DEDICATION

To my dear parents and family for their continuous support throughout this course.
ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for giving me the strength and ability to carry out this project.

I would also like to thank my supervisor, Dr. H. A. Ouma, for being a source of guidance throughout the duration of the project.

My appreciation goes out to my classmates for their suggestions and opinions on the project.

Lastly, I would like to appreciate my family for their continuous support.
DECLARATION AND CERTIFICATION

This is my original work and has not been presented for a degree award in this or any other university.

é é é é é é é é é é é é é é é é é é ..

MUCHIRI ROY BRIAN WAMBUGU

F17/1753/2006

This report has been submitted to the Department of Electrical and Information Engineering, The University of Nairobi with my approval as supervisor:

é é é é é é é é é é é é é

Dr. H. A. Ouma

Date: é é é é é é é é é
# Table of Contents

DEDICATION .............................................................................................................. 2  
ACKNOWLEDGEMENTS .......................................................................................... 3  
DECLARATION AND CERTIFICATION .................................................................. 4  
Table of Contents .................................................................................................... 5  
List of Figures .......................................................................................................... 7  
ABSTRACT ................................................................................................................. 8  

1 INTRODUCTION ...................................................................................................... 9     
1.1 Problem Definition ............................................................................................... 9  
1.2 Objectives ............................................................................................................. 9  
1.3 Justification .......................................................................................................... 10  
1.4 Scope .................................................................................................................... 10  

2 LITERATURE REVIEW ............................................................................................ 11  
2.1 SMART CARDS .................................................................................................. 11     
2.1.1 Classification of smart cards ........................................................................... 12  
2.1.1.1 Memory cards ............................................................................................ 12  
2.1.1.2 Microprocessor cards ................................................................................. 12  
2.1.1.3 Contact smart cards .................................................................................. 13  
2.1.1.4 Contact-less smart card ............................................................................ 13  
2.1.2 Smart card standards and specifications ......................................................... 13  
2.1.3 Smart Card Communications .......................................................................... 15  
2.1.4 Summary of operation of a microprocessor smart card ................................... 15  
2.1.5 Areas of use of smart cards ............................................................................. 15  
2.2 Attendance Register Systems .............................................................................. 17  
2.2.1 Components of an Attendance Register System ............................................. 17  
2.2.2 Attendance register systems incorporating smart cards .................................. 17  

3 DESIGN .................................................................................................................. 18  
3.1 System specifications ........................................................................................... 18  
3.1.1 System description ........................................................................................... 19  
3.1.1.1 Registration ............................................................................................... 19  
3.1.1.2 Recording student attendance .................................................................... 19  
3.1.1.3 Report generation ....................................................................................... 19  
3.1.2 System Flowcharts ........................................................................................... 20  
3.1.2.1 Registration Process .................................................................................. 20  
3.1.2.2 Recording of student attendance ............................................................... 21
3.1.2.3  Reporting ......................................................................................................................... 22
3.2  Requirements .......................................................................................................................... 23
3.3  Software design ........................................................................................................................ 23
4  ANALYSIS AND RESULTS ....................................................................................................... 24
  4.1  Registration .......................................................................................................................... 24
  4.2  Recording student attendance .............................................................................................. 27
  4.3  Report generation .................................................................................................................. 28
5  CONCLUSIONS AND RECOMMENDATIONS ........................................................................... 29
  5.1  Conclusions .......................................................................................................................... 29
  5.2  Recommendations for future work ....................................................................................... 29
APPENDIX ..................................................................................................................................... 31
List of Figures

Figure 2.1: Sizes of ID-1, ID-00 and ID-000 formats................................................................. 11
Figure 2.2: Contact smart card connector .................................................................................. 13
Figure 3.1: Flowchart of the overall attendance register system............................................... 18
Figure 3.2: Flowchart detailing the registration process.............................................................. 20
Figure 3.3: Flowchart detailing the recording of student attendance........................................... 21
Figure 3.4: Flowchart detailing the reporting process ................................................................. 22
Figure 4.1: Main User Interface.................................................................................................. 24
Figure 4.2: Edit Lecturer Details ............................................................................................... 24
Figure 4.3: Edit student details .................................................................................................. 25
Figure 4.4: Edit course details ................................................................................................. 25
Figure 4.5: Editing lesson details .............................................................................................. 26
Figure 4.6: Writing student details to a smart card................................................................. 26
Figure 4.7: Student lesson attendance interface ......................................................................... 27
Figure 4.8: Lesson attendance report generation interface.......................................................... 28
ABSTRACT

This report handles the design and implementation of a smart card based attendance register system.

The main aim of the project is to build a system that eases the process of monitoring how the students in the department attend the lectures. An analysis of the current system is covered which enables one to identify the most critical information in the system.

The report also analyses the system designed, in terms of registration of students, issuance of smart cards and usage of smart cards for recording attendance.

Finally, it is seen that such a system is feasible to enable implementation in the department.
1 INTRODUCTION

A smart card is a device with an integrated microprocessor used to store data and perform computations relating to the access of data stored on it. The microprocessor is included on the device to restrict access to the device to selected users.

The smart card comprises a SIM card sized microprocessor which is mounted on a larger plastic card. This arrangement eases the use and handling of the device, in addition to extending the life of the smart card owing to the protection offered by the plastic housing.

Smart cards are usually programmed with the data required for when the smart card is used later on.

1.1 Problem Definition

Monitoring of access to facilities is a high priority in many organizations. This is done for a variety of reasons such as:

- For security purposes i.e. to restrict access to certain users.
- For logging purposes i.e. to show which people had access to the facilities at a given time.

The most common method of tracking user movement around facilities has been paper based. In this method, users are provided with a form on which they record their details when they enter a certain area. Additionally, some organizations employ the use of ID cards with which the users identify themselves.

Most of these systems are wrought with problems, such as:

- Difficulty in analysis of the data e.g. to find if a particular user was present in a facility at a given time.
- Storage of records, especially in paper based systems.
- Authentication even in the case of issued ID cards is cumbersome as it usually requires human input.

Thus, the aim of the project is to design and build a computerized system which will be able to perform the functions of an attendance register easily.

1.2 Objectives

The objectives of the project are to design and implement a smart-card based attendance register. Specifically, the system should be able to accomplish the following functions:
1. To provide a means of capturing the students' details, storing them and providing a smart card to the student for purposes of identification.

