NEAR EAST UNIVERSITY
LIBRARY
1988 - LEFKOSA

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

A JAVA APPLICATION
COMPUTER-BASED EXAMINATION

Graduation Project
COM 400

Student: Müslüm Şenel(20010811)

Supervisor: Asst. Prof. Dr Adil Amirjanov

Nicosia -2005

ACKNOWLEDGMENTS

I would like to acknowledge and thank to Adil Amirjanov for being such a great supervisor for helping me all along my project and in designing every phase of this applications's documentation.

Special thanks to my family and friends, chiefly my Mother and Father, who simply are the best.

Thanks also goes out to roommate and brother who were generous enough to allot their precious time day and night in every detail of application and sharing their suggestions with me in documentation to bring about this valuable work.

ABSTRACT

In today's world, computer applications play a very important and implacable role in our daily life tasks. Almost everything can be managed using computers in one of two ways. One way is making use of its hardware applications and the other one is by its software applications.

To develop a software application an environment is need which is called Programming Language. Programming languages give programmers the chance to write codes that will generate an output to do a job which would be very hard for human beings in terms of time, constraints or accuracy to perform in the same manner. In this respect Java is one and probably the most promising Programming Language and platform to build software applications. It offers a great deal of features and capabilities for programmers to develop applications for a variety of missions and odd jobs. Graphical User Interface and Database connectivity are two of its features that will be given much importance throughout this document and be employed to devise an application.

In this project a Java application is to be designed that will supersede old-way of test taking process with a novel and exciting form, computer-based test taking. With this program one will be able to take the test without writing down anything, and get a feedback about the results and performance at the end of each section, and also have the opportunity to save the results for later use. Such an interactive test taking fashion not only eases the banal paper-based test-taking process but also makes it fun and enjoyable.

TABLE OF CONTENTS

LIST OF ACRONYMS

JDBC	Java Database Connectivity.
GUI	Graphical User Interface.
API	Application Program Interface.
JVM	Java Virtual Machine.
OOP	Object Oriented Programming.
SQL	Structured Query Language.
ODBC	Open Database Connectivity.
ANSI	American Natioanal Standarts Institute.
UI	User Interface.
HTML	HyperText Markup Language.
RDBMS	Relational Database Management System.
DDL	Data Definition Language.
DML	Data Manipulation Labnguage.

INTRODUCTION

Java is a new programming environment and language but offers a lot of facilities and features. Programmers make great use of java for its novel approach in programming world. Java gets most of its base from its ancestors such as C and C++ and by including some completely new characteristics it becomes much efficient and much preferable language by programmers and companies.

Database Management Systems play a very important role in data storing. They might be used alone or they can be merged with any programming language to empower their usability and functionality.

In this project a software program is going to be designed in java for a computer-based examination by interweaving with Database.

Chapter one deals with the one of the most important features of java which is Object Oriented approach. This chapter gives information about classes, objects and the way they are used.

In chapter two java fundamentals are described briefly. That is; simple UI components and primitive data types and variable declaration.

Chapter three reviews the very basics of Database in general. It defines the database, the relational database system and SQL statements.

In chapter four the Java Database Connectivity is explained explicitly with sample codes showing how to connect to the database, execute SQL statements and retrieve the data stored in database.

Last chapter in this project presents everything done to develop a software program which is a computer based test in java and using Microsoft Access as Database.

CHAPTER ONE: OBJECT-ORIENTED PROGRAMMING CONCEPTS

One thing that makes java so strong and easy-to-use is being Object-Oriented .Here we will go over the basics of OOP and how they can be used in terms of codes. We will define OOP and then discuss its concepts and then examine class structure, method declaration, instance creation and everything regarding OOP that have been used in this application will be given special consideration and detail about them.

1.1 WHAT IS OOP?

OOP stands for object-oriented programming, and in reality it means that the programming is based on creating destroying objects. Every component in java is object except simple data types, all other components must be initialized before to use them.

In classic, procedural programming we try to make the real world problem we're attempting to solve fit a few, pre-determined data types: integers, floats, Strings, and arrays perhaps. In object oriented programming we create a model for a real world system. Classes are programmer-defined types that model the parts of the system.

A class is a programmer defined type that serves as a blueprint for instances of the class. We can still have ints, floats, Strings, and arrays; but we can also have cars, motorcycles, people, buildings, clouds, dogs, angles, students, courses, bank accounts, and any other type that's important to our problem.

Classes specify the data and behavior possessed both by themselves and by the objects built from them. A class has two parts: the fields and the methods. Fields describe what the class is. Methods describe what the class does.

Using the blueprint provided by a class, we can create any number of objects, each of which is called an instance of the class. Different objects of the same class have the same fields and methods, but the values of the fields will in general differ. For example, all humans have eye color but the color of each human's eyes can be different from others.

1.2. OOP CONCEPTS

1.2.1 Inheritance

Inheritance is an important parameter in OOP world. By this concept we can inherit the methods and variables declared in a class to another class which is called child class. This way we avoid writing much code and therefore save our recourses. We haven't use inheritance directly in our application but we did indirectly. For example:

```
public class About extends JPanel  {}
```

This implements that the class About will include all the features of a Panel and later on we may add some more methods and attributes.

1.2.2 Reusability

That is another OOP concept that we have used frequently in this project. OOP give the programmer the chance to reuse the codes written for another purpose with some modifications or just as its original form. For example the following code segment is class and has been used more than ten times throughout the program that connects to the Database:

```
db.connecttodatabase(deletetest);
```

Here db is a class that we will go over in the next section and connecttodatabase(deletetest) is a method that establishes the connection with the given parameter being a SQL query.

1.3 OBJECTS AND CLASSES

Object-oriented programming is modeled because, in the real world, objects are made up of many kinds of smaller objects. However, the capability to unite objects is only one general aspect of object-oriented programming. Object-oriented programming provides several other concepts and features to make the creation and use of objects easier and more flexible. The most important of these features is the class.

A class is a template for multiple objects with similar features. Classes embody all the features of a particular set of objects. When we write a program in an object-oriented language, we don't define individual objects. We define classes of objects.

For example, we might have a Tree class that describes the features of all trees (each tree has branches and roots, grows, and creates chlorophyll). The Tree class serves as an abstract model for the concept of a tree. To reach out and grab, or interact with, or cut down a tree, we must have a concrete instance of that tree. Of course, once we have a Tree class, we can create lots of different instances of that tree, and each different tree instance can have different features (it can be short, tall, bushy, drop leaves in autumn, and so on), yet still behave like a tree and be immediately recognizable as a tree.

An instance of a class is an actual object of that class. The class is the general, abstract representation of an object, and an instance is its concrete representation. So what precisely, is the difference between an instance and an object? The term object is used more generally, but both instances and objects are the concrete representations of a class. In fact, the terms instance and object often are used interchangeably in OOP terminology. A Tree instance and a Tree object are the same thing.

In an example closer to the kind of thing we may want to do with Java, we can create a class for an on-off switch. The LightSwitch class defines the following features of an on-off switch:

- Its label
- Its size
- Its appearance

The class also defines how an on-off switch behaves, as follows:

- Whether it needs a single click or a double click to be activated
- Whether it changes color when clicked
- What it does when it's activated

Once we define the LightSwitch class, we easily can create instances of that switch-in other words, LightSwitch objects. The instances all take on the basic features of a switch as defined by the class, but each instance can have different appearances and behaviors based on what we want that particular switch to do. By creating a LightSwitch class, we don't have to keep rewriting the code for each switch we want to use in our program. Also, we can reuse the LightSwitch class to create different kinds of switches as we need them-in any program.

When we write a Java program, we design and construct a set of classes. When our program runs, instances of those classes are created and discarded as needed. Our task, as a Java programmer, is to create the right set of classes to accomplish what our program needs to accomplish.

Fortunately, we don't have to start from scratch. The Java environment comes with a library of classes that implement a lot of the basic behaviors we need. A class library is a set of classes. The Java library has classes to handle basic programming tasks (math functions, arrays, strings, and so on) as well as classes to handle graphics and networking behavior. In many cases, the Java class libraries may be sufficient for our needs; all we have to do in our Java program is to create a single class that uses the standard class libraries. For complicated Java programs, however, we may have to create a whole set of classes with defined interactions between them.

1.4 ATTRIBUTES OF AN OBJECT

Attributes are the individual things that differentiate one object from another and determine the appearance, state, or other qualities of that object. Consider how a theoretical class called myclass could be created. The attributes of a myclass might include the following:

- Color: red, orange, yellow
- Sex: male, female
- Appetite: full, hungry

Attributes of an object also can include other information about its state. For example, we can have features for my class's attitude (enraged or calm) or its current health (alive or dead).

Attributes are defined by variables; in fact, we can consider attributes to be similar to global variables for the entire object. Because each instance of a class can have different values for its variables, each variable is called an instance variable. Instance variables define the attributes of an object. The class defines the type of the attribute, and each instance stores its own value for that attribute.

Each attribute, as the term is used here, has a single corresponding instance variable; changing the value of a variable changes the attribute of that object. Instance

variables can be set when an object is created and stay constant throughout the life of the object, or they can change as the program runs.

Class variables apply to the class itself and to all of its instances. Instance variable values are stored in the instance; class variable values are stored in the class itself.

1.5 CREATING A CLASS

A class is a template that describes the data and behavior associated with instances of that class. When we instantiate a class we create an object that looks and feels like other instances of the same class. The data associated with a class or object is stored in variables; the behavior associated with a class or object is implemented with methods. In the Java language, the simplest form of a class definition is

```
class name
    {
     . . .
    }
```

Now we will give the codes of a class then analyze its variables and its structure.

```
   class  departments
{String deptname;
   public String getdeptname(String name)
    {  if(tfdept.getText.equal("Com"))
         deptname="Computer Engineering";
       if(tfdept.getText.equal("Civ"))
         deptname="Civil Engineering";
       return deptname;
    }
  public void setdeptname(String name)
   { deptname=name;
   }
}
```

This class has the name departments, and has two methods one is getdeptname() and the other is setdeptname(). The first method reads the values in a TextField and checks the data in it. If data is equal to Com returns Computer Engineering, if equal to Civ returns the department name as Civil Engineering. The second method sets the department name. User can call this method anytime and change the department name as. Of course an instance of this class must be initialized before to be able to do that. e.g.:

```
deptinstance.setdeptname="Mechanical Engineering";
```

1.6 CLASS METHODS

Class methods are like function but within in a class. When a class instance is created automatically its methods are too. Therefore we can call a class's methods using the following syntax:

```
classname.classmethod();
ie. currentuser.getuser();
```

Here currentuser is an instance of Currentuser and getuser() is a method of that class that returns the identification of present user. The code for getuser is shown below:

```
public String getuser()
    { try{
        filereader=new FileReader("curuser.class");
            br=new BufferedReader(filereader);
            usernormal=br.readLine();
            }
    catch(Exception e){}
    return usernormal;
    }
```

1.7 MAIN METHOD

Every single application must have one main () method. Otherwise we will not be able to run our program. The main method as it's name reveals, is the main or the primary method with which we run the application.

```
public static void main(String args[])
 {
   Mymain main=new Mymain();
   main.show(true);
 }
```

This code segment creates an instance of Mymain class and sets it to visible status. Without this we would not be able view our output program.

1.8 CONSTRUCTORS

When we create an object for the first time it is quite to initialize all the variables within that class. To avoid this hardship we set them up in the constructors of the class. The constructor has the same name as Class and constructor we assign the class variables to their initial position. For example:

```
   public class Analogies extends JInternalFrame implements
ActionListener
   {JButton,butstart,butquit,butnext,butprev,butconfirm;
```

```
        ...
boolean retake=false,quit=false,viewresult=false;
   Currentuser currentuser=new Currentuser();
   Analogies()
   {super("Analogies Test20Qs",false,false,false,false);
    Container container=getContentPane();
          ...
    butstart=new Jbutton("Start);
    this.setSize(510,220);
  setVisible(true);
    }
}
```

As you can see Analogies class has a constructor with the same name in this constructor we have assigned variables to their initial position. Thus when we invoke class Analogies by convention the constructor will create and add them to frame.

