Master Thesis Number: 92

A Tool for Course Scheduling

A thesis proposal submitted in
partial fulfillment of the requirements for the degree of
Master of Science in
Computer Science and Engineering

By

Hua Wang

Informatics and Mathematical Modelling Department
Technical University of Denmark

December 31, 2004

Advisor: Paul Fischer

Abstract

Each year at the beginning of a new academic semester, most advisors face a very common and particularly tedious and time-consuming problem: deciding for each teacher what course schedule would be ideal for the following semester. The factors that have to be considered vary from specific requirements such as course pre-requisites, spring and fall offerings and so on.

In this master thesis, a software tool has been developed to help solving the presented problem. This tool can display the courses by different scheduling. The course placement can be changed and updated automatically. The user can also view or edit any specific course information by this tool. In addition, a data generator was developed for reading the course data from a web site and converting it to a text format file, which was finally converted to an xml format file as data source in our system. We have successfully tested the tool.

The code was written in Java and xml data format was used to store the course data. A user-friendly graphical user interface was also designed and implemented by Java Swing. The code has been kept open to future modifications and the whole structure of the program was designed to allow an easy extension at any given point.

Finally conclusion was got and future work was given to the reader.

Acknowledgements

This work has been done for Informatics and Mathematical Modelling department in the Technical University of Denmark.

I am very grateful to my supervisor Paul Fischer, for his support and encouragement in my thesis work.

I would also like to thank my examiner Gert Schmeltz Pedersenfor.

Furthermore, I wish to thank my family for their enduring support and patience.

Finally, I would like to dedicate this thesis to all the friends I made during my years of studying, who made this time the great and unforgettable experience it has been.

Table of Contents

Chapter 1 Introduction 1
1.1 Background 1
1.2 Objectives 1
1.3 Aim and Limitations 2
1.4 Structure of Thesis 2
Chapter 2 Theory 3
2.1 Java 3
2.2 Java Swing 3
2.3 XML 4
2.4 XML Parser 6
2.5 Conflict Theory 7
Chapter 3 Analysis and Design 9
3.1 Implementation Language 9
3.2 User Requirements 10
3.3 System Architecture 10
3.4 Database Design 12
3.5 User Interface Design 14
3.5.1 Design Principle of User Interface 14
3.5.2 Prototype of User Interface 15
3.6 Java Program Design 19
3.6.1 Design Principle 19
3.6.2 Structure of Java Program 19
Chapter 4 Implementation 23
4.1 Class URLReader 23
4.2 Class txtToXml 24
4.3 Class CourseTool 24
4.4 Class CourseSchedulingGUI 26
4.5 Class CourseListGUI 28
4.6 Class CourseInfoGUI 29
4.7 Class CourseDataHandler 30
4.8 Class SAXModelBuilder 32
4.9 Class SimpleElement 33
4.10 Class Course 33
4.11 Class CourseList 34
Chapter 5 Tests and Results 35

Chapter 6 Conclusions and Future Prospects 36
6.1 Conclusions 36
6.2 Future Prospects 37
Appendix 1 References 38
Appendix 2 User Manual 39
Appendix 3 Source Code 46
1 Class URLReader 46
2 Class TxtToXml 51
3 Class CourseTool 52
4 Class CourseScheduleGUI 60
5 Class CourseListGUI 69
6 Class CourseInfoGUI 74
7 Class CourseDataHandler 79
8 Class SAXModelBuilder 83
9 Class SimpleElement 85
10 Class Course 86
11 Class CourseList 89

Chapter 1 Introduction

1.1 Background

The courses at IMM (Informatics and Mathematical Modelling), a department in the Technical University of Denmark, have to be scheduled. This is done by one or more teachers, and it is a quite time consuming business. This normally happens once a year, but minor changes are had to make in the mean time. There are a number of problems to consider when scheduling courses:

- Man power (No teacher can give two courses at the same time).
- Conflicts between courses (Some courses may not run at the same time, because the students have to follow both).
- Some ten-point courses have to run on a single day
- The correct placement in the course of the study
- Personal preference of the teachers.
- Availability of data bars and other resources
- Some more considerations that come up spontaneously.

As the basic conditions frequently change (Due to changes in staff of request form the outside), it is not possible to automate the placement. However a tool to support a manual placement is urgently needed. This tool should allow multiple views on the data, support rearrangements and inform the user about unwanted situations.

1.2 Objectives

This project is to design and implement such a tool for course scheduling. The tool should visualize courses and their relations. It should allow the user to change the course data and placement interactively. It should keep record of courses given in previous years. The software should consist of three parts: database, user interface and Java program. The interactions among them should be as below:

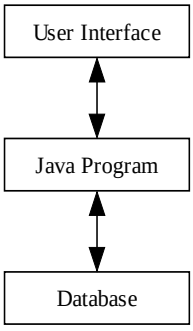


Figure 1.2.1

1.3 Aim and Limitations

Java programming language is used in this project. It contains an open source library, which might be useful for a further extension of this work. Due to the time limit, the analysis had to be restricted in some points. At this point, only part of courses is selected from IMM. It has the advantage of not only being easy for test but also increasing the running time of the program. The conflict courses are not calculated by program, but given by my supervisor in advance. Therefore it reduces the complexity of this program. Furthermore, the code has been written so that it is possible to add additional features with relative ease at a future point.

1.4 Structure of Thesis

Chapter 2 presents an overview of the conflict theory used in the project as well as a short introduction to Java and XML.

Chapter 3 gives user requirements of the system, followed by the system architecture. The design principle and the general structure of the program are also explained in this chapter.

Chapter 4 shows the detail implementation, giving a summary of all classes and methods and their respective functions.

Chapter 5 is working with the test of system.

Chapter 6 gives a summary of the thesis and future work.

Thesis closes with various appendices.

Chapter 2 Theory

2.1 Java

In the course of this report it is not possible to give a detailed insight to the programming language Java. However, this chapter tries to give some basic understanding about its principles. As an object orientated programming language, most programs written in Java consists of various classes, which each represents a different type of object (e.g. a course). Every object (an instantiation of a class) can have various attributes, which are called variables. Variables are either different for each instance of the class or can be referred as static variables, which are valid for the whole class. The same system is applied to the methods, which are also implemented in each class either as normal or static ones. Methods can have input and output variables. The advantage of this structure lies in the interchangeability of the classes. It is relatively easy to replace classes or to extend the whole program by the implementation of additional classes. To make this possible it is vitally important to think about a logical structure in the program architecture.

An advantage of Java is its cross-platform compatibility and also its open source, which makes it easy to develop (not to mention that it is available free of charge on the internet). For more details about Java programming with this language, you can read one of the many books published on this subject.

2.2 Java Swing

Java Swing has been used widely in the field of graphical interface design. The Swing classes eliminate Java's biggest weakness: its relatively primitive user interface toolkit. Swing provides many new components and containers that allow you to build sophisticated user interfaces, far beyond what was possible with AWT. The old components have been greatly improved, and there are many new components, like trees, tables, and even text editors. It also adds several completely new features to Java's user interface capabilities: drag-and-drop, undo, and the ability to develop your own "look and feel" or the ability to choose between several standard looks. The Swing components are all "lightweight," and therefore provide more uniform behavior across platforms, making it easier to test your software.

Although Swing is separate from the AWT, it is implemented in terms of basic AWT classes. The AWT provides the interface between the underlying native windowing system and the Java GUI components. Swing uses this interface, but does not rely on

AWT components that make use of native windowing system objects. Instead, Swing components are written in pure Java. This provides significant advantages. It allows Swing components to be independent of the native windowing system, which means they can run on any windowing system that supports the AWT. It also allows Swing components to be independent of any limitations of the native windowing systems. This independence allows Swing to control and tailor its look and feel

All these new features mean that there's a lot to learn. Swing is an undoubtedly way ahead of AWT -- or, for that matter, any widely available user interface toolkit -- but it's also a lot more complicated. It's still easy to do simple things. But once you've seen what's possible, you won't want to do the simple things.

2.3 XML

XML is an Extensible Markup Language. It allows the flexible development of user-defined document types. It provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the Web. XML not only makes us easier to program, but also makes us easier to change the data structure later.

XML is a specification for storing and exchanging data that the World Wide Web Consortium (W3C) created in 1996 to standardize information delivery across the Internet. The W3C defines XML as a subset of SGML. XML is not a new language; instead, you can think of it as a language specification. XML is similar to HTML. XML uses tags and attributes to define data in the same way that HTML uses tags to define formatting. However, instead of having a fixed set of tags, as HTML has, XML lets the creator define the tags its XML streams use. Therefore, you can use tags to make content self-evident.

Many software developers across the world are integrating XML into their applications to benefit from the various advantages of this language, which are:

Simplicity: XML is simple because the author and the provider can design their own document type using XML (vs. HTML)

(Example of XML presentation)
< Courses>
< Course>
<Name>Software Engineering< Name>
<Id>02260</ Id>
</ Course >
</ Courses >

Intelligence: XML is smart data. While the hypertext markup language (HTML) shows how the data should look, XML tells the user what the data means.

Simplified subset of SGML: XML removes many of the underlying complexities of SGML in favor of a more flexible model; so writing programs to handle XML will be much easier than doing the same for full SGML.

Accessible and reusable information: XML eliminates the chaos associated with HTML while allowing better management and reuse of structured documents.

Improvement of performance through granular updates: XML enables granular updating. Developers do not have to send the entire structured data set each time there is a change. With granular updating, only the changed element must be sent from the server to the client. The changed data can be presented without the need to refresh the entire page or table.

Independence: The tags in XML documents are not predefined. Any descriptive string can be used (as long as it uses the approved character set). Moreover, the data defined by these tags can be displayed in any way you like and on any platform.

Adaptation: XML' s adaptation is infinite.

Maintenance: When the DTD is provided, XML data become easy to maintain.

Due to so many advantages of XML, XML data format is preferred in our design.

2.4 XML Parser

XML is a technology for marking up structured data so that any software with an XML parser can understand and use its content. Data independence, the separation of content from its presentation, is the essential characteristic of XML. XML documents are simply text files that are marked up in a special way, so XML is intelligible to both humans and machines. Any application can conceivably process XML data. That is why XML is ideal for data exchange.

Using text to exchange data is a common problem in programming. A parser is an application that reads a document and understands its formatting conventions, usually enforcing some rules about the content. With XML for the common types of information that we exchange, we should no longer have to write parsers that deal with basic syntax and string manipulation. In conjunction with document-verifying components (DTDs or XML Schema), much of the complex error checking is also done automatically

SAX is a low-level, event-style mechanism for parsing XML documents. SAX originated in Java, but has been implemented in many languages. The primary motivation for using SAX is that it is lightweight and event-driven. SAX doesn't require maintaining the entire document in memory. If, for example, you need to grab the text of just a few elements from a document, or if you need to extract elements from a large stream of XML, you can do so efficiently with SAX. The event-driven nature of SAX also allows you to take actions as the beginning and end tags are parsed. This can be useful for directly manipulating your own models without first going through another representation. A SAX Parser reports a XML document to an application as a series of events as below:

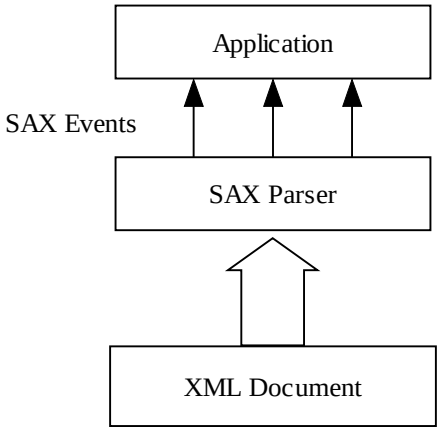


Figure 2.4.1 SAX parser

So SAX-based parser for XML is preferred in our design.

2.5 Conflict Theory

When the courses have to be scheduled in such a way that no two courses with common students have overlapping meeting times, and also the same teacher can not teach two courses at the same time. Usually the administration does not have any guarantee that their scheduling is correct. Hence they don't know if all the courses appear at the right placement. This leads to conflicts. Somehow the administration samples the information consisting of this kind of courses. This information is visualized as the conflict graph, where the courses are the vertices, and two such vertices are joined by an edge iff there is time conflict, i.e. if the courses have common placement.

In our case, each course is represented as a vertex. Whenever there is a conflict between two courses, an edge is drawn to connect these two vertices. That means the two courses connected by the same edge cannot run at the same time.

For example:
Course a is conflict with courses b and e.
Course b is conflict with course d.
Course c is conflict with course e.
Course d is conflict with course a.

The conflict graph based on the above course description can be drawn as below:

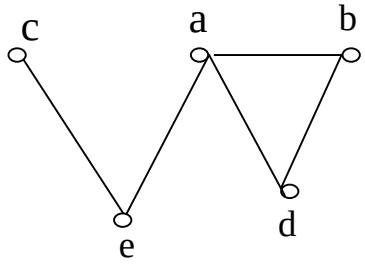


Figure 2.5.1 Conflict graph example

In this figure, course a, b and d can not run at the same placement, but course e and d may run at the same placement, because there is no edge between course e and d, so they are not conflict courses according to our definition.

Once the conflict graph is made, all edges have to be checked in the conflict graph when scheduling the courses. Course a and b are conflict courses according to Figure 2.5.1. If these two courses are placed at the same time period, for example both on Monday Morning, then a warning should be shown. For example color both course a and b as red or use other symbols.

To make a good understanding, we assume an course schedule example is as below:

	Monday	Tuesday	Wednesday	Thursday	Friday
Morning	a	c		e	
Afternoon			d		b

Table 2.5.1 Original course schedule

When we reset the time of course b from Friday Afternoon to Monday Morning, then course b will be conflict with course a according to conflict graph in Figure 2.5.1. The system should show this conflict information to the user as below, where (*) means they are conflict courses.

	Monday	Tuesday	Wednesday	Thursday	Friday
Morning	a(*), b(*)	c		e	
Afternoon			d		

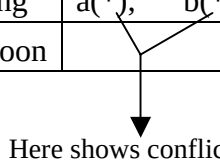


Table 2.5.2 Result course schedule

The same thing will also happen when course b runs on Wednesday Afternoon, as course b and course d are also conflict courses according to conflict graph in Figure 2.5.1.

The conflict theory is used in our software for deciding whether a course will be conflict with other courses, when the user changes a course placement.

.

Chapter 3 Analysis and Design

3.1 Implementation Language

Prior to start developing the application, we must consider which programming language to use. We have the following requirements:

- Object oriented language – the extensibility of the program increases when an object-oriented language is used and the class hierarchy is well designed.
- Ability to parse XML files.
- Platform independence – not necessary, but at least the programming language compiler/interpreter should be available on multiple platforms (Windows/Linux).
- Available developing environments.
- Capabilities and tools for creation of GUI.
- Other capabilities (standard libraries).

C++ and Java programming languages can be selected, each of which has its own advantages and disadvantages. The comparisons are summarized in Table 3.1.1 as below:

Programming Language	XML-parser	Developing Environment	Platforms
Java	Yes	Eclipse, NetBeans, JBuilder	Multiple, independent
C++	Yes	VC++ (Windows), Emacs(Linux)	Multiple

Table 3.1.1 Comparison of C++ and Java

According to the above table we can see that both C++ and Java have XML-parses. The problem for C++ is that it requires additional work to port the code to another platform, strictly more than by Java. This concerns especially GUI. For example, GUI created for Windows platform cannot be ported to Linux easily, and vice versa. Libraries available under Windows are not automatically available under Linux. Although we do not require platform independence, we still require that the portability should be as easy as possible. In Java the situation is different. Libraries, which do not contain additional dynamic libraries, are usable on all platforms. So Java programming language is selected in our design.

3.2 User Requirements

According to the project description, the tool should fulfill the following user requirements.

1) It should have a database, which contains the information on courses given at IMM.

2) The course information should have course basic information, such as course number, course name, teacher etc. Furthermore, it should also consist of relational information like pre-requisites and conflicting courses.

3) It should keep a record of courses given in previous years.

4) It should allow the user to change the course data and placement interactively

5) It should have the functionality to find out conflict courses and display them to the user.

6) It should display courses by different scheduling, such as by year, placement, semester, week.

7) It should allow the user to add, view, edit, delete and search courses in the database.

8) It should automatically update the course data in the database after the user has saved changes to the database.

3.3 System Architecture

Prior to start coding the program, the developers often consider the main system architecture, which is more or less independent from the applied programming language. Since our user requirements have been defined, the system architecture is determined simply by formalization of our basic ideas, according to which the following diagram can be drawn.

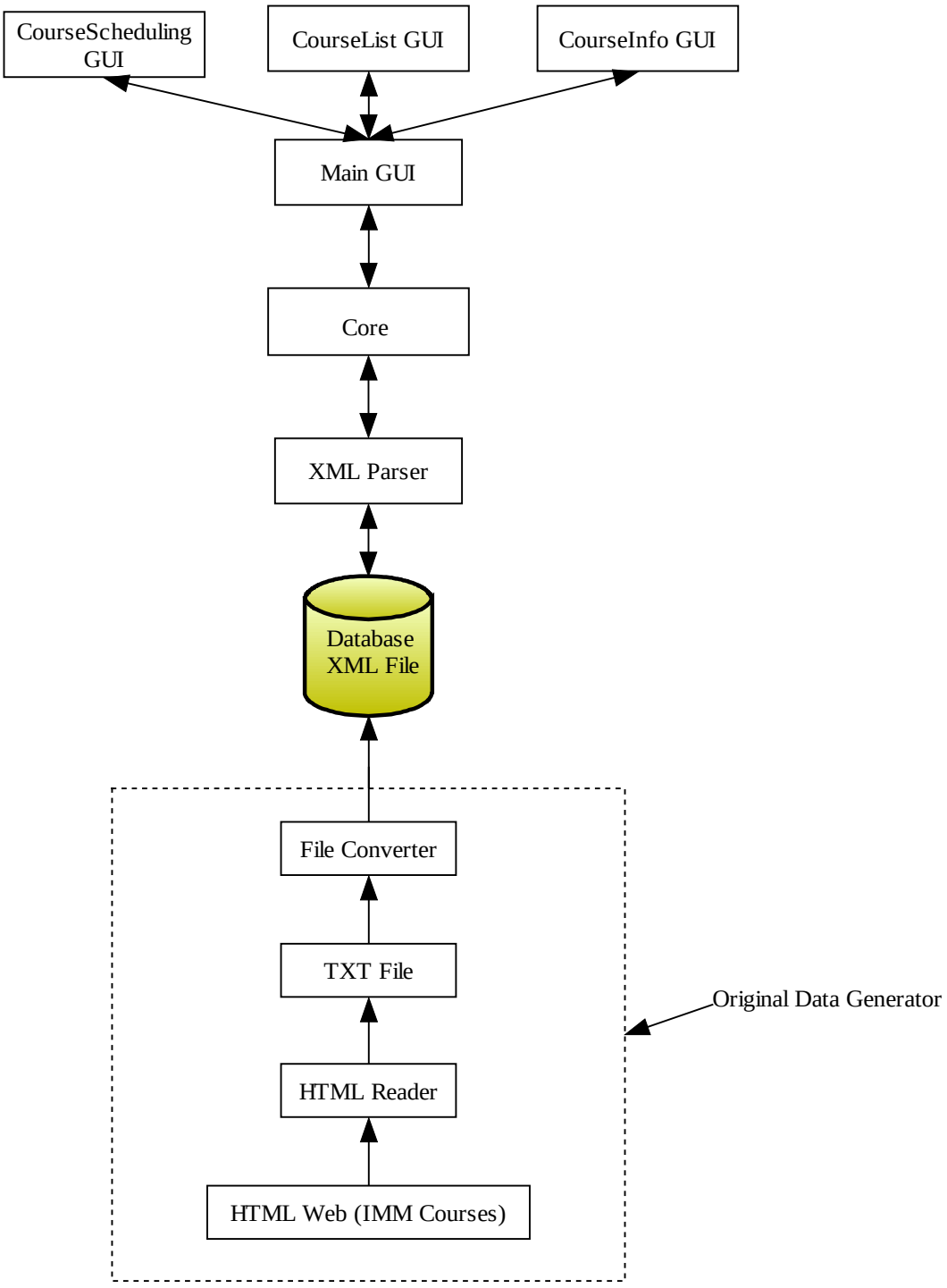


Figure 3.3.1 System architecture

The main application is represented by the Main GUI component. While the Main GUI component contains the main user interface, and it behaves from user's side transparently. Core component has significant importance for the functionality of the whole application,

because it is responsible for handling messages between the database and the user interface. We require the user interface to be designed using graphic means (GUI). The Main GUI component serves as a container for CourseScheduling GUI, CourseList GUI and CourseInfo GUI. Thus, we can get the user interface of CourseScheduling, CourseList and CourseInfo from Main GUI component. The data for the software is XML-based format. Its structure will be introduced later. XML Parser component is used to parse data from an xml document to Core component. The components inside the dashed rectangular are an extra tool to generate the original data for database. Original Data Generator reads an html file from a website to a text format file, and then converts the text format file to an xml file, which will be used in our system.