2. To provide a means of capturing and recording a student's attendance details when they attend a class.

3. To generate reports detailing a student's attendance details over a certain period of time (typically over a semester)

1.3 **Justification**

The current system employed to meet the requirements of logging student attendance is cumbersome as it involves huge amounts of paper to record all the details of the students. Also, the current system poses a problem when analysis of student attendance is to be performed.

The proposed system will record details of student attendance electronically. This is advantageous as it will release the resources that are used to maintain the current system (paper, printing costs etc.). Also, it will provide an easier way of analyzing student attendance as most of these operations will be offloaded to a computer.

1.4 **Scope**

The project will be limited to only performing authentication and registration of students using smart cards. This will be achieved through use of software on a host device to communicate with the smart card and perform the necessary computations.

The project does not involve the production of smart cards to be used in the system.
2 LITERATURE REVIEW

2.1 SMART CARDS

Smart cards are electronic devices comprising an embedded chip and external circuitry to enable communication between the chip and other electronic devices. The small physical size of the smart cards necessitates the housing of the card in a larger card, thus providing physical protection and easing the use and handling of the card. The size of the larger card is governed by the ISO 7810 standard while the position of the contacts on the card is governed by the ISO 7816-2 standard. The standard defines three formats of the larger card, namely:

1. ID-1 format
2. ID-00 format
3. ID-000 format

Figure 2.1 shows the dimensions of the above formats.

Figure 2.1: Sizes of ID-1, ID-00 and ID-000 formats
Primarily, smart cards have the capacity for large data storage (more than 256 KB), thus giving them a significant advantage over their magnetic stripe (1000 bits) and bar code card counterparts. In addition, some smart cards have additional security features which offer protection against unauthorized data access. Thus, the smart card can only be used on authorized terminals.

2.1.1 Classification of smart cards

The chip can contain either a microprocessor or a memory card, depending on the use of the smart card.

2.1.1.1 Memory cards

Memory cards are used in situations where simple identification of the parties is the main concern. In this case, the most significant action of the system is to read the data on the smart card. Thus, some cards have little or no security measures in place to control access to the data on them.

2.1.1.2 Microprocessor cards

Microprocessor cards have a CPU embedded in the smart card chip. The CPU is a fully fledged system with its own operating system. This enhanced capability enables features such as running of other programs (such as encryption modules) on the smart card. This greatly increases the scope of use of the smart card since programs can be loaded arbitrarily and run on the smart card. Additionally, the enhanced processor power means that this smart card offers a higher degree of security due to its ability to control data access by means of software running on the smart card.

The external circuitry of the chip in the smart card is dependent on the type of smart card, namely:

1. Contact-less smart card
2. Contact smart card
2.1.1.3 **Contact smart cards**

In these types of smart cards, the embedded chip is housed in a connector which has electrical contacts that are bonded to it. The connector provides electrical connectivity to the chip via the various contacts. A basic contact smart card connector is shown in Figure 2.2.

![Contact smart card connector](image)

*Figure 2.2: Contact smart card connector*

The electrical connector provides:

1. Voltage supply to the chip ($V_{cc}$).
2. Clock signal for the chip and for timing of data transmission ($CLK$).
3. Input and output line for data transmission ($I/O$).
4. Ground connection ($GRD$).

2.1.1.4 **Contact-less smart card**

In these types of smart cards, the electrical connector of the contact type smart card is replaced with a RF (Radio Frequency) interface which facilitates electrical connectivity. Interaction with contact-less smart cards is based on RFID (Radio Frequency Identification) techniques. Power for the chip is obtained by inductive coupling, whereby the smart card draws its energy from the magnetic field set up by the card terminal. Data transmission occurs via high frequency modulation techniques.

2.1.2 **Smart card standards and specifications**

Due to the common use and proliferation of smart cards, a system is needed to control the various aspects of smart cards in order to ensure interoperability. The system in use to govern these is the ISO 7816 family of standards. These standards govern the following:

1. ISO 7816-1: This section describes the physical characteristics of integrated circuit cards. This section is of importance to card manufacturers as it determines the physical limits of the smart cards.
2. ISO 7816-2: This section describes the dimensions and locations of the contacts. It
also defines the number and function of the contacts. The contacts are defined as in Table 1:

Table 1: List of smart card contacts

<table>
<thead>
<tr>
<th>Contact</th>
<th>Designation</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Vcc</td>
<td>Power connection through which operating power is supplied to the microprocessor chip in the card.</td>
</tr>
<tr>
<td>C2</td>
<td>RST</td>
<td>Reset line through which the terminal can signal to the smart card's microprocessor chip to initiate its reset sequence of instructions.</td>
</tr>
<tr>
<td>C3</td>
<td>CLK</td>
<td>Clock signal line through which a clock signal can be provided to the microprocessor chip. This line controls the operation speed and provides a common framework for data communication between the terminal and the smart card.</td>
</tr>
<tr>
<td>C4</td>
<td>RFU</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>C5</td>
<td>GND</td>
<td>Ground line providing common electrical ground between the terminal and the smart card.</td>
</tr>
<tr>
<td>C6</td>
<td>Vpp</td>
<td>Programming power connection used to program EEPROM of first generation smart cards.</td>
</tr>
<tr>
<td>C7</td>
<td>I/O</td>
<td>Input/output line that provides a half-duplex communication channel between the reader and the smart card.</td>
</tr>
<tr>
<td>C8</td>
<td>RFU</td>
<td>Reserved for future use.</td>
</tr>
</tbody>
</table>

3. ISO 7816-3: This section describes the electronic signals and transmission protocols of smart cards.

4. ISO 7816-4: This section describes the smart card commands used to interact with a smart card present in the terminal. This section is of particular importance to application software developers as it assists them to interact with a wide range of smart cards.
2.1.3 Smart Card Communications

As mentioned earlier, the methods of communicating with a smart card are defined by ISO standards, specifically the ISO 7816-3 and ISO 7816-4 standards. The ISO 7816-3 standard defines among others various parameters such as:

1. Protocol to use for communication
2. Speed of transmission of data

The ISO 7816-4 family of standards defines the structure of communications between a terminal device and a smart card after the terminal has identified the smart card as a suitable device. Communications between the terminal and the smart card use packets called APDU (application protocol data unit). These are analogous to packets in a networking setup. The terminal initiates communication with the smart card and sends commands that are encapsulated using APDUs. The smart card reads the data sent by the terminal, processes it and responds with a suitable return value. The range of commands and return values is defined by the ISO 7816-4 standards.

