

T.E.I. PIRAEUS

Faculty of Engineering

Departments of Electronics and Automation

MSc in DATA COMMUNICATIONS

COURSEWORK

MODULE:

Developing Object Oriented Solutions

CSESM00: CIM234

Module Coordinator:

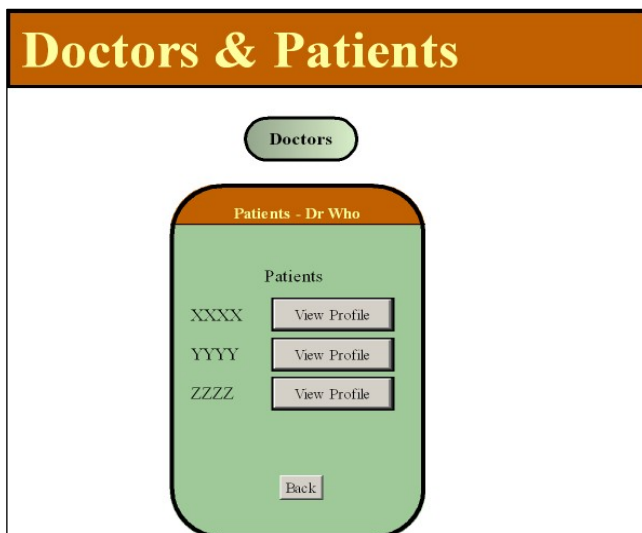
Dr Adamopoulos Dionisios

Date of Module:

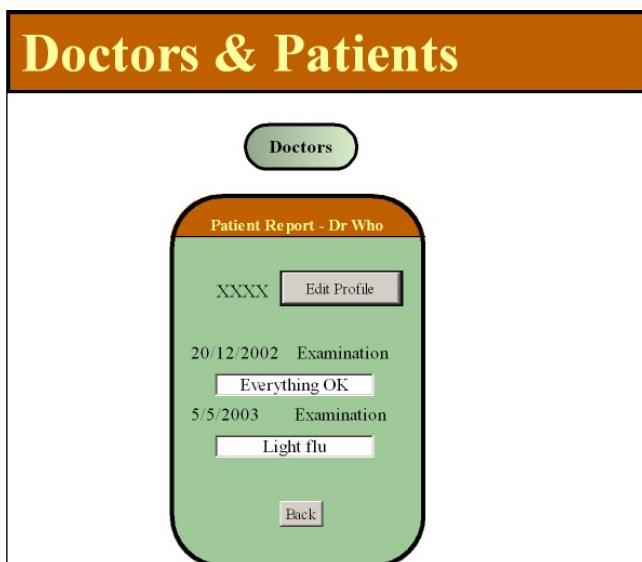
3rd Term (Fall 2005-2006)

Name of Student:

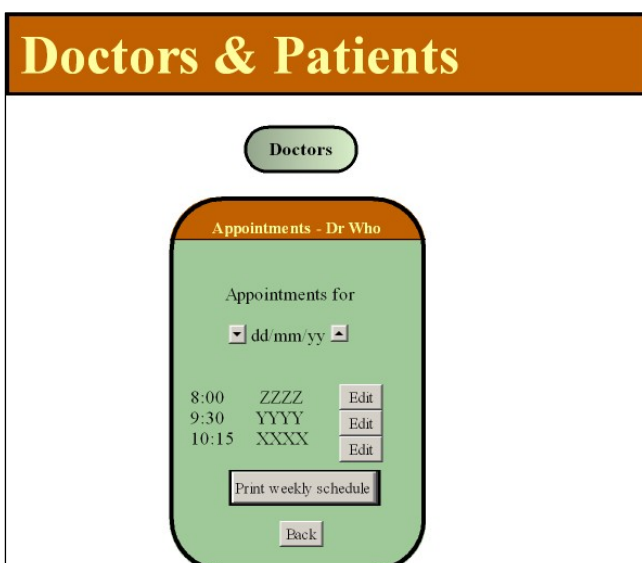
Vasileios Balafas



Doctor views his assigned patients and he can have a report of their detailed profile.

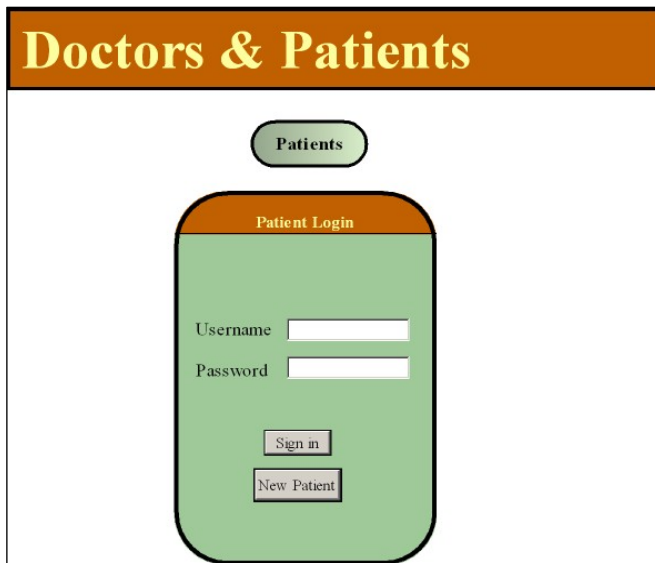


The detailed report of a patient gives information about past examinations and conditions of the patient. Doctor can edit the profile.

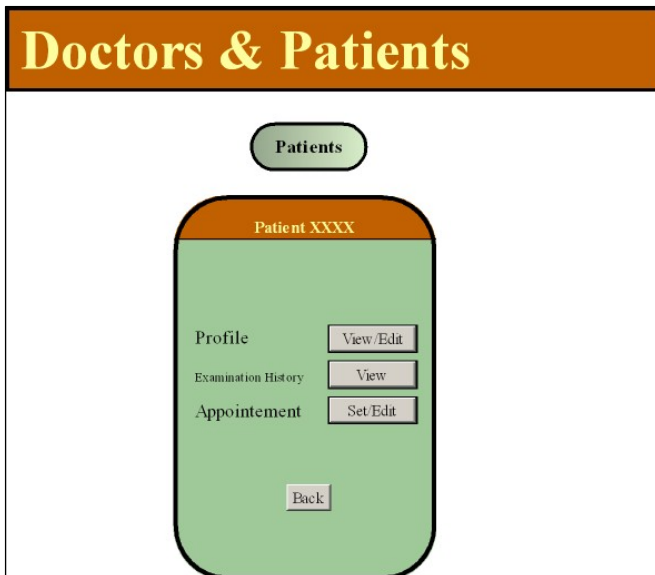


In the initial screenshot if doctor selects appointments, he can see his schedule and edit details for each appointment or cancel one. Doctor can also print a weekly appointment report - schedule.

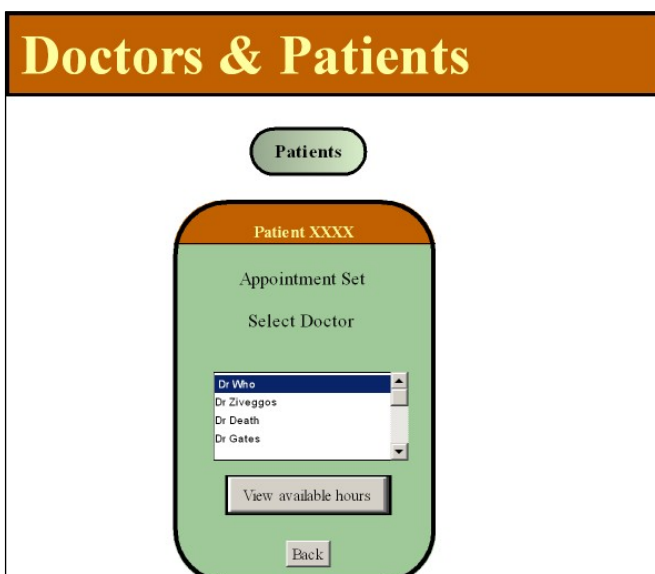
b) For Patient Login



Patient logins in the system after selecting Patients in the main screen. Here a new patient can be registered in the system.



After login, patient can see and edit his personal profile (he can edit specified fields and not all) according to his rights. He can view a report about his examination history and arrange or edit an appointment.



By selecting appointment at the previous screen, patient can access a list of doctors, select the preferable one and view the available hours for meeting him.

The screenshot shows a web application interface for 'Doctors & Patients'. The main content area is titled 'Patients' and contains a form for 'Patient XXXX'. The form has a section for 'Appointment Set' with a 'Doctor Who' dropdown menu and a date selector 'dd/mm/yy'. Below this is a list of times from 8:00 to 12:00, each with a 'Set' button. The 9:00 and 11:00 slots are marked 'N/A'. At the bottom of the form are two buttons: 'View your appointment list' and 'Back'.

