Schaltfläche *Bezahlen mit Geldkarte* wird zunächst auf das Applet der Geldkarte zugegriffen und der Kontostand kontrolliert. Reicht dieser aus, wird der entsprechende Geldbetrag abgeboben. War der Vorgang erfolgreich und die ResponseAPDU signalisiert dies, wird das Ticket generiert. Der Gültigkeitszeitraum und die Beschreibung des Tickets (Kurzstrecke, etc.) wird dann zur Karte als eine Einheit gesendet. Jederzeit kann der Fahrgast über die Schaltfläche *Aktuelles Ticket abfragen* den Gültigkeitszeitraum der gekauften Fahrkarte abfragen.



▼ Ticketschalter

Wählen Sie ein Ticket aus: Kurzstrecke

Ihr Preis: 1,80€ oder 270Punkte

Gültigkeitsdauer: 20 Minutes

5,00€ -> 3,20€

0P

Bezahlen mit Geldkarte

Bezahlen mit Bonuspunkten

Gültig bis: Kein Ticket vorhanden

Aktuelles Ticket abfragen

Abbildung 4: Ticketschalter

Zwei weitere OffCard-Anwendungen simulieren die Zugangskontrolle. Bei Fahrtantritt wird geprüft ob ein gültiges Ticket vorhanden ist. Ist dies der Fall, so „öffnet“ sich die Zugangskontrolle und der Fahrtbeginn, sowie der Name der Station wird auf der Karte hinterlegt. Bei dem Beenden der Fahrt wird am Ausgangskontrolle die Fahrtzeit geprüft und mit der Gültigkeitsdauer des Tickets verglichen.

▼ Zugangskontrolle (Fahrtbeginn)

Bitte passieren Sie die Schranke.

Karte einlegen

Abbildung 5: Zugangskontrolle

### 3.0.3 Verwaltung der Personendaten

Die Personendaten sollen der Kontrolle der Fahrgäste dienen. Hierzu werden im Vorfeld entscheidende Daten zur Person erhoben. Hierzu zählen unter anderem der Name sowie das Geburtsdatum, aber auch ein Bild zur Identifikation soll gespeichert werden.

**OnCard** Für die Speicherung der Daten auf der Karte wurde eine eigene Klasse erstellt, welche die erforderlichen Personeninformationen hält.

**Tabelle 5:** Klassenbyte und AID

<b>CLA</b>	E0
<b>AID</b>	FD 75 42 61 79 50 65 72 73 53 74 6F 72 65

**Tabelle 6:** Funktionsübersicht

Daten	Größe (Byte)	Funktion	INS
Vorname	10	setzen	0x1A
		abfragen	0x1B
Nachname	10	setzen	0x2A
		abfragen	0x2B
Geburtsdatum	10	setzen	0x3A
		abfragen	0x3B
Wohnort	15	setzen	0x4A
		abfragen	0x4B
Straße	25	setzen	0x5A
		abfragen	0x5B
Telefonnummer	15	setzen	0x6A
		abfragen	0x6B
Bild	6750	setzen	0x7A
		abfragen	0x7B

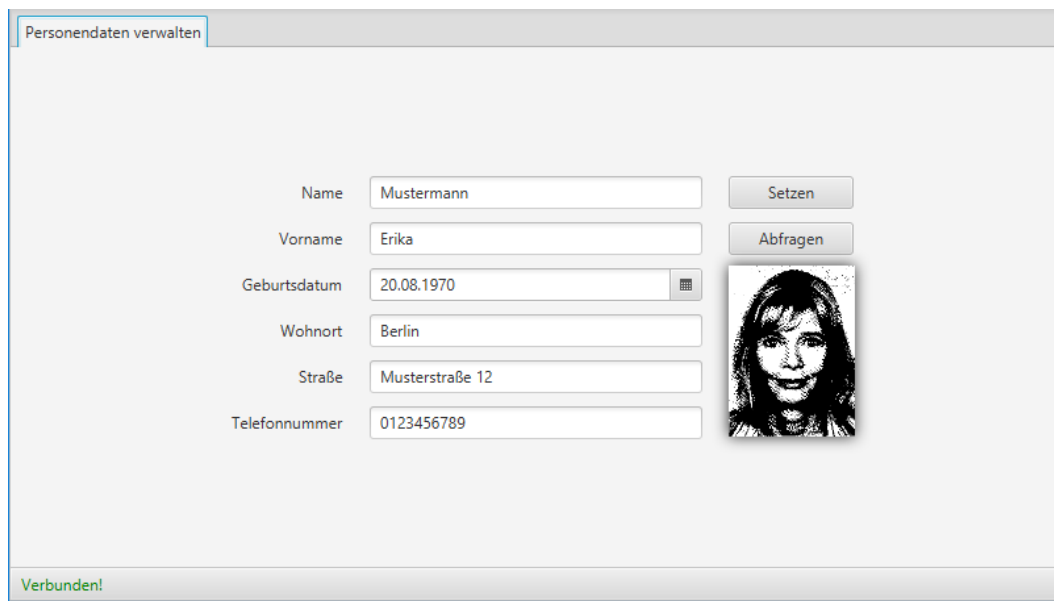
In Tabelle 6 sind alle Funktionen zum setzen und lesen der Daten ersichtlich. Ferner lässt sich der reservierte Speicherplatz erkennen. Alle Informationen werden durch lesen des Buffers unter Nutzung der Funktion *Util.arrayCopy()* entgegengenommen. Eine Übertragung einer längeren Zeichenfolge zur Karte ist möglich, wird aber bedingt durch den reservierten Speicherplatz abgeschnitten.