Due to the proliferation of smart card manufacturers, the range of smart cards and smart card operating systems is large. Thus the ISO 7816-4 standards also define the base commands that each smart card should support, giving room for manufacturers to implement extra functions for their own smart cards.

2.1.4 Summary of operation of a microprocessor smart card

Microprocessor smart cards contain a fully functional CPU embedded in them. For efficient use of the resources offered by the CPU, a tested operating system is usually loaded on it, mostly at the time of manufacture of the smart card. The smart card operating system is used to effectively manage the resources of the smart card. This OS forms a very important part of the smart card due to the immense resource constraints offered by the hardware. The small size of the smart card chip limits the amount of RAM and non-volatile storage available. Thus, efficiency is paramount in such systems.

However, this resource constrained system opens up a wide range of possibilities. For example, the smart card can be used to perform resource intensive calculations such as encryption and decryption of data. In addition, the presence of the OS enables the smart card to run auxiliary applications i.e. applications designed and installed by third parties. This makes the smart card flexible in terms of use and operation.

2.1.5 Areas of use of smart cards

Smart card usage has become very popular, mostly due to the proliferation of mobile phones. The SIM card that is used in the mobile phone is an example of a smart card. The SIM card is
used to authenticate the mobile phone to the user. The microprocessor in the SIM card is used to carry out the encryption algorithms as defined in the GSM standards. The SIM card enables running of STK (SIM Toolkit) applications e.g. M-PESA on the host phone. Additionally, the SIM card offers memory storage which is used to store phone numbers.

Smart cards are also used in mass transit systems, where authentication of the users needs to be done quickly to grant access. In such systems, contact-less smart cards have become widely adopted. This is because the user only needs to place the card near the terminal, reducing the length of the interaction with the system. Authentication is performed quickly and if necessary, data such as the amount of money still unused can be stored on the smart card. The security offered by the smart card can enable such functions since the terminal need not be connected to a central system to perform these functions. The terminal can then perform all these functions based on data obtained from the smart card. Examples of usage of smart cards being used in this way are:

1. Octopus Card system in use in Hong Kong. It is primarily used for managing the ticketing system for their various modes of transport.
2. KAPS (Kenya Airports Parking Services) system in use at the Jomo Kenyatta International Airport, Nairobi.

In some countries, smart cards are also used as identification cards for the public. The large data storage capabilities and enhanced security of the smart card lends itself to such applications. An example of this type of usage is the MyKAD system in use in Malaysia.
2.2 **Attendance Register Systems**

Attendance Register Systems are those which are designed to maintain attendance profiles of the users. Among the simplest of these systems are lists containing names of people. Each person is required to sign against his/her name to confirm attendance. These systems are usually designed and implemented to ensure attendance of the events for which the system has been designed.

2.2.1 **Components of an Attendance Register System**

These systems typically comprise the following sections:

1) **Capturing of details:** The event for which such a system is designed has the required attendees. Thus, a method of recording the details of the attendees is required prior to usage of the system. This is because all subsequent processing will be based on these details.

2) **Authentication and recording of attendance:** As certain events are not open to all people, only authenticated users should be allowed to record their attendance. Authentication is performed by checking whether the user's details are present on the list of allowed attendees. If the user is authenticated, the user is deemed to have attended the event and this is recorded in the system.

3) **Compilation and analysis:** On conclusion of the event, the data collected is analyzed. Analysis might include finding out which users missed the event, rate of attendance of the events e.t.c.

2.2.2 **Attendance register systems incorporating smart cards**

The properties of smart cards lend them to effective use in attendance register systems. Among the most common of are those that are used to monitor the movement of employees in a certain environment. The most common setup is whereby the employees are issued with RFID cards which have been programmed with their information. Additionally, a smart card reader is installed at a common point through which all the employees must pass. The employees then swipe their cards on the reader. The reader verifies the card and obtains the employee data stored on the card. Once verification is complete, the system can perform operations such as recording the time of entry or allowing access to a certain area.

Smart cards are also in use in the health industry to enable easier identification of patients. Since some health centers are located in remote areas, smart cards are useful in this area because it enables the identification system to be decoupled from the central health systems. The security offered by the smart cards enables the data stored on the card to be trustworthy.
3 DESIGN

3.1 System specifications

The attendance register system consists of three main components which inter-operate as shown in Figure 3.1:

Figure 3.1: Flowchart of the overall attendance register system
3.1.1 System description

3.1.1.1 Registration

The flow of information in this process is as defined in Figure 3.2.

The program running on the server is used to create a list of courses that are offered in the department. Details such as the course code and the name of the course are obtained in this step.

The server program is used to capture the details of a student and save them to the database. Then, the student’s details, inclusive of the courses registered, are written to a smart card which is issued to the student.

The server program is also used to capture the details of the lecturers in the department. Additionally, the details of courses handled by the lecturers are obtained in this step.

Finally, clients are then connected to the server and the necessary data for capturing student attendance data is downloaded onto them. This data includes the list of lecturers, courses and optionally lessons that have been loaded onto the server. The list of students is not transferred due to the large size of this data.

3.1.1.2 Recording student attendance

The flow of information in this process is as defined in Figure 3.3.

The program running on the client is used to create lessons for a particular lecturer. Then, the program is used to capture attendance information. Once a student enters his/her smart card into the reader, the program obtains the details stored on the card. The program verifies that the student is in the right lesson by comparing the list of courses stored on the card to the course in progress. If a match is found, the student is added to the list of students who have attended that lesson.

After the lesson has ended, the client is connected to the server and the attendance data is uploaded to the server.

3.1.1.3 Report generation

The flow of information in this process is as defined in Figure 3.4

A program running on the server lets lecturers log on to the system and view attendance details for courses they are in charge of. The program ensures that the lesson in question has already ended and then generates a list of students registered for that course, indicating whether or not they attended that lesson.
3.1.2 System Flowcharts

3.1.2.1 Registration Process

START

Course information is added to database

Lecturers’ and students’ information is added to database.

Lecturer or Student?

Is lecturer

Lecturer picks courses he/she is in charge of. Data is stored in database.

Is student

Students register for courses they are taking in the current semester.

Student information is written to smart card.