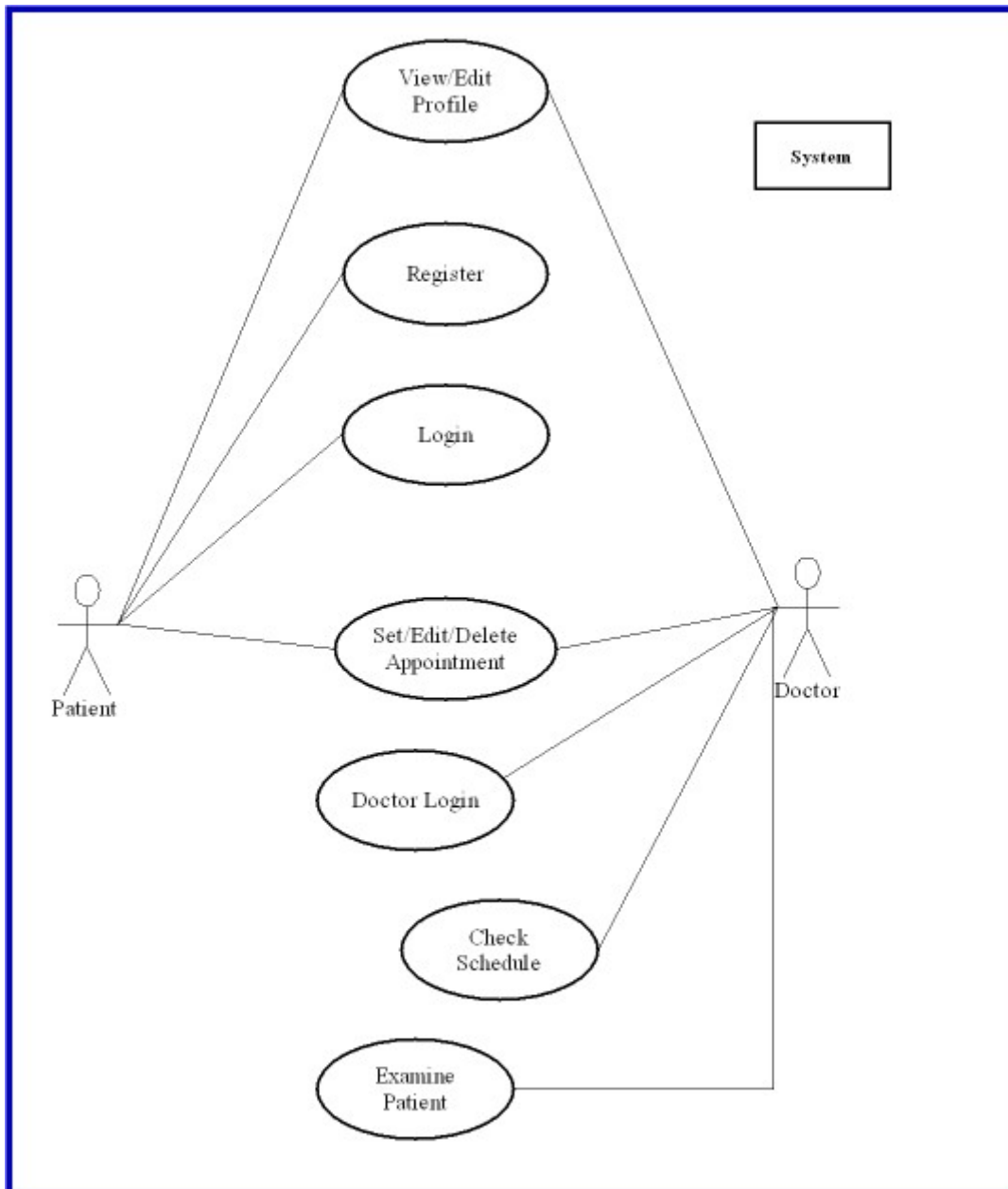
After choosing the doctor, patient can see the available hours for an appointment with the specific doctor and he can SET an appointment by pressing the relative button. Patient can also see a list of appointments where he can change or cancel appointments by selecting the “View your appointment list button”.

The above screenshots designed with SmartDraw Suite Edition v.7, depict a prototype of the system's user interface. It can be enriched but it shows satisfactorily the sequence of the screens that a user may see in a LAN or in an internet environment. Those screens are depicting a user friendly interface and are fully compatible with Web browsing. A database at the background maintains and stores information that is useful for the user. The administrator controls and shares the appropriate rights to users in a way that users cannot alter information that they should not.

The proposed application has advanced capabilities and can inform patients if a doctor cancels an appointment, or reminding them their scheduled appointment via e-mail.

2) Use Case Diagram

At the following design we can see the Use Case Diagram of the system.



The use case diagram is the starting point when designing a new system using UML. It is used to enumerate the requirements of a system in a way that everybody can understand. The above use case diagram depicts the requirements of the system according to the actors who are the doctors and the patients. It shows exactly what the system should be able to perform. Patients can log in the system, view and edit their personal profile, manage the appointments. New Patients should be able to be registered to the system.

On the other hand, Doctors login with their already given username and password, manage their appointments, view and print their schedule and write reports about the examinations.

3) *Detailed Use Cases*

a) Appointment Arrangement

For the appointment of the arrangements, we can have two categories; those arranged by patients and those arranged by doctors after a patient's phone call or after a visit at the doctor.

For the first category we have the following steps for a successful scenario:

- The patient logs in the system, or if he is a new one, he can register
- He selects the doctor with whom he wants to have an appointment for an examination.
- He searches for a time that doctor will be available.
- He selects the preferable available day and time and finally he creates the appointment.

In an unsuccessful scenario, the patient could forget his username or password and then the system could send him this information via email using some predefined questions in order to identify him uniquely.

For the second category, appointments arranged by doctors, we can conclude the following steps:

- The patient calls the doctor to arrange a visit.
- Doctor logs in the system.
- Doctor checks his schedule and if he is available at the requested time.
- Doctors creates the appointment

OR

- Doctor proposes a different time or date
- Doctor creates the appointment at the new specified date.

The unsuccessful case would be the unavailability of the chosen doctor at all requested dates and times. The system could provide an operation by witch the patient could be proposed another doctor available.

b) Patient Examination

Patient examination is a very important operation of the application because the doctor can see the special needs of each patient, view a history report about past examination and note conclusions and treatments.

Studying the requirements we can conclude the following detailed steps of a successful case for this process.

- Doctor logs in the system.
- Doctor checks patient's existing profile and reviews the medical record.
- Doctor actually examines the patient physically.
- After examination, Doctor updates patient's profile and medical record.
- If necessary, Doctor arranges a new appointment with the patient and confirms his schedule.

An unsuccessful scenario could be the inexistence of the patient in the system. Doctor could register on-line the new patient, input his profile and provide the patient with his brand new username and password.

4) *Requirements and main concepts*

Reading the application's requirements, the following table can be derived which includes the candidate classes.

Patient	The first actor of the application. A specific user with characteristics that help the system to identify him and to offer him services.
Patient Data	The requirements define the ability to add new fields to a patient that are not predefined. A separate class is needed, associated to the patient, in order to store those new fields.
Doctor	The second actor. A specific entity with unique attributes. Each doctor has his identification and has to have access to his personal data.
Appointment	A basic entity that will bridge doctors and patients. Examination information will be stored in this class and later will be used to generate the medical record of a patient.
System	This will be the central class of the application. The main class. It will provide the interface with the actors and will have advanced capabilities such as the sending of emails.