3.4 Database Design

As the original course information is coming from IMM. It is a time consuming work, if we input all the course information to the database by hand. So a java class called URLReader is designed to automatically get the course information from IMMs web page and then save them in into a file named courseinfo.txt.

To support flexibility and reusability, XML data format will be used to save the course data. The advantages of XML have been given in section 2.2. And an extra java class called TxtToXml is used to convert the text format file to a XML format file named courseinfo.xml, which is used for saving our course data.

The above process is displayed as below:

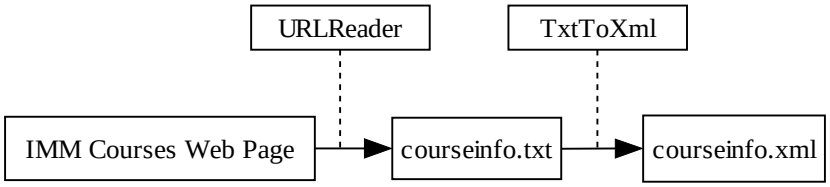


Figure 3.4.1 Original data generator

The data generator in Figure 3.4.1 is only used once for the first time we generate the original course data for our system.

The data structures for class TxtToXML and URLReader are as below:

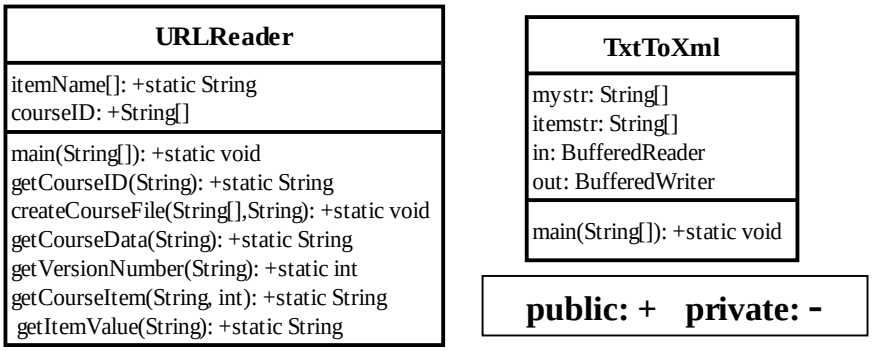


Figure 3.4.2 URLReader and TxtToXml

Class Course is needed to represent the course information. Its attributes are listed in the following table:

Attribute	Description
CourseNumber	Identity number of course
EnglishName	English name of course
DanishName	Danish name of course
ShortName	Short name of course
Placement	Time schedule of course
Frequency	Frequency of course
ResponsiblePerson	Responsible person of course
Teacher	Teacher of course
Ets	Credit of course
Censoring	Censor of exam
ExamType	Type of censoring
Curricula	Curricula level of course
Prerequisites,	Request for course
ConflictCourses	Conflict courses
Remark	Remark for course
NumberOfStudents	Student number for course

Table 3.4.1 Attributes of class Course

The data structure for course in XML document is as below:

```
<Course >
        <CourseNumber>02115</CourseNumber>
        <EnglishName>Java Programming</EnglishName>
        <DanishName>Java-programmering</DanishName>
        <ShortName>JP</ShortName>
        <Placement>Evening</Placement>
        <Frequency>Every year</Frequency>
        <Rperson>Jens Thyge Kristensen</Rperson>
        <Teacher>Jens Thyge Kristensen</Teacher>
        <Ets>10</Ets>
        <Censoring>Scale of marks (0-13), internal examiner</Censoring>
        <ExamType>Evaluation of experiments and reports. </ExamType>
        <Curricula>M.Sc.</Curricula>
        <Prerequisites>Basic programming experience</Prerequisites>
        <ConflictCourses>02100    / 02199</ConflictCourses>
        <Remark>The course is primarily for International Masters students</Remark>
        <NumberOfStudents>140</NumberOfStudents>
    </Course>
```

3.5 User Interface Design

3.5.1 Design Principle of User Interface

In the design we should use both knowledge in the world and knowledge in the head to simplify the structure of tasks. The GUI must conform to the Java Look and Feel Guidelines and to the generally expected behavior of a modern graphical user interface. Consistency should be kept in the design.

Menus must always appear in the same order and a standard “Help” menu must be available. The same user interface elements should be used for the same or similar tasks throughout the application. This should also lead to better code reuse.

The GUI must provide a good conceptual model of the system. A good conceptual model enables the users to predict what results their actions will cause in the system. This makes users more confident to use the system, because they know what will happen when they initiate an action. A good conceptual model helps users to choose which action they should take and what they want to do. All-important functions should be accessible easily. Users should be able to switch freely between functions.

3.5.2 Prototype of User Interface

Java Swing is used to design user interface. In this part, some prototypes of the user interface will be shown.

1) CourseSchedulingGUI Interface

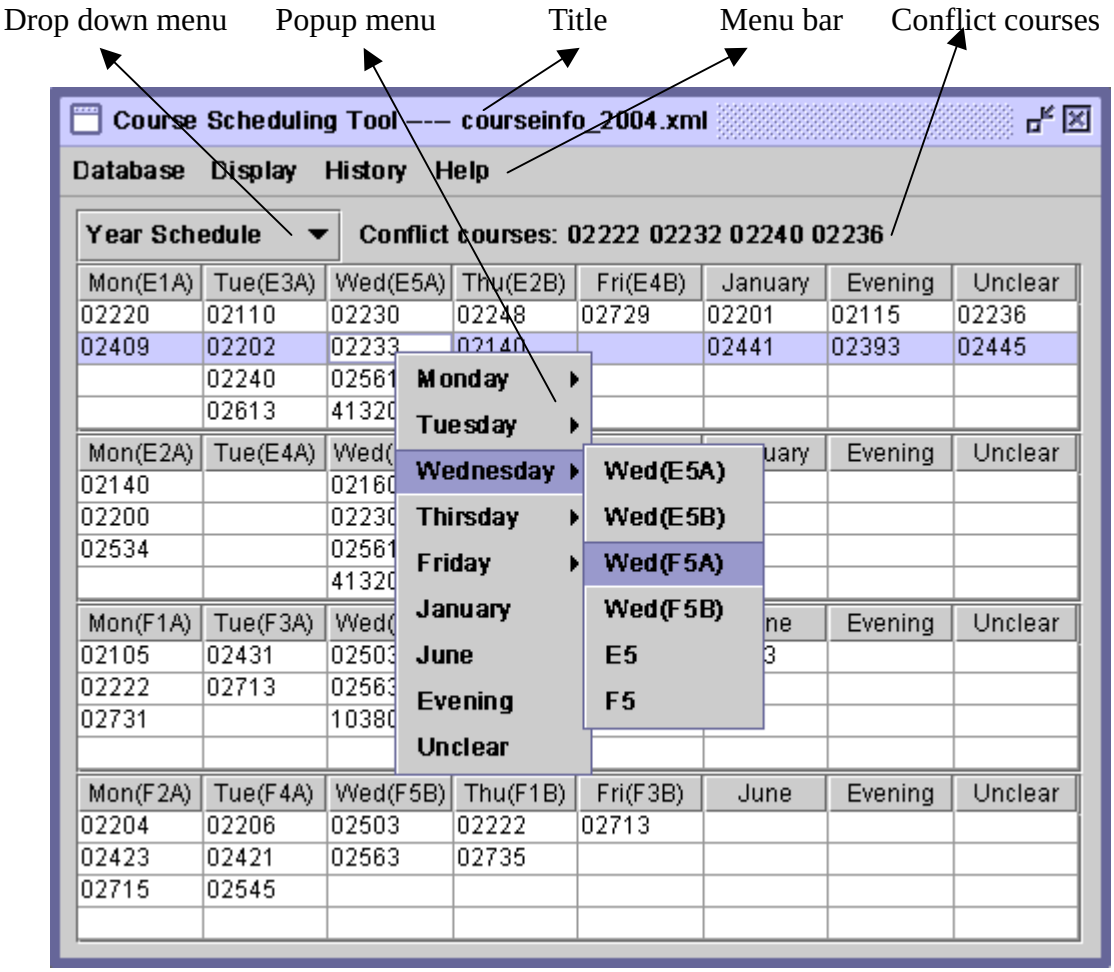


Figure 3.5.2.1 CourseSchedulingGUI

When the program is started, the interface in Figure 3.5.2.1 is presented to the user. The current database name courseinfo_2004.xml is shown on the title of the interface. A menu bar lies in the upper side of the interface. From the menu bar, the user can perform different operations on courses and database. In the following, detail information about menu bar will be introduced.

The menu bar contains the following options:

Menu Item	Sub Item	Description
Database		
	Open Course Database	Open a database file.
	Save Course Database	Save course data to current database.
	Save Course Database As	Save course data to another database file.
	Close Course Database	Close the current database.
	Exit	Terminate the program.
Display		
	Course Display	Display CourseListGUI interface.
	Schedule Display	Display CourseSchedulingGUI interface.
History		
	Year 2004	Change to course data in Year 2004.
	Year 2003	Change to course data in Year 2003.
	Year 2002	Change to course data in Year 2002.
	Year 2001	Change to course data in Year 2001.
	Year 2000	Change to course data in Year 2000.
Help		Helpful information to the user.

Table 3.5.2.1 Menu bar design

The course placement can be changed among courses in CourseSchedulingGUI interface. When a course is clicked, a popup menu containing all the possible course placements will occur as shown in Figure 3.5.2.1. Then the new course placement can be selected from the popup menu and shown to the user. If the courses selected by user are in the conflict course list, then the corresponding conflict courses for this course will be also displayed to the user. If the changing of the course placement causes a conflict with other courses, an extra symbol (*) will be added to these courses and displayed to the user. This functionality is an important user requirement for course tool.

The drop down menu shown in Figure 3.5.2.1 contains three items: “Autumn Schedule”, “Spring Schedule” and “Year Schedule”, which enable the user to view the scheduling by different ways.

2) CourseListGUI Interface

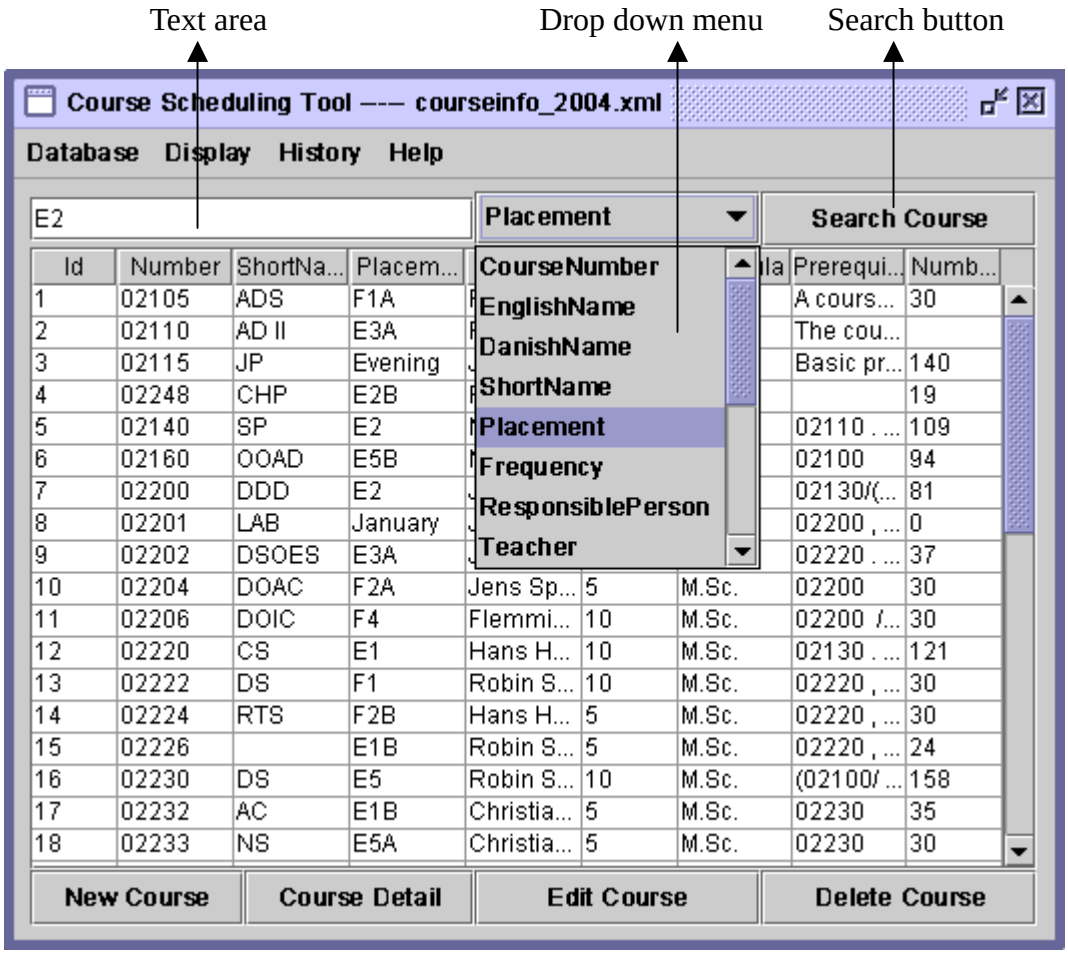


Figure 3.5.2.2 CourseListGUI

CourseListGUI interface is designed to display all the courses in the current database. It has the functionality to add, view, edit or delete courses by the four buttons in the bottom, which are “New Course”, “Course Detail”, “Edit Course” and “Delete Course”. If one of the buttons is selected, the current interface is switched to CourseInfoGUI interface, which will be introduced in the following page.

The search function is provided in the CourseListGUI interface, and the user can find specified courses by it. The search can be processed by selecting the course properties in the drop down menu and together with the key words the user inputs in the text area in Figure 3.5.2.2.

3) CourseInfoGUI Interface

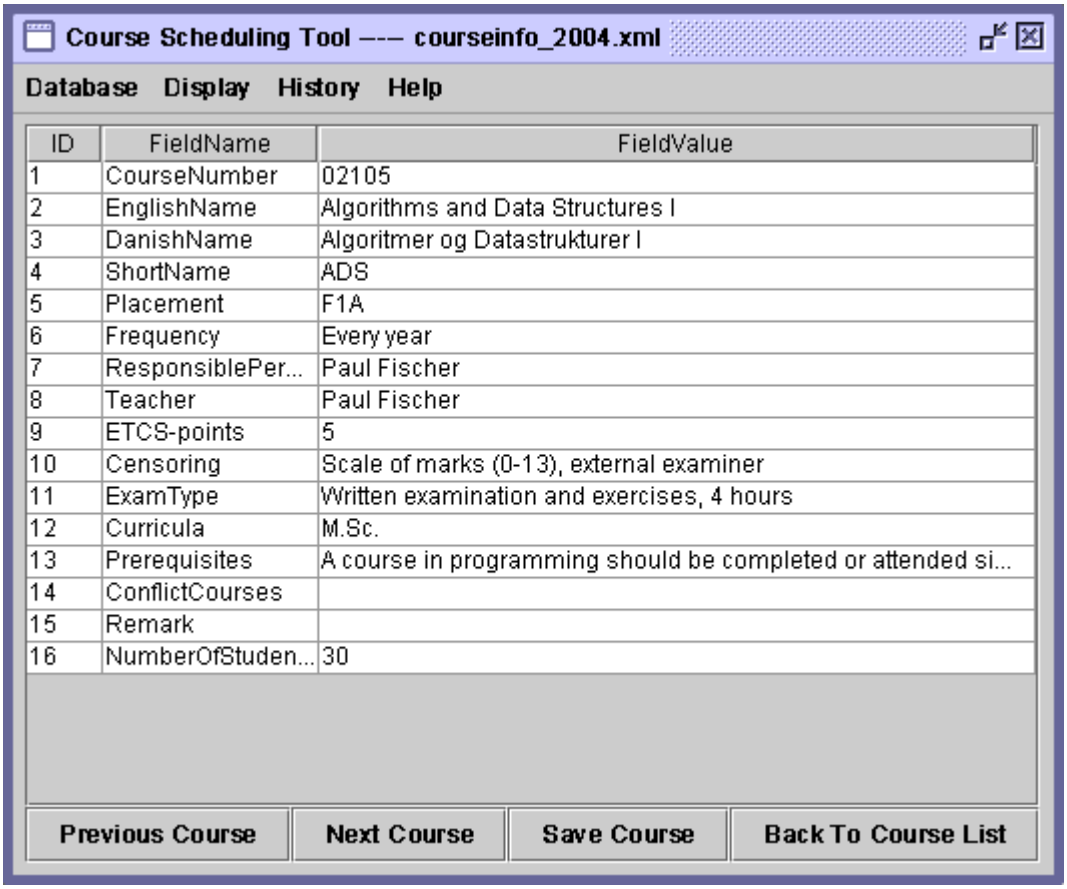


Figure 3.5.2.3 CourseInfoGUI

The four bottom buttons realizes the functionality of CourseInfoGUI interface. They are "Previous Course," "Next Course", "Save Course" and "Back to Course List" respectively. The "Save Course" button is used to save the changed course data, for example after a new course is added or course data is edited. For the user's convenient, button "Previous Course" and "Next Course" is designed to go to the previous or next course for the current course. By pressing button "Back to Course List", the current interface is switched to CourseListGUI interface.