1.9 CREATING AN INITIALIZING OBJECT INSTANCES

A class at the same time is an object. We have created a sample class in Defining a class with the name MaskUnmask. After creating this class now we can create instances of this class(now becomes object) anywhere in the program. Let's give the syntax of creating an object:

```
classname variablename=new classname();
```

This statement will create an instance of classname and appoint it to variablename. Here is the code to create an instance of MuskUnmask class:

```
MaskUnmask maskunmask=new MaskUnmask();
```

By writting this code we activate the class MaskUnmask and all its method. Or

```
Currentuser currentuser=new Currentuser();
```

1.10 USING AN INSTANCE METHOD OR VARIABLE

In the creating object instances we have created an instance for Currentuser class and labeled is ass currentuser. And by the way Currentuser has a method that returns the name and family name of the present user.If we say:

```
currentuser.getuser();
```

We will get the name and the surname of current user, so by creating an object instance we can reach its methods and variables that avoid us programmers to write less

code but availability to reuse them, as we have done in this program. To use an instance variable.

```
currentuser.usernormal="Muslum Senel" ;
```

This will appoint the String type usernornal the vaule of "Muslum Senel

CHAPTER TWO: JAVA FUNDAMENTALS

2.1 WHAT IS JAVA?

This project has been brought about using Java language therefore, I do find it needful to give a brief coverage on the Java Language, its background and the APIs that are available in the version that I have worked with.

Java was developed at Sun Microsystems. Work on Java originally began with the goal of creating a platform-independent language and operating system for consumer electronics. At the first phase, they thought to extend C++ and thus to create a platform-neutral language but later on as their work got ahead, they realized that it would be better to design a new Language instead of improving it.

What we know today as Java is both a programming language and an environment for executing programs written in the Java language. Unlike traditional compilers, which convert source code into machine-level instructions, the Java compiler translates Java source code into instructions that are interpreted by the runtime Java Virtual Machine (JVM). So Java is an interpreted language. That is why sometimes Java is named as Platform and sometimes and a language. In fact Java has both capabilities.

Java can be considered as both a general-purpose language and also an internet language. For internet purposes we use Java applets which are very useful and secure on the net. Applications are the Java programs that are written for general-purposes and run directly on Java platform. So our project can be classified into Application class.

2.2 FEATURES OF THE JAVA LANGUAGE

We have covered the background of Java above and now I will discuss some features that make Java an appealing and mostly used language very briefly. I will emphasize on one of them which is Object Oriented Programming approach in detail in upcoming section because in this program I have used this approach or Java is based on that approach.

2.2.1 Simple

The first feature of Java is its simplicity. It does look like other languages in syntax and that makes it easier for programmers to code in Java. Another thing that makes Java simple is that, it has only three primitive data types, integer, Boolean and arrays all other things are classes and therefore objects.

2.2.2 Object-Oriented

This is probably the most important characteristic of Java language. Java is an object oriented programming language because in Java everything is represented as an object and whenever you want to make use of it you should first initiate it and then use it. We are going to discuss this feature in detail later on. So that is enough for now about 0bject oriented side of Java.

2.2.3 Architecture-Neutral

This is another important attribute of java. The application written in Java can run on any platform. And this property of Java is called platform-independent. I wrote this project in Windows platform but it will work on any platform.

2.2.4 Multithreaded

Writing a computer program that only does a single thing at a time is a constraint that we face in most programming languages. With Java, we no longer face this limitation. If we have two things to be done at the same time then we could use multithreads that work synchronized.

2.2.5 Dynamic

Java is a dynamic language that is it can extend itself during execution. For example suppose we created a class but we do not need it in the running phase of program so the Java Virtual Machine will not link it to domain class but create it in the of execution and this makes the execution process faster.

2.3 JAVA BASICS

2.3.1 Variables

Every object needs variable to store their states in. Every time we want to use a variable in our application we should explicitly provide a name for it based on defined principles such as not starting with a numeric value. And we also need to specify a data type for that variable which will hold values of that type only. This is how we do this variable declaration and type assigning.

```
char keyread = 'S';
boolean status = true;
int mynumber;
String  name="muslum";
```

2.3.2 Data Types

Every variable must have a data type. A variable's data type determines the values that the variable can contain and the operations that can be performed on it. The following chart gives data types and details.

Keyword	Description
byte	Byte-length integer
short	short integer
int	integer
long	long integer
float	Single-precision floating point
double	Double-precision floating point
char	defines a single character
boolean	either true or false

2.3.3 CREATING AND USING ARRAYS

Arrays define a set of dynamic cells that temporarily holds the predefined size and type of variables. Here's a simple program that creates the array, puts some values in it, and displays the values.

```
  int[] anArray;  // declare an array of integers
anArray = new int[10]; // create an array of integers
   // assign a value to each array element and print
for (int i = 0; i < anArray.length; i++)
 { anArray[i] = i;
   System.out.print(anArray[i] + " ");}
```

2.3.4 CONTROL FLOW STATEMENTS

When we write a program, we type statements into a file. Without control flow statements, the interpreter executes these statements in the order they appear in the file from left to right, top to bottom. We can use control flow statements in our programs to conditionally execute statements, to repeatedly execute a block of statements, and to otherwise change the normal, sequential flow of control.

2.3.4.1 The Switch Statement

We use the switch statement to conditionally perform statements based on an integer expression. A simple program that uses switch can be like:

```
int depts;
switch (dept)
{case 1:System.out.println("Computer");break;
case 2:System.out.println("Civil"); break;
case 3:System.out.println("Mechanical"); break;
case 4:System.out.println("Electrical"); break;
}
```

2.3.4.2 Exception Handling Statements

The Java programming language provides a mechanism known as exception to help programs report and handle errors. When an error occurs, the program throws an exception. That is, the normal flow of the program is interrupted and that the runtime environment attempts to find an exception handler, a block of code that can handle a particular type of error. The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program. Three statements play a part in handling exceptions:

The try statement identifies a block of statements within which an exception might be thrown.

The catch statement must be associated with a try statement and identifies a block of statements that can handle a particular type of exception. The statements are executed if an exception of a particular type occurs within the try block.

The finally statement must be associated with a try statement and identifies a block of statements that are executed regardless of whether or not an error occurs within the `try` block. Here's the general form of these statements:

```
try {statement(s)}
catch (exceptiontype name)
{statement(s)}
finally {statement(s)}
```

2.3.4.3 The For Statement

The for statement provides a compact way to iterate variables for a range of values. The general form of the for statement can be expressed like this:

```
for (initialization; termination; increment)
{
   statement
}
```

The initialization is an expression that initializes the loop. The termination expression determines when to terminate the loop. This expression is evaluated at the top of each iteration. When the expression evaluates to false, the loop terminates. Finally, increment is an expression that gets invoked after every iteration through the loop. Example:

```
for(int i=0;i<11;i++)
System.out.print("Numbers:"+i);
```

This will output all numbers from 0 to 10.

2.3.4.4 The If-Else Statements

This is an important statement because even in the very simple application we have cases where we should prefer only one and for such cases we use if else statement. So the *if* statement enables us to execute some statements based on some criteria. A simple if-else statement and the syntax are:

```
if (condition)
{ statement(s)}
```

Example:

```
if(deptno==111)
deptname="Computer Engineering";
else
deptname="Not Computer Engineering";
```

2.3.4.5 The While and Do-While Statements

You use a while statement to continually execute a block of statements while a condition remains true. The general syntax of the `while` statement is:

```
while (expression)
 {
  statement
 }
```

First, the while statement evaluates expression, which must return a boolean value. If the expression returns true, then the while statement executes the statement(s) associated with it. The while statement continues testing the expression and executing its block until the expression returns false.

```
String copy = "Copy this string until the letter 'g'.";
StringBuffer show= new StringBuffer();

  int i = 0;
  char c = copy.charAt(i);

  while (c != 'g') {
      show.append(c);
      c = copy.charAt(++i);
  }
  System.out.println(show);
```

The value printed by the last line is: Copy this strin.

The Java programming language provides another statement that is similar to the while statement the do-while statement. The general syntax of the do-while is:

```
do {
    statement(s)
   } while (expression);
```

Instead of evaluating the expression at the top of the loop, `do-while` evaluates the expression at the bottom. Thus the statements associated with a `do-while` are executed at least once.

2.4 SWING GUI COMPONENTS

Here in this section we will talk about the Swing UI components, How to create and use those components with simple codes taken from our program. The components that are not used in our project will not be discussed here.

2.4.1 JLabel

With the JLabel, we can display unelectable single line text or images. If we need to create a component that will displays a string or an image we can do so by using JLabel.

The following code will display the message"Click the start button to take the test".

```
JLabel lcommand=new JLabel("Click start button to take the test");
```

Sometimes we may want to screen multiple-line text to do so we should embed HTML codes in a label. I used this approach in the About frame of my application and the sample code is given right below.

```
JLabel l=new JLabel("<html><li> Graduation Project <br><li>
Copyright © 2005 Muslum SENEL <br><li> Version V1.0"+ "<br><li> All
rights reserverd <br> <li> Freeware product"+"<br><li> <a href
mailto:msenel2000@yahoo.com>"+ "msenel2000@yahoo.com</a></html>");.
```

2.4.2 JTextField

This is the Swing component that lets the user to type small amount of text. The text entered to this field can be edited. The ActionListener Interface can be associated with this to take certain step when user hits enter key after inputting some test. Here is the code portion to create a JTextField and couple it with a Listener.

```
   JTextField tfname=new JTextField(10);
   JTextField tfsname=new JTextField(10);
tfsname.addActionListener(this);
```

To get the data entered into the JTextField we simple use getText () method as:

```
tfsname.getText();
```

This will retrieve the content of the text field.

2.4.3 JButton

Button is the very basic component of Swing that generates an event when hit with mouse. With JButton we use an ActionEvent method that deals with the action performed by any JButton object in the container.

Here is the code to create a JButton object, associate the ActionListener interface so to perform some action in the case of hitting it and adding it to main window.

```
JButton button=new JButton("Click");
button.addActionListener(this);
This.getContentPane().add(buttton);
public void actionPerformed(ActionEvent ae)
{
 if(ae.getActionCommand().equal("Click"))
   {.......}
}
```

This program piece creates a button with the Start label and associates a listener to it and gives the method ActionListener that button will use when clicked.

2.4.4 JRadioButton

Radio buttons are groups of buttons in which, by custom, only one button at a time can be selected. They are used to provide some options to the user to select only one among them as in our case the user will pick only one answer at a time. Of course by default this is not possible, only by adding multiple JRadioButtons to a Swing component called ButtonGroup this task can be accomplished.

Now let us give the codes to create five JRadioButtons and add them to a ButtonGroup to pack them.

```
        JRadioButton rb1,rb2;
ButtonGroup bg;
rb1=new JRadioButton();
   rb1.addActionListener(this);
   rb2=new JRadioButton();
   rb2.addActionListener(this);
bg=new ButtonGroup();
   bg.add(rb1);
   bg.add(rb2);
```

2.4.5 JMenu

A menu provides a space-saving way to let the user choose one of several options. Unlike other components they are not placed in any container instead they are placed at the top of window. There are two types of JMenus, they are Pop up and Menu bar here in this application we utilized the latter one.

This code segment shows how to create a Menu bar and set it active in a JFrame. This code is taken from my application but here I will abridge it to make simpler to comprehend.

```
JMenuBar menubar=new JMenuBar();
JMenu  menuone,menutwo;
JMenuItem item1,item2,item3;
menuone=new JMenu("   Menu1   ");
menutwo=new JMenu("   Menu2   ");
item1=new JMenuItem("   Item1");
item1.addActionListener(this);
item2=new JMenuItem("   Item2");
item2.addActionListener(this);
item3=new JMenuItem("   Item3");
item3.addActionListener(this);
menuone.add(item1);
menuonel.add(item2);
menutwo.add(item3);
menubar.add(menuone);
menubar.add(menutwo);
setJMenuBar(menubar);
```

2.4.6 JPanel

The JPanel provides general-purpose containers to hold lightweight components such as JLabel, JButton and etc. This is an abstract class that we can not see it but when we add components to it we can feel its importance. It is much like a frame in holding components point of view.

