MINISTRY OF EDUCATION AND SCIENCE

OF THE REPUBLIC OF KAZAKHSTAN

KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGIES

INFORMATION SYSTEMS AND MANAGEMENT DEPARTNENT

Report

On the project work on the topic:

“Developing an Intranet using OOP principles”

Submitted by: Students of 2nd year, FIT

Alinur Sabit, Dariya Mukhatova, Maksat Sangerbayev

Course: Object-oriented Programming and Design

ID: 16BD02113, 16BD02130 ,16BD02024

Submitted to: Shamoi P.S.

Almaty 2017

“Developing an Intranet using OOP principles”

The Intranet was developed in three steps: building use case diagram, class diagram and finally writing program code on java language. The Intranet performs well all functions. The project corresponds to all principles of the OOP.

The task: To create University system using OOP principles.

The main components:

- TopCoder UML Tool;
- Eclipse IDE;
- Java Language;

The Intranet system has the following features:

- The main classes are serializable and comparable;
- Ability to create new objects of users;
- Ability to work with files;

Contents

Introduction

An Intranet is a University Information System used by university staff and students. All personal data about users, all courses and schedule are stored in database of Intranet. The aim of the project work on developing Intranet system is to properly use theoretical knowledge obtained at he lectures in practice on laboratory exercises.

The Intranet developing consist of 3 main steps. Firstly, the general plan of the whole project was done using use cases diagram on the TopCoder UML Tool. Main actors: Admin, Manager, Teacher, Student and Executor and their actions in the Intranet, use cases: Viewing course files, putting marks, approving registration, etc. were fully described.

Secondly, the class diagram of the project was drawn, again using Top Coder UML. On this step of the project use case diagram from the precious step was used as a template for creating classes. The aim of class diagram is to create classes (actors), interfaces, abstract classes, enum. All necessary fields and methods have been defined and described. Furthermore, the finished class diagram was used to generate a java code.

Finally, we started working on the main step, the Intranet itself. The work was complete on Eclipse – integrated development environment (IDE). All classes were created on code generating step. Step by step we implemented OOP principles which are encapsulation, abstraction, inheritance and polymorphism. Moreover, project requires the usage of our knowledge we got on OOP course and implement interfaces, further methods: Serializable, Comparable, Cloneable, equals(), hashcode(), toString(), etc.

Steps

Use Case Diagram

The use case diagram is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well (Gemino, 2009).

Use case diagram is drawn on the TopCoder UML Tool: an easy to use, consistent modeling tool for use in Design and Development competitions. The new tool was designed and developed entirely by the TopCoder Community to model sequence, class, use case, and activity diagrams. In addition, all those elements can have documentation easily attached to them.

The use case diagram entirely describes the functionality on the Intranet, the relationship between actors and use cases. It is a good planning in a big project, because everything is drawn in detail, so that programmers will have a clear vision of further steps. It is noticeable on the figure1, that all steps, actions and relationships are shown.