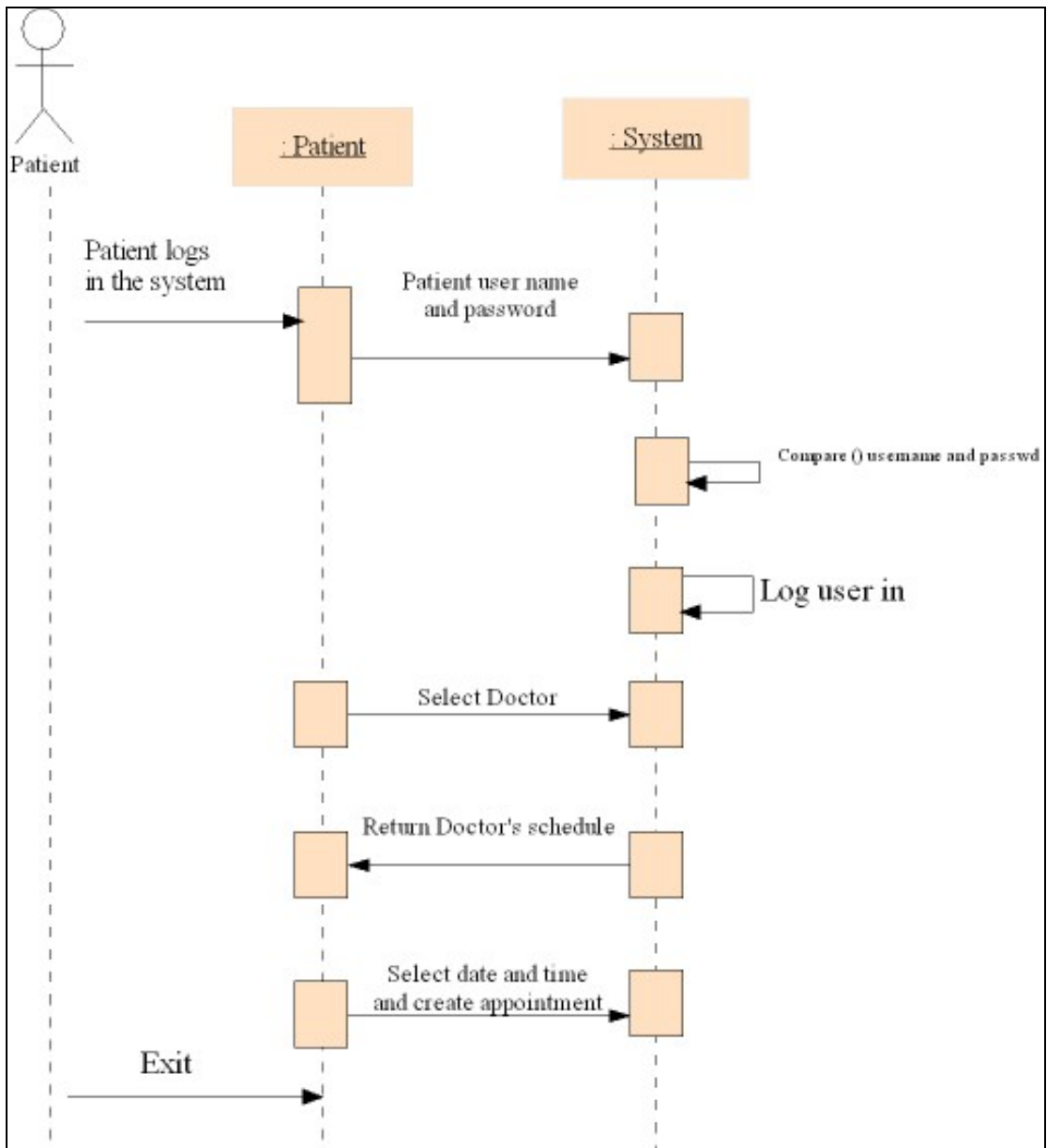
The candidate classes represent the main logical elements of the application. Doctor, Patient and Appointment classes are referred to the basic entities of the application. Patient Data is an extra assistant class that will store new not predefined fields. System will be the core of the application, the "engine starter" that can perform external operations.

5) *Sequence Diagram for "Appointment Arrangement"*

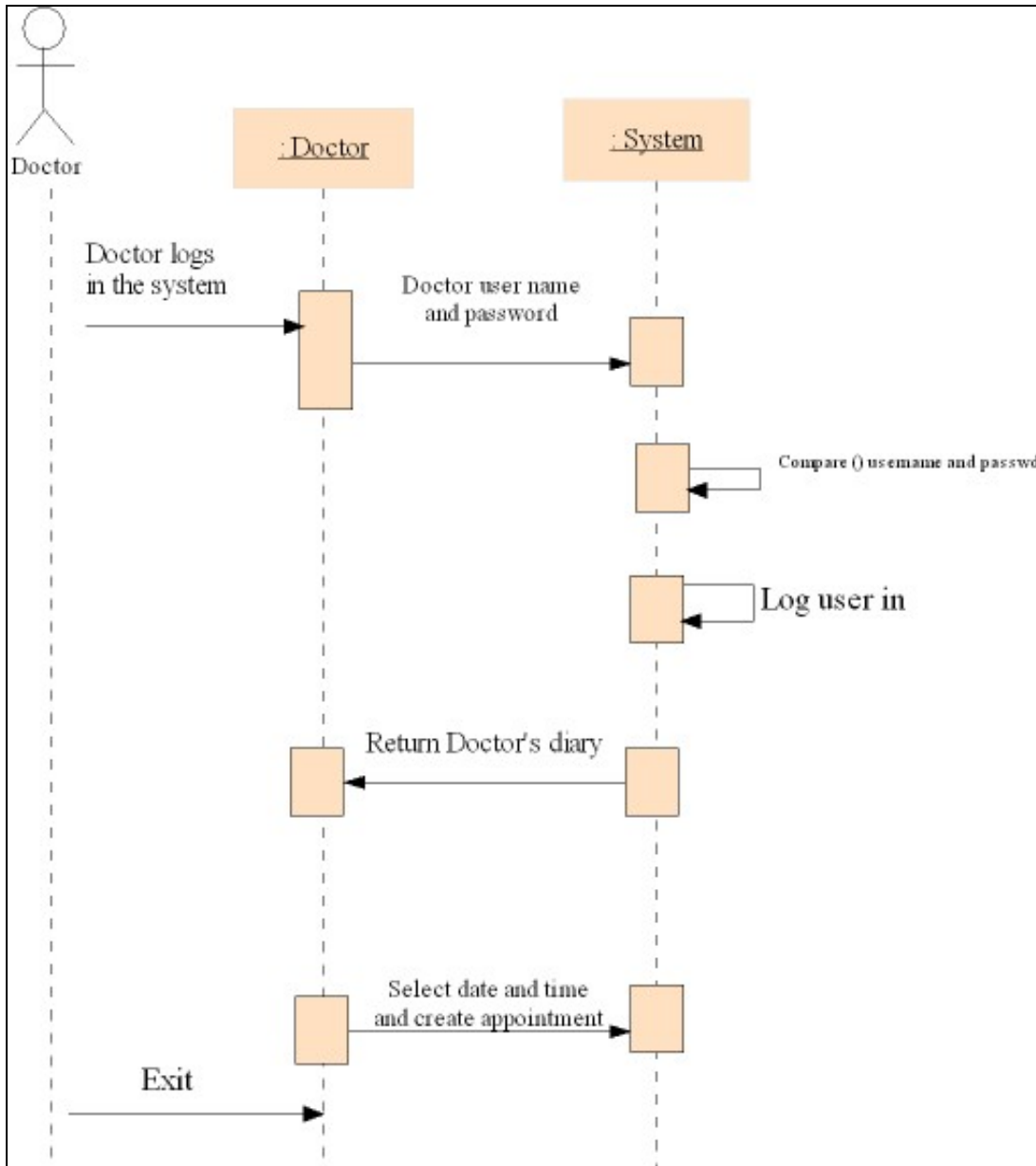
Sequence Diagrams are used to show interaction between actors and objects and other objects. Messages sent from actor to object, object to object and object to actor are depicted to show the flow in a system. Sequence Diagrams are very helpful for the understanding of use cases and the way by which those cases are going to be fulfilled.

The following figures show Sequence Diagrams relative to the "Appointment Arrangement" use case. According to the analysis of Question 3 there are 2 categories for the arrangement. The ones arranged by patients and those arranged by patients.

Appointments arranged by patients:



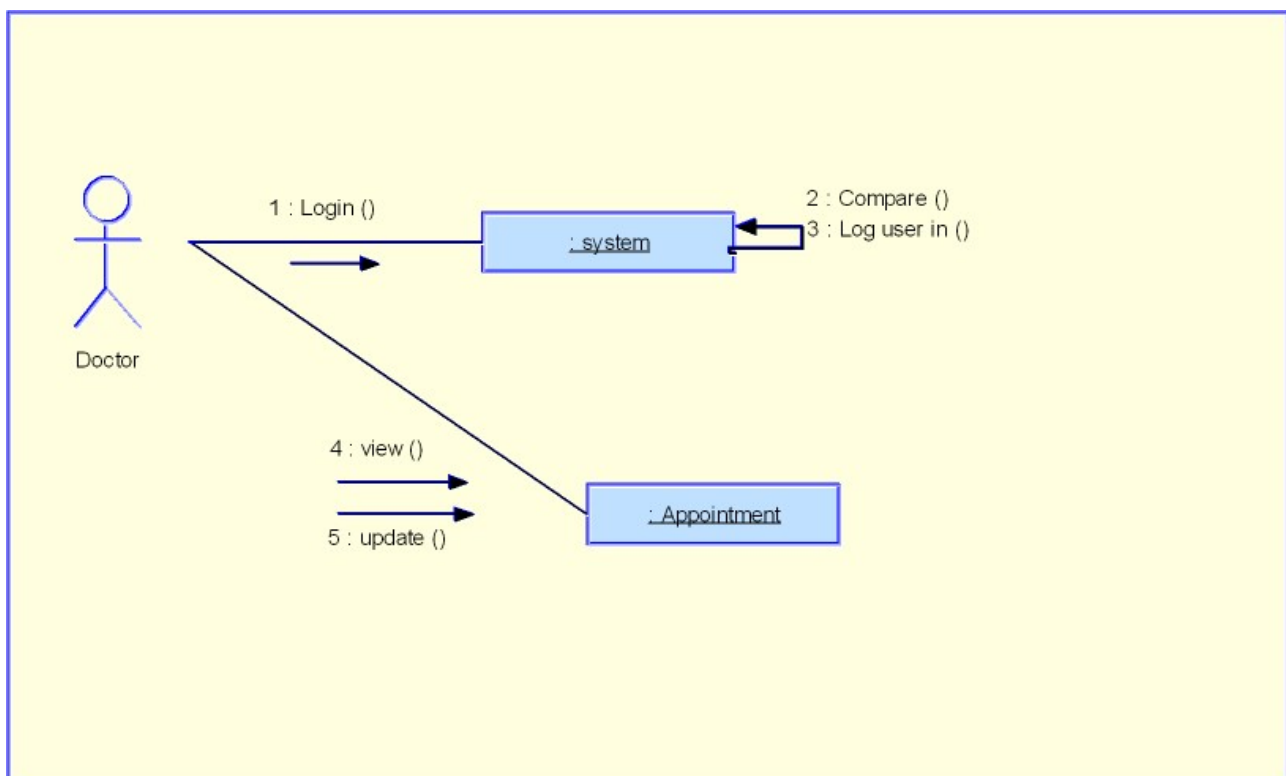
Appointments arranged by doctors:



6) Collaboration Diagram for “Patient Examination”

Collaboration Diagrams are used to bring class diagrams to the next step. Representing the interaction and the relationships between the created objects, these diagrams can be used to model messages exchanged between objects.

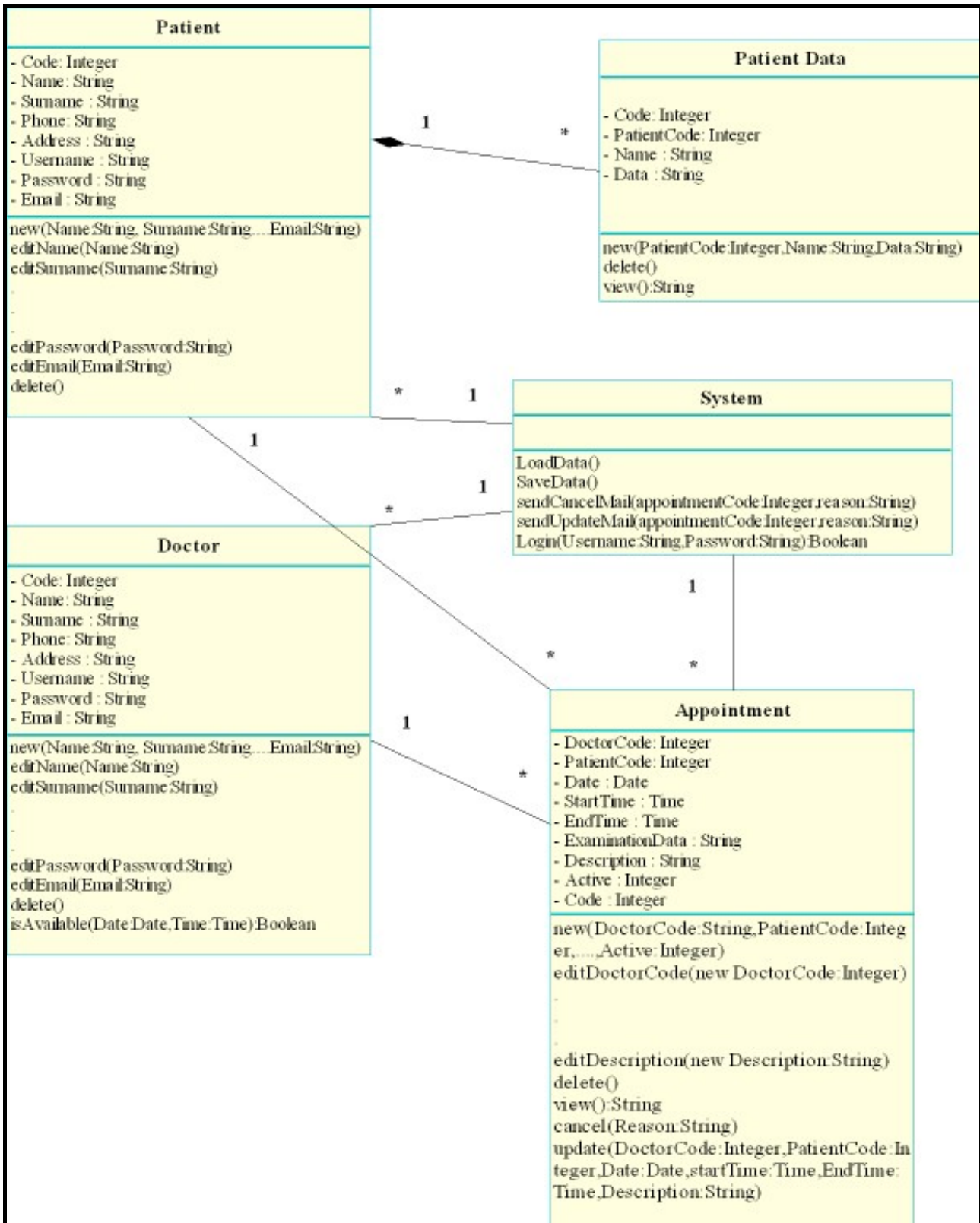
Patient Examination Collaboration Diagram in the following picture depicts the classes involved and the actions made for each step.



Doctor logs in the system, selects the patient, gets his medical report and after examination he can update the patient’s medical status.

7) Detailed Class Diagram

Class Diagrams represent the different underlying pieces and their relationships. They include attributes, operations (methods), roles and associations. The detailed Class Diagram of the proposed application follows depicting the object-oriented solution that is designed for Component 1 of the coursework.



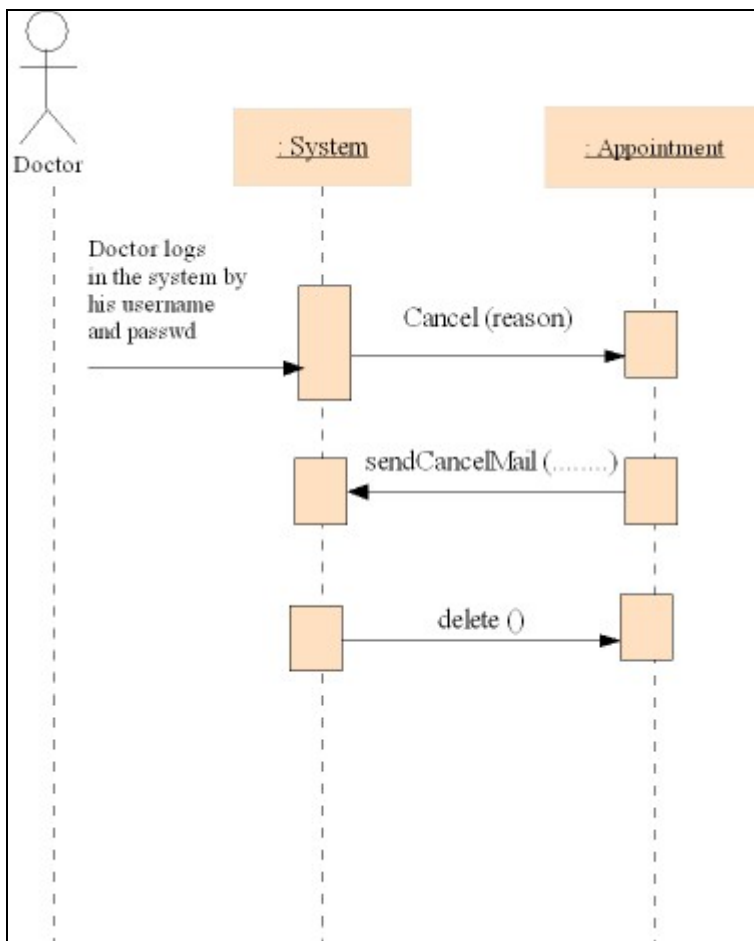
The general role of each class presented in Question 4. The rationale is the creation of sets of objects that can cooperate in order to offer the required services from the system. Classes communicate each other by their methods and important are the LoadData and SaveData methods of class System that can offer interoperability with a database.

Referring to the two requested use cases, the above Class Diagram is designed having the following concept. Each doctor's schedule can be formed from all the Appointment objects that correspond to his unique code. By adopting the same logic each patient's medical record can be derived from the corresponding Appointment objects to his code containing the attribute ExaminationData which offers information about past examination results.

8) *Class Diagram – Requirements*

The proposed Class Diagram fulfills the requirements given by implementing all needed characteristics for each application entity in form of attributes. The depicted methods offer the requested interaction between those entities and the characteristic example is the class Patient Data which is designed to support the need for undefined fields that may contain useful information about the patient. The use of attribute PatientCode helps the accumulation of all Patient Data objects concerning a unique patient.