Now let's discuss what we can do with JPanels by sample codes, starting with creating a Panel Object.

```
JPanel pbuts=new JPanel();
```

We can assign Layout to the JPanels to make attractive UIs.

```
pbuts.setLayout(new BorderLayout()); or
pbuts.setLayout(new GridLayout(5,1));
```

To add components to the JPanel created beforehand we just use the method add().

```
pbuts.add(butstart,BorderLayout.CENTER);
pbuts.add(butnext,BorderLayout.NORTH);
```

2.4.7 Separators

The Separator class provides a horizontal or vertical dividing line or an empty space. It is most commonly used in menus and tool bars. In our application we used to divide the JMenus and the use of Separators is as follows

```
menufile.add(menuverbal);
menufile.add(menuquantative);
menufile.addSeparator();
```

The interesting thing about Separator is that we can use it without creating by just invoking add () method.

2.4.8 JInternal Frame

With the JInternalFrame class we can display a JFrame-like window within another window. But it is not possible to display a JInternal Frame within a JFrame to use them we should create a Desktop Pane and then insert them onto Window. The entire test frames are JInternalFrame because we have used JDesktopPane in our main class and therefore they are to be JInternalFrame. The code gives below show how to use JInternalFrame and add it to desktop Pane.

```
   public class Antonyms extends JInternalFrame implements
ActionListener
    {......
      Antonyms()
         {super("Antonyms Test20Qs",false,false,false,false);
         ....
      }}
      antonyms=new Antonyms();
         desktop.add(antonyms);
         antonyms.setVisible(true);
         try{ antonyms.setSelected(true);}
         catch(java.beans.PropertyVetoException e){}
```

2.4.9 JFrame

A frame is a window that typically has decorations such as a border, a title, and buttons for closing and iconifying the window.

Here we will create a frame and add components to that frame.

```
JFrame mainframe=new JFrame("Main Window");
Mainframe.add(butstart);
Mainframe.add(labelname);
```

2.4.10 JDesktop Pane

With this component we can add internal frames. In this project all the test classes are extended to JIntenal Frame component so to be able to use them we should have JDesktop Pane. Here is how to create one and add an internal frame:

```
public JDesktopPane desktop=new JDesktopPane();
      public void run()
        {  datanalysis=new DataAnalysis();
             desktop.add(datanalysis);
             datanalysis.setVisible(true);
       try{
         datanalysis.setSelected(true);
          }
      catch(java.beans.PropertyVetoException e)
       {}
        }
```

CHAPTER THREE: DATABASE & SQL BASICS

In this chapter we will contemplate over database, specifically relational database and Structured Query language and try to explore them, if possible codes will be presented to make it simpler and charming. Of course since Microsoft Access has been used in this application a short tour will be available and also an answer to what is Open Database Connectivity will be presented as well.

3.1 DATABASE SYSTEMS

Database is the collection of related data. Databases are very important, because keeping a lot of data in files can be difficult and there is a possibility of loosing the track relations between retained data. Therefore databases are used not only to store huge number of data but also to arrange them in such a way that will be easy-access and reliable links with one another. There are many database models in today's computing world but the most effective and the one that we used in this application is Relational Database. In succeeding sections we will discuss it very briefly.

3.2 RELATIONAL DATABASE

Relational database management systems (RDBMSs) are the most popular type of database in the market today. There is an absolute set of criteria that defines a relational database; the following informal description is probably more useful:

- Data consists of records stored in tables as rows.
- Each record includes a fixed set of fields, with each field corresponding to the columns of the table.
- One column must be a primary key—a required and unique value—so each record can be exclusively located.
- Views—alternate ways of looking at a table or a set of tables—must include support for inserting, updating and
- Deleting the appropriate data in the underlying table or tables.
- The database must support null—an unknown value not equivalent to zero or a blank.
- A high-level relational language—not necessarily, but usually Structured Query- Language (SQL)—must be provided to support data definition, data manipulation, and database administration.

- Data constraints must be enforced by the database and cannot be avoided.

RDBMSs are the standard today for a number of simple reasons: They are well studied and well understood. They scale well. Companies such as Oracle have invested extraordinary amounts of time and money in developing versatile, flexible, and powerful products. RDBMSs are so well established that we really need to change the question from, "What is the best way to provide persistence in an object-oriented system?" to "What is the best way to use a relational database to provide persistence for an object-oriented system?" That is, given that relational databases are the industry standard (and for good reasons), what is the best way to bridge the impedance mismatch?

One way to bridge the gap between an object-oriented system and a relational database is to use a mapping layer. Because this layer creates a degree of independence between the database and the rest of the system, it can protect each from the effects of changes in the other. This can improve the maintainability and reusability of the system, both key goals of object-oriented design.

2.3 MICROSOFT ACCESS:

In this application we used Microsoft Access as our database system. The reason to use Access was that we did not have many tables and queries and therefore it would be inefficient if we were to use Oracle or any other very-high Database system.

Microsoft Access gives us to create tables and do any manipulation on them and to create queries and retrieve data from any combination of pre-defined user tables.

For this program we have created(manually) nine tables in Access seven of them hold the test attributes for different test that we have prepared and the other two hold the user information and their test status, whether any test is taken or not.

2.4 ODBC

ODBC stands for Open Database Connectivity. Microsoft is the vendor of this configuration to connect any Database API in Windows platform. For this application we did base the Java Database API with Open Database Connectivity to talk to database and get data from database.

2.5 SQL&BASIC SQL STATEMENTS

SQL is a special-purpose language, sometimes described as a relational language, which can be used with a database for a number of different purposes. SQL can be considered a standard language, with some qualification. There are two significant standards, SQL-92 and SQL-99. Most database vendors, including Oracle, are largely compliant with SQL-92. SQL-99 greatly extends the scope of SQL, introducing new features such as persistent modules and multidimensional analytical capabilities. Most database vendors are largely compliant with the core features of SQL-99, but not the new features. This core compliance allows them to claim that they are compliant with SQL-99, even though this is virtually the same as being compliant with SQL-92. To confuse things further, most vendors, including Oracle, also have extensions to standard SQL, which are sometimes unavoidable. SQL commands can generally be grouped into a number of general categories, according to their purpose:

- Data Definition Language (DDL), used for defining tables and the relationships between the tables and the data in them.
- Data Manipulation Language (DML), used for adding, retrieving, modifying, and deleting data.
- Transaction control commands, used to group set of DML statements into a single unit of work; used to ensure data integrity.
- Database administration commands.
- SQL/PSM, used for writing procedural programs using persistent stored modules.

The first two categories of commands, DDL and DML, are the core SQL commands used for defining data models and for storing and querying data. These are the commands we will be concentrating on in this chapter. These commands are generally implemented in a standard way by vendors. We will generally adhere to the features defined in the SQL-92 standard for DML and DDL commands.

Transaction control commands are used to isolate groups of commands so that they can be treated as a unit. SQL-92 defines very basic support for transactions; there is no command to mark the start of a transaction, for example. We will largely use this basic model, but we will also consider SQL-99 features as implemented in Oracle.

Database administration commands are largely vendor-dependent. In Oracle, they are used to maintain the physical structure of the database, create users, grant rights to users, create database policies of various sorts, etc. Database administration is a large topic in its own right and well beyond the scope of this book. We'll cover only a few essential commands incidentally, when we set up a sample database.

The last category is an optional part of the SQL standard to allow procedural programming. It defines persistent stored modules that provide control flow statements and bind variables, similar to Oracle's PL/SQL. This standard was accepted in 1996, but since then, vendors have begun to converge on Java stored procedures.

Structured Query Language (SQL) is an ANSI standard computer programming language used to query relational databases.
In this application we sometimes used SQL statement to interact with tables; these are basically Select, Update, Delete, Insert and create table statements. In this way we will review their use briefly.

2.5.1 Select Statement

This statement is used to get data from a specified table in a specific Database: the syntax and a sample are illustrated right below:

```
Select columnname1,columnnae2,……,columnnamen from tablename
[where condition];

String queryview="select qn,youranswer,correct,status from
table"+currentuser.getuser()+"analogy";
```

2.5.2 Delete Statement

Delete statement can be used to delete a row from table with a condition, or if used with without condition can delete the all rows in table. This statement is used in the form:

```
delete * from table where [condition];

String deletetest="delete * from table"+
currentuser.getuser()+"analogy";
```

2.5.3 Insert Statement

To add new a row of data to a table, Insert statement should be used. In order to insert data into a table the number of columns in the table and in the insert statement must match. That is if we have five columns in TableA then our insert statement also must have five parameters. Insertion into a table is done like:

```
insert into tablename values(value1,value2...valuen) [where
condition];
String submitresults="insert into
table"+currentuser.getuser()+"analogy
values("+qn+",'"+youranswer[qn]+"','"+correct[qn]+"','"+status[qn]+"')
";
```

2.5.3 Update Statement

This SQL statement is used in the times we need to change the content of a table partially. In other words when we need to update our table we use Update command. Because sometimes we need to modify only one column of a specific row and in this case update is very beneficial.

```
update tablename set column1=new value1,column2=new value2,.... where
[contition]
String queryq="update usersandtakentests set sentence=true where
user='"+currentuser.getuser()+"'";
```

2.5.4 Create Table

Sometimes we need to create table just when it is needed. For example in this application when the user takes a test and want to save his results then we create a table with the name of user and test so to distinguish from other users and other tests. To code to do so is as:

```
String createtable="create table
table"+currentuser.getuser()+"problem(qn number,status text)";
db.connecttodatabase(createtable);
```

CHAPTER FOUR: JAVA DATABASE CONNECTIVITY

So far we have dealt with Graphical User Interface components and object oriented and database concepts in prior chapter. After creating our UI and objects now we are ready to connect to database and manipulate data. Because our program retains its question in database and our test results will be stored in database therefore we need to take a look at Database concepts.

This chapter will include how to establish a connection between java database connectivity framework and open database connectivity. And later will cover all necessary steps to make this connection possible with codes from various classes used in this application.

4.1 THE BASICS OF JDBC

The Java Database Connectivity Application Programming Interface (API) is an API currently being designed by Sun Microsystems that provides a Java language interface to connect to database Systems. The most visible implementation of this is Microsoft's ODBC (Open Database Connectivity). This API defines a common SQL syntax and function calls that can be used by developers to send SQL commands to and retrieve data from SQL databases. ODBC-enabled applications make use of database drivers installed on the system that allow applications to communicate to a vendor's database.

The JDBC API is expressed as a series of abstract Java interfaces within the `java.sql` package that exists in every JSDK release after 1.1 here are the most commonly used interfaces:

- `java.sql.DriverManager`-Manages the loading and unloading of database drivers from the underlying system.
- `java.sql.Connection`-Handles the connection to a specific database.
- `java.sql.Statement`-Contains an SQL statement to be passed to the database; two subtypes in this interface are the `PreparedStatement` (for executing a precompiled SQL statement) and the `CallableStatement` (for executing a database stored procedure).
- `java.sql.ResultSet`-Contains the record result set from the SQL statement passed to the database.

4.2 SETTING UP DATABASE

To set up a database first of all we should have some data in DBMS.as we mentioned in above we have nine tables ready in gretest.mdb file to process data manipulation. To reach from Java to those table to retrieve or store date we will use java.sql.* package and the interfaces in to connect, execute and project data.

4.3 ESTABLISHING CONNECTION

The first thing we need to do is establish a connection with the DBMS we want to use in this case is Microsoft Access and again Microsoft's Open Database Connectivity. This involves two steps: (1) loading the driver and (2) making the connection.

4.3.1 Loading Drivers

Loading the driver or drivers we want to use is very simple and involves just one line of code. As we said before we will use JDBC: ODBC Bridge and therefore the code to load this driver as:

```
try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
catch(Exception e){e.printStackTrace();}
```

Loading driver throws an exception in case it fails to load driver. When you have loaded a driver, it is available for making a connection with a DBMS.

