



**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**PRIVATE TEACHING INSTITUTION SYSTEM  
USING JAVA X**

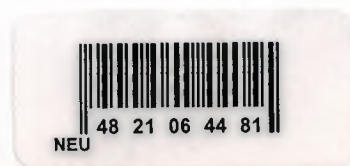
**GRADUATION PROJECT**

**COM-400**

**Student = Burak Mahir CİNKARATAŞ (20031230 )**

**Supervisor = Asst. Prof. Dr. Adil AMİRJANOV**

**Lefkoşa - 2007**



## ACKNOWLEDGMENT



First of all, I want to thank Dr. Adil Amirjanov. The semester before I graduate from NEU, He made me to shape my future on this sector by teaching the Object Oriented logic. He never tired of answering my questions. Whenever I knocked his door, he greeted me friendly and helped me to solve the points where I stucked. I succesfully overcome many problems under his guidance.

Thanks to all Engineering Department Academics to help me to get many experiences and knowledges about my future life also in Computer Science.

All in my life there is a person which I can not tell her helps on me by using the words in dictionaries. I wish from god to give her a long and peaceful life. Furthermore , I hope she will be proud of her son in her life evermore. Thanks Mum, for being and grown me up while facing many difficulties.

## **ABSTRACT**

In order to respond to technological needs of the companies, we have a huge duty as Computer Engineers. This software program maybe very useful if it will be developed.

While this system is being designed this system, the point I was thinking is to improve easily. So a few UML diagrams had been designed to help the other developers who are going to come after me.

This software will help to students how they have are experienced with the last exams. Also help the guide teachers how their students are levelling up or down till they have started to the institution.

This software may be improved with some funny and useful things . For example the exam results can be send to students e-mail or the their achivements can be shown by using curves.

## TABLE OF CONTENTS

<b>ACKNOWLEDGMENT</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>CONTENTS</b>	<b>iii</b>
<b>INTRODUCTION</b>	<b>1</b>
 <b>CHAPTER ONE : JAVA PROGRAMMING LANGUAGE</b>	
 <b>1.1 About Java Technology</b>	<b>2</b>
1.1.1 What is Java	2
1.1.2 Primary Goals of Java Technology	3
1.1.3 Where Java Can Be Run	3
1.1.4 A Difference – Garbage Collection	3
<b>1.2 Java Language Basics</b>	<b>4</b>
1.2.1 Java Variables	4
1.2.2 Exception Handling	5
1.2.3 Try – Catch and Finally Statement	5
<b>1.3 Object Oriented Programming</b>	<b>5</b>
1.3.1 What is an Object	6
1.3.2 What is a Class	6
1.3.3 What is Inheritance	6
1.3.4 What is Interface	7
<b>1.4 GUI Elements of Java</b>	<b>7</b>
1.4.1 Jbutton	7
1.4.2 JTextField	7
1.4.3 Jmenu	8

1.4.4 Jtable	9
1.4.5 Jframe	9

## **CHAPTER TWO : UML**

2.1 Overview	10
2.2 UML Diagrams	10
2.2.1 Use Case Diagrams	10
2.2.2 Class Diagrams	12
2.2.3 Object Diagrams	14
2.2.4 Sequence Diagrams	16
2.2.5 Collaboration Diagrams	18
2.2.6 Activity Diagrams	18

## **CHAPTER THREE : ORACLE DATABASE**

3.1 What is RDBMS	21
3.2 Overview	21
3.3 Background	22
3.4 What is SQL* Plus	22
3.5 DDL & DML	23
3.5.1 Data Definition Language	23
3.5.2 Data Manipulation Language	25
3.6 An Introduction to PL/SQL	27
3.6.1 Advantages of PL/SQL	27
3.6.2 Structure and An Example	28

## **CHAPTER FOUR : PRIVATE TEACHING INSTITUTION SYSTEM**

4.1 Database Design of System	30
4.2 UML Design of System	34
4.2.1 Use Case Diagram	34

4.2.2 Class Diagrams	35
4.2.3 Activity Diagram (Add Student)	37
4.3 Java Part Of System	38
4.3.1 Main Menu	39
4.3.2 Add Student	39
4.3.3 Add Exam Results	40
4.3.4 Show Exam Result	41
4.3.5 Add Teacher	42
4.3.6 Add Exam	43
CONCLUSION	47
REFERENCES	48
APPENDIX	49



## INTRODUCTION

In every developing countries technology has become the main part of the companies nowadays. As being a computer engineer, all we have to do is to answer this demand accordingly.

Every high school in Turkiye, has to enter an examination in order to study at a university. Of course it is a bad idea to test all the students' knowledge in 3 hours, but this is the way it is. In case the schools can not help students to gain advantage at this exam. Therefore, PRIVATE TEACHING INSTITUTIONS' has appeared everywhere in my country. Just like the other companies those institutions need some automation system.

All it is done in my project is to help the company to collect data about the students' examination results. And show these when needed, while that project is going to be improved by me. The first aim has to be transmitting these results simultaneously to the student via e-mail.

This book has been formed by 4 chapters.

The first chapter makes a clean and short explanation about the language used in the project which has great object oriented skills JAVA. The fundamentals and object concept is introduced to the reader.

The second Chapter talks about Relational Database Systems where gives some information about SQL \*Plus and PL/SQL.

At the third Chapter Unified Modeling language has briefly introduced to the reader. The diagrams and Object concept is dealed with a few pages.

The last Chapter talks about the Software Project about Private Teaching Institution System. At sub chapters it is deeply described how the database, domain and data access classes has been designed.

# **CHAPTER ONE**

## **JAVA PROGRAMMING LANGUAGE**

### **1.1 About Java Technology**

Java Technology has become a complete software ecosystem that represents different values to different types of consumer and business users. It offers developers a choice of three Java platform editions depending on the need:

- Java technology in small and mobile devices
- Java technology in PC desktops
- Java technology in medium to large businesses

#### **1.1.1 What is Java**

The Java platform is a fundamentally new way of computing, based on the power of networks and the idea that the same software should run on many different kinds of computers, consumer gadgets, and other devices. With Java technology, you can use the same application from any kind of machine -- a PC, a Macintosh computer, a network computer, or even new technologies like Internet screen phones.[1]

#### **1.1.2 Primary Goals Of Java Technology**

There were five primary goals in the creation of the Java language:

1. It should use the object-oriented programming methodology.
2. It should allow the same program to be executed on multiple operating systems.
3. It should contain built-in support for using computer networks.
4. It should be designed to execute code from remote sources securely.
5. It should be easy to use by selecting what were considered the good parts of other object-oriented languages.

To achieve the goals of networking support and remote code execution, Java programmers sometimes find it necessary to use extensions such as CORBA, Internet Communications Engine, or OSGi.



### **1.1.3 Where Java Can Be Run**

One characteristic, platform independence, means that programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere.

This is achieved by most Java compilers by compiling the Java language code "halfway" to bytecode (specifically Java bytecode)—simplified machine instructions specific to the Java platform. The code is then run on a virtual machine (VM), a program written in native code on the host hardware that interprets and executes generic Java bytecode. (In some JVM versions, bytecode can also be compiled to native code, resulting in faster execution.) Further, standardized libraries are provided to allow access to features of the host machines (such as graphics, threading and networking) in unified ways. Note that, although there is an explicit compiling stage, at some point, the Java bytecode is interpreted or converted to native machine instructions by the JIT compiler.[1]

### **1.1.4 A Difference – Garbage Collection**

One of the ideas behind Java's automatic memory management model is that programmers be spared the burden of having to perform manual memory management. In some languages the programmer allocates memory for the creation of objects stored on the heap and the responsibility of later deallocating that memory thus resides with the programmer. If the programmer forgets to deallocate memory or writes code that fails to do so, a memory leak occurs and the program can consume an arbitrarily large amount of memory. Additionally, if the program attempts to deallocate the region of memory more than once, the result is undefined and the program may become unstable and may crash. Finally, in non garbage collected environments, there is a certain degree of overhead and complexity of user-code to track and finalize allocations. Often developers may box themselves into certain designs to provide reasonable assurances that memory leaks will not occur.

In Java, this potential problem is avoided by automatic garbage collection. The programmer determines when objects are created, and the Java runtime is responsible for managing the object's lifecycle. The program or other objects can reference an object by holding a reference to it (which, from a low-level point of view, is its address on the heap). When no references to an object remain, the Java garbage collector automatically deletes the unreachable object, freeing memory and preventing a memory leak. Memory leaks may still occur if a programmer's code holds a reference to an object that is no longer needed—in other words, they can still occur but at higher conceptual levels.[3]

## 1.2 Java Language Basics

Here are some Java syntax which is commonly used in Java.

### 1.2.1 Java Variables

The Java programming language defines the following kinds of variables:

#### a. Instance Variables (Non-Static Fields)

Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword. Non-static fields are also known as instance variables because their values are unique to each instance of a class (to each object, in other words); the `currentSpeed` of one bicycle is independent from the `currentSpeed` of another.

#### b. Class Variables (Static Fields)

A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated. A field defining the number of gears for a particular kind of bicycle could be marked as static since conceptually the same number of gears will apply to all instances. The code `static int numGears = 6;` would create such a static field. Additionally, the keyword `final` could be added to indicate that the number of gears will never change.

#### c. Local Variables

Similar to how an object stores its state in fields, a method will often store its temporary state in *local variables*. The syntax for declaring a local variable is similar to

declaring a field (for example, `int count = 0;`). There is no special keyword designating a variable as local; that determination comes entirely from the location in which the variable is declared — which is between the opening and closing braces of a method. As such, local variables are only visible to the methods in which they are declared; they are not accessible from the rest of the class.

### **1.2.2 Exception Handling**

An exception is a class that descends from either `java.lang.Exception` or `java.lang.RuntimeException` that defines mild error conditions your program might encounter. Rather than letting the program terminate, you can write code to handle exceptions and continue program execution.

### **1.2.3 Try – Catch and Finally Statement**

Java has a robust, but complicated, exception handling framework. By using a try block, a developer can wrap a suspect block of code. If the code throws an exception, the accompanying catch block allows the developer to process, or handle, the exception. Handling the exception often involves logging it and then determining if the application can continue or if the application should exit.

An optional finally block can follow the catch block. This block of code gives the developer a chance to always run a bit of code regardless of whether an exception was thrown in the try block. finally blocks are often used for cleanup of database connections and other resources. [4]

## **1.3 Object Oriented Programming**

Object-oriented programming (OOP) is a programming paradigm that uses "objects" to design applications and computer programs. It utilizes several techniques from previously established paradigms, including inheritance, modularity, polymorphism, and encapsulation. Even though it originated in the 1960s, OOP was not commonly used in mainstream software application development until the 1990s. Today, many popular programming languages support OOP. [5]



### 1.3.1 What is an Object

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner. Here is two objects of Bicycle Class instantiates bicycles.

```
Bicycle bike1 = new Bicycle();  
Bicycle bike2 = new Bicycle();
```

### 1.3.2 What is a Class

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior. Here is a Bicycle Class with variables and methods shown with.

```
class Bicycle {  
    int cadence = 0;  
    int speed = 0;  
    int gear = 1;  
    void changeCadence(int newValue) {  
        cadence = newValue;  
    }  
    void changeGear(int newValue) {  
        gear = newValue;  
    }  
    void speedUp(int increment) {  
        speed = speed + increment;  
    }  
    void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
    void printStates() {  
        System.out.println("cadence:"+cadence+" speed:"+speed+"  
        gear:"+gear);  
    }  
}
```

### 1.3.3 What is Inheritance

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the

simple syntax provided by the Java programming language. Here a Mountain Bike inherits from Bicycle class.

```
class MountainBike extends Bicycle {  
    // new fields and methods defining a mountain bike would go here  
}
```

### 1.3.4 What is Interface

An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it. [6]

## 1.4 GUI Elements of Java

A graphical user interface (GUI, often pronounced *gooey* or *goo-ee*) is a type of user interface which allows people to interact with a computer and computercontrolled devices which employ graphical icons, visual indicators or special graphical elements called "widgets", along with text labels or text navigation to represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements.

### 1.4.1 JButton

Simple uses of JButton are very similar to Button. You create a JButton with a String as a label, and then drop it in a window. Events are normally handled just as with a Button: you attach an ActionListener via the addActionListener method.

Usage: **JButton button1 = new JButton("Java");**

### 1.4.2 JTextField

JTextField is a lightweight component that allows the editing of a single line of text.

Usage for Input: **Declare** a JTextField as an instance variable. Reason: If it's an instance variable, it can be seen in all methods in the class.

1. Assign an initial value to this variable by calling the JTextField constructor. Specify the approximate field width in the constructor.

Example: `JTextField yourInputFieldt = new JTextField(16);`

2. **Add** the text field to a container.

`content.add(yourInputField);` or to add it to a `JPanel p`

`p.add(yourInputField);`

3. **Input** is done by calling the `getText()`.

4. Get the string in the text field by calling `yourTextField.getText()` method

whenever you need it. This is probably the most common way.

`String x = yourInputField.getText();`

5. Attach an action listener to the text field. It is called whenever the user types

Enter in that field. The listener can then get the text and process it.

Usage For Output : Using a `JTextField` for output is almost the same as for input, but . . .

1. Set the text field with `yourTextField.setText(someString)`

2. If it's only for output, call `.setEditable(false)` so the user can't change the field.

Here is the sequence.

1. **Declare and initialize a `JTextField`** as a field variable (instance variable).

Example:

`JTextField myOutput = new JTextField(16);`

You can also set the initial value in the field

`JTextField myOutput = new JTextField("someInitialValue", 20);`

2. **Add the text field to a container.** For example, to add it to `JPanel p`.

`p.add(myOutput);`

3. **Setting the value of the text field.** Whenever you want put a string value in the text field, call `myOutput.setText("Some text")`.

`myOutput.setText("some text");`

### 1.4.3 JMenu

An implementation of a menu -- a popup window containing `JMenuItems` that is displayed when the user selects an item on the `JMenuBar`. In addition to `JMenuItems`, a `JMenu` can also contain `JSeparators`.

In essence, a menu is a button with an associated `JPopupMenu`. When the "button" is pressed, the `JPopupMenu` appears. If the "button" is on the `JMenuBar`, the



menu is a top-level window. If the "button" is another menu item, then the JPopupMenu is "pullright" menu.