RECORDING OF STUDENT ATTENDANCE PROCESS

Figure 3.2: Flowchart detailing the registration process
3.1.2.2 Recording of student attendance

Figure 3.3: Flowchart detailing the recording of student attendance
3.1.2.3 **Reporting**

![Flowchart](image.png)

**Figure 3.4**: Flowchart detailing the reporting process
3.2 **Requirements**

The project entailed the use of smart cards to monitor attendance. Hence, a card reader and a smart card were necessary in order to implement the project. The specifications of the equipment used were as follows:

1. Smart Card Reader:
   a. Manufacturer: SCM Microsystems
   b. Model: SPR337
   c. Name: Biometric Smart Card reader
2. Smart Card:
   a. Manufacturer: Gemplus
   b. Model: Gemplus MPCOS EMV 1 byte sector

A host environment for proper usage of the smart card was necessary. Based on the equipment available, a Windows programming environment was chosen. This was because of the availability of the driver files for the smart card reader. Additionally, the Windows operating system offers smart card support out of the box, since it has implemented the PC-SC interface using its WinSCard implementation.

3.3 **Software design**

The project entailed the entry, manipulation and storage of various data for proper working of the system. Hence, a database management system was required. Additionally, a proper means of manipulating the database was required in order to achieve efficient operation of the system. Thus, the MySQL DBMS was chosen for data storage and the Java programming language was chosen for program design.
4 ANALYSIS AND RESULTS

The system responds differently depending on the state in which it is in.

4.1 Registration

The user will see the interface defined in Figure 4.1 at the time of registration:

![Main User Interface](image1)

*Figure 4.1: Main User Interface*

When adding or editing a lecturer’s details, the user will see the interface defined in Figure 4.2:

![Edit Lecturer Details](image2)

*Figure 4.2: Edit Lecturer Details*
When adding or editing a student’s details the user will see the interface defined in Figure 4.3:

![Edit Student's Details](image)

**Figure 4.3: Edit student details**

When adding or editing a course’s details, the user will see the interface defined in Figure 4.4:

![Edit Course Details](image)

**Figure 4.4: Edit course details**
When adding or editing a lesson’s details, the user will see the interface defined in Figure 4.5:

![Figure 4.5: Editing lesson details](image)

Finally, when recording a student’s details to a smart card, the user will see the interface defined in Figure 4.6:

![Figure 4.6: Writing student details to a smart card](image)
4.2 **Recording student attendance**

A lecturer will use the interface defined in Figure 4.7 to enable students to sign in for a particular lesson:

![Student lesson attendance interface](image)

*Figure 4.7: Student lesson attendance interface*
4.3 **Report generation**

A lecturer will use the interface defined in Figure 4.8 order to view the attendance of a certain lesson:

![Attendance Details](image)

*Figure 4.8: Lesson attendance report generation interface*
5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The main objective of the project, which was to develop an attendance register system that incorporates smart cards, was achieved.

The system showed the ability to store the data necessary for proper system implementation. This included identifying students and generating attendance reports for a particular lesson.

5.2 Recommendations for future work

Smart cards manufacturers continue to produce smart cards that are significantly more powerful than their predecessors. They can incorporate dedicated hardware that can be used to perform cryptographic functions easily. Additionally, smart cards have the capacity to run programs that are installed on them. Coupled with their increased memory capacities, newer smart cards can then be used to perform a bigger number of calculations at a much higher rate.

Thus, the recommendations for future work are as follows:

1. Investigate how to store and retrieve data, which is cryptographically altered, on a smart card.

2. Investigate how to protect the student’s details stored on the smart card by other means, for example the student’s fingerprint. Thus, upon inserting a smart card into the reader, the student is required to provide a fingerprint (using a fingerprint reader). The fingerprint is then passed to the smart card which upon authenticating the student releases the student’s information for processing by the terminal.

3. Investigate how to improve portability of the system. For example, the use and effectiveness of smart phones as the client computers in the system.
REFERENCES


[3.] Microsoft MSDN Documentation on smart cards


[6.] http://en.wikipedia.org/Smart_card

[7.] http://www.smartcardbasics.com


[9.] http://www.tensor.co.uk

APPENDIX

APPENDIX A: Program code

The program comprises 2 modules, namely:

a) The server module, which handles the registration and reporting processes.

b) The client module, which handles the recording of student attendance details.

Each module contains sub-modules which define:

a) The database operations and methods of data access (the data models).

b) The presentation of the data stored in the database to the system user (the view).

The modules are organized as follows:

![Module Organization Diagram]

The code handling the database operations is contained in the `attendanceregister.model` package. Sample files from the package are detailed below to illustrate the methods defined for database operations.

![Sample Files Diagram]
package attendanceregister.model;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

/**
 * Course generated by hbm2java
 */
@Entity
@Table(name = "course", catalog = "attendance_register", uniqueConstraints = {
    @UniqueConstraint(columnNames = "code")
})
public class Course implements java.io.Serializable {

    private Integer id;
    private String code;
    private String name;
    private int yearOfStudy;

    private Set students = new HashSet(0);

    public Set getStudents() {
        return students;
    }

    public void setStudents(Set students) {
        this.students = students;
    }

    private Set studentCourses = new HashSet(0);
    private Set lecturerCourses = new HashSet(0);

    public Course() {
    }

    public Course(String code, String name, int yearOfStudy) {
        this.code = code;
        this.name = name;
        this.yearOfStudy = yearOfStudy;
    }

    public Course(String code, String name, int yearOfStudy, Set studentCourses, Set lecturerCourses) {
        this.code = code;
    }
this.name = name;
this.yearOfStudy = yearOfStudy;
this.studentCourses = studentCourses;
this.lecturerCourses = lecturerCourses;
}

@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "id", unique = true, nullable = false)
public Integer getId()
{
    return this.id;
}

public void setId(Integer id)
{
    this.id = id;
}

@Column(name = "code", unique = true, nullable = false, length = 10)
public String getCode()
{
    return this.code;
}

public void setCode(String code)
{
    this.code = code;
}

@Column(name = "name", nullable = false)
public String getName()
{
    return this.name;
}

public void setName(String name)
{
    this.name = name;
}

@Column(name = "year_of_study", nullable = false)
public int getYearOfStudy()
{
    return this.yearOfStudy;
}

public void setYearOfStudy(int yearOfStudy)
{
    this.yearOfStudy = yearOfStudy;
}

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "course")
public Set getStudentCourses()
{
    return this.studentCourses;
}

public void setStudentCourses(Set studentCourses)
this.studentCourses = studentCourses;