4.3.2 Making a Connection

The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea:

```
Connection con =
DriverManager.getConnection(url,"myLogin","myPassword");
```

In our case the code is:

```
connection=DriverManager.getConnection("jdbc:odbc:test","admin",
"20010811");
```

And of course connection is an instance of Connection Interface of java.sql package.

getConnection() method has three parameter. The first parameter specifies the url,that is the connection bridge plus the source name for database. Connection bridge depends on the DBMS system and the source name is just any name give by the programmer. The second parameter is the user name for Database and the last one is the password assigned to Database.

The connection returned by the method `DriverManager.getConnection` is an open connection we can use to create JDBC statements that pass our SQL statements to the DBMS. In the previous example, `connection` is an open connection, and we will use it in the examples that follow.

4.4 THE DRIVER INTERFACE

A `Driver` is essentially a `Connection` factory. The `DriverManager` uses a `Driver` to determine whether it can handle a given URL. If one of the `Drivers` in its list can handle the URL, that `Driver` should create a `Connection` object and return it to the `DriverManager`. Because an application only indirectly references a `Driver` through the `DriverManager`, applications are rarely concerned with this interface.

4.5 THE DRIVERMANAGER CLASS

The DriverManager class of java.sql is used to manage the JDBC drivers that are installed on our system. These drivers may be installed by setting the jdbc.drivers system property or by loading the driver class using the forName() method of the Class class as we did under Making Connection heading of this chapter.

DriverManager provides three form of getConnection() methods according to the number of parameter neede. If you recall from previous section we prefered the second method.

```
getConnection(String url)
getConnection(String url,String userID,String password)
getConnection(String url,Properties arguments)
```

4.6 THE STATEMENT STATEMENT

Now that the connection has been established with the database, a statement needs to be created so that it can be passed to the database for processing. This is done by calling the `connection` class's `createStatement()` method. Here's an example of this:

```
Statement statement = connection.createStatement();
```

4.7 EXECUTING A QUERY

Using the statement we have just created, an SQL query can be executed by using the statement's `executeQuery()` method. This method returns a `ResultSet` object so that the query's results can be examined. Here's a simple example:

```
String queryselect=" "select qn,youranswer,correct,status from
table"+currentuser.getuser()+"sentence";
ResultSet result = statement.executeQuery(queryselect);
```

4.8 UPDATING TABLES

Updating a table is different than executing a normal SQL select or delete statement. To update a row in a table the executeUpdate() method of statement is used.

```
String queryq="update usersandtakentests set problem=true
where user='"+currentuser.getuser()+"'";
statement.executeUpdate(queryq);
```

4.9 RETRIEVEING DATA FROM A RESULTSET

Now we have load our drivers established a connection and now we are ready to retrieve data using ResultSet interface.

```
ResultSet resultset  //instance creation.
String query="select question,a,b,c,d,e,answer,qnumber from Analogy
where qnumber="+questionnumber;
```

For example to execute the String type query give above all we have to do is just type:

```
resultset=statement.executeQuery(query);
```

This statement will hold all data fetched from database by SQL Select statement. But still we are not able top view this data. To do this we need to use the next () method

of ResultSet and append or set them to labels to visualize. The next topic will cover the next () method and how to display them on the screen for.

4.10 USING THE METHOD NEXT ()

The variable resultset which is an instance of `ResultSet` , contains the rows of questions, answer choices an answers from table Analogy shown in the resultset example above. In order to access those values in each row and retrieve them we need to use the same data types. The method `next` moves what is called a cursor to the next row and makes that row (called the current row) the one upon which we can operate. Since the cursor is initially positioned just above the first row of a `ResultSet` object, the first call to the method `next` moves the cursor to the first row and makes it the current row. As we call next method the cursor moves down until it reaches the bottom row table. An example is as:

```
while(resultset.next())
 {String x=resultset.getString("question");
     qn=resultset.getInt("qnumber");
     lquestion.setText(""+qn+")   "+x);
     String a1=resultset.getString("a");
     rb1.setText(""+a1);
     String a2=resultset.getString("b");
     rb2.setText(""+a2);
     String a3=resultset.getString("c");
     rb3.setText(""+a3);
     String a4=resultset.getString("d");
     rb4.setText(""+a4);
     String a5=resultset.getString("e");
     rb5.setText(""+a5);
     ans=resultset.getString("answer");
     correct[qn]=ans;
                          }
```

We use the next () method within the while loop so to continue accessing every row that has been transferred to resultset.

4.11 THE GETXXX METHODS

We use the getXXX method of the appropriate type to retrieve the value in each column. For example, the first column in each row of ResultSet is question, which stores a value of SQL type `VARCHAR`. The method for retrieving a value of SQL type `VARCHAR` is `getString()` .now lets give the syntax and samples of using this method for various kinds of data types:

To retrieve string type:

```
resultset.getString(columnname);
String a1=resultset.getString("a");
```

To retrieve integer type:

```
resultset.getInt(columnname);
qn=resultset.getInt("qnumber");
```

To retrieve float type:

```
resultset.getFloat(columnname);
```

To retrieve boolean type:

```
resultset.getBoolean(columnname);
boolean problemin=resultset.getBoolean("problem");
```

CHAPTER FIVE: A JAVA APPLICATION: COMPUTER-BASED EXAMINATION

In the previous four chapters we gave theoretical things that were necessary to understand this main chapter. We covered GUI elements, object creation and class definitions and then we gave basics of Database and structured Query Language and how JDBC: ODBC bridge is used to connect Java applications to Database system.

Now we will design a software application for a Computer-based examination. This test will have two sections as Verbal and Quantitative, and eight tests in total. All the questions and answers to those tests are kept in procreated tables using Microsoft Access. Therefore we will consider those tables as created and continue designing accordingly.

5.1 THE CLASS RELATIONS

Before designing this software we should specify that the following classes will be created and the correlation between those classes is as in Figure5.1. In this application class Users is the class that contains main method as discussed in Main method (chapter2) and therefore is our main class.

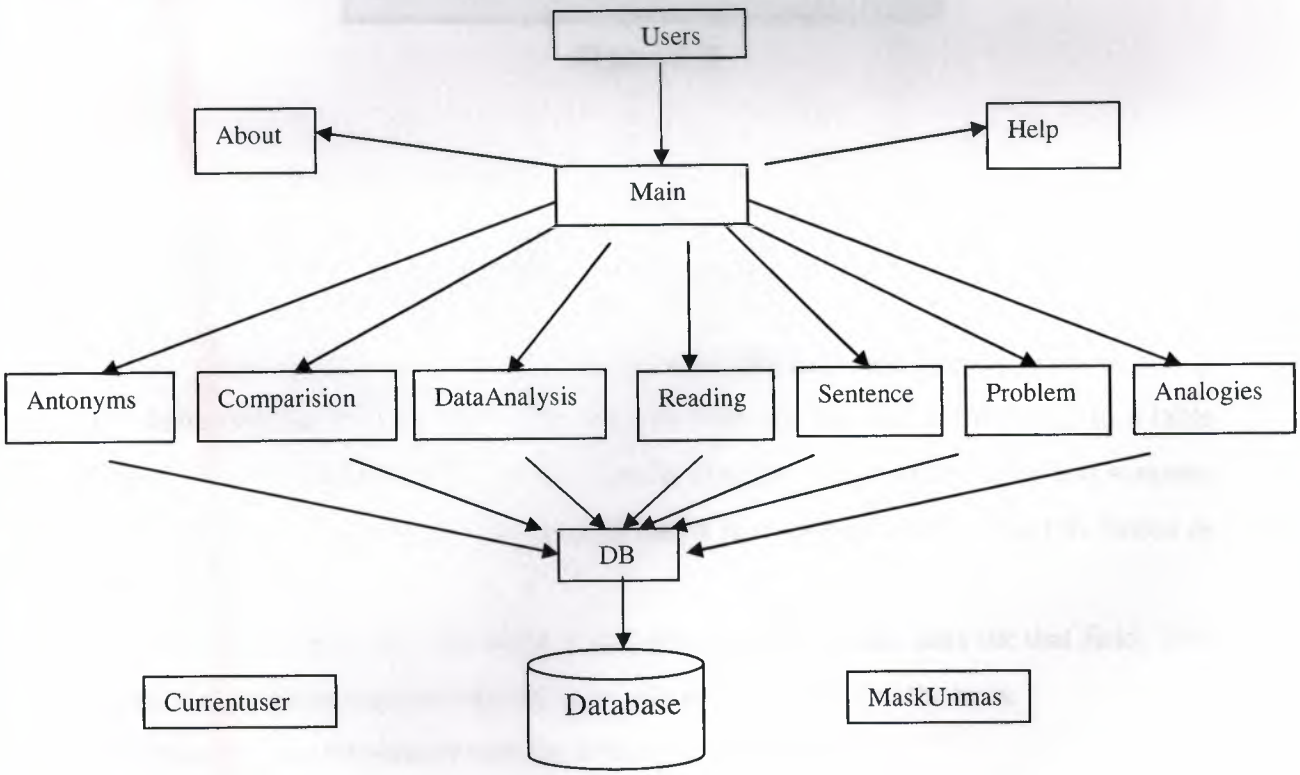

Figure 5.1

5.2 THE COMPONENTS OF COMPUTER BASED EXAMINATION

To develop the software discussed above we have to create a collection classes. Those class are; Users, main, Antonyms, Analogies, Reading, Sentence, DataAnalysis, Problem and comparison. In the succeeding sections we will analyze those classes one by one and give every detail about them.

5.2.1THE USERS CLASS

Let's begin designing with the main class for this application. To create the main class we have to use GUI components as discussed in Chapter2 Swing GUI components. To do this we will create two instances for each of JButton, JLabel, JTextField

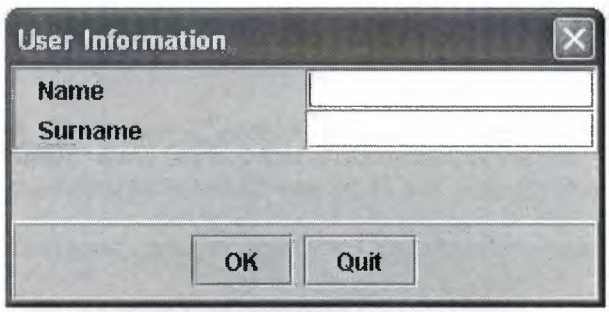

Figure 5. 2

```
JButton butok,butquit;
JLabel lname=new JLabel("    Name"),lsname=new JLabel("Surname");
JTextField tfname=new JTextField(10);
JTextField tfsname=new JTextField(10);
butok=new JButton("OK");
```

Here we have to check whether the user info provided already exists or not. If it is first time that user is logging in then we will add the user information to a table named Users and set all taken tests to false for that user and also the name and surname of that user will be printed on the top of Window that appears after hitting OK button as seen in the figure 5.2.

But if any fields are left blank it will prompt user to enter data for that field. That is without supplying enough info the user will not be able get to the tests.

This process is accomplished with the following code portion:

```
   if(key.equals("OK")|| ae.getSource()instanceof JTextField){
        name=tfname.getText();
        sname=tfsname.getText();
        user=name+sname;
        try{filewriter=new FileWriter("curuser.class");
        filewriter.write(""+name+sname);
        filewriter.close();}
        catch(Exception e){}
        String query="select name,surname from Users";
        String querynew="insert into Users
values('"+name+"','"+sname+"')";
String querytestst aken="INSERT INTO usersandtakentests
VALUES('"+user+"',"+"false,"+"false,"+"false,"+"false,"+"false,"+"fals
e,"+"false)";
```

The above codes register the user and taken tests to database to prevent double user problem and double-taking test tests. Here we write those user info to a file every time a user logs in his/her login info will be overwritten on the previous info and then as we will see later will be compared to users already in the database if it is new it will be added otherwise his/her status will be modified. And this task is done as:

```
if(!tfname.getText().equals("")&&!tfsname.getText().equals(""))
{  try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
catch(Exception e){e.printStackTrace();}
try{
connection=DriverManager.getConnection("jdbc:odbc:test","admin","20010
811");
statement=connection.createStatement();
resultset=statement.executeQuery(query);
while(resultset.next())
{  nin=resultset.getString("name");
sin=resultset.getString("surname");
if(name.equals(""+nin)&&sname.equals(""+sin))
exist=true; }
if(!exist)
{statement.executeUpdate(querynew);
statement.executeUpdate(querytestst aken);
}}
```

When user processing process is completed (either registered or updated) the user is ready to get to the test window. And task is achieved by calling an instance of Main(see Class Main) which has all seven test instances and all other relative components. Here we are crating an instance with the same properties as Main but with the name main (see Creating an Initializing Object Instances in Chapter1) and set it visible. Since we are done with this class we can dispose it.

```
main=new Main();
main.show(true);
this.dispose();
```

5.2.2 CLASS MAIN

When the user info processing finishes the Users window (figure 5.2) vanishes and instead the window shown in figure 5.3 will be apparent. Of course as we mention in (Object Initialization) we create an instance for the class Main at the end of class Users.

