NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

# ONLINE COURSE REGISTRATION

Graduation Project
COM- 400

Student: Ahmad Salman (20021194)

Supervisor: Assist. Prof. Dr Adil Amirjanov

Nicosia - 2006

## *ACKNOWLEDGMENTS*

*First of all I would like to thank my supervisor Assist. Prof. Dr. Adil Amirjanov advisor for intellectual support, encouragement, and enthusiasm, which made this project possible.*

*Second, I thank my dearest family for their constant encouragement and support during the completion of my study*

*Third, my sincerest thanks go to my friend, Salem Al- Marashdah, Mohammad Alsheyab and Alaa Abu- hadbh and all friends who shared their suggestion throughout the completion of my project.*

*And above, I thank God for giving me stamina and courage to achieve my objectives.*

# ABSTRACT

With all the developments and the technology which grows up every year in our university and we are all witness's on these improvements, the idea of this project was decided trying to push up the improvements performance and give some hand to whom it may concern.

In the past, most computer applications ran on stand-alone computers but today's application can be written to communicate with millions of computers, the question comes here, why don't we use these application's power?

The main aim of the project is the development of system for course registration which enables the students to register their courses online so to eliminate the traffic of student registration process on the advisors doors. Other benefits of the system are: allows for distributing instructor's announcement, instructor's office hour's view and edit, displaying each course student list, provides the students with each term offered courses and what courses they are currently taking as well as course grades, GPA and CGPA.

# TABLE OF CONTENTS

# INTRODUCTION

In the past, most computer applications ran on stand-alone computers but today's application (web application) can be written to communicate with millions of computers, the question comes here, why don’t we use these application's power?

The objective of the project is to enable students to register their courses online, so to eliminate student's traffic on the advisors doors while trying to register their courses and to make this process possible where ever the student is. Other benefits of the system are: allows staffs for distributing announcement, office hour's view and edit, displaying and controlling each course student list, provides students with information about the offered courses and what courses they are currently taking as well as courses grades, GPA and CGPA. The project consists of introduction, five chapters and conclusion.

Chapter One introduces the Java language, explains its adjectives and gives some related definitions.

Chapter Two presents java server Pages (JSP) fundamental, has an explanation about JSP files and how JSP works. JSP files technique was used to make dynamic pages which serve both student and staff with the information they need, what is the mechanism of handling JSP requests is explained in this chapter.

Chapter Three introduces JavaScript and explains some of its fundamental concepts.

Chapter Four describes the relational database model and explains how the data base of the project was analyzed and created. The full entity relationship diagram is represented and explained. The structure and description of each table is given.

Chapter Five is devoted to the development of "On Line Course Registration" system. The structure of the system described by the unified modeling language (UML) is given (use case diagrams, class diagrams and sequence diagrams). Some graphical user interfaces (GUI) of the system are represented and illustrated.

# CHAPTER 1: JAVA PROGRAMMING

## 1.1 What is Java

Defining Java is not easy because it is actually many things. According to the Sun white paper on Java, "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." Remove all the adjectives and you are left with the essence of the definition: "Java is a . . . language."

However, Java is more than just another programming language. It is both a language and a platform for developing applications to run on the Internet or in corporate intranet environments.

## 1.2 Java as a Language

Java was designed as a language for object-oriented programming. Unlike procedural programming, where functions are coded to follow step-by-step algorithms, object-oriented programming revolves around data structures known as objects. For all objects that belong to a particular class, there is a well-defined set of routines, or "methods," for operating on the data contained within the object. This defines the interface for working with the object. As long as the interface remains the same, programmers can write the code that implements the methods however they want.

C++ added object-oriented capabilities to the procedural style of C, but it is a difficult language to learn. Its many complicated features and constructs have tended to produce unreliable, buggy applications, even in the hands of fairly competent programmers. On a superficial level, Java is similar to C++ in that the syntax for expressions and control statements is almost the same in the two languages. However, Java is neither a superset nor a subset of C++. Its designers wanted a new language that was easier to learn and use.

Java is more purely object-oriented than C++ in that it requires the use of objects. There are no global variables or functions; all interfaces with an object's data must be defined in the class libraries. Java's creators threw out some of the C++ features that were confusing to developers, such as multiple inheritance (inheriting properties and methods from more than one object) and operator overloading (using one identifier to refer to multiple items in the same scope). To be able to securely execute code that has been downloaded from a network, they eliminated explicit memory pointers so that a Java program cannot access memory addresses it doesn't own. They also added some things C++ doesn't have, such as multithreading (the ability to execute multiple processes concurrently) and garbage collection (automatically recovering previously used memory for reuse by other applications).

## 1.3 Java is Platform Independence

C and C++ code is cross-platform in source form, but has to be specially compiled to form binary files that will run on different processors (see Figure 1.1).

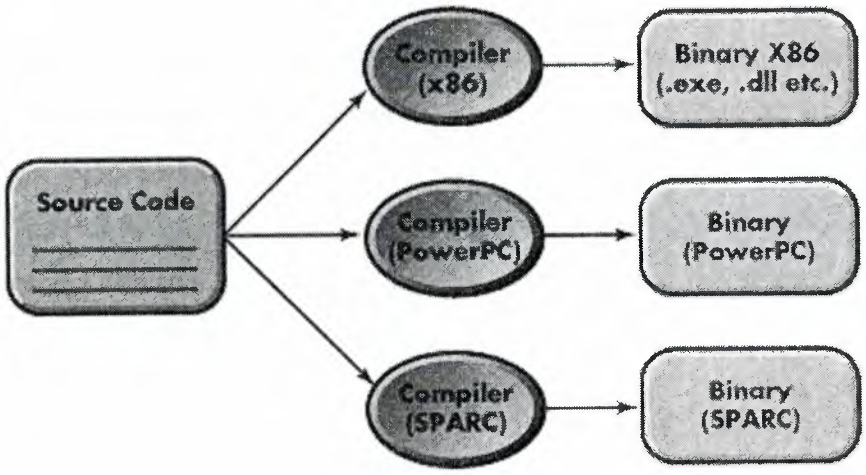


Figure 1.1: Traditional Compilers

Java, on the other hand, is designed to be cross-platform in both source and compiled binary form. Of course, it is impossible to run the same binary file on Windows, Unix, and Macintosh machines. So what Java does is compile its source code into an intermediate, platform-independent byte code. The byte code is then interpreted at execution time by a platform-specific interpreter called a Java Virtual Machine (JVM), as shown in Figure 1.2.

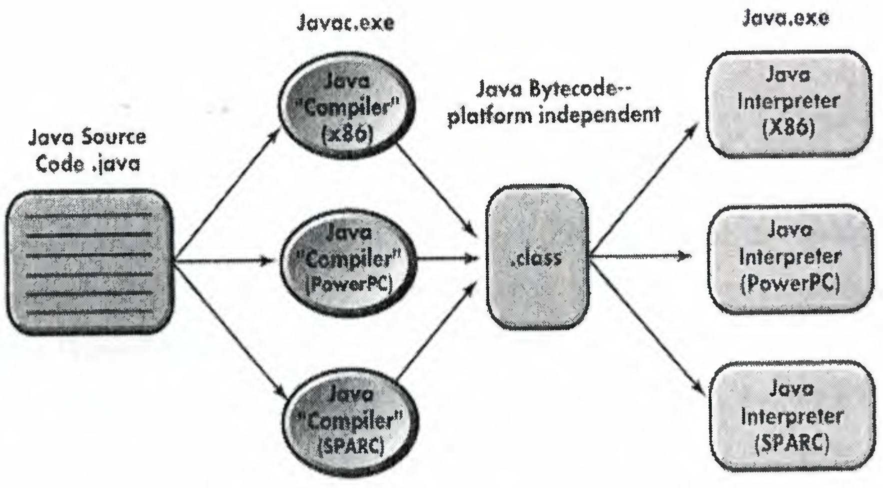


Figure 1.2: Java Compilers and Byte Code.

The interpreted, dynamic nature of the Java language is what allows Java-based applications to be platform-independent. The source and byte code can be written once, and only the JVM needs to be ported to each platform. Currently supported platforms include Windows 95 and Windows NT, Windows 3.x, OS/2, Macintosh, Unix, and the new Network Computer (NC). Thus Java brings software developers closer to the elusive ideal of "write once, run anywhere" code--so much so that JavaSoft has trademarked the phrase!

## 1.4 Java as an Application Development Platform

The Java language can be used to create standalone applications that run outside of a web browser. All a developer needs is the JVM for the desired platform and the appropriate class libraries (.class files) for the functions to be performed. The advantages of using Java are portability and smaller code size, since byte code is usually much smaller than the equivalent native code. However, because Java code is interpreted, it runs far slower than native code.

There are basically two ways to speed up Java applications. One way is for vendors to write their own Java compilers that generate native machine code instead of byte code. While this does bring the speed of Java applications in par with traditional applications, you lose the advantage of portability. The second way to speed up Java is by using "just-in-time" (JIT) compilation. With a JIT compiler, the Java Virtual Machine translates the byte code into native code just before the application is executed. This improves performance over the interpreted method, but since JIT compilers can't do much in the way of code optimization, the resulting code still runs slower than equivalent native code.

## 1.5 Java Applets

Most of the excitement surrounding Java centers around its ability to create small programs called "applets". A Java applet is a Java program designed to be included in a HyperText Markup Language (HTML) document and run inside a web browser. The applet code is embedded in an HTML page via an <APPLET> tag that references the applet's .class file. The applet is then downloaded and executed by an integrated byte code interpreter within the browser (see Figure 1.3).

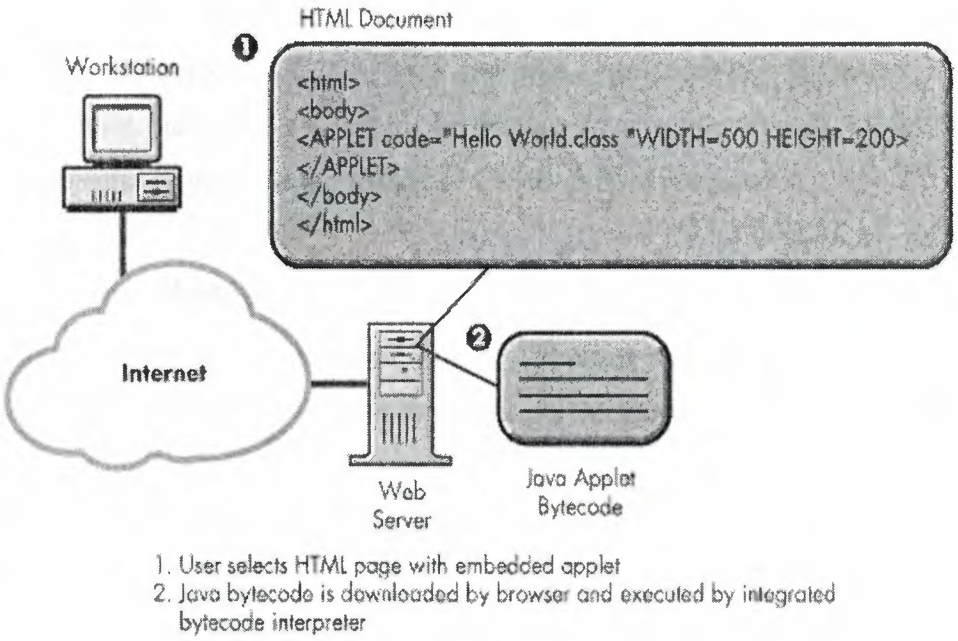

HTML Document
Workstation
1
<html>
<body>
<APPLET code="Hello World.class "WIDTH=500 HEIGHT=200>
</APPLET>
</body>
</html>
Internet
2
Web Server
Java Applet Bytecode
1. User selects HTML page with embedded applet
2. Java bytecode is downloaded by browser and executed by integrated bytecode interpreter

Figure 1.3: Java Applets.

Besides telling the browser what code to retrieve, the <APPLET> tag also defines how large the applet's display area will be through the width and height parameters. The applet controls everything that happens within its own display area. It can create menus, scroll bars, push buttons, and other means of interacting with the user. It can also communicate with other applets running on the same page, or cause the browser to load a new page. But an applet cannot affect any page contents that are specified by HTML tags.

## 1.6 Java Adjectives

### 1.6.1 Java is Simple

Simple to learn in that it has much in common with C++. It is also simple to understand, since many of the difficult features of C++ are absent from Java: Multiple inheritance, automatic casting, hidden constructor and destructor calls, and operator overloading.

Simple can also mean small. Java was developed to run in very little memory, such as embedded controllers.

### 1.6.2 Java is Object Oriented

C++ is a hybrid language. C++ maintains compatibility with C, which is a traditional imperative language. This has been augmented with classes, an implementation of objects. A C++ programmer can operate in either style. Java is much more pure object-oriented, in that no Java application or applet lacks objects. The objects are very similar to that of C++ but more prevalent.

### 1.6.3 Java is Distributed

Java is web and internet-oriented. Java applications and applets can be scattered across different sites on the net. It accesses items across the web as easily as local file items.

Java accepts the client-server model nicely, and has built in support for web protocols like HTTP or FTP.

### 1.6.4 Java is Interpreted

Java is both interpreted and compiled. The javac compiler converts Java source into byte code. Byte code is the machine language for the Java Virtual Machine. Byte code is more compact and much faster to interpret than the Java source code. Byte code can be interpreted on any machine or architecture.

There is nothing about Java that demands interpretation. Native code compilers exist, but there are some advantages to interpretation, as well.

### 1.6.5 Java is Robust

The type checking of Java is at least as strong as that of C++, so that Java programs are well-checked at compile-time. Strong run-time checking by the interpreter catches many other errors. Type checking and interpretation make Java programs crash-proof. The program can fail, but not crash the rest of the system. Many of the errors that hang systems, such as bad subscripts and bad pointers, are specifically detected, preventing a Java program from damaging the rest of the system.

### 1.6.6 Java is Secure

Security is a byproduct of interpretation. Java programs have definite limits of what they can access on the computer. Applets are even more secure than applications, since they cannot even access files on the system they are running on.

### 1.6.7 Java is Architecturally Neutral

Java byte code makes no unusual assumptions about the underlying hardware. Any type of computer should be able to implement a Java Virtual Machine, that is, the byte code interpreter. There are no machine dependencies and no machine variants.

### 1.6.8 Java is Portable

Portability is a consequence of good design and the architecture-neutral approach. Every machine has its own unique byte code interpreter, since that must be coded for each platform. However, they all accept the same byte code. There are standards for the suitable values of the simple types. For example, an int must be a 32 bit 2's complement binary integer on all machines, regardless of how the local machine implements integers. The Java compiler is itself written in Java, as are most of the other tools, so they are exactly the same for all platforms. Thus, Java is portable.

### 1.6.9 Java is High-Performance

Historically, interpreted languages have been much slower than compiled languages. For example, a BASIC interpreter has to parse each line each time it is executed. Java compiles the source into byte code. The overhead of interpretation is the time it takes to look up the byte code and find the right subroutine. On some architectures, that may be very small. The other overhead is that of the run-time checking of the operands, which is not, strictly speaking, an interpretation problem. The byte code keeps the parsing costs low, so that Java code is only somewhat slower than native code. However, that somewhat slower is in the range 1.1 to 10 times slower.

### 1.6.10 Java is MultiThreaded

Java has support for multiple threads or concurrent execution. A thread or a process is a line of execution that executes independently or concurrently with other threads. Most languages have no provision for concurrent execution. Some like C++ have it as an add on, through library function calls. Java has built it in, since it is so needed in the language. The two main areas where threads are useful are the ability to do something while waiting for results shipped over the net, and animations in web applets. There are also a number of synchronization items. These allow two or more threads to communicate reliably.

### 1.6.11 Java is a Dynamic Language

Java links (or binds) things later than C, more like LISP. Most languages like C or Pascal use static binding to attach subroutines to the main program. Static binding occurs at compile time. Java and LISP use dynamic binding. In dynamic binding, the needed routine is not found until it is needed, at run-time. Some languages like C++ use both. This makes the updating of libraries easier to accomplish, without recompilation. Thus, should a new library version occur as soon as it's installed, all Java programs will use it rather than the earlier version.

## 1.7 What Is a Class?

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an instance of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building them, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a class (see figure 1.4).

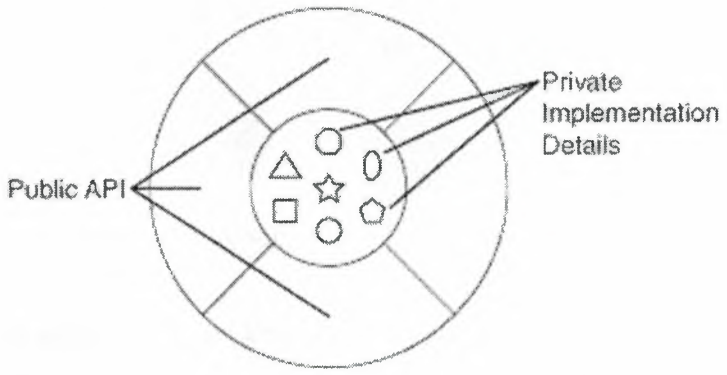


Figure 1.4: A Visual representation of a Class.

## 1.8 What Is an Object?

Objects are key to understanding object-oriented technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

Real-world objects share two characteristics: They all have state and behavior. For example, dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).

Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.

You can represent real-world objects by using software objects. You might want to represent real-world dogs as software objects in an animation program or a real-world bicycle as a software object in the program that controls an electronic exercise bike. You can also use software objects to model abstract concepts. For example, an event is a common object used in window systems to represent the action of as user pressing a mouse button or a key on the keyboard. The following illustration (figure 1.5) is a common visual representation of a software object.

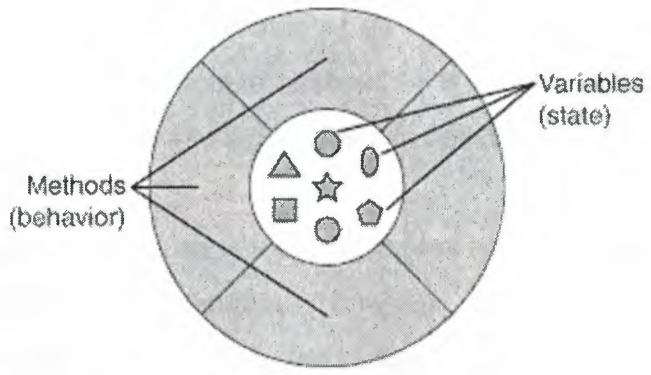


Figure 1.5: A Software Object.

Everything the software object knows (state) and can do (behavior) is expressed by the variables and the methods within that object. A software object that models your real-world bicycle would have variables that indicate the bicycle's current state: Its speed is 18 mph, its pedal cadence is 90 rpm, and its current gear is 5th. These variables are formally known as instance variables because they contain the state for a particular bicycle object; in object-oriented terminology, a particular object is called an instance. The following figure (figure 1.6) illustrates a bicycle modeled as a software object.

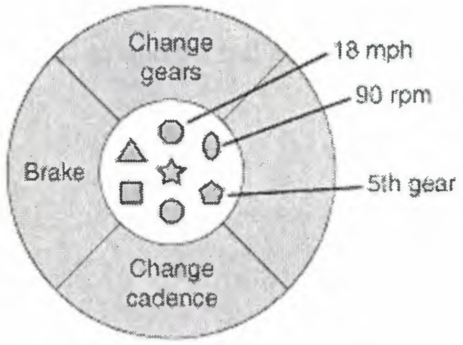


Figure 1.6: A Bicycle Modeled as a Software Object.

In addition to its variables, the software bicycle would also have methods to brake, change the pedal cadence, and change gears. (It would not have a method for changing its speed because the bike's speed is just a side effect of which gear it's in and how fast the rider is pedaling.) These methods are known formally as instance methods because they inspect or change the state of a particular bicycle instance.

Object diagrams show that an object's variables make up the center, or nucleus, of the object. Methods surround and hide the object's nucleus from other objects in the program. Packaging an object's variables within the protective custody of its methods is called encapsulation. This conceptual picture of an object — a nucleus of variables packaged within a protective membrane of methods — is an ideal representation of an object and is the ideal that designers of object-oriented systems strive for. However, it's not the whole story.

Often, for practical reasons, an object may expose some of its variables or hide some of its methods. In the Java programming language, an object can specify one of four access levels for each of its variables and methods. The access level determines which other objects and classes can access that variable or method.

Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

- **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system. You can give your bicycle to someone else, and it will still work.
- **Information-hiding:** An object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting other objects that depend on it. You don't need to understand a bike's gear mechanism to use it.

# CHAPTER 2: JAVA SERVER PAGES (JSP)

## 2.1 What is JSP

Java Server Pages (JSP) provide a facility whereby you can write sever-side scripted pages using the full power of the Java programming language and the rich set of class libraries associated with Java. However there is a price to pay in that JSP, like most things to do with Java, is significantly more complex than other techniques.

Java Server Pages (JSP) is a Sun Microsystems specification for combining Java with HTML to provide dynamic content for Web pages. When you create dynamic content, JSPs are more convenient to write than HTTP servlets because they allow you to embed Java code directly into your HTML pages, in contrast with HTTP servlets, in which you embed HTML inside Java code. JSP is part of the Java 2 Enterprise Edition (J2EE).

JSP enables you to separate the dynamic content of a Web page from its presentation. It caters to two different types of developers: HTML developers, who are responsible for the graphical design of the page, and Java developers, who handle the development of software to create the dynamic content.

Because JSP is part of the J2EE standard, you can deploy JSPs on a variety of platforms, including WebLogic Server. In addition, third-party vendors and application developers can provide JavaBean components and define custom JSP tags that can be referenced from a JSP page to provide dynamic content.

## 2.2 How JSP Works

JSP pages are, at first glance, rather similar to ASP and PHP pages in that special pseudo-tags are used to mark out parts of otherwise normal HTML pages that are to handled specially. The difference is that, whereas ASP and PHP are processed by a server plug-in, JSP is handled by a separate server process. For JSP this process runs software known as Tomcat which takes the JSP page and converts the entire page into a Java programme which is then compiled. The raw HTML in the original page is wrapped in Java output methods. On requesting a JSP page, the request is transferred from the normal WWW server (e.g. Apache) to Tomcat which first checks whether it has cached an already compiled version of the Java programme associated with the page. Recompilation takes place if the last modification date stamp on the JSP page is more recent than the last modification date stamp on any cached compiled version.