@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "course")
public Set getLecturerCourses()
{
    return this.lecturerCourses;
}

public void setLecturerCourses(Set lecturerCourses)
{
    this.lecturerCourses = lecturerCourses;
}

@Override
public boolean equals(Object other)
{
    if (this == other)
    {
        return true;
    }
    if (!(other instanceof Course))
    {
        return false;
    }
    final Course course = (Course) other;
    if (!course.getCode().equals(this.getCode()))
    {
        return false;
    }
    return true;
}

@Override
public int hashCode()
{
    int hash = 7;
    hash = 17 * hash + (this.code != null ? this.code.hashCode() : 0);
    return hash;
}

File: Course.hbm.xml

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-/Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="attendanceregister.model.Course" table="course"
catalog="attendance_register">
        <id name="id" type="java.lang.Integer">
            <column name="id" />
            <generator class="identity" />
        </id>
        <property name="code" type="string">
            <column name="code" length="10" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>
<column name="name" not-null="true" />
</property>

<property name="yearOfStudy" type="int">
    <column name="year_of_study" not-null="true" />
</property>

<set name="students" table="student_course" inverse="true">
    <key column="course_id" />
    <many-to-many column="student_id" class="attendanceregister.model.Student" />
</set>

<set name="studentCourses" inverse="true">
    <key>
        <column name="course_id" not-null="true" />
    </key>
    <one-to-many class="attendanceregister.model.StudentCourse" />
</set>

<set name="lecturerCourses" inverse="true">
    <key>
        <column name="course_id" not-null="true" />
    </key>
    <one-to-many class="attendanceregister.model.LecturerCourse" />
</set>

class="attendanceregister.model.Student" />
</set>
</hibernate-mapping>

File: Student.java

package attendanceregister.model;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

/**
 * Student generated by hbm2java
 */
@Entity
@Table(name="student",catalog="attendance_register",
    uniqueConstraints = @UniqueConstraint(columnNames="reg_no")
)
public class Student implements java.io.Serializable {

    private Integer id;
    private String regNo;
    private String fullNames;
    private int currentYear;

    private Set courses = new HashSet(0);
@ManyToMany(cascade= CascadeType.ALL, fetch= FetchType.LAZY,mappedBy="student")
public Set getCourses()
{
    return courses;
}

public void setCourses(Set courses)
{
    this.courses = courses;
}

private Set studentCourses = new HashSet(0);
private Set studentLessons = new HashSet(0);

public Student()
{
}

public Student(String regNo, String fullNames, int currentYear) {
    this.regNo = regNo;
    this.fullNames = fullNames;
    this.currentYear = currentYear;
}

public Student(String regNo, String fullNames, int currentYear, Set studentCourses, Set studentLessons) {
    this.regNo = regNo;
    this.fullNames = fullNames;
    this.currentYear = currentYear;
    this.studentCourses = studentCourses;
    this.studentLessons = studentLessons;
}

@Id @GeneratedValue(strategy=IDENTITY)
@Column(name="id", unique=true, nullable=false)
public Integer getId() {
    return this.id;
}

public void setId(Integer id) {
    this.id = id;
}

@Column(name="reg_no", unique=true, nullable=false, length=15)
public String getRegNo() {
    return this.regNo;
}

public void setRegNo(String regNo) {
    this.regNo = regNo;
}

@Column(name="full_names", nullable=false)
public String getFullNames() {
    return this.fullNames;
}

public void setFullNames(String fullNames) {
    this.fullNames = fullNames;
@Column(name="current_year", nullable=false)
public int getCurrentYear() {
    return this.currentYear;
}

public void setCurrentYear(int currentYear) {
    this.currentYear = currentYear;
}

@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.LAZY,
            mappedBy="student")
public Set getStudentCourses() {
    return this.studentCourses;
}

public void setStudentCourses(Set studentCourses) {
    this.studentCourses = studentCourses;
}

@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.LAZY,
            mappedBy="student")
public Set getStudentLessons() {
    return this.studentLessons;
}

public void setStudentLessons(Set studentLessons) {
    this.studentLessons = studentLessons;
}

@Override
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }
    if (!(other instanceof Student)) {
        return false;
    }
    final Student student = (Student) other;
    if (!student.getRegNo().equals(this.getRegNo())) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 83 * hash + (this.regNo != null ? this.regNo.hashCode() : 0);
    return hash;
}
File: Student.hbm.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated May 15, 2011 3:10:22 PM by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
  <class name="attendanceregister.model.Student" table="student"
    catalog="attendance_register">
    <id name="id" type="java.lang.Integer">
      <column name="id" />
      <generator class="identity" />
    </id>
    <property name="regNo" type="string">
      <column name="reg_no" length="15" not-null="true" unique="true" />
      <comment>Student registration number</comment>
    </column>
  </property>
  <property name="fullNames" type="string">
    <column name="full_names" not-null="true" />
    <comment>Full names of the student</comment>
  </column>
  <property name="currentYear" type="int">
    <column name="current_year" not-null="true" />
  </property>
  <set name="courses" table="student_course">
    <key column="student_id" not-null="true" />
    <many-to-many column="course_id" class="attendanceregister.model.Course" />
  </set>
  <set name="studentCourses" inverse="true" cascade="all">
    <key>
      <column name="student_id" not-null="true" />
    </key>
    <one-to-many class="attendanceregister.model.StudentCourse" />
  </set>
  <set name="studentLessons" inverse="true">
    <key>
      <column name="student_id" not-null="true" />
    </key>
    <one-to-many class="attendanceregister.model.StudentLesson" />
  </set>
</class>
</hibernate-mapping>

File: StudentCourse.java

```java
package attendanceregister.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
```
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

/**
 * StudentCourse generated by hbm2java
 */
@Entity
@Table(name="student_course",
catalog="attendance_register",
    uniqueConstraints = @UniqueConstraint(columnNames={"student_id",
"course_id"}))
public class StudentCourse implements java.io.Serializable {

    private Integer id;
    private Course course;
    private Student student;

    public StudentCourse() {
    }

    public StudentCourse(Course course, Student student) {
        this.course = course;
        this.student = student;
    }