This window is has a DestopPane(see JDesktop Pane in Chpater2) which will hold the test that has JInternalFrame.

The following code will define instances for all subclass that we will cal in this Main program.

```
private Antonyms antonyms;
private Analogies analogies;
private Sentence sentence;
private Comparision comparision;
private Reading reading;
DataAnalysis datanalysis;
private Help help;
```

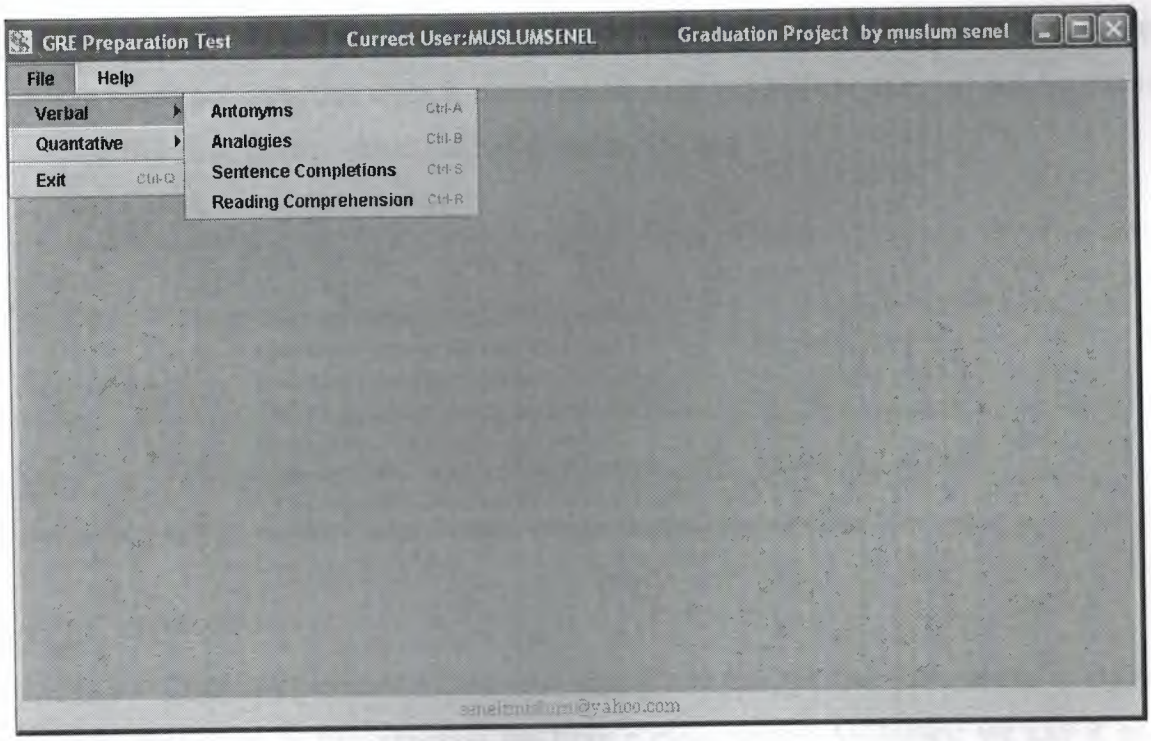

Figure 5.3

Creating the GUI for this window requires lots of codes but we wont put them all here instead segment will be provide. So lets begin with creating the menu bar (see JMenus in chapter 2).

```
menufile=new JMenu("    File    ");
menuverbal=new JMenu("    Verbal");
menuquantative=new JMenu("    Quantative");
itemantonym=new JMenuItem("    Antonyms");
itemanalogy=new JMenuItem("    Analogies");
itemdatanalysis=new JMenuItem("    Data Analysis");
itemcomparision=new JMenuItem("    Comparison");
```

Assigning shortcuts to these menu items is accomplished as:

```
itemantonym.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,Action
Event.CTRL_MASK));
itemanalogy.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_B,Action
Event.CTRL_MASK));
```

Of course these all item should be associated with the ActionListener so when they are pressed to fire an action. This is same as adding an ActionListener to a button (JButton Chapter2). To take the Analogies test that appears in the menu the user will click that menu and he/she will see the Window in figure 5.3 here we have two cases the first one is if the user is taking this test for the first time and the other is if the test has been taken before. Hence before showing a normal window the program will check the user's test status and act accordingly. If you can notice from the codes shown below when the itemanlogy fires an action it first call the `chechusers()` subclass. According to result got from this class it makes another decision if the user just has logged in for the first time or user exists but has not taken the test.

```
if(ae.getSource()==itemanalogy)
 {checkusers();
   if(!userexists||!analogydone)
     {Thread runner=new Thread(){
       public void run()
       {analogies=new Analogies();
       desktop.add(analogies);
       analogies.setVisible(true);
       try{analogies.setSelected(true);}
       catch(java.beans.PropertyVetoException e){}
       }
     };
runner.start();}
```

Codes till here are for the case that if user has not taken the test or is logged for the first time. In this situation we have a class named Analogies; we call it as it is, without making any changes in the methods or variables.

The code portion that starts with else is for the reverse case meaning either the user hasn't taken Analogies test or has registered lately. When the else part is executed we make some alterations on the initial structure of Analogies test. Because in this way

we give the user to retake it or view his outcomes, therefore GUI has to be modified somewhat. The required codes to do this mission are as below:

```
  else {Thread runner =new Thread(){
   public void run()
        { analogies=new Analogies();
          analogies.butstart.setVisible(true);
          analogies.butstart.setText("Retake");
          analogies.butquit.setVisible(true);
          analogies.butquit.setText("View Results");
          analogies.lcommand.setText("You have already taken the
test");
          analogies.lcommand.setForeground(Color.blue);
          desktop.add(analogies);
          analogies.setVisible(true);
          try{ analogies.setSelected(true);}
          catch(java.beans.PropertyVetoException e){}
         }}; runner.start();}
         }
```

To take other test all we have to do is do the same procedure for other test classes. Because they all work in the same manner and just changing the name of the object will be sufficient to get this job done.

5.2.3 CLASS ANTONYMS

The following screenshot is the window that appears when the user takes Antonyms test for the first time. It may seem simple but in reality many codes and manipulations are done while this class is instantiated and being executed. Now let's get them one by one and see how they make this whole class work so perfect.

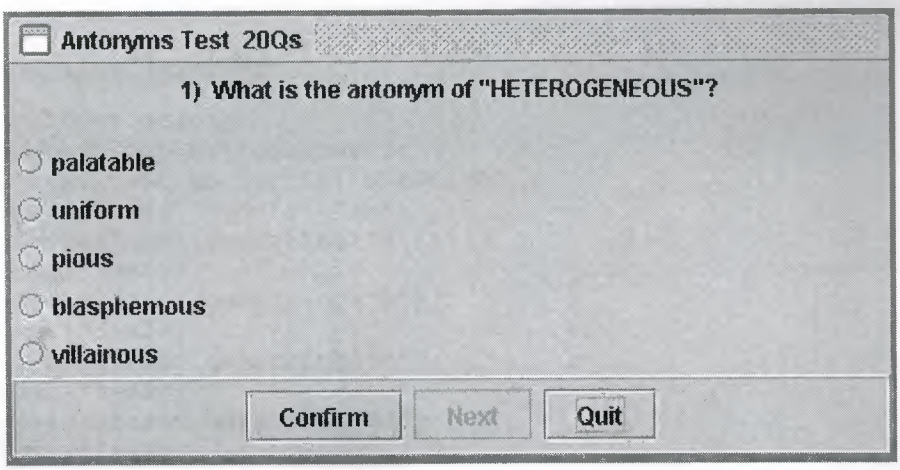

Figure 5.4

First of all we have to initialize instances of all classes that will be used in this process and these class and instantiation is give below.

```
DB db=new DB();
Currentuser currentuser=new Currentuser();
MaskUnmask maskunmask=new MaskUnmask();
```

Now let's talk about GUI components and their functions. At the top we can see the question as a Label text and right beneath are five options as radio button providing answer choices to choose from. We have used three buttons, one to quit the Antonyms test and one to move forward and the other is for confirming the answer given to a question.

The following code is for next button. When the user replays to first answer to move the next question we have to read new question and answer from Database and replace it with older one as:

```
   if(ae.getSource()==butnext)
   { String a="taciturn";
   butnext.setEnabled(false);
   butconfirm.setEnabled(true);
   questionnumber++;
   String query="select question,a,b,c,d,e,answer,qnumber from Antonym
where qnumber="+questionnumber;
   butstart.setEnabled(false);
   try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   catch(Exception e){e.printStackTrace();}
   try{
connection=DriverManager.getConnection("jdbc:odbc:test","admin","20010
811");
   statement=connection.createStatement();
   resultset=statement.executeQuery(query);
```

This portion reads the data from table Antonyms and set the label of question and five radio buttons to current data read from table with a select statement(see Basic SQL statements in Chapter3);

```
while(resultset.next())
{ qn=resultset.getInt("qnumber");
String x=resultset.getString("question");
lquestion.setText(""+qn+")   "+x);
String a1=resultset.getString("a");
rb1.setText(""+a1);
String a2=resultset.getString("b");
rb2.setText(""+a2);
String a3=resultset.getString("c");
rb3.setText(""+a3);
String a4=resultset.getString("d");
rb4.setText(""+a4);
String a5=resultset.getString("e");
rb5.setText(""+a5);
ans=resultset.getString("answer");
}}
catch(SQLException sqlex) {}
```

This will continue until all twenty questions are completed. At the end the user will have the option to see his results and success rate.

This is the case when test is taken for the first time, but what if it is being taken for the second or third time. In this case when the itemanatonyms is fired it will check the status of test and will set the label of buttonnext to "Retake" and confirm becomes "View Results". Since the actions are checked with labels if we see retake label then the action for retake will be shot.

5.2.4 CLASS ANALOGIES

The second test in the Verbal section (as seen in figure 5.3) is Analogies test. The appearance of this test after clicking the Start button is given in figure 5.5 If you pay attention to the UI you can easily realize that it is the same as UI of Antonyms but the only different is that their sizes are different. The subclasses that are imported to this Class are also the very same classes that were used in Antonyms test.