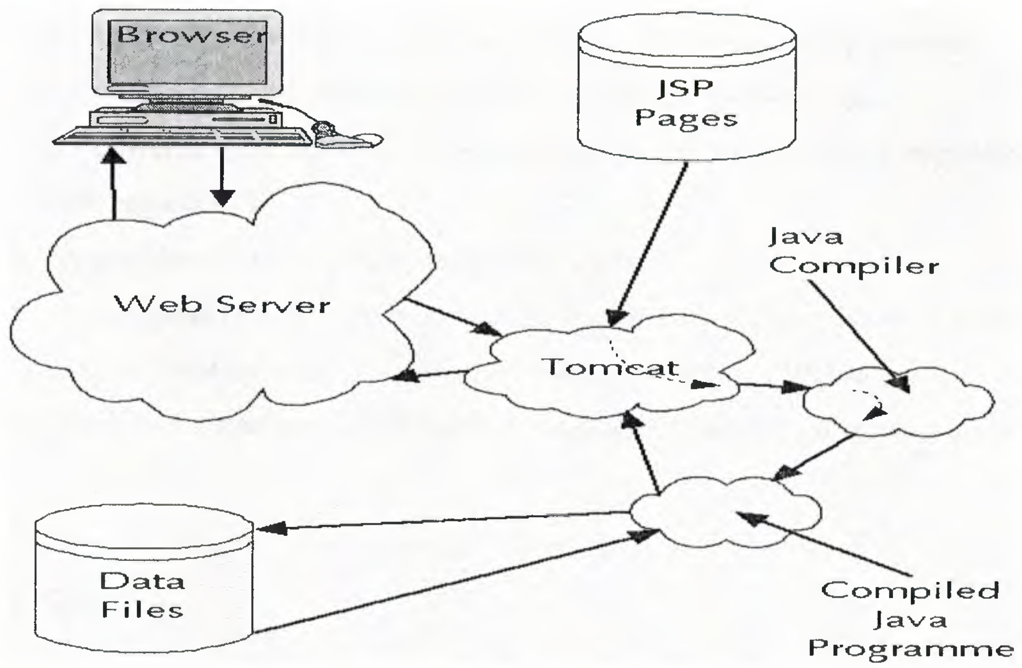


Figure 2.1: How JSP Request Pages are Handled.

## 2.3 How JSP Requests Are Handled

WebLogic Server handles JSP requests in the following sequence:

1. A browser requests a page with a .jsp file extension from WebLogic Server.
2. WebLogic Server reads the request.
3. Using the JSP compiler, WebLogic Server converts the JSP into a servlet class that implements the javax.servlet.jsp.JspPage interface. The JSP file is compiled only when the page is first requested, or when the JSP file has been changed. Otherwise, the previously compiled JSP servlet class is re-used, making subsequent responses much quicker.
4. The generated JspPage servlet class is invoked to handle the browser request.

It is also possible to invoke the JSP compiler directly without making a request from a browser. For details, see Using the WebLogic JSP Compiler. Because the JSP compiler creates a Java servlet as its first step, you can look at the Java files it produces, or even register the generated JspPage servlet class as an HTTP servlet.

## 2.4 Servlets Advantages

There are five main advantages of servlets:

1. It's efficient: the java virtual machine stays up, and each request is handled by lightweight java thread, not a heavyweight operating system process.
2. It's convenient: you are able to use java rather than learn perl too.
3. It's powerful: you can easily do several things that are difficult or impossible with regular CGI.
4. It's portable: follows a well standardized API.
5. It's inexpensive: there are a number of free or very inexpensive web servers available that are good for "personal" use or low-volume Web sites.

❖ Since java server pages (JSP) are servlets, all the benefits of servlets pertain to JSP.

## 2.5 JSP Syntax

### 2.5.1 Directives

JSPs live in an object called a container, which is essentially a server. JSPs can define information for the container with directives.

Here is what directives look like in a general form:

<%@ directive attribute="someValue" attribute="anotherValue" ... %>

There are three directives:

- <%@ page ... %> specifies information that affects the page
- <%@ include ... %> includes a file at the location of the include directive (parsed)
- <%@ taglib ... %> allows the use of custom tags in the page
- <%@ page language="java" %> will always be the first line of every JSP file.

### 2.5.2 Declarations

Declarations are used to specify supplemental methods and variables. You can think of these are the page's private functions; they can only be called by the JSP where they are defined, or by another JSP that includes it (using the <@ include > directive).

Here is a sample declaration:

```
<%!
    // this integer can be used anywhere in this JSP page
 private int myVariable = -1;
// this function can be called from anywhere in this JSP page
public boolean isPositive() {
   return ( myVariable > 0 );
}
%>
```

### 2.5.3 Scriptlets

Scriptlets are bits of Java code. They can do anything (the full power of Java is available in every JSP), but they will most likely concentrate on generating HTML.

### 2.5.4 Expressions

Expressions are special-purpose mini-scriptlets used for evaluating expressions. This could be something as simple as outputting the value of a variable, or a more complicated Java expression, like calling a function and outputting the result.

Table 2.1 JSP Syntax Summary

| JSP Element | Syntax | Interpretation |
|---|---|---|
| JSP Expression | `<%= expression %>` | Expression is evaluated and placed in output. out.println(expression); |
| JSP Scriptlet | `<% code %>` | Java Code is inserted in service method. |
| JSP Declaration | `<%! code %>` | inserted in body of servlet class, outside of service method. |
| JSP page Directive | `<%@ page att="val" %>` | Directions to the servlet engine about general setup. |
| JSP Comment | `<%-- comment --%>` | Comment; ignored when JSP page is translated into servlet. |
| JSP include Directive | `<%@ include file="url" %>` | A file on the local system to be included when the JSP page is translated into a servlet. |

# CHAPER 3: HTML & JAVASCRIPT

## 3.1 Introduction to HTML

(**H**yper**T**ext **M**arkup **L**anguage) The document format used on the Web. Web pages are built with HTML tags (codes) embedded in the text. HTML defines the page layout, fonts and graphic elements as well as the hypertext links to other documents on the Web. Each link contains the URL, or address, of a Web page residing on the same server or any server worldwide, hence "World Wide" Web.

HTML 2.0 was defined by the Internet Engineering Task Force (IETF) with a basic set of features, including interactive forms capability. Subsequent versions added more features such as blinking text, custom backgrounds and tables of contents. However, each new version requires agreement on the tags used, and browsers must be modified to implement those tags.

HTML Itself Is Not a Programming Language; HTML is a markup language (the ML in HTML) that uses a fixed set of markup tags. A markup language can also be thought of as a "presentation language," but it is not a programming language. You cannot "if this-do that" like you can in Java, JavaScript or C++. However, in order to make pages interactive, programming code can be embedded in an HTML page. For example, JavaScript is widely interspersed in Web pages (HTML pages) for that purpose.

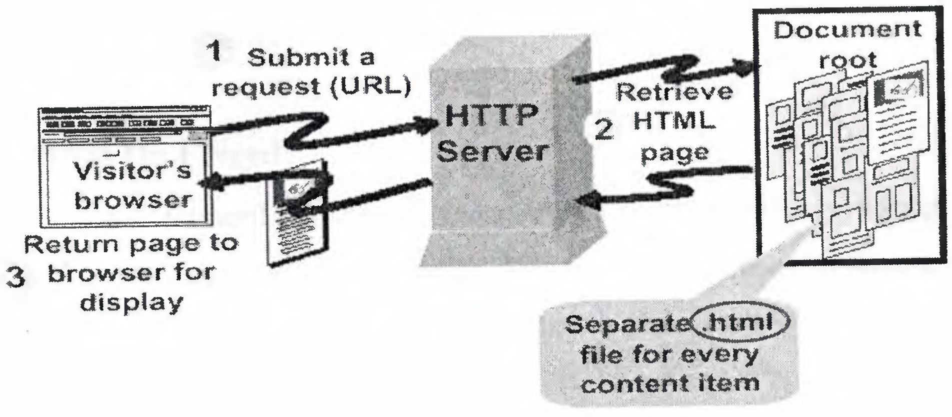


Figure 3.1: Html Pages are transferred by HTTP Protocol.

HTML was conceived as a simple markup language to render research documents. No one envisioned Web pages turning into multimedia extravaganzas.

HTML pages have been reworked, jury-rigged and extended into full-blown applications. As a result, the source code behind today's Web pages is often a hideous concoction of tags and scripting.

## 3.2 Introduction to JavaScript

Thousands of sites around the world use JavaScript but it is still not a particularly well known programming language (compared to HTML). If you have seen anything interactive on a website like a calculation, pop-up-window, some web counters and even some navigation systems then you have probably seen JavaScript. Unfortunately, JavaScript has changed from being a language which improves web sites to a language which destroys them. This is because there are huge JavaScript sites which have thousands of scripts for download. These usually involve things which do not benefit a website at all, like status bar effects and scrolling text which do not add muchtoawebsite.

JavaScript must not be confused with Java. Java is a completely different programming language. It is usually used for text effects and games, although there are some JavaScript games around.

So why should you use JavaScript? Well, JavaScript can allow you to create new things on your website that are both dynamic and interactive, allowing you to do things like find out some information about a user (like monitor resolution and browser), check that forms have been filled in correctly, rotate images, make random text do calculations and many other things.

### 3.2.1 What Do I Need?

You will not need any special hardware or software to write JavaScript, you can just use Notepad or any other text editor. You do not even need to have any special software on your webserver. JavaScript will run on any webserver anywhere! All you will need to do is make sure that you have at least a version 3 browser, as versions of Internet Explorer and Netscape Navigator before this do not support JavaScript, so you will not be able to see your creations.

### 3.2.2 Declaring JavaScript

Adding JavaScript to a web page is actually surprisingly easy! To add a JavaScript all you need to add is the following (either between the <head></head> tags or between the <body></body> tags.

```
<script language="JavaScript">
        JavaScript
</script>
```

As you can see the JavaScript is just contained in a normal HTML tag set. The next thing you must do is make sure that the older browsers ignore it. If you don't do this the code for the JavaScript will be shown which will look awful. There are two things you must do to hide the code from older browsers and show something instead:

```
<script language="JavaScript">
<!—Begin Hide
JavaScript
//End Hide-->
</script>
<noscript>
HTML Code
</noscript>
```

As you can see this makes the code look a lot more complex, but it is really quite simple. If you look closely you can see that all that has been done is that the JavaScript is now contained in an HTML comment tag. This is so that any old browsers which do not understand <script> will just think it is an HTML comment and ignore it. Because of this the <noscript> tag was created. This will be ignored by browsers which understand <script> but will be read by the older browsers. All you need to do is put between <noscript> and </noscript> what you want to appear if the browser does not support JavaScript. This could be an alternative feature or just a message saying it is not available. You do not have to include the tag if you don't want anything shown instead.

# CHAPTER 4: DATABASE DESIGN

## 4.1 Introduction

Many web developers are self-taught, learning HTML, then moving on to a programming language such as JSP. From there, they often learn to integrate this with a database. Too few though have a good theoretical knowledge of databases. Mention foreign keys, or referential integrity, and you're met with a blank stare. Small databases can be easily designed with little database theory knowledge. But large databases can easily get out of hand when badly designed, leading to poor performance, and resulting in the whole database needing to be rebuilt later. This article is a brief introduction to the topic of relational databases, and will hopefully whet your appetite for further exploration.

## 4.2 The Relational Database Model

*A database can be understood as a collection of related files. How those files are* related depends on the model used. Early models included the hierarchical model (where files are related in a parent/child manner, with each child file having at most one parent file), and the network model (where files are related as owners and members, similar to the network model except that each member file can have more than one owner).

The relational database model was a huge step forward, as it allowed files to be related by means of a common field. In order to relate any two files, they simply need to have a common field, which makes the model extremely flexible.

## 4.3 The Entity-Relationship Model

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design for the database designer, the utility of the ER model is:

- It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.

- It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.
- In addition, the model can be used as a design plan by the database developer to implement a data model in a specific database management software.

### 4.3.1 Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities.

#### 4.3.1.1 Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, INVOICES. An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one that does not rely on another for identification. A dependent entity is one that relies on another for identification.

An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

#### 4.3.1.2 Special Entity Types

Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

#### 4.3.1.3 Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

- employees are assigned to projects
- projects have subtasks
- departments manage one or more projects

Relationships are classified in terms of degree, connectivity, cardinality, and existence. These concepts will be discussed below.

### 4.3.1.4 Attributes

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

### 4.3.1.5 Classifying Relationships

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

### 4.3.1.6 Degree of a Relationship

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

### 4.3.1.7 Connectivity and Cardinality

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

### 4.3.1.8 Direction

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

### 4.3.1.9 Type

An identifying relationship is one in which one of the child entities is also a dependent entity. A non-identifying relationship is one in which both entities are independent.

### 4.3.1.10 Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either mandatory or optional. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory.

## 4.3.2 ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is widely used in academics texts and journals but rarely seen in either CASE tools or publications by non-academics. Today, there are a number of notations used; among the more common are Bachman, crow's foot, and IDEFIX.

Here are the notations (crow's foot notation) used in the E-R diagram of the project (see figure 4.1):

- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.
- Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
- Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

## 4.4 Relational Database of Online Course Registration:

After talking about E-R model and having some knowledge about it, now it is the time to implement the shown E-R diagram (figure 4.1) so to obtain the real model(the needed database).

The aimed database is needed for the faculty of computer engineering at our university, information about students, staffs, courses, terms and some process which belongs either to student or to staff are to be stored into the database.

There are 10 tables which were implemented on Ms. Access to cover the need of storing the necessary information.

The Student table contains student information. studentId is the primary key to uniquely identity each student. firstName and surname fields keep the student full name. The table contains advisorId as foreign key to reference the Staff table so to keep track of student advisor's information. password field is used to verify the student while trying to log into his account.

The Staff table contains staffId field as the primary key. name and surName fields save the staff full name. Staff degree is obtained by the foreign key degreeId field which references the staffDegree table. The roomId field is used to get the room number of the staff. password field is added to achieve what the Student table's password field does.

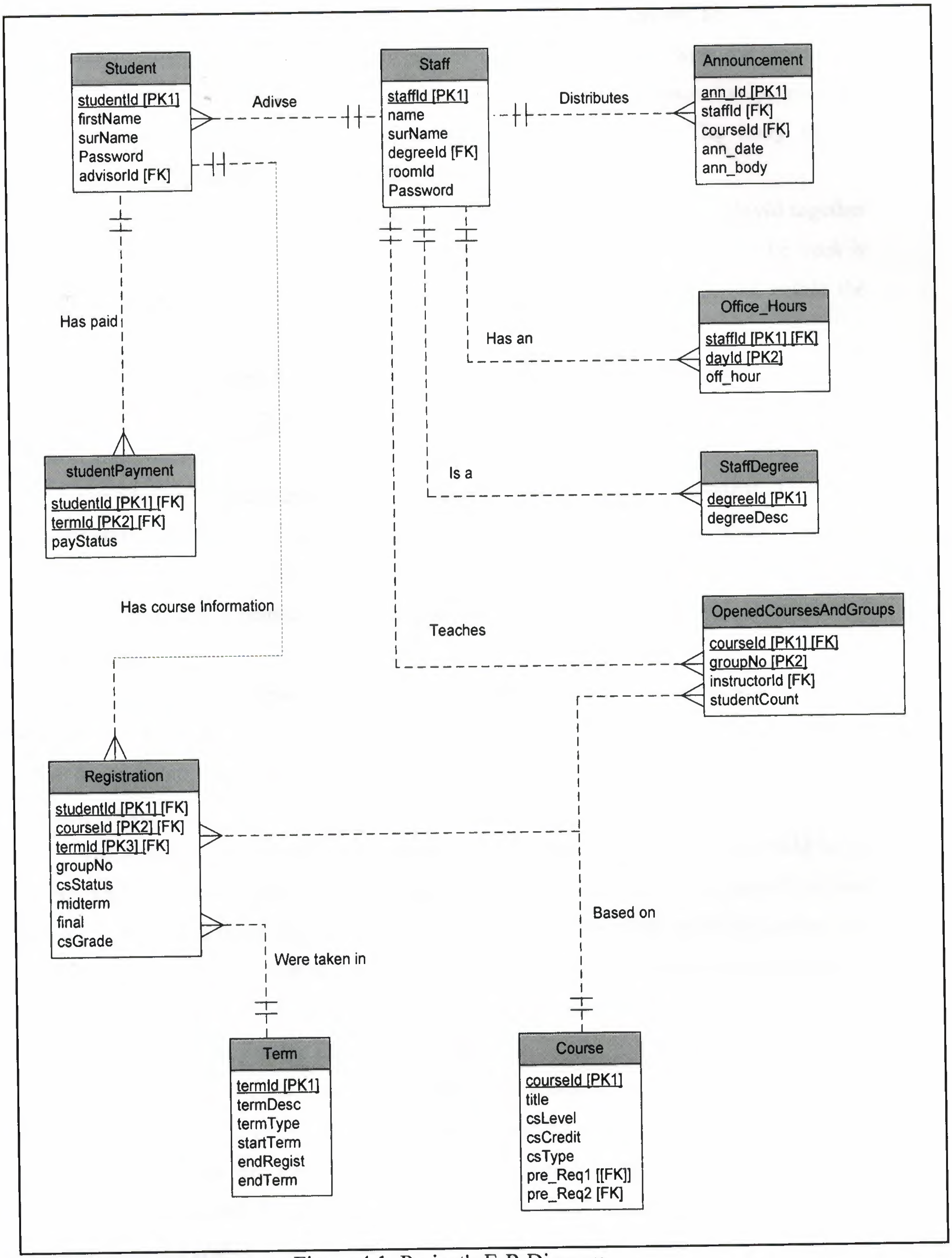


Figure 4.1: Project's E-R Diagram

The Announcement table records are identified by the primary key ann_Id. The foreign keys staffId and courseId fields are to reference the staff who did added the announcement and the course which the announcement does belong respectively. ann_date column marks the date on which the announcement was added. The body of the announcement is kept within the column ann_body.

The Office_Hours table is structured with three columns. staffId and dayId together eliminate the possibility of the duplication (composite primary key). Each day of the week is assigned with an Id (dayId field). Staff office hours for each day are defined within the off_hour field.

The StaffDegree table contains all possible degree classification (Mr., Dr., Prof...etc) with the primary key degreeId. Degree with full string representation is hold within degreeDesc field.

The StudentPayment table keeps a track of student term's fee payment. This table most probably is a view, so it takes the records from the university main database. studentId and termId are composite primary key and on the same time foreign keys that reference Student and Term tables respectively. Term fee payment status (paid, not paid) is determined by the column payStatus.

The Term table (primary key termId) contains term information. termDesc field holds the term title (e.g. Fall 2005/2006). The fields startTerm, endRegist, endTerm are used to define the start date of the term, the dead date for course registration and the end date of the term respectively.

The Course table contains the courses which the university offers. courseId is the primary key. The title of the course is saved in the field title. csLevel field specify at what term level the course is scheduled to be taken. csCredit defines what credit the course does has. csType determines the course type (elective, technical, non-technical). The prerequisites information are kept in the pre_Req1 and pre_Req2 columns which serve as foreign keys and reference the primary key of the same Course Table.

The Registration table contains each student's schedule for every registration term. studentId, courseId and termId are the composite primary key. The table contains two "lookup" tables which are Course and Term. The course status (failed, passed, currently taken) can be obtained from csStatus column. The table provides each taken course's midterm, final and grade results.

The OpenedCoursesAndGroups table contains the offered courses and groups in every term. courseId and groupNo do uniquely identify each record. The table contains a single foreign key "instructorId" column which references the Staff table so to keep a track of course group instructor information.

The following figure shows the entities relationships implemented on Ms.

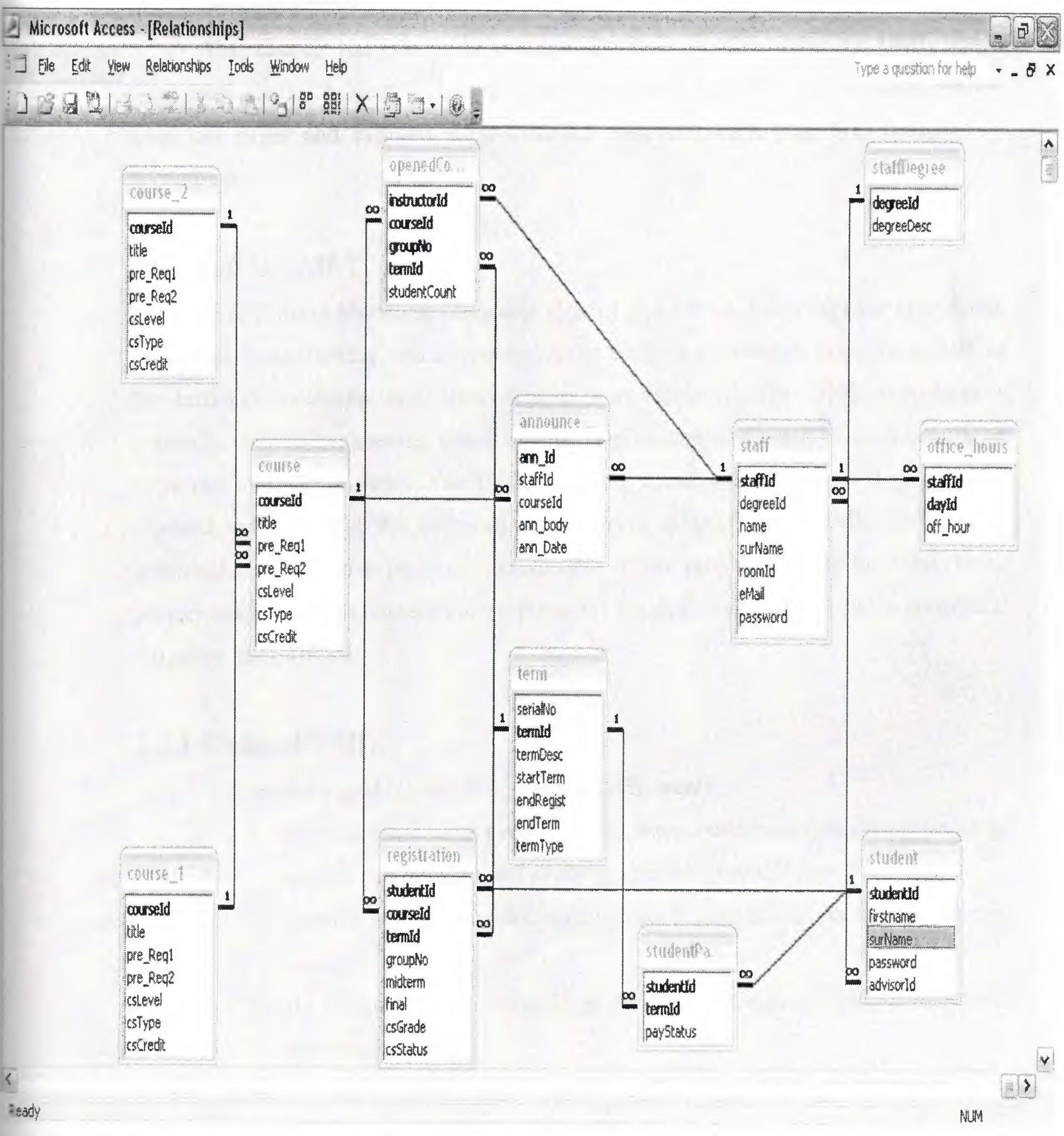


Figure 4.2: The Database of the Project was created on MS Access.