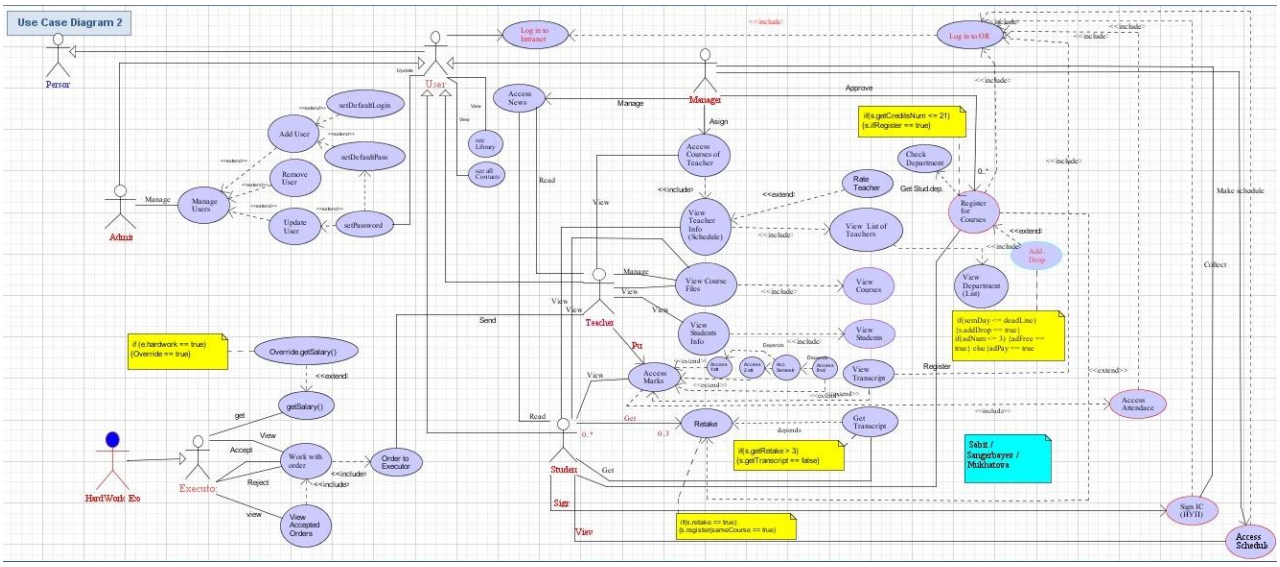


Figure 1. Use Case Diagram of the Intranet

There is an abstract class Person extended by all users of the Intranet. There 5 actors: Admin, Manager, Teacher, Student and Executor; 41 uses cases. Each actor has some relations with use cases. Main are: Viewing course files, putting marks and accessing marks, registering for courses and approving them, sending orders to and working with these orders, managing users, and the last but the most important is a user verification. Comments provided where needed. Association, directed association, include and extend relationships were used between actors and use cases.

Class Diagram

The class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their fields, methods, and the relationships among objects (Scott, 2009).

The class diagram is the main building block of object-oriented modelling. It is used for general conceptual modelling of the systematic of the application, and for detailed modelling generating the programming code from models.

The diagram helps to build classes: interfaces, abstract classes, users, enum. In properties we could create attributes, operations (methods), write their data type. It is useful to write the documentation of class diagram, as everything will be described and commented in the project. Building a good class diagram in a big project is a perfect way of modeling a project, because everything is written in detail, so that coders will just have to generate code and work on existing project classes. We can see in the class diagram on the figure2, 3, 4 that all classes are described.

Classes in class diagram were separated into three packages according to their specifics: Users, System and Enum Packages. Last Package contains all 6 enumerations of the system.

Users Package consists of:

- Abstract class Person
- Employee
- Admin
- Executor
- Manager
- Teacher
- Student

The main abstract class in the system is Person class, which is inherited by all other actor classes of Users Package.

Let's start from Student class. It has two hashMaps, marks and attestation. Marks hashMap is used to see and setMark () (by Teacher) marks for courses of a student, to getTranscript (), viewCourses () and to registerForCourses () too, where mark for course equals to null by default, meanwhile attendance is used to have a look at students' attendance and putAttemdance (). By the way, there is an opportunity for a student to rateTeacher().

Teacher, basically, has almost the same methods as Student has. The main differences are following: Teacher can sendOrder () to an Executor, viewAllStudents () list, addFile () and deleteFile () in course files. For sure, Teacher can getRating().

Manager can manage news in Database by methods addNews(News newNews), deleteNews(News oldNews) and showNews(). The next Actor is Executor who getOrders() and can acceptOrders()/rejectOrders() and viewOrders(). Moreover, this user can setSalary(double bonus).

And the last and most important staff among users is Admin. This actor is able to manage users, or more precisely addUsers(), updateUser() and removeUser(). In addition, Admin should setUserLogin() and setUserDefaultPassword() for all users. All of Admin's methods are addressed

directly to the Database class in System Package. Executor class is one of the simplest classes in the system, because its functional consist of working with orders.