#### 1.4.4 JTable

The JTable is used to display and edit regular two-dimensional tables of cells. The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

For example, to set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel() {
    public int getColumnCount() { return 10; }
    public int getRowCount() { return 10; }
    public Object getValueAt(int row, int col) { return new Integer(row*col); }
};
JTable table = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(table);
```

#### 1.4.5 JFrame

The JFrame class is slightly incompatible with Frame. Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. An example with a JFrame and 3 JButtons. [7]

```
import java.awt.*;
import javax.swing.*;
public class JFrameExample {
    public static void main(String[] args) {
        WindowUtilities.setNativeLookAndFeel();
        JFrame f = new JFrame("This is a test");
        f.setSize(400, 150);
        Container content = f.getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
        content.add(new JButton("Button 1"));
        content.add(new JButton("Button 2"));
        content.add(new JButton("Button 3"));
        f.addWindowListener(new ExitListener());
        f.setVisible(true);
    }
}
```

## CHAPTER TWO

### UML

#### 2.1 Overview

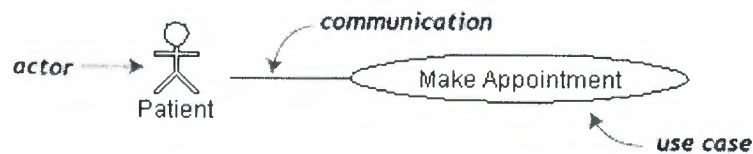
Large enterprise applications - the ones that execute core business applications, and keep a company going - must be more than just a bunch of code modules. They must be structured in a way that enables scalability, security, and robust execution under stressful conditions, and their structure - frequently referred to as their *architecture* - must be defined clearly enough that maintenance programmers can (quickly!) find and fix a bug that shows up long after the original authors have moved on to other projects. That is, these programs must be *designed* to work perfectly in many areas, and business functionality is not the only one (although it certainly is the essential core). Of course a well-designed architecture benefits any program, and not just the largest ones as we've singled out here. We mentioned large applications first because structure is a way of dealing with complexity, so the benefits of structure (and of modeling and design, as we'll demonstrate) compound as application size grows large. Another benefit of structure is that it enables *code reuse*: Design time is the easiest time to structure an application as a collection of self-contained modules or components. Eventually, enterprises build up a library of models of components, each one representing an implementation stored in a library of code modules. When another application needs the same functionality, the designer can quickly import its module from the library. At coding time, the developer can just as quickly import the code module into the application.

##### 2.2.1 Use Case Diagrams

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how*.

Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system. Here is a scenario for a medical clinic(see figure 2.1).

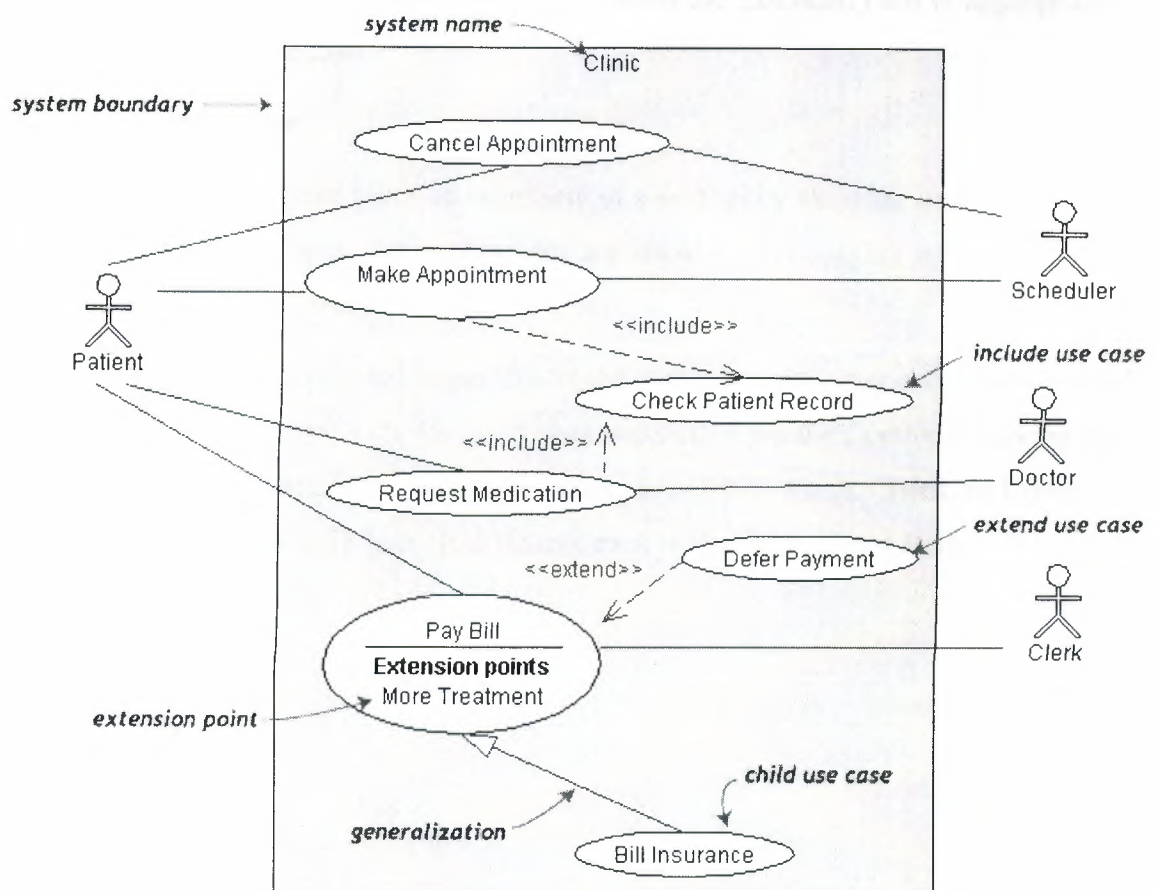
"A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. "



(fig 2.1 A Use Case Diagram)

Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. We've put Make Appointment as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.



(fig 2.2 A Use Case Diagram with system boundary)

A system boundary rectangle separates the clinic system from the external actors(see figure 2.2).

A use case generalization shows that one use case is simply a special kind of another. Pay Bill is a parent use case and Bill Insurance is the child. A child can be substituted for its parent whenever necessary. Generalization appears as a line with a triangular arrow head toward the parent use case.

Include relationships factor use cases into additional ones. Includes are especially helpful when the same use case can be factored out of two different use cases. Both Make Appointment and Request Medication include Check Patient Record as a subtask. In the diagram, include notation is a dotted line beginning at base use case ending with an arrows pointing to the include use case. The dotted line is labeled <<include>>.

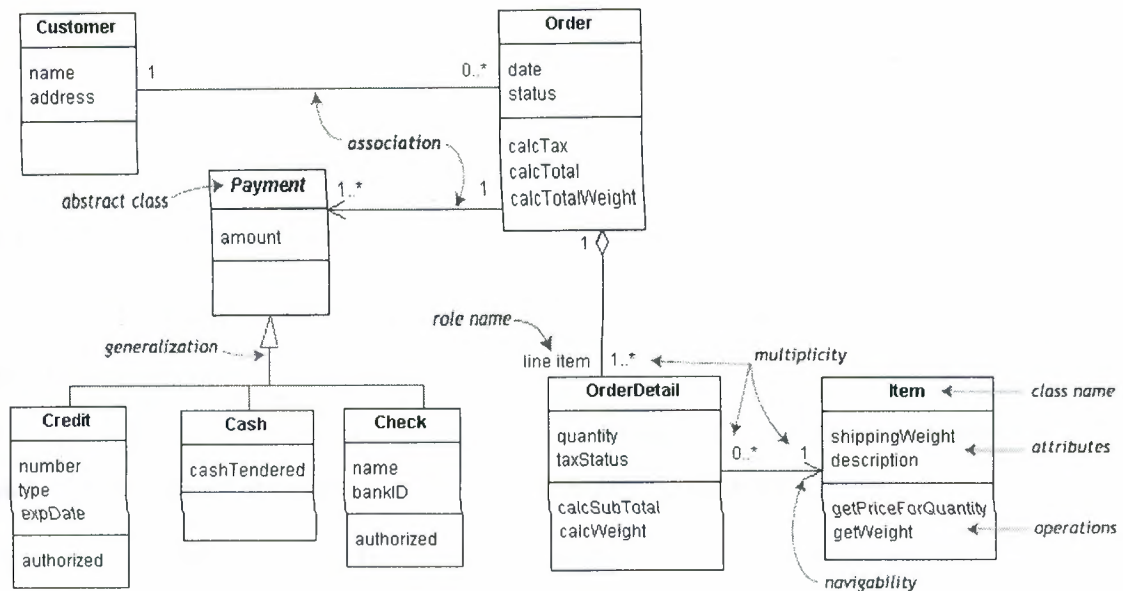
An extend relationship indicates that one use case is a variation of another. Extend notation is a dotted line, labeled <<extend>>, and with an arrow toward the base case. The extension point, which determines when the extended case is appropriate, is written inside the base case.

### **2.2.2 Class Diagrams**

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

The class diagram below(see figure 2.3) models a customer order from a retail catalog. The central class is the Order. Associated with it are the Customer making the purchase and the Payment. A Payment is one of three kinds: Cash, Check, or Credit. The order contains OrderDetails (line items), each with its associated Item.





(fig 2.3 A Class Diagram)

UML class notation is a rectangle divided into three parts: class name, attributes, and operations. Names of abstract classes, such as *Payment*, are in italics. Relationships between classes are the connecting links.

Our class diagram has three kinds of relationships.

- **association** -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.
- **aggregation** -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole. In our diagram, Order has a collection of OrderDetails.
- **generalization** -- an inheritance link indicating one class is a superclass of the other. A generalization has a triangle pointing to the superclass. *Payment* is a superclass of Cash, Check, and Credit.

An association has two ends. An end may have a **role name** to clarify the nature of the association. For example, an **OrderDetail** is a line item of each **Order**.

A **navigability** arrow on an association shows which direction the association can be traversed or queried. An **OrderDetail** can be queried about its **Item**, but not the other way around. The arrow also lets you know who "owns" the association's implementation; in this case, **OrderDetail** has an **Item**. Associations with no navigability arrows are bi-directional.

The **multiplicity** of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers. In our example, there can be only one **Customer** for each **Order**, but a **Customer** can have any number of **Orders**.

This table gives the most common multiplicities(see Table 2.1).

Multiplicities	Meaning
0..1	zero or one instance. The notation <i>n . . m</i> indicates <i>n</i> to <i>m</i> instances.
0..* or *	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

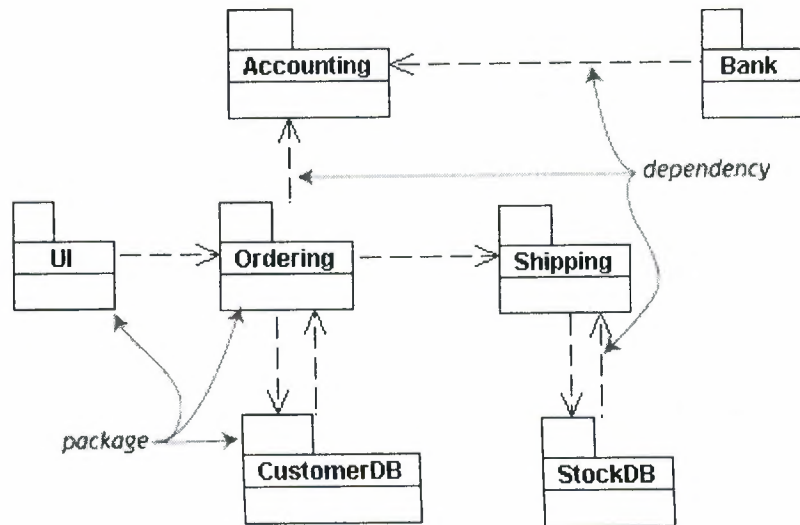
(Table 2.1 Possible Multiplicities)

Every class diagram has classes, associations, and multiplicities. Navigability and roles are optional items placed in a diagram to provide clarity.

### 2.2.3 Object Diagrams

To simplify complex class diagrams, you can group classes into **packages**. A package is a collection of logically related UML elements. The diagram below(see figure 2.4) is a business model in which the classes are grouped into packages.



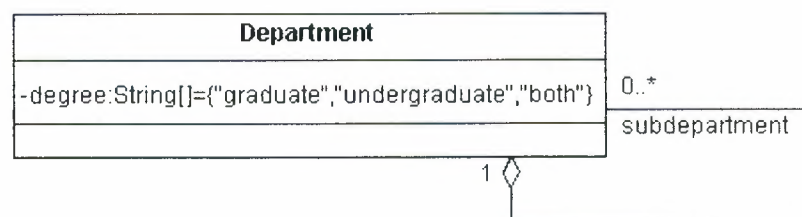


(fig 2.4 An Object Diagram)

Packages appear as rectangles with small tabs at the top. The package name is on the tab or inside the rectangle. The dotted arrows are **dependencies**. One package depends on another if changes in the other could possibly force changes in the first.

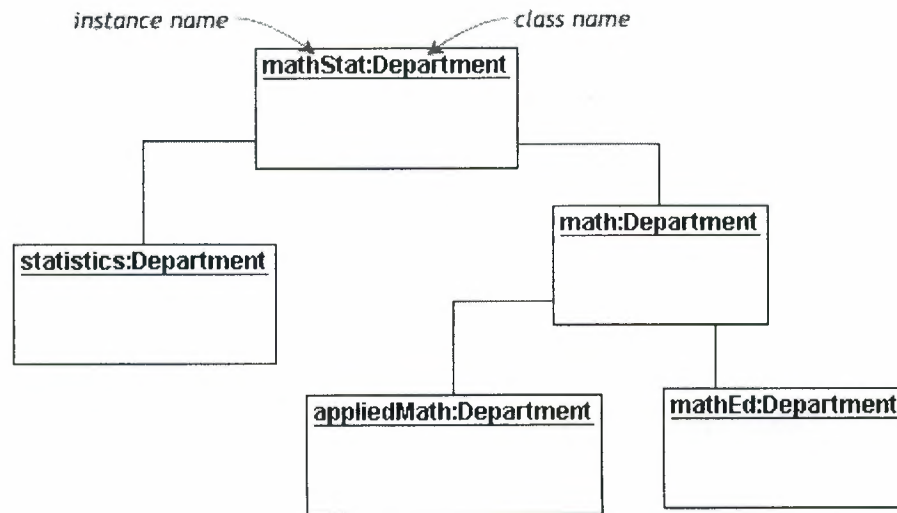
**Object diagrams** show instances instead of classes. They are useful for explaining small pieces with complicated relationships, especially recursive relationships.

This small class diagram(see figure 2.5) shows that a university **Department** can contain lots of other **Departments**.



(fig 2.5 A Class Diagram with Multiplicity)

The object diagram below(see figure 2.6) instantiates the class diagram, replacing it by a concrete example.



(fig 2.6 Objec Diagram)

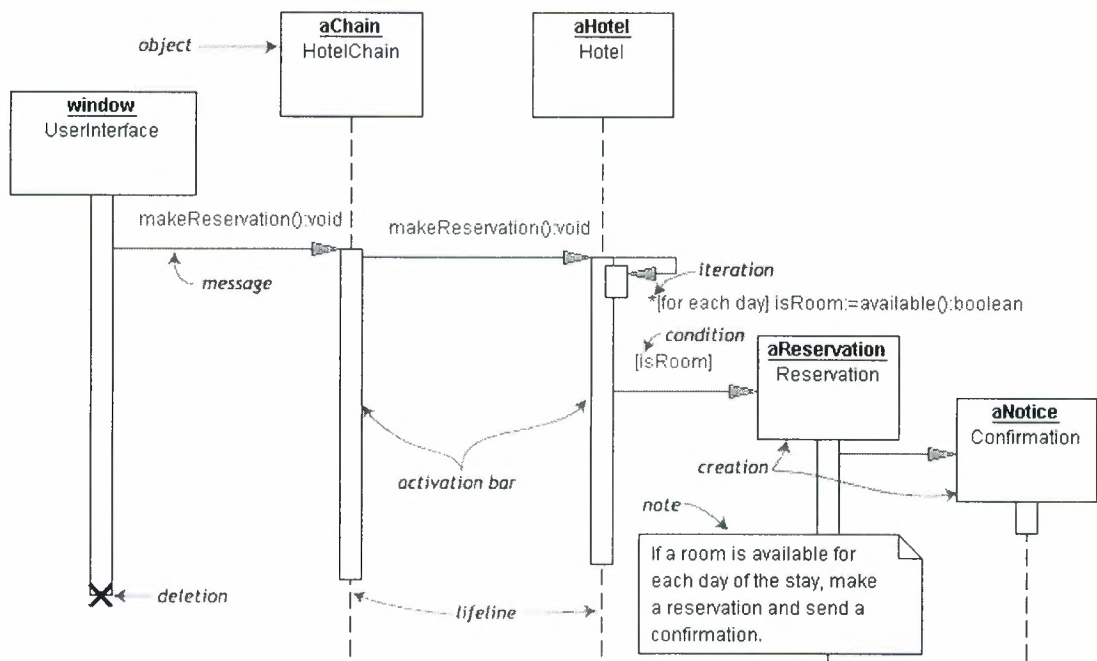
Each rectangle in the object diagram corresponds to a single instance. Instance names are underlined in UML diagrams. Class or instance names may be omitted from object diagrams as long as the diagram meaning is still clear.

## 2.2.4 Sequence Diagrams

Class and object diagrams are static model views. **Interaction diagrams** are dynamic. They describe how objects collaborate.

A **sequence diagram** is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram(see figure 2.7) for making a hotel reservation. The object initiating the sequence of messages is a **Reservation window**.



(fig 2.7 A sequence Diagram)

The **Reservation window** sends a `makeReservation()` message to a **HotelChain**. The **HotelChain** then sends a `makeReservation()` message to a **Hotel**. If the **Hotel** has available rooms, then it makes a **Reservation** and a **Confirmation**.

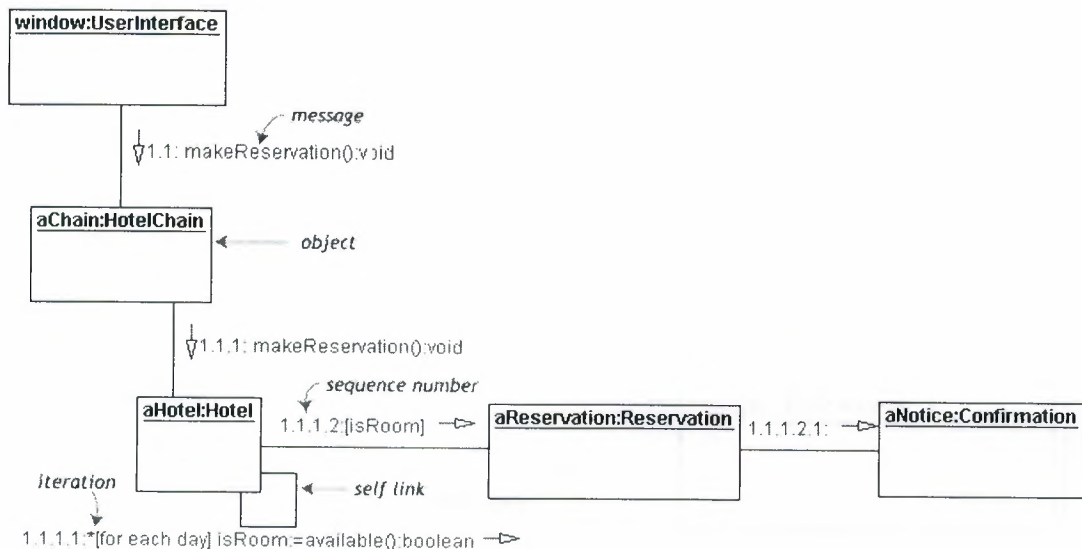
Each vertical dotted line is a **lifeline**, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the **activation bar** of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In our diagram, the **Hotel** issues a **self call** to determine if a room is available. If so, then the **Hotel** creates a **Reservation** and a **Confirmation**. The asterisk on the self call means **iteration** (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, `[ ]`, is a **condition**.

The diagram has a clarifying **note**, which is text inside a dog-eared rectangle. Notes can be put into any kind of UML diagram.

## 2.2.5 Collaboration Diagrams

**Collaboration diagrams** are also interaction diagrams. They convey the same information as sequence diagrams, but they focus on object roles instead of the times that messages are sent. In a sequence diagram, object roles are the vertices and messages are the connecting links.



(fig 2.8 A Collaboration Diagram)

The object-role rectangles are labeled with either class or object names (or both). Class names are preceded by colons ( : ).

Each message in a collaboration diagram has a **sequence number**. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

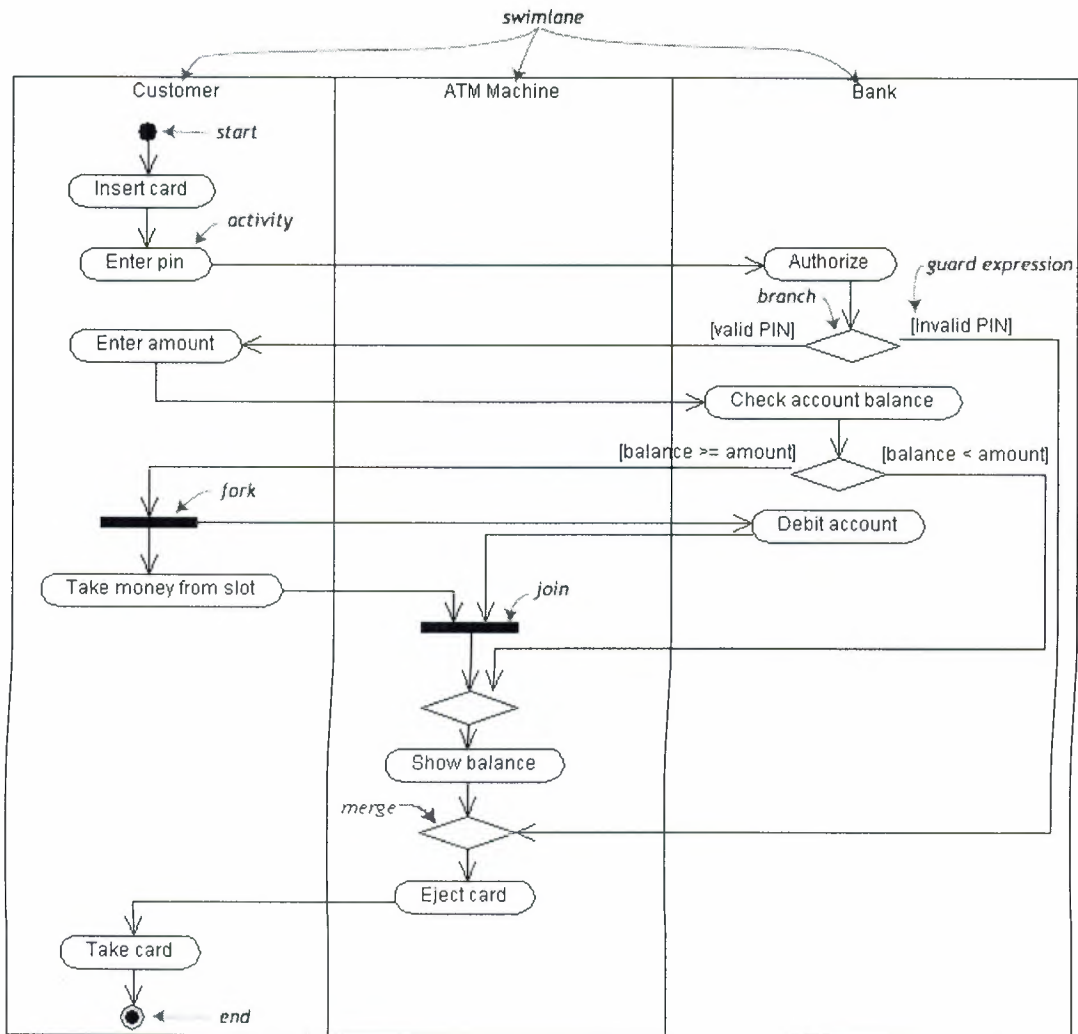
## 2.2.6 Activity Diagrams

An **activity diagram** is essentially a fancy flowchart. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

For our example(see figure 2.9), we used the following process.

"Withdraw money from a bank account through an ATM."

The three involved classes (people, etc.) of the activity are **Customer**, **ATM**, and **Bank**. The process begins at the black start circle at the top and ends at the concentric white/black stop circles at the bottom. The activities are rounded rectangles.



(fig 2.9 An Activity Diagram)

Activity diagrams can be divided into object **swimlanes** that determine which object is responsible for which activity. A single **transition** comes out of each activity, connecting it to the next activity.

A transition may **branch** into two or more mutually exclusive transitions. **Guard expressions** (inside [ ]) label the transitions coming out of a branch. A branch and its subsequent **merge** marking the end of the branch appear in the diagram as hollow diamonds.



A transition may **fork** into two or more parallel activities. The fork and the subsequent **join** of the threads coming out of the fork appear in the diagram as solid bars. [8]



## CHAPTER THREE

### ORACLE DATABASE

In computing, a **database** can be defined as a structured collection of records or data that is stored in a computer so that a program can consult it to answer queries. The records retrieved in answer to queries become information that can be used to make decisions. The computer program used to manage and query a database is known as a database management system (DBMS). The properties and design of database systems are included in the study of information science.

#### 3.1 What is RDBMS

RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields.

RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use SQL as database query language.

Edgar Codd introduced the relational database model. Many modern DBMS do not conform to the Codd's definition of a RDBMS, but nonetheless they are still considered to be RDBMS.

The most popular RDBMS are MS SQL Server, DB2, Oracle and MySQL.

#### 3.2 Overview

An Oracle **database** is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a **server** reliably manages

a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

### **3.3 Background**

Oracle is a relational database management system developed by Oracle Corporation.

The Oracle RDBMS is used in many database applications on several OS platforms, including Unix and Windows. Oracle database was initially developed by Larry Ellison, Bob Miner and Ed Oates. Their company Software Development Laboratories was founded in 1977 (renamed to Relational Software, Inc. in 1979). In 1983 the company was renamed to Oracle Corporation.

Oracle database was the first database complying with the ANSI SQL standard.

### **3.4 What is SQL\* Plus?**

SQL\*Plus is an interactive and batch query tool that is installed with every Oracle Database Server or Client installation. It has a command-line user interface, a Windows Graphical User Interface (GUI) and the iSQL\*Plus web-based user interface.

SQL\*Plus has its own commands and environment, and it provides access to the Oracle Database. It enables you to enter and execute SQL, PL/SQL, SQL\*Plus and operating system commands to perform the following:

- Format, perform calculations on, store, and print from query results
- Examine table and object definitions
- Develop and run batch scripts
- Perform database administration

We can use SQL\*Plus to generate reports interactively, to generate reports as batch processes, and to output the results to text file, to screen, or to HTML file for browsing on the Internet. You can generate reports dynamically using the HTML output facility of SQL\*Plus, or using the dynamic reporting capability of iSQL\*Plus to run a script from a web page.[9]

### 3.5 DDL & DML

SQL statements are divided into two major categories: **data definition language (DDL)** and **data manipulation language (DML)**. Both of these categories contain far more statements than we can present here, and each of the statements is far more complex than we show in this introduction.

#### 3.5.1 Data Definition Language

DDL statements are used to build and modify the structure of your tables and other objects in the database. When you execute a DDL statement, it takes effect immediately.

- The create table statement does exactly that:

```
CREATE TABLE <table name> (  
  <attribute name 1> <data type 1>,  
  ...  
  <attribute name n> <data type n>);
```

The **data types** that you will use most frequently are character strings, which might be called VARCHAR or CHAR for variable or fixed length strings; numeric types such as NUMBER or INTEGER, which will usually specify a precision; and DATE or related types. Data type syntax is variable from system to system; the only way to be sure is to consult the documentation for your own software.

- The alter table statement may be used as you have seen to specify primary and foreign key constraints, as well as to make other modifications to the table structure. Key constraints may also be specified in the CREATE TABLE statement.

```
ALTER TABLE <table name>
```

```
ADD CONSTRAINT <constraint name> PRIMARY KEY (<attribute list>);
```

You get to specify the constraint name. Get used to following a convention of tablename\_pk (for example, Customers\_pk), so you can remember what you did later. The attribute list contains the one or more attributes that form this PK; if more than one, the names are separated by commas.

- **The alter table statement may be used as you have seen to specify primary and foreign key constraints, as well as to make other modifications to the table structure. Key constraints may also be specified in the CREATE TABLE statement.**

```
ALTER TABLE <table name>  
ADD CONSTRAINT <constraint name> PRIMARY KEY (<attribute list>);
```

- The foreign key constraint is a bit more complicated, since we have to specify both the FK attributes in this (child) table, and the PK attributes that they link to in the parent table.

```
ALTER TABLE <table name>  
ADD CONSTRAINT <constraint name> FOREIGN KEY (<attribute list>)  
REFERENCES <parent table name> (<attribute list>);
```

Name the constraint in the form childtable\_parenttable\_fk (for example, Orders\_Customers\_fk). If there is more than one attribute in the FK, all of them must be included (with commas between) in both the FK attribute list and the REFERENCES (parent table) attribute list.

You need a separate foreign key definition for each relationship in which this table is the child.

You get to specify the constraint name. Get used to following a convention of tablename\_pk (for example, Customers\_pk), so you can remember what you did later. The attribute list contains the one or more attributes that form this PK; if more than one, the names are separated by commas.



- If you totally mess things up and want to start over, you can always get rid of any object you've created with a drop statement. The syntax is different for tables and constraints.

```
DROP TABLE <table name>;  
  
ALTER TABLE <table name>  
DROP CONSTRAINT <constraint name>;
```

This is where consistent constraint naming comes in handy, so you can just remember the PK or FK name rather than remembering the syntax for looking up the names in another table. The DROP TABLE statement gets rid of its own PK constraint, but won't work until you separately drop any FK constraints (or child tables) that refer to this one. It also gets rid of all data that was contained in the table—and it doesn't even ask you if you really want to do this!

- All of the information about objects in your schema is contained, not surprisingly, in a set of tables that is called the **data dictionary**. There are hundreds of these tables most database systems, but all of them will allow you to see information about your own tables, in many cases with a graphical interface. How you do this is entirely system-dependent.

### 3.5.2 Data Manipulation Language

DML statements are used to work with the data in tables. When you are connected to most multi-user databases (whether in a client program or by a connection from a Web page script), you are in effect working with a private copy of your tables that can't be seen by anyone else until you are finished (or tell the system that you are finished). You have already seen the SELECT statement; it is considered to be part of DML even though it just retrieves data rather than modifying it.

- The **insert** statement is used, obviously, to add new rows to a table.

```
INSERT INTO <table name>  
VALUES (<value 1>, ... <value n>);
```

The comma-delimited list of values must match the table structure exactly in the number of attributes and the data type of each attribute. Character type values are always enclosed in single quotes; number values are never in quotes; date values are often (but not always) in the format 'yyyy-mm-dd' (for example, '2006-11-30').

Yes, you will need a separate INSERT statement for every row.

- The update statement is used to change values that are already in a table.

```
UPDATE <table name>  
SET <attribute> = <expression>  
WHERE <condition>;
```

The update expression can be a constant, any computed value, or even the result of a SELECT statement that returns a single row and a single column. If the WHERE clause is omitted, then the specified attribute is set to the same value in every row of the table (which is usually not what you want to do). You can also set multiple attribute values at the same time with a comma-delimited list of attribute=expression pairs.

- The **delete** statement does just that, for rows in a table.