9) *Sequence Diagram for A10*



The above Sequence Diagram presents the case that a doctor cancels an appointment due to an emergency situation. The system is expected to send an email to the patient and inform him about the cancellation.

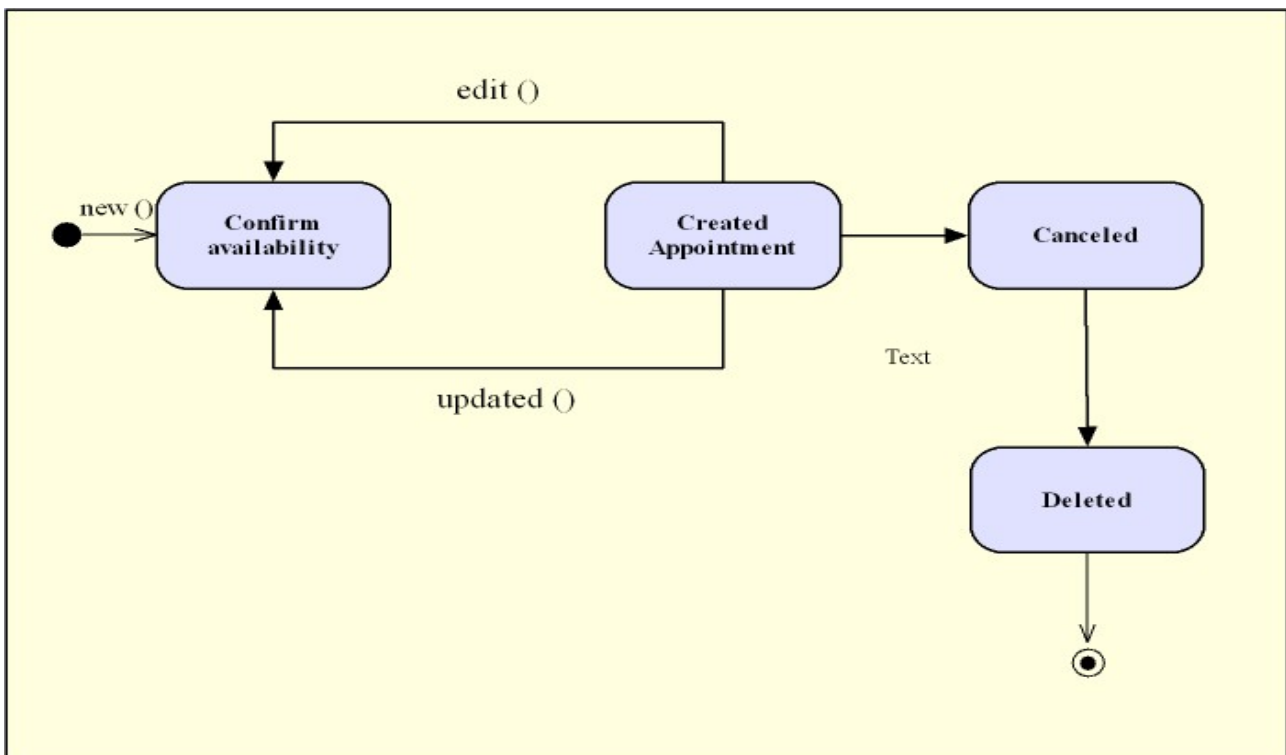
The email is formed by the reason that the doctor declares and is stored by the cancel method of the class appointment. sendCancelMail method identifies the canceled appointment by its unique code and has the string reason as input in order to create the email to the patient.

Finally the appointment is deleted in order to keep the doctor's appointment list updated.

10) *State Diagram for the class of appointment*

State Diagrams are used to model the interactions with classes and the system interface, and to realize use cases. They are used between the analysis and design phases and visualize the flow of an application. The following picture tries to implement this concept regarding the class that models an appointment.

The operations depicted are the confirmation of the availability of a doctor and the manipulation of the lifecycle of an appointment (Created – Canceled – Deleted).



Note for Component 1

The designs contained in Component 1 were made with SmartDraw Suite Edition v.7. They can be also found in the submitted CD of the Coursework in the folder Designs in two types (sdr and jpeg) if further and more review is needed.

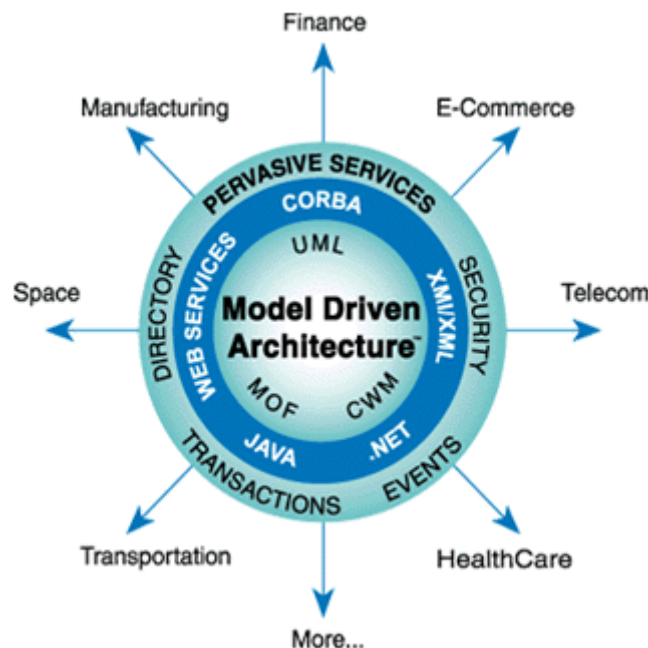
Component 2

Model Driven Architecture by Object Management Group

When looking through the internet and the bibliography for information about the MDA (Model Driven Architecture), it is easy to understand that it is a “hot” topic in the programming community. Several analysts and programming engineers present their opinion and a lot of work has been made about this issue. The majority agrees that the technologies have grown more complex over the last years and that there are several tools for programming. Those tools have become independent from the logical essence of modeling, which has also become a significant and important – most believe vital – part of the programming proceedings.

Defining modeling in a simple way, we can say that modeling is a way to think issues through before coding at a higher abstraction level without needing to solve a problem or designing a solution during the production of code. Nowadays, this is a common consensus and it is considered as the best practice among software developers.

MDA is proposed by the OMG [1], founded in April 1989 by eleven companies as a non-profit corporation. OMG’s purpose is to provide specifications and standards to the software industry. In the fullness of time, OMG evolved as the leader in providing vendor and language independent interoperability standards to the software enterprise and includes more than 800 members. The main target is to offer the guides for heterogeneous computing environments for software development, independent from hardware platforms or operating systems.



The above picture [2] is the classic mark of MDA and shows that MDA adopts and includes well adopted and mature techniques such as UML (Unified Modeling Language), MOF (Meta-Object Facility) and CWM (Common Warehouse Metamodel) for the specification of models and metamodels. The MDA initiative considers that those elements are the primary artifacts of software development, and not the programs [3]. Those exact components constitute the core of the MDA initiative.

UML is a semantic language that can be applied to any software development process. Several types of UML diagrams can be used in order to model object-oriented software systems.

Currently UML has reached version 2.0, while version 1.4.2 has become an official ISO specification [4].

MOF is a set of standard interfaces that can be used to define and manipulate a set of interoperable meta-models and their corresponding models [5].

CWM specifies interfaces that can be used to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments [6].

It is not purposeful to get into details about the components of MDA and examine them intensively. The key idea is that MDA offers the way to create diachronic models about an application once, and those models are offering interoperability, portability and independence from platforms in order to get over middleware problems. Many tools are presented in order to make the transition between MDA and coding more automated and efficient. Besides, MDA is the effort to integrate several specifications that are very helpful but they cannot find a common place in order to be interoperable.

An important technical detail is that MDA adds great value to the environment of differing system architectures with the PIM (Platform Independent Model) that maps PSM (Platform Specific Model) to application interfaces, code, GUI descriptors and other vital parts of an integrated application. Current research is focusing in how PIM and PSM will be transformed into efficient source code automatically [7]. This code must be highly understandable in order to be debugged, maintained and extended.

Making here a little parenthesis, it is obvious that MDA is a higher level of software development consciousness. Traditional engineers claim that models aren't the product but an abstraction of it but in the same time, they recognize that models are very significant for their work. It is also obvious that MDA is not just about a modeling tool, but a general and complete consideration or proposition for building programs.

Returning to the main subject of the coursework, we could separate the process of implementing MDA into three main steps - views [8]. The Operational View, the Systems' View and the Technical View. At the first view we meet the PIM where we need to examine what is needed to be built and what functions should be offered. At the next level there is the PSM and the need to define how all this can be implemented. Finally, at the Technical View, we must define the standards for the implementation. All the above views-steps are very well supported by the UML. Besides, UML holds the leading role in MDA, being the main tool for the facture of the MDA initiative.