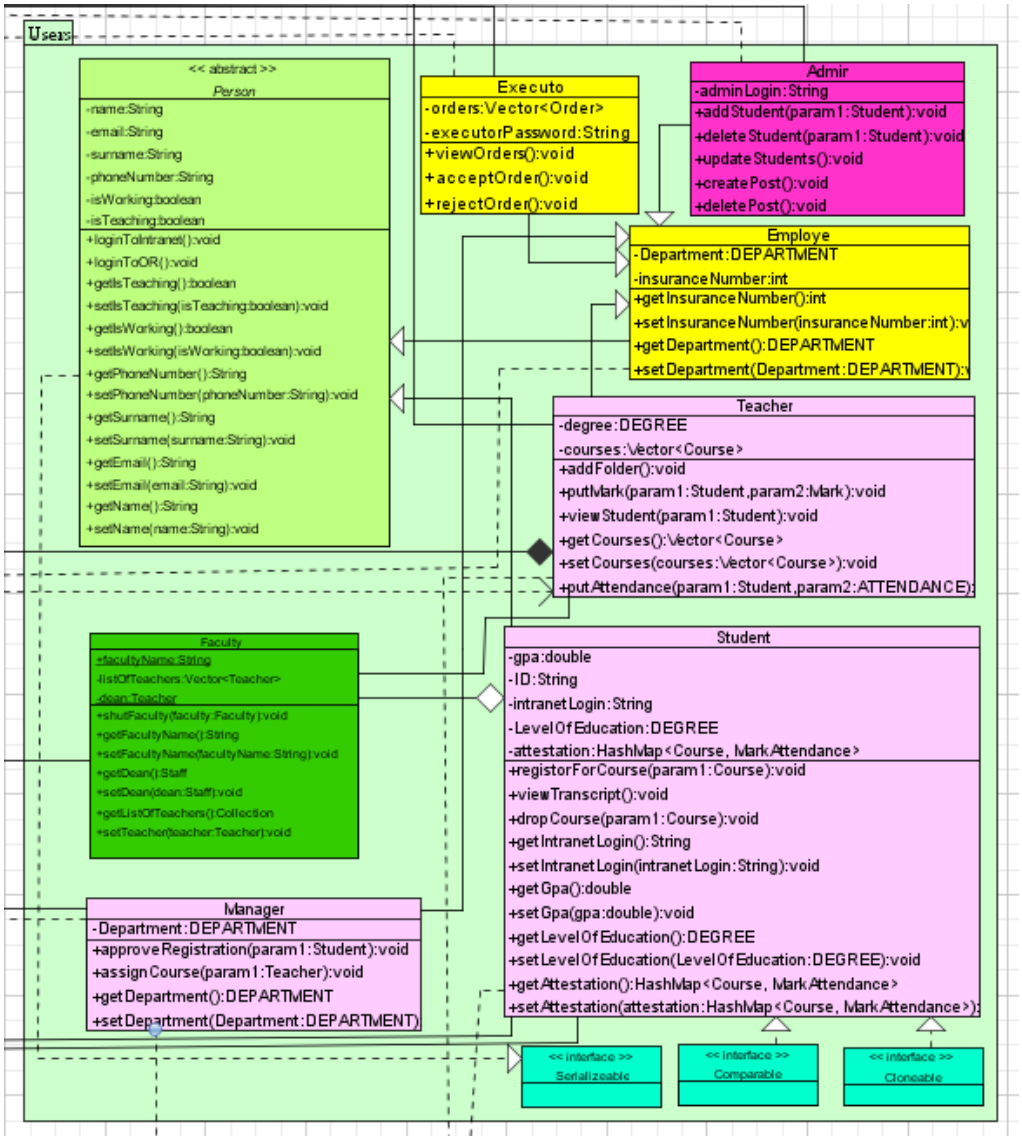


Figure 2. Users package

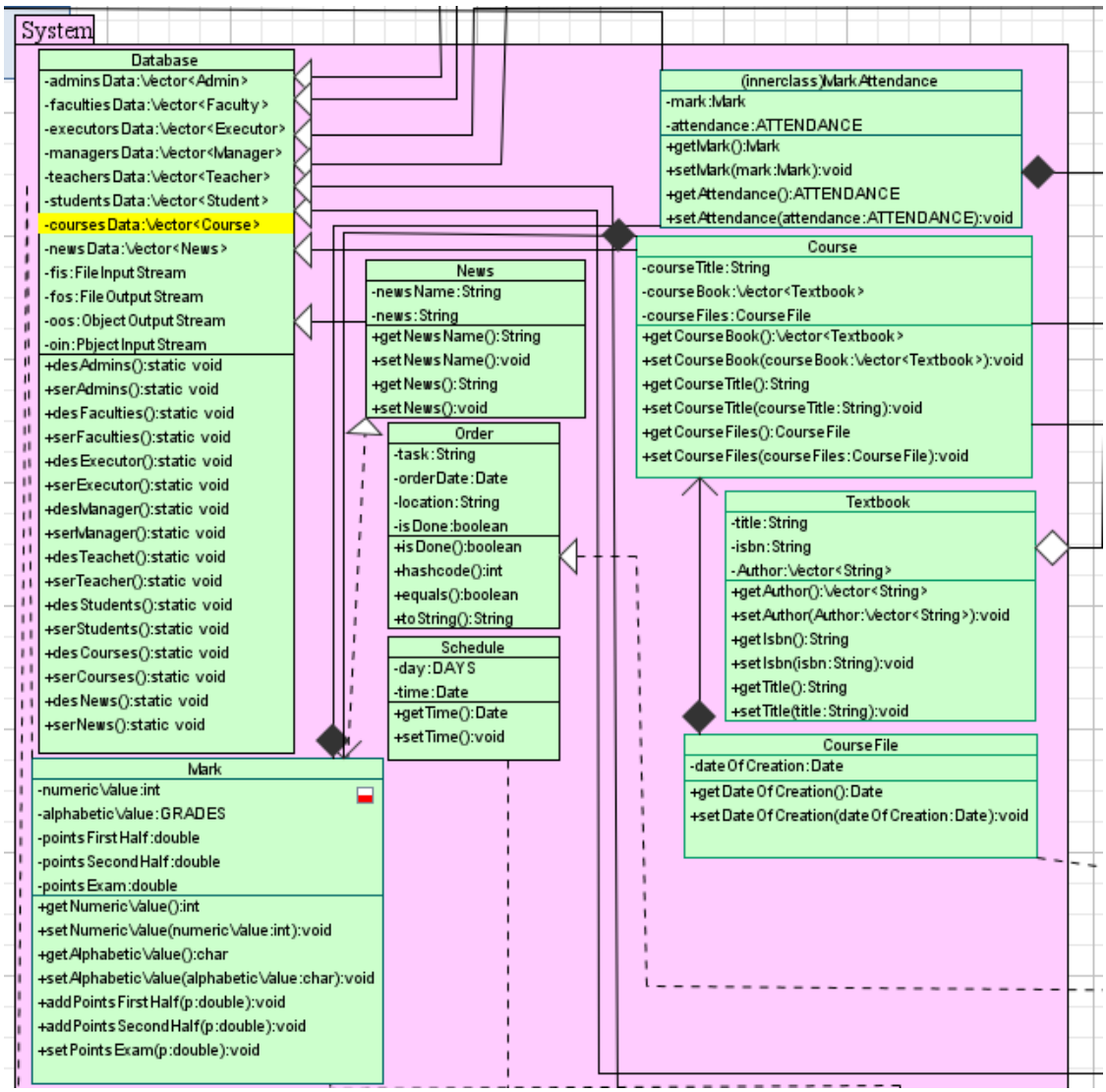
System Package consists of:

- Database
- Course
- CourseFile
- Textbook
- Mark
- MarkAttendance
- Order
- Schedule
- News
- Faculty

Database class is the class created for serializing and deserializing Vectors of objects of Classes, what makes it the main storage system of the Intranet.

Mark class has such fields as numericValue and alphabeticValue of marks, pointsFirstHalf, pointsSeconfHalf and pointsExam for 2 attestations and final exam points, and add, set and getter.

Class MarkAttendance was created for controlling the attendance rate of students. CourseFile class describes the course by giving date of creation and description. Class Textbook contains data about course book. News class is the simplest class in this project, as it requires only adding new String of news.