3.6 Java Program Design

3.6.1 Design Principle

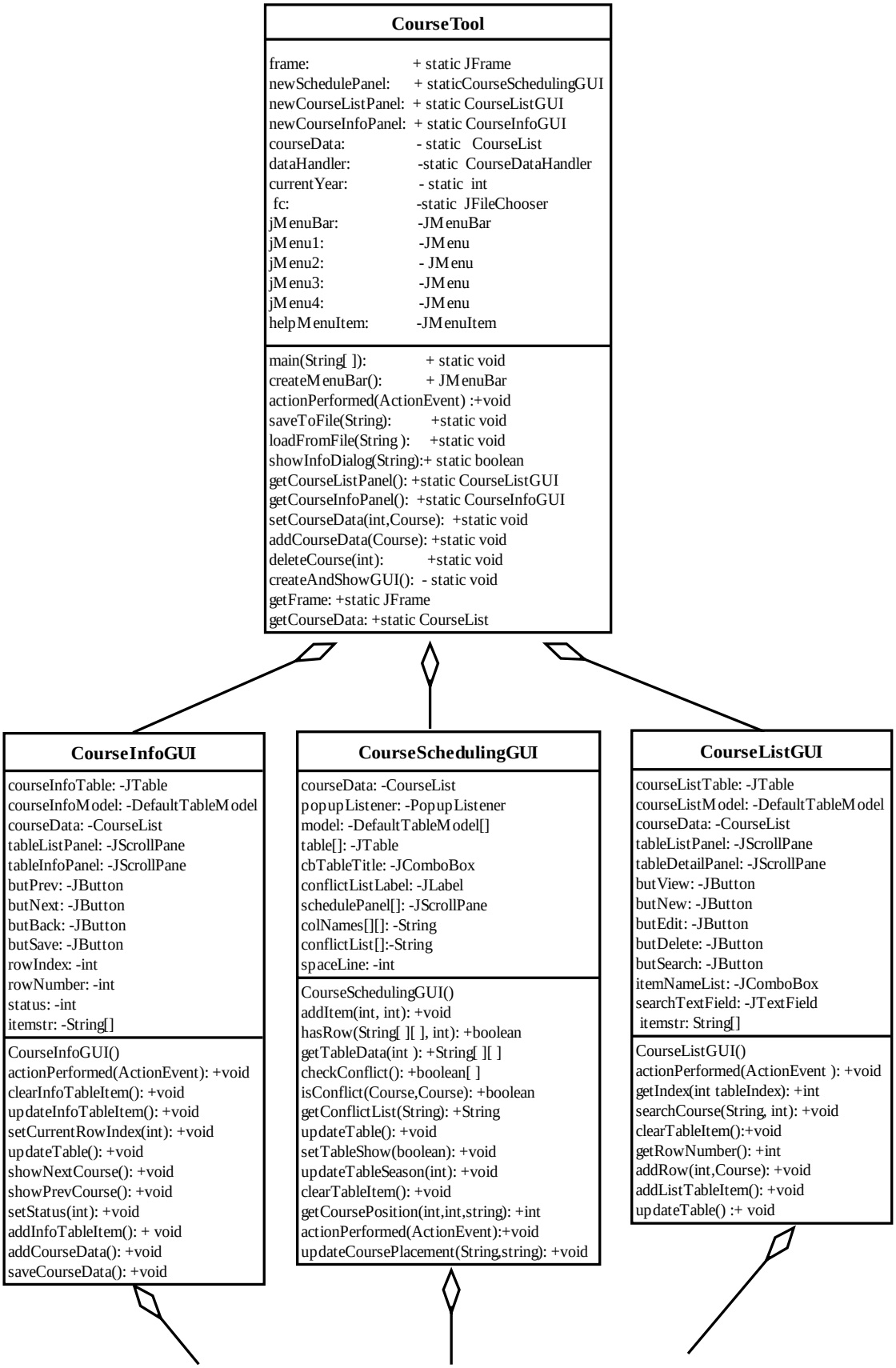
The architecture of any program written in an object oriented programming language (OOP) should follow some basic rules in order to keep the program understandable and extendable.

When various classes are introduced it is important to decide which function is associated with each class. In general, a class should contain all methods needed to manipulate all attributes, which are associated with it. This ensures that single classes can be easily extended or replaced. Often it is not necessary to know how certain methods of a class work, as long as it is specified what they are doing. This black box principle makes it possible to develop a large program with several programmers simultaneously, as they can use functions of other parts of the program without in depth knowledge of their implementation. Another advantage of OOP is the possibility to inherit certain functions and attributes from so called mother classes. This allows setting certain standards for a group of classes, without the need to implement the same method again and again. On the other hand it is also possible to define interfaces and abstract classes, which define functions and variables, which a certain group of classes have to implement. This structure is helpful to ensure that certain classes that are not yet implemented later fit in with the rest of the program. As with all programming languages, all variables and functions should be named with descriptive names so that it is possible to understand their purpose with relative ease.

3.6.2 Structure of Java Program

The program is structured into several classes, which makes the layout of the program easy to understand; on the other hand it also allows many points on which the program could be extended in the future.

The final structure of the program is shown on the next two pages. More details about each of the methods can be found in chapter 4 Implementation. Due to the simplicity of drawing, some attributes of class Course are omitted. Only coursenumber attribute is illustrated and similarly we only show the set and get methods related to coursenumber attribute.


CourseTool
frame: + static JFrame
newSchedulePanel: + staticCourseSchedulingGUI
newCourseListPanel: + static CourseListGUI
newCourseInfoPanel: + static CourseInfoGUI
courseData: - static CourseList
dataHandler: -static CourseDataHandler
currentYear: - static int
fc: -static JFileChooser
jMenuBar: -JMenuBar
jMenu1: -JMenu
jMenu2: - JMenu
jMenu3: -JMenu
jMenu4: -JMenu
helpMenuItem: -JMenuItem
main(String[ ]): + static void
createMenuBar(): + JMenuBar
actionPerformed(ActionEvent) :+void
saveToFile(String): +static void
loadFromFile(String): +static void
showInfoDialog(String):+ static boolean
getCourseListPanel(): +static CourseListGUI
getCourseInfoPanel(): +static CourseInfoGUI
setCourseData(int,Course): +static void
addCourseData(Course): +static void
deleteCourse(int): +static void
createAndShowGUI(): - static void
getFrame: +static JFrame
getCourseData: +static CourseList
CourseInfoGUI
courseInfoTable: -JTable
courseInfoModel: -DefaultTableModel
courseData: -CourseList
tableListPanel: -JScrollPane
tableInfoPanel: -JScrollPane
butPrev: -JButton
butNext: -JButton
butBack: -JButton
butSave: -JButton
rowIndex: -int
rowNumber: -int
status: -int
itemstr: -String[]
CourseInfoGUI()
actionPerformed(ActionEvent): +void
clearInfoTableItem(): +void
updateInfoTableItem(): +void
setCurrentRowIndex(int): +void
updateTable(): +void
showNextCourse(): +void
showPrevCourse(): +void
setStatus(int): +void
addInfoTableItem(): + void
addCourseData(): +void
saveCourseData(): +void
CourseSchedulingGUI
courseData: -CourseList
popupListener: -PopupListener
model: -DefaultTableModel[]
table[]: -JTable
cbTableTitle: -JComboBox
conflictListLabel: -JLabel
schedulePanel[]: -JScrollPane
colNames[][]: -String
conflictList[]:-String
spaceLine: -int
CourseSchedulingGUI()
addItem(int, int): +void
hasRow(String[ ][ ], int): +boolean
getTableData(int): +String[ ][ ]
checkConflict(): +boolean[ ]
isConflict(Course,Course): +boolean
getConflictList(String): +String
updateTable(): +void
setTableShow(boolean): +void
updateTableSeason(int): +void
clearTableItem(): +void
getCoursePosition(int,int,string): +int
actionPerformed(ActionEvent):+void
updateCoursePlacement(String,string): +void
CourseListGUI
courseListTable: -JTable
courseListModel: -DefaultTableModel
courseData: -CourseList
tableListPanel: -JScrollPane
tableDetailPanel: -JScrollPane
butView: -JButton
butNew: -JButton
butEdit: -JButton
butDelete: -JButton
butSearch: -JButton
itemNameList: -JComboBox
searchTextField: -JTextField
itemstr: String[]
CourseListGUI()
actionPerformed(ActionEvent): +void
getIndex(int tableIndex): +int
searchCourse(String, int): +void
clearTableItem():+void
getRowNumber(): +int
addRow(int,Course): +void
addListTableItem(): +void
updateTable() :+ void

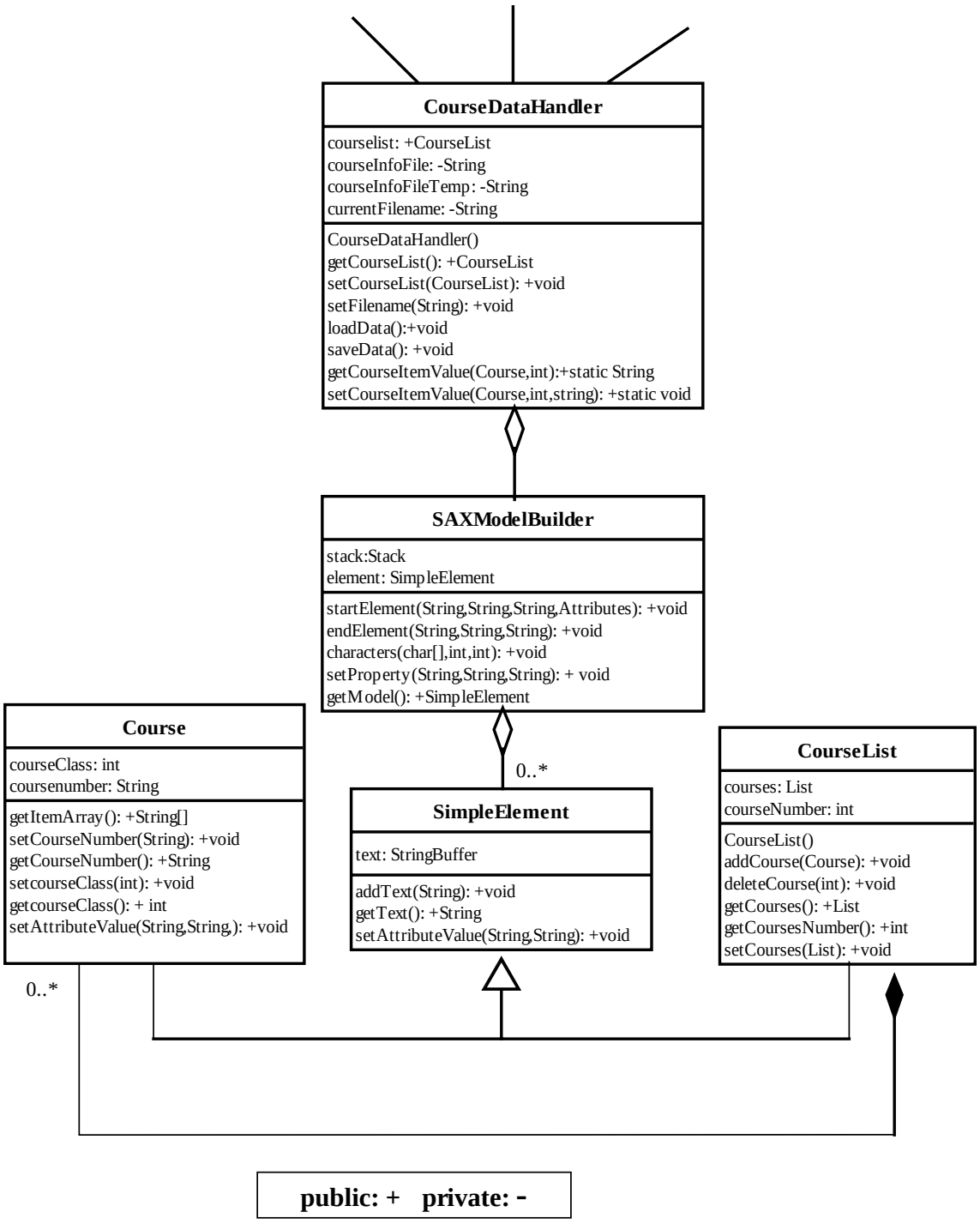


Figure3.6.2.1 Structure of Java program

Our data format is designed to XML format. The course information in the database is stored as an XML document with the attributes as tags. Therefore a tool is required to parse the XML document. So class SAXModelBuilder is created for this purpose.

Each course list consists of a certain number of courses. Therefore it was reasonable to write a class course, which contains all course information as well as the methods needed to manipulate them. Similarly, class CourseList is implemented to work with the operations on all the courses.

For convenience, we'll have our model objects extend an abstract SimpleElement class that handles text content for any element. It helps us in two ways. First, it provides a method allowing us to pass attributes to the model class. Next, we use SimpleElement as a placeholder when no class exists for an element, allowing us to store the text of the tag. Both class Course and class CourseList are subclasses of SimpleElement.

The data transfer between database and Java program demands the programming of another class, CourseDataHandeler, which loads all the course data from the database or saves the course data to the database.

Class CourseSchedulingGUI, class CourseListGUI and class CourseInfoGUI are then implemented as graphic user interfaces.

Finally, the class CourseTool was introduced as the core function of the system. It is designed to contain the main method of the program. This class also coordinates the interface transfer among CourseSchedulingGUI, CourseListGUI and CourseInfoGUI. Furthermore, it also contains all menus needed for the system.

Chapter 4 Implementation

The class containing the main method is called CourseTool. The default course data is loaded by using methods of the class CourseDataHandler, which performs the data transformation between user interface and course database. The rest of this chapter gives a detailed overview over all the methods used in the program in order to make the program more understandable.

4.1 Class URLReader

The Java class URLReader is used to get the course information from IMMs web page and then writes them to the result file courseInfo.txt. It is executed only once, only at the first time when we generate the course data.

The following is the pseudocode, which describes how class URLReader works.

```
Start
   Get all the courseIDs from IMMs homepage
   Create .txt file named courseinfo.txt
   Write XML header
   For each courseID
         Read corresponding course information from IMMs homepage
         Write the course information into courseinfo.txt
   End for
   Write end tags for XML
End
```

The detail function description is as below

main

- Within this method routines getCourseID and createCourseFile are called. The results of the routine getCourseID are stored in the variable array *courseID*, which takes as the first parameter of method createCourseFile.

getCourseID

- Return the *courseID* as a string array according the IMM course information URL, which is a parameter of this method.

createCourseFile

- Get the file courseInfo.txt.

getCourseData

- Get information for one specified course. Routine getCourseItem is called in this method.

getCourseItem

- Get the attribute value for course.

getVersionNumber

- Get the version number of one course. The default course version number is integer 3. If the course is in the special course list stored in variable *specalCourse* as an array, the course version number will be set to integer 1.

4.2 Class txtToXml

This class only contains the main method. Its functionality is to convert file courseInfo.txt to courseInfo.xml. It is also executed only once like class URLReader.

4.3 Class CourseTool

This class contains main method of the system. It is a subclass of Jframe. The menu bar in the user interface is created in this class. All the actions performed by these menu items are also implemented here. By executing the routine createAndShowGUI, the whole program begins. The default user interface is CourseSchedulingGUI, and courses are scheduled by year. The default database file is courseinfo_2004.xml.

The detail function description is as below

main

- The whole program begins here by calling routine createAndShowGUI.

createAndShowGUI

- Create and show the default user interface. Routines CourseDataHandler.loadData and CourseDataHandler.getCourseList are called within this method to load the course data from the default database file.

createMenuBar

- Here the menu bar is created. It contains four menu items. According to the order from left to right they are given as the following: “Database”, “Display”, “History” and “Help”.

actionPerformed

- After the user selects the menu item defined above, an action event will occur and be automatically transferred to the method actionPerformed as a parameter. Then actionPerformed method is invoked to execute the corresponding action.

saveToFile

- Save the current course data to database file. The database file name is checked before saving. If the file name is not empty, then routine Handler.setFilename will be called and sets the current file name. Otherwise the default database file name will be used.

loadFromFile

- Load course data from a XML file. A file chooser is used to select which data file to load.

showInfoDialog

- Show a confirmation dialog before running the user selected operation.

getCourseListPanel

- Return the current instance of class CourseListGUI, which is a sub class of JPanel

getCourseInfoPanel

- Return the current instance of class CourseInfoGUI, which is a sub class of JPanel

getCourseData

- Return the current course data.

getFrame

- Return the current frame.

setCourseData

- Update the modified course information into the current course data.

addCourseData

- Add a course to the end of the current course list.

deleteCourse

- Delete a course from a given position of the current course list.

4.4 Class CourseSchedulingGUI

It inherits the class JPanel. CourseSchedulingGUI interface will be displayed after the menu item “Schedule Display” has been selected. Courses can be scheduled by autumn, spring or year. The default course scheduling is by year, which is ordered by autumn morning, autumn afternoon, spring morning and spring afternoon. Course placement can also be changed in this interface. The possible course placements are stored in a string array variable *colNames*. Conflict course are listed in variable *conflictList*.

Important method: **isConflict.**

This method is to check if two course are conflict according to the given conflict list. The conflict theory introduced in section2.4 is used.

In our course database the conflict courses are stored in a string array variable *conflictList* as below:

```
"02222,02232,02233",
"02222,02240",
"02240,02232,02233",
"02222,02224",
"02233,02236";
```

The courses in each line are conflict each other. The corresponding conflict graph is in the following figure:

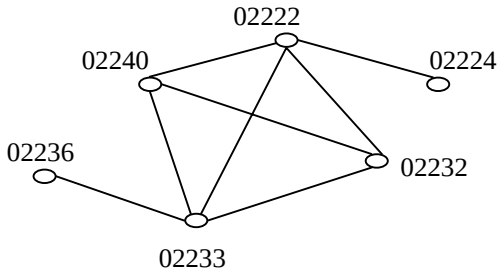


Figure 4.4.1 Conflict graph

In this method every edge in figure 4.4.1 is checked by a for loop

```
for (int j=0;j<conflictList.length;j++){
   if((conflictList[j].indexOf(courseNumber1)!=-1)
   &&(conflictList[j].indexOf(courseNumber2)!=-1))
                isconflict=true;
 }
```

If two courses are conflict, then Boolean variable *isconflict* will return true, otherwise return false.

The detail function description is as below:

CourseSchedulingGUI

- This method is the constructor of class CourseSchedulingGUI. No parameter is designed for this method. The components in this interface are set by GridBagLayout. A popup menu is created when the course number in the scheduling table is selected. Then the course placement can be changed.

addSchedulingTableItem

- There are two parameters in this method. The first one specifies the table number to be added; and the second one specifies the type of the scheduling table. This method adds the course information to the corresponding scheduling table. The size of the scheduling table is fixed. The empty scheduling table cell will be filled by null.

getTableData

- This method is implemented to get the scheduling table content according to the table type given as parameter. Routine checkConflict is called to check if two courses are conflict. If conflict, courses are displayed by adding symbol **(*)** at the end of the course name.

checkConflict

- Check if two courses are conflict. If conflict, return true, otherwise false.

getCoursePosition

- This method returns an integer value, which represents the course placement in the corresponding four scheduling tables described above.

setTableShow

- The parameter for this method is a Boolean value. If true, courses are scheduled by year. All the four scheduling tables are visible. Otherwise course are scheduled by spring or autumn. In either case only two scheduling tables are displayed.

actionPerformed

- It will call method updateTableSeason, if the user makes a selection for the drop down menu. Otherwise it will call method updateCoursePlacement if the popup is selected.

updateTableSeason

- When the course schedule is changed, this method will be called. First update the current scheduling table information, and then add the corresponding course information.

clearTableItem

- Delete all the course information from the current table.

updateCoursePlacement

- After the course placement is changed, this method will update the content of current interface by calling method updateTable. Method CourseDataHandler and setCourseItemValue are also called within this routine.

updateTable

- Update current scheduling table content by calling routine clearTableItem and addSchedulingTableItem.

hasRow

- Check if the row of the scheduling table is empty.

getConflictList

- Check if the current course has conflict courses. If it has, the corresponding conflict courses will be returned. Otherwise return a string "No Conflict Courses".

4.5 Class CourseListGUI

This class inherits the class JPanel. CourseListGUI will be displayed after the menu item “Course Display” has been selected. Button “Search Course” can be used to search courses according to the input keywords. Course related operations can be done by the four buttons: “New Course”, “Course Detail”, “Edit Course” and “Delete Course”.

The detail function description is as below

CourseListGUI

- This method is the constructor of class CourseListGUI. Within this method, the course list table containing all course information is created. Four buttons related to course operations are set together with the above table component according to GridBagLayout.

addListTableItem

- Add an item to the current course list table. Routine addRow is called to help this method adding all the courses of the current database into the course list table.

addRow

- Add a row to the current course list table.

actionPerformed

- This method listens to the action event performed in the frame, and then takes the corresponding action.

updateTable

- Update the content of current course list table by calling method clearTableItem and addListTableItem.

clearInfoTableItem

- Delete all the course information from the current course list table.

searchCourse

- Search courses with keywords and course attributes. The search results are shown on the course list table.

getIndex

- Get the index in a course list table.

getRowNumber

- Count the total courses number in the course list table.

4.6 Class CourseInfoGUI

Class JPanel is the supper class of class CourseInfoGUI. Through this class, new course information can be added and course information can be viewed or edited. The modified course information can be updated by clicking the button “Save Course”.