    @Id @GeneratedValue(strategy=IDENTITY)
    @Column(name="id", unique=true, nullable=false)
    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="course_id", nullable=false)
    public Course getCourse() {
        return this.course;
    }

    public void setCourse(Course course) {
        this.course = course;
    }

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="student_id", nullable=false)
    public Student getStudent() {
        return this.student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    @Override
    public boolean equals(Object other) {
        if (this == other) {
        }
```java
return true;
}
if (!(other instanceof StudentCourse))
{
    return false;
}
final StudentCourse sc = (StudentCourse) other;
if (!((this.getCourse().equals(sc.getCourse())) &&
(this.getStudent().equals(sc.getStudent()))))
{
    return false;
}
return true;
}

@Override
public int hashCode()
{
    int hash = 7;
    hash = 61 * hash + (this.course != null ? this.course.hashCode() : 0);
    hash = 61 * hash + (this.student != null ? this.student.hashCode() : 0);
    return hash;
}

File: StudentCourse.hbm.xml

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--[-- Generated May 15, 2011 3:10:22 PM by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
    <class name="attendanceregister.model.StudentCourse"
            table="student_course" catalog="attendance_register">
        <id name="id" type="java.lang.Integer"
            generator class="identity" />
        <many-to-one name="course" class="attendanceregister.model.Course"
            fetch="select">
            <column name="course_id" not-null="true" />
        </many-to-one>
        <many-to-one name="student" class="attendanceregister.model.Student"
            fetch="select">
            <column name="student_id" not-null="true" />
        </many-to-one>
    </class>
</hibernate-mapping>
```
The `attendanceregister.view` package holds the program code necessary to implement the presentation layer of the system. Sample files from the package are detailed below to illustrate this functionality.

File: `AttendanceDialog.java`

```java
package attendanceregister.view;

import attendanceregister.model.HibernateUtil;
import attendanceregister.model.Lesson;
import attendanceregister.model.Student;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javax.swing.table.DefaultTableModel;
import org.hibernate.Session;

public class AttendanceDialog extends javax.swing.JDialog {
    public Lesson getCurLesson() {
        return curLesson;
    }

    public void setCurLesson(Lesson curLesson) {
        this.curLesson = curLesson;
    }

    /** Creates new form AttendanceDialog */
    public AttendanceDialog(java.awt.Frame parent, boolean modal, Lesson curLesson) {
        super(parent, modal);
        this.curLesson = curLesson;
        initComponents();
    }

    private void initComponents() {
        // Initialize components...
    }
}
```
ScrollPane = new javax.swing.JScrollPane();
attendanceTable = new javax.swing.JTable();
label1 = new javax.swing.JLabel();
totalStudentsTextField = new javax.swing.JTextField();
label2 = new javax.swing.JLabel();
totalStudentsAttendedTextField = new javax.swing.JTextField();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Attendance Details");

attendanceTable.setModel(new javax.swing.table.DefaultTableModel(new Object [][] {
}, new String [] {
    "Name", "Number", "Attended"
}) {
    Class [] types = new Class [] {
        java.lang.String.class, java.lang.String.class,
        java.lang.Boolean.class
    };
    boolean [] canEdit = new boolean [] {
        false, true, false
    };

    public Class getColumnClass(int columnIndex) {
        return types [columnIndex];
    }

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
ScrollPane.setViewportView(attendanceTable);
updateStudentsAttended();

jLabel1.setText("Number of students registered");
totalStudentsTextField.setEditable(false);

jLabel2.setText("Number of students who attended");
totalStudentsAttendedTextField.setEditable(false);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 452, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addContainerGap());
group.addGroup(group.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(group.createSequentialGroup())
    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 452, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 452, javax.swing.GroupLayout.PREFERRED_SIZE))
});
addGroup(layout.createSequentialGroup()
    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
        .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    .addGap(33, 33, 33)
    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addContainerGap(118, Short.MAX_VALUE))
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 265, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 265, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 265, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
        .addGap(26, 26, 26)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel1)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabel2)
            .addComponent(totalStudentsAttendedTextField, javax.swing.GroupLayout.PREFERRED_SIZE, javax.sw
private javax.swing.JTextField totalStudentsAttendedTextField;
private javax.swing.JTextField totalStudentsTextField;
// End of variables declaration
private Lesson curLesson = null;

private void updateStudentsAttended()
{
    String studentsInThatCourse = "select s from Student s left join s.studentCourses sc right join sc.course c where c = ?";
    String studentsInThatLesson = "select s from Student s left join s.studentLessons sl join sl.lesson l where l = ?";
    int numStudentsAttended = 0;
    Session ss = HibernateUtil.getSessionFactory().openSession();
    ss.beginTransaction();
    Set<Student> setStudentsInThatCourse = new HashSet<Student>(ss.createQuery(studentsInThatCourse).
        setEntity(0, this.getCurLesson().getLecturerCourse().getCourse()).
        list());
    totalStudentsTextField.setText(String.valueOf(setStudentsInThatCourse.size()));
    ss.getTransaction().commit();
    ss.flush();
    ss.beginTransaction();
    Set<Student> setStudentsInThatLesson = new HashSet<Student>(ss.createQuery(studentsInThatLesson).
        setEntity(0, this.getCurLesson()).
        list());
    ss.getTransaction().commit();
    ss.flush();

    DefaultTableModel dtm = (DefaultTableModel) attendanceTable.getModel();
    for (Student sInCourse: setStudentsInThatCourse)
    {
        boolean studentAttended = setStudentsInThatLesson.contains(sInCourse);
        if (studentAttended) ++numStudentsAttended;
        dtm.addRow(new Object[] {
            sInCourse.getFullNames(), sInCourse.getRegNo(),
            studentAttended
        });
    }
    totalStudentsAttendedTextField.setText(String.valueOf(numStudentsAttended));
}
}
Additionally, the server module also contains a module which handles writing of student data to the smart card. The code is contained in the `com.SmartCardFEE.writer` package.