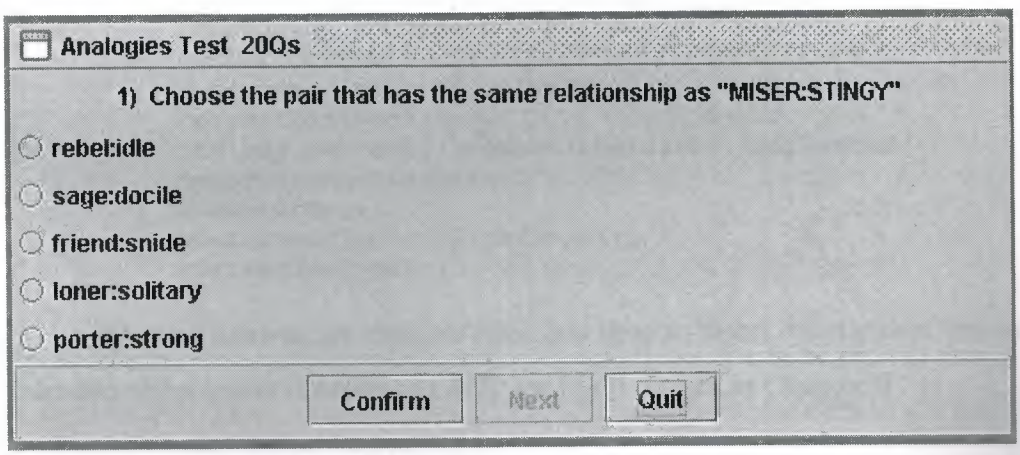

Figure 5.5

Since the User Interface is same as the one of previous we will not spend much on it in this part but instead we will concentrate on how questions are obtained from database and the right and wrong calculation of test. This process is done when the test taker hits the Confirm button. Thus here you will see the codes associated with this button.

```
if(ae.getSource()==butconfirm)
   { butconfirm.setEnabled(false);
     butnext.setEnabled(true);
```

The next button is enabled to move forward and confirm button is set disabled since the answer for this question is selected.

There are three different answers for each question but will present cases for the first and last answer choices. Not to forger that the same thing is valid for others, the only thing that differs is the radiobuttons' names.

```
if(rb1.isSelected()&&rb1.getText().equals(""+ans))
{status[qn]="Correct";
numbercorrect++;
youranswer[qn]=rb1.getText();
correct[qn]=ans;}
```

Two possible events can occur. One is when the answer and the radio characters are equals; the other is when the wrong answer is chosen.

```
if(rb1.isSelected()&&!rb1.getText().equals(""+ans))
{youranswer[qn]=rb1.getText();
correct[qn]=ans;
status[qn]="Incorrect";
numberwrong++;
}
...
if(rb5.isSelected()&&rb5.getText().equals(""+ans))
{status[qn]="Correct";
numbercorrect++;
youranswer[qn]=rb5.getText();
correct[qn]=ans;}
if(!rb5.getText().equals(ans)&&rb5.isSelected())
{status[qn]="Incorrect";
numberwrong++;
youranswer[qn]=rb5.getText();
correct[qn]=ans;
}
```

When all answer are checked then it is time to insert the status of the current into the table using insert statement of SQL (see SQL basics in Chapter3)

```
String submitresults="insert into
table"+currentuser.getuser()+"analogy
values("+qn+",'"+youranswer[qn]+"','"+correct[qn]+"','"+status[qn]+"')
";
db.connecttodatabase(submitresults);}
```

NEAR EAST UNIVERSITY
LIBRARY
1988 - LEFKOŞA

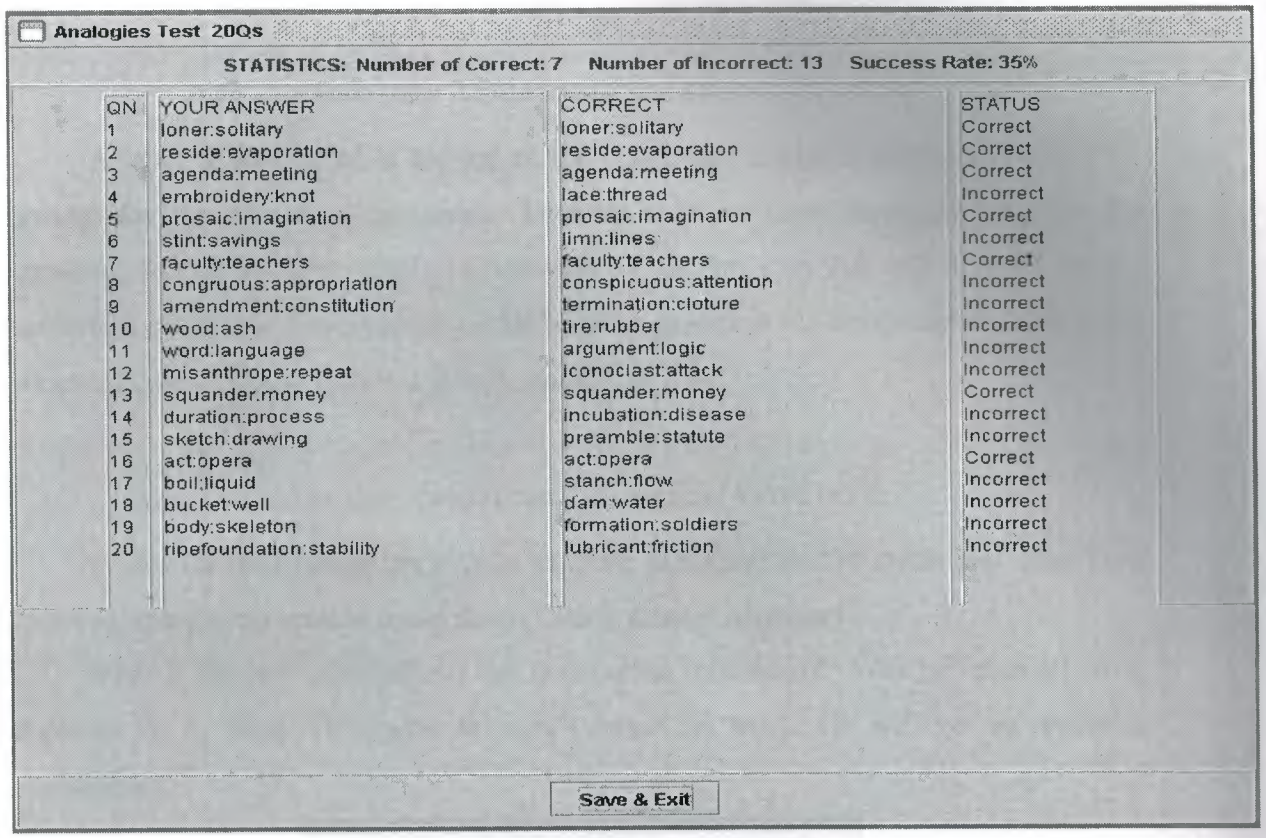

Figure 5.6

This process continues until all the questions are given an answer and when this is true the status for all questions is retrieved from database and displayed on the screen an example of the answer is shown in figure 5.6

5.2.5 CLASS DATAANALYSIS

Now we will spend some time on creating Class DataAnalyis. To create this class again we need to extend the class to JIntenalFrame(see JInternalFrame Chapter2) with:

```
public class DataAnalysis extends JInternalFrame implements
ActionListener {}
```

And create instance for classes that will be used as done in class Antonyms.

Then the graphical user Interface elements are added to this frame. There is one special thing about this class comparing to the other classes. As you could see from the screenshot for this class in figure 5.8 we have an extra image that contains a figure showing the data to be analyzed. The image seen in screenshot is added to the frame by deploying the succeeding code block:

```
Jlabel =new Jlabel();
lpic.setIcon(new ImageIcon("datapics/3.JPG"));
```

A label is positioned at the top of the frame and image is attached that will be present for the first four questions. Then as the test taker navigates through the questions the label image will be superseded by another icon that will be used for the upcoming questions. The following code block is placed in the next button's body so as we come cross the fifth question it will modify itself as

```
if(qn==5)
lpic.setIcon(new ImageIcon("datapics/4.JPG"));
```

Codes for next button are similar to those explained in Antonyms and other tests therefore there is no need to recap them. Check them if necessary.

What if the test taker presses the quit button mistakenly? Will he loose all work done so far or what. Of course he won't loose his work. He will get an attention message as:

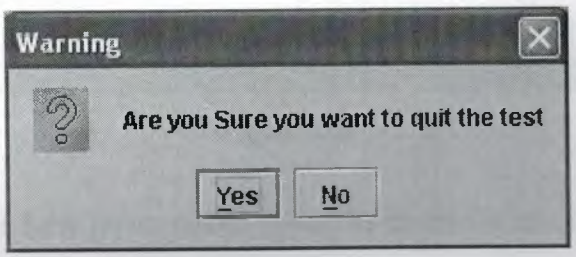

Figure 5.7

```
if(ae.getActionCommand().equals("Quit"))
{int option =JOptionPane.showConfirmDialog(this,"Are you Sure you want
to quit the test","Warning",JOptionPane.ERROR_MESSAGE);
if(option==JOptionPane.YES_OPTION)
this.dispose();
    }
```

This code has been used for all quit buttons and all test frames, hence this very same code can be applied to any quit button.

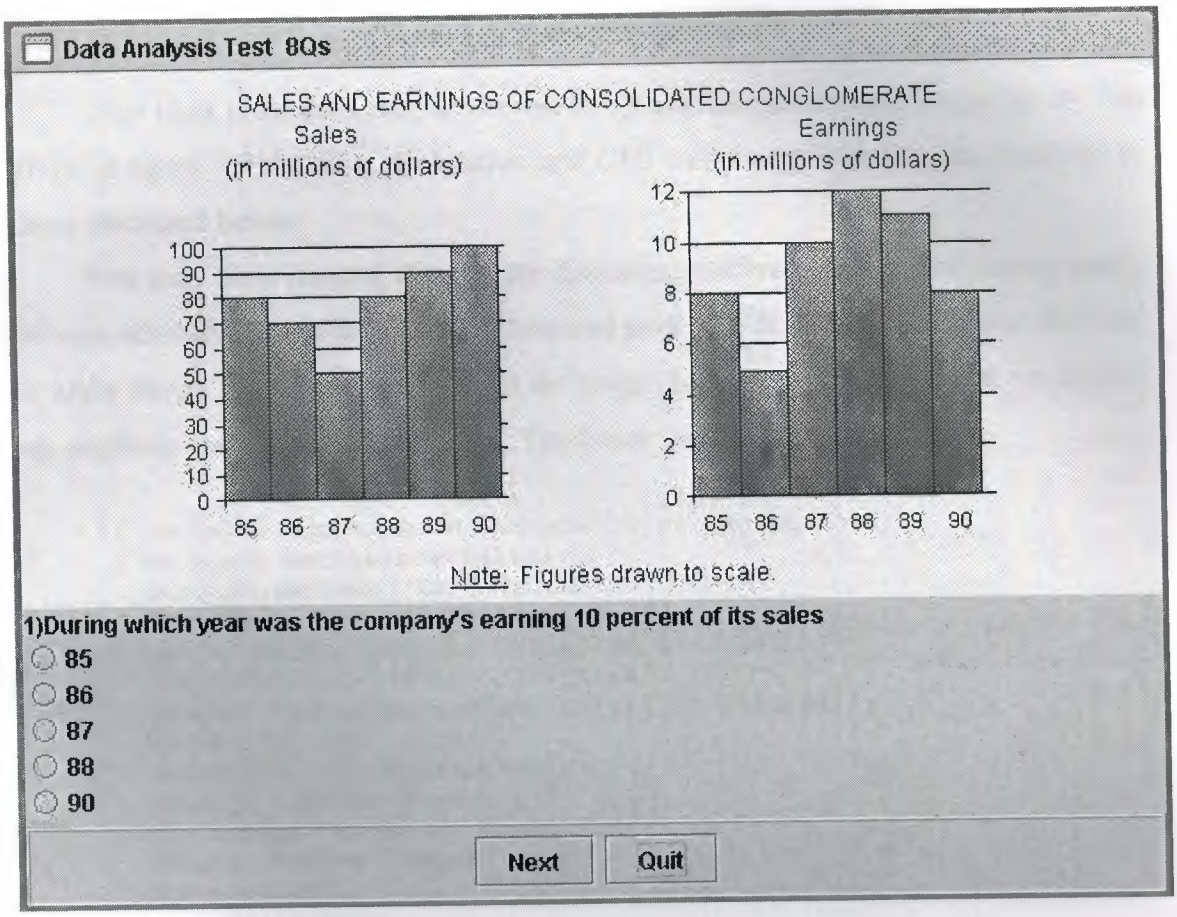

Figure 5.8

So far we have never talked about the codes for retake situation but we did mention that there are two cases. Let's analyze the code for the retake case. How will the program now that the user is taking this test for the second time. If you remember from the main class we said every time user logs in it updates the users table and usersandteststaken table. If test is already taken we get a true values for testtaken variable and then the label for next button becomes "Retake" and here is the code when button with Retake label fires an action:

```
if(ae.getActionCommand().equals("Retake"))
   {retake=true;
    butquit.setText("Quit");
    String deletetest="delete * from
table"+currentuser.getuser()+"datanalysis";
db.connecttodatabase(deletetest);
}
```

We clear all the data in this table with deletetest query and send it to db instance of DB with the method connecttodatabase().