The detail function description is as below

CourseInfoGUI

- This method is the constructor of class CourseInfoGUI. Within this method the course info table containing information of one course is created. Four buttons related to course operations are set together with the above table component according to GridBagLayout.

setStatus

- The value of status is set in this method. It is up to which button is pressed in the CourseListGUI. For button “New Course”, status is set to 0. In this case, button “Previous Course” and “Next Course” are set disable. For button “Course Detail”, status is set to 1. In this case， only “Save Course” are set disable. For “Edit Course”, status is set to 2. In this case nothing is set disable.

addInfoTableItem

- The content of courseInfoTable is added by this method. Normally the detail course information will be added here except that the value of status is equal to zero, and in this case an empty content will be added.

actionPerformed

- This method listens to the action event performed by the four buttons and then takes the corresponding action.

showPrevCourse

- Display the previous course information of the current course. If the current course is the first, the previous course will be the last course of the current course list.

setCurrentRowIndex

- Set the current index in the course info table.

clearInfoTableItem

- Delete the content of the course info table.

updateInfoTableItem

- Update the content of current course list by calling method clearInfoTableItem and addInfoTableItem.

showNextCourse

- Display the next course information for the current course. If the current course is in the last, the next course will be the first course in the current course list.

addCourseData

- Add a new course to the current course list.

saveCourseData

- Update the modification of the course data to the current course list.

updateTable

- Update the content of course info table with the current course list.

4.7 Class CourseDataHandler

This class is responsible for transfer data between user interface and database. It loads the database data by using method loadData and saves data to the database by using method saveData.

Important methods: loadData and saveData

loadData

This method checks the file name first. If the file name is empty, nothing will be loaded. Otherwise an XML file will be parsed and the results will be saved as java objects. To perform the parsing, a parser from the javax.xml.parsers package is needed. The process of getting a parser is abstracted through a factory pattern, allowing different parser implementations to be plugged into the Java platform. A SAXParser object and an XMLReader are constructed to parse a file. To read an XML document with SAX, an instance of class SAXModelBuilder is registered to the parser. Class SAXModelBuilder will be introduced later.

saveData

An object of class BufferedWriter is used to write the course data into a XML format file. The last updated time is also added to this XML file. The root element, < *CourseList* >, contains all the < *Course* > elements as children. < Course > contains sixteen simple text elements for things like "CourseNumber", "EnglishName", "DanishName" and so on. Finally, note that the < Course > element has one attribute (class) that describes the type of course. The routine getCourseItemValue is called for getting the course information.

The detail function description is as below

CourseDataHandler

- This constructor sets the default database file as courseinfo_2004.xml.

getCourseItemValue

- Get the attribute value of the course by a switch function, which takes the index of attributes as a parameter.

setCourseItemValue

- The attribute value of the course is set by a switch function, which takes the index of attributes as a parameter.

setCourseList

- The value of variable *courselis*t is set as its parameter.

getCourseList

- Return the value of variable *courselis*t.

setFilename

- The current file name is set within this method.

4.8 Class SAXModelBuilder

The SAXModelBuilder we create in this class receives SAX events from parsing an XML file and constructs classes corresponding to the names of the tags. The method startElement, endElement, and characters receive information from the XML document. Our model builder is simple, but it handles the most common structures: elements with text or simple element data.

The detail function description is as below

startElement

This method is called when an opening tag in XML document is encountered. Because SAX events follow the structure of the XML document, we use a simple stack to keep track of which object we are currently parsing in this method. At the start of each element, the model builder attempts to create an instance of a class with the same name and push it onto the top of the stack. Each nested opening tag creates a new object on the stack until we encounter a closing tag.

endElement

This method is called when a tag is closed. Upon reaching an end of the element, we pop the current object off the stack and attempt to apply its value to its parent (the enclosing element), which is the new top of the stack. The final closing tag leaves the stack empty, but we save the last value in the result variable.

characters

This method can be invoked repeatedly to supply more text as it is read, but it often gets the whole string in one byte.

setProperty

This method uses reflection and the standard JavaBeans naming conventions to look for the appropriate property "setter" method to apply a value to its parent object. First we check for a method named add<Property> or set<Property>, accepting an argument of the child element type (for example, the addCourse (Course course) method of our CourseList object). Failing that, we look for an "add" or "set" method accepting a String argument and use it to apply any text content of the child object.

getModel

This method returns the parsing result.

4.9 Class SimpleElement

We create objects for each of the complex element types in our XML using the standard JavaBeans property design patterns ("setters" and "getters") so that our builder can automatically use them later. For convenience, we'll have our model objects extend a base SimpleElement class that handles text content for any element.

The detail function description is as below

addText

- Append the parameter string to an object of class StringBuffer.

getText

- Convert the result to a string.

setAttributeValue

- This method is an abstract class and it is not implemented in this class.

4.10 Class Course

This class is a sub class of class SimpleElement. In this class abstract method setAttributeValue is implemented. In order to match our class SAXModelBuilder, most of the methods related to course are implemented as set<attribute> and get<attribute>. In the following we only show the setCourseNumber and getCourseNumber methods. The methods related to other attributes are omitted.

The detail function description is as below

setCourseNumber

- Set the value of CourseNumber attribute.

getCourseNumber

- Return the value of CourseNumber attribute.

setcourseClass

- Set the type of course.

getcourseClass

- Return the type of course.

getItemArray

- In this method all the attributes of course will be returned as a string array.

setAttributeValue

- Set the value of the class property for course.

4.11 Class CourseList

This class inherits also from class SimpleElement. Courses are described by List in this class. An instance of class CourseList can contain zero or more instances of class Course. All the operations related to course list are implemented in this class.

The detail function description is as below

CourseList

- This method is the constructor of this class. In this method, course number is initialized to zero.

addCourse

- Add a course at the end of the current course list and increase the total course number by one.

deleteCourse

- Delete the selected course from the current course list and decrease the total course number by one.

setCourses

- Set the value of a course list.

getCourses

- Return the current course list.

getCoursesNumber

- Get the total number of a course list.

Chapter 5 Tests and Results

Test is an indispensability step in the process of software engineering, especially for large system. Although our system is not so large, the test is still needed. Usability of our system for course tool will be tested. The usability will be evaluated according to the user requirements.

All tests have been performed on a standard PC with an Intel CPU 2.60 GHz processor and 512 MB RAM under Windows 2000. Java code was used in Java2 JDK 1.4.1.

The testing consists of the following tasks:

1) Testing of Menu bar
2) Testing of CourseSchedulingGUI interface
3) Testing of CourseListGUI interface
4) Testing of CourseInfoGUI interface

The following table shows the detail test operations and the results.

Test part	Operations	Result
Menu bar	Select every item on the Menu bar and test if all the menu items work as the corresponding function.	Ok
CourseSchedulingGUI	1) Test the drop down menu by selecting button “Autumn Schedule”, “Spring Schedule” and “Year Schedule”. 2) Change course placements among courses with and without conflict.	Ok
CourseListGUI	1) Search courses with different keywords and the corresponding attribute. 2) Click the buttons at the bottom to add, view, edit and delete courses.	Ok
CourseInfoGUI	1) Go through all the courses by clicking button.”Previous Course” and “Next Course”. 2) Modify courses and save to the system. 3) Go back to CourseListGUI by clicking the button “Back to Course List”.	Ok

Table 5.1 Tests and Results

According to our test, the system works as user requirements. All the menu items and buttons show the correct performance.

Chapter 6 Conclusions and Future Prospects

6.1 Conclusions

This thesis is done for the Informatics and Mathematical Modelling Department in Technical University of Denmark. In this paper, we present a software tool designed to aid the administration with the tedious and time-consuming course scheduling tasks that every semester administration needs to go through.

The designed application can virtually eliminate the time that the administration would spend on scheduling courses, optimizing the arrangement for classroom and facilitating teacher’s time, thus allowing time for more specific and important issues.

Now the thesis is finished as expected. The construction of the thesis involves three main sections: database, user interface and Java program.

First, refer to the database part. The design task was ended completely. It works as the project requirements. Database is designed and implemented by XML, which makes the data structure more flexible and easy to extend in the future.

Second, the user interface part. A user interface design focusing on usability was developed using Java Swing. The friendly graphic user interface is easy to operate by the user. It has made both of them communicate with each other successfully.

Third, for the Java program part. Every function is accomplished as designed. The design principle is easy to replace classes or to extend the whole program by the implementation of additional classes later.

Currently we have successfully tested this course scheduling tool and it provided excellent results as requirement specification. We hope this Course Scheduling Tool will give help with course scheduling in IMM department later.

6.2 Future Prospects

Although in its current stage the tool has enough features to be conveniently used for administration, there are still features that need attention.

- The program can also take in consideration the total number of students a course can maximum have. It will make the scheduling more accurate and efficient.

- Once the school specific data and requirements have been set, for any specific teacher information, the tool will output the reasonable course schedule automatically.

- The conflict course list is given in advance for the current system. An extra function can be needed, if the system can load the conflict course list from an external file.

- The software will be probably converted to a completely web based interface too, which would link and maintain a database, which stores all the course majors and all the students information as well.

Appendix 1 References

[1] *The Java(TM) Programming Language*
Publisher: Addison-Wesley Pub Co; 3rd edition (June 5, 2000)
ISBN: 0201704331

[2] *A Guide to Constructing GUIs*
Publisher: Addison-Wesley Professional; 2nd edition (February 27, 2004)
ISBN: 0201914670

[3] *An Introduction to Object-Oriented Programming*
Publisher: Skylight Pub (May 1, 2001)
ISBN: 0965485331

[4] *A Guide to SAX, DOM, JDOM, JAXP, and TrAX*
Publisher: Addison-Wesley Professional; 1st edition (November 5, 2002)
ISBN: 0201771861

[5] *Solutions to Real-World Problems*
Publisher: O'Reilly; 2nd edition (September, 2001)
ISBN: 0596001975

[6] *Java Swing*
Publisher: Robert Eckstein, Marc Loy, Dave Wood; 1st Edition (September, 1998)
ISBN: 156592455-X

[7] *Learning Java*
Publisher: Pat Niemeyer, Jonathan Knudsen; 2nd Edition (June, 2002)
ISBN: 0596002858

[8] *Introduction to Graphics with JAVA -Swing*
Publisher:Paul Fischer; Version 1.0 (December 17, 2002)

[9] Sun Microsystems. Java native interface, May 1997.
http://java.sun.com/products//jdk/1.2/docs/guide/jni/index.html.

[10] Java™ 2 Platform Standard Edition 5.0 API Specification
http://java.sun.com/j2se/1.5.0/docs/api/index.html

Appendix 2 User Manual

This manual has briefly described the usage of course scheduling tool developed in this thesis. The tool is designed and implemented to aid the administrator to schedule the courses. In the following we will give some instructions to guide the user how to use it.

1 System Requirements

The system can be Microsoft Windows, Linux or Unix. The tool is written in Java code. So Java 2 SDK 1.4.1 or higher is needed to interpreter the program.

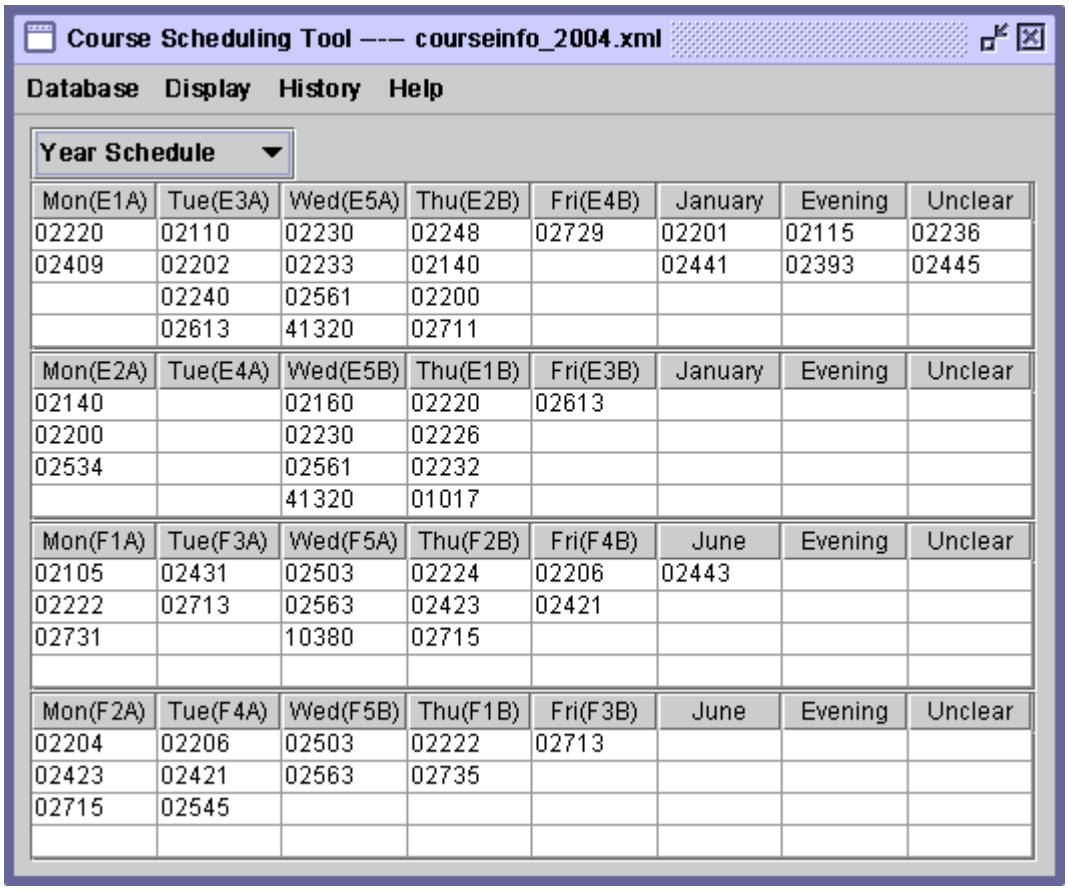
2 Starting Program

The simplest way to do this is typically to invoke it directly from the command line with a command such as:

java CourseTool

.

The application window is shown in Figure1 as below:



Mon(E1A)	Tue(E3A)	Wed(E5A)	Thu(E2B)	Fri(E4B)	January	Evening	Unclear
02220	02110	02230	02248	02729	02201	02115	02236
02409	02202	02233	02140		02441	02393	02445
	02240	02561	02200				
	02613	41320	02711				

Mon(E2A)	Tue(E4A)	Wed(E5B)	Thu(E1B)	Fri(E3B)	January	Evening	Unclear
02140		02160	02220	02613			
02200		02230	02226				
02534		02561	02232				
		41320	01017				

Mon(F1A)	Tue(F3A)	Wed(F5A)	Thu(F2B)	Fri(F4B)	June	Evening	Unclear
02105	02431	02503	02224	02206	02443		
02222	02713	02563	02423	02421			
02731		10380	02715				

Mon(F2A)	Tue(F4A)	Wed(F5B)	Thu(F1B)	Fri(F3B)	June	Evening	Unclear
02204	02206	02503	02222	02713			
02423	02421	02563	02735				
02715	02545						

Figure 1

The default course database is for year 2004, and the corresponding file name of database courseinfo_2004.xml is displayed as the title of this window. The default course schedule is Year Schedule.

3 Database Menu

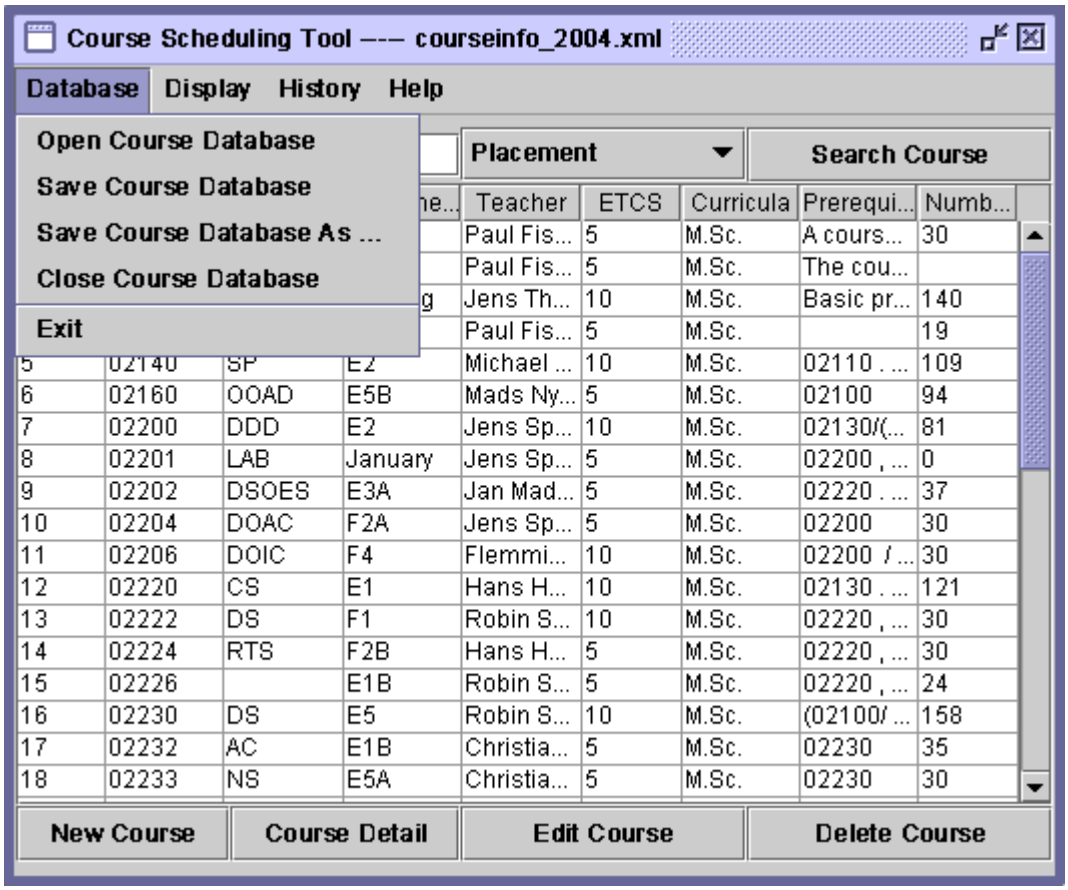


Figure2

To open a new database file, select “Open Course Database” from “Database”. To save changes to current database file, select “Save Course Database”. To save changes to another database file, select “Save Course Database As”. To close current database file, select “Close Course Database”. To close current window, select “Exit.”

4 Display Menu

There are two sub menu items under Display menu: “Course Display” and “Schedule Display”. Select “Course Display” can go to interface containing all the courses. Select “Schedule Display” can go to course scheduling interface.

5 Help Menu
Select “Help” menu item can make user get help information.

6 History Menu
To view the previous database file, select one of sub items of “History” as below:

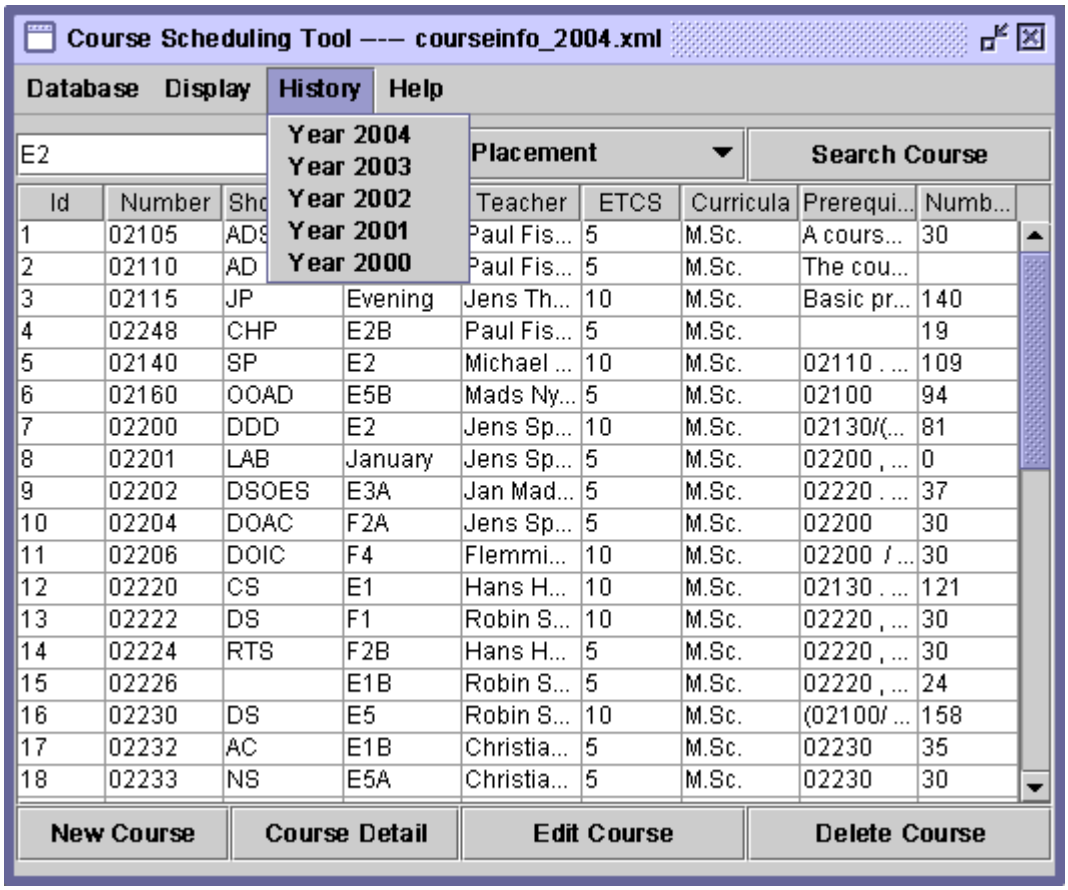


Figure3

7 Change Course Schedule

User can change the course schedule by selecting the items in the drop down menu shown as below .For example in the following figure 4 “Autumn Schedule” is selected.

