Report.

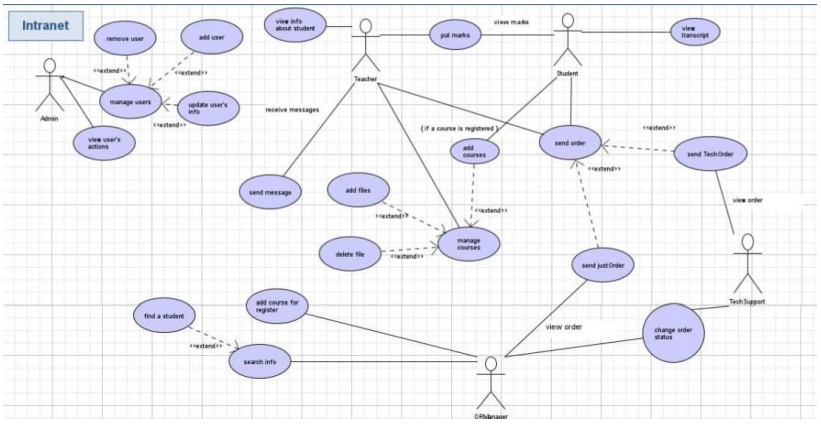
Abdikarimova Dariga
Adilbek Abylaikhan

The subject of this report is a team project in which task was to create some modelling type of University intranet system. The aim of the project was to implement and use OOP features in real-life project conditions.
On 18th of November our team firstly met each other and we tried to start teammate relationship adjustment. The reason for making this decision is that we think that distance teamwork needs extra connection between students to avoid some internal disagreements and disrespectful attitude to the common project. After this meeting Dariga started creating Telegram chat to exchange ideas and send some files, fragments of code, team channel in the Microsoft Teams for doing video-sessions with screen demonstration, and GitHub repository to upload and download project files.

According to the task, The University intranet system has special functionality and features that we needed to implement.
They are:

- Separate log in by roles (admin, student, teacher,executor, OR Manager)
- Creating DataBase
- Serialization and deserialization
- Getting connected with each other classes
- Opportunity to work with files (.out, .txt)
- Special opportunities for every actor
- Ability to create objects related to specific class

First of all, to accomplish this task we need to visualize it and make some structure of our project. It means that the first real step of our project was doing UML diagrams, including UseCase and Class Diagram, which help us to correctly manage the future coding part of the job and responsibility distribution at all. We choose the “TopCoder UML” program as our tool to implement the visual part and structurizing of the project. During the first meeting every member of the group drew his/her own draft versions according to his/her own vision. After a long discussion and decision making process we choose Dariga’s version as the main structure of the project, but Abylaikhan and Zhanym have their ideas that were taken into account during the meeting and were included to the final draft version in the TopCoder.



First of all, we'd like to start from UML diagram, it consists of use case and class diagram. Here's the use case, which has 5 actors: admin, teacher, student, TechSupport and ORManager. Each of them have their own responsibilities, fields and methods.

There's a class diagram. All used classes, interfaces and enumerations are represented here. As you can see, types of connections are also shown.

- **ADMIN:**
  