```
DELETE FROM <table name>  
WHERE <condition>;
```

If the WHERE clause is omitted, then every row of the table is deleted (which again is usually not what you want to do)—and again, you will not get a “do you really want to do this?” message.

- If you are using a large multi-user system, you may need to make your DML changes visible to the rest of the users of the database. Although this might be done automatically when you log out, you could also just type:

```
COMMIT;
```

Which is done in Oracle Library like : **setAutoCommit(true);** or **setAutoCommit(False);**

- If you’ve messed up your changes in this type of system, and want to restore your private copy of the database to the way it was before you started (this only works if you haven’t already typed COMMIT), just type:

```
ROLLBACK;
```

Although single-user systems don’t support **commit** and **rollback** statements, they are used in large systems to control **transactions**, which are sequences of changes to the database. Transactions are frequently covered in more advanced courses. [10]



### 3.6 Introduction to PL/SQL

**PL/SQL (Procedural Language/Structured Query Language)** is Oracle Corporation's proprietary server-based procedural extension to the SQL database language. (Some other SQL database management systems offer languages similar to PL/SQL.) Its syntax strongly resembles that of Ada.

#### 3.6.1 Advantages Of PL/SQL

PL/SQL is a completely portable, high-performance transaction processing language that offers the following advantages:

- Support for SQL
- Support for object-oriented programming
- Better performance
- Higher productivity
- Full portability
- Tight integration with Oracle
- *Tight security*

As we see above PL/SQL has many advantages to programmers' but we only need to describe the advantages to Support for SQL and Support Object Oriented Programming parts.

##### **a. Support for SQL**

SQL has become the standard database language because it is flexible, powerful, and easy to learn. A few English-like commands such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` make it easy to manipulate the data stored in a relational database.

PL/SQL lets you use all the SQL data manipulation, cursor control, and transaction control commands, as well as all the SQL functions, operators, and pseudocolumns. This extensive SQL support lets you manipulate Oracle data flexibly and safely. Also, PL/SQL fully supports SQL datatypes, reducing the need to convert data passed between your applications and the database.

PL/SQL also supports dynamic SQL, a programming technique that makes your applications more flexible and versatile. Your programs can build and process SQL data definition, data control, and session control statements at run time, without knowing details such as table names and WHERE clauses in advance.

### **b. Support for Object Oriented Programming**

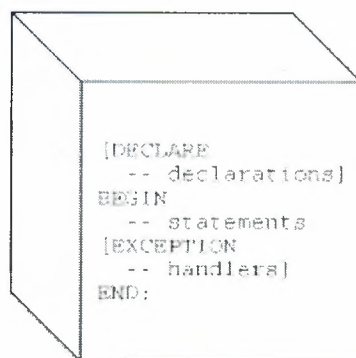
Object types are an ideal object-oriented modeling tool, which you can use to reduce the cost and time required to build complex applications. Besides allowing you to create software components that are modular, maintainable, and reusable, object types allow different teams of programmers to develop software components concurrently.

By encapsulating operations with data, object types let you move data-maintenance code out of SQL scripts and PL/SQL blocks into methods. Also, object types hide implementation details, so that you can change the details without affecting client programs.

In addition, object types allow for realistic data modeling. Complex real-world entities and relationships map directly into object types. This direct mapping helps your programs better reflect the world they are trying to simulate.

### **3.6.2 Structure and An Example**

This illustration below(see figure 3.1) is described in here. It shows the structure of a typical PL/SQL block, with an optional DECLARE section, a mandatory BEGIN-END block, and an optional EXCEPTION section right before the END statement of the main block.



(fig 3.1 Oracle Structure)

Declaring variables here as follow :

```
part_no NUMBER(4);
```

```
in_stock BOOLEAN;
```

Assigning values to variables as follow :

```
tax := price * tax_rate;
```

```
valid_id := FALSE;
```

Here is a full PL/SQL example, which processes a bank transaction. Before allowing you to withdraw \$500 from account 3, it makes sure the account has sufficient funds to cover the withdrawal. If the funds are available, the program debits the account. Otherwise, the program inserts a record into an audit table.

```
DECLARE
    acct_balance NUMBER(11,2);
    acct          CONSTANT NUMBER(4) := 3;
    debit_amt     CONSTANT NUMBER(5,2) := 500.00;
BEGIN
    SELECT bal INTO acct_balance FROM accounts
    WHERE account_id = acct
    FOR UPDATE OF bal;
    IF acct_balance >= debit_amt THEN
        UPDATE accounts SET bal = bal - debit_amt
        WHERE account_id = acct;
    ELSE
        INSERT INTO temp VALUES
            (acct, acct_balance, 'Insufficient funds');
        -- insert account, current balance, and message
    END IF;
    COMMIT;
END; [11]
```

## CHAPTER FOUR

### PRIVATE TEACHING INSTITUTION SYSTEM

#### 4.1 Database Design of The System

The system has 6 tables. The creation and the created versions are shown, below(see figure 4.1 , 4.2, 4.3 , 4.4 , 4.5 , 4.6 ).

```
CREATE TABLE "TSTUDENT"
(
    "STTCID" VARCHAR2(11),
    "STID" VARCHAR2(8),
    "STNAME" VARCHAR2(20),
    "STSURNAME" VARCHAR2(20),
    "STCITY" VARCHAR2(20),
    "STTOWN" VARCHAR2(20),
    "STZIP" VARCHAR2(5),
    "STADDRESS" VARCHAR2(50),
    "STBIRTHDATE" DATE,
    "STGRADSCHOOL" VARCHAR2(30),
    "STGRADTYPE" VARCHAR2(10),
    "STCGPA" VARCHAR2(4),
    "STPHONE" VARCHAR2(11),
    "STVNAME" VARCHAR2(20),
    "STVSURNAME" VARCHAR2(20),
    "STVPHONE" VARCHAR2(11),
    "STEMAIL" VARCHAR2(20),
    PRIMARY KEY ("STTCID") ENABLE
)
```

Column Name	Data Type	Nullable	Default	Primary Key
STTCID	VARCHAR2(11)	No	-	1
STID	VARCHAR2(8)	Yes	-	-
STNAME	VARCHAR2(20)	Yes	-	-
STSURNAME	VARCHAR2(20)	Yes	-	-
STCITY	VARCHAR2(20)	Yes	-	-
STTOWN	VARCHAR2(20)	Yes	-	-
STZIP	VARCHAR2(5)	Yes	-	-
STADDRESS	VARCHAR2(50)	Yes	-	-
STBIRTHDATE	DATE	Yes	-	-
STGRADSCHOOL	VARCHAR2(30)	Yes	-	-
STGRADTYPE	VARCHAR2(10)	Yes	-	-
STCGPA	VARCHAR2(4)	Yes	-	-
STPHONE	VARCHAR2(11)	Yes	-	-
STVNAME	VARCHAR2(20)	Yes	-	-
STVSURNAME	VARCHAR2(20)	Yes	-	-
STVPHONE	VARCHAR2(11)	Yes	-	-
STEMAIL	VARCHAR2(20)	Yes	-	-

(fig 4.1 TSTUDENT Table)

```

CREATE TABLE "TDOCTOR"
(
    "DCTCID" VARCHAR2(11),
    "DCNAME" VARCHAR2(20),
    "DCSURNAME" VARCHAR2(20),
    "DCCITY" VARCHAR2(20),
    "DCTOWN" VARCHAR2(20),
    "DCZIP" VARCHAR2(5),
    "DCADDRESS" VARCHAR2(50),
    "DCBIRTHDATE" DATE,
    "DCGRADSCHOOL" VARCHAR2(20),
    "DCGRADMAJOR" VARCHAR2(20),
    "DCEMAIL" VARCHAR2(20),
    "DCPHONE" VARCHAR2(11),
    PRIMARY KEY ("DCTCID") ENABLE
)

```



Column Name	Data Type	Nullable	Default	Primary Key
DCTCID	VARCHAR2(11)	No	-	1
DCNAME	VARCHAR2(20)	Yes	-	-
DCSURNAME	VARCHAR2(20)	Yes	-	-
DCCITY	VARCHAR2(20)	Yes	-	-
DCTOWN	VARCHAR2(20)	Yes	-	-
DCZIP	VARCHAR2(5)	Yes	-	-
DCADDRESS	VARCHAR2(50)	Yes	-	-
DCBIRTHDATE	DATE	Yes	-	-
DCGRADSCHOOL	VARCHAR2(20)	Yes	-	-
DCGRADMAJOR	VARCHAR2(20)	Yes	-	-
DCEMAIL	VARCHAR2(20)	Yes	-	-
DCPHONE	VARCHAR2(11)	Yes	-	-

(fig 4.2 TDOCTOR Table)

```
CREATE TABLE "TCOURSE"
(
    "COURSEID" VARCHAR2(10),
    "COURSETITLE" VARCHAR2(20),
    PRIMARY KEY ("COURSEID") ENABLE
)
```

Column Name	Data Type	Nullable	Default	Primary Key
COURSEID	VARCHAR2(10)	No	-	1
COURSETITLE	VARCHAR2(20)	Yes	-	-

(fig 4.3 TCOURSE Table)

```
CREATE TABLE "TEXAM"
(
    "EXAMID" VARCHAR2(5),
    "EXAMDATE" DATE
)
```

Column Name	Data Type	Nullable	Default	Primary Key
EXAMID	VARCHAR2(5)	Yes	-	-
EXAMDATE	DATE	Yes	-	-

(fig 4.4 TEXAM Table)