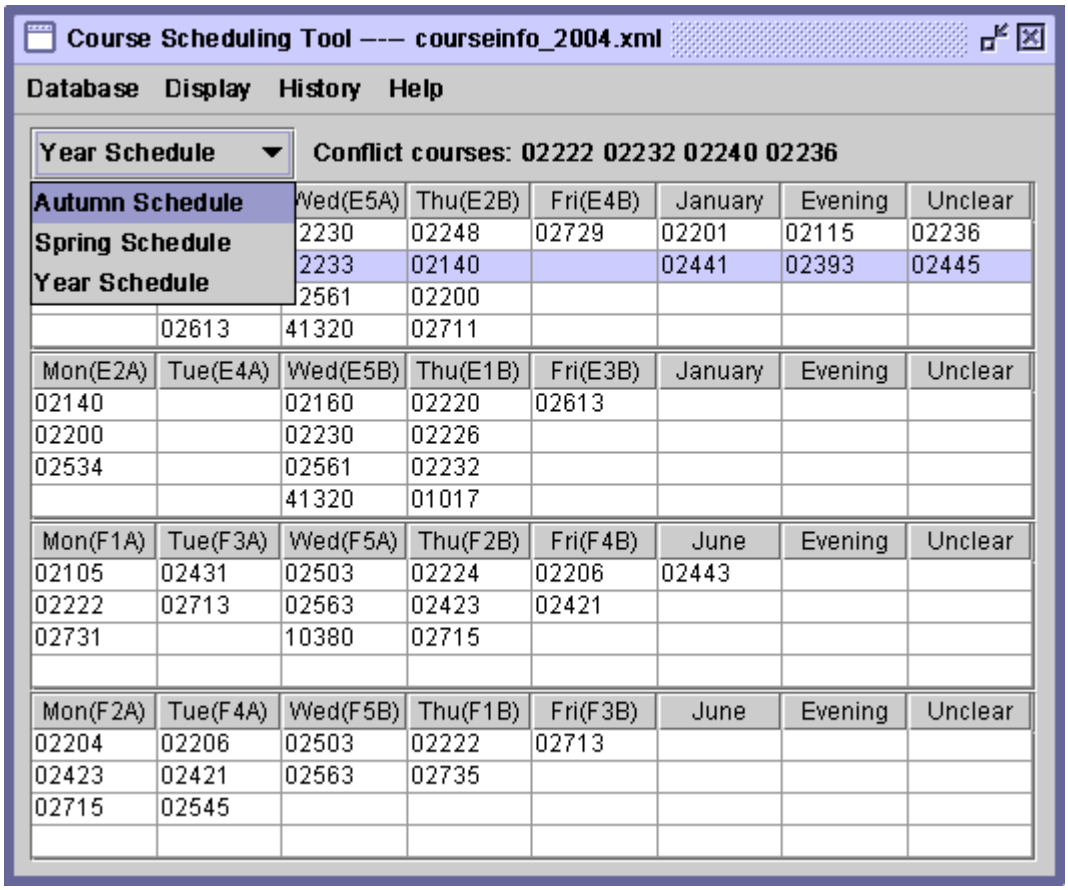


Figure 4

8 Change Course Placement

The course placement can be changed among courses in the course scheduling interface by following steps. First click the course you want to change, and then a popup menu containing all the possible course placements will occur and we can select the new course placement for the selected course. Afterwards the course placement will be updated and displayed to the user. If the courses selected by user are in the conflict course list, then the corresponding conflict courses with this course will be also displayed to the user. If the courses in the conflict course list have the same course placement except Unclear, an extra symbol (*) will be added to these courses and displayed. For example course 02233 is selected, and we want to change its course placement to Thu(E1B) in figure 5.

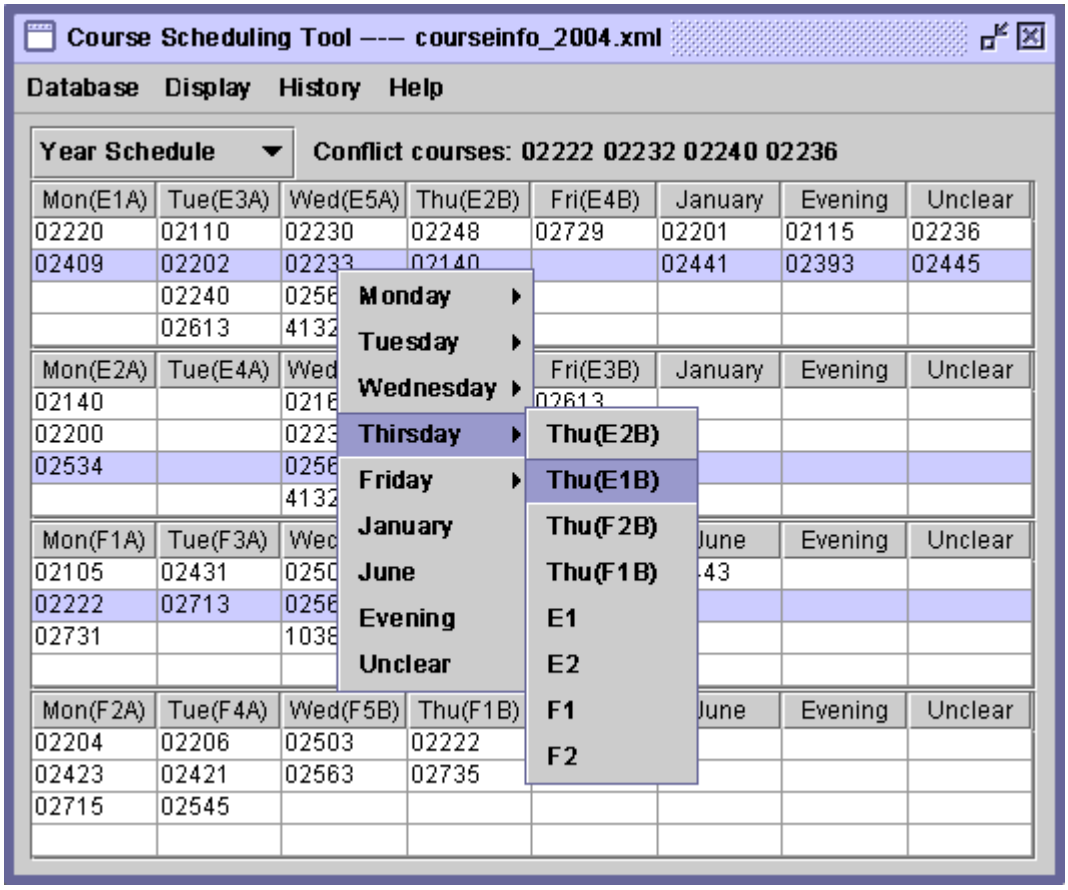


Figure5

9 Display All the Courses

Select “Course Display” from “Display” menu item, we can get all the courses in the current database shown in Figure 6.

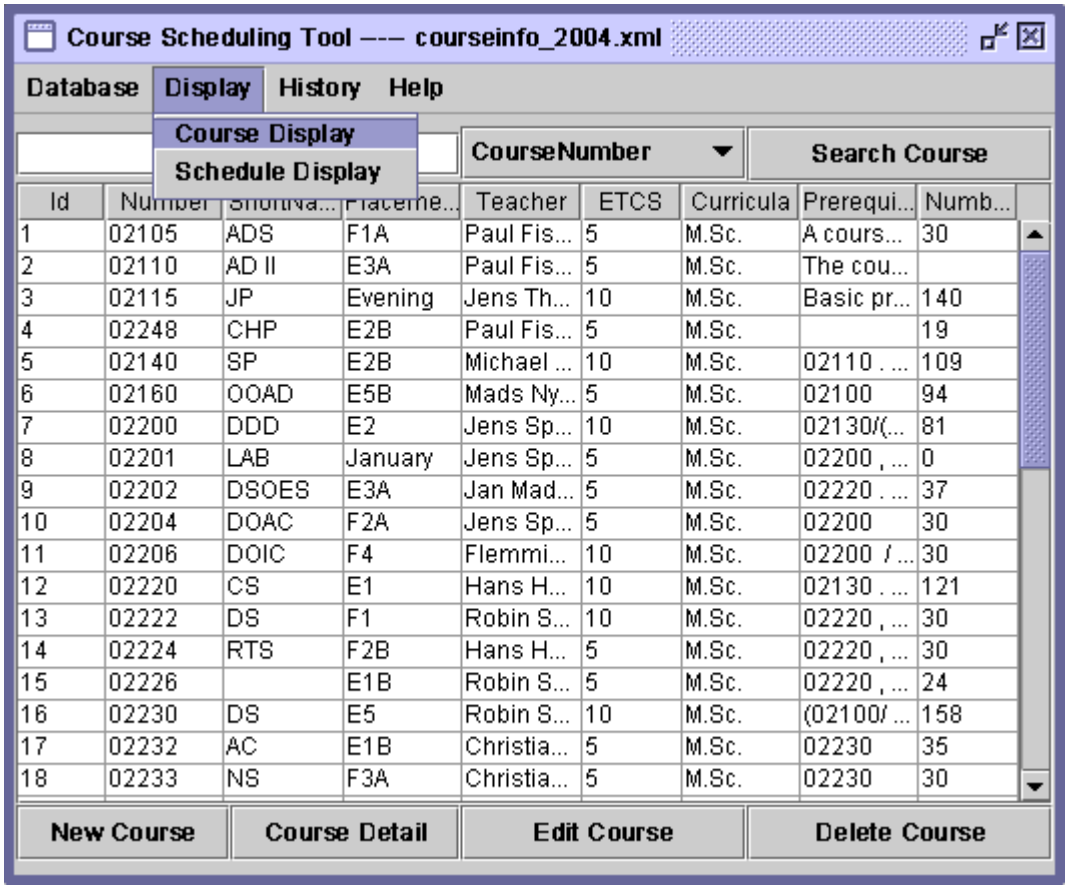


Figure 6

In this user interface user can add a new course to the database by selecting button “New Course”, view a course detail by selecting button “Course Detail”, edit a course data by selecting button “Edit Course”, or delete a course from database by selecting button “Delete Course”.. We can also search courses through button “Search Course”.

10 Add a New Course

Select New Course button in Figure 6, we can add a new course.

11 Display Course Detail

Select “Course Detail” button in Figure 6, we can view a course detail.

12 Edit Course

Select “Edit Course” button in Figure 6, we can edit a course.

13 Delete Course

Select “Delete Course” button in Figure 6, we can delete a course

14 Search Course

To search courses, first input some key words in the text area in the following figure 7, then select the corresponding item in the drop down menu, finally select button “Search Course”**.**

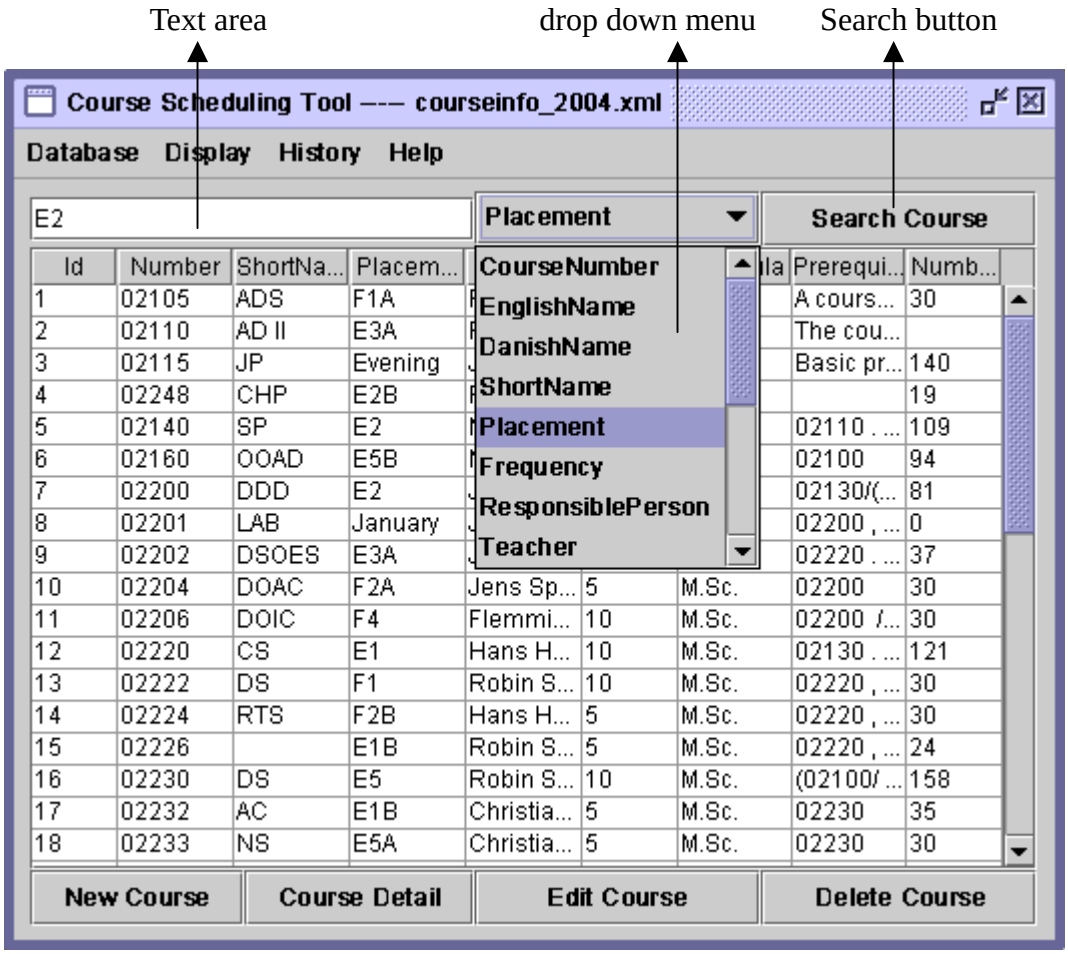


Figure 7

Appendix 3 Source Code

1 Class URLReader

```
/* File Name: URLReader.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: Web course data reader
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

/* This class reads the course information from DTU web set, and then transfers it
to a text file*/

import java.net.*;
import java.io.*;

public class URLReader
{
    public static String itemName[]={"courseNumber",    //0
           "<label class=\"header_white\">",    //1   EnglishName
            "Danish title:",    //2
            "",    //3
           "Schedule",    //4
            "",    //5
           "Responsible",    //6
            "Responsible",    //7
           "Credit Points (ECTS):",    //8
            "Evaluation:",    //9
            "Evaluation:",    //10
            "Type:",    //11
            "Prerequisites:",    //12
           "Not applicable",    //13
           "Remarks",    //14
           ""   //15  };

    public static void main(String[] args) throws Exception
    {
      String[] courseID=new String[200];

     courseID=getCourseID("http://www.kurser.dtu.dk/search/search-result.asp?men
     ulanguage=dk&txtCourseNumber=&txtSearchKeyword=&lstDepartment=2&lstLanguage
     =&lstYearGroup=2004-08-01&lstYearGroup2=20042005&lstEducation=&lstCourseTyp
     e=&btnSearch=S%F8g+i+kursusbasen");

     createCourseFile(courseID, "courseinfo.txt");

    }

// Get course id
    public static String[] getCourseID(String inURL)throws Exception
    {
       URL myURL = new URL(inURL);

        String urlString="";

        BufferedReader in = new BufferedReader(
```

```
                                new InputStreamReader(
                                    myURL.openStream()));
       String[] courseID=new String[200];

       int index=0;

       String inputLine;

       String         seperator="<a         target=\"\"         class=\"course\"
href=\"../presentation/presentation.asp?&amp;menulanguage=dk&amp;coursecode=";

       while ((inputLine = in.readLine()) != null)

                if (inputLine.indexOf(seperator)!=-1)

    {courseID[index]=inputLine.substring(seperator.length(),seperator.length()+;
                index++;
    }
            in.close();
            return courseID;
    }

// Create txt file by course id
    public static void createCourseFile(String[] courseID, String fileName) throws
Exception
    {
            String cFile = fileName;
            FileOutputStream cOutputStream = new FileOutputStream(cFile);
            PrintStream cPrint = new PrintStream(cOutputStream);
            int index=0;
            String courseStr="";
            String[] courseData=new String[16];
            while (courseID[index]!=null)
           {
            courseStr=getCourseData(courseID[index]);
            cPrint.println(courseStr);
            index++;
            }

            cPrint.close();
            cOutputStream.close();
        }

// Get all the information of one course by course id
    public static String getCourseData(String courseID) throws IOException
    {
        String courseStr=courseID;
        int i;
        String [] courseData=new String[16];
        for(i=1;i<15;i++){
            courseData[i]=getCourseItem(courseID, i);
            courseStr=courseStr+";"+courseData[i];
        }
        courseStr=courseStr+";"+"30";
        return courseStr;
    }

// Get version of one course
    public static int getVersionNumber(String courseID)
    {
        String
specalCourse[]={"02121","02130","02160","02161","02170","02207","02225","02233",
"02236","02350","02357","02359","02402","02403","02445","02405","02534","02536",
```

```
"02569","02729","02914","01017","31330"};
        int versionID=3;
        for (int i=0;i<specalCourse.length;i++)
            if (courseID.equals(specalCourse[i])){
                versionID=1;
                break;
            }
        return versionID;
}
//Get the specified item vale of one selected course
 public static String getCourseItem(String courseID, int itemNameId) throws
IOException
    {
        int versionID=3;
        versionID=getVersionNumber(courseID);
        String itemValue="";
        String
inURL="http://www.kurser.dtu.dk/presentation/presentation.asp?menulanguage=en-gb
&coursecode="+courseID+"-"+versionID+"&version=full";
        URL myURL = new URL(inURL);
       String urlString="";
            BufferedReader in = new BufferedReader(
                        new InputStreamReader(
                 myURL.openStream()));
            String inputLine;
            String tmpStr;
            boolean isSetCensor=false;
            boolean isSetExamType=false;

            while ((inputLine = in.readLine()) != null)
            {
                if (inputLine.indexOf(itemName[itemNameId])!=-1){
                    switch(itemNameId){
                        case 1:
    itemValue=inputLine.substring(itemName[itemNameId].length()+5,inputLine.leng
th()-8);
                          break;
                        case 2:
                            inputLine = in.readLine();

    itemValue=inputLine.substring(18,inputLine.length()-5);
                            break;
                        case 4:
                            inputLine = in.readLine();
                if (inputLine.indexOf("<td class=\"value\" align=\"left\">")==-1)
                                    inputLine = in.readLine();
                            tmpStr=inputLine.substring(31,inputLine.length());
                            itemValue=tmpStr.substring(0,tmpStr.indexOf("<"));
                            break;
                        case 6:
                        case 7:
                           inputLine = in.readLine();
if
(inputLine.indexOf("http://www.dtubasen.dtu.dk/tlf.aspx?form=off&amp;type=employ
ee&amp;matrikel_id")==-1)
                            inputLine = in.readLine();
                            itemValue=getItemValue(inputLine);
                            break;
                        case 8:

itemValue=inputLine.substring(inputLine.indexOf(itemName[itemNameId])+30,inputLi
ne.length()-5);
                            break;
```

```
                        case 9:
                            if (!isSetCensor){
                                inputLine = in.readLine();
                                while(inputLine.indexOf("Evaluation:")==-1){
                                    inputLine = in.readLine();
                                    if (inputLine==null)
                                        break;
                                }
                            while(inputLine.indexOf("<label class=\"value\">")==-1){
                                        inputLine = in.readLine();
                                        if (inputLine==null)
                                            break;
                                    }
                                    if (inputLine!=null)

itemValue=inputLine.substring(inputLine.indexOf("<label
class=\"value\">")+21,inputLine.length()-8);
                                    isSetCensor=true;
                                }
                                break;
                            case 10:
                                if (!isSetExamType){
                            while(inputLine.indexOf("<label class=\"value\">")==-1){
                                        inputLine = in.readLine();
                                        if (inputLine==null)
                                           break;
                                    }

    itemValue=inputLine.substring(inputLine.indexOf("<label
class=\"value\">")+21,inputLine.length()-8);
                                    isSetExamType=true;
                                    }
                                break;
                            case 11:
                        while(inputLine.indexOf("<tr valign=\"top\">")==-1){
                                    inputLine = in.readLine();
                                    if (inputLine==null)
                                       break;
                                }
                                inputLine = in.readLine();
                                if (inputLine!=null)
                                    itemValue=getItemValue(inputLine);
                                break;
                            case 12:
                            while(inputLine.indexOf("<label class=\"value\">")==-1){
                                    inputLine = in.readLine();
                                    if (inputLine==null)
                                       break;
                                }
                                if (inputLine!=null)
                            itemValue=inputLine.substring(inputLine.indexOf("<label
class=\"value\">")+21,inputLine.length()-8);
                                break;
                            case 13:
                            while(inputLine.indexOf("<label class=\"value\">")==-1){
                                    inputLine = in.readLine();
                                    if (inputLine==null)
                                       break;
                                }
                                if (inputLine!=null)
                            itemValue=inputLine.substring(inputLine.indexOf("<label
class=\"value\">")+21,inputLine.length()-8);
                                break;
```

```
                        case 14:
                            inputLine = in.readLine();

    itemValue=inputLine.substring(inputLine.indexOf(">")+1,inputLine.length()-5)
;
                            break;
                        }
                    }

            }
            in.close();
            return itemValue;
        }

public static String getItemValue(String inputLine)
        {
            String
tmpStr=inputLine.substring(inputLine.indexOf(">")+1,inputLine.length());
            String itemValue=tmpStr.substring(0,tmpStr.indexOf("<"));
            return itemValue;
        }

}
```