5.2.6 CLASS COMPARISION

This class is for the Comparison test for Quantities Section and the output for this given in figure 5.9 the class declaration and GUI component initialization is similar to those discussed before.

You may have noticed that we are discussing different sections of classes partly because all classes have the same structure and partly we don't want to include all codes to make things seem complex. We put an image that shows the results but we did not say anything about the code view of it. The below code block does this job:

```
if(ae.getActionCommand().equals("View Results"))
 { butstart.setVisible(false);
   butquit.setText("OK");
   JTextArea tqn=new JTextArea(4,10);
   tqn.setBackground(new Color(204,204,204));
   JTextArea tstatus=new JTextArea(4,15);
   tstatus.setBackground(new Color(204,204,204));
   tqn.setEditable(false);
   tstatus.setEditable(false);
   tstatus.append("Status\n");
   tqn.append("Question Number\n");
   JPanel pta=new JPanel();
   pta.add(tqn);
   pta.add(tstatus);
```

When we want to show results on the screen we have make some alterations in the User Interface, such as adding new components and removing unnecessary ones. This job is done in the above code segment.

After making modifications in GUI of the new window we are ready to get the data from the table that retains result of the current user and test. For this we have get connected to the database and get data one by one using the getnext () method of ResultSet interface as explained in Chapter 3. Programming code for this mission is right below:

```
try{
connection=DriverManager.getConnection("jdbc:odbc:test","admin","20010
811");
statement=connection.createStatement();
String getresult="select qn,status from
table"+currentuser.getuser()+"comparision";
resultset=statement.executeQuery(getresult);
while(resultset.next())
{ qnin=resultset.getInt("qn");
```

```
String statusin=resultset.getString("status");
tqn.append(""+qnin+"\n");
tqn.setBorder(BorderFactory.createEtchedBorder(0));
tstatus.append(""+statusin+"\n");
tstatus.setBorder(BorderFactory.createEtchedBorder(0));
     }}
catch(Exception e){e.printStackTrace();}
```

And this last statements are to add altered Swing components to modified positions on the window using getContentPane.add() method.

```
    this.setSize(310,390);
    this.getContentPane().add("North",lquestion);
    this.getContentPane().add("West",pta);
    this.getContentPane().add("South",pbuts);
}
```

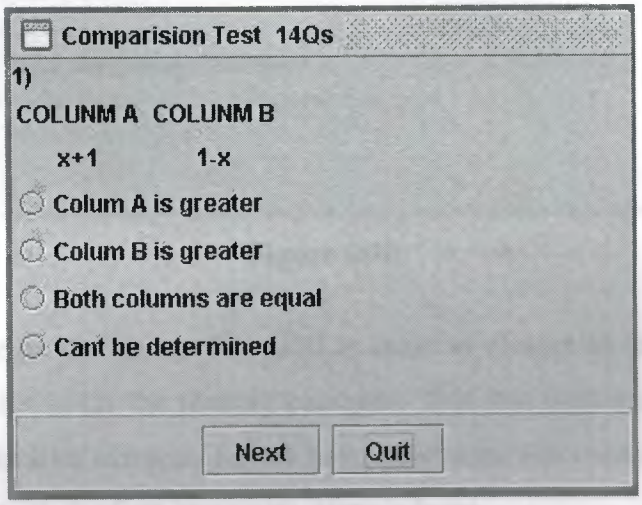

Figure 5.9

5.2.7 CLASS READING

This class is a submenu in verbal section. As its name implies it serves a test of reading comprehension. After designing Class reading we should get something like the screenshot given in figure 5.10.

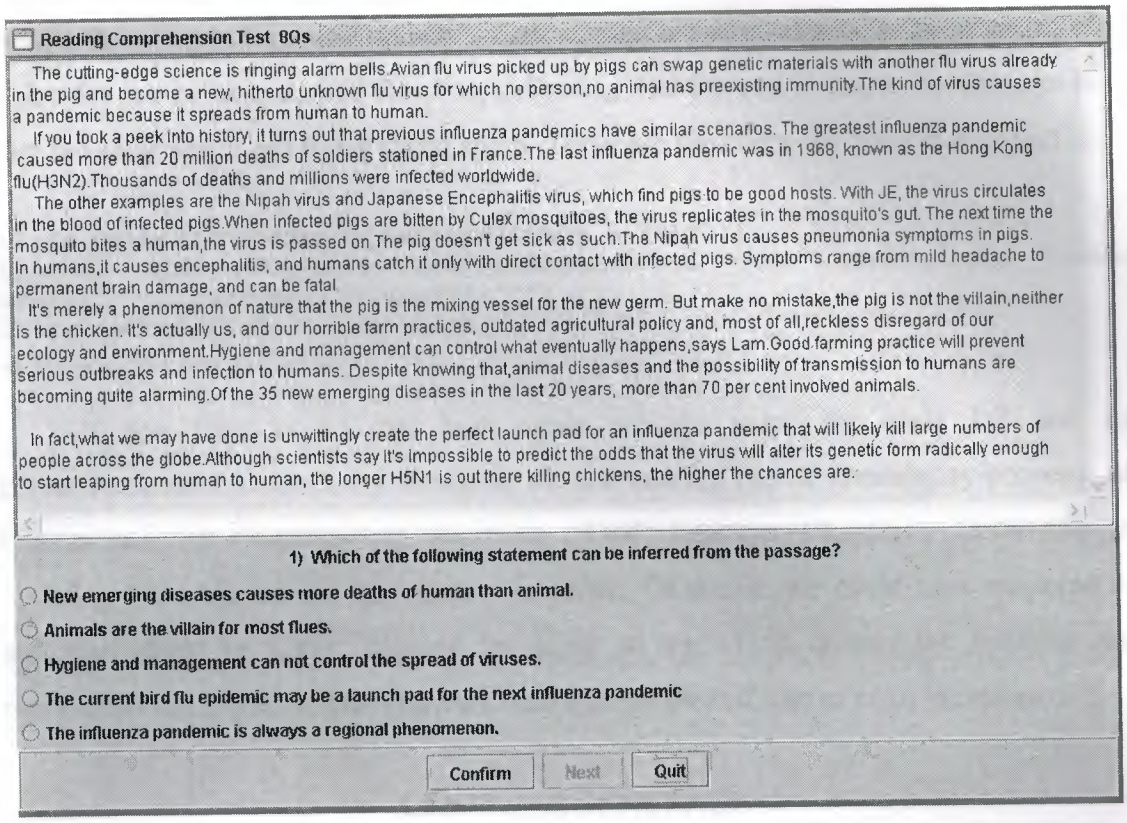

Figure 5.10

Now let's begin designing. The GUI is same as classes already discussed apart from a TextArea that holds the reading passages. This test contains eight questions in total and two distinctive extracts. So we have to change the content of the TextArea when we are done with the first paragraph. Here is how:

```
if(questionnumber>4&&questionnumber<9 )
passage1.setText(reading2);
```

By the way the functions of the buttons given at the bottom of frame are exactly same as we have discussed in prior chapter. So you may revise them if you need so. But here I want to explain the idea of moving ahead in questions in terms of algorithm. The idea here is that, at first the next button will be in disabled position and when the test taker selects an answer and hits confirm button the answer will be registered to the database and next button will change its state to enable and after that the next question will be seen as pressed on the next button. This routine will continue till no question is left to answer. As soon as we get to the end the radios and all buttons will perish only one with "See Results" caption will be showed. Then the codes given in Comparison class are going to be performed. After this a window like in figure 6 will appear. And the window will remain visible until test taker hits Save & Exit button. What does this

button do? This button will update a table named usersandteststaken by inserting true value for the column named Reading indicating that the test is taken. So to make sure that the same test is not being taken again. These are the codes for Save & Exit button:

```
if(ae.getActionCommand().equals("Save & Exit"))
{String queryq="update usersandtakentests set reading=true where
user=+'"+currentuser.getuser()+"'";
db.connecttodatabase(queryq);
this.setVisible(false);}
```

If you have noted that here again we are using the class DB and its connecttodatabse method. By doing so we are making use of Reusability property of Object-Oriented Programming approach. (OOP Features Chapter1). Following to updating the table we set the window invisible. Of course we could have disposed it, and that would be better, because by doing so we would destroy the instance and reallocated the memory. Therefore we may change the last line to `this.dispose()`.

5.2.8 CLASS SENTENCE

The below (figure 5.11) screenshot is for test Sentence Completion that got when we run Class Completion. The GUI and all other facilities for this class are also same as in the other classes therefore you can refer to them for further knowledge. That is why we are not going to rewrite the same codes again. But we will give a bridged code segment from the constructor of Sentence (Constructor Chapter1) to show how a constructor is constructed in terms of codes:

```
        Sentence()
        { super("Sentence Completion Test
15Qs",false,false,false,false);
          Container container=getContentPane();
          butstart=new JButton("Start");
          butstart.addActionListener(this);
          butquit=new JButton("Quit");
          butquit.addActionListener(this);
          butnext=new JButton("Next");
          butnext.addActionListener(this);
```

A constructor gets the name of the class that it is belong to. So do our Sentence constructor. Then we do define it with five parameter in super () method setting the header and window states such as closable, minimize able etc. Then the Content Pane is instantiated to add components later on.

```
this.setSize(554,240);
setVisible(true);
      }
```

At the end of constructor we can define its size and location and should set it visible so when the instance is called to see it, otherwise it will be created but can not be seen.

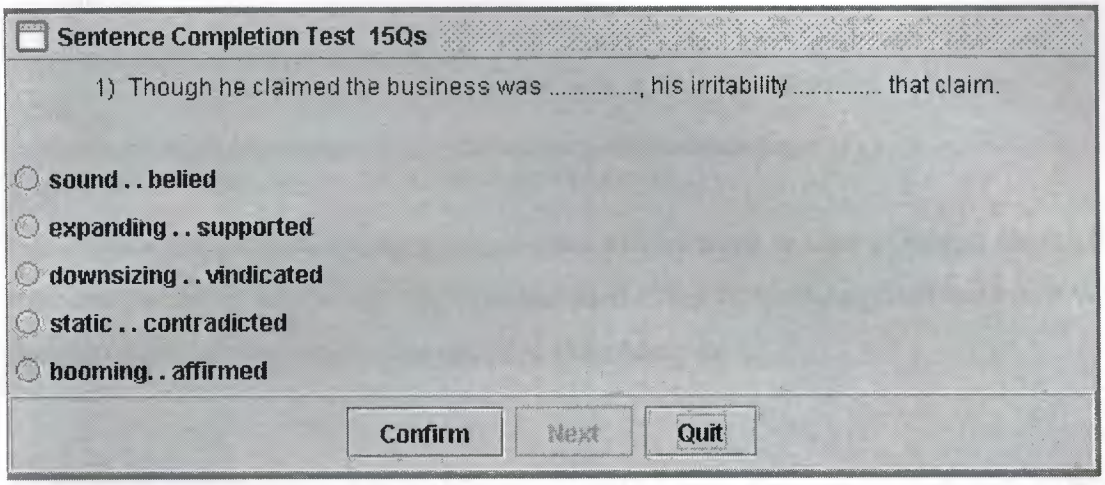

Figure 5.11

5.2.9 CLASS PROBLEM

This class resembles to DataAnalysis because it makes use of icon sometimes in labels. See how an icon can be attached to a label in Class DataAnalysis.

Everything including GUI and the Database connection for this class is no different than the other classes.