# CHAPTER 5: Online Course Registration Software Design

## 5.1 Introduction

In The previous four chapters were introduced the techniques (Java, JSP, HTML, JavaScript and RDBMS applied on Ms. Database) which were used for implementing the project.

This chapter has two main sections: first section talks about the UML which were used during project designing phase; the second section shows some system graphical pages and explains what technical functions each page was designed to accomplish.

## 5.2 What is UML?

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

### 5.2.1 Goals of UML

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.

6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

### 5.2.2 Use Case

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

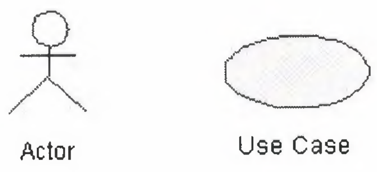


Figure 5.1

### 5.2.3 Use Case Diagrams Depict:

- Use cases. A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.
- Actors. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.
- Associations. Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicating the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. (It is possible to have association between tow use cases)

In the next section it is shown the use cases of the project. (Figure 5.2)

### 5.2.3.1 Project Use Case Diagrams

Since the system (project) belongs to both students and staffs, all possible actions are initiated and/or participated by them, therefore the system has tow actors, student and instructor see figure 5.2.

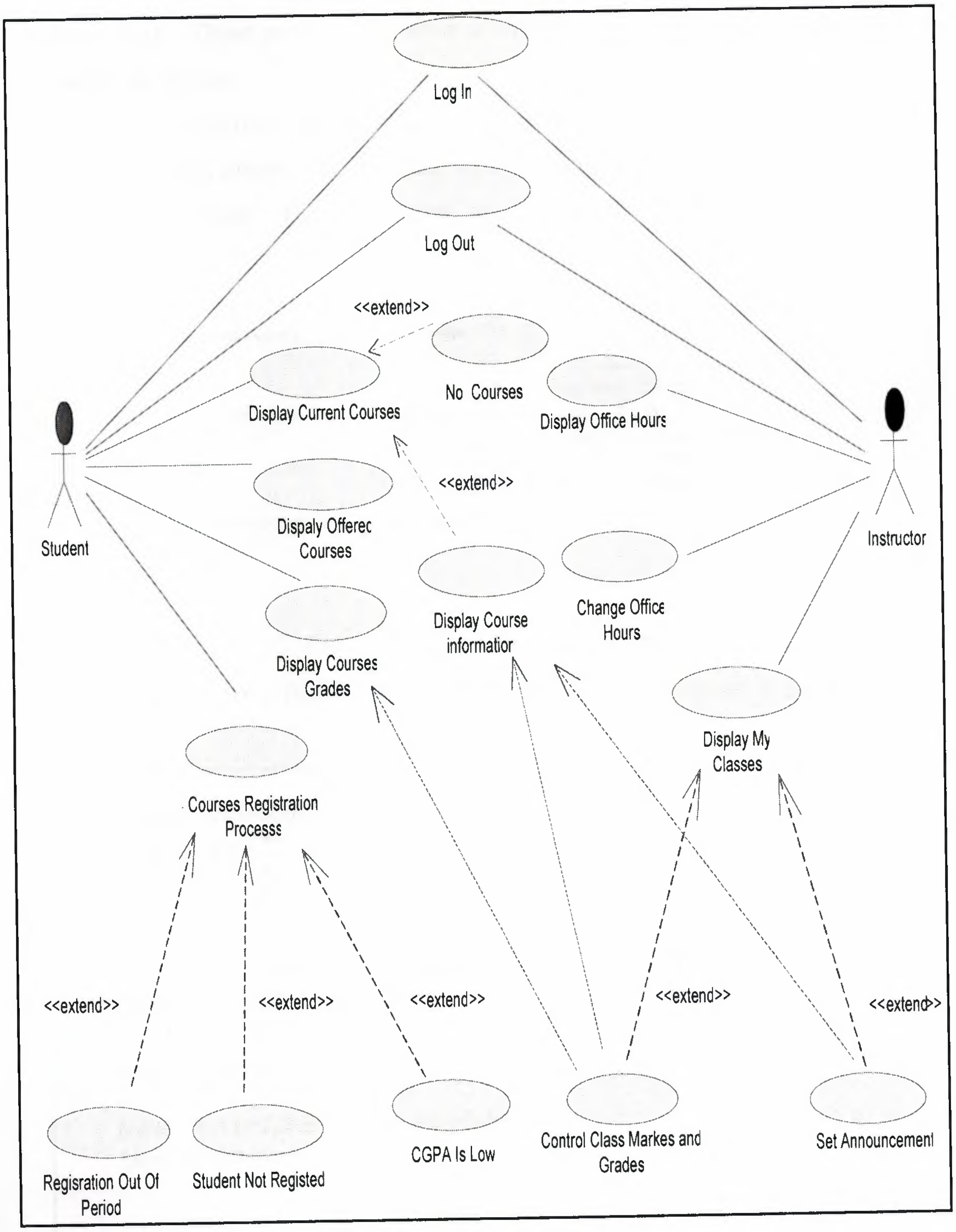


Figure 5.2: Project Use Case Diagrams

### 5.2.4 Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation (figure 5.3). These perspectives become evident as the diagram is created and help solidify the design.

Class diagrams also display relationships such as containment, inheritance, associations and others. The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes.

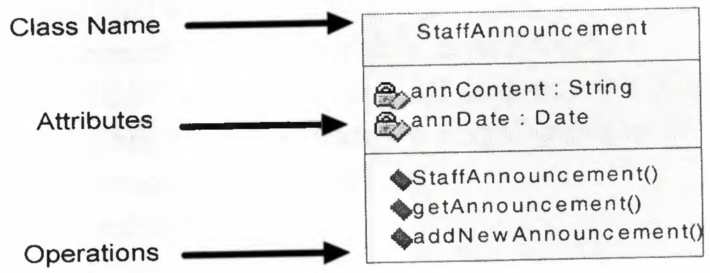
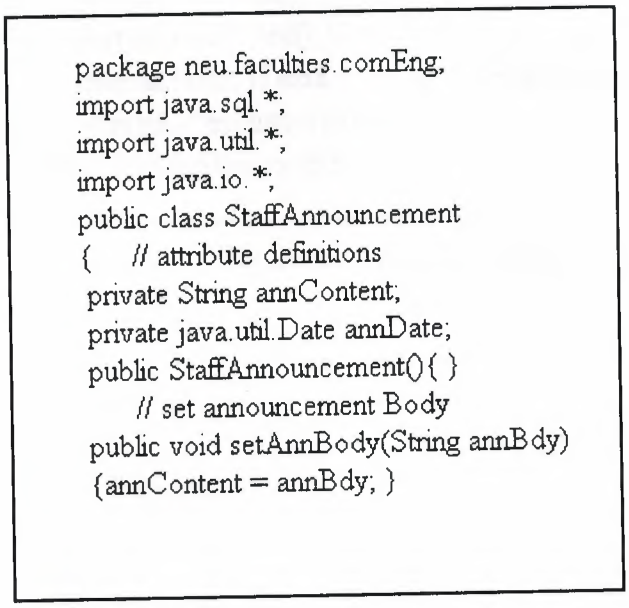


Figure 5.3: Class Diagram Description

The following figure shows the class "StaffAnnouncement" source code.



```
package neu.faculties.comEng;
import java.sql.*;
import java.util.*;
import java.io.*;
public class StaffAnnouncement
{    // attribute definitions
 private String annContent;
 private java.util.Date annDate;
 public StaffAnnouncement() { }
      // set announcement Body
 public void setAnnBody(String annBdy)
 {annContent = annBdy; }
```

```
            // set announcement Date
public void setAnnDate(java.util.Date ann_Date)
 { annDate = ann_Date; }
public String getAnnBody()
 {  return annContent; }
public java.util.Date getAnnDate()
 {    return annDate; }
            // method is to handle the decleration of a new announcement
public boolean addNewAnnouncement(StaffInfo sf, CourseInfo cs, String announcement) throws SQLException
 { String annUpdate = "INSERT INTO announcement(staffId, courseId, ann_body)"
        + " VALUES('" + sf.getstaffId().trim() + "','" + cs.getCourseId().trim() + "','" + announcement.trim() + "')";
   DomainDB.handleUpdate(annUpdate);
   return true; } // end method addNewAnnouncement()
            // the method returns all announcement that belongs to the staff of the specified course
public Vector getAnnouncement(StaffInfo sf, CourseInfo cs ) throws SQLException
  { String annQuery = "SELECT ann_body, ann_Date FROM announcement  WHERE staffId = '" + sf.getstaffId
                         ().trim() + "'"  + " AND courseId = '" + cs.getCourseId().trim() + "'" ;
    ResultSet annSet = DomainDB.handleQuery(annQuery);
    boolean moreAnn = annSet.next();
    if(moreAnn)
    { Vector annVector = new Vector();
  while(moreAnn)
  { StaffAnnouncement stfAnn = new StaffAnnouncement();
    stfAnn.setAnnBody(annSet.getString(1));
    stfAnn.setAnnDate(annSet.getDate(2));
    annVector.add(stfAnn);
    moreAnn = annSet.next(); } // end while
    return annVector; } // end if  // Staff has not declared any announement for the specified course.
    return null } // end method getAnnouncement() }
// END Class StaffAnnouncement
```

Figure 5.4: StaffAnnouncement Class Source Code

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

# ONLINE COURSE REGISTRATION

Graduation Project
COM- 400

Student: Ahmad Salman (20021194)

Supervisor: Assist. Prof. Dr Adil Amirjanov

Nicosia - 2006

# *ACKNOWLEDGMENTS*

*First of all I would like to thank my supervisor Assist. Prof. Dr. Adil Amirjanov advisor for intellectual support, encouragement, and enthusiasm, which made this project possible.*

*Second, I thank my dearest family for their constant encouragement and support during the completion of my study*

*Third, my sincerest thanks go to my friend, Salem Al- Marashdah, Mohammad Alsheyab and Alaa Abu- hadbh and all friends who shared their suggestion throughout the completion of my project.*

*And above, I thank God for giving me stamina and courage to achieve my objectives.*

# ABSTRACT

With all the developments and the technology which grows up every year in our university and we are all witness's on these improvements, the idea of this project was decided trying to push up the improvements performance and give some hand to whom it may concern.

In the past, most computer applications ran on stand-alone computers but today's application can be written to communicate with millions of computers, the question comes here, why don't we use these application's power?

The main aim of the project is the development of system for course registration which enables the students to register their courses online so to eliminate the traffic of student registration process on the advisors doors. Other benefits of the system are: allows for distributing instructor's announcement, instructor's office hour's view and edit, displaying each course student list, provides the students with each term offered courses and what courses they are currently taking as well as course grades, GPA and CGPA.

**TABLE OF CONTENTS**

# INTRODUCTION

In the past, most computer applications ran on stand-alone computers but today's application (web application) can be written to communicate with millions of computers, the question comes here, why don't we use these application's power?

The objective of the project is to enable students to register their courses online, so to eliminate student's traffic on the advisors doors while trying to register their courses and to make this process possible where ever the student is. Other benefits of the system are: allows staffs for distributing announcement, office hour's view and edit, displaying and controlling each course student list, provides students with information about the offered courses and what courses they are currently taking as well as courses grades, GPA and CGPA. The project consists of introduction, five chapters and conclusion.

Chapter One introduces the Java language, explains its adjectives and gives some related definitions.

Chapter Two presents java server Pages (JSP) fundamental, has an explanation about JSP files and how JSP works. JSP files technique was used to make dynamic pages which serve both student and staff with the information they need, what is the mechanism of handling JSP requests is explained in this chapter.

Chapter Three introduces JavaScript and explains some of its fundamental concepts.

Chapter Four describes the relational database model and explains how the data base of the project was analyzed and created. The full entity relationship diagram is represented and explained. The structure and description of each table is given.

Chapter Five is devoted to the development of "On Line Course Registration" system. The structure of the system described by the unified modeling language (UML) is given (use case diagrams, class diagrams and sequence diagrams). Some graphical user interfaces (GUI) of the system are represented and illustrated.

# CHAPTER 1: JAVA PROGRAMMING

## 1.1 What is Java

Defining Java is not easy because it is actually many things. According to the Sun white paper on Java, "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." Remove all the adjectives and you are left with the essence of the definition: "Java is a . . . language."

However, Java is more than just another programming language. It is both a language and a platform for developing applications to run on the Internet or in corporate intranet environments.

## 1.2 Java as a Language

Java was designed as a language for object-oriented programming. Unlike procedural programming, where functions are coded to follow step-by-step algorithms, object-oriented programming revolves around data structures known as objects. For all objects that belong to a particular class, there is a well-defined set of routines, or "methods," for operating on the data contained within the object. This defines the interface for working with the object. As long as the interface remains the same, programmers can write the code that implements the methods however they want.

C++ added object-oriented capabilities to the procedural style of C, but it is a difficult language to learn. Its many complicated features and constructs have tended to produce unreliable, buggy applications, even in the hands of fairly competent programmers. On a superficial level, Java is similar to C++ in that the syntax for expressions and control statements is almost the same in the two languages. However, Java is neither a superset nor a subset of C++. Its designers wanted a new language that was easier to learn and use.

Java is more purely object-oriented than C++ in that it requires the use of objects. There are no global variables or functions; all interfaces with an object's data must be defined in the class libraries. Java's creators threw out some of the C++ features that were confusing to developers, such as multiple inheritance (inheriting properties and methods from more than one object) and operator overloading (using one identifier to refer to multiple items in the same scope). To be able to securely execute code that has been downloaded from a network, they eliminated explicit memory pointers so that a Java program cannot access memory addresses it doesn't own. They also added some things C++ doesn't have, such as multithreading (the ability to execute multiple processes concurrently) and garbage collection (automatically recovering previously used memory for reuse by other applications).

## 1.3 Java is Platform Independence

C and C++ code is cross-platform in source form, but has to be specially compiled to form binary files that will run on different processors (see Figure 1.1).

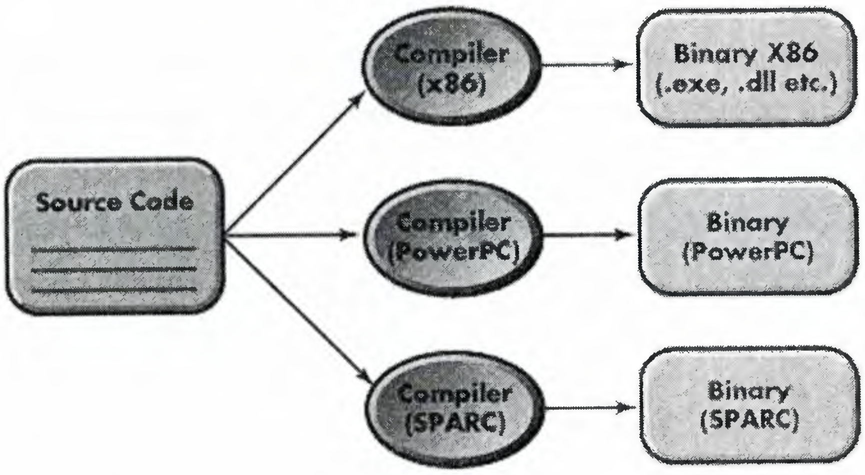


Figure 1.1: Traditional Compilers

Java, on the other hand, is designed to be cross-platform in both source and compiled binary form. Of course, it is impossible to run the same binary file on Windows, Unix, and Macintosh machines. So what Java does is compile its source code into an intermediate, platform-independent byte code. The byte code is then interpreted at execution time by a platform-specific interpreter called a Java Virtual Machine (JVM), as shown in Figure 1.2.

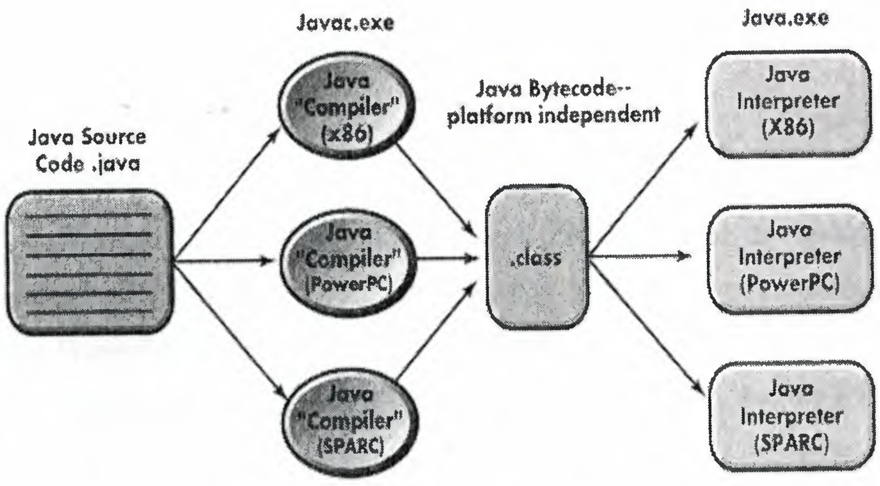


Figure 1.2: Java Compilers and Byte Code.

The interpreted, dynamic nature of the Java language is what allows Java-based applications to be platform-independent. The source and byte code can be written once, and only the JVM needs to be ported to each platform. Currently supported platforms include Windows 95 and Windows NT, Windows 3.x, OS/2, Macintosh, Unix, and the new Network Computer (NC). Thus Java brings software developers closer to the elusive ideal of "write once, run anywhere" code--so much so that JavaSoft has trademarked the phrase!

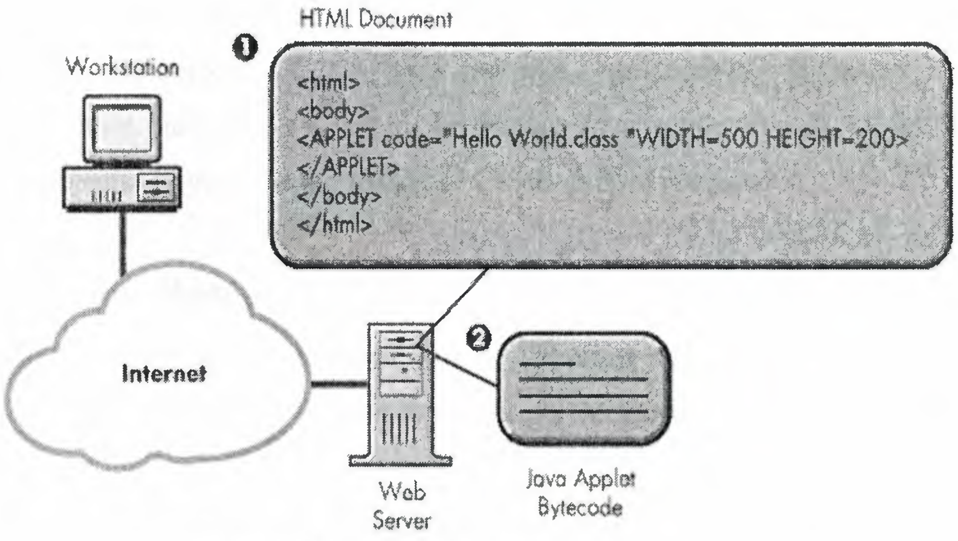
## 1.4 Java as an Application Development Platform

The Java language can be used to create standalone applications that run outside of a web browser. All a developer needs is the JVM for the desired platform and the appropriate class libraries (.class files) for the functions to be performed. The advantages of using Java are portability and smaller code size, since byte code is usually much smaller than the equivalent native code. However, because Java code is interpreted, it runs far slower than native code.

There are basically two ways to speed up Java applications. One way is for vendors to write their own Java compilers that generate native machine code instead of byte code. While this does bring the speed of Java applications in par with traditional applications, you lose the advantage of portability. The second way to speed up Java is by using "just-in-time" (JIT) compilation. With a JIT compiler, the Java Virtual Machine translates the byte code into native code just before the application is executed. This improves performance over the interpreted method, but since JIT compilers can't do much in the way of code optimization, the resulting code still runs slower than equivalent native code.

## 1.5 Java Applets

Most of the excitement surrounding Java centers around its ability to create small programs called "applets". A Java applet is a Java program designed to be included in a HyperText Markup Language (HTML) document and run inside a web browser. The applet code is embedded in an HTML page via an <APPLET> tag that references the applet's .class file. The applet is then downloaded and executed by an integrated byte code interpreter within the browser (see Figure 1.3).



1. User selects HTML page with embedded applet
2. Java bytecode is downloaded by browser and executed by integrated bytecode interpreter

Figure 1.3: Java Applets.

Besides telling the browser what code to retrieve, the <APPLET> tag also defines how large the applet's display area will be through the width and height parameters. The applet controls everything that happens within its own display area. It can create menus, scroll bars, push buttons, and other means of interacting with the user. It can also communicate with other applets running on the same page, or cause the browser to load a new page. But an applet cannot affect any page contents that are specified by HTML tags.

## 1.6 Java Adjectives

### 1.6.1 Java is Simple

Simple to learn in that it has much in common with C++. It is also simple to understand, since many of the difficult features of C++ are absent from Java: Multiple inheritance, automatic casting, hidden constructor and destructor calls, and operator overloading.