```
CREATE TABLE "TEXAMRESULT"
```

```

(  "EXAMID" VARCHAR2(5),
    "STID" VARCHAR2(8),
    "MATNET" VARCHAR2(4),
    "PHYNET" VARCHAR2(4),
    "CHEMNET" VARCHAR2(4),
    "BIONET" VARCHAR2(4),
    "TURKNET" VARCHAR2(4),
    "HISNET" VARCHAR2(4),
    "GEONET" VARCHAR2(4),
    "PHYLONET" VARCHAR2(4),
    "TUR_RESULT" VARCHAR2(7),
    "SOS_1_RESULT" VARCHAR2(7),
    "MAT_1_RESULT" VARCHAR2(7),
    "FEN_1_RESULT" VARCHAR2(7),
    "ED_SOS_RESULT" VARCHAR2(7),
    "SOS_2_RESULT" VARCHAR2(7),
    "MAT_2_RESULT" VARCHAR2(7),
    "FEN_2_RESULT" VARCHAR2(7),
    CONSTRAINT "COM_PK" PRIMARY KEY ("EXAMID", "STID") ENABLE
)

```

Column Name	Data Type	Nullable	Default	Primary Key
EXAMID	VARCHAR2(5)	No	-	1
STID	VARCHAR2(8)	No	-	2
MATNET	VARCHAR2(4)	Yes	-	-
PHYNET	VARCHAR2(4)	Yes	-	-
CHEMNET	VARCHAR2(4)	Yes	-	-
BIONET	VARCHAR2(4)	Yes	-	-
TURKNET	VARCHAR2(4)	Yes	-	-
HISNET	VARCHAR2(4)	Yes	-	-
GEONET	VARCHAR2(4)	Yes	-	-
PHYLONET	VARCHAR2(4)	Yes	-	-
TUR_RESULT	VARCHAR2(7)	Yes	-	-
SOS_1_RESULT	VARCHAR2(7)	Yes	-	-
MAT_1_RESULT	VARCHAR2(7)	Yes	-	-
FEN_1_RESULT	VARCHAR2(7)	Yes	-	-
ED_SOS_RESULT	VARCHAR2(7)	Yes	-	-
SOS_2_RESULT	VARCHAR2(7)	Yes	-	-
MAT_2_RESULT	VARCHAR2(7)	Yes	-	-
FEN_2_RESULT	VARCHAR2(7)	Yes	-	-

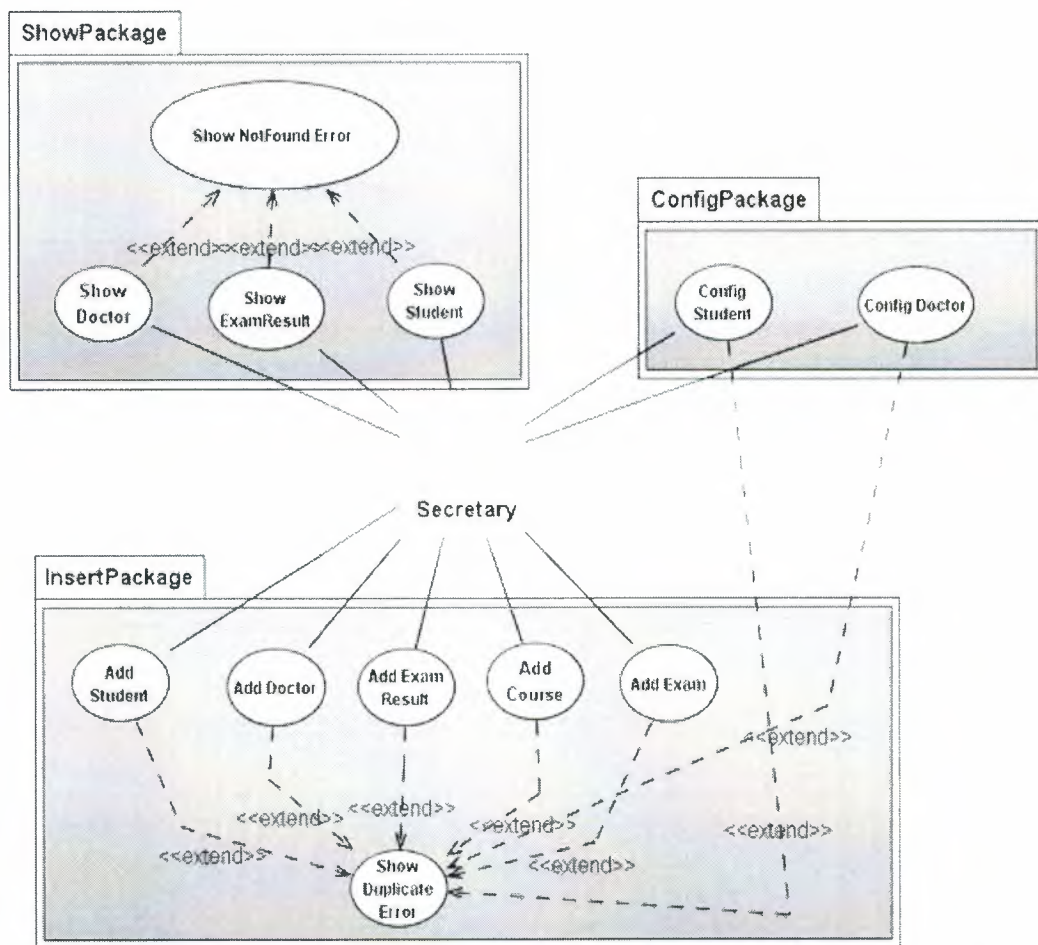
(fig 4.6 TEXAMRESULT Table)

## 4.2 UML Design Of System

Designing a software with UML before coding, will help the developer make less mistakes and great efficiency to improve the software later. Here in my project before coding UML diagrams have been created. Not all of the diagrams but the most needed ones.

### 4.2.1 Use Case Diagram

A use case illustrates a unit of functionality provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system. Here shown (see figure 4.7) that how the user acts with the database.



(fig 4.7 Use Case Diagram Of Project)

As shown above, only one actor interacts with the systems which is Secretary.

There are three systems used in the project.

1.Insert to Database

2.Config Data

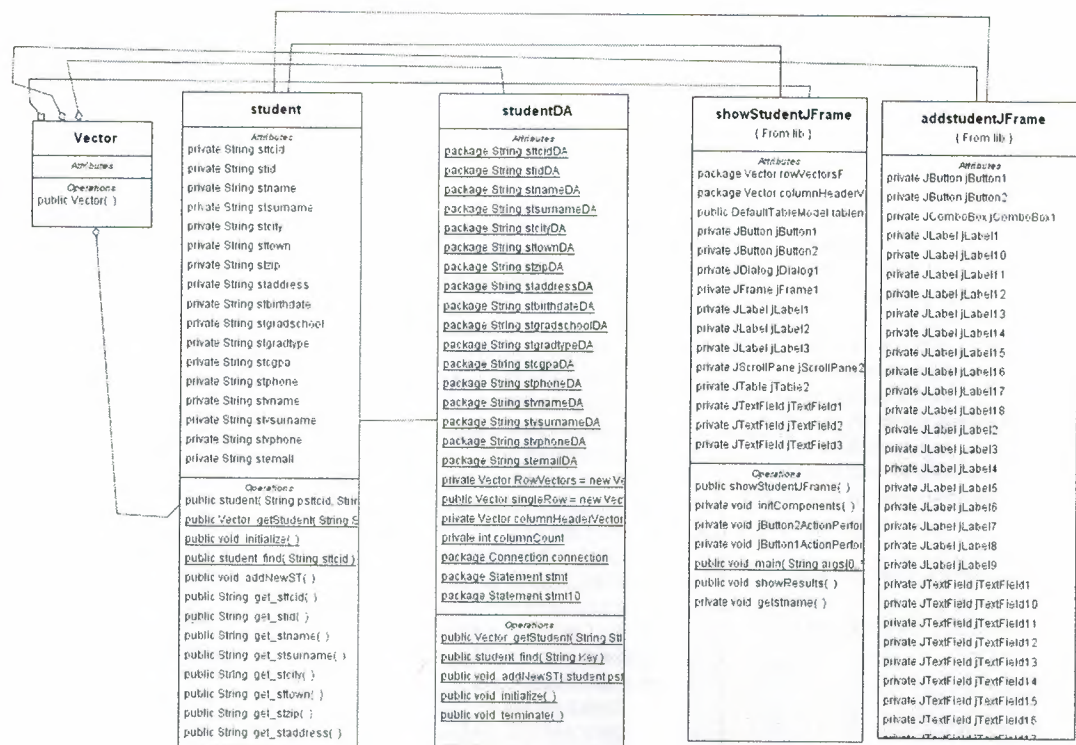
3.Show Data

It is possible to have errors according to the usage of the system. They are controlled with <extend> rows. Which is controlled by Exception classes in Java.

#### **4.2.2 Class Diagrams**

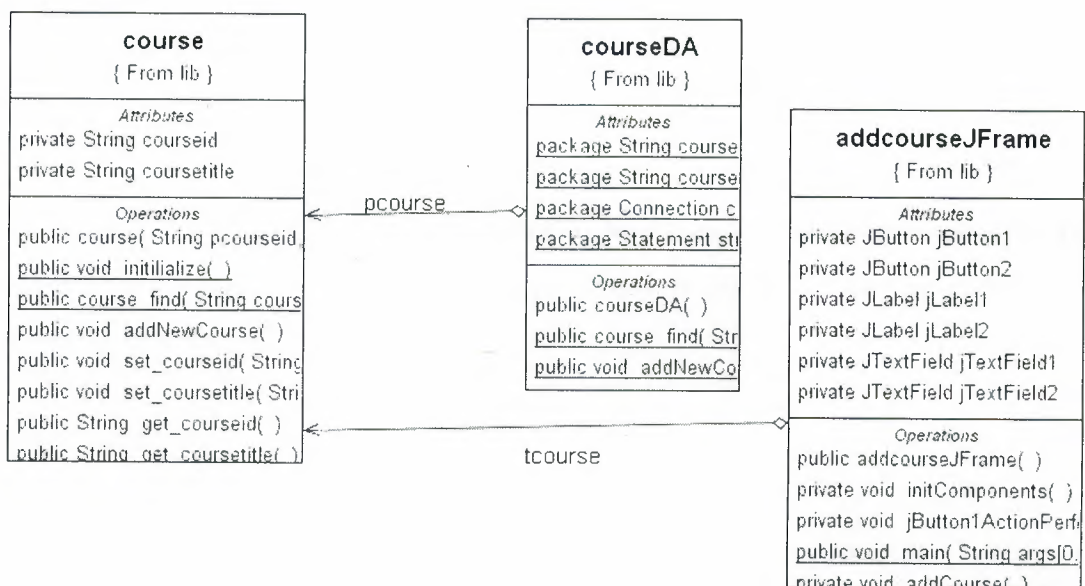
Here a few Class Diagrams of a Student Institute System. Class Diagrams beyond students are shown below. The user uses the JFrame classes to manipulate data in DB by using student Class. After all, student class associates with studentDA data access class to connect , fetch, insert, and update data.



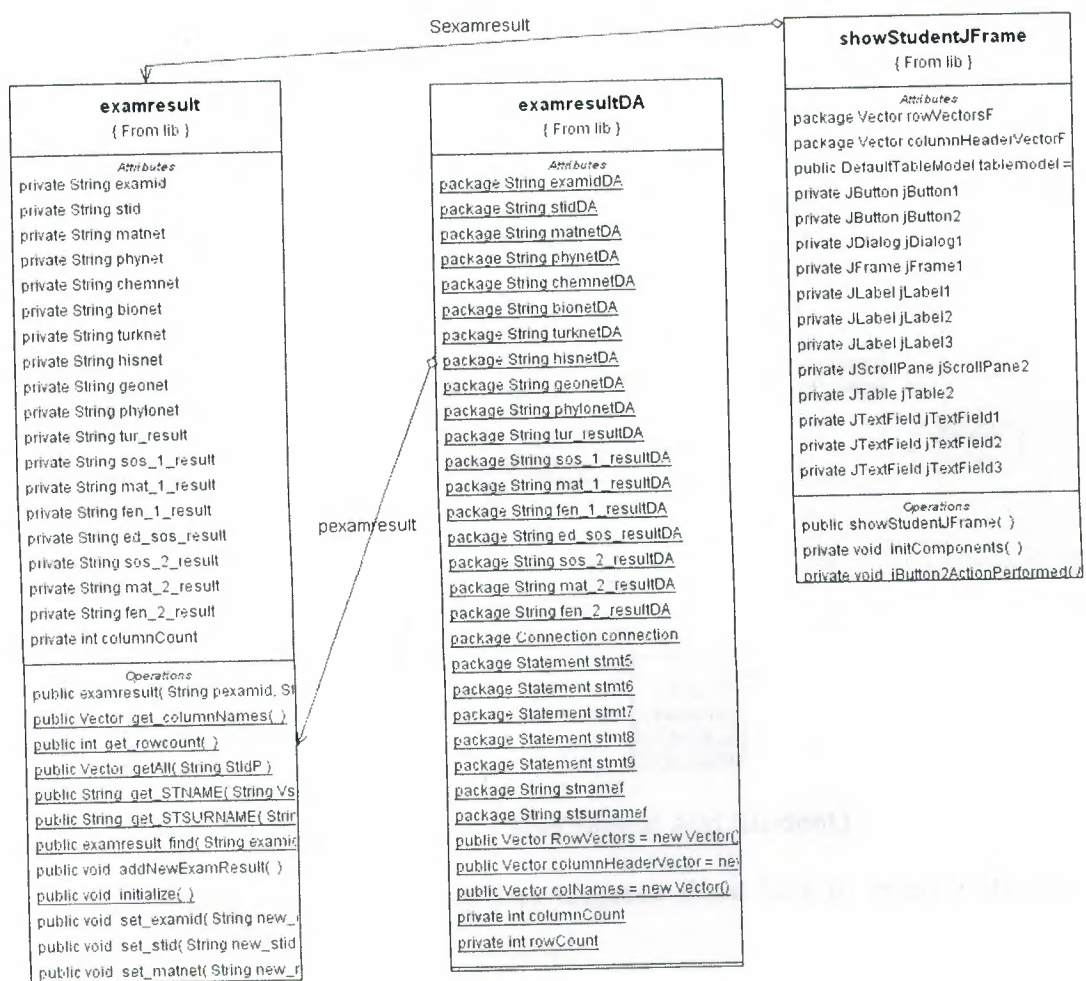


(fig 4.8 Student Class Diagram Of The System)

By using this model, user never interacts with database objects directly. It will be clean and safe for the programmer.



(fig 4.9 Course Class Diagram Of the System)



(fig 4.10 Examresult Class Diagram of the system)

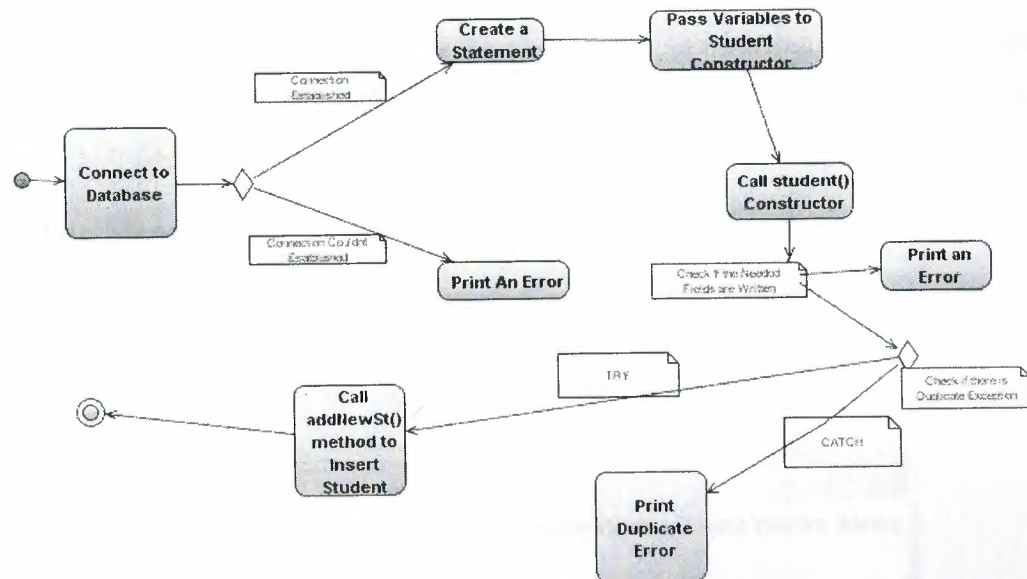
Aggregations between classes show us how they are communicates each other. ExamresultDA connects to examresult by pexamresult object. So user never touches to data access classes with this model.

### 4.2.3 Activity Diagram (Add Student)

Activity diagrams represent the business and operational workflows of a system. An Activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state.

So, what is the importance of an Activity diagram, as opposed to a State diagram? A State diagram shows the different states an object is in during the

lifecycle of its existence in the system(see fig 4.11), and the transitions in the states of the objects. These transitions depict the activities causing these transitions, shown by arrows.



(fig 4.11 Activity Diagram of Add Student)

This figure shows us an activity diagram about how to insert a student into database.

### 4.3 Java Part of The System

The system supports the user the following things;

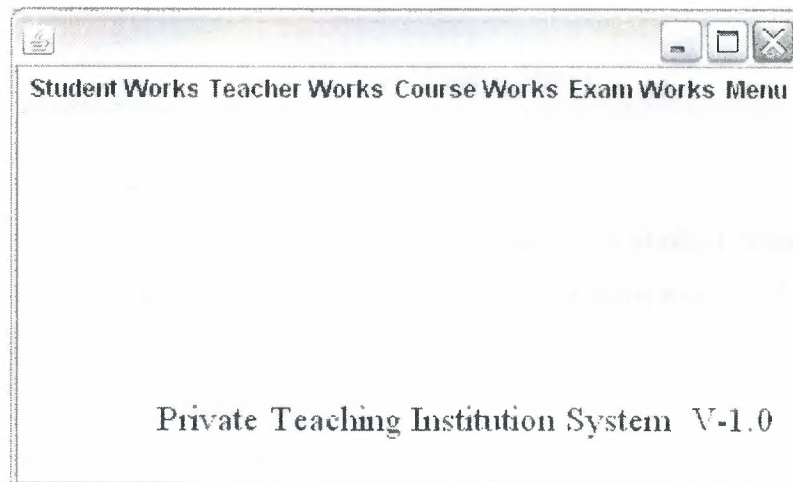
- Register student to database.
- Show student details.
- Add an exam result for a student.
- Show exam results of the student.
- Register a doctor to database.
- Show doctor details.

- Config Doctor details.
- Add a course to database
- Add an exam to database.

This project is modelled with the same view, so inserting a record to database shows us the same way. The only thing it changes is the variable names. This section I will try to tell what project codes do.

#### 4.3.1 Main Menu

The aim of the main menu(see figure 4.12) is to allow user to use the functionalities of the program easily.



(fig 4.12 Main Menu)

#### 4.3.2 Add Student

On this frame(see figure 4.13), user inserts a new student to database. TCID is the primary key at TSTUDENT table. Without inserting data to TC ID, ID , Name and Surname parts it gives an error to fill in the boxes.



ID	<input type="text"/>	Name	<input type="text"/>
SurName	<input type="text"/>	Sex	Male ▼
City	<input type="text"/>	Town	<input type="text"/>
Zip	<input type="text"/>	Address	<input type="text"/>
Birthdate	<input type="text"/>	E-mail	<input type="text"/>
School	<input type="text"/>	CGPA	<input type="text"/>
Graduation Type	<input type="text"/>	Phone	<input type="text"/>
Parent Name	<input type="text"/>	Parent SurName	<input type="text"/>
Parent Phone	<input type="text"/>	TC ID	<input type="text"/>
		Clear	Insert

(fig 4.13 Add Student Menu)

### 4.3.3 Add Exam Result

This menu allows user to add an exam result of a student where the stid and exam id fields should be declared before. Therefore the known exam and known student by the system makes a relation together. They are collected together in TEXAMRESULT table. There are a few stuff fetches inside the program to calculate the result(see Table 4.1), which is shown in figure below.

AÖSS PUANLARI	Tür	Sos-1	Mat-1	Fen-1	Ed-Sos	Sos-2	Mat-2	Fen-2	Dil
AÖSS-SÖZ-1	1.0	0.7	0.3	0.2	-	-	-	-	-
AÖSS-SAY-1	0.3	0.2	1.0	0.7	-	-	-	-	-
AÖSS-EA-1	0.8	0.3	0.9	0.2	-	-	-	-	-
AÖSS-DİL	0.5	0.2	0.2	0.1	-	-	-	-	1.2
AÖSS-SÖZ-2	0.5	0.35	0.3	0.2	0.5	0.35	-	-	-
AÖSS-SAY-2	0.3	0.2	0.5	0.35	-	-	0.5	0.35	-
AÖSS-EA-2	0.4	0.3	0.45	0.2	0.4	-	0.45	-	-

(Table 4.1 Calculating the Exam Results)

Student ID	<input type="text"/>	Find	Exam ID	<input type="text"/>	Find
Student Name	<input type="text"/>				
Student SurName	<input type="text"/>				
	Correct		Incorrect		
Maths	<input type="text"/>		<input type="text"/>		
Physics	<input type="text"/>		<input type="text"/>		
Chemistry	<input type="text"/>		<input type="text"/>		
Biology	<input type="text"/>		<input type="text"/>		
Turkish	<input type="text"/>		<input type="text"/>		
History	<input type="text"/>		<input type="text"/>		
Geography	<input type="text"/>		<input type="text"/>		
Phylosophy	<input type="text"/>		<input type="text"/>		
					Insert

(fig 4.14 Add Exam Result Menu)

By pressing the Find button it looks to Db to find if there is any student record with a stid that you write to editbox. So the user will control if he/she inset the correct student' s exam result.

#### 4.3.4 Show Exam Results

The crucial point of the system is to show the exam results of the student. I will give a brief explanation how it works , what it does and why we need it.

A student has more than 303 exam in a year which is arranged by the institute, in order to know the results of a student in a whole year or a spesified period it is elegant to record these results in a database. This section is what we do.

If we have a look into how is it happening in the Java language we need to understand a a few stuff. Firstly, the data ahs to be fetched from database by using a query as shown here;

```
" SELECT * FROM TEXAMRESULT WHERE STID = ' " + StIdP + " ' ";
```

Now, we have the rows we need to show to user. At this time we will save this data in a dynamic array which is called VECTOR. The codes are shoown below.