Schwieriger gestaltete sich die Übertragung des Bildes. Dieses kommt als 6750 Byte großes Array zur Karte. Da der Buffer nur 255 Byte entgegennimmt, muss mit der Funktion *apdu.receiveBytes()* Bereichsweise das Bild in den Speicher geschrieben werden. Um die Übertragung der hohen Anzahl Bytes zu ermöglichen, wird der Sendevorgang im Extended-Mode initialisiert. Dies bedeutet, dass für das Feld LC (Länge des Datenbytes) nicht 1 Byte zur Verfügung steht, sondern 3 Byte. Zur Signalisierung der APDU wird das eigentliche Byte auf 0x00 gesetzt gefolgt von zwei weiteren Byte, welche die neue Länge des Datenfeldes repräsentieren.

```
1 // Aktuell gelesene Bytes
2 short read = apdu.setIncomingAndReceive();
3
4 // Gesamtlänge
5 short footage = apdu.getIncomingLength();
6
7 // Ersten Bereich in Speicher schreiben – beginnend vom Offset
8 personaldata.setPicPart(buf, (short)(ISO7816.OFFSET_EXT_CDATA), (short) 0,
    read);
9
10 // Zeiger im Speicher verschieben und restliche Bereiche schreiben
11 while(read < footage) {
12     short read_now = apdu.receiveBytes((short) 0);
13     personaldata.setPicPart(buf, (short) 0, read, read_now);
14     read += read_now;
15 }
```

Das zurücksenden der Daten zur OffCard-Anwendung ist im Gegensatz stark vereinfacht. Sofern das Interface *ExtendedLength* zum Applet hinzugefügt wurde, kann mit der Funktion *apdu.sendBytesLong()* das komplette Bild (6750 Byte) gesendet werden.

**OffCard** Das Problem der OffCard-Anwendung bestand in der Verkleinerung des Bildes für die Karte, da nur begrenzter Speicherplatz zur Verfügung steht. Für das Verkleinern des Bildes wurde die Klasse *BufferedImage* verwendet. Hierzu kann nicht nur das Bild in Höhe und Weite verkleinert werden, sondern auch in ein Binärbild umgewandelt werden. Hierzu muss im Konstruktor der *BufferedImage* der Parameter *TYPE\_BYTE\_BINARY* angegeben werden. Die Konvertierung des Bildes erfolgt automatisch.



The screenshot shows a web interface titled "Personendaten verwalten". It contains several input fields for personal data, each with a corresponding label on the left. The fields are: "Name" (Mustermann), "Vorname" (Erika), "Geburtsdatum" (20.08.1970), "Wohnort" (Berlin), "Straße" (Musterstraße 12), and "Telefonnummer" (0123456789). To the right of the "Name" and "Vorname" fields are buttons labeled "Setzen" and "Abfragen" respectively. Below the "Geburtsdatum" field is a small calendar icon. To the right of the "Geburtsdatum" field is a small portrait photo of a woman. At the bottom left of the interface, there is a green status message that says "Verbunden!".

Feld	Wert
Name	Mustermann
Vorname	Erika
Geburtsdatum	20.08.1970
Wohnort	Berlin
Straße	Musterstraße 12
Telefonnummer	0123456789

Abbildung 6: Administration der Personendaten

Nach dem Betätigen des Send-Buttons werden alle Felder, sowie das Bild einzeln zur Karte gesendet. Gleiches gilt für die Abfrage der Daten.

## 4 Zusammenfassung & Fazit