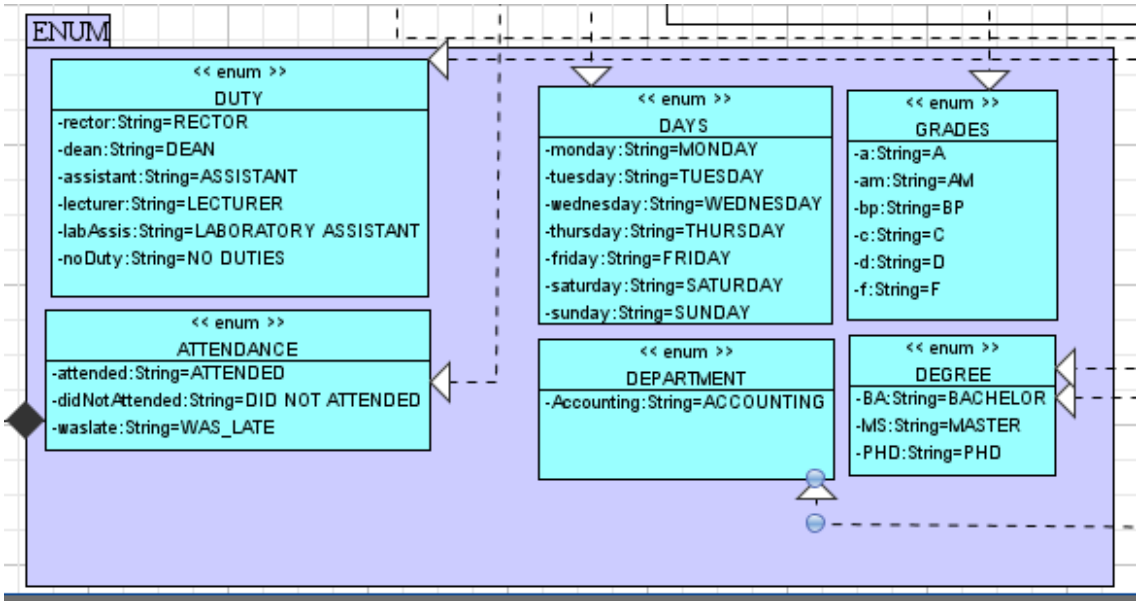


3. System package

Enum Package consist of:

- ATTENDANCE
- DAYS
- DEGREE
- DEPARTMENT
- DUTY
- GRADE

These enumeration are used in both Users and System mode.



4. Enum package

As a result of thorough work on the structural design of the Intranet using UML diagrams – Use Case Diagram and Class Diagrams, we reached our planned design of the project work.

Creating a Java code

Writing Java code is the final step of the whole project. In this step we write code for all methods. In the process of developing the system we worked on Eclipse IDE.

In order to save users data, course files and marks, etc. we use serialization that helps to serialize objects into text file, so that all data will be saved and reachable each time we run the Intranet program. There are three serialize methods provided as an example of serialization.

```
public static void desAdmins() throws IOException, ClassNotFoundException{
    try {
        fis = new FileInputStream("adminsData.out");
        oin = new ObjectInputStream(fis);
        adminsData = (Vector<Admin>) oin.readObject();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    finally {
        fis.close();
        oin.close();
    }
}
public static void serAdmins()throws IOException{
    try {
        fos = new FileOutputStream("adminsData.out");
        oos = new ObjectOutputStream(fos);
        oos.writeObject(adminsData);
    }catch(Exception e) {
        e.printStackTrace();
    }finally {
        oos.close();
        fos.close();
    }
}
```

Figure 5. Database class desAdmins() and serAdmins()

Piece of code from Admin class:

```
public void addUser(Object o) {
    if(o instanceof Student) {
        Student s = (Student)o;
        Database.studentsData.add(s);
    }

    if(o instanceof Teacher) {
        Teacher t = (Teacher)o;
        Database.teachersData.add(t);
    }

    if(o instanceof Manager) {
        Manager m = (Manager)o;
        Database.managersData.add(m);
    }

    if(o instanceof Executor) {
        Executor e = (Executor)o;
        Database.executorsData.add(e);
    }
}
```

Figure 6. Admin class addUser() method

Example of hashcode() method for all classes. The method from Student class:

```
@Override
    public int hashCode() {
            final int prime = 31;
            int result = super.hashCode();
            result = prime * result + ((ID == null) ? 0 : ID.hashCode());
            result = prime * result + ((attestation == null) ? 0 : attestation.hashCode());
            long temp;
            temp = Double.doubleToLongBits(gpa);
            result = prime * result + (int) (temp ^ (temp >>> 32));
            result = prime * result + ((intranetLogin == null) ? 0 : intranetLogin.hashCode());
            result = prime * result + ((levelOfEducation == null) ? 0 : levelOfEducation.hashCode());
            result = prime * result + ((marks == null) ? 0 : marks.hashCode());
            result = prime * result + ((specialty == null) ? 0 : specialty.hashCode());
            result = prime * result + yearOfStudy;
            return result;
    }
```