```
while (getResults.next()) {
    Vector singleRow = new Vector();
    for (int i=0; i< columnCount ; i++)
        singleRow.addElement(getResults.getObject(i+1));
    RowVectors.addElement(singleRow);
}
```

This query result will be added into getResults resultset in order to get the data.

EXAMID	STID	MATN	PHYN	CHEM	BIONET	TURK	HISNET	GEON	PHYL	TUR	SOS	MAT_1	FEN	ED_S	SOS	MAT_2	FEN
07011	2003...	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75
07012	2003...	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75	3.75
12312	2003...	42.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12334	2003...	43.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12332	2003...	41.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12342	2003...	41.25	20.5	32.75	50.25	10.75	66.75	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

(fig 4.15 Show Exam Result Menu)

The figure above(see figure 4.15) shows us a sample output of a student's exam results.

### 4.3.5 Add Teacher

Just like adding a student , adding a teacher has the same methods too. Only the attributes points a difference. I will explain how to insert data to database later times whereas Add Exam Class.

TCID	<input type="text"/>	Name	<input type="text"/>
Surname	<input type="text"/>	City	<input type="text"/>
Town	<input type="text"/>	Address	<input type="text"/>
Zip	<input type="text"/>	Birthdate	<input type="text"/>
Grad. School	<input type="text"/>	Majority	<input type="text"/>
E-Mail	<input type="text"/>	Phone	<input type="text"/>
<input type="button" value="Clear"/>		<input type="button" value="Insert"/>	

(fig 4.16 Add Doctor Menu)

By pressing **Insert** button. The data' s is inserted to Db which the user has entered to edit boxes.

The aim of the **Clear** button is to clear the fields filled before.

#### 4.7 Add Exam

As I mentioned before exams are the most important stuff in this project, according to this importance I will try to explain how to insert an exam detail to Database.

In java we are working with classes. In this case we should have a Exam class in order to declare the attributes and the methods of an Exam.

Here is the codes written in Java which I coded in my project.

```
public class exam {
    private String examid;
    private String examdate;
    /** Creates a new instance of exam */
    public exam(String pexamid, String pexamdate) {
        set_examid(pexamid);
        set_date(pexamdate);
    }
    public static void initialize(){
```



```

        examDA.initialize();
    }
    public static exam find(String examid) throws NotFoundException {
        return examDA.find(examid);
    }
    public void addNewExam() throws DuplicateException {
        examDA.addNewExam(this);
    }
    public void set_examid(String new_examid){
        examid = new_examid;
    }
    public void set_date(String new_date){
        examdate = new_date;
    }
    public String get_date(){
        return examdate;
    }
    public String get_examid(){
        return examid;
    }
}

```

As we have seen above some methods are related with another class called examDA.

Data access classes are classes to access to database as it can be understood from its name. By this approach domain classes communicates with data access classes and DA classes communicates with Database. Of course the user communicates with database by using GUI classes. So our Class diagrams works here again. I will give a couple of code example about DA classes below.

```

public class examDA {
    static String examidDA;
    static String examdateDA;
    static exam pexam;
    static Connection connection;
    static Statement stmt4;

    /** Creates a new instance of examDA */
    public examDA() {
    }
    public static exam find(String Key)throws NotFoundException {

        pexam = null;
        String findQuery = "SELECT * FROM texam WHERE examid = '" + Key + "'";
        //Execute the findQuery
        try {

```

```

ResultSet rs1 = stmt4.executeQuery(findQuery);
boolean found = rs1.next();
    if(found) {
        System.out.println("found");

        pexam = new exam(examidDA,examdateDA);
    }else {
        //System.out.println("not found");
        throw (new NotFoundException("notfound"));
    }
    rs1.close();
}
catch(SQLException e){
    System.out.println(e);}

return pexam;
}

public static void addNewExam(exam pexam) throws DuplicateException {
    examidDA = pexam.get_examid();
    examdateDA = pexam.get_date();

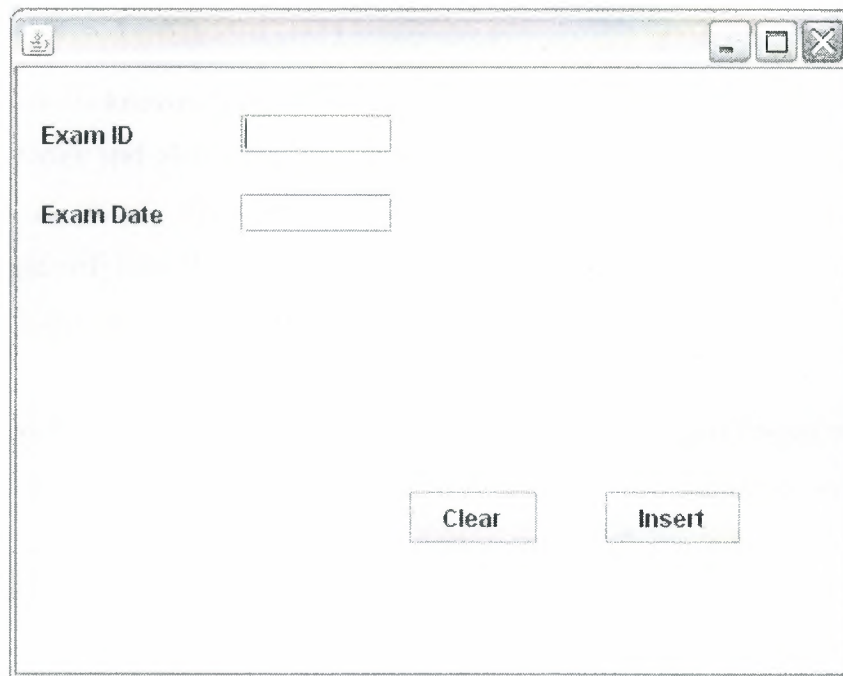
    System.out.println("examda addnew works");
    //control statement
    //System.out.println(pstudent.get_sttcid());

    try {
        System.out.println("control point #1");
        exam c = find(examidDA);

        throw (new DuplicateException("exam exists in db"));
    }
    catch(NotFoundException e){
        try {
            int result = stmt4.executeUpdate(sqlInsert);
        }
        catch(SQLException ee)
        {
            System.out.println(ee);
        }
    }
}
}

```

These are the stuff running at background. Now, we see how the user interacts with the data(see figure 4.17).



(fig 4.17 Add Exam Menu)

The codes below gets the string from the edit boxes and send it directly to exam class. We did see what these codes make with the data.

```
public void addExam() {
    String examidF = jTextField1.getText();
    String dateF = jTextField2.getText();
    texam = new exam(examidF,dateF);
    if (jTextField1.getSelectedText() == "" || jTextField2.getSelectedText() == "") {
        System.out.println("You should insert at least these parameters");
    }
    else {

        try
        {
            texam.addNewExam();
            //System.out.println("student added somthing");
        }
        catch(DuplicateException e) {
            System.out.println("there is an error");
        }
    }
}
```

## CONCLUSION

As it is known computers have entered too many fields in our life like, conomy, factories and also education. This program tries to help education sector. The project tries to satisfy the user needs to control the students success on exams. This system design will help the developers to increase the availabilities of their knowledge and the usefulness of the program time by time.

A high technology programming language Java is used to design the system. It has user friendly interfaces o communicate with the user. Also a database system is used to save, add, edit elements in the database. And so the details of students, doctors, exams, and courses has recorded in order to fetch and to use later.

As it is mentioned above , some instances ,refinements and modifications may be added to this software to increase the efficiency and performance activity also.



## REFERENCES

- [1] [java.sun.com/](http://java.sun.com/)
- [2] [java.about.com/](http://java.about.com/)
- [3] [http://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Java_(programming_language))
- [4] [java.sun.com/docs/books/tutorial/java/nutsandbolts/](http://java.sun.com/docs/books/tutorial/java/nutsandbolts/)
- [5] [http://en.wikipedia.org/wiki/Object-oriented\\_programming](http://en.wikipedia.org/wiki/Object-oriented_programming)
- [6] <http://java.sun.com/docs/books/tutorial/java/concepts/>
- [7] [java.sun.com/j2se/1.4.2/docs/api/javax/swing/JButton.html](http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JButton.html)
- [8] [www.uml.org/](http://www.uml.org/)
- [9] [http://en.wikipedia.org/wiki/Oracle\\_database](http://en.wikipedia.org/wiki/Oracle_database)
- [10] <http://www.tomjewett.com/dbdesign/dbdesign.php?page=ddlddl.php>
- [11] [http://orafaq.com/faq/what\\_are\\_the\\_difference\\_between\\_ddl\\_dml\\_and\\_dcl\\_commands](http://orafaq.com/faq/what_are_the_difference_between_ddl_dml_and_dcl_commands)

## APPENDIX

```
/*
 * student.java
 *
 * Created on March 8, 2007, 11:42 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package lib;
import java.sql.*;
import java.io.*;
import java.util.Vector;

/**
 * @author Burak
 */
public class student {
    private String sttcid;
    private String stid;
    private String stname;
    private String stsurname;
    private String stcity;
    private String sttown;
    private String stzip;
    private String staddress;
    private String stbirthdate;
    private String stgradschool;
    private String stgradtype;
    private String stcgpa;
    private String stphone;
    private String stvname;
    private String stvsurname;
    private String stvphone;
    private String stemail;
    private String stdept;
    public student(String psttcid, String pstid, String pstname, String pstsurname ,String
pstcity, String psttown,
        String pstzip, String pstaddress, String pstbirthdate, String pstgradschool, String
pstgradtype, String pstcgpa,
        String pstphone, String pstvname, String pstvsurname, String pstvphone, String
pstemail ,String pstdept)
    {
        // public student(String sttcid, String stid, String stname, String stsurname ,String
stcity, String sttown, String stzip, String staddress, String stbirthdate, String
```



```
stgradschool, String stgradtype, String pstcgpa, String pstphone, String pstvname,
String pstvsurname, String pstvphone, String pstemail ) {
    /*sttcid = psttcid;
    stid = pstid;
    stname = pstname;
    stsurname = pstsurname;
    stcity = pstcity;
    sttown = psttown;
    stzip = pstzip;
    staddress = pstaddress;
    stbirthdate = pstbirthdate;
    stgradschool = pstgradschool;
    stgradtype = pstgradtype;
    stcgpa = pstcgpa;
    stphone = pstphone;
    stvname = pstvname;
    stvsurname = pstvsurname;
    stvphone = pstvphone;
    stemail=pstemail;*/
    set_stid(pstid);
    set_sttcid(psttcid);
    set_stname(pstname);
    set_stsurname(pstsurname);
    set_stcity(pstcity);
    set_sttown(pstown);
    set_stzip(pstzip);
    set_staddress(pstaddress);
    set_stbirthdate(pstbirthdate);
    set_stgradschool(pstgradschool);
    set_stgradtype(pstgradtype);
    set_stcgpa(pstcgpa);
    set_stvphone(pstphone);
    set_stvname(pstvname);
    set_stvsurname(pstvsurname);
    set_stemail(pstemail);
    set_stdept(pstdept);
}

public static Vector getStudent(String StIdP){
    return studentDA.getStudent(StIdP);
}

public static String get_stdept(String StIdP){
    return studentDA.get_stdept(StIdP);
}

public static void initialize() {
    studentDA.initialize();
}

public static student find(String sttcid) throws NotFoundException{
    return studentDA.find(sttcid);
}
```

```

public void addNewST() throws DuplicateException {
    studentDA.addNewST(this);
}
public void set_stdept(String new_stdept){
    stdept = new_stdept;
}
public String get_stdept(){
    return stdept;
}
public String get_sttcid() {
    return sttcid;
}

public String get_stid() {
    return stid;
}

public String get_stname() {
    return stname;
}

public String get_stsurname() {
    return stsurname;
}
public String get_stcity() {
    return stcity;
}

}
public String get_sttown() {
    return sttown;
}
}
public String get_stzip() {
    return stzip;
}
}
public String get_staddress() {
    return staddress;
}
}
public String get_stbirthdate() {
    return stbirthdate;
}
}
public String get_stgradschool() {
    return stgradschool;
}
}
public String get_stgradtype() {
    return stgradtype;
}
}
public String get_stcgpa() {
    return stcgpa;
}
}

```



```

public String get_stphone() {
    return stphone;
}
public String get_stvname() {
    return stvname;
}
public String get_stvsurname() {
    return stvsurname;
}
public String get_stemail() {
    return stemail;
}
public String get_stvphone() {
    return stvphone;
}

public void set_stid(String new_stid)
{
    stid = new_stid;
}
public void set_sttcid(String new_sttcid) {
    sttcid = new_sttcid;
}
public void set_stname(String new_stname){
    stname = new_stname;
}
public void set_stsurname(String new_stsurname){
    stsurname = new_stsurname;
}
public void set_stcity(String new_stcity){
    stcity = new_stcity;
}
public void set_sttown(String new_sttown){
    sttown = new_sttown;
}
public void set_stzip(String new_stzip){
    stzip = new_stzip;
}
public void set_staddress(String new_staddress){
    staddress = new_staddress;
}
public void set_stbirthdate(String new_stbirthdate){
    stbirthdate = new_stbirthdate;
}
public void set_stgradschool(String new_stgradschool){
    stgradschool = new_stgradschool;
}
public void set_stgradtype(String new_stgradtype){
    stgradtype = new_stgradtype;
}

```

```

    }
    public void set_stcgpa(String new_stcgpa){
        stcgpa = new_stcgpa;
    }
    public void set_stvphone(String new_stvphone){
        stvphone = new_stvphone;
    }
    public void set_stvname(String new_stvname){
        stvname = new_stvname;
    }
    public void set_stvsurname(String new_stvsurname){
        stvsurname = new_stvsurname;
    }
    public void set_stemail(String new_stemail){
        stemail=new_stemail;
    }

    public static void terminate() {
        studentDA.terminate();
    }
}
/*
 * studentDA.java
 *
 * Created on March 14, 2007, 8:55 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

```

```

package lib;

```

```

import java.sql.*;
import java.io.*;
import java.util.Vector;
import oracle.jdbc.driver.*;
//Package for Oracle Extensions to JDBC
import oracle.jdbc.pool.OracleDataSource;

```