MDA's vision is that with its core elements applications could easily cooperate with other applications and future needs could be fulfilled without needing to implement structural changes or redesign the application. The optimistic intention is to contribute decisively and to be the pioneer in distributed ubiquitous and pervasive computing. Integration of different technologies and cooperation of different devices and applications in heterogeneous environment is the total vision of the computing community. All branches of the Informatics Science are migrating and researching this purpose that will bring a new revolution in Information Technology. MDA seems to be the contribution of OMG for this project and programming community reckons that MDA may become a widely acceptable aspect and not a utopia.

OMG presents at [9] lots of successful implementations of the MDA initiative and the most famous is the case of the Deutsche Bank Bauspar AG witch is referred in many scientific and research related works. Those implementations show the potency of MDA and many software architects agree that MDA came to stay and compare its future success to the success of the 3GL Programming Languages that were presented some decades ago and still remain at the center of the programming process.

On the other hand, there are some researchers who believe that are not very excited with MDA. From the above short analysis, in this topic of the coursework, has been made clear that MDA is very sophisticated, complicated and some times can become inefficacious in terms of time and effort. Furthermore, from my short experience, it is very difficult to teach someone modeling

and how to create sequence diagrams for example. It also difficult to separate the designing deliberation from the expected source code and I dare to say that maybe this is an education matter. Most people care about the “product” and how soon will be ready and not about the way, the philosophy of production and the future needs that will arise. That’s why several commercial applications broke down in the near past. Designers and programmers did not think about future needs and maintenance.

Some experienced analysts claim that similar visions existed in the past too, such as I-Case, Application Development Cycle and even CORBA and failed to fulfill their potential. They also believe that the tools offered at the moment are not enough or efficient. But, the most powerful argument is that every time standards tried to be set, vendors finally implemented their own version for competitive reasons.

The future only will show if they are right.

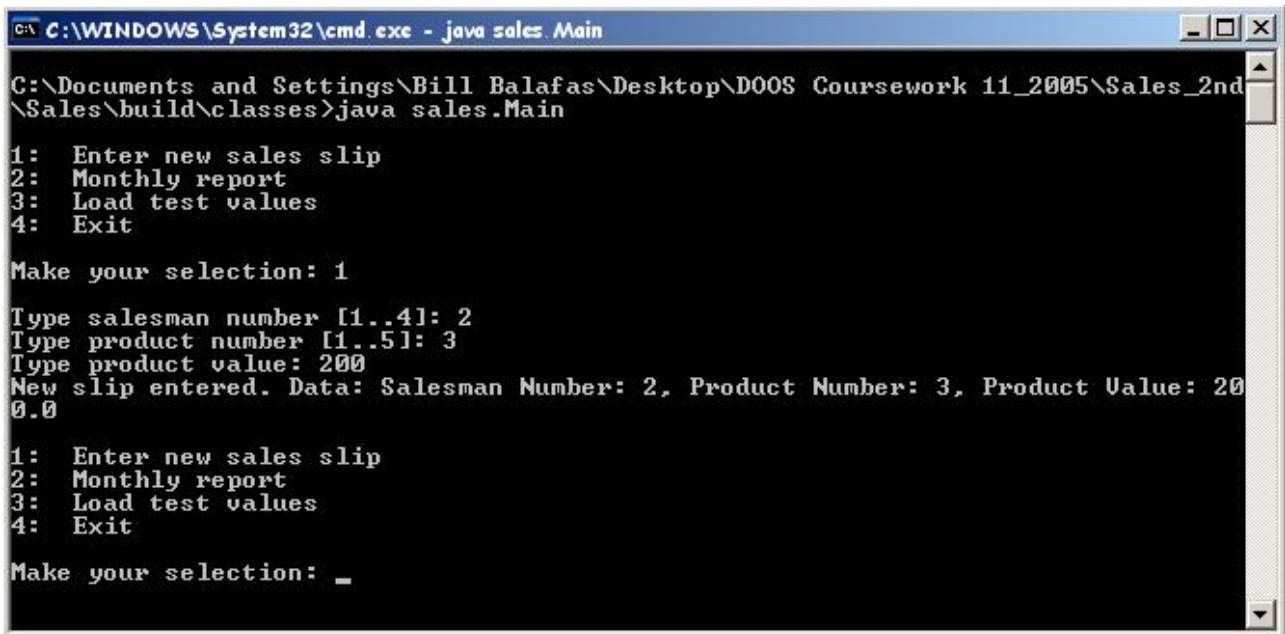
The fact is that Model Driven Architecture is an “umbrella” of standards for platform independent development of applications and software solutions. It supports and promotes a new direction for system development that will offer integration, interoperability and portability. Finding fitting at a higher lever of abstraction, it demands very good knowledge of its specifications and minds capable to comprehend and conceive modeling in an expansive way. MDA seems to be a total solution for developers and designers but has to be less complicated and offer efficacious tools that will increase productivity.

References

- [1] <http://www.omg.org/gettingstarted/overview.htm>, URL
- [2] <http://www.omg.org/mda/>, URL
- [3] K. Thramboulidis, “Using UML in Control and Automation: A Model Driven Approach”, 2nd IEEE International Conference on Industrial Informatics, Germany, Berlin 2004.
- [4] J. Roff, “UML, A Beginner’s Guide”, Mc Graw Hill, 2003, pp. 2-12.
- [5] http://www.service-architecture.com/web-services/articles/meta-object_facility_mof.html, UML
- [6] http://www.service-architecture.com/web-services/articles/common_warehouse_meta-model_cwm.html, UML
- [7] MDA drafting team, “A technical Perspective”, Draft, February 2001
- [8] A. Tolk, “Avoiding another Green Elephant – A Proposal for the Next Generation HLA based on the Model Driven Architecture”, Simulation Interoperability Workshop Fall 2002, Orlando, Florida, September 2002.
- [9] http://www.omg.org/mda/products_success.htm, UML

Component 3

In this part of the coursework the subject is to create an application for the maintenance and the presentation of the monthly progress of sales of some products. Salesmen have to insert their salesman number, the number of the product and the value of the amount sold. The initial screen is a simple menu as it shown at the following screenshot.



```
C:\WINDOWS\System32\cmd.exe - java sales.Main
C:\Documents and Settings\Bill Balafas\Desktop\DOOS Coursework 11_2005\Sales_2nd
\Sales\build\classes>java sales.Main
1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit
Make your selection: 1
Type salesman number [1..4]: 2
Type product number [1..5]: 3
Type product value: 200
New slip entered. Data: Salesman Number: 2, Product Number: 3, Product Value: 200.0
1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit
Make your selection: 2
```

The main menu provides 4 options. Option 1 begins the process of entering data, the option 2 shows the monthly report which can be taken at any time in order to provide daily information, option 3 is a test environment providing a tool to show the capabilities of the application. Finally, option 4 ends the program.