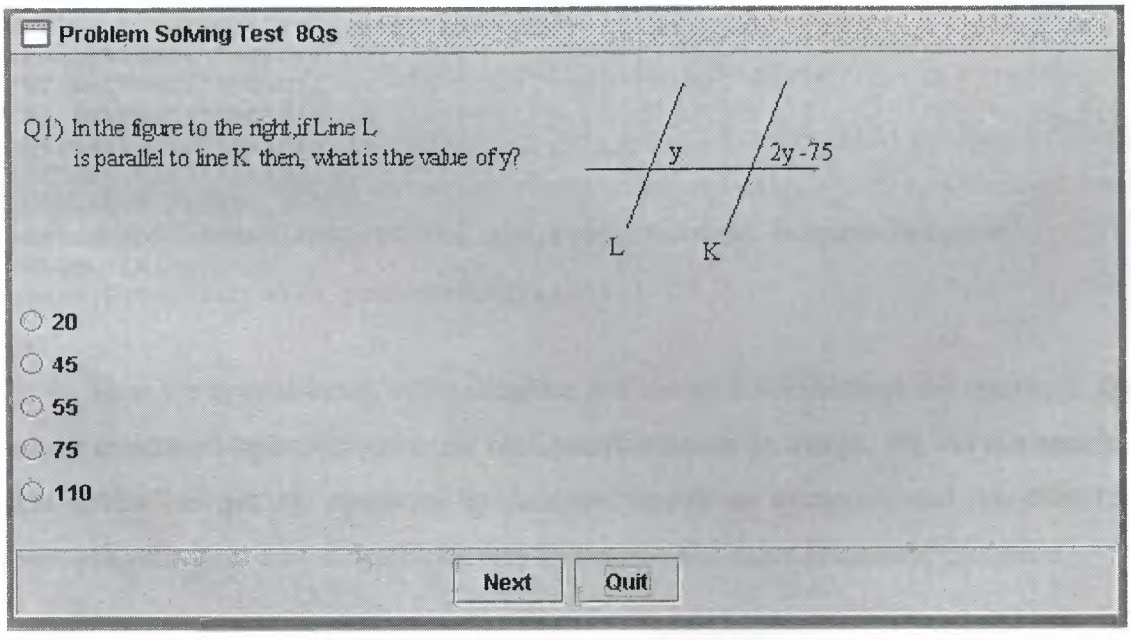

Figure 5.12

Actually when this test is opened we first see a small window telling us to click the start button in order to take the test. This is same for all classes including this one. Now let's give the code block that runs when we hit the start button.

```
if(ae.getActionCommand().equals("Start")||retake)
   {qn=1;
    butstart.setVisible(false);
    lcommand.setVisible(false);
    String query="select qn,answer,a,b,c,d,e from Problem where
qn="+qn;
    try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
    catch(Exception e){e.printStackTrace();}
```

Since the test that we are going to take will be same in case of taking for the first time and for rating that is why we have included either for pressing Start button or when the retake is true. See retake case codes in DataAnalysis.

```
try{
connection=DriverManager.getConnection("jdbc:odbc:test","admin","20010
811");
statement=connection.createStatement();
resultset=statement.executeQuery(query);
while(resultset.next())
{lcommand.setForeground(Color.black);
lquestion.setIcon(new ImageIcon("images/p1.JPG"));
qn=resultset.getInt("qn");
ans=resultset.getInt("answer");
correct[qn]=ans;
String ain=resultset.getString("A");
String bin=resultset.getString("B");
String cin=resultset.getString("C");
String din=resultset.getString("D");
String ein=resultset.getString("E");
rb1.setText(""+ain);
rb2.setText(""+bin);
rb3.setText(""+cin);
rb4.setText(""+din);
rb5.setText(""+ein);}
this.setSize(600,350);
maskunmask.unmask(rb1,rb2,rb3,rb4,rb5,1,butnext,butnext,butnext);
retake=false;}
catch(Exception e){e.printStackTrace();}
  }
```

Then we getconnected to the database and assign RadioButtons the questions and attach an icon to lquestion since our first question needs an image. We did not mention that before but getting connected to database throws an exception and therefore has been placed within a try-catch block (see Exception Handling Statement Chapter 2).

5.2.10 CLASS ABOUT

The about class is to show information about the application and the author of application. See the output screenshot in figure 5.13. The whole code for this class is demonstrated below.

```
public class About extends JPanel
{ public About()
{ this.setLocation(20,20);
setBackground(new Color(153,153,204));
JLabel l=new JLabel("<html><li>  Graduation Project <br><li> Copyright
© 2005 Muslum SENEL <br><li> Version  V1.0"+
"<br><li> All rights reserverd <br> <li> Freeware product"+
"<br><li> <a href mailto:senelmuslum@yahoo.com>"+
"senelmuslum@yahoo.com</a></html>");
this.add("East",l);
}}
```

As has been discussed in JLabel in chapter 2 a HTML code block can be embedded in a label as the caption of JLabel but it won't display the text as its instead will recognize it as HTML and display accordingly.

Figure 5.13

5.2.11 CLASS HELP

To give information about how to use this program we have created a class named Help. The screenshot of this class is in figure 5.14.

The help option are JButtons and been added ActionListeners and the help text is contained in a Text Area. UI creation is done in the same manner as in the previous classes.

Here is a code segment that shows the function of DataAnalysis-labelled button:

```
else if(key.equals("Data Analysis"))
{textarea.setText("");
textarea.append("DATA ANALYSIS TEST\nThis test examines your aptitude
to    figure \n out the data presented in the form of charts\n"+
" or graphs and then analyse that data.\nDiagrams are just for
visualization and \n therefore not to considered as base.");}
```

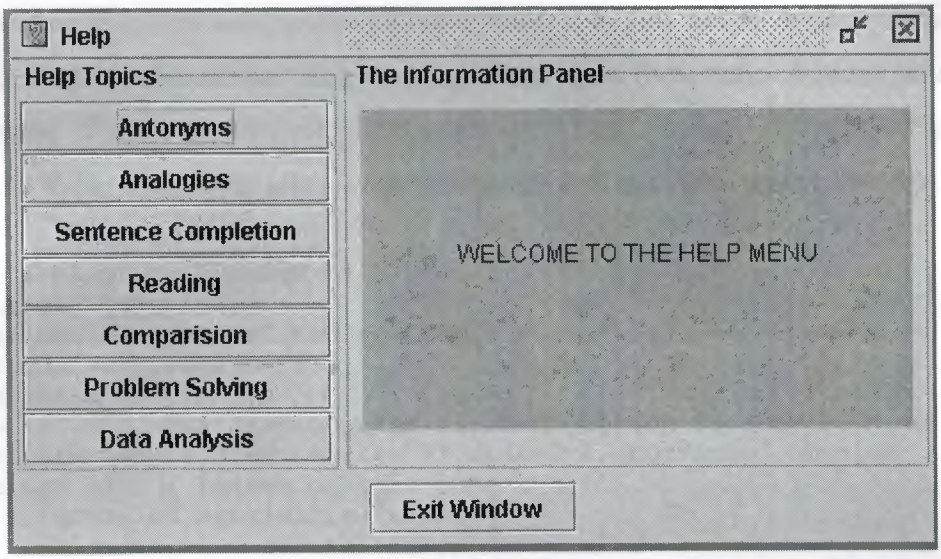

Figure 5.14

5.2.12 CLASS DB

As its name infer this class establishes a database connection. This class doesn't have many lines of code but because we have used in many classes and sometimes twice in a class so wanted to make it a single class. By the way the details of how to create a database connection are in Chapter four. This is a class with single method which creates a connection. So to set up a connection we have to the connecttodatabse() method of DB instance.

```
public class DB
   {Connection connection;
    Statement statement;
    ResultSet resultset;
    public void connecttodatabase(String q)
    {String queryexe=q;
     try{Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
    catch(Exception e){e.printStackTrace();}
try{
connection=DriverManager.getConnection("jdbc:odbc:test","admin","20010
811");
statement=connection.createStatement();
resultset=statement.executeQuery(queryexe);
```

```
connection.close();
statement.close();}
catch(Exception e){e.printStackTrace();}}
}
```

5.2.13 CLASS MASKUNMASK

Sometimes in the tests we may want some components to be hidden for a while and then reappear. For such cases I have created the class MaskUnmask to mask and unmask some components. Those components are five radio buttons and other components. These classes have three methods. Mask, unmask and masks. Since the content of those methods is given as parameter we will not show the whole code.

```
  public class MaskUnmask
  { JLabel lquestion;
   JRadioButton rb1,rb2,rb3,rb4,rb5;
   JButton but1,but2,but3;
   JTextArea t;
   public void mask(JRadioButton r1,JRadioButton r2,JRadioButton
r3,JRadioButton r4,JRadioButton r5,JLabel l,JButton b1,JButton
b2,JButton b3)  { lquestion=l;
        lquestion.setVisible(false);
        rb1=r1;
        rb1.setVisible(false);
        rb2=r2;
        rb2.setVisible(false);
        rb3=r3;
        rb3.setVisible(false);
        rb4=r4;
        rb4.setVisible(false);
        rb5=r5;
        rb5.setVisible(false);
        but1=b1;
        but1.setVisible(false);
        but2=b2;
        but2.setVisible(false);
        but3=b3;
        but3.setVisible(false);
        } }
```

In other methods we will set the above hidden components to visible just by changing the boolean value in the setVisible() method to true.

5.2.14. CLASS CURRENTUSER

This simple class returns the currentuser information every time a test is taken or the program is being logged in. By doing so, we get the chance of finding out whether the test has been taken by that user or not and also does that user exist or not. Code block of this class is as follow:

```
class Currentuser{
FileReader filereader;
BufferedReader br;
String usernormal;
```

Since we are retaining user info in file temporarily before adding it to the database we should get these data from the file named curuser.class. To do so we need to have a FileReader and BufferedReader instances.

```
public String getuser()
{try{ filereader=new FileReader("curuser.class");
br=new BufferedReader(filereader);
usernormal=br.readLine();
}
catch(Exception e){}
return usernormal; // returns a string type variable when
called.
}}
```

5.3 GETTING IT ALL TOGETHER

By this point we have finished designing our computer-based examination. We gave all the classes and components separately, but if we compile those classes in one directory and run the Users class we will get the screenshots discussed above. One thing that I should mention again is that for all tests and results we use Database via JDBC: ODBC Bridge.

CONCLUSION

This project was to develop a java application namely a Computer-Based Examination (CBE) package. Since Java is an Object-Oriented programming language an overview of OOP was served in the first chapter as a reference for succeeding sections.

To bring this application about an interactive Graphical user interface and database connectivity was needed, for that reason Java Swing components and Java Database connectivity with simple SQL statements were explored in second, third and fourth chapters.

Before getting to last chapter everything was ready and digested to build a CBE. As we started developing this application we realized that the Object-Oriented approach and GUI features that Java offers were quite sufficient and suitable to reach our aim. Hence one can easily put forward that Java is a powerful and an easy-to-use programming language and environment.

The CBE developed here was intended for Standalone and offline purposes, however if further work can be employed this application can easily be improved. Java is flexibility and therefore if this application can be applied to Java Applets then a Web-based online CBE could be possible, or by using the network capabilities of java a concurrent multi-user CBE framework could be accomplished in which users may compete with each other against time.

To conclude; Java is a high level programming language offering a rich number of facilities with which various applications can be constructed without difficulty and if needed might be adapted and transformed to new environments.

REFERENCES

[1] Herbert Schildt, The Complete Reference: Java 2, McGraw Hill Companies, Osborne, 2003.

[2] David J. Gallardo, Java Oracle Database Development, Prentice Hall PTR, New Jersey 2002.

[3] Mark Wutka, Java Expert Solutions, Que Corporation, IN 1997.

[4] Mike Cohn, Bryan Morgan , Michael Morrison, Michael T. Nygard , Dan Joshi ,Tom Trinko, java Developer's Reference, Sams.net Publishing Indianapolis, IN.

[5] The Java Tutorial, A practical guide for programmers, Sun Microsystems, Santa Clara, CA,2004.

[6] How do I learn JDBC? http://www.javaskyline.com/learnjdbc.html