```

/**
 *
 * @author Burak
 */

```

```

public class studentDA {
    //static Connection connection;
    static student pstudent;
    static String sttcidDA;

```

```

static String stdDA;
static String stnameDA;
static String stsurnameDA;
static String stcityDA;
static String sttownDA;
static String stzipDA;
static String staddressDA;
static String stbirthdateDA;
static String stgradschoolDA;
static String stgradtypeDA;
static String stcgpaDA;
static String stphoneDA;
static String stvnameDA;
static String stvsurnameDA;
static String stvphoneDA;
static String stemailDA;
static String stdeptDA;
private static Vector RowVectors = new Vector();
public static Vector singleRow = new Vector();
private static Vector columnHeaderVector = new Vector();
private static int columnCount;
//dbConnection db;
// ResultSet rs = null;
static Connection connection;
static Statement stmt,stmt10;
public static String get_stdept(String StIdP){
    String getstdeptQuery = "SELECT stdept FROM TSTUDENT WHERE STID = '" +
StIdP + "'";
    try {
        ResultSet getResults = stmt10.executeQuery(getstdeptQuery);

        stdeptDA = getResults.getObject(0).toString();
    }
    catch(SQLException e){
        System.out.println(e);
    }
    return stdeptDA;
}
public static Vector getStudent(String StIdP){
    RowVectors.clear();
    columnHeaderVector.clear();
    String getstQuery = "SELECT * FROM TSTUDENT WHERE STID = '" + StIdP
+ "'";
    try {
        ResultSet getResults = stmt10.executeQuery(getstQuery);
        columnCount = getResults.getMetaData().getColumnCount();
        System.out.println(columnCount);
        while (getResults.next()) {

```

```

        Vector singleRow = new Vector();
        for (int i=0; i< columnCount ; i++)
            singleRow.addElement(getResults.getObject(i+1));
        // Row Vectors.addElement(singleRow);
    }
} catch (SQLException e) {
    System.out.println(e);
}

return singleRow;
}
/** Creates a new instance of studentDA */
public static student find(String Key)throws NotFoundException {

    pstudent = null;
    String findQuery = "SELECT * FROM tstudent WHERE sttcid = '" + Key + "'";
    //Execute the findQuery
    try {
        ResultSet rs = stmt.executeQuery(findQuery);
        boolean found = rs.next();
        if(found) {
            System.out.println("There is a student with this ID");

            pstudent = new student(sttcidDA,stidDA, stnameDA, stsurnameDA ,
            stcityDA, sttownDA, stzipDA, staddressDA, stbirthdateDA, stgradschoolDA,
            stgradtypeDA, stcgpaDA, stphoneDA, stvnameDA, stvsurnameDA, stvphoneDA,
            stemailDA,stdeptDA);
        }else {
            //System.out.println("not found");
            throw (new NotFoundException("notfound"));
        }
        rs.close();
    }
    catch(SQLException e){
        System.out.println(e);}

return pstudent;
}

public static void addNewST(student pstudent) throws DuplicateException {
    sttcidDA = pstudent.get_sttcid();
    stidDA = pstudent.get_stid();
    stnameDA = pstudent.get_stname();
    stsurnameDA = pstudent.get_stsurname();
    stcityDA =pstudent.get_stcity();
    sttownDA = pstudent.get_sttown();
    stzipDA = pstudent.get_stzip();
    staddressDA = pstudent.get_staddress();
    stbirthdateDA = pstudent.get_stbirthdate();
    stgradschoolDA = pstudent.get_stgradschool();

```



```

stgradtypeDA = pstudent.get_stgradtype();
stcgpaDA = pstudent.get_stcgpa();
// stphoneDA = pstudent.get_stphone();
stvnameDA = pstudent.get_stvname();
stvsurnameDA = pstudent.get_stvsurname();
stvphoneDA = pstudent.get_stvphone();
stemailDA = pstudent.get_stemail();
stdeptDA = pstudent.get_stdept();
System.out.println("studentda addnew works");
//control statement
//System.out.println(pstudent.get_sttcid());
//create sql statement
//String sqlInsert = "INSERT INTO Tstudent " + "VALUES (" + sttcid + ",
"" + stid + "", "" + stname + "", "" + stsurname + "")";
String sqlInsert = "INSERT INTO
tstudent(sttcid,stid,stname,stsurname,stcity,stown,stzip,staddress,stbirthdate,stgradscho
ol,stgradtype,stcgpa,stvphone,stvname,stvsurname,stemail,stdept) VALUES (" +
sttcidDA + "", "" + stidDA + "", "" + stnameDA + "", "" + stsurnameDA + "", "" + stcityDA +
"", "" + sttownDA + "", "" + stzipDA + "", "" + staddressDA + "", "" + stbirthdateDA + "", "" +
stgradschoolDA + "", "" + stgradtypeDA + "", "" + stcgpaDA + "", "" + stvphoneDA + "", "" +
stvnameDA + "", "" + stvsurnameDA + "", "" + stemailDA + "", "" + stdeptDA + "")";

//control if anyother student exists
try {
    System.out.println("control point #1");
    student c = find(sttcidDA);

    throw (new DuplicateException("student exists in db"));
}
catch(NotFoundException e){
    try {
        int result = stmt.executeUpdate(sqlInsert);
    }
    catch(SQLException ee)
    {
        System.out.println(ee);
    }
}

}

public static void initialize() {
try {
    System.out.println("Trying to connect to the Database");

    // Load the properties file to get the connection information
    //Properties prop = null;

```

```

// Create a OracleDataSource instance
OracleDataSource ods = new OracleDataSource();
ods.setDriverType("oci");
ods.setDatabaseName("XE");
ods.setURL("jdbc:oracle:oci:@");
ods.setUser("system");
ods.setPassword("6484");
connection=ods.getConnection();
connection.setAutoCommit(true);
System.out.println(" Connected to database");
stmt = connection.createStatement();
stmt10 = connection.createStatement();
System.out.println("Statement created");
} catch(SQLException ex) { // Trap SQL errors
    System.out.println("Error in Connecting to the Database");
}

/* Return false if failed to obtain connection object
if( connection != null )
    return true;

return false;
//will create database connection

*/
}
public static void terminate()
{
    try
    {
        // close everything
        stmt.close();
        connection.close();
    }
    catch (SQLException e)
    { System.out.println(e); }
}

}
/*
 * addstudentJFrame.java
 *
 * Created on March 9, 2007, 12:50 AM
 */

package lib;
import java.awt.*;
import javax.swing.*;

```

```

/**
 *
 * @author Burak
 */
public class addstudentJFrame extends javax.swing.JFrame {
    student tstudent;
    mainJFrame main;
    /** Creates new form addstudentJFrame */
    public addstudentJFrame() {

        initComponents();
        tstudent.initialize();

        //studentDA stda = new studentDA();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code "> //GEN-BEGIN: initComponents
    private void initComponents() {
        jLabel6 = new javax.swing.JLabel();
        jLabel14 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox();
        jLabel12 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jButton1 = new javax.swing.JButton();
        jLabel9 = new javax.swing.JLabel();
        jLabel1 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jLabel7 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jTextField2 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jTextField6 = new javax.swing.JTextField();
        jTextField7 = new javax.swing.JTextField();
        jTextField8 = new javax.swing.JTextField();
    }

```

```

jTextField9 = new javax.swing.JTextField();
jTextField10 = new javax.swing.JTextField();
jTextField11 = new javax.swing.JTextField();
jTextField12 = new javax.swing.JTextField();
jLabel13 = new javax.swing.JLabel();
jLabel15 = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
jLabel17 = new javax.swing.JLabel();
jTextField13 = new javax.swing.JTextField();
jTextField14 = new javax.swing.JTextField();
jTextField15 = new javax.swing.JTextField();
jTextField16 = new javax.swing.JTextField();
jLabel18 = new javax.swing.JLabel();
jTextField17 = new javax.swing.JTextField();
jButton2 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE)
;
setResizable(false);
jLabel6.setText("E-mail");

jLabel14.setText("Zip");

jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[]
{ "Fen-Mat", "Turkce-Mat", "Sosyal" }));

jLabel12.setText("Parent SurName");

jLabel5.setText("Town");

jLabel11.setText("Parent Phone");

jLabel4.setText("City");

jLabel10.setText("Parent Name");

jButton1.setText("Insert");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

jLabel9.setText("Phone");

jLabel1.setText("ID");

jLabel8.setText("CGPA");

```





```

        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup())
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)

        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextField15,
        89,

        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextField13,
        89,

        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextField14,
        89,

        javax.swing.GroupLayout.DEFAULT_SIZE, 133, Short.MAX_VALUE)
        .addComponent(jTextField1,
        javax.swing.GroupLayout.DEFAULT_SIZE, 133, Short.MAX_VALUE)
        .addComponent(jTextField2,
        javax.swing.GroupLayout.DEFAULT_SIZE, 133, Short.MAX_VALUE)
        .addComponent(jTextField3,
        javax.swing.GroupLayout.DEFAULT_SIZE, 133, Short.MAX_VALUE)
        .addComponent(jTextField4,
        89,

        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextField5,
        89,

        javax.swing.GroupLayout.PREFERRED_SIZE,
        javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(67, 67, 67)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
TRAILING)
        .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
        .addGroup(layout.createSequentialGroup()
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.R
ELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)

        .addComponent(jLabel2)
        .addComponent(jLabel5)
        .addComponent(jLabel15)
        .addComponent(jLabel6)
        .addComponent(jLabel8)
        .addComponent(jLabel9)
        .addComponent(jLabel12)
        .addComponent(jLabel18)))
        .addComponent(jLabel13))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELA
TED, 40, Short.MAX_VALUE)

```

```

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                                .addComponent(jComboBox1, 0,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(jTextField9,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField11,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField12,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField10,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField8,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField6,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField7,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)
                                .addComponent(jTextField17,
javax.swing.GroupLayout.DEFAULT_SIZE, 89, Short.MAX_VALUE)))
        .addComponent(jButton2))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
, 32, Short.MAX_VALUE)
        .addComponent(jButton1)
        .addContainerGap()
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel1)
                                .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel2))
            .addGap(14, 14, 14)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jLabel3)
                                .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,

```



```

javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGroup(layout.createSequentialGroup()
        .addGap(3, 3, 3)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.BASELINE)
            .addComponent(jLabel13)
                                .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
        .addGap(17, 17, 17)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
                    .addComponent(jLabel4)
                                .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(3, 3, 3)
                        .addComponent(jLabel5)))
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignm
ent.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addGap(88, 88, 88)
                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
                                .addComponent(jLabel17)
                                    .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                                .addGroup(layout.createSequentialGroup()
                                    .addGap(15, 15, 15)
                                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Al
ignment.LEADING)
                                        .addComponent(jLabel14)
                                            .addComponent(jTextField16,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))

```



```

        .addGap(14, 14, 14)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel16)
            .addComponent(jTextField4,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGroup(layout.createSequentialGroup()
            .addGap(18, 18, 18)
            .addComponent(jLabel15))
        .addGroup(layout.createSequentialGroup()
            .addGap(15, 15, 15)
            .addComponent(jTextField8,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addGap(15, 15, 15)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel7)
            .addComponent(jTextField15,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 7, Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel10)
            .addComponent(jTextField14,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 67, Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jTextField12,
                javax.swing.GroupLayout.PREFERRED_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addComponent(jLabel6)
        .addGap(14, 14, 14)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jTextField11,
                javax.swing.GroupLayout.PREFERRED_SIZE,

```



```

        private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {GEN-
FIRST:event_jButton2ActionPerformed
// TODO add your handling code here:
        jTextField1.setText("");
        jTextField2.setText("");
        jTextField3.setText("");
        jTextField4.setText("");
        jTextField5.setText("");
        jTextField6.setText("");
        jTextField7.setText("");
        jTextField8.setText("");
        jTextField9.setText("");
        jTextField10.setText("");
        jTextField11.setText("");
        jTextField12.setText("");
        jTextField13.setText("");
        jTextField14.setText("");
        jTextField15.setText("");
        jTextField16.setText("");
        jTextField17.setText("");

    } //GEN-LAST:event_jButton2ActionPerformed

```

```

        private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {GEN-
FIRST:event_jButton1ActionPerformed
// TODO add your handling code here:
        addStudent();
        //trial if the set and get methods works or not works or not
        System.out.println("addstudent works");
    } //GEN-LAST:event_jButton1ActionPerformed

```

```

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {

            new addstudentJFrame().setVisible(true);
        }
    });
}

```

```

private void addStudent() {
    int num = jComboBox1.getItemCount();
    // for (int i=0;i<=num;++i){
    //     Object key = jComboBox1.getItemAt(i);
    // }
}

```



```

Object key = jComboBox1.getSelectedItem();
System.out.println(key);
//String stdeptF = jComboBox1.getItemAt(key);
String sttcidF = jTextField17.getText();
String stidf = jTextField1.getText();
String stnameF = jTextField7.getText();
String stsurnameF = jTextField2.getText();
//String stsex = jComboBox1.getSelectedItem();
String stcityF = jTextField3.getText();
String sttownF = jTextField6.getText();
String stzipF = jTextField16.getText();
String staddressF = jTextField8.getText();
String stbirthdateF = jTextField4.getText();
String stemailF = jTextField12.getText();
String stgradschoolF = jTextField5.getText();
String stcgpaF = jTextField11.getText();
String stgradtypeF = jTextField15.getText();
String stphoneF = jTextField10.getText();
String stvnameF = jTextField14.getText();
String stvsurnameF = jTextField9.getText();
String stvphoneF = jTextField13.getText();
String stdeptF = key.toString();
    tstudent= new student(sttcidF,stidf, stnameF, stsurnameF , stcityF, sttownF,
stzipF, staddressF, stbirthdateF, stgradschoolF, stgradtypeF, stcgpaF, stphoneF,
stvnameF, stvsurnameF, stvphoneF, stemailF, stdeptF);
        if (jTextField1.getText() == "" || jTextField2.getText()== "" ||
jTextField7.getText()== "" || jTextField17.getText()== "" ) {
            System.out.println("You should insert at least these parameters");
        }

    else {

        try
        {
            tstudent.addNewST();
            //System.out.println("student added something");
        }
        catch(DuplicateException e) {
            System.out.println("duplication occurred");
        }
    }

}

}

public void ShutDown() {

    tstudent.terminate();
    main.setVisible(true);

```



```

System.out.println("connection ans stmt been closed");
}

```

```

// Variables declaration - do not modify//GEN-BEGIN:variables

```

```

private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JComboBox jComboBox1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel12;
private javax.swing.JLabel jLabel13;
private javax.swing.JLabel jLabel14;
private javax.swing.JLabel jLabel15;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel18;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField10;
private javax.swing.JTextField jTextField11;
private javax.swing.JTextField jTextField12;
private javax.swing.JTextField jTextField13;
private javax.swing.JTextField jTextField14;
private javax.swing.JTextField jTextField15;
private javax.swing.JTextField jTextField16;
private javax.swing.JTextField jTextField17;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
private javax.swing.JTextField jTextField4;
private javax.swing.JTextField jTextField5;
private javax.swing.JTextField jTextField6;
private javax.swing.JTextField jTextField7;
private javax.swing.JTextField jTextField8;
private javax.swing.JTextField jTextField9;
// End of variables declaration//GEN-END:variables

```

```

}

```

```

/*

```

```

* examresult.java

```

```

*

```

```

* Created on March 9, 2007, 12:08 AM

```

```

*
* To change this template, choose Tools | Template Manager
* and open the template in the editor.
*/

```