Selecting option 1 the application waits to accept values and information from the salesman, in other words waits for the sales slip. As we can see the program ensures that the right information will be given by reminding to the user the range of the values that he has to insert. If the information given is acceptable, a data message appears which informs the user that his slip is entered successfully and summarizes his input.


```

C:\WINDOWS\System32\cmd.exe - java sales.Main
C:\Documents and Settings\Bill Balafas\Desktop\DOOS Coursework 11_2005\Sales_2nd
\Sales\build\classes>java sales.Main
1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit
Make your selection: 1
Type salesman number [1..4]: 1
Type product number [1..5]: 3
Type product value: 50
New slip entered. Data: Salesman Number: 1, Product Number: 3, Product Value: 50
.0
1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit
Make your selection: 1
Type salesman number [1..4]: 1
Type product number [1..5]: 3
Type product value: 50
New slip entered. Data: Salesman Number: 1, Product Number: 3, Product Value: 50
.0
1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit
Make your selection: 2
Monthly Report
-----
Salesman/
Product S1      S2      S3      S4      Total
P1      0.0      0.0      0.0      0.0      0.0
P2      0.0      0.0      0.0      0.0      0.0
P3     100.0      0.0      0.0      0.0     100.0
P4      0.0      0.0      0.0      0.0      0.0
P5      0.0      0.0      0.0      0.0      0.0
Total   100.0      0.0      0.0      0.0

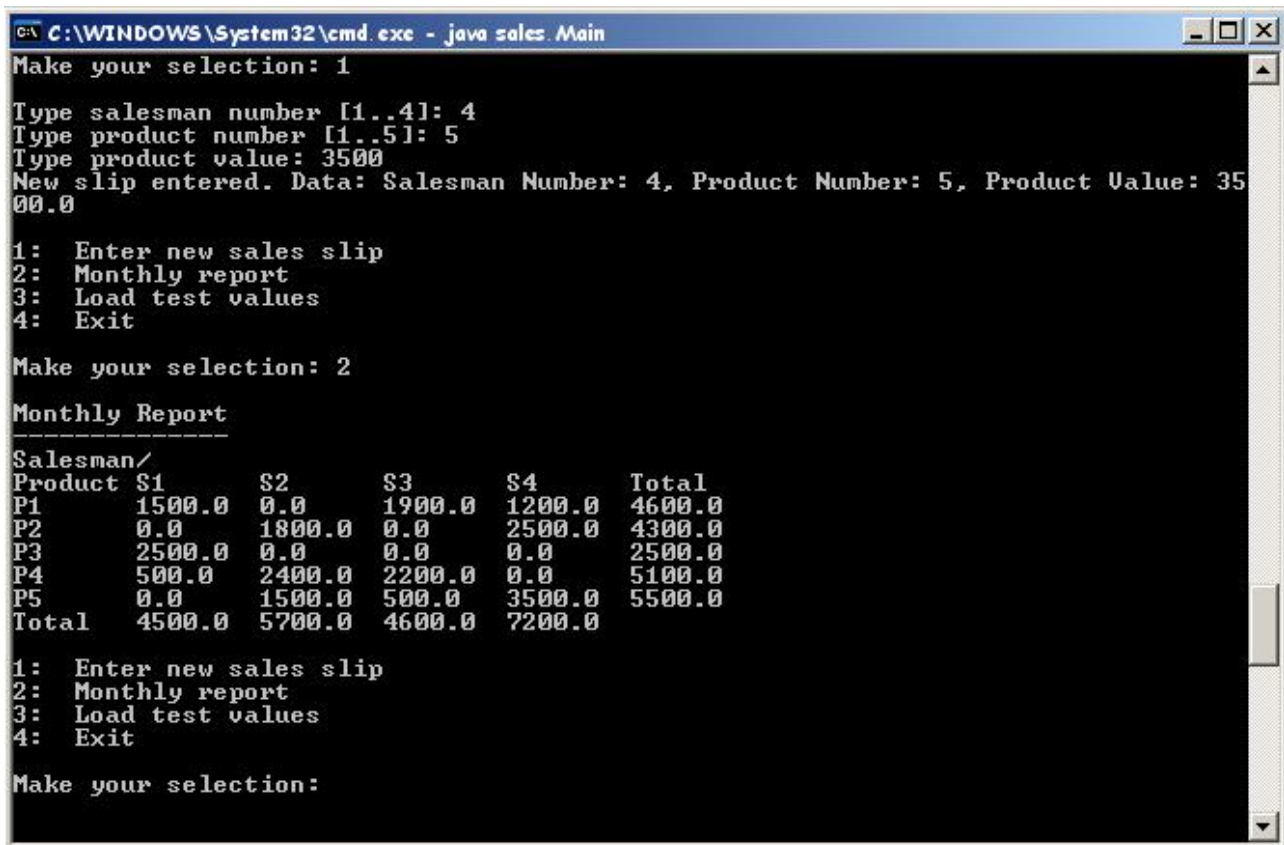
```

At this screenshot we can see a simple example of a salesman, who enters his sales slips referring at the third product and each one has a product value of 50 €. The monthly report shows the total of the amounts that salesman one sold. Making it more complex, you may enter the following data in order to ensure the required operations of the program.

The proposed scenario is the following:

- Salesman 1 sells 1500 € of product 1, 2500 € of product 3, and 500 € of product 4.
- Salesman 2 sells 1800 € of product 2, 2400 € of product 4, and 1500 € of product 5.
- Salesman 3 sells 1900 € of product 1, 2200 € of product 4, and 500 € of product 5.
- Salesman 4 sells 1200 € of product 1, 2500 € of product 2, and 3500 € of product 5.

The monthly report appears at the next screenshot, showing the expected result.



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\System32\cmd.exe - java sales.Main". The application prompts the user to "Make your selection: 1". The user enters "1", and the application prompts for "Type salesman number [1..4]: 4", "Type product number [1..5]: 5", and "Type product value: 3500". It then displays "New slip entered. Data: Salesman Number: 4, Product Number: 5, Product Value: 3500.0". The application prompts for another selection, and the user enters "2". The application then displays a "Monthly Report" table.

```
Make your selection: 1
Type salesman number [1..4]: 4
Type product number [1..5]: 5
Type product value: 3500
New slip entered. Data: Salesman Number: 4, Product Number: 5, Product Value: 3500.0

1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit

Make your selection: 2

Monthly Report
-----
Salesman/
Product $1      $2      $3      $4      Total
P1      1500.0  0.0    1900.0  1200.0  4600.0
P2      0.0    1800.0  0.0    2500.0  4300.0
P3      2500.0  0.0    0.0    0.0    2500.0
P4      500.0   2400.0  2200.0  0.0    5100.0
P5      0.0    1500.0  500.0   3500.0  5500.0
Total   4500.0  5700.0  4600.0  7200.0

1: Enter new sales slip
2: Monthly report
3: Load test values
4: Exit

Make your selection:
```

Source Code of Component 3***Main Class***

```

package sales;

import java.util.Vector;
import java.io.*;

public class Main {
    private static Vector salesSlips = new Vector();

    public static void getData() { // sets sales slip data
        Main.salesSlips.add(new salesSlip(1, 5, 500));
        Main.salesSlips.add(new salesSlip(1, 2, 200));
        Main.salesSlips.add(new salesSlip(1, 3, 350));
        Main.salesSlips.add(new salesSlip(2, 1, 100));
        Main.salesSlips.add(new salesSlip(2, 2, 120));
        Main.salesSlips.add(new salesSlip(2, 3, 340));
        Main.salesSlips.add(new salesSlip(3, 4, 1156));
        Main.salesSlips.add(new salesSlip(3, 3, 574));
        Main.salesSlips.add(new salesSlip(3, 1, 748));
        Main.salesSlips.add(new salesSlip(4, 4, 212));
        Main.salesSlips.add(new salesSlip(4, 3, 428));
        Main.salesSlips.add(new salesSlip(4, 2, 975));
    } // end of getData method

    public static void main(String[] args) {
        salesSlip mySlip = null;
        int selection;
        int salesManNumber;
        int productNumber;
        double productValue;
        double lineTotal = 0.00;
        double columnTotal = 0.00;
        Double[][] sales = new Double[6][5];

        while (true) {
            System.out.println();
            System.out.println("1: Enter new sales slip");
            System.out.println("2: Monthly report");
            System.out.println("3: Load test values");
            System.out.println("4: Exit");
            System.out.println("");
            System.out.print("Make your selection: ");
            try {
                selection = Integer.parseInt(new BufferedReader(new
InputStreamReader(System.in)).readLine());
            } // end try
            catch (Exception e) {
                selection = 0;
            } // end catch

```

```

switch (selection) {
    case 0: {
        break;
    } // end case 0
    case 1: {
        try {
            System.out.println("");
            System.out.print("Type salesman number [1..4]: ");
            salesManNumber = Integer.parseInt(new BufferedReader(new
InputStreamReader(System.in)).readLine());
            if (salesManNumber < 1 || salesManNumber > 4) {
                System.out.println("Invalid value. Try again");
                break;
            }
            System.out.print("Type product number [1..5]: ");
            productNumber = Integer.parseInt(new BufferedReader(new
InputStreamReader(System.in)).readLine());
            if (productNumber < 1 || productNumber > 5) {
                System.out.println("Invalid value. Try again");
                break;
            }
            System.out.print("Type product value: ");
            productValue = Double.parseDouble(new BufferedReader(new
InputStreamReader(System.in)).readLine());
            mySlip = new salesSlip(salesManNumber, productNumber, productValue);
            Main.salesSlips.add(mySlip);
            System.out.println("New slip entered. Data: " + mySlip);
        } // end try
        catch (Exception e) {
            System.out.println("Invalid format. Try again");
        } // end catch
        break;
    }
    case 2: {
        System.out.println("");
        System.out.println("Monthly Report");
        System.out.println("-----");

        for (int i = 0; i < 6; i++) { // make all places of the table equal to zero
            for (int k = 0; k < 5; k++) {
                sales[i][k] = 0.00;
            } // end columns for
        } // end lines for
        for (int i = 0; i < Main.salesSlips.size(); i++) { // populate the table
            mySlip = (salesSlip)Main.salesSlips.elementAt(i);
            salesManNumber = mySlip.getSalesPerson();
            productNumber = mySlip.getProductNumber();
            productValue = mySlip.getTotalValue();
            sales[productNumber-1][salesManNumber-1] += productValue;
        } // end for
        for (int i = 0; i < 5; i++) { // calculate the line totals
            lineTotal = 0.00;