File: Writer.java

```java
package com.SmartCardFEE.writer;

import com.SmartCardFEE.writer.model.Student;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.SmartCardFEE.Constants;
import com.SmartCardFEE.writer.model.StudentCourse;
import javax.smartcardio.Card;
import javax.smartcardio.CardChannel;
import javax.smartcardio.CardException;
import javax.smartcardio.CardTerminal;
import javax.smartcardio.CardTerminals;
import javax.smartcardio.CommandAPDU;
import javax.smartcardio.ResponseAPDU;
import javax.smartcardio.TerminalFactory;

public class Writer {
    private void writeToCard(String command) {
        String line = "";
        OutputStream stdin = null;
        InputStream stderr = null;
        InputStream stdout = null;
        try {
            Process process = Runtime.getRuntime().exec(new String[] {
                "C:\Program Files\OpenSC Project\OpenSC\opensc-explorer.exe", "-c", "gpk",
            });
            stderr = process.getErrorStream();
            stdout = process.getInputStream();
            stdin = process.getOutputStream();
            stdin.write(command.getBytes());
            stdin.flush();
            BufferedReader brCleanUp = new BufferedReader(new InputStreamReader(stdout));
            while ((line = brCleanUp.readLine()) != null) {
                System.out.println("[Stdout] " + line);
            }
            brCleanUp.close();
        } finally {
            if (stdin != null) stdin.close();
            if (stderr != null) stderr.close();
            if (stdout != null) stdout.close();
        }
    }
}
```
System.out.println("[Process exit value] " +
process.exitValue());

}  catch (IOException ex)
{
    Logger.getLogger(Writer.class.getName()).log(Level.SEVERE, null, ex);
}

public void formatCard()
{
    command += "mkdir " + Constants.FEE_FOLDER + " 00
    command += "quit\n";
    
    writeToCard(command);
    CardTerminals cts = TerminalFactory.getDefault().terminals();
    try
    {
        CardTerminal ct =
        cts.list(CardTerminals.State.CARD_PRESENT).get(0);
        Card c = ct.connect("*");
        CardChannel cc = c.getBasicChannel();
        openDF(cc, (Constants.FEE_FOLDER & 0xFF00) >> 8,
        (Constants.FEE_FOLDER & 0x00FF));
        createRegNoFile(cc);
        if (openEF(cc, (Constants.REG_NO_FILE & 0xFF00) >> 8,
        (Constants.REG_NO_FILE & 0x00FF)))
        {
            updateBinary(cc,
            makeEmptyString(Constants.REG_NO_LENGTH).getBytes());
            createFullNamesFile(cc);
            if (openEF(cc, (Constants.NAME_FILE & 0xFF00) >> 8,
            (Constants.NAME_FILE & 0x00FF)))
            {
                updateBinary(cc,
                makeEmptyString(Constants.NAME_FILE_LENGTH).getBytes());
            }
            createCoursesFile(cc);
            if (openEF(cc, (Constants.COURSE_FILE & 0xFF00) >> 8,
            (Constants.COURSE_FILE & 0x00FF)))
            {
                for (int i = 0; i < Constants.NUM_COURSES; )
                {
                    updateRecord(cc, ++i,
                    makeEmptyString(Constants.COURSE_RECORD_LENGTH).getBytes());
                }
            }
        }
        c.disconnect(true);
    }  catch (CardException ex)
public void writeStudentDetails(Student s) {
    CardTerminals cts = TerminalFactory.getDefault().terminals();
    try {
        CardTerminal ct = cts.list(CardTerminals.State.CARD_PRESENT).get(0);
        Card c = ct.connect("*");
        CardChannel cc = c.getBasicChannel();
        if (openEF(cc, (Constants.REG_NO_FILE & 0xFF00) >> 8,
            (Constants.REG_NO_FILE & 0x00FF)))
            updateBinary(cc, s.getRegNo().getBytes());
        if (openEF(cc, (Constants.NAME_FILE & 0xFF00) >> 8,
            (Constants.NAME_FILE & 0x00FF)))
            updateBinary(cc, s.getFullNames().getBytes());

        createCoursesFile(cc);
        if (openEF(cc, (Constants.COURSE_FILE & 0xFF00) >> 8,
            (Constants.COURSE_FILE & 0x00FF)))
            java.util.Set<Object> setSC = s.getStudentCourses();
            java.util.Iterator i = setSC.iterator();
            int record = 0;
            while (i.hasNext())
                { StudentCourse sc = (StudentCourse)i.next();
                    updateRecord(cc, ++record,
                        sc.getCourse().getName().getBytes());
                }
        c.disconnect(true);
    } catch (CardException ex) {
        Logger.getLogger(Writer.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void createRegNoFile(CardChannel cc) {
    System.out.println((Constants.REG_NO_FILE & 0xFF00) >> 8);
    transmit(cc, toByteArray(new int[] {
        0x80, 0xE0, 0x02, 0x00, 0x0C,
        /*Data*/
private void createFullNamesFile(CardChannel cc) {
    transmit(cc, toByteArray(new int[] {
        0x80, 0xE0, 0x02, 0x00, 0x0C,  // Data
        (Constants.NAME_FILE & 0xFF00) >> 8,  // File Identifier */
        0x01, /* File Descriptor Byte - Transparent EF*/
        0x00, /* Record length - Binary EF */
        0x00, (Constants.NAME_FILE_LENGTH & 0x00FF), /* File Size */
    }));
}

private void createCoursesFile(CardChannel cc) {
    transmit(cc, toByteArray(new int[] {
        0x80, 0xE0, 0x02, 0x00, 0x0C,  // Data
        (Constants.NAME_FILE & 0xFF00) >> 8,  // File Identifier */
        0x02, /* File Descriptor Byte - Linear Fixed EF*/
        Constants.COURSE_RECORD_LENGTH, /* Record length - Linear Fixed EF */
        0x00, (Constants.COURSE_FILE_LENGTH & 0x00FF), /* File Size */
    }));
}

private void updateBinary(CardChannel cc, byte[] byte_data) {
    byte[] data = new byte[6 + byte_data.length];
    data[0] = 0x00;
    data[1] = (byte) 0xD6;
    data[2] = 0x00;
    data[3] = 0x00;
    data[4] = (byte) byte_data.length;
    data[data.length - 1] = 0x00;
    int counter = 5;
    for (byte b : byte_data)
    {
        data[counter++] = b;
    }
}
transmit(cc, data);
}