Simple can also mean small. Java was developed to run in very little memory, such as embedded controllers.

### 1.6.2 Java is Object Oriented

C++ is a hybrid language. C++ maintains compatibility with C, which is a traditional imperative language. This has been augmented with classes, an implementation of objects. A C++ programmer can operate in either style. Java is much more pure object-oriented, in that no Java application or applet lacks objects. The objects are very similar to that of C++ but more prevalent.

### 1.6.3 Java is Distributed

Java is web and internet-oriented. Java applications and applets can be scattered across different sites on the net. It accesses items across the web as easily as local file items.

Java accepts the client-server model nicely, and has built in support for web protocols like HTTP or FTP.

### 1.6.4 Java is Interpreted

Java is both interpreted and compiled. The javac compiler converts Java source into byte code. Byte code is the machine language for the Java Virtual Machine. Byte code is more compact and much faster to interpret than the Java source code. Byte code can be interpreted on any machine or architecture.

There is nothing about Java that demands interpretation. Native code compilers exist, but there are some advantages to interpretation, as well.

### 1.6.5 Java is Robust

The type checking of Java is at least as strong as that of C++, so that Java programs are well-checked at compile-time. Strong run-time checking by the interpreter catches many other errors. Type checking and interpretation make Java programs crash-proof. The program can fail, but not crash the rest of the system. Many of the errors that hang systems, such as bad subscripts and bad pointers, are specifically detected, preventing a Java program from damaging the rest of the system.

### 1.6.6 Java is Secure

Security is a byproduct of interpretation. Java programs have definite limits of what they can access on the computer. Applets are even more secure than applications, since they cannot even access files on the system they are running on.

### 1.6.7 Java is Architecturally Neutral

Java byte code makes no unusual assumptions about the underlying hardware. Any type of computer should be able to implement a Java Virtual Machine, that is, the byte code interpreter. There are no machine dependencies and no machine variants.

### 1.6.8 Java is Portable

Portability is a consequence of good design and the architecture-neutral approach. Every machine has its own unique byte code interpreter, since that must be coded for each platform. However, they all accept the same byte code. There are standards for the suitable values of the simple types. For example, an int must be a 32 bit 2's complement binary integer on all machines, regardless of how the local machine implements integers. The Java compiler is itself written in Java, as are most of the other tools, so they are exactly the same for all platforms. Thus, Java is portable.

### 1.6.9 Java is High-Performance

Historically, interpreted languages have been much slower than compiled languages. For example, a BASIC interpreter has to parse each line each time it is executed. Java compiles the source into byte code. The overhead of interpretation is the time it takes to look up the byte code and find the right subroutine. On some architectures, that may be very small. The other overhead is that of the run-time checking of the operands, which is not, strictly speaking, an interpretation problem. The byte code keeps the parsing costs low, so that Java code is only somewhat slower than native code. However, that somewhat slower is in the range 1.1 to 10 times slower.

### 1.6.10 Java is MultiThreaded

Java has support for multiple threads or concurrent execution. A thread or a process is a line of execution that executes independently or concurrently with other threads. Most languages have no provision for concurrent execution. Some like C++ have it as an add on, through library function calls. Java has built it in, since it is so needed in the language. The two main areas where threads are useful are the ability to do something while waiting for results shipped over the net, and animations in web applets. There are also a number of synchronization items. These allow two or more threads to communicate reliably.

### 1.6.11 Java is a Dynamic Language

Java links (or binds) things later than C, more like LISP. Most languages like C or Pascal use static binding to attach subroutines to the main program. Static binding occurs at compile time. Java and LISP use dynamic binding. In dynamic binding, the needed routine is not found until it is needed, at run-time. Some languages like C++ use both. This makes the updating of libraries easier to accomplish, without recompilation. Thus, should a new library version occur as soon as it's installed, all Java programs will use it rather than the earlier version.

## 1.7 What Is a Class?

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an instance of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building them, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a class (see figure 1.4).

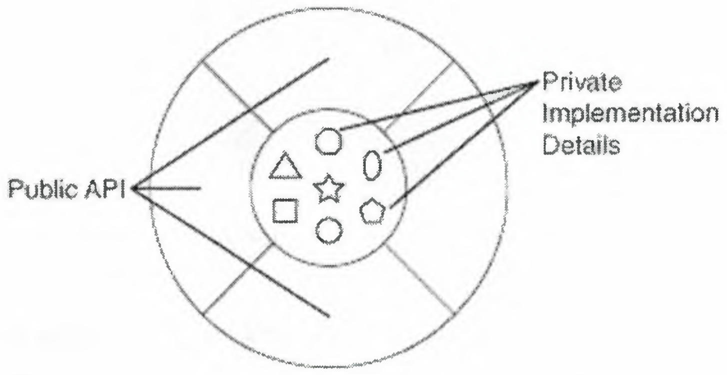


Figure 1.4: A Visual representation of a Class.

## 1.8 What Is an Object?

Objects are key to understanding object-oriented technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

Real-world objects share two characteristics: They all have state and behavior. For example, dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).

Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.

You can represent real-world objects by using software objects. You might want to represent real-world dogs as software objects in an animation program or a real-world bicycle as a software object in the program that controls an electronic exercise bike. You can also use software objects to model abstract concepts. For example, an event is a common object used in window systems to represent the action of as user pressing a mouse button or a key on the keyboard. The following illustration (figure 1.5) is a common visual representation of a software object.

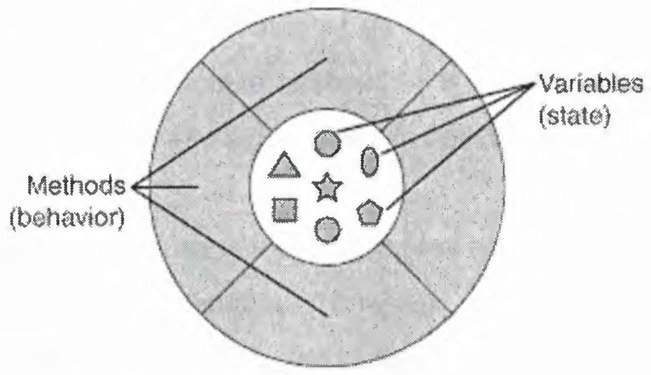


Figure 1.5: A Software Object.

Everything the software object knows (state) and can do (behavior) is expressed by the variables and the methods within that object. A software object that models your real-world bicycle would have variables that indicate the bicycle's current state: Its speed is 18 mph, its pedal cadence is 90 rpm, and its current gear is 5th. These variables are formally known as instance variables because they contain the state for a particular bicycle object; in object-oriented terminology, a particular object is called an instance. The following figure (figure 1.6) illustrates a bicycle modeled as a software object.

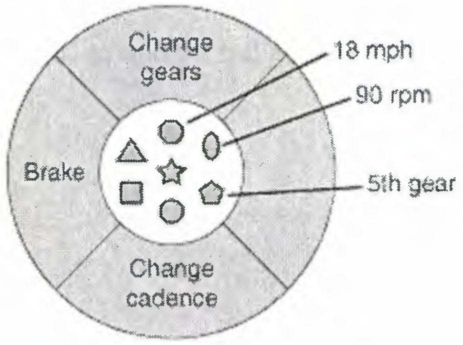


Figure 1.6: A Bicycle Modeled as a Software Object.

In addition to its variables, the software bicycle would also have methods to brake, change the pedal cadence, and change gears. (It would not have a method for changing its speed because the bike's speed is just a side effect of which gear it's in and how fast the rider is pedaling.) These methods are known formally as instance methods because they inspect or change the state of a particular bicycle instance.

Object diagrams show that an object's variables make up the center, or nucleus, of the object. Methods surround and hide the object's nucleus from other objects in the program. Packaging an object's variables within the protective custody of its methods is called encapsulation. This conceptual picture of an object — a nucleus of variables packaged within a protective membrane of methods — is an ideal representation of an object and is the ideal that designers of object-oriented systems strive for. However, it's not the whole story.

Often, for practical reasons, an object may expose some of its variables or hide some of its methods. In the Java programming language, an object can specify one of four access levels for each of its variables and methods. The access level determines which other objects and classes can access that variable or method.

Encapsulating related variables and methods into a neat software bundle is a simple yet powerful idea that provides two primary benefits to software developers:

- **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Also, an object can be easily passed around in the system. You can give your bicycle to someone else, and it will still work.
- **Information-hiding:** An object has a public interface that other objects can use to communicate with it. The object can maintain private information and methods that can be changed at any time without affecting other objects that depend on it. You don't need to understand a bike's gear mechanism to use it.

# CHAPTER 2: JAVA SERVER PAGES (JSP)

## 2.1 What is JSP

Java Server Pages (JSP) provide a facility whereby you can write sever-side scripted pages using the full power of the Java programming language and the rich set of class libraries associated with Java. However there is a price to pay in that JSP, like most things to do with Java, is significantly more complex than other techniques.

Java Server Pages (JSP) is a Sun Microsystems specification for combining Java with HTML to provide dynamic content for Web pages. When you create dynamic content, JSPs are more convenient to write than HTTP servlets because they allow you to embed Java code directly into your HTML pages, in contrast with HTTP servlets, in which you embed HTML inside Java code. JSP is part of the Java 2 Enterprise Edition (J2EE).

JSP enables you to separate the dynamic content of a Web page from its presentation. It caters to two different types of developers: HTML developers, who are responsible for the graphical design of the page, and Java developers, who handle the development of software to create the dynamic content.

Because JSP is part of the J2EE standard, you can deploy JSPs on a variety of platforms, including WebLogic Server. In addition, third-party vendors and application developers can provide JavaBean components and define custom JSP tags that can be referenced from a JSP page to provide dynamic content.

## 2.2 How JSP Works

JSP pages are, at first glance, rather similar to ASP and PHP pages in that special pseudo-tags are used to mark out parts of otherwise normal HTML pages that are to handled specially. The difference is that, whereas ASP and PHP are processed by a server plug-in, JSP is handled by a separate server process. For JSP this process runs software known as Tomcat which takes the JSP page and converts the entire page into a Java programme which is then compiled. The raw HTML in the original page is wrapped in Java output methods. On requesting a JSP page, the request is transferred from the normal WWW server (e.g. Apache) to Tomcat which first checks whether it has cached an already compiled version of the Java programme associated with the page. Recompilation takes place if the last modification date stamp on the JSP page is more recent than the last modification date stamp on any cached compiled version.

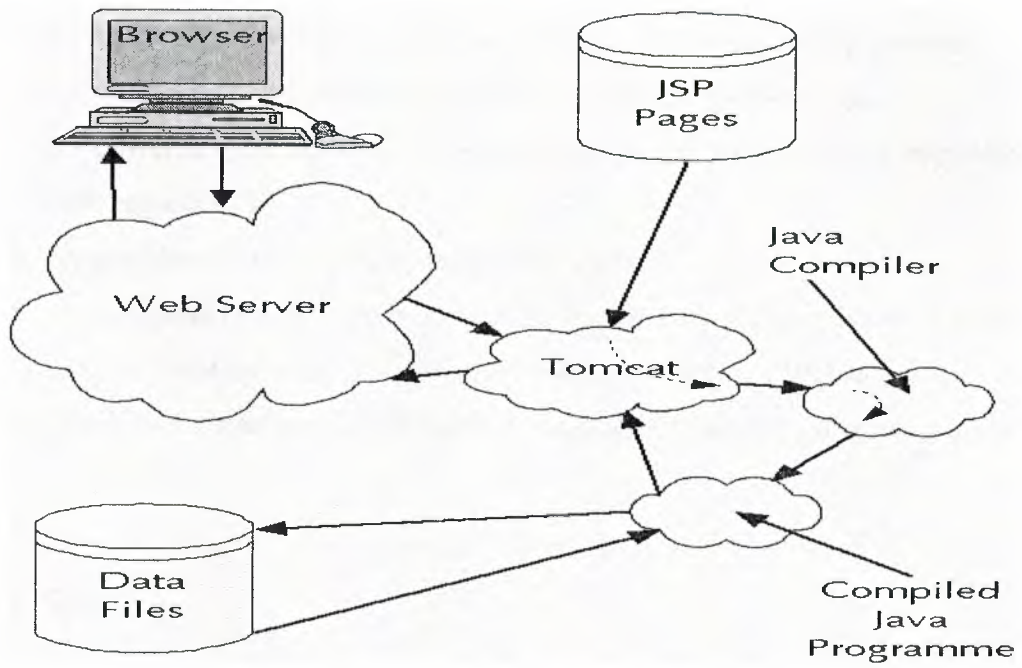


Figure 2.1: How JSP Request Pages are Handled.

## 2.3 How JSP Requests Are Handled

WebLogic Server handles JSP requests in the following sequence:

1. A browser requests a page with a .jsp file extension from WebLogic Server.
2. WebLogic Server reads the request.
3. Using the JSP compiler, WebLogic Server converts the JSP into a servlet class that implements the javax.servlet.jsp.JspPage interface. The JSP file is compiled only when the page is first requested, or when the JSP file has been changed. Otherwise, the previously compiled JSP servlet class is re-used, making subsequent responses much quicker.
4. The generated JspPage servlet class is invoked to handle the browser request.

It is also possible to invoke the JSP compiler directly without making a request from a browser. For details, see Using the WebLogic JSP Compiler. Because the JSP compiler creates a Java servlet as its first step, you can look at the Java files it produces, or even register the generated JspPage servlet class as an HTTP servlet.

## 2.4 Servlets Advantages

There are five main advantages of servlets:

1. It's efficient: the java virtual machine stays up, and each request is handled by lightweight java thread, not a heavyweight operating system process.
2. It's convenient: you are able to use java rather than learn perl too.
3. It's powerful: you can easily do several things that are difficult or impossible with regular CGI.
4. It's portable: follows a well standardized API.
5. It's inexpensive: there are a number of free or very inexpensive web servers available that are good for "personal" use or low-volume Web sites.

❖ Since java server pages (JSP) are servlets, all the benefits of servlets pertain to JSP.

## 2.5 JSP Syntax

### 2.5.1 Directives

JSPs live in an object called a container, which is essentially a server. JSPs can define information for the container with directives.

Here is what directives look like in a general form:

<%@ directive attribute="someValue" attribute="anotherValue" ... %>

There are three directives:

- <%@ page ... %> specifies information that affects the page
- <%@ include ... %> includes a file at the location of the include directive (parsed)
- <%@ taglib ... %> allows the use of custom tags in the page
- <%@ page language="java" %> will always be the first line of every JSP file.

### 2.5.2 Declarations

Declarations are used to specify supplemental methods and variables. You can think of these are the page's private functions; they can only be called by the JSP where they are defined, or by another JSP that includes it (using the <@ include > directive).

Here is a sample declaration:

```
<%!
    // this integer can be used anywhere in this JSP page
 private int myVariable = -1;
// this function can be called from anywhere in this JSP page
public boolean isPositive() {
   return ( myVariable > 0 );
}
%>
```

### 2.5.3 Scriptlets

Scriptlets are bits of Java code. They can do anything (the full power of Java is available in every JSP), but they will most likely concentrate on generating HTML.

### 2.5.4 Expressions

Expressions are special-purpose mini-scriptlets used for evaluating expressions. This could be something as simple as outputting the value of a variable, or a more complicated Java expression, like calling a function and outputting the result.

Table 2.1 JSP Syntax Summary

| JSP Element | Syntax | Interpretation |
|---|---|---|
| JSP Expression | `<%= expression %>` | Expression is evaluated and placed in output. out.println(expression); |
| JSP Scriptlet | `<% code %>` | Java Code is inserted in service method. |
| JSP Declaration | `<%! code %>` | inserted in body of servlet class, outside of service method. |
| JSP page Directive | `<%@ page att="val" %>` | Directions to the servlet engine about general setup. |
| JSP Comment | `<%-- comment --%>` | Comment; ignored when JSP page is translated into servlet. |
| JSP include Directive | `<%@ include file="url" %>` | A file on the local system to be included when the JSP page is translated into a servlet. |

# CHAPER 3: HTML & JAVASCRIPT

## 3.1 Introduction to HTML

(**H**yper**T**ext **M**arkup **L**anguage) The document format used on the Web. Web pages are built with HTML tags (codes) embedded in the text. HTML defines the page layout, fonts and graphic elements as well as the hypertext links to other documents on the Web. Each link contains the URL, or address, of a Web page residing on the same server or any server worldwide, hence "World Wide" Web.

HTML 2.0 was defined by the Internet Engineering Task Force (IETF) with a basic set of features, including interactive forms capability. Subsequent versions added more features such as blinking text, custom backgrounds and tables of contents. However, each new version requires agreement on the tags used, and browsers must be modified to implement those tags.

HTML Itself Is Not a Programming Language; HTML is a markup language (the ML in HTML) that uses a fixed set of markup tags. A markup language can also be thought of as a "presentation language," but it is not a programming language. You cannot "if this-do that" like you can in Java, JavaScript or C++. However, in order to make pages interactive, programming code can be embedded in an HTML page. For example, JavaScript is widely interspersed in Web pages (HTML pages) for that purpose.

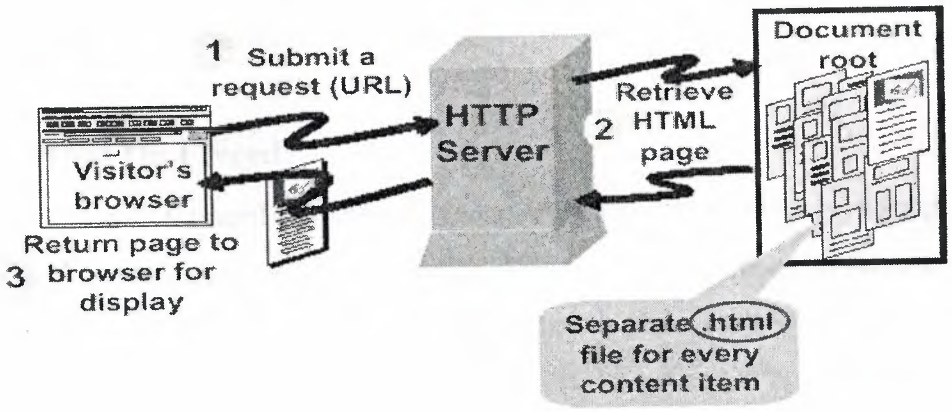


Figure 3.1: Html Pages are transferred by HTTP Protocol.

HTML was conceived as a simple markup language to render research documents. No one envisioned Web pages turning into multimedia extravaganzas.

HTML pages have been reworked, jury-rigged and extended into full-blown applications. As a result, the source code behind today's Web pages is often a hideous concoction of tags and scripting.

## 3.2 Introduction to JavaScript

Thousands of sites around the world use JavaScript but it is still not a particularly well known programming language (compared to HTML). If you have seen anything interactive on a website like a calculation, pop-up-window, some web counters and even some navigation systems then you have probably seen JavaScript. Unfortunately, JavaScript has changed from being a language which improves web sites to a language which destroys them. This is because there are huge JavaScript sites which have thousands of scripts for download. These usually involve things which do not benefit a website at all, like status bar effects and scrolling text which do not add muchtoawebsite.

JavaScript must not be confused with Java. Java is a completely different programming language. It is usually used for text effects and games, although there are some JavaScript games around.

So why should you use JavaScript? Well, JavaScript can allow you to create new things on your website that are both dynamic and interactive, allowing you to do things like find out some information about a user (like monitor resolution and browser), check that forms have been filled in correctly, rotate images, make random text do calculations and many other things.

### 3.2.1 What Do I Need?

You will not need any special hardware or software to write JavaScript, you can just use Notepad or any other text editor. You do not even need to have any special software on your webserver. JavaScript will run on any webserver anywhere! All you will need to do is make sure that you have at least a version 3 browser, as versions of Internet Explorer and Netscape Navigator before this do not support JavaScript, so you will not be able to see your creations.

### 3.2.2 Declaring JavaScript

Adding JavaScript to a web page is actually surprisingly easy! To add a JavaScript all you need to add is the following (either between the <head></head> tags or between the <body></body> tags.

```
<script language="JavaScript">
        JavaScript
</script>
```

As you can see the JavaScript is just contained in a normal HTML tag set. The next thing you must do is make sure that the older browsers ignore it. If you don't do this the code for the JavaScript will be shown which will look awful.

There are two things you must do to hide the code from older browsers and show something instead:

```
<script language="JavaScript">
<!—Begin Hide
JavaScript
//End Hide-->
</script>
<noscript>
HTML Code
</noscript>
```

As you can see this makes the code look a lot more complex, but it is really quite simple. If you look closely you can see that all that has been done is that the JavaScript is now contained in an HTML comment tag. This is so that any old browsers which do not understand <script> will just think it is an HTML comment and ignore it.

Because of this the <noscript> tag was created. This will be ignored by browsers which understand <script> but will be read by the older browsers. All you need to do is put between <noscript> and </noscript> what you want to appear if the browser does not support JavaScript. This could be an alternative feature or just a message saying it is not available. You do not have to include the tag if you don't want anything shown instead.

# CHAPTER 4: DATABASE DESIGN

## 4.1 Introduction

Many web developers are self-taught, learning HTML, then moving on to a programming language such as JSP. From there, they often learn to integrate this with a database. Too few though have a good theoretical knowledge of databases. Mention foreign keys, or referential integrity, and you're met with a blank stare. Small databases can be easily designed with little database theory knowledge. But large databases can easily get out of hand when badly designed, leading to poor performance, and resulting in the whole database needing to be rebuilt later. This article is a brief introduction to the topic of relational databases, and will hopefully whet your appetite for further exploration.

## 4.2 The Relational Database Model

*A database can be understood as a collection of related files. How those files are* related depends on the model used. Early models included the hierarchical model (where files are related in a parent/child manner, with each child file having at most one parent file), and the network model (where files are related as owners and members, similar to the network model except that each member file can have more than one owner).

The relational database model was a huge step forward, as it allowed files to be related by means of a common field. In order to relate any two files, they simply need to have a common field, which makes the model extremely flexible.

## 4.3 The Entity-Relationship Model

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represent data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design for the database designer, the utility of the ER model is:

- It maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.

- It is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.
- In addition, the model can be used as a design plan by the database developer to implement a data model in a specific database management software.