2 Class TxtToXml

```
/* File Name: TxtToXml.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: Txt to Xml converter
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */


// This class transfer txt file to xml file.
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class TxtToXml
{
   public static void main(String[] args)
    {
        String[] mystr=new String[16];

       String[] itemstr={  "CourseNumber",      "EnglishName",      "DanishName",
"ShortName","Placement",  "Frequency",  "Rperson",  "Teacher",  "Ets","Censoring",
"ExamType",    "Curricula",    "Prerequisites",    "ConflictCourses",    "Remark",
"NumberOfStudents"};
        String str;
        try {
             BufferedReader       in       =       new       BufferedReader(new
FileReader("courseinfo.txt"));
              BufferedWriter      out      =      new      BufferedWriter(new
FileWriter("courseinfo.xml"));

              out.write("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n");
              out.write("<CourseList>\n");



while ((str = in.readLine()) != null)
               {       mystr=str.split(";");
                       out.write("   <Course class=\"normal\">\n");

                       for(int i=0;i<16;i++)
                       out.write("
<"+itemstr[i]+">"+mystr[i].trim()+"</"+itemstr[i]+">\n");
                       out.write("   </Course>\n");
                       }
               out.write("</CourseList>");
               out.close();
               in.close();
            }
        catch (IOException e) {
            System.err.println("File not found " + e);
        }

    }
}
```

3 Class CourseTool

```
/* File Name: CourseTool.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: Main program
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

/*This class contains Main method. The whole program starts from this class. The menu
bar is also set up in this class. The default database file is courseinfo_2004.xml
and default course schedule is set Year Schedule.*/

import java.awt.event.*;
import java.io.File;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JSeparator;
import javax.swing.JMenuBar;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JFrame;


public class CourseTool extends JFrame implements ActionListener {
    JTextArea output;
    JScrollPane scrollPane;
    String newline = "\n";
    public static JFrame frame;
    public static CourseSchedulingGUI newSchedulePanel;//for course schedule
    public static CourseListGUI newCourseListPanel;// for all courses information
    public static CourseInfoGUI newCourseInfoPanel;//only for one course information
    private static CourseList courseData;
    private static CourseDataHandler dataHandler;
    private static int currentYear;
    public static boolean isModified;
     private static JFileChooser fc;
     private JMenuBar jMenuBar;
     private JMenu jMenu1;//database menu
     private JMenuItem openDatabaseMenuItem;
     private JMenuItem newDatabaseMenuItem;
     private JMenuItem saveDatabaseMenuItem;
     private JMenuItem saveAsDatabaseMenuItem;
     private JMenuItem closeDatabaseMenuItem;
     private JSeparator jSeparator1;
     private JMenuItem exitMenuItem;

     private JMenu jMenu2;//display menu
     private JMenuItem courseInfoMenuItem;
     private JMenuItem scheduleMenuItem;

     private JMenu jMenu3;//history menu
     private JMenuItem y2004MenuItem;
     private JMenuItem y2003MenuItem;
```

```
    private JMenuItem y2002MenuItem;
    private JMenuItem y2001MenuItem;
    private JMenuItem y2000MenuItem;

    private JMenu jMenu4;//help menu
    private JMenuItem helpMenuItem;

//Set up the menu items
    public JMenuBar createMenuBar()
    {

       jMenuBar = new JMenuBar();

        jMenu1 = new JMenu();
        newDatabaseMenuItem = new JMenuItem();
        openDatabaseMenuItem = new JMenuItem();
        saveDatabaseMenuItem = new JMenuItem();
        saveAsDatabaseMenuItem = new JMenuItem();
        closeDatabaseMenuItem = new JMenuItem();
        jSeparator1 = new JSeparator();
        exitMenuItem = new JMenuItem();

        jMenu2 = new JMenu();
        courseInfoMenuItem = new JMenuItem();
        scheduleMenuItem = new JMenuItem();

        jMenu3 = new JMenu();
        y2004MenuItem = new JMenuItem();
        y2003MenuItem = new JMenuItem();
        y2002MenuItem = new JMenuItem();
        y2001MenuItem = new JMenuItem();
        y2000MenuItem = new JMenuItem();

        jMenu4 = new JMenu();
        helpMenuItem = new JMenuItem();

 // ---------------Menu Item [Database]------------------
       jMenu1.setText("Database");
       jMenu1.setVisible(true);
       jMenuBar.add(jMenu1);

    openDatabaseMenuItem.setText("Open Course Database");
    openDatabaseMenuItem.setVisible(true);
    openDatabaseMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    openDatabaseMenuItem.addActionListener(this);
    jMenu1.add(openDatabaseMenuItem);

    saveDatabaseMenuItem.setText("Save Course Database");
    saveDatabaseMenuItem.setVisible(true);
    saveDatabaseMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    saveDatabaseMenuItem.addActionListener(this);
    jMenu1.add(saveDatabaseMenuItem);

    saveAsDatabaseMenuItem.setText("Save Course Database As ...");
    saveAsDatabaseMenuItem.setVisible(true);
    saveAsDatabaseMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    saveAsDatabaseMenuItem.addActionListener(this);
    jMenu1.add(saveAsDatabaseMenuItem);

    closeDatabaseMenuItem.setText("Close Course Database");
    closeDatabaseMenuItem.setVisible(true);
    closeDatabaseMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    closeDatabaseMenuItem.addActionListener(this);
```

```
    jMenu1.add(closeDatabaseMenuItem);

    jSeparator1.setVisible(true);
    jSeparator1.setBounds(new java.awt.Rectangle(5,5,60,30));
    jMenu1.add(jSeparator1);

    exitMenuItem.setText("Exit");
    exitMenuItem.setVisible(true);
    exitMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    exitMenuItem.addActionListener(this);
    jMenu1.add(exitMenuItem);

//      ---------------Menu Item [Display]-----------------
    jMenu2.setText("Display");
    jMenu2.setVisible(true);
    jMenuBar.add(jMenu2);

    courseInfoMenuItem.setText("Course Display");
    courseInfoMenuItem.setVisible(true);
    courseInfoMenuItem.setPreferredSize(new java.awt.Dimension(27,16));
    courseInfoMenuItem.setBounds(new java.awt.Rectangle(5,5,27,16));
    courseInfoMenuItem.addActionListener(this);
    jMenu2.add(courseInfoMenuItem);

    scheduleMenuItem.setText("Schedule Display");
    scheduleMenuItem.setVisible(true);
    scheduleMenuItem.setBounds(new java.awt.Rectangle(5,5,60,30));
    scheduleMenuItem.addActionListener(this);
    jMenu2.add(scheduleMenuItem);


//      ---------------Menu Item [History]-----------------
        jMenu3.setText("History");
        jMenu3.setVisible(true);
        jMenuBar.add(jMenu3);

        y2004MenuItem.setText("Year 2004");
        y2004MenuItem.setVisible(true);
        y2004MenuItem.setPreferredSize(new java.awt.Dimension(100,16));
        y2004MenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
        y2004MenuItem.addActionListener(this);
        jMenu3.add(y2004MenuItem);

        y2003MenuItem.setText("Year 2003");
        y2003MenuItem.setVisible(true);
        y2003MenuItem.setPreferredSize(new java.awt.Dimension(70,16));
        y2003MenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
        y2003MenuItem.addActionListener(this);
        jMenu3.add(y2003MenuItem);

        y2002MenuItem.setText("Year 2002");
        y2002MenuItem.setVisible(true);
        y2002MenuItem.setPreferredSize(new java.awt.Dimension(70,16));
        y2002MenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
        y2002MenuItem.addActionListener(this);
        jMenu3.add(y2002MenuItem);

        y2001MenuItem.setText("Year 2001");
        y2001MenuItem.setVisible(true);
        y2001MenuItem.setPreferredSize(new java.awt.Dimension(70,16));
        y2001MenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
        y2001MenuItem.addActionListener(this);
        jMenu3.add(y2001MenuItem);
```

```
        y2000MenuItem.setText("Year 2000");
        y2000MenuItem.setVisible(true);
        y2000MenuItem.setPreferredSize(new java.awt.Dimension(70,16));
        y2000MenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
        y2000MenuItem.addActionListener(this);
        jMenu3.add(y2000MenuItem);

//      ---------------Menu Item [Help]-----------------
    jMenu4.setText("Help");
    jMenu4.setVisible(true);
    jMenuBar.add(jMenu4);

    helpMenuItem.setText("Help");
    helpMenuItem.setVisible(true);
    helpMenuItem.setPreferredSize(new java.awt.Dimension(70,16));
    helpMenuItem.setBounds(new java.awt.Rectangle(5,5,60,16));
    helpMenuItem.addActionListener(this);
    jMenu4.add(helpMenuItem);

    return jMenuBar;
   }

 public void actionPerformed(ActionEvent e)
   {
      JMenuItem source = (JMenuItem)(e.getSource());
      if(source.getText().equals("Course Display"))
      {
       newCourseListPanel.updateTable();
       frame.setContentPane(newCourseListPanel);
       frame.show();
      }
      if(source.getText().equals("Schedule Display"))
      {
       frame.setContentPane(newSchedulePanel);
       frame.show();
      }
      if(source.getText().equals("Help"))
      {File file = new File("help.html");
       String cmd="cmd /E:ON /c start IEXPLORE.EXE "+file.getAbsolutePath();
       try {Runtime.getRuntime().exec(cmd);
           } catch (IOException e1) {
               e1.printStackTrace();
           }
      }
       if(source==saveDatabaseMenuItem)
       {
           String dlgInfo="Are you sure to save?";
           if(showInfoDialog(dlgInfo))
           {
             saveToFile("");
             isModified=false;
           }
       }
       if(source==saveAsDatabaseMenuItem)
       {
           int returnVal = fc.showOpenDialog(frame);
           if (returnVal == JFileChooser.APPROVE_OPTION)
           {
                   File file = fc.getSelectedFile();
                   saveToFile(file.getName());
           }
           isModified=false;
```

```
        }
        if(source.getText().indexOf("Year 200")!=-1)
        {
            if (isModified)
            {
            String dlgInfo="The data in the database is modified\n Would you like
to save it before close?";
                if(showInfoDialog(dlgInfo))
                    saveToFile("");}
            String tmpStr=source.getText().substring(5,9);
            tmpStr="courseinfo_"+tmpStr+".xml";
            loadFromFile(tmpStr);
            isModified=false;
        }
        if(source==openDatabaseMenuItem)
        {
            if (isModified)
            {
            String dlgInfo="The data in the database is modified\n Would you like
to save it before close?";
                if(showInfoDialog(dlgInfo))
                    saveToFile("");}

            int returnVal = fc.showOpenDialog(frame);
            if (returnVal == JFileChooser.APPROVE_OPTION)
            {
                File file = fc.getSelectedFile();
                loadFromFile(file.getName());
                closeDatabaseMenuItem.setEnabled(true);
                saveDatabaseMenuItem.setEnabled(true);
                saveAsDatabaseMenuItem.setEnabled(true);
            }
            isModified=false;

        }
        if(source==closeDatabaseMenuItem){
            if (isModified)
            {
            String dlgInfo="The data in the database is modified\n Would you like
to save it before close?";
                if(showInfoDialog(dlgInfo))
                    saveToFile("");}
            loadFromFile("");
            closeDatabaseMenuItem.setEnabled(false);
            saveDatabaseMenuItem.setEnabled(false);
            saveAsDatabaseMenuItem.setEnabled(false);
            isModified=false;
        }

        if(source.getText().equals("Exit"))
        {
            if (isModified)
            {
            String dlgInfo="The data in the database is modified\n Would you like
to save it before close?";
                if(showInfoDialog(dlgInfo))
                    saveToFile("");}
            String dlgInfo=" Are you sure to exit?";
                if(showInfoDialog(dlgInfo))
                    System.exit(0);
            }
        }
 //Save data to database
```

```
    public static void saveToFile(String filename)
    {
        if (!filename.equals(""))
        {
            frame.setTitle("Course tool -----  "+filename);
            dataHandler.setFilename(filename);
        }

        try
        {
               dataHandler.saveData();
        }
        catch (Exception es)
        {
               es.printStackTrace();
        }

    }
//Load data from database
    public static void loadFromFile(String filename)
    {
         frame.setTitle("Course tool -----  "+filename);
        dataHandler.setFilename(filename);
        try {
               dataHandler.loadData();
               courseData=dataHandler.getCourseList();
        }
        catch (Exception es)
        {
               es.printStackTrace();
        }
        newCourseListPanel.updateTable();
        newSchedulePanel.updateTable();
        frame.show();
     }

     public static boolean showInfoDialog(String info)
     {
        int n = JOptionPane.showConfirmDialog(
              frame, info,
              "Confirmation",
              JOptionPane.YES_NO_OPTION);
       if (n == JOptionPane.YES_OPTION) {
          return true;
       } else if (n == JOptionPane.NO_OPTION) {
        return false;
       } else {
        return false;
       }
     }

    public static CourseListGUI getCourseListPanel()
    {
     return newCourseListPanel;
    }

    public static CourseInfoGUI getCourseInfoPanel()
     {
           return newCourseInfoPanel;
      }
     public static CourseList getCourseData()
     {
               return courseData;
```

```
 }
 public static void setCourseData(int rowIndex,Course mycourse)
 {
        courseData.getCourses().remove(rowIndex);
        courseData.getCourses().add(rowIndex,mycourse);

 }
 public static void addCourseData(Course mycourse)
 {
    courseData.addCourse(mycourse);

 }
 public static void deleteCourse(int rowIndex)
 {
     courseData.deleteCourse(rowIndex);

 }
 public static JFrame getFrame()
 {
    return frame;
 }
 /*
  * Create the GUI and show it.  For thread safety,
  * this method should be invoked from the
  * event-dispatching thread.
  */
 private static void createAndShowGUI()
 {
     dataHandler=new CourseDataHandler();
     try
     {
         dataHandler.loadData();
         courseData=dataHandler.getCourseList();
     }
     catch (Exception e)
     {
         e.printStackTrace();
     }

//Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);
    //Create and set up the window.
    frame = new JFrame();
    frame.setTitle("Course Scheduling Tool ----- courseinfo_2004.xml");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setResizable(false);

//Create and set up the content pane.
    CourseTool mySchedule = new CourseTool();
    frame.setJMenuBar(mySchedule.createMenuBar());
    fc = new JFileChooser();
    fc.addChoosableFileFilter(new MyFilter());
    try
    {
         File f = new File(new File(".").getCanonicalPath());
         fc.setCurrentDirectory(f);
     } catch (IOException e1)
     {
         e1.printStackTrace();
     }
     isModified=false;
     newCourseListPanel = new CourseListGUI();
     frame.setContentPane(newCourseListPanel);
```

```
        newCourseInfoPanel = new CourseInfoGUI();
        frame.setContentPane(newCourseInfoPanel);

        newSchedulePanel = new CourseSchedulingGUI();
        frame.setContentPane(newSchedulePanel);

 //Display the window.
       frame.setLocation(150,50);
       frame.setSize(530, 440);
       frame.setVisible(true);
    }

    public static void main(String[] args)
    {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable(){
         public void run()
         {createAndShowGUI();}}
       );
    }
}
class MyFilter extends javax.swing.filechooser.FileFilter
{
       public boolean accept(File file)
       {
           String filename = file.getName();
           return filename.endsWith(".xml");
       }
       public String getDescription()
       {
           return "*.xml";
       }
   }
```