```

```

        for (int k = 0; k < 4; k++) {
            lineTotal += sales[i][k];
        } // end columns for
        sales[i][4] = lineTotal;
    } // end lines for
    for (int i = 0; i < 4; i++) { // calculate the column totals
        columnTotal = 0.00;
        for (int k = 0; k < 5; k++) {
            columnTotal += sales[k][i];
        } // end columns for
        sales[5][i] = columnTotal;
    } // end lines for
    System.out.println("Salesman\r\nProduct\tS1\tS2\tS3\tS4\tTotal"); // start printing the
table
    for (int i = 0; i < 6; i++) {
        if (i < 5) {
            System.out.print("P");
            System.out.print(i+1);
            System.out.print("\t");
        } // end if
        else {
            System.out.print("Total\t");
        } // end else
        for (int k = 0; k < 5; k++) {
            if (i==5 && k==4) {
                // do nothing
            }
            else {
                System.out.print(sales[i][k] + "\t");
            }
        } // end columns for
        System.out.println(""); // when previous line is finished start a new line
    } // end lines for
    break;
}
case 3: {
    Main.getData();
    break;
}
case 4: {
    System.exit(0);
    break;
}
} // end switch
} // end while
//Main.getData();
} // end of main method

} // end of Main class

```


Assistant class salesSlip.java

```
package sales;

public class salesSlip {
    private int salesPerson;
    private int productNumber;
    private double totalValue;

    public salesSlip(int salesPerson_c, int productNumber_c, double totalValue_c) {
        salesPerson = salesPerson_c;
        productNumber = productNumber_c;
        totalValue = totalValue_c;
    } // end of constructor

    public int getSalesPerson() {
        return (salesPerson);
    } // end of getSalesPerson method

    public int getProductNumber() {
        return (productNumber);
    } // end of getProductNumber method

    public double getTotalValue() {
        return (totalValue);
    } // end of getTotalValue method

    public String toString() {
        return ("Salesman Number: " + salesPerson + ", Product Number: " + productNumber + ",
Product Value: " + totalValue);
    } // end of toString method
}
```

Component 4

Component 4 initial screenshot

```
Command Prompt - java storehouse.Main
C:\>cd Documents and Settings\Bill Balafas\Desktop\D00S Coursework 11_2005\storeHouse\build\classes
C:\Documents and Settings\Bill Balafas\Desktop\D00S Coursework 11_2005\storeHouse\build\classes>java storehouse.Main
Welcome to the StoreHouse

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 1

Enter product code: 100
Enter product description: Product A
Enter product quantity: 100
Added Product: Code: 100, Description: Product A, Available Quantity: 100

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 1

Enter product code: 200
Enter product description: Product B
Enter product quantity: 150
Added Product: Code: 200, Description: Product B, Available Quantity: 150

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 1

Enter product code: 300
Enter product description: Product C
Enter product quantity: 1000
Added Product: Code: 300, Description: Product C, Available Quantity: 1000

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 3

Code: 100, Description: Product A, Available Quantity: 100
Code: 200, Description: Product B, Available Quantity: 150
Code: 300, Description: Product C, Available Quantity: 1000
3 products found

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 2

Enter product code: 100
Code: 100, Description: Product A, Available Quantity: 100

1: Insert a new product
2: Display a product
3: Display all products
4: Exit

Enter selection: 1

Enter product code: 100
Enter product description: Product A
Enter product quantity: 100
A product with this code already exists
```

At the screenshot above we can see how this little application works and manages the products of the Storehouse.

When the application starts a welcome message is shown and the main menu appears with the basic options. By selecting option 1 we can add products and their characteristics as it is clear at the description of the coursework. If the insertion is successful, the program shows the *“Added Product:”* message with the product’s inserted values.

The program has the ability to show us all the products by selecting option 3 and at the last part of the screenshot we can see that the program controls and checks the integrity of the information that we may enter. For example if we add the product code 100 again, then there is a message that informs us that the product already exists in the table.

The main table which stores the information assures the maximum number of products which has to be 100, by the declaration of the table which is:

```
private static product[] products = new product[100];
```

The program finishes its operation by selecting the option 4 of the menu.

Source Code of Component 4

Main Class

```
package storehouse;

import java.io.*;

public class Main {
    private static product[] products = new product[100];

    public static void main(String[] args) {
        int selection, i, productsFound;
        boolean productFound = false;
        product myProduct = null;

        String code, description = new String();
        long quantity;
        System.out.println("Welcome to the StoreHouse");
        while (true) { // always return here and display menu until user selects 4
            System.out.println("");
            System.out.println("1: Insert a new product");
            System.out.println("2: Display a product");
            System.out.println("3: Display all products");
            System.out.println("4: Exit");
            System.out.println("");
            System.out.print("Enter selection: ");
            try { // in case parseInt or readLine throws an exception
                selection = Integer.parseInt(new BufferedReader (new
InputStreamReader(System.in)).readLine());
            } // end try
            catch(Exception e) {
                System.out.println("Invalid Selection");
                selection = 0; // selection=0 = case 0 = break and return to menu
            } // end catch
        }
    }
}
```

```

switch (selection) {
    case 0:
        break; // return to menu
    case 1:
        try { // in case an exception is thrown by the data entry
            System.out.println("");
            System.out.print("Enter product code: ");
            code = (new BufferedReader (new InputStreamReader(System.in)).readLine());
            System.out.print("Enter product description: ");
            description = (new BufferedReader (new InputStreamReader(System.in)).readLine());
            System.out.print("Enter product quantity: ");
            quantity = Long.parseLong(new BufferedReader (new
InputStreamReader(System.in)).readLine());
            productFound = false;
            for (i = 0; i < products.length; i++) {
                if (products[i] == null) {
                    // do nothing
                } // end if
                else {
                    if (products[i].getCode().equals(code)) { // check if a product with this code
already exists
                        productFound = true;
                        break;
                    } // end if
                } // end else
            } // end for
            if (productFound) {
                System.out.println("A product with this code already exists");
                break;
            } // end if
            for (i = 0; i < products.length; i++) {
                if (products[i] == null) { // find the first available place in the array (i.e. in case a
product was deleted from the array)
                    products[i] = new product(code, description, quantity); // add product to the array
                    System.out.println("Added Product: " + products[i]);
                    break;
                } // end if
                else {
                    // do nothing
                } // end else
            } // end for
        } // end try
        catch (Exception e) {
            System.out.println("Invalid format!");
            break;
        } // end catch
        break;
    case 2:
        try { // in case an exception is thrown by the data entry
            System.out.println("");
            System.out.print("Enter product code: ");
            code = (new BufferedReader (new InputStreamReader(System.in)).readLine());

```

```

    productFound = false;
    productsFound = 0;
    for (i = 0; i < products.length; i++) {
        if (products[i] == null) {
            // do nothing
        } // end if
        else{
            productsFound++;
            if (products[i].getCode().equals(code)) { // check if a product with this code exists
in the array
                productFound = true;
                myProduct = products[i];
                break;
            } // end if
        } // end else
    } // end for
    if (productFound) { // if a product with this code exists, print it
        System.out.println(myProduct);
    } // end if
    else if (productsFound == 0) {
        System.out.println("No products found"); // the array is empty
    } // end else if
    else {
        System.out.println("No product found with this code"); // the array is not empty but
no product found with this code
    } // end else
} // end try
catch (Exception e) {
    System.out.println("Invalid format!");
    break;
} // end catch
break;
case 3:
    System.out.println("");
    productsFound = 0;
    for (i = 0; i < products.length; i++) {
        if (products[i] == null) {
            // do nothing
        } // end if
        else {
            System.out.println(products[i]); // print product found
            productsFound++;
        } // end else
    } // end for
    System.out.println(productsFound + " products found"); // sum of products found or 0
if array empty
    break;
case 4:
    System.exit(0); // exit from program
    break;
} // end switch
} // end while

```

```
    } // end main method  
}  
} // end Main class
```

Assistant class product.java

```
package storehouse;  
  
public class product {  
    private String code = new String();  
    private String description = new String();  
    private long quantity;  
  
    public product(String code_c, String description_c, long quantity_c) {  
        code = code_c;  
        description = description_c;  
        quantity = quantity_c;  
    } // end of constructor  
  
    public String toString() {  
        return("Code: " + code + ", Description: " + description + ", Available Quantity: " + quantity);  
    }  
  
    public void setCode(String code_c) {  
        code = code_c;  
    }  
  
    public void setDescription(String description_c) {  
        description = description_c;  
    }  
  
    public void setQuantity(long quantity_c) {  
        quantity = quantity_c;  
    }  
  
    public String getCode() {  
        return(code);  
    }  
  
    public String getDescription() {  
        return(description);  
    }  
  
    public long getQuantity() {  
        return(quantity);  
    }  
}
```

Note for Components 3 & 4

The programs of Component 3 & 4 were written in NetBeans IDE 4.1 programming environment using Java edition 1.5.0_05. They can be also found in the submitted CD of the Coursework in the folder Programs and each folder containing the project folder is named Sales and Storehouse respectively. If NetBeans IDE 4.1 is used for reviewing them they can be accessed directly from File → Open Project selection from the menu.

----- The End -----