### 4.3.1 Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities.

#### 4.3.1.1 Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, INVOICES. An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one that does not rely on another for identification. A dependent entity is one that relies on another for identification.

An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

#### 4.3.1.2 Special Entity Types

Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

#### 4.3.1.3 Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

- employees are assigned to projects
- projects have subtasks
- departments manage one or more projects

Relationships are classified in terms of degree, connectivity, cardinality, and existence. These concepts will be discussed below.

### 4.3.1.4 Attributes

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

### 4.3.1.5 Classifying Relationships

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

### 4.3.1.6 Degree of a Relationship

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

### 4.3.1.7 Connectivity and Cardinality

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

### 4.3.1.8 Direction

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

### 4.3.1.9 Type

An identifying relationship is one in which one of the child entities is also a dependent entity. A non-identifying relationship is one in which both entities are independent.

### 4.3.1.10 Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either mandatory or optional. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory.

## 4.3.2 ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is widely used in academics texts and journals but rarely seen in either CASE tools or publications by non-academics. Today, there are a number of notations used; among the more common are Bachman, crow's foot, and IDEFIX.

Here are the notations (crow's foot notation) used in the E-R diagram of the project (see figure 4.1):

- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.
- Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
- Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

## 4.4 Relational Database of Online Course Registration:

After talking about E-R model and having some knowledge about it, now it is the time to implement the shown E-R diagram (figure 4.1) so to obtain the real model(the needed database).

The aimed database is needed for the faculty of computer engineering at our university, information about students, staffs, courses, terms and some process which belongs either to student or to staff are to be stored into the database.

There are 10 tables which were implemented on Ms. Access to cover the need of storing the necessary information.

The Student table contains student information. studentId is the primary key to uniquely identity each student. firstName and surname fields keep the student full name. The table contains advisorId as foreign key to reference the Staff table so to keep track of student advisor's information. password field is used to verify the student while trying to log into his account.

The Staff table contains staffId field as the primary key. name and surName fields save the staff full name. Staff degree is obtained by the foreign key degreeId field which references the staffDegree table. The roomId field is used to get the room number of the staff. password field is added to achieve what the Student table's password field does.

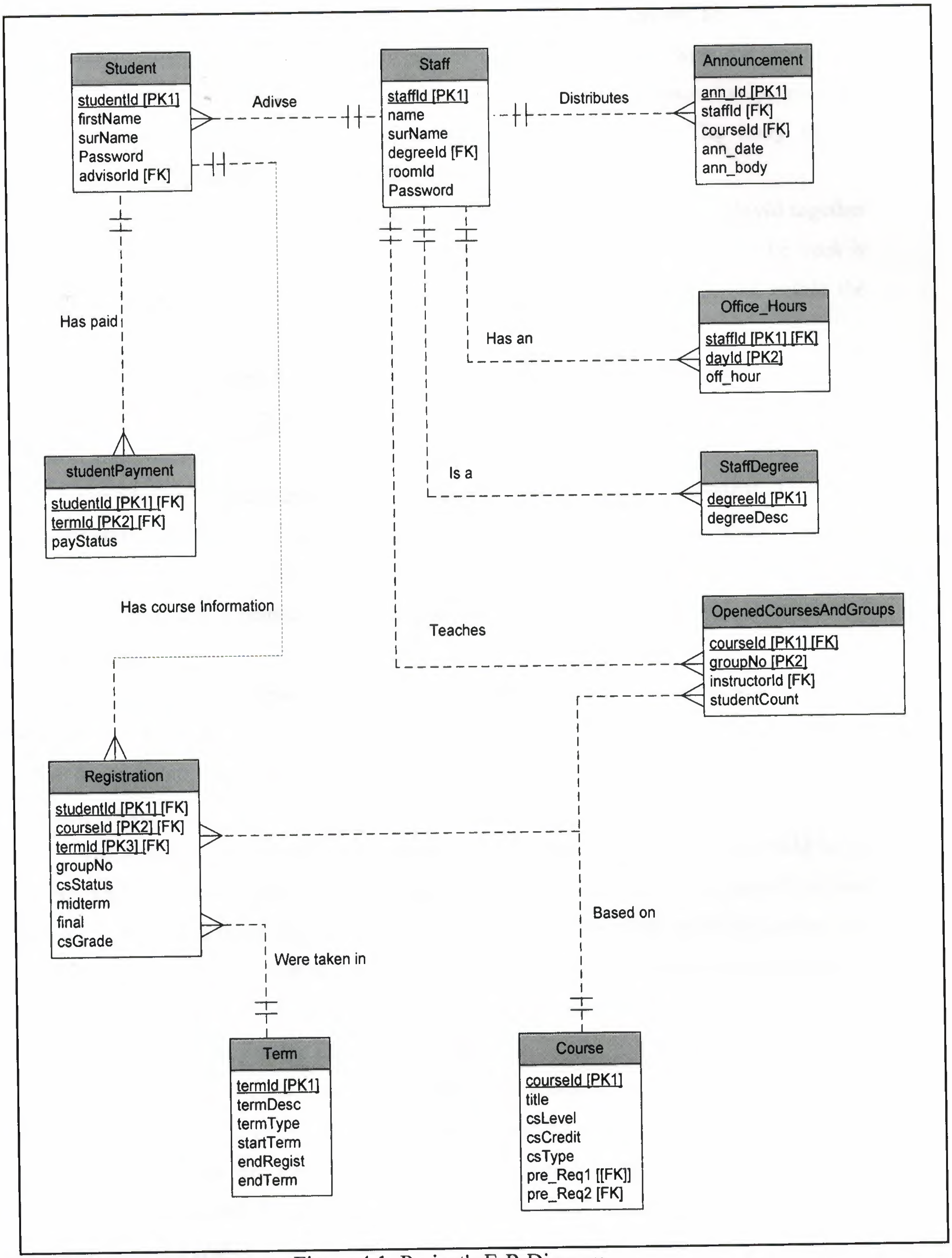


Figure 4.1: Project's E-R Diagram

The Announcement table records are identified by the primary key ann_Id. The foreign keys staffId and courseId fields are to reference the staff who did added the announcement and the course which the announcement does belong respectively. ann_date column marks the date on which the announcement was added. The body of the announcement is kept within the column ann_body.

The Office_Hours table is structured with three columns. staffId and dayId together eliminate the possibility of the duplication (composite primary key). Each day of the week is assigned with an Id (dayId field). Staff office hours for each day are defined within the off_hour field.

The StaffDegree table contains all possible degree classification (Mr., Dr., Prof...etc) with the primary key degreeId. Degree with full string representation is hold within degreeDesc field.

The StudentPayment table keeps a track of student term's fee payment. This table most probably is a view, so it takes the records from the university main database. studentId and termId are composite primary key and on the same time foreign keys that reference Student and Term tables respectively. Term fee payment status (paid, not paid) is determined by the column payStatus.

The Term table (primary key termId) contains term information. termDesc field holds the term title (e.g. Fall 2005/2006). The fields startTerm, endRegist, endTerm are used to define the start date of the term, the dead date for course registration and the end date of the term respectively.

The Course table contains the courses which the university offers. courseId is the primary key. The title of the course is saved in the field title. csLevel field specify at what term level the course is scheduled to be taken. csCredit defines what credit the course does has. csType determines the course type (elective, technical, non-technical). The prerequisites information are kept in the pre_Req1 and pre_Req2 columns which serve as foreign keys and reference the primary key of the same Course Table.

The Registration table contains each student's schedule for every registration term. studentId, courseId and termId are the composite primary key. The table contains two "lookup" tables which are Course and Term. The course status (failed, passed, currently taken) can be obtained from csStatus column. The table provides each taken course's midterm, final and grade results.

The OpenedCoursesAndGroups table contains the offered courses and groups in every term. courseId and groupNo do uniquely identify each record. The table contains a single foreign key "instructorId" column which references the Staff table so to keep a track of course group instructor information.

The following figure shows the entities relationships implemented on Ms.

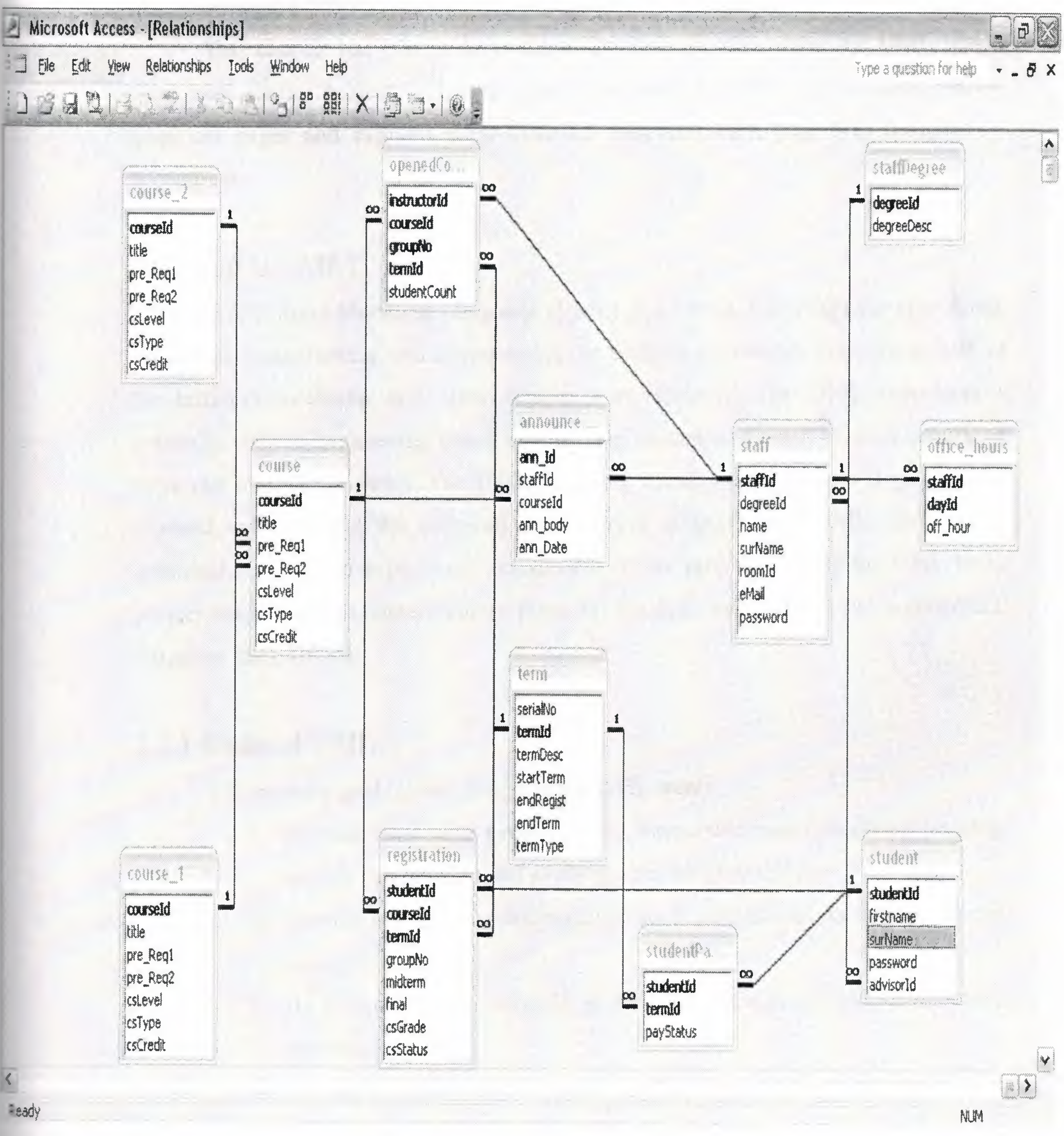


Figure 4.2: The Database of the Project was created on MS Access.

# CHAPTER 5: Online Course Registration Software Design

## 5.1 Introduction

In The previous four chapters were introduced the techniques (Java, JSP, HTML, JavaScript and RDBMS applied on Ms. Database) which were used for implementing the project.

This chapter has two main sections: first section talks about the UML which were used during project designing phase; the second section shows some system graphical pages and explains what technical functions each page was designed to accomplish.

## 5.2 What is UML?

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

### 5.2.1 Goals of UML

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.

6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

### 5.2.2 Use Case

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

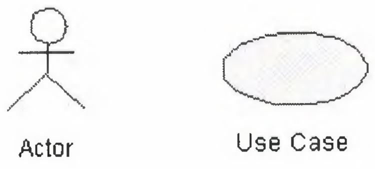


Figure 5.1

### 5.2.3 Use Case Diagrams Depict:

- Use cases. A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.
- Actors. An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.
- Associations. Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicating the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. (It is possible to have association between tow use cases)

In the next section it is shown the use cases of the project. (Figure 5.2)

### 5.2.3.1 Project Use Case Diagrams

Since the system (project) belongs to both students and staffs, all possible actions are initiated and/or participated by them, therefore the system has tow actors, student and instructor see figure 5.2.

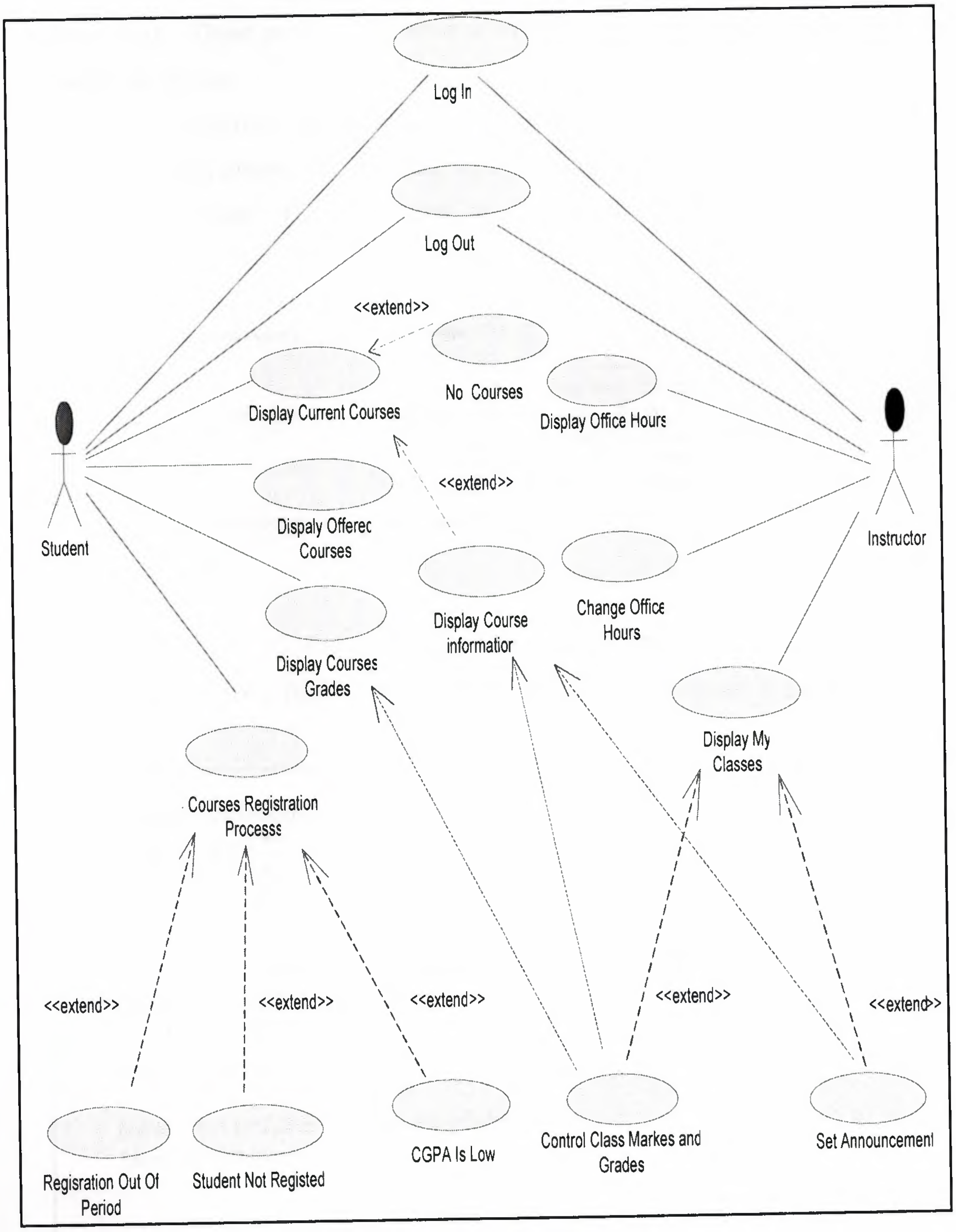


Figure 5.2: Project Use Case Diagrams

## 5.2.4 Class Diagrams

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation (figure 5.3). These perspectives become evident as the diagram is created and help solidify the design.

Class diagrams also display relationships such as containment, inheritance, associations and others. The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes.

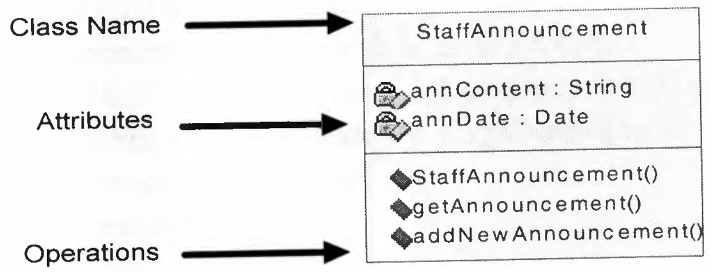


Figure 5.3: Class Diagram Description

The following figure shows the class “StaffAnnouncement” source code.

```
package neu.faculties.comEng;
import java.sql.*;
import java.util.*;
import java.io.*;
public class StaffAnnouncement
{    // attribute definitions
 private String annContent;
 private java.util.Date annDate;
 public StaffAnnouncement() { }
       // set announcement Body
 public void setAnnBody(String annBdy)
 {annContent = annBdy; }
```

```
            // set announcement Date
public void setAnnDate(java.util.Date ann_Date)
 { annDate = ann_Date; }
public String getAnnBody()
 {  return annContent; }
public java.util.Date getAnnDate()
 {    return annDate; }
            // method is to handle the decleration of a new announcement
public boolean addNewAnnouncement(StaffInfo sf, CourseInfo cs, String announcement) throws SQLException
 { String annUpdate = "INSERT INTO announcement(staffId, courseId, ann_body)"
       + " VALUES('" + sf.getstaffId().trim() + "','" + cs.getCourseId().trim() + "','" + announcement.trim() + "')";
   DomainDB.handleUpdate(annUpdate);
   return true; } // end method addNewAnnouncement()
            // the method returns all announcement that belongs to the staff of the specified course
public Vector getAnnouncement(StaffInfo sf, CourseInfo cs ) throws SQLException
  { String annQuery = "SELECT ann_body, ann_Date FROM announcement  WHERE staffId = '" + sf.getstaffId
                        ().trim() + "'"  + " AND courseId = '" + cs.getCourseId().trim() + "'" ;
   ResultSet annSet = DomainDB.handleQuery(annQuery);
   boolean moreAnn = annSet.next();
   if(moreAnn)
   { Vector annVector = new Vector();
  while(moreAnn)
  { StaffAnnouncement stfAnn = new StaffAnnouncement();
    stfAnn.setAnnBody(annSet.getString(1));
    stfAnn.setAnnDate(annSet.getDate(2));
    annVector.add(stfAnn);
    moreAnn = annSet.next(); } // end while
    return annVector; } // end if  // Staff has not declared any announement for the specified course.
    return null } // end method getAnnouncement() }
// END Class StaffAnnouncement
```

Figure 5.4: StaffAnnouncement Class Source Code

### 5.2.4.1 Project Class Diagrams

Now let us take a look on the project class diagrams which only shows the classes and the class's relationships for simplicity:

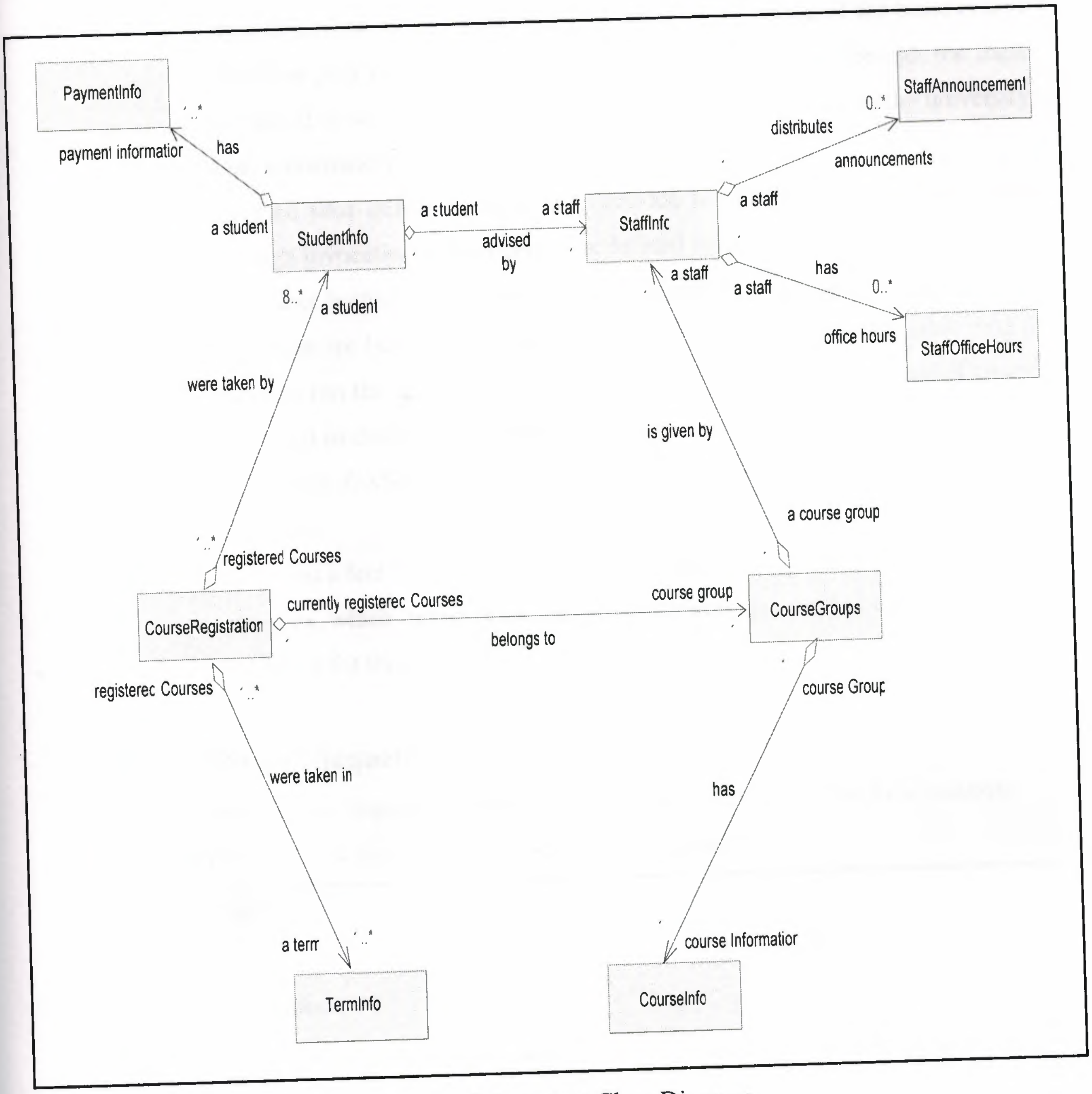


Figure 5.5: Project Class Diagram

### 5.2.5 Sequence Diagrams

UML Sequence diagrams are a dynamic modeling technique, as are collaboration diagrams and activity diagrams. UML sequence diagrams are typically used to:

1. Validate and flesh out the logic of a usage scenario. A usage scenario is exactly what its name indicates – the description of a potential way that your system is used. The logic of a usage scenario may be part of a use case, perhaps an alternate course; one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action plus one or more alternate scenarios; or a pass through the logic contained in several use cases, for example a student enrolls in the university then immediately enrolls in three seminars.
2. Explore your design because they provide a way for you to visually step through invocation of the operations defined by your classes.
3. To detect bottlenecks within an object-oriented design. By looking at what messages are being sent to an object, and by looking at roughly how long it takes to run the invoked method, you quickly get an understanding of where you need to change your design to distribute the load within your system. In fact some CASE tools even enable you to simulate this aspect of your software.
4. Give you a feel for which classes in your application are going to be complex, which in turn is an indication that you may need to draw state chart diagrams for those classes.