4 Class CourseScheduleGUI

```
/* File Name: CourseScheduleGUI.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: CourseScheduleGUI
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

 /* This class can display different course schedules after user chooses menu item
Schedule Display from Display.User can also change course placement among courses
in this interface.*/
import java.awt.*;
import java.awt.event.*;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JPopupMenu;
import javax.swing.JMenuItem;
import javax.swing.JTable;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;


public class CourseSchedulingGUI extends JPanel implements ActionListener
{
    private CourseList courseData;
    private PopupListener popupListener;
    private DefaultTableModel[] model=new DefaultTableModel[4];
    private JTable table[]=new JTable[4];
    private JComboBox cbTableTitle;
    private JLabel conflictListLabel;
    private JScrollPane schedulePanel[]=new JScrollPane[4];
    private String colNames[][]={
                    {"Mon(E1A)","Tue(E3A)","Wed(E5A)","Thu(E2B)","Fri(E4B)",
"January", "Evening","Unclear"},//autumn morning
                    {"Mon(E2A)","Tue(E4A)","Wed(E5B)","Thu(E1B)","Fri(E3B)",
"January", "Evening","Unclear"},//autumn afternoon

                    {"Mon(F1A)","Tue(F3A)","Wed(F5A)","Thu(F2B)","Fri(F4B)",
"June", "Evening","Unclear"},//spring morning
                    {"Mon(F2A)","Tue(F4A)","Wed(F5B)","Thu(F1B)","Fri(F3B)",
"June", "Evening","Unclear"}};//spring afternoon

// Set conflict courses
    private String conflictList[]={    "02222,02232,02233",
                                       "02222,02240",
                                       "02240,02232,02233",
                                       "02222,02224",
                                       "02233,02236"};

   private int spaceLine=4;
    //Constructor
    CourseSchedulingGUI()
    {
        for (int i=0;i<4;i++)
```

```
          model[i]=new DefaultTableModel();
 // Get data of course scheduling table

    courseData=CourseTool.getCourseData();

     TableColumn col;
// Set the coresponding column width of tables
        int columnWidth[]={70,70,70,70,70,70,70,70};
/*  Set  scheduling  tables.   table0:autumn  morning   table1:autumn  afternoon
table2:spring morning table3:spring afternoon*/
    for(int i=0;i<4;i++)
    {
            for (int j=0;j<8;j++)
            model[i].addColumn(colNames[i][j]);
            table[i] = new JTable(model[i]);
            for(int vColIndex = 0; vColIndex<8; vColIndex++)
            {
                col = table[i].getColumnModel().getColumn(vColIndex);
                col.setPreferredWidth(columnWidth[vColIndex]);
table[i].getColumnModel().getColumn(vColIndex).setHeaderValue(colNames[i][vColIn
dex]);}
            addSchedulingTableItem(i,i);
            table[i].getTableHeader().setReorderingAllowed(false);
            schedulePanel[i] = new JScrollPane(table[i]);
    }

    setTableShow(true);

    String tableTitle[]={"Autumn Schedule","Spring Schedule","Year Schedule",};
     cbTableTitle = new JComboBox(tableTitle);
     cbTableTitle.setSelectedIndex(2);
     cbTableTitle.setActionCommand("TableTitle");
     cbTableTitle.addActionListener(this);
     conflictListLabel=new JLabel("");

     this.setLayout(new GridBagLayout());
     GridBagConstraints c = new GridBagConstraints();
     c.fill = GridBagConstraints.HORIZONTAL;

        c.gridx = 0;
        c.gridy = 0;
        add(cbTableTitle, c);
        c.gridx = 1;
        c.gridy = 0;
        add(conflictListLabel, c);
        c.gridwidth = 2;
        for(int i=0;i<4;i++)
        {
            c.gridx = 0;
            c.gridy = i+1;
            add(schedulePanel[i], c);
        }

//Set up the popup memo
        JMenuItem menuItem;
        String
subMenuItem[]={"Monday","Tuesday","Wednesday","Thirsday","Friday"};
        String
menuItemExtra[][]={{"E1","E2","F1","F2"},{"E3","E4","F3","F4"},{"E5","F5"}};

       JPopupMenu popup = new JPopupMenu();
       for (int i=0;i<5;i++)
       {
```

```
        JMenu submenu = new JMenu(subMenuItem[i]);

            for (int j=0;j<4;j++)
            {
                menuItem = new JMenuItem(colNames[j][i]);
                menuItem.addActionListener(this);
                submenu.add(menuItem);
          }

          for (int k=0;k<menuItemExtra[i%3].length;k++)
          {
                menuItem = new JMenuItem(menuItemExtra[i%3][k]);
                menuItem.addActionListener(this);
                submenu.add(menuItem);
          }
          popup.add(submenu);
       }

        menuItem = new JMenuItem("January");
        menuItem.addActionListener(this);
        popup.add(menuItem);

        menuItem = new JMenuItem("June");
        menuItem.addActionListener(this);
        popup.add(menuItem);

        menuItem = new JMenuItem("Evening");
        menuItem.addActionListener(this);
        popup.add(menuItem);

        menuItem = new JMenuItem("Unclear");
        menuItem.addActionListener(this);
        popup.add(menuItem);

//Add listener to popup menu
       popupListener = new PopupListener(popup);

       for(int i=0;i<4;i++)
          table[i].addMouseListener(popupListener);
    }
/*TableNumber specifes the table number(from 0 to 3) shown in the interface;
    autumn morning:tableType=0;autumn afternoon:tableType=1;
    spring morning:tableType=2;spring afternoon:tableType=3;*/

public void addSchedulingTableItem(int tableNumber, int tableType)
   {
       String[][] tableData=null;;
        try
        {
          tableData = getTableData(tableType);
        } catch (Exception e)
        {
           e.printStackTrace();
        }
        int i=0;
        while (hasRow(tableData,i))
        {
        model[tableNumber].addRow(new Object[]{tableData[0][i],tableData[1][i],
        tableData[2][i],   tableData[3][i],   tableData[4][i],   tableData[5][i],
tableData[6][i],tableData[7][i]});
           i++;
        }
```

```
        for(int j=i;j<spaceLine;j++)
                model[tableNumber].addRow(new Object[]{"","","","","","","",""});
    }


// Check if the row of table is empty
    public boolean hasRow(String[][] tableData, int index)
    {
         boolean notEmpty=false;
         for(int i=0;i<8;i++)
            if (tableData[i][index]!=null){
              notEmpty=true;
              break;
             }
        return notEmpty;
    }

// Get the  schedule table information corresponding to the table type
    public  String[][] getTableData(int tableID) throws Exception
    {
          Course mycourse;
          String str;
          int colum[]={0,0,0,0,0,0,0,0};
          String content[][]=new String[8][100];
          int coursePosition;
          boolean conflict[]=new boolean[150];
          boolean addItem=false;
          conflict=checkConflict();
          for(int i=0;i<courseData.getCoursesNumber();i++)
          {
              mycourse = (Course)(courseData.getCourses().get(i));
              addItem=false;
              coursePosition=getCoursePosition(tableID,
mycourse.getPlacement(),mycourse.getFrequency());

              if(coursePosition==-1)
              {
                 if(tableID==0)
                    addItem=true;
              }
              else{
                 if (coursePosition<7)
                 {
                    addItem=true;
                  }
              }
              if (addItem)
              {
                if(coursePosition==-1)
                    coursePosition=7;
                if(conflict[i])

    content[coursePosition][colum[coursePosition]]=mycourse.getCourseNumber()+"(
*)";
                else

    content[coursePosition][colum[coursePosition]]=mycourse.getCourseNumber();

                colum[coursePosition]++;
              }

           }
           return content;
```

```
    }
//Check if courses are conflict
    public boolean[] checkConflict()
    {
          boolean conflict[]=new boolean[150];
          Course mycourse1;
          Course mycourse2;
            for(int i=0;i<courseData.getCoursesNumber();i++)
            {
                conflict[i]=false;
                mycourse1=(Course)(courseData.getCourses().get(i));

                for(int j=0;j<i;j++)
                {
                    mycourse2=(Course)(courseData.getCourses().get(j));
                    if (isConflict(mycourse1,mycourse2))
                    {
                        conflict[i]=true;
                        conflict[j]=true;
                    }
                }

            }

         return conflict;
    }
//Check if two courses are conflict
    public boolean isConflict(Course mycourse1, Course mycourse2)
    {
        boolean isconflict=false;
        String courseNumber1;
        String courseNumber2;
        String pl1;
        String pl2;
        pl1=mycourse1.getPlacement();
        pl2=mycourse2.getPlacement();
        if ((pl1.indexOf(pl2)!=-1)||(pl2.indexOf(pl1)!=-1))

          if((pl1.indexOf(" ")==-1)&&(pl2.indexOf(" ")==-1))
           if((pl1.indexOf("Unclear")==-1)&&(pl2.indexOf("Unclear")==-1))
          {
              courseNumber1=mycourse1.getCourseNumber();
              courseNumber2=mycourse2.getCourseNumber();
               for(int j=0;j<conflictList.length;j++)
               {
                if((conflictList[j].indexOf(courseNumber1)!=-1)
                        &&(conflictList[j].indexOf(courseNumber2)!=-1))
                    isconflict=true;
               }
          }
        return isconflict;
    }
//Get the corresponding conflict courses for one course
    public String getConflictList(String courseNumber)
    {
        String courseList="";
        for(int i=0;i<conflictList.length;i++)
        {
        if(conflictList[i].indexOf(courseNumber)!=-1)
        {
            String []tmpCourses=conflictList[i].split(",");
            for(int j=0;j<tmpCourses.length;j++)
```

```
if((courseList.indexOf(tmpCourses[j])==-1)&&(tmpCourses[j].indexOf(courseNumber)
==-1))
        courseList=courseList+tmpCourses[j]+" ";
        }

     }
        if(courseList=="")
        return "No Conflict Couses";
        else
        return courseList;
    }
    //Updata course scheduling table
    public void updateTable()
    {
        courseData=CourseTool.getCourseData();
        clearTableItem();
        for(int i=0;i<4;i++)
        {
            addSchedulingTableItem(i, i);
        }

   }
   /*Set which course scheduling will be displayed:Autumn Schedule, Spring
Schedule,Year Schedulue*/
    public void setTableShow(boolean isShow)
    {
        String colNames0[]={"","","","","", "", "",""};
        int tableHeight[]={146,146,0,0};
        if (isShow)
        {
            for (int i=0;i<4;i++)
                  tableHeight[i]=64;
            schedulePanel[2].setVisible(true);
            schedulePanel[3].setVisible(true);
        }
        else
        {
            schedulePanel[2].setVisible(false);
            schedulePanel[3].setVisible(false);
        }
        for (int i=0;i<4;i++)
         table[i].setPreferredScrollableViewportSize(new          Dimension(500,
         tableHeight[i]));

  }

//Update scheduling table information
    public void updateTableSeason(int seasonKey)
    {
        clearTableItem();
        int tableID=0;
        if (seasonKey==1)// Spring Schedule
            tableID=2;
       if (seasonKey==2)// Year Schedule
        {
            spaceLine=4;
            setTableShow(true);
            CourseTool.getFrame().show();
        }
        else{
            spaceLine=9;
            setTableShow(false);
            CourseTool.getFrame().show();
```

```
        }
        for(int i=0;i<2;i++)
        {
            addSchedulingTableItem(i, tableID+i);
            for(int j=0;j<8;j++)

table[i].getColumnModel().getColumn(j).setHeaderValue(colNames[tableID+i][j]);

        }
        for(int i=2;i<4;i++)
        {
            addSchedulingTableItem(i, i);
        }

 }
//Clear cheduling table information
    public void clearTableItem()
    {
        int rowCount;
        int index;
        for(int i=0;i<4;i++)
        {
            rowCount=model[i].getRowCount();
            for(index=0;index<rowCount;index++)
            {
                model[i].removeRow(0);
            }
        }

    }
//Get course placement for one course
    public  int  getCoursePosition(int  tableID,  String  courseTime,  String
courseItmeLimit)
    {
      int i;
String   coursePositionList[][]={{"E1A",    "E3A",    "E5A",    "E2B",    "E4B",
"January","Evening"},{"E2A",  "E4A",  "E5B",  "E1B",  "E3B",  "",  ""},{"F1A",  "F3A",
"F5A",  "F2B",  "F4B",  "June",  ""},{"F2A",  "F4A",  "F5B",  "F1B",  "F3B",  "",  ""}};
      for(i=0;i<7;i++)
      {
          if(coursePositionList[tableID][i].startsWith(courseTime))
          break;
       }
      boolean checkUnclear=true;
      for(int j=0;j<4;j++)
        for(int k=0;k<7;k++)
            if(coursePositionList[j][k].startsWith(courseTime))
            {
                checkUnclear=false;
                break;
            }
      if (checkUnclear)
        return -1;
      else
        return i;
    }

   public void actionPerformed(ActionEvent e)
   {
      if (e.getActionCommand().equals("TableTitle"))
      {
       updateTableSeason(cbTableTitle.getSelectedIndex());
```

```
       }
       else
       {
        JMenuItem source = (JMenuItem)(e.getSource());
          JTable currentTable=popupListener.getTable();
          String
courseNumber=(String)currentTable.getValueAt(currentTable.getSelectedRow(),curre
ntTable.getSelectedColumn());
          updateCoursePlacement(courseNumber, source.getText());

       }
    }
  //Update course placement
    public void updateCoursePlacement(String courseNumber, String selectItem)
    {
        String newPlacement;
        Course mycourse=null;
        if (courseNumber.indexOf("(*)")!=-1)
            courseNumber=courseNumber.substring(0,courseNumber.length()-3);
        if (selectItem.indexOf("(")!=-1)
             newPlacement=selectItem.substring(4,selectItem.length()-1);
        else
                newPlacement=selectItem;

        for(int i=0;i<courseData.getCoursesNumber();i++)
        {
            mycourse = (Course)(courseData.getCourses().get(i));
            if (courseNumber.equals(mycourse.getCourseNumber()))
              break;
        }
        CourseDataHandler.setCourseItemValue(mycourse,4,newPlacement);
        updateTable();

    }

class PopupListener extends MouseAdapter
    {
       JPopupMenu popup;
       JTable mytable=null;
       PopupListener(JPopupMenu popupMenu)
        {
          popup = popupMenu;
       }

       public void mousePressed(MouseEvent e)
       {
          maybeShowPopup(e);
       }

       public void mouseReleased(MouseEvent e)
       {
          maybeShowPopup(e);
       }
       public JTable getTable()
       {
          return mytable;
       }
       private void maybeShowPopup(MouseEvent e)
       {
        if (e.getClickCount()>0) {
                mytable=(JTable)e.getSource();
                if (mytable.getSelectedRow()!=-1){
```

```
String
courseNumber=(String)mytable.getValueAt(mytable.getSelectedRow(),mytable.getSele
ctedColumn());
                        if (courseNumber!=null)
                            if (!(courseNumber.equals("")))
                            {
                              conflictListLabel.setText("     Conflict   courses:
"+getConflictList(courseNumber));
                          popup.show(e.getComponent(),e.getX(), e.getY());
                            }
                }

          }

       }
    }
}
```

5 Class CourseListGUI

```
/* File Name: CourseListGUI.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: CourseListGUI
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

/* This class displays all the courses in the database after user chooses menu item
Course Display from Display. User can search courses, add new courses, view course
detail, edit courses or delete courses in this interface. */
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CourseListGUI extends JPanel implements ActionListener
{
   private JTable courseListTable;
    private DefaultTableModel courseListModel = new DefaultTableModel();
    private CourseList  courseData;
    private JScrollPane tableListPanel;
    private JScrollPane tableDetailPanel;
    private JButton butView;
    private JButton butNew;
    private JButton butEdit;
    private JButton butDelete;
    private JButton butSearch;
    private JComboBox itemNameList;
    private JTextField searchTextField;
public String[] itemstr={  "CourseNumber",      "EnglishName",      "DanishName",
"ShortName","Placement",     "Frequency",     "ResponsiblePerson",     "Teacher",
"ETCS-points","Censoring",               "ExamType",                "Curricula",
"Prerequisites","ConflictCourses", "Remark", "NumberOfStudents"};

//Constructor
   public CourseListGUI()
   {
        //------------------------ courseListTable---------------
        int i;
// Columns of course dispaly table
        String      colListTableNames[]={"Id",       "Number",       "ShortName",
"Placement","Teacher","ETCS", "Curricula", "Prerequisites","NumberOfStudents"};
//Width of course dispaly table
        int columnListTableWidth[]={10, 30,30,30,30,20,30,30,20};
        courseListTable = new JTable(courseListModel);
        TableColumn col;
```

```
        //add column
        for (i=0;i<9;i++)
            courseListModel.addColumn(colListTableNames[i]);

//Add column width
        for (i=0;i<9;i++)
        {
            col = courseListTable.getColumnModel().getColumn(i);
            col.setPreferredWidth(columnListTableWidth[i]);
        }
// Add table data
        courseData=CourseTool.getCourseData();
        addListTableItem();
    courseListTable.setPreferredScrollableViewportSize(new Dimension(500, 290));
        courseListTable.getTableHeader().setReorderingAllowed(false);

        searchTextField = new JTextField("");

        itemNameList = new JComboBox(itemstr);
        itemNameList.setSelectedIndex(0);
        itemNameList.setActionCommand("searchItem");
        itemNameList.addActionListener(this);
        //add Search Course button
        butSearch= new JButton("Search Course");
        butSearch.setActionCommand("CourseSearch");
        butSearch.addActionListener(this);

// Add four bottom buttons
        butView= new JButton("Course Detail");
        butView.setActionCommand("CourseDetail");
        butView.addActionListener(this);

        butNew= new JButton("New Course");
        butNew.setActionCommand("CourseNew");
        butNew.addActionListener(this);

        butEdit= new JButton("Edit Course");
        butEdit.setActionCommand("CourseEdit");
        butEdit.addActionListener(this);

        butDelete= new JButton("Delete Course");
        butDelete.setActionCommand("CourseDelete");
        butDelete.addActionListener(this);

        tableListPanel = new JScrollPane(courseListTable);

        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;


//  Add the scroll pane to this panel.
        //-------Line 1 -----------
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 2;
        add(searchTextField, c);

        c.gridx = 2;
        c.gridy = 0;
        c.gridwidth = 1;
        add(itemNameList, c);
```

```
        c.gridx = 3;
        c.gridy = 0;
        add(butSearch, c);

        //-------Line 2 -----------
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 4;
        add(tableListPanel, c);
        //-------Line 3 -----------
        c.gridx = 0;
        c.gridy = 2;
        c.gridwidth = 1;
        add(butNew, c);

        c.gridx = 1;
        c.gridy = 2;
        add(butView, c);

        c.gridx = 2;
        c.gridy = 2;
        add(butEdit, c);

        c.gridx = 3;
        c.gridy = 2;
        add(butDelete, c);
    }
    // New course:status0, course detail:status1, edit course:status2
     public void actionPerformed(ActionEvent e)
     {
        int rowIndex,rowIndexTmp;
        CourseInfoGUI myPanel=CourseTool.newCourseInfoPanel;
        JFrame myFrame=CourseTool.getFrame();
        if (("CourseDetail".equals(e.getActionCommand()))
            ||("CourseEdit".equals(e.getActionCommand())))
        {
            rowIndexTmp = courseListTable.getSelectedRow();
            rowIndex = getIndex(rowIndexTmp)-1;
            if (rowIndex!=-1)
                myPanel.setCurrentRowIndex(rowIndex);
            else
                myPanel.setCurrentRowIndex(0);

            if ("CourseEdit".equals(e.getActionCommand())){
                myPanel.setStatus(2);
            }
            else{
                myPanel.setStatus(1);
            }

            myPanel.updateTable();
            myFrame.setContentPane(myPanel);
            myFrame.show();
        }
        if ("CourseNew".equals(e.getActionCommand()))
        {
            myPanel.setStatus(0);
            myPanel.updateTable();
            myFrame.setContentPane(myPanel);
            myFrame.show();
        }
```

```
      if ("CourseDelete".equals(e.getActionCommand()))
        {

           String dlgInfo="Are you sure to delete?";
            if(CourseTool.showInfoDialog(dlgInfo))
            {
                CourseListGUI myListPanel=CourseTool.newCourseListPanel;
                CourseSchedulingGUI mySchedulePanel=CourseTool.newSchedulePanel;
                rowIndex = courseListTable.getSelectedRow();
                CourseTool.deleteCourse(rowIndex);
                myListPanel.updateTable();
                mySchedulePanel.updateTable();
                CourseTool.isModified=true;
                myFrame.show();
            }
        }

        if ("CourseSearch".equals(e.getActionCommand()))
        {

            String searchKey=searchTextField.getText();
            int searchItemIndex=itemNameList.getSelectedIndex();
            CourseTool.newCourseListPanel.searchCourse(searchKey,
searchItemIndex);
        }

    }


public int getIndex(int tableIndex)
    {
        String indexStr;
        if (tableIndex==-1)
            tableIndex=0;
        indexStr= (String)courseListModel.getValueAt(tableIndex,0);
        return Integer.parseInt(indexStr);
    }
// Get the result of Search Course button
    public void searchCourse(String searchKey, int itemName)
    {
        Course mycourse;
        String courseItemValue;
        clearTableItem();
        for(int i=0;i<courseData.getCoursesNumber();i++)
        {
            mycourse = (Course)(courseData.getCourses().get(i));
    courseItemValue=CourseDataHandler.getCourseItemValue(mycourse, itemName);
            String courseItemValue1=courseItemValue.toUpperCase();
            String searchKey1=searchKey.toUpperCase();
            if (courseItemValue1.indexOf(searchKey1)!=-1)
                addRow(i+1, mycourse);
        }
    }

// Empty the table
    public void clearTableItem()
    {
        int rowCount=courseListModel.getRowCount();
        int index;

        for(index=0;index<rowCount;index++)
        {
                courseListModel.removeRow(0);
```

```
        }
    }

// Get the total number of courses
    public int getRowNumber()
    {
       return courseListModel.getRowCount();
    }

 // Add one course  as a row to course list table
    public void addRow(int id, Course mycourse)
    {
        courseListModel.addRow(new Object[]{  Integer.toString(id),
                mycourse.getCourseNumber(),
                mycourse.getShortName(),
                mycourse.getPlacement(),
                mycourse.getTeacher(),
                mycourse.getEts(),
                mycourse.getCurricula(),
                mycourse.getPrerequisites(),
                mycourse.getNumberOfStudents()});
   }

// Add all the courses to the course list table
    public void addListTableItem()
    {
         Course mycourse;
         for(int i=0;i<courseData.getCoursesNumber();i++)
        {
            mycourse = (Course)(courseData.getCourses().get(i));
            addRow(i+1, mycourse);
        }
    }

//Update the course list table information

    public void updateTable()
    {
        courseData=CourseTool.getCourseData();
        clearTableItem();
        addListTableItem();

    }

}
```