```

package lib;
import java.sql.SQLException;
import java.util.Vector;
/**

```

```

*
* @author Burak
*/

```

```

public class examresult {
    private String examid;
    private String stid;
    private String matnet;
    private String phynet;
    private String chemnet;
    private String bionet;
    private String turknet;
    private String hisnet;
    private String geonet;
    private String phylonet;
    private String tur_result;
    private String sos_1_result;
    private String mat_1_result;
    private String fen_1_result;
    private String ed_sos_result;
    private String sos_2_result;
    private String mat_2_result;
    private String fen_2_result;
    private int columnCount;

```

```

/** Creates a new instance of examresult */

```

```

    public examresult(String pexamid,String pstid,String pmatnet, String pphynet, String
    pchemnet,String pbionet,String pturknet,String phisnet, String pgeonet, String
    pphylonet,String ptur_result,String psos_1_result,String pmat_1_result, String
    pfen_1_result, String ped_sos_result, String psos_2_result, String pmat_2_result, String
    pfen_2_result ) {

```

```

        set_examid(pexamid);
        set_stid(pstid);
        set_matnet(pmatnet);
        set_phynet(pphynet);
        set_chemnet(pchemnet);
        set_bionet(pbionet);
        set_turknet(pturknet);
        set_hisnet(phisnet);
        set_geonet(pgeonet);
        set_phylonet(pphylonet);

```

```

        set_tur_result(ptur_result);
        set_sos_1_result(psos_1_result);
        set_mat_1_result(pmat_1_result);
        set_fen_1_result(pfen_1_result);
        set_ed_sos_result(ped_sos_result);
        set_sos_2_result(psos_2_result);
        set_mat_2_result(pmat_2_result);
        set_fen_2_result(pfen_2_result);
    }/*end of constructor*/
    public static Vector get_columnNames(){
        return examresultDA.getColumnNamesDA();
    }
    public static int get_rowcount(){
        return examresultDA.get_rowCountDA();
    }
    /* -- burada ö?renci ad?n? ve soyad?n? veritaban?ndan çekip framedki text boxlara
    yazd?racag?z---*/
    public static Vector getAll(String StIdP){
        return examresultDA.getAll(StIdP);
    }
    public static String get_STNAME(String Vstname) throws NotFoundException{
        return examresultDA.get_STNAME(Vstname);
    }
    public static String get_STSURNAME(String Vstsurname) throws
    NotFoundException {
        return examresultDA.get_STSURNAME(Vstsurname);
    }
    public static examresult find(String examid, String stid) throws NotFoundException{
        return examresultDA.find(examid,stid);
    }
    public void addNewExamResult() throws DuplicateException {
        examresultDA.addNewExamResult(this);
    }
    public static void initialize(){
        examresultDA.initialize();
    }

    public void set_examid(String new_exam_id){
        examid = new_exam_id;
    }
    public void set_stid(String new_stid){
        stid = new_stid;
    }
    public void set_matnet(String new_matnet){
        matnet = new_matnet;
    }
    public void set_phynet(String new_phynet){
        phynet = new_phynet;
    }

```

```

    }
    public void set_chemnet(String new_chemnet){
        chemnet = new_chemnet;
    }
    public void set_bionet(String new_bionet){
        bionet = new_bionet;
    }
    public void set_turknet(String new_turknet){
        turknet = new_turknet;
    }
    public void set_hisnet(String new_hisnet){
        hisnet = new_hisnet;
    }
    public void set_geonet(String new_geonet){
        geonet = new_geonet;
    }
    public void set_phylonet(String new_phylonet){
        phylonet = new_phylonet;
    }
    public void set_tur_result(String new_tur_result){
        tur_result = new_tur_result;
    }
    public void set_sos_1_result(String new_sos_1_result){
        sos_1_result = new_sos_1_result;
    }
    public void set_sos_2_result(String new_sos_2_result){
        sos_2_result = new_sos_2_result;
    }
    public void set_mat_1_result(String new_mat_1_result){
        mat_1_result = new_mat_1_result;
    }
    public void set_mat_2_result(String new_mat_2_result){
        mat_2_result = new_mat_2_result;
    }
    public void set_fen_1_result(String new_fen_1_result){
        fen_1_result = new_fen_1_result;
    }
    public void set_fen_2_result(String new_fen_2_result){
        fen_2_result = new_fen_2_result;
    }
    public void set_ed_sos_result(String new_ed_sos_result){
        ed_sos_result = new_ed_sos_result;
    }
    public String get_examid(){
        return examid;
    }
    public String get_stdid(){
        return stdid;
    }
}

```



```

public String get_matnet(){
    return matnet;
}
public String get_phynet(){
    return phynet;
}
public String get_chemnet(){
    return chemnet;
}
public String get_bionet(){
    return bionet;
}
public String get_turknet(){
    return turknet;
}
public String get_hisnet(){
    return hisnet;
}
public String get_geonet(){
    return geonet;
}
public String get_phylonet(){
    return phylonet;
}
public String get_ed_sos_result(){
    return ed_sos_result;
}
public String get_fen_1_result(){
    return fen_1_result;
}
public String get_fen_2_result(){
    return fen_2_result;
}
public String get_mat_1_result(){
    return mat_1_result;
}
public String get_mat_2_result(){
    return mat_2_result;
}
public String get_sos_1_result(){
    return sos_1_result;
}
public String get_sos_2_result(){
    return sos_2_result;
}
public String get_tur_result(){
    return tur_result;
}
}
/*end of class*/

```

```

/*
 * examresultDA.java
 *
 * Created on April 11, 2007, 8:18 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package lhr;
import java.sql.*;
import java.io.*;
import java.util.Vector;
//Package for Oracle Extension to JDBC
import oracle.jdbc.driver.*;
import oracle.jdbc.pool.OracleDataSource;
/**
 *
 *
 * @author Banki
 */
public class examresultDA {
    static String examidDA;
    static String studDA;
    static String namedDA;
    static String physidDA;
    static String chemidDA;
    static String himedDA;
    static String techidDA;
    static String hometDA;
    static String genidDA;
    static String physamedDA;
    static String tar_resultDA;
    static String ws_1_resultDA;
    static String mat_1_resultDA;
    static String fin_1_resultDA;
    static String ed_ws_resultDA;
    static String ws_2_resultDA;
    static String mat_2_resultDA;
    static String fin_2_resultDA;
    static examresult peramresult;
    static Connection connection;
    static Statement stmt1,stmt2,stmt3,stmt8,stmt9;
    static String stmt1, stmt2, stmt3, stmt8, stmt9;
    public static Vector RowVector = new Vector();
    public static Vector columnHeaderVector = new Vector();
    public static Vector colNames = new Vector();
    private static int columnCount;
    private static int rowCount;
    /** Creates a new instance of examresultDA */

```

```

public examresultDA() {
}
public static int get_rowCountDA(){
    return columnCount;
}
public static Vector getColumnNamesDA() {
    String TableName = "TEXAMRESULT";
    Vector colNames = new Vector();
    // String getcolNamesQuery = "select column_name from user_tab_cols where
table_name = 'TEXAMRESULT'";
    try {
        ResultSet getColResults =
connection.getMetaData().getColumns(null,null,TableName,null);
        //columnCount = getColResults.getMetaData().getColumnCount();
        while (getColResults.next()) {

            colNames.addElement(getColResults.getString("COLUMN_NAME"));
            columnHeaderVector.addElement(colNames);
        }
    } catch (SQLException e) {
        System.out.println(e);
    }
    return colNames;
}
public static Vector getAll(String StIdP){
/*    Vector examresults = new Vector();
    String getAllQuery = "SELECT * FROM TEXAMRESULT";
    try {
        ResultSet getResults = stmt8.executeQuery(getAllQuery);
        boolean moreResults = getResults.next();
        while (moreResults){
examidDA = getResults.getString(1);
stidDA =getResults.getString(2);
matnetDA =getResults.getString(3);
phynetDA =getResults.getString(4);
chemnetDA =getResults.getString(5);
bionetDA =getResults.getString(6);
turknetDA =getResults.getString(7);
hisnetDA =getResults.getString(8);
geonetDA =getResults.getString(9);
phylonetDA =getResults.getString(10);
tur_resultDA =getResults.getString(11);
sos_1_resultDA =getResults.getString(12);
mat_1_resultDA =getResults.getString(13);
fen_1_resultDA =getResults.getString(14);
ed_sos_resultDA =getResults.getString(15);
sos_2_resultDA =getResults.getString(16);
mat_2_resultDA =getResults.getString(17);
fen_2_resultDA =getResults.getString(18);

```

```

        //create an exam result instance //
        pexamresult = new examresult(examidDA, stdDA, matnetDA, phynetDA,
chemnetDA, bionetDA,
turknetDA, hisnetDA, geonetDA, phylonetDA, tur_resultDA, sos_1_resultDA, mat_1_resul
tDA, fen_1_resultDA,
ed_sos_resultDA, sos_2_resultDA, mat_2_resultDA, fen_2_resultDA);
        examresults.addElement(pexamresult);
        moreResults = getResults.next();
        } //end of while
    } //end of try

    catch (SQLException e)
    { System.out.println(e); }
return examresults; */
RowVectors.clear();
columnHeaderVector.clear();
String getAllQuery = "SELECT * FROM TEXAMRESULT WHERE STID = " +
StdP + """;
try {
    ResultSet getResults = stmt8.executeQuery(getAllQuery);
    //rowCount = getResults.getMetaData().
    columnCount = getResults.getMetaData().getColumnCount();
    System.out.println(columnCount);
    while (getResults.next()) {
        Vector singleRow = new Vector();
        for (int i=0; i< columnCount ; i++)
            singleRow.addElement(getResults.getObject(i+1));
        RowVectors.addElement(singleRow);
    }
} catch (SQLException e) {
    System.out.println(e);
}

return RowVectors;
}

public static String get_STNAME(String Vstid) throws NotFoundException{
    String getQuery = "SELECT stname FROM tstudent WHERE stid =" + Vstid + ""
;
    try {
        System.out.println("nokta 1");
        ResultSet result = stmt6.executeQuery(getQuery);

        boolean caught = result.next();
        if(caught){
            System.out.println("student found");
            stnamef = result.getString(1);
        }
    }

```



```

else { System.out.println("student has not found in the database");

} //stmt6.close();
} /* end of try */
catch ( SQLException e) {
    e.printStackTrace();
}
return stnamef;
}

public static String get_STSURNAME(String Vstid2) throws NotFoundException {
    String getQuery = "SELECT stsurname FROM tstudent WHERE stid =" + Vstid2
+ "" ;
    try {
        System.out.println("nokta 1");
        ResultSet result = stmt7.executeQuery(getQuery);

        boolean caught = result.next();
        if(caught){
            System.out.println("student found");
            stsurnamef = result.getString(1);
        } //stmt7.close();
    } /* end of try */
    catch ( SQLException e) {
        e.printStackTrace();
    }
    return stsurnamef;
}

```

```

public static examresult find(String Key , String Key2) throws NotFoundException {

    pexamresult = null;
    String findQuery = "SELECT * FROM texamresult WHERE examid = " + Key + " and
stid = " + Key2 + "" ;
    //Execute the findQuery
    /* --- burada primary key composite olacak --- */
    try {
        ResultSet rs1 = stmt5.executeQuery(findQuery);
        boolean found = rs1.next();
        if(found) {
            System.out.println("found");

            pexamresult = new examresult(examidDA,stidDA,
matnetDA,phynetDA,chemnetDA,bionetDA,turknetDA,hisnetDA,geonetDA,phylonetD
A,tur_resultDA,sos_1_resultDA,mat_1_resultDA,fen_1_resultDA,ed_sos_resultDA,sos
_2_resultDA,mat_2_resultDA,fen_2_resultDA);
        } else {
            //System.out.println("not found");
            throw (new NotFoundException("notfound"));
        }
    }
}

```

```

        rs1.close();
    }
    catch(SQLException e){
        System.out.println(e);
    }

    return pexamresult;
}

```

```

        public static void addNewExamResult(examresult pexamresult) throws
        DuplicateException {

```

```

        examidDA = pexamresult.get_examid();
        stidDA = pexamresult.get_stid();
        matnetDA = pexamresult.get_matnet();
        phynetDA = pexamresult.get_phynet();
        chemnetDA = pexamresult.get_chemnet();
        bionetDA = pexamresult.get_bionet();
        turknetDA = pexamresult.get_turknet();
        hisnetDA = pexamresult.get_hisnet();
        geonetDA = pexamresult.get_geonet();
        phylonetDA = pexamresult.get_phylonet();
        tur_resultDA = pexamresult.get_tur_result();
        sos_1_resultDA = pexamresult.get_sos_1_result();
        mat_1_resultDA = pexamresult.get_mat_1_result();
        fen_1_resultDA = pexamresult.get_fen_1_result();
        ed_sos_resultDA = pexamresult.get_ed_sos_result();
        sos_2_resultDA = pexamresult.get_sos_2_result();
        mat_2_resultDA = pexamresult.get_mat_2_result();
        fen_2_resultDA = pexamresult.get_fen_2_result();

```

```

        System.out.println("examresultda addnew works");
        //control statement
        //System.out.println(pstudent.get_sttcid());
        //create sql statement
        //String sqlInsert = "INSERT INTO Tstudent " + "VALUES (" + sttcid + ",
        "" + stid + "", "" + stname + "", "" + stsurname + "")";
        String sqlInsert = "INSERT INTO texamresult VALUES (" + examidDA + "", "" +
        stidDA + "", "" + matnetDA + "", "" + phynetDA + "", "" + chemnetDA + "", "" + bionetDA
        + "", "" + turknetDA + "", "" + hisnetDA + "", "" + geonetDA + "", "" + phylonetDA + "", ""
        + mat_1_resultDA + "", "" + mat_2_resultDA + "", "" + fen_1_resultDA + "", "" +
        fen_2_resultDA + "", "" + sos_2_resultDA + "", "" + sos_1_resultDA + "", "" +
        ed_sos_resultDA + "", "" + tur_resultDA + "")";

```

```

//control if anyother student exists

```

```

try {
    System.out.println("control point #1");
    examresult c = find(examidDA,stidDA);

```

```

        throw (new DuplicateException("you have already registered this student with this
exam to db"));
    }
    catch(NotFoundException e){
        try {
            int result = stmt5.executeUpdate(sqlInsert);
        }
        catch(SQLException ee)
        {
            System.out.println(ee);
        }
    }
}

public static void initialize() {
    try {
        System.out.println("Trying to connect to the Database");

        // Load the properties file to get the connection information
        //Properties prop = null;

        // Create a OracleDataSource instance
        OracleDataSource ods = new OracleDataSource();
        ods.setDriverType("oci");
        ods.setDatabaseName("XE");
        ods.setURL("jdbc:oracle:oci:@");
        ods.setUser("system");
        ods.setPassword("6484");
        connection=ods.getConnection();
        connection.setAutoCommit(true);
        System.out.println(" Connected to database");
        stmt5 = connection.createStatement();
        stmt6 = connection.createStatement();
        stmt7 = connection.createStatement();
        stmt8 = connection.createStatement();
        stmt9 = connection.createStatement();
        System.out.println("Statement created");
    } catch(SQLException ex) { // Trap SQL errors
        System.out.println("Error in Connecting to the Database");
    }

    /* Return false if failed to obtain connection object
    if( connection != null )
        return true;

    return false;
    //will create database connection
    }
}

```