### 5.2.5.1 Project Sequence Diagrams

Here is some sequence diagrams of the system which describe how requests (invocations) are sent and handled between various classes:

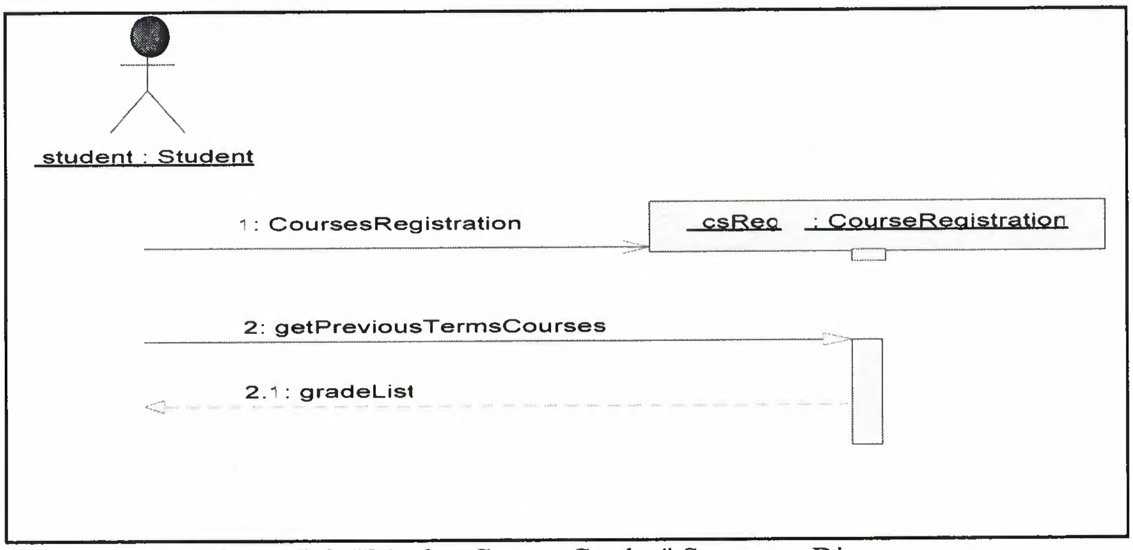


Figure 5.6: "Display Course Grades" Sequence Diagram.

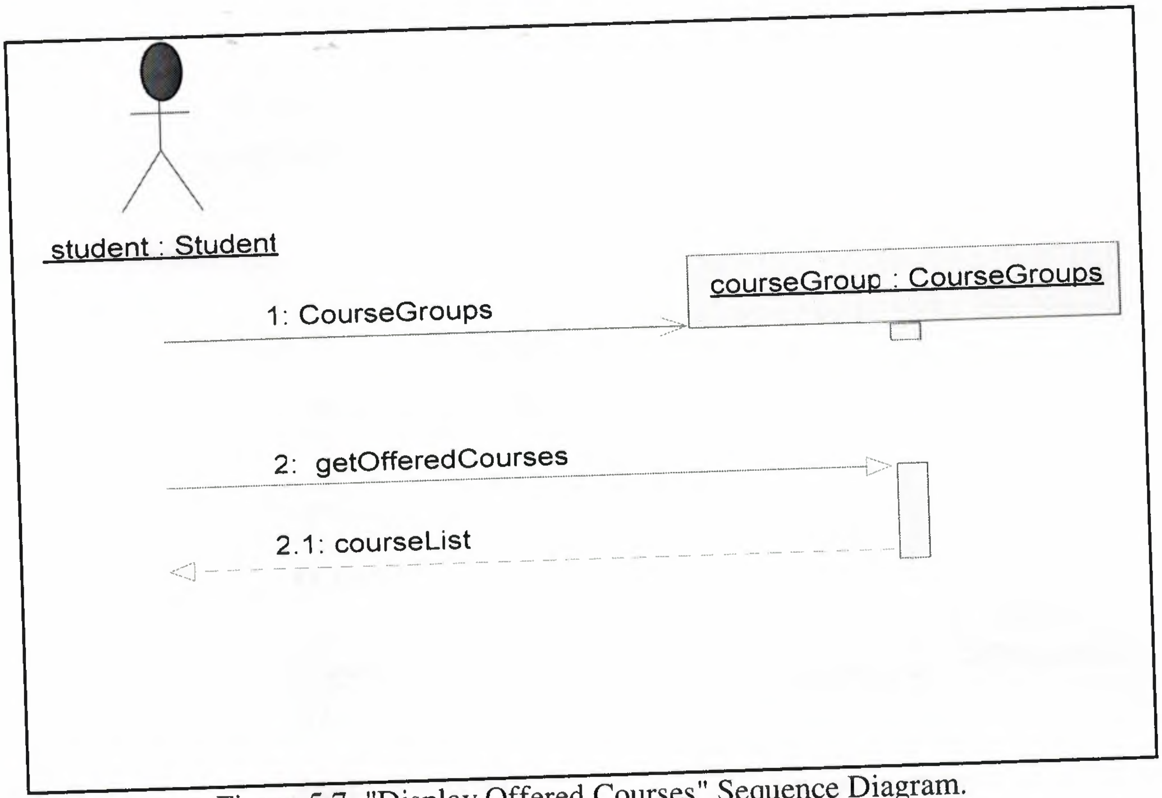


Figure 5.7: "Display Offered Courses" Sequence Diagram.

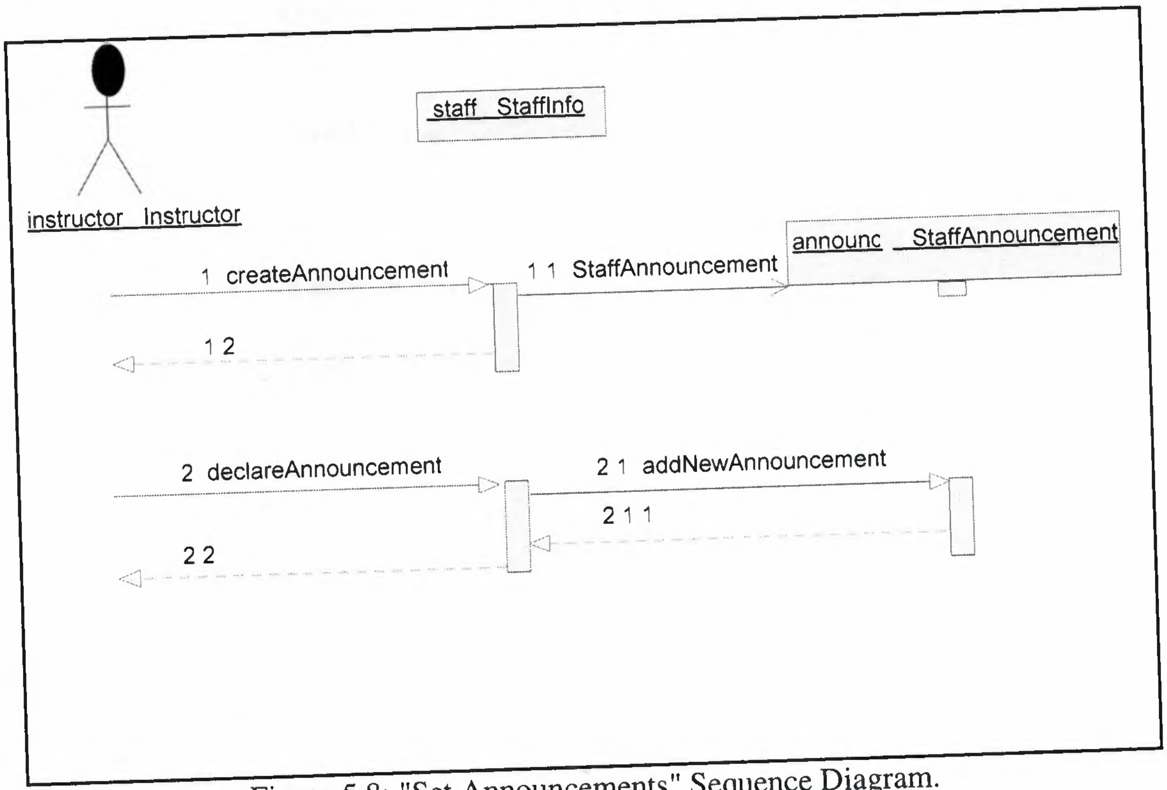


Figure 5.8: "Set Announcements" Sequence Diagram.

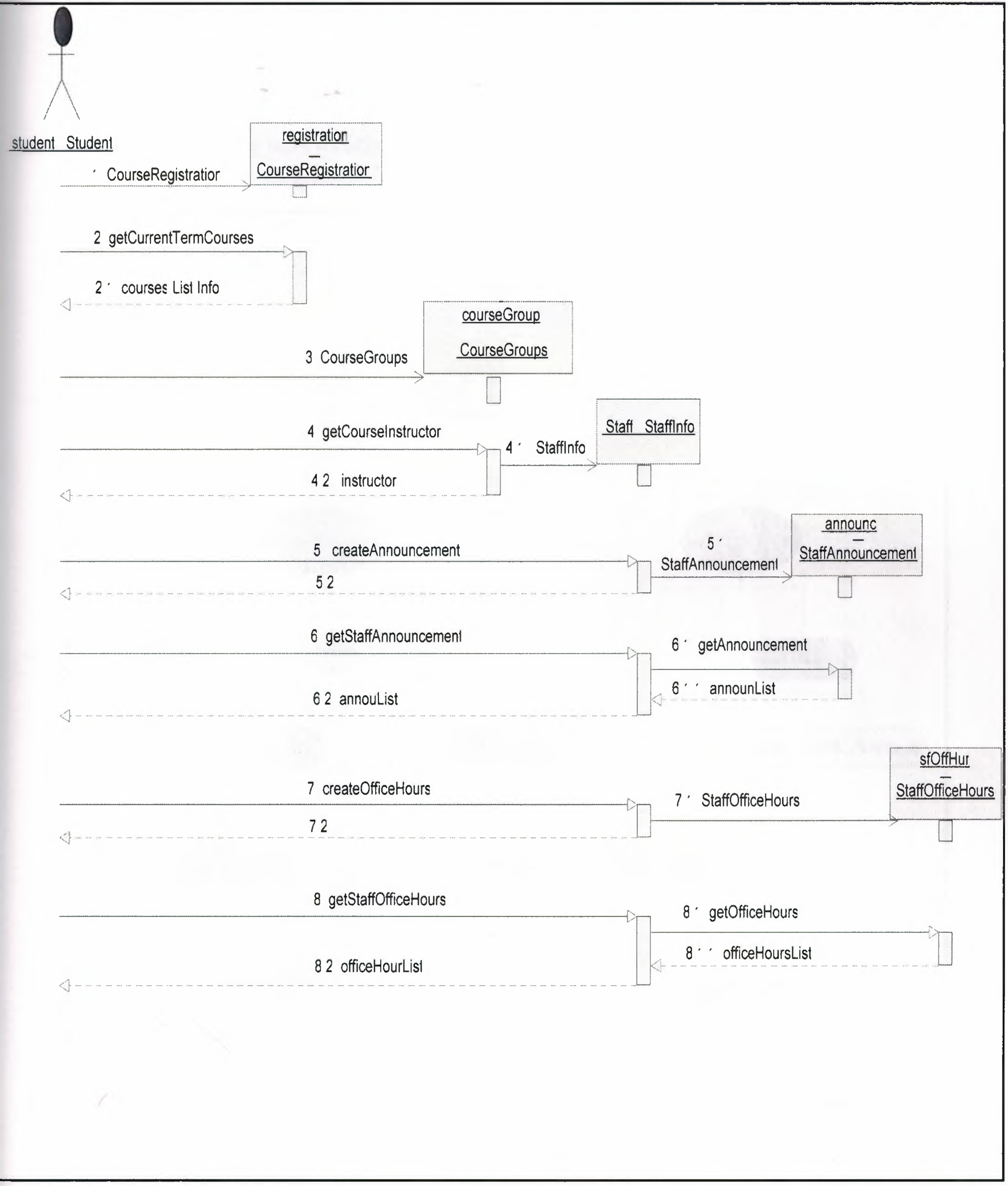


Figure 5.9: "Display Course Information" Sequence Diagram.

## 5.3 Project Files Diagram

The following diagram shows how the project's files are related to each other. The diagram illustrates how the program structure looks like. Some files will be discussed in the next section.

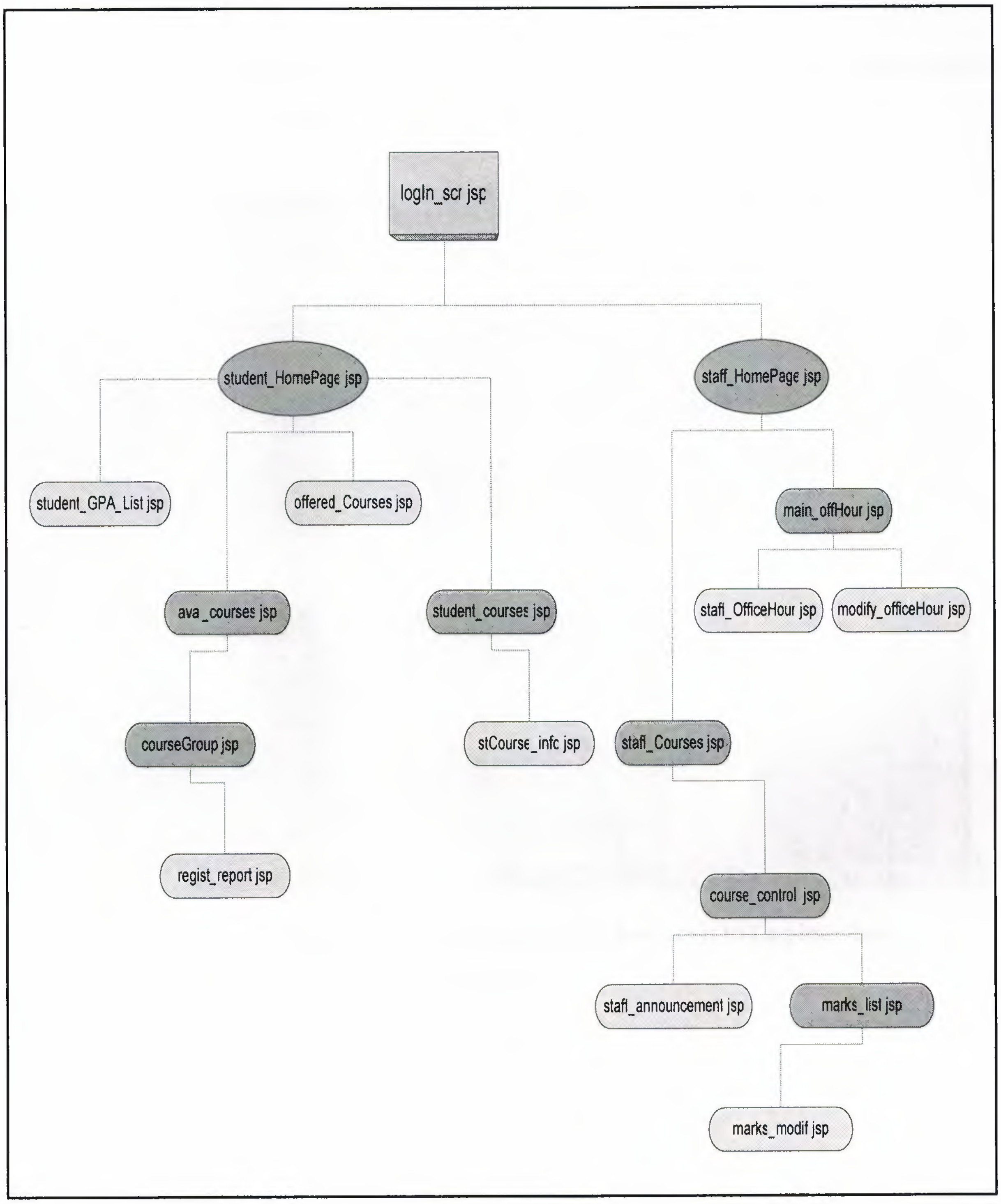


Figure 5.10: Project Files Association Diagram.

## 5.4 Project Graphical User Interface (GUI)

A graphical user interface is the part which gives a sense and presents a user - friendly mechanism for interacting with a program.

This section presents some pages (GUI) of the project and gives a brief description of the technical purposes of each page.

The figure below shows the system home page, it enables both students and staffs (instructors) to log into their accounts.

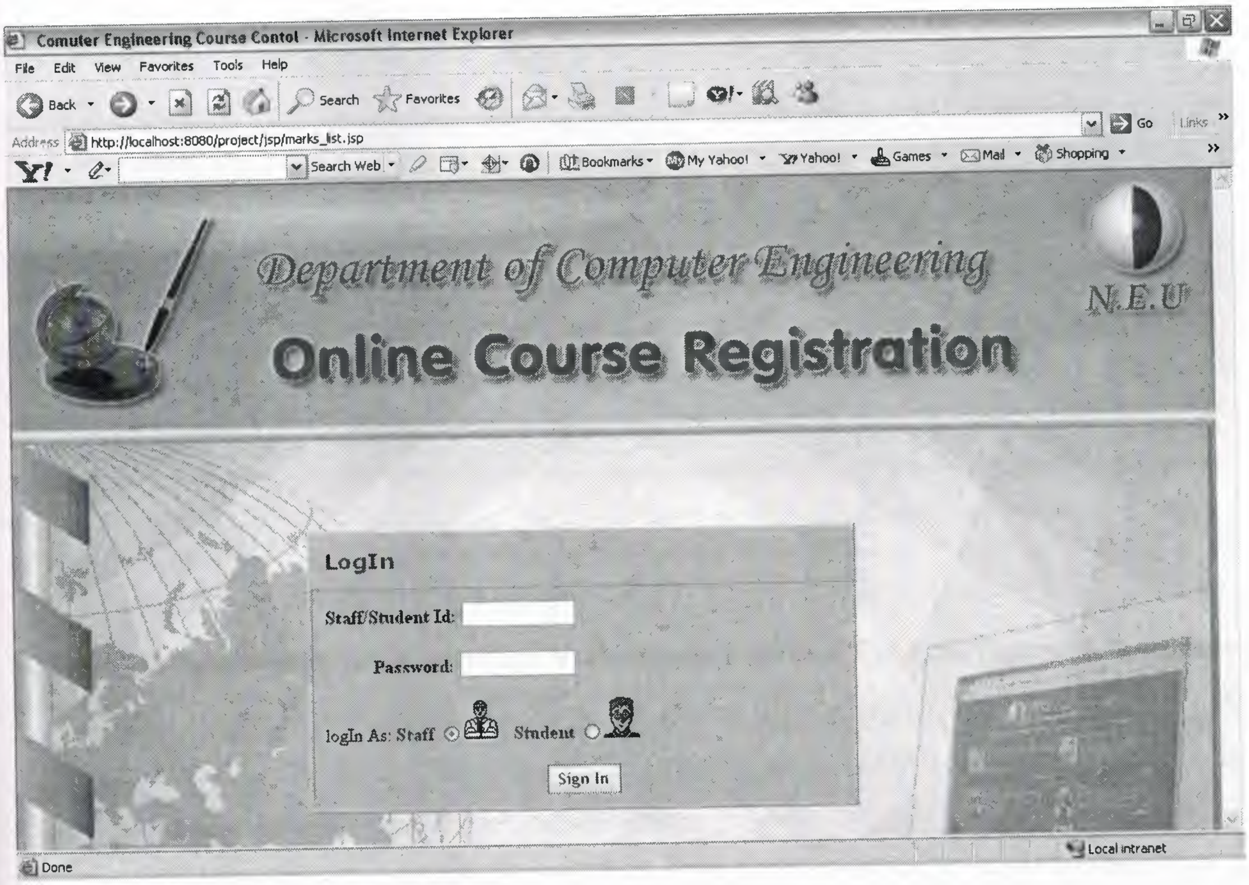

Comuter Engineering Course Contol - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://localhost:8080/project/jsp/marks_list.jsp
Department of Computer Engineering
Online Course Registration
N.E.U
LogIn
Staff/Student Id:
Password:
logIn As: Staff Student
Sign In
Done
Local intranet

Figure 5.11: Home Page Enables Both Student and Instructor to Log into their Accounts.