Figure 7. Student class hashcode() method

Example of equals() method for all classes. The method from Teacher class:

```
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (!super.equals(obj))
                return false;
        if (getClass() != obj.getClass())
                return false;
        Teacher other = (Teacher) obj;
        if (courses == null) {
                if (other.courses != null)
                        return false;
        } else if (!courses.equals(other.courses))
                return false;
        if (degree != other.degree)
                return false;
        return true;
}
```

Figure 8. Teacher class equals() method

Example of accessors - get() and mutators - set() methods, also there is a toString() method: The code piece is taken from Textbook class:

```
public void setAuthor(String Author) {
    this.author.add(Author);
}

public String getIsbn() {
    return isbn;
}

public void setIsbn(String isbn) {
    this.isbn = isbn;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String toString(){
    return "Textbook title: " + title + ". ISBN: " + isbn + ". Author(s): " + getAuthor();
}
```

Figure 9. Textbook class get() and set(), toString() methods

Example of class constructors. The Executor() constructor from Executor class:

```
public Executor(String name, String email,
        String surname, String phoneNumber,
        boolean isWorking, boolean isTeaching,
        Vector<Order> orders) {
    super(name, email, surname, phoneNumber,
    isWorking, isTeaching);
    this.orders = orders;
}
```

Figure 10.Executor class Executor() constructor

Message Dialog of main() method from IntranetTester class. This window appears when the tester class is opened:

Figure 11. Tester class Message Dialog

The main part of the Intranet system is writing driver class. In driver class user of the system can choose admin mode or user mode. The main difference in modes is that in first one is created only for admins who can manage users. In the second mode user can choose manager, teacher or student mode. Respectively, users can take advantage of the opportunities, number of methods, provided to them.

```
menu: while(true) {
    System.out.println("********MENU********");
    System.out.println("1 --- Enter as Admin");
    System.out.println("2 --- Enter as User");
    System.out.println("3 --- Exit");
    int choise = Integer.parseInt(br.readLine());
    if(choise = 1){
        adminMode:
            System.out.println("Login: ");
            String login = br.readLine();
            System.out.println("Password: ");
            String password = br.readLine();
            Admin currentAdmin = new Admin();
            for(Admin a: Database.adminsData) {
                if(a.getAdminLogin().equals(login) && a.getAdminPassword() == password.hashCode()) {
                    System.out.println("Welcome!");
                    currentAdmin = a;
                    break;
                }
            }
            if(currentAdmin.getName() != null) {
                System.out.println("What would you like to do?");
                System.out.println("1 --- Add user");
                System.out.println("2 --- Delete student");
                System.out.println("3 --- Change password");
                int adminChoice = Integer.parseInt(br.readLine());
                switch(adminChoice) {
                    case 1:
                        System.out.println("Choose whom to add");
                        System.out.println("1 --- Student");
                        System.out.println("2 --- Teacher");
                        System.out.println("3 --- Manager");
                        int addChoice = Integer.parseInt(br.readLine());
                        switch(addChoice) {
                            case 1:
```

Figure 12. Tester class: menu navigation. The piece of code of Admin mode

Documentation

ENUM	Constants
ATTENDANCE	ATTENDED, DID_NOT_ATTENDED, WAS_LATE
DAYS	MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
DEGREE	BACHELOR, MASTER, PHD
DEPARTMENT	ACCOUNTING, OR
DUTY	RECTOR, DEAN, ASSISTANT, LECTURER, LABORATORY_ASSISTANT, NO_DUTIES
GRADES	A, AM, BP, B, BM, CP, C, CM, DP, D, F

Conclusion

As a result, we could develop full functional Intranet System for university. Three steps of building this system were performed step by step – designing UML Use Case diagram as project planning, building Class Diagram and generating the code, and finally, implementation of diagrams in Java programming language.

In Use Case diagram we created 5 main actors (users): Admin, Teacher, Student, Manager and Executor. In the second diagram we wrote all fields and methods for all classes.

The aim of the project was to thorough use of our knowledge in OOP&D in practice, developing our skills. Group of Hard Working Executors could execute all the goals.

References

1. Gemino, A., Parker, D. (2009) "Use case diagrams in support of use case modeling: Deriving understanding from the picture", *Journal of Database Management*, 20(1), 1-24.
2. Scott W. Ambler (2009) “UML 2 Class Diagrams”, *Webdoc 2003-2009*. Accessed Dec 2, 2009