6 Class CourseInfoGUI

```
/* File Name: CourseInfoGUI.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: CourseInfoGUI
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

/* This class shows the detail information of one selected course. User can edit or
add a new course to the database here.
The next or previous course information for one selected course can also be displayed
to user. */
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CourseInfoGUI extends JPanel implements ActionListener
{

    private JTable courseInfoTable;
    private DefaultTableModel courseInfoModel = new DefaultTableModel();
    private CourseList courseData;
    private JScrollPane tableListPanel;
    private JScrollPane tableInfoPanel;
    private JButton butPrev;
    private JButton butNext;
    private JButton butBack;
    private JButton butSave;
    private int rowIndex;
    private int rowNumber;
    private int status;  // 0: for new   1 for detail  2 for edit
    public String[] itemstr={  "CourseNumber",    "EnglishName",    "DanishName",
"ShortName","Placement",     "Frequency",     "ResponsiblePerson",     "Teacher",
"ETCS-points","Censoring",     "ExamType",     "Curricula",     "Prerequisites",
"ConflictCourses", "Remark", "NumberOfStudents"};
//Constructor
   public CourseInfoGUI()
   {
        int i;
        status=1;
        rowIndex=0;
        //the number of all the courses
        rowNumber=CourseTool.newCourseListPanel.getRowNumber();

        TableColumn col;

        courseData=CourseTool.getCourseData();
//------------------------ courseInfoTable---------------
        String colListTableNames[]={"ID", "FieldName", "FieldValue"};
```

```
        boolean isEnabled[]={false, false, true};
        int columnListTableWidth[]={30,100,350};

        courseInfoTable = new JTable(courseInfoModel);
//Set up column names
        for (i=0;i<3;i++)
        courseInfoModel.addColumn(colListTableNames[i]);
        //set up column width
        for (i=0;i<3;i++)
        {
            col = courseInfoTable.getColumnModel().getColumn(i);
            col.setPreferredWidth(columnListTableWidth[i]);
            col.setCellEditor(new CourseTableCellEditor(isEnabled[i]));
        }

        addInfoTableItem();
    courseInfoTable.setPreferredScrollableViewportSize(new Dimension(500, 320));
        courseInfoTable.getTableHeader().setReorderingAllowed(false);

//Add bottom buttons
        butPrev= new JButton("Previous Course");
        butPrev.setActionCommand("PrevCourse");
        butPrev.addActionListener(this);

        butNext= new JButton("Next Course");
        butNext.setActionCommand("NextCourse");
        butNext.addActionListener(this);

        butSave= new JButton("Save Course");
        butSave.setActionCommand("CourseSave");
        butSave.addActionListener(this);

        butBack= new JButton("Back To Course List");
        butBack.setActionCommand("Back");
        butBack.addActionListener(this);

        tableInfoPanel = new JScrollPane(courseInfoTable);

        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

//Add the scroll panel to this panel.
//-------Line 1 -----------
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 4;
        add(tableInfoPanel, c);

        //-------Line 2 -----------
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        add(butPrev, c);

        c.gridx = 1;
        c.gridy = 1;
        add(butNext, c);

        c.gridx = 2;
        c.gridy = 1;
        add(butSave, c);
```

```
        c.gridx = 3;
        c.gridy = 1;
        add(butBack, c);

}
    public void actionPerformed(ActionEvent e)
    {
        int rowIndex;
        if ("Back".equals(e.getActionCommand()))
        {
            CourseTool.getFrame().setContentPane(CourseTool.newCourseListPanel);
            CourseTool.getFrame().show();
        }
        if ("PrevCourse".equals(e.getActionCommand())) {

                showPrevCourse();
        }
        if ("NextCourse".equals(e.getActionCommand())) {

                showNextCourse();
        }
        if ("CourseSave".equals(e.getActionCommand()))
        {
            String dlgInfo="Are you sure to save?";
            if(CourseTool.showInfoDialog(dlgInfo))
            {
                if(status==0)
                addCourseData();
               if(status==2)
               saveCourseData();
            }
        }

    }
//Clear table information
    public void clearInfoTableItem()
    {
        int rowCount=courseInfoModel.getRowCount();
        int index;

        for(index=0;index<rowCount;index++)
        {
            courseInfoModel.removeRow(0);
        }

    }

//Update table information
    public void updateInfoTableItem()
    {
       clearInfoTableItem();
        addInfoTableItem();
    }
    //set the selected row index
    public void setCurrentRowIndex(int rowIndex)
    {
       this.rowIndex=rowIndex;
    }
    public void updateTable()
    {
        courseData=CourseTool.getCourseData();
        updateInfoTableItem();
```

```
    }
// Show next course
    public void showNextCourse()
    {
        setCurrentRowIndex((rowIndex+1)%rowNumber);
         updateInfoTableItem();
    }

// Show previous course
    public void showPrevCourse()
    {
         setCurrentRowIndex((rowIndex+rowNumber-1)%rowNumber);
         updateInfoTableItem();
    }

    public void setStatus(int status)
    {
            this.status=status;
            switch(status)
            {
              case 0:  // add a new
                   courseInfoTable.setEnabled(true);
                   butPrev.setEnabled(false);
                   butNext.setEnabled(false);
                   butSave.setEnabled(true);
                    break;
              case 1:  // show detailcourse information
                    courseInfoTable.setEnabled(false);
                    butPrev.setEnabled(true);
                    butNext.setEnabled(true);
                    butSave.setEnabled(false);
                    break;
              case 2:  // edit course
                   courseInfoTable.setEnabled(true);
                    butPrev.setEnabled(true);
                    butNext.setEnabled(true);
                    butSave.setEnabled(true);
                    break;

            }
//Add detail information to table
    public void addInfoTableItem()
    {
        Course mycourse;
        mycourse = (Course)(courseData.getCourses().get(rowIndex));
        String courseItemArray[]=mycourse.getItemArray();
        for (int i=0;i<16;i++)
        {
            if (status!=0)
                courseInfoModel.addRow(new         Object[]{String.valueOf(i+1),
itemstr[i], courseItemArray[i]});
            else
                courseInfoModel.addRow(new         Object[]{String.valueOf(i+1),
itemstr[i], ""});

        }

    }
//Save changed information to the current interface
    public void saveCourseData()
    {
        CourseListGUI myListPanel=CourseTool.newCourseListPanel;
        CourseSchedulingGUI mySchedulePanel=CourseTool.newSchedulePanel;
```

```
        Course mycourse;
        mycourse = (Course)(courseData.getCourses().get(rowIndex));
        for(int i=0;i<16;i++)
CourseDataHandler.setCourseItemValue(mycourse,i,(String)courseInfoModel.getValue
At(i,2));
        CourseTool.setCourseData(rowIndex,mycourse);
        myListPanel.updateTable();
        mySchedulePanel.updateTable();
        CourseTool.isModified=true;

    }
//Add new added course information
    public void addCourseData()
    {
        CourseListGUI myListPanel=CourseTool.newCourseListPanel;
        CourseSchedulingGUI mySchedulePanel=CourseTool.newSchedulePanel;
        Course mycourse;
        mycourse = new Course();
        for(int i=0;i<16;i++)
CourseDataHandler.setCourseItemValue(mycourse,i,(String)courseInfoModel.getValue
At(i,2));
        CourseTool.addCourseData(mycourse);
        myListPanel.updateTable();
        mySchedulePanel.updateTable();
        CourseTool.isModified=true;


    }

}
```

7 Class CourseDataHandler

```
/* File Name: CourseDataHandler.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: Course Data Loader and Saver
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */
// This class mainly gets all the course information  from  a .xml file
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Calendar;
import java.util.GregorianCalendar;
import org.xml.sax.*;
import javax.xml.parsers.*;

public class CourseDataHandler
{
    public  CourseList courselist;
    private String courseInfoFile;
    private String courseInfoFileTemp;
    private String currentFilename;
    //constructor
    CourseDataHandler()
    {
        currentFilename="courseinfo_2004.xml";
    }

// Get all the courses as a list
   public CourseList getCourseList()
   {
      return courselist;
   }
  public void setCourseList(CourseList courselist)
  {
      this.courselist=courselist;
   }
   public void setFilename(String filename)
   {
          this.currentFilename=filename;
    }
//Load data from database and return courses as a list
   public void loadData() throws Exception
   {
    if (!currentFilename.equals(""))
        {
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser saxParser = factory.newSAXParser();
            XMLReader parser = saxParser.getXMLReader();
            SAXModelBuilder mb = new SAXModelBuilder();
            parser.setContentHandler(mb);
            parser.parse(currentFilename);
            courselist = (CourseList)mb.getModel();
        }
        else{
            courselist = new CourseList();
        }
```

```
  }
 //Save data to database as XML format
  public void saveData() throws Exception
  {
   String[] itemstr={  "CourseNumber", "EnglishName", "DanishName", "ShortName",
                       "Placement", "Frequency", "Rperson", "Teacher", "Ets",
                       "Censoring", "ExamType", "Curricula", "Prerequisites",
                       "ConflictCourses", "Remark", "NumberOfStudents"};

       Course mycourse;
       try {
               BufferedWriter      out      =      new      BufferedWriter(new
FileWriter(currentFilename));
               Calendar cal = new GregorianCalendar();

               out.write("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n");
                    out.write("<!-- Lastupdate: "+cal.getTime()+" -->\n");
                    out.write("<CourseList>\n");

            for(int i=0;i<courselist.getCoursesNumber();i++)
                  {   mycourse = (Course)(courselist.getCourses().get(i));
                      out.write("   <Course class=\"normal\">\n");
                      for(int j=0;j<16;j++)
                      {
                        out.write("
<"+itemstr[j]+">"+getCourseItemValue(mycourse, j)+"</"+itemstr[j]+">\n");

                      }
                        out.write("   </Course>\n");

                    }

                    out.write("</CourseList>");
                    out.close();

       }
       catch (IOException e)
       {
                 System.err.println("File not found " + e);
       }

    }

// Get the corresponding attribute value of class Course
    public static String getCourseItemValue(Course mycourse, int index)
    {
        switch(index)
        {
        case 0:
            return mycourse.getCourseNumber();
        case 1:
            return mycourse.getEnglishName();
        case 2:
            return mycourse.getDanishName();
        case 3:
            return mycourse.getShortName();
        case 4:
            return mycourse.getPlacement();
        case 5:
            return mycourse.getFrequency();
        case 6:
            return mycourse.getRperson();
```

```
         case 7:
             return mycourse.getTeacher();
         case 8:
             return mycourse.getEts();
         case 9:
             return mycourse.getCensoring();
         case 10:
             return mycourse.getExamType();
         case 11:
             return mycourse.getCurricula();
         case 12:
             return mycourse.getPrerequisites();
         case 13:
             return mycourse.getConflictCourses();
         case 14:
             return mycourse.getRemark();
         case 15:
             return mycourse.getNumberOfStudents();
         }
         return "";

    }

// Set the corresponding attribute value of class Course
     public static void setCourseItemValue(Course mycourse, int index, String value)
     {
         switch(index)
         {
         case 0:
             mycourse.setCourseNumber(value);
             break;
         case 1:
             mycourse.setEnglishName(value);
             break;
         case 2:
             mycourse.setDanishName(value);
             break;
         case 3:
             mycourse.setShortName(value);
             break;
         case 4:
             mycourse.setPlacement(value);
             break;
         case 5:
             mycourse.setFrequency(value);
             break;
         case 6:
             mycourse.setRperson(value);
             break;
         case 7:
             mycourse.setTeacher(value);
             break;
         case 8:
             mycourse.setEts(value);
             break;
         case 9:
             mycourse.setCensoring(value);
             break;
         case 10:
             mycourse.setExamType(value);
             break;
         case 11:
```

```
            mycourse.setCurricula(value);
            break;
        case 12:
            mycourse.setPrerequisites(value);
            break;
        case 13:
            mycourse.setConflictCourses(value);
            break;
        case 14:
            mycourse.setRemark(value);
            break;
        case 15:
            mycourse.setNumberOfStudents(value);
            break;
        }


    }

}
```

8 Class SAXModelBuilder

```
/* File Name: SAXModelBuilder.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: XML Parser
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.*;
import java.lang.reflect.*;
/* This class receives SAX events from parsing an XML file and constructs classes
corresponding to the names of the tags.*/

public class SAXModelBuilder extends DefaultHandler
{
   Stack stack = new Stack();
   SimpleElement element;
//This method is called when an opening tag in XML document is encountered.
   public void startElement(String namespace, String localname, String qname,
Attributes atts)
     throws SAXException
    {
      SimpleElement element = null;
      try {
           element = (SimpleElement)Class.forName(qname).newInstance();
          }
      catch ( Exception e ) {}

      if ( element == null )
        element = new SimpleElement();

      for(int i=0; i<atts.getLength(); i++)
        element.setAttributeValue(atts.getQName(i), atts.getValue(i));
        stack.push(element);
   }
 //This method is called when a tag is closed.
  public void endElement( String namespace, String localname, String qname)
    throws SAXException
  {
    element = (SimpleElement)stack.pop();
    if (!stack.empty())
       try {
          setProperty( qname, stack.peek(), element );
       }
       catch ( Exception e ) {
       throw new SAXException("Error: "+e );
       }
  }
 //This method can be invoked repeatedly to supply more text as it is read.
  public void characters(char[] ch, int start, int len )
  {
    String text = new String( ch, start, len );
    ((SimpleElement)(stack.peek())).addText(text);
  }
```

```
/*This method uses reflection and the standard JavaBeans naming conventions to look
for the appropriate property "setter" method to apply a value to its parent object.
*/
    void setProperty( String name, Object target, Object value )
      throws SAXException
   {
      Method method = null;
      try {
           method = target.getClass().getMethod(
           "add"+name, new Class[] {value.getClass()});
      }
      catch ( NoSuchMethodException e ){
          }
      if ( method == null )
         try {
               method = target.getClass().getMethod(
               "set"+name, new Class[] { value.getClass() } );
          }
          catch ( NoSuchMethodException e ) {
          }
      if ( method == null )
         try {
              value = ((SimpleElement)value).getText();
              method = target.getClass().getMethod(
              "add"+name, new Class[]{String.class});
          }
          catch (NoSuchMethodException e) {
          }
      try {
         if ( method == null )
           method = target.getClass().getMethod(
              "set"+name, new Class[]{String.class});
         method.invoke( target, new Object [] { value } );
      }
      catch (Exception e){
          throw new SAXException(e.toString());
      }
   }
   public SimpleElement getModel(){return element;}
}
```

9 Class SimpleElement

```
/* File Name: SimpleElement.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: XML Simple Element class
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

// This is the base class of class Course and class CourseList. This class handles
text content for any element. */

public class SimpleElement
{

   StringBuffer text = new StringBuffer();
 //This method appends the parameter string to an object of class StringBuffer
   public void addText(String s)
   {
     text.append(s);
   }
 //This method transfers the result to a string
   public String getText()
   {
     return text.toString();
   }
 //This method is an abstract class and it is not implemented in this class
   public void setAttributeValue(String name, String value)
   {
      throw new Error(getClass()+": No attributes allowed");
   }
}
```

10 Class Course

```
/* File Name: Course.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: Course Data Structure
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

//This class simply sets and gets the content of one course. It is a superclass of
class SimpleElement */

public class Course extends SimpleElement {
int courseClass;
String coursenumber, englishname, danishname, shortname,placement,
frequency, rperson, teacher, ets,censoring, examtype, curricula,
prerequisites,conflictcourses, remark, numberofstudents;

public String[] getItemArray()
{ String itemArray[]={coursenumber, englishname, danishname, shortname,
                          placement, frequency, rperson, teacher, ets,
                          censoring, examtype, curricula, prerequisites,
                          conflictcourses, remark, numberofstudents};
         return itemArray;
    }
public void setCourseNumber(String coursenumber)
{ this.coursenumber = coursenumber ; }

public String getCourseNumber()
{ return coursenumber; }

public void setEnglishName(String englishname)
{ this.englishname = englishname ; }

public String getEnglishName()
{ return englishname; }

public void setDanishName(String danishname)
{ this.danishname = danishname ; }

public String getDanishName()
{ return danishname; }

public void setShortName(String shortname)
{ this.shortname = shortname ; }

public String getShortName()
{ return shortname; }

public void setPlacement(String placement)
{ this.placement = placement ; }

public String getPlacement()
{ return placement; }

public void setFrequency(String frequency)
{ this.frequency = frequency ; }
```

```
public String getFrequency()
 { return frequency; }

public void setRperson(String rperson)
{ this.rperson = rperson ; }

public String getRperson()
{ return rperson; }

public void setTeacher(String teacher)
{ this.teacher = teacher ; }

public String getTeacher()
{ return teacher; }

public void setEts(String ets)
{ this.ets = ets ; }

public String getEts()
{ return ets; }

public void setCensoring(String censoring)
{ this.censoring = censoring ; }

public String getCensoring()
{ return censoring; }

public void setExamType(String examtype)
{ this.examtype = examtype ; }

public String getExamType()
{ return examtype; }

public void setCurricula(String curricula)
{ this.curricula = curricula ; }

public String getCurricula()
{ return curricula; }

public void setPrerequisites(String prerequisites)
{ this.prerequisites = prerequisites ; }

public String getPrerequisites()
{ return prerequisites; }

public void setConflictCourses(String conflictcourses)
{ this.conflictcourses = conflictcourses ; }

public String getConflictCourses()
{ return conflictcourses; }

public void setRemark(String remark)
{ this.remark = remark ; }

public String getRemark()
{ return remark; }

public void setNumberOfStudents(String numberofstudents)
{ this.numberofstudents = numberofstudents ; }
```

```
public String getNumberOfStudents()
{ return numberofstudents; }

public void setcourseClass(int courseClass)
{ this.courseClass = courseClass; }

public int getcourseClass()
{ return courseClass; }

public void setAttributeValue(String name, String value)
{
    if (name.equals("class") && value.equals("normal"))
      setcourseClass(1);
   else
      throw new Error("Invalid attribute: "+name);
}

}
```

11 Class CourseList

```
/* File Name: CourseList.java
 *
 * Last Modification: Dec. 08, 2004
 *
 * Authors:  Hua Wang (s020916@student.dtu.dk)
 *
 * Content:  Course Tool System: CourseList Data Structure
 *
 * Copyright (c) 2004. Informatics and Mathematical Modeling
 *                  Technical University of Denmark
 */

//This class deals with the course list and extends from class SimpleElement.

import java.util.ArrayList;
import java.util.List;

public class CourseList extends SimpleElement
{
List courses = new ArrayList();
int courseNumber;
//constructor
CourseList()
{
    courseNumber=0;
}
// Add a new course
public void addCourse(Course course)
{
        courses.add(course);
       courseNumber++;
}
// Delete a course
public void deleteCourse(int index)
{
        courses.remove(index);
        courseNumber--;
}
// Get all the courses as a list
public List getCourses()
{
         return courses;
}

// Get the total number of courses
public int getCoursesNumber()
{
            return courseNumber;
}
//Set the value of course list
public void setCourses(List courses)
{
        this.courses = courses;
}

}
```