When the student login, his home page will be displayed (figure 5.11).

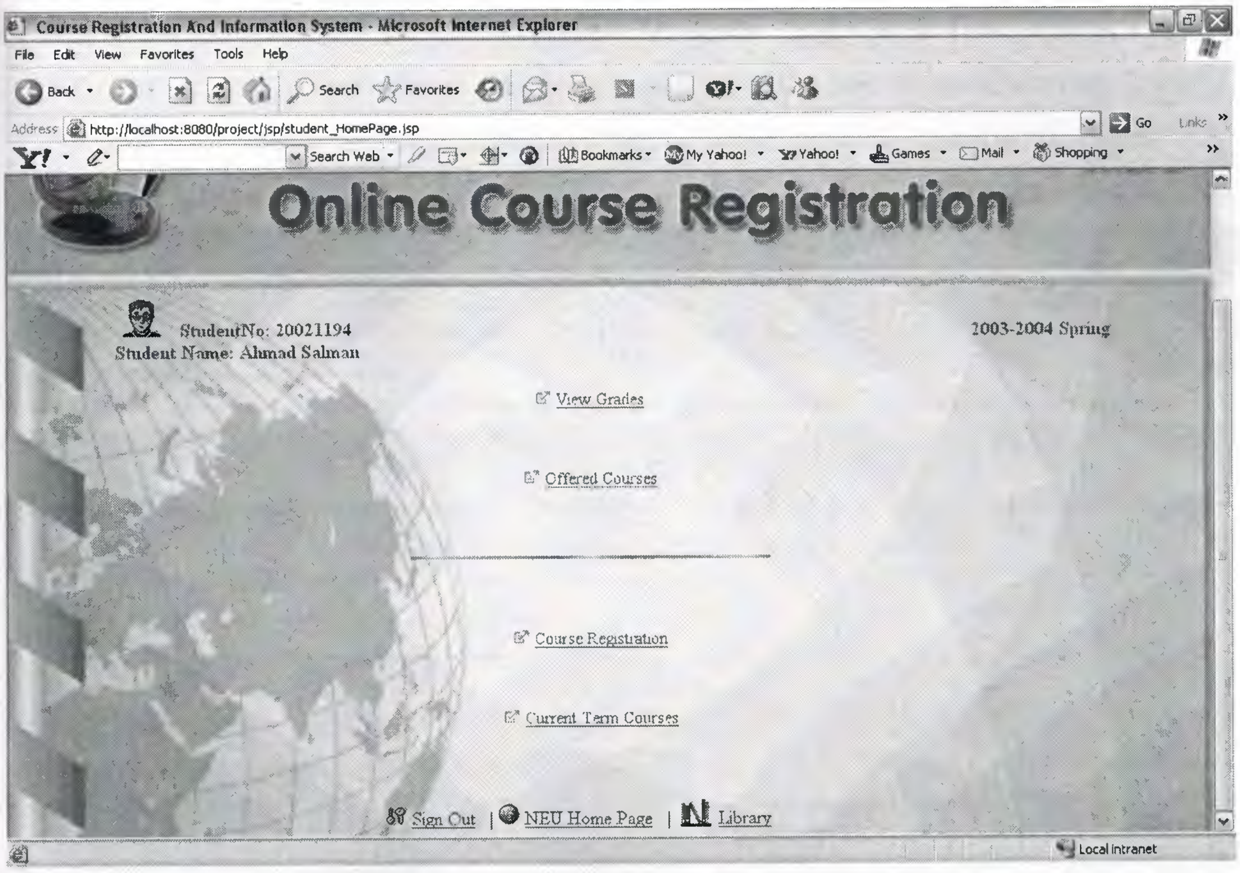

Course Registration And Information System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/project/jsp/student_HomePage.jsp
Online Course Registration
StudentNo: 20021194
Student Name: Ahmad Salman
2003-2004 Spring
View Grades
Offered Courses
Course Registration
Current Term Courses
Sign Out | NEU Home Page | Library
Local intranet

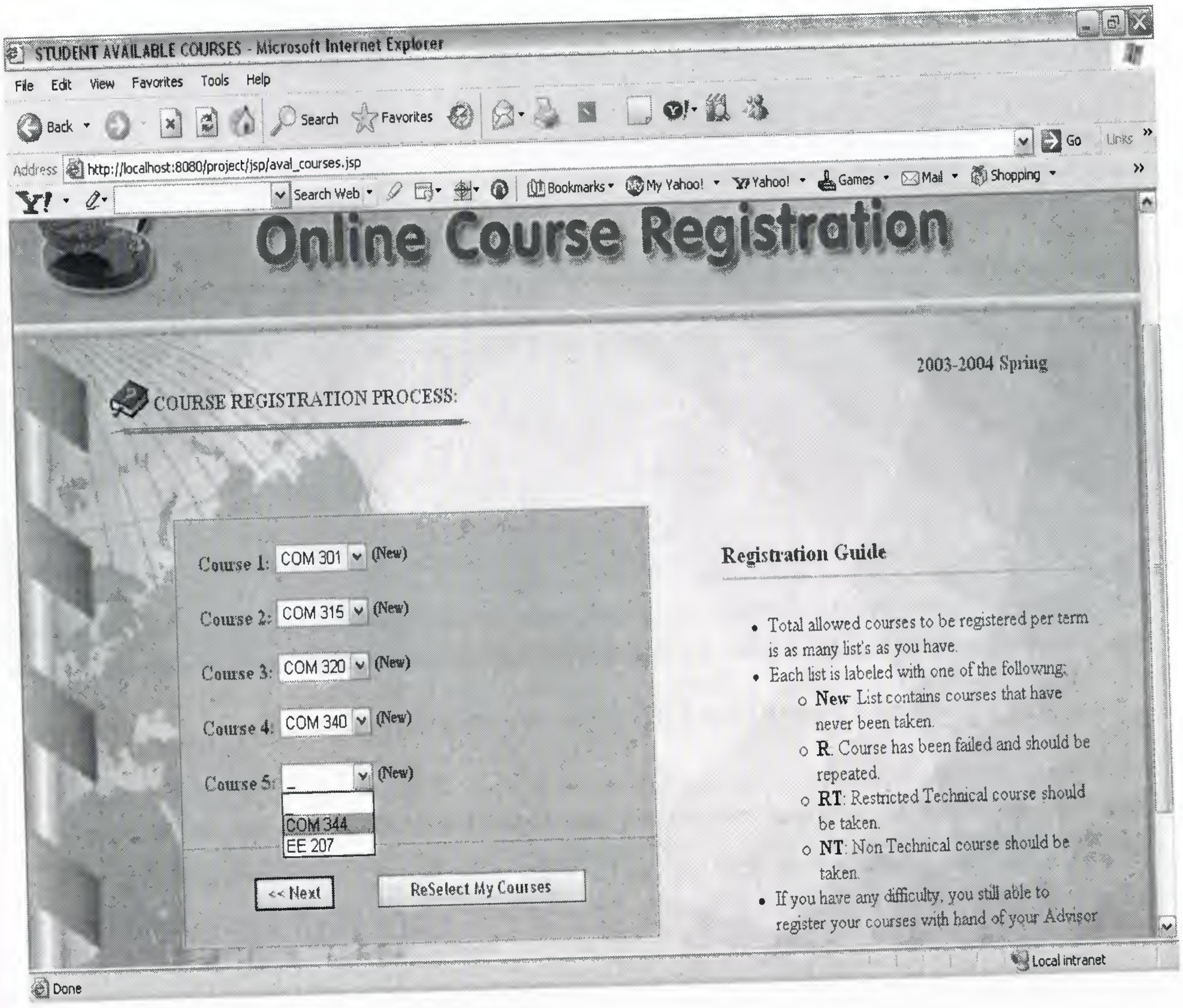
Figure 5.12: Student Home Page.

The page has four links: View Grades, Offered Courses, Course Registration and Current Term Courses. The first link allows the student to check his grades, GPA and CGPA. The second link provides information about the offered courses and groups as well as the instructor of each course group.

Course Registration link directs the student to the page which provides the courses that are available and can be registered by the student (See figure 5.13).

The available courses are decided by taking into the consideration many points, student passed courses, CGPA and student achieved term level are the main restrictions that find out the available courses for each student.

The available courses are marked with "New", "R" or "NT". The word "New" stand to mention that the course has not been taken before. The letter R (repeat) says that this course has to be repeated (student has failed this course) and it is the time to do it. Finally NT (non technical) forces the student to register one of the non technical courses. "New", "R" and "NT" courses decision making depends on some course registration algorithms.


STUDENT AVAILABLE COURSES - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost:8080/project/jsp/aval_courses.jsp
Go
Links
Search Web
Bookmarks
My Yahoo!
Yahoo!
Games
Mail
Shopping
Online Course Registration
COM 344
EE 207
<< Next
ReSelect My Courses
Done
Local intranet

2003-2004 Spring

COURSE REGISTRATION PROCESS:

Course 1: COM 301 (New)

Course 2: COM 315 (New)

Course 3: COM 320 (New)

Course 4: COM 340 (New)

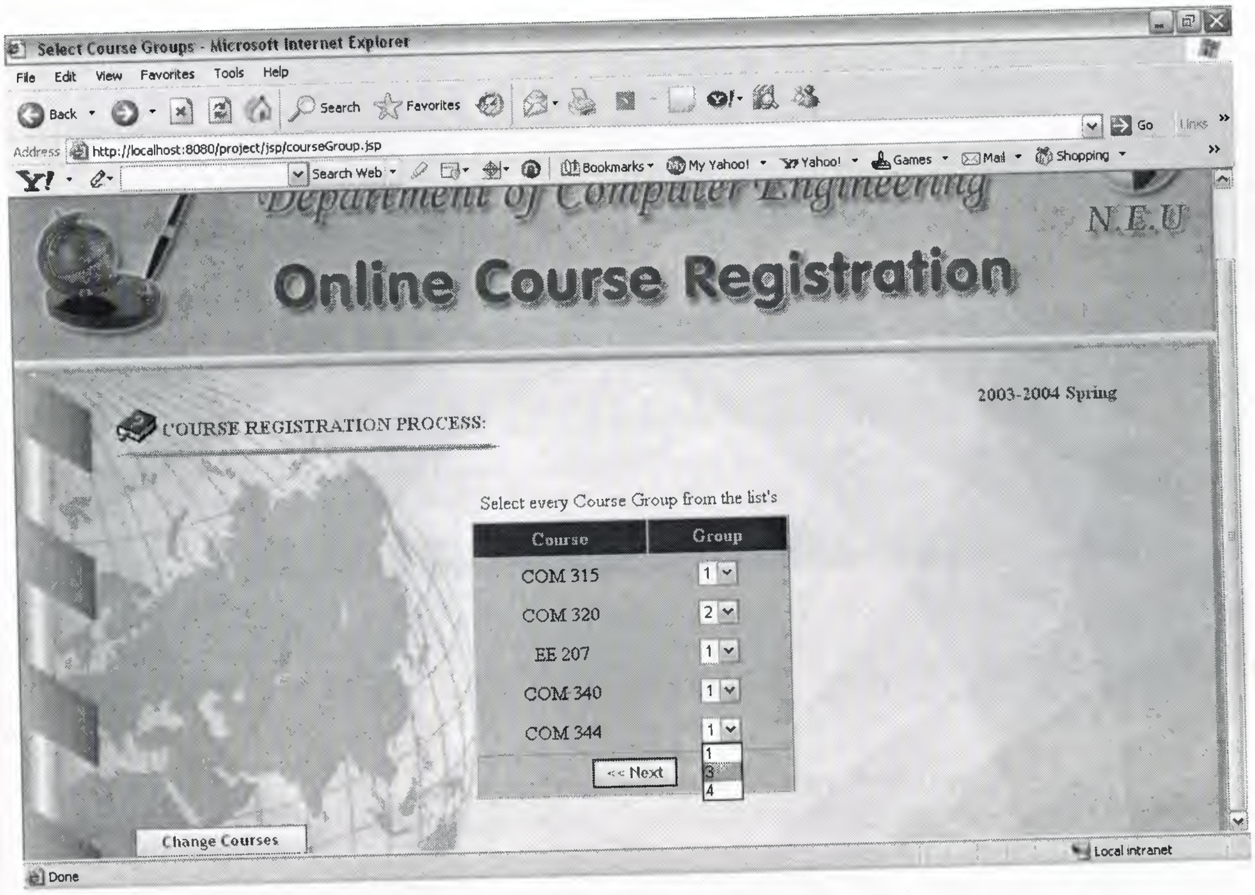
Course 5: _ (New)

Registration Guide

- Total allowed courses to be registered per term is as many list's as you have.
- Each list is labeled with one of the following:
  - New: List contains courses that have never been taken.
  - R: Course has been failed and should be repeated.
  - RT: Restricted Technical course should be taken.
  - NT: Non Technical course should be taken.
- If you have any difficulty, you still able to register your courses with hand of your Advisor

Figure 5.13: Student Available Courses during Registration Process.

After selecting the term courses, the student can select the group number of each course like the following:

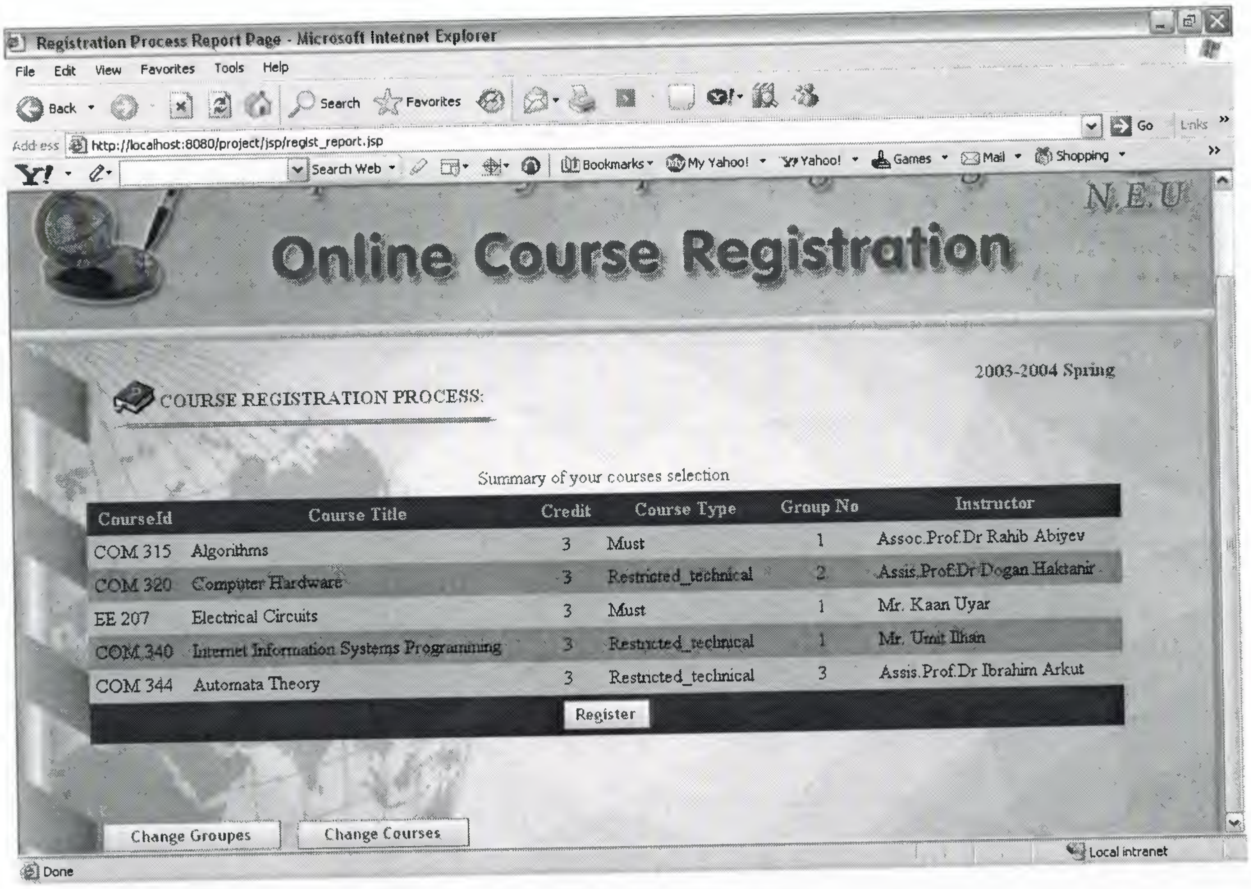

Select Course Groups - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://localhost:8080/project/jsp/courseGroup.jsp
Department of Computer Engineering
N.E.U
Online Course Registration
2003-2004 Spring
COURSE REGISTRATION PROCESS:
<< Next
Change Courses
Done
Local intranet

Select every Course Group from the list's

| Course | Group |
|---|---|
| COM 315 | 1 |
| COM 320 | 2 |
| EE 207 | 1 |
| COM 340 | 1 |
| COM 344 | 1 |

Figure 5.14: Student can select his Term Courses Groups.

At the end, student will be shown his courses and groups' selection and other related information. Course title, credit, type as well as the instructor name will be displayed. (See figure 5.14)


Registration Process Report Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://localhost:8080/project/jsp/regist_report.jsp
Online Course Registration
N.E.U
Change Groupes
Change Courses
Done
Local intranet

2003-2004 Spring

COURSE REGISTRATION PROCESS:

Summary of your courses selection

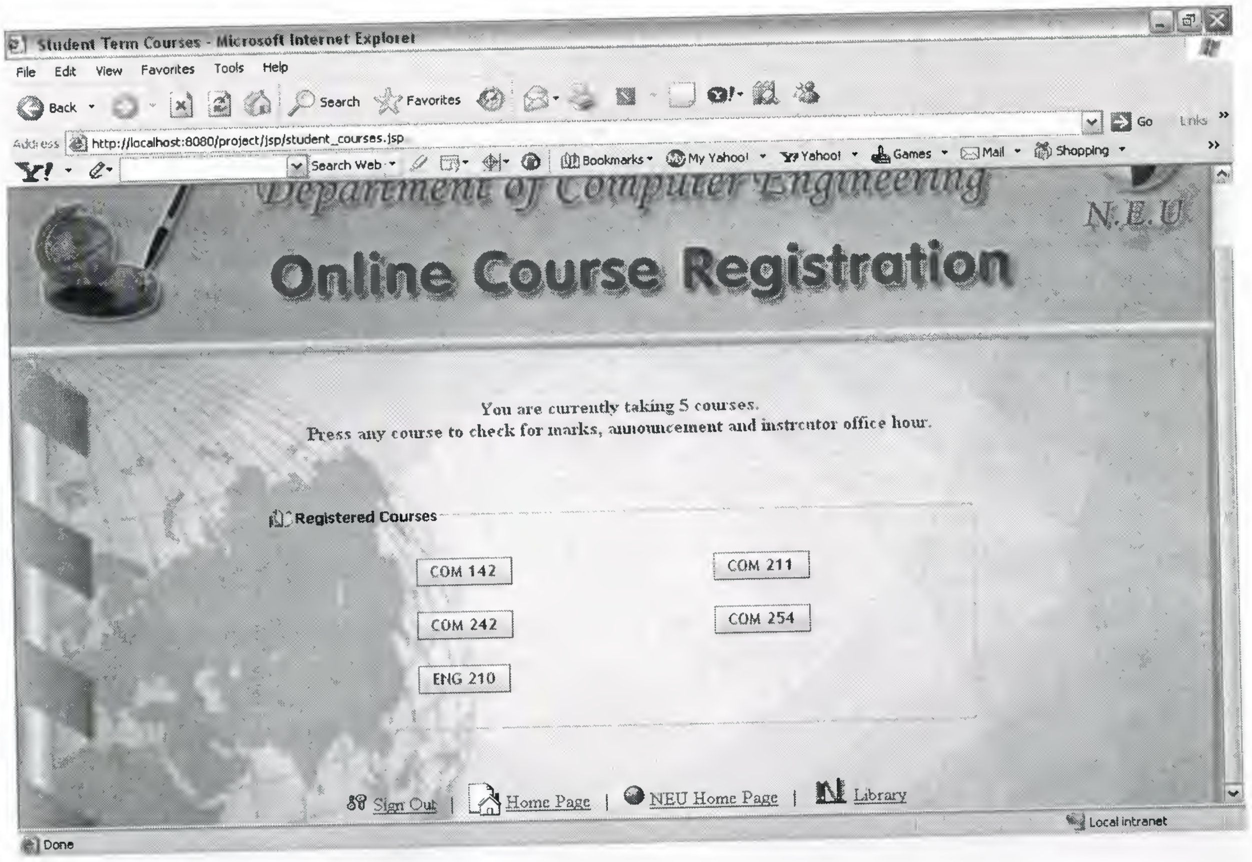
| CourseId | Course Title | Credit | Course Type | Group No | Instructor |
|---|---|---|---|---|---|
| COM 315 | Algorithms | 3 | Must | 1 | Assoc.Prof.Dr Rahib Abiyev |
| COM 320 | Computer Hardware | 3 | Restricted_technical | 2 | Assis.Prof.Dr Dogan Haktanir |
| EE 207 | Electrical Circuits | 3 | Must | 1 | Mr. Kaan Uyar |
| COM 340 | Internet Information Systems Programming | 3 | Restricted_technical | 1 | Mr. Umit Ilhan |
| COM 344 | Automata Theory | 3 | Restricted_technical | 3 | Assis.Prof.Dr Ibrahim Arkut |

Register

Figure 5.15: Summary of Courses Selection.

During the term, students need to check for instructor office hours, announcement and exam result; these needs are covered by the fourth link "Current Term Courses".

This link will direct the student to the page which contains the registered courses each represented by a button (figure 5.15). Course information can be viewed by clicking on that course button (figure 5.16).