// Update currently selected linear EF
private void updateRecord(CardChannel cc, int record, byte[] byte_data)
{
    byte[] data = new byte[6 + byte_data.length];
    data[0] = 0x00;
    data[1] = (byte) 0xDC;
    data[2] = (byte) record;
    data[3] = 0x00;
    data[4] = (byte) byte_data.length;
    data[data.length - 1] = 0x00;
    int counter = 5;
    for (byte b : byte_data)
    {
        data[counter++] = b;
    }
    transmit(cc, data);
}

private String makeEmptyString(int length)
{
    return String.format(String.format("%%0%dd", length), 0).replace("0", " ");
}

private byte[] transmit(CardChannel cc, byte[] apdu_bytes)
{
    CommandAPDU cAPDU = new CommandAPDU(apdu_bytes);
    try
    {
        ResponseAPDU rAPDU = cc.transmit(cAPDU);
        printBytes(rAPDU.getBytes());
        return rAPDU.getBytes();
    }
    catch (CardException ce)
    {
        ce.printStackTrace();
    }
    return null;
}

private byte[] toByteArray(int[] intArray)
{
    byte[] bArray = new byte[intArray.length];
    for (int i = 0; i < intArray.length; i++)
    {
        bArray[i] = (byte)intArray[i];
    }
    return bArray;
}

private boolean openEF(CardChannel cc, int msb, int lsb)
{
    byte[] bArray = transmit(cc, toByteArray(new int[]
    {
        0x00, 0xA4, 0x02, 0x00, 0x02, msb, lsb
    }));
The client module also includes code needed to access the smart card and retrieve information which is stored on it. The com.SmartCardFEE.reader package contains the code necessary to access the smart card and retrieve information.

File: Reader.java

```java
package com.SmartCardFEE.reader;

import javax.smartcardio.Card;
import javax.smartcardio.CardChannel;
import javax.smartcardio.CardException;
import javax.smartcardio.CardTerminal;
import javax.smartcardio.CardTerminals;
import javax.smartcardio.CommandAPDU;
import javax.smartcardio.ResponseAPDU;
import javax.smartcardio.TerminalFactory;
import com.SmartCardFEE.Constants;

public class Reader {
    private java.util.Set<ReaderListener> readerListeners = new java.util.HashSet<ReaderListener>(0);
    private static Reader reader = new Reader();
    private static Thread cardThread;

    private Reader()
    {
        Runnable cardRunnable = new Runnable()
        {
            @Override
            public void run()
            {
                // Card access logic
            }
```
```
CardTerminals cts = TerminalFactory.getDefault().terminals();
while (true)
{
    try
    {
        for (CardTerminal terminal : cts.list(CardTerminals.State.CARD_INSERTION))
        {
            Card c = terminal.connect("*");
            CardChannel cc = c.getBasicChannel();
            //cd to EF 0x0505
            if (!openDF(cc, (Constants.FEE_FOLDER & 0xFF00) >> 8, (Constants.FEE_FOLDER & 0x00FF)))
            {
                c.disconnect(true);
                break;
            }
            //Read regNo File
            StudentInfo si = new StudentInfo();
            if (openEF(cc, (Constants.REG_NO_FILE & 0xFF00) >> 8, (Constants.REG_NO_FILE & 0x00FF)))
            {
                String str = new String(readSelectedBinaryEF(cc, Constants.REG_NO_LENGTH));
                si.setRegNo(str.substring(0, Constants.REG_NO_LENGTH));
            }
            //Read fullNames file
            if (openEF(cc, (Constants.NAME_FILE & 0xFF00) >> 8, (Constants.NAME_FILE & 0x00FF)))
            {
                String str = new String(readSelectedBinaryEF(cc, Constants.NAME_FILE_LENGTH));
                si.setFullNames(str.substring(0, Constants.NAME_FILE_LENGTH));
            }
            //Read courses records
            if (openEF(cc, (Constants.COURSE_FILE & 0xFF00) >> 8, (Constants.COURSE_FILE & 0x00FF)))
            {
                for (int i = 0; i < Constants.NUM_COURSES; )
                {
                    String course = new String(readSelectedLinearEF(cc, ++i, Constants.COURSE_RECORD_LENGTH));
                    if (course.trim().length() > 0)
                    {
                        si.getCourses().add(course);
                    }
                }
            }
        }
    }
    //Perform check on at least the length of the reg no to ensure unempty fields
    if (si.getRegNo().trim().length() > 0)
    {
        callReaderListeners(si);
    }
    c.disconnect(true);
public void start()
{
    cardThread.start();
}

public void stop()
{
    cardThread.stop(); // Is deprecated
}

private void callReaderListeners(StudentInfo si)
{
    java.util.Iterator<ReaderListener> i =
        readerListeners.iterator();
    while (i.hasNext())
    {
        i.next().cardInserted(si);
    }
}

public void addReaderListener(ReaderListener rl)
{
    readerListeners.add(rl);
}

public void removeReaderListener(ReaderListener rl)
{
    readerListeners.remove(rl);
}

private static void printBytes(byte[] apdu_bytes)
{
    for (byte b : apdu_bytes)
    {
        System.out.print(String.format("%02X ", b));
    }
    System.out.println();
}

private static byte[] transmit(CardChannel cc, byte[] apdu_bytes)
{
    CommandAPDU cAPDU = new CommandAPDU(apdu_bytes);
try {
    ResponseAPDU rAPDU = cc.transmit(cAPDU);
    return rAPDU.getBytes();
} catch (CardException ce) {
    ce.printStackTrace();
} return null;

private static byte[] toByteArray(int[] intArray) {
    byte[] bArray = new byte[intArray.length];
    for (int i = 0; i < intArray.length; i++) {
        bArray[i] = (byte)intArray[i];
    } return bArray;
}

private boolean openEF(CardChannel cc, int msb, int lsb) {
    byte[] bArray = transmit(cc, toByteArray(new int[]{
        0x00, 0xA4, 0x02, 0x00, 0x02, msb, lsb
    }));
    return ((bArray[bArray.length - 2] == 0x90) && (bArray[bArray.length - 1] == 0x00));
}

private boolean openDF(CardChannel cc, int msb, int lsb) {
    byte[] bArray = transmit(cc, toByteArray(new int[]{
        0x00, 0xA4, 0x01, 0x00, 0x02, msb, lsb
    }));
    return ((bArray[bArray.length - 2] == 0x90) && (bArray[bArray.length - 1] == 0x00));
}

//Read currently selected EF - Binary EF
private byte[] readSelectedBinaryEF(CardChannel cc, int length) {
    return transmit(cc, toByteArray(new int[]{
        0x00, 0xB0, 0x00, length
    }));
}

//Read record from currently selected EF - Linear EF
private byte[] readSelectedLinearEF(CardChannel cc, int record, int length) {
    return transmit(cc, toByteArray(new int[]{
        0x00, 0xB0, 0x00, length
    }));
}
0x00, 0xB2, record, 0x00, length
})
})
}