Department of Computer Engineering

Online Course Registration

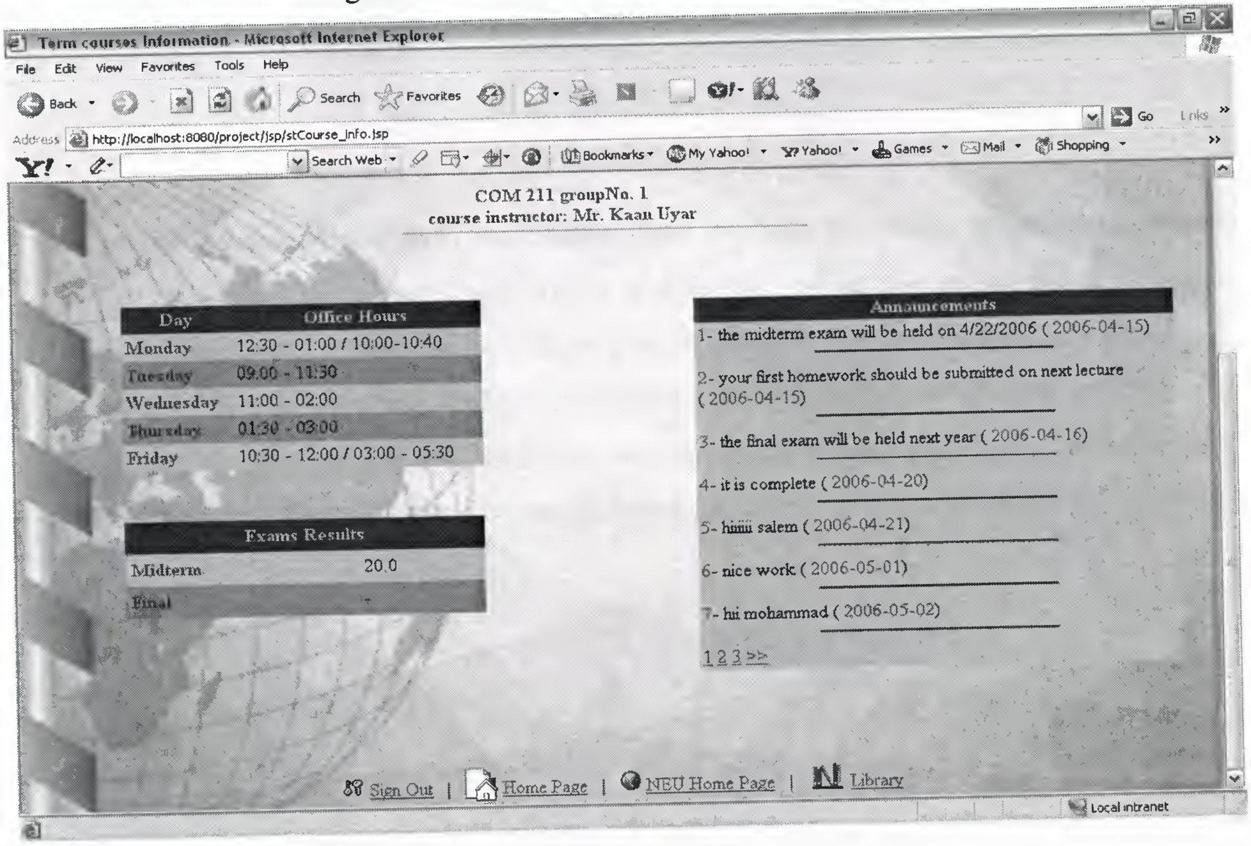
You are currently taking 5 courses.
Press any course to check for marks, announcement and instrcutor office hour.

Registered Courses

COM 142 | COM 211
COM 242 | COM 254
ENG 210

Student Term Courses - Microsoft Internet Explorer
http://localhost:8080/project/jsp/student_courses.jsp
N.E.U
Sign Out | Home Page | NEU Home Page | Library
Done
Local intranet

Figure 5.16: Student Currently Taken Courses.


COM 211 groupNo. 1
course instructor: Mr. Kaan Uyar

| Day | Office Hours |
|---|---|
| Monday | 12:30 - 01:00 / 10:00-10:40 |
| Tuesday | 09:00 - 11:30 |
| Wednesday | 11:00 - 02:00 |
| Thursday | 01:30 - 03:00 |
| Friday | 10:30 - 12:00 / 03:00 - 05:30 |

| Exams Results | |
|---|---|
| Midterm | 20.0 |
| Final | - |

Announcements

1- the midterm exam will be held on 4/22/2006 ( 2006-04-15)

2- your first homework should be submitted on next lecture ( 2006-04-15)

3- the final exam will be held next year ( 2006-04-16)

4- it is complete ( 2006-04-20)

5- hiiiii salem ( 2006-04-21)

6- nice work ( 2006-05-01)

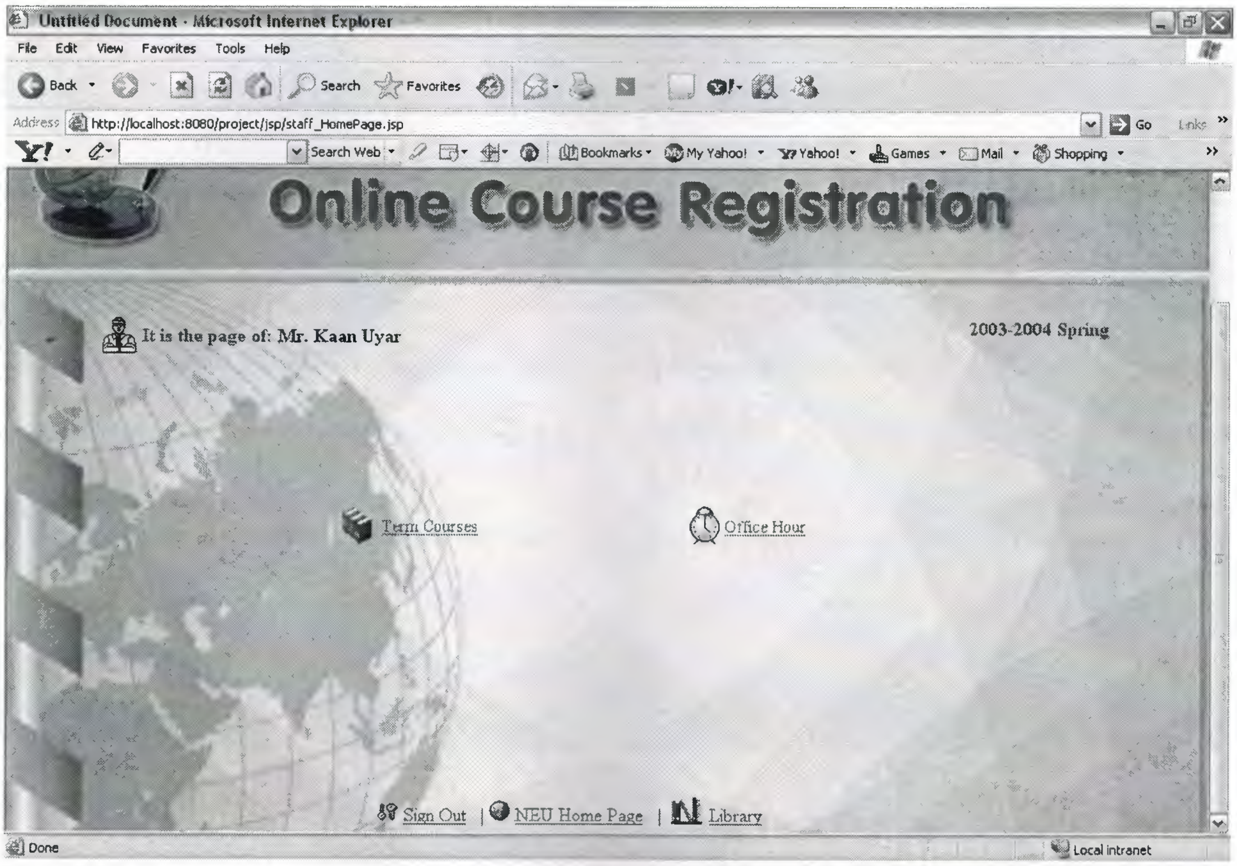
7- hii mohammad ( 2006-05-02)

1 2 3 >>

Term courses Information - Microsoft Internet Explorer
http://localhost:8080/project/jsp/stCourse_info.jsp
Sign Out | Home Page | NEU Home Page | Library
Local intranet

Figure 5.17: Requested Course Information

On the other, when the instructor login then the following page will be displayed:


Untitled Document - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/project/jsp/staff_HomePage.jsp
Go Links
Search Web Bookmarks My Yahoo! Yahoo! Games Mail Shopping
Done
Local intranet

Online Course Registration

It is the page of: Mr. Kaan Uyar

2003-2004 Spring

Term Courses

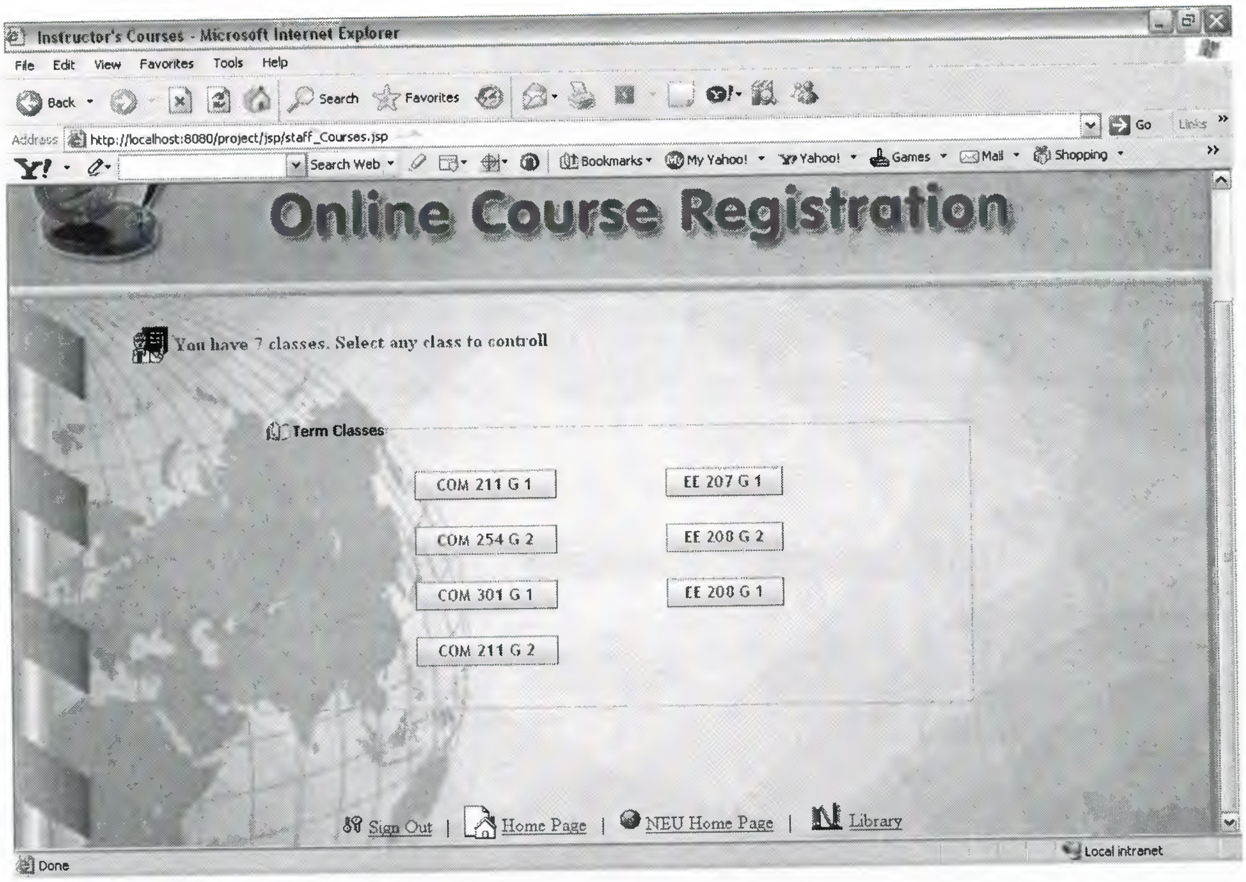
Office Hour

Sign Out | NEU Home Page | Library

Figure 5.18: Instructor Home Page.

As the page shows, staff home page contains tow links: Term Courses and Office Hours. Office hour's link, briefly, enables the staff to display and quickly change his office hours, so students well know when to *make a visit* to the instructor's office.

If it comes true of being instructors start forgetting what courses they are giving, then the second link would be helpful. As shown in figure 5.18, Term Courses link provides the instructor with the courses he is currently responsible about.


Instructor's Courses - Microsoft Internet Explorer
File Edit View Favorites Tools Help
http://localhost:8080/project/jsp/staff_Courses.jsp
Online Course Registration
Term Classes
COM 211 G 1
EE 207 G 1
COM 254 G 2
EE 208 G 2
COM 301 G 1
EE 208 G 1
COM 211 G 2
Sign Out | Home Page | NEU Home Page | Library
Done
Local intranet

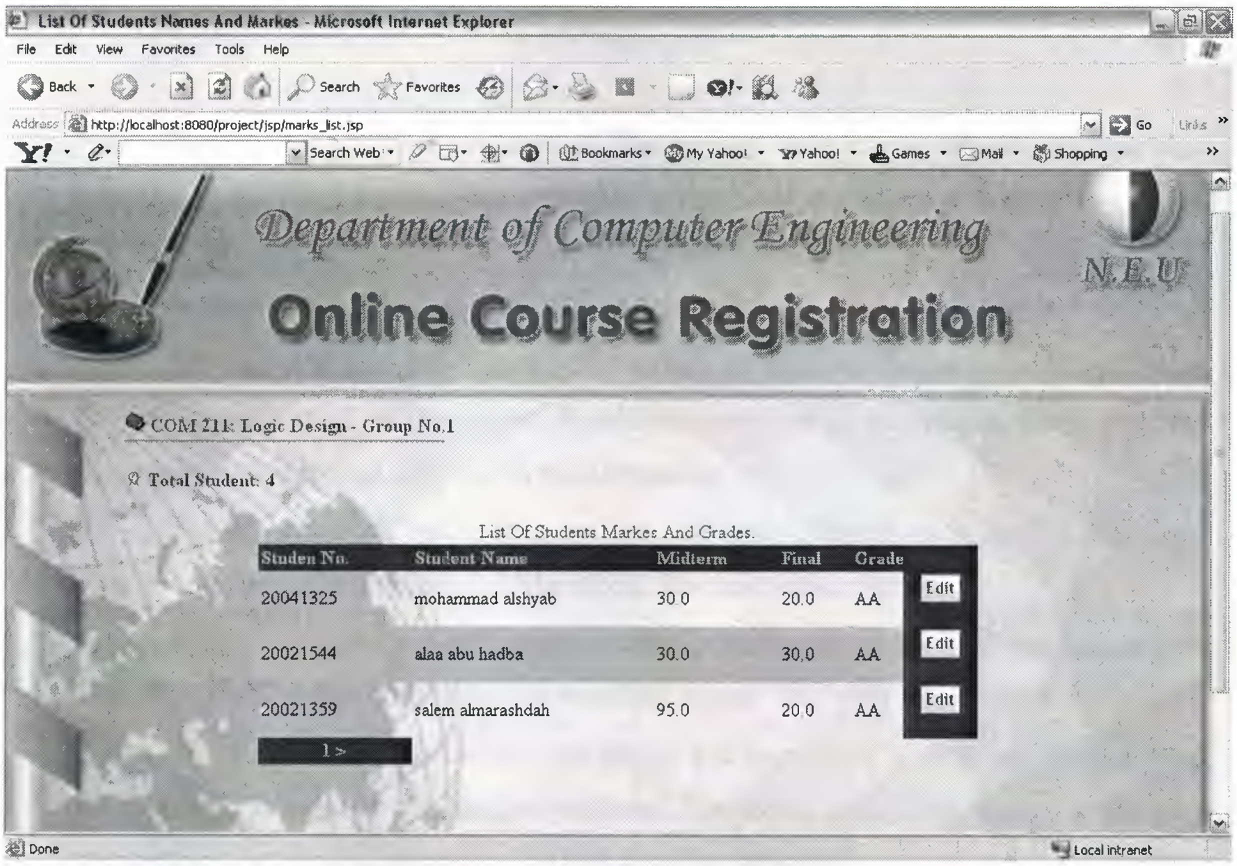
You have 7 classes. Select any class to controll

Figure 5.19: Instructor's Term Courses.

For each course, announcements can be distributed and student's list and marks be viewed and controlled.

Figure 5.19 shows course students list, every student's marks (midterm, final) and grade are modifiable and that is done by the provided page of figure 5.20.

Keep in mind that these changes directly affect on the students web pages which provide them with their marks, so from now on there is no need to distribute the marks list and announcements on any board, therefore to whom it may concern: you can take your boards off.



List Of Students Names And Markes - Microsoft Internet Explorer

http://localhost:8080/project/jsp/marks_list.jsp

Department of Computer Engineering

Online Course Registration

N.E.U

COM 211: Logic Design - Group No.1

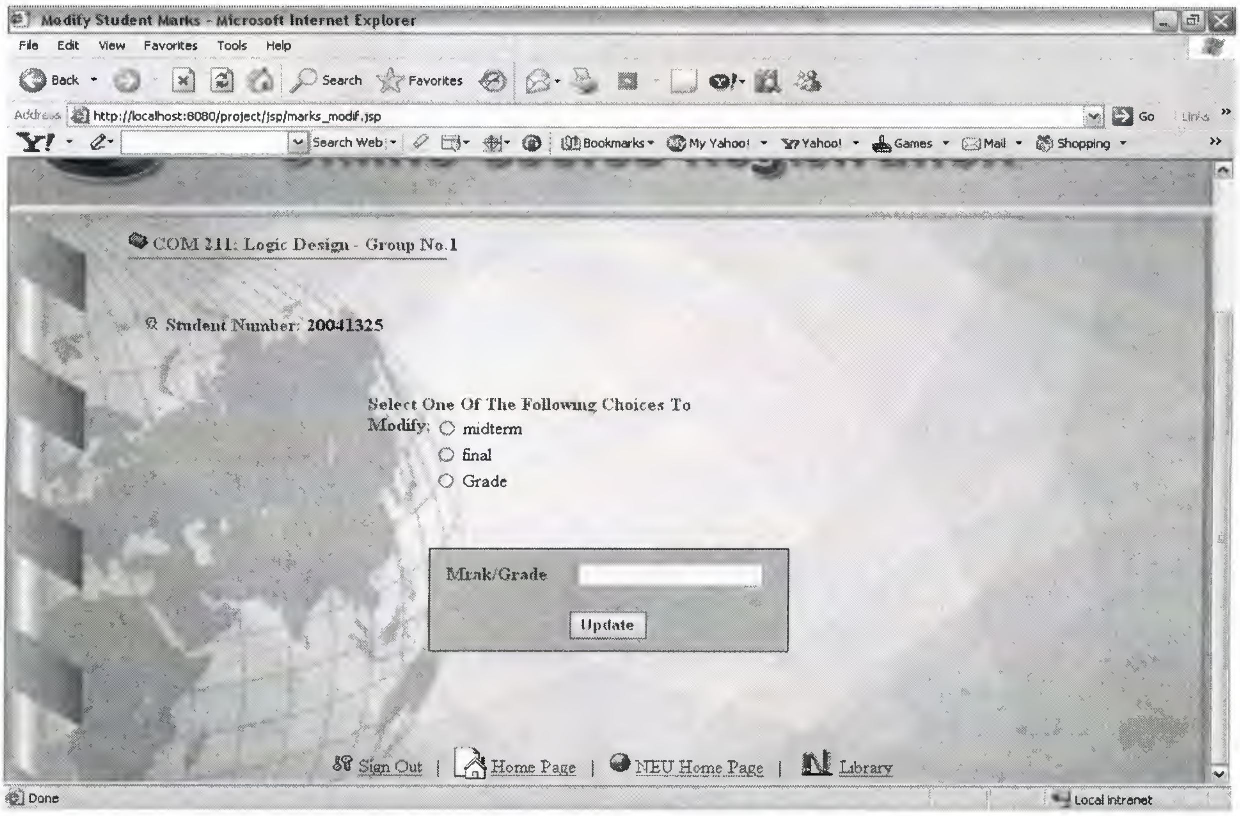
Total Student: 4

List Of Students Markes And Grades.

| Studen No. | Student Name | Midterm | Final | Grade | |
|---|---|---|---|---|---|
| 20041325 | mohammad alshyab | 30.0 | 20.0 | AA | Edit |
| 20021544 | alaa abu hadba | 30.0 | 30.0 | AA | Edit |
| 20021359 | salem almarashdah | 95.0 | 20.0 | AA | Edit |

1 >

Figure 5.20: Course Students List.



Modify Student Marks - Microsoft Internet Explorer

http://localhost:8080/project/jsp/marks_modif.jsp

COM 211: Logic Design - Group No.1

Student Number: 20041325

Select One Of The Following Choices To Modify:
- ○ midterm
- ○ final
- ○ Grade

Mrak/Grade

Update

Sign Out | Home Page | NEU Home Page | Library

Figure 5.21: Student Marks and Grade are Under Controlled

# CONCLUSION

During the point at the end of the project that using clever programming techniques and simple design, we can make a very well organized web page so that the user can be interact easily. There are many techniques for the web designing, that technique used is Java Server Page (JSP) and JavaScript.

The advantage of JSP file can contain text, HTML tags and scripts. Scripts in an JSP file are executed on the server. In addition, we can do to execute the connection with the server and send only report to the browser.

The advantage of JavaScript, we can call simple functions in head section and body section to make our web site like clock, date and animation, ect.

From the Database, ODBC can be used to access databases from our web pages. The ODBC Connection Object is used to create an open connection to a data source. Through this connection, we can access and manipulate a database. We can save the memory, that is we use simple database, that using many information to making connection. We come across the use of database in web site.

# REFERENCES

[1] ***Internet and World Wide web, How To program.*** Third edition, Deitel, H. M

[2] *Database System Using Oracle*, Second edition, Nilesh Shah

[3] ***Java How To Program,*** fifvth edition,H.M.Deitel

[4] **http://perseus.herts.ac.uk**

[5] **http://ei.cs.vt.edu/book/chap1/refs.html**

[6] **http://euler.vcsu.edu:7000/329/**

[7] **http://www.scit.wlv.ac.uk/~jphb/**

[8] **http://edocs.bea.com/wls/docs81/jsp/intro.html#49333**

[9] **http://directory.google.com/Top/Computers/Programming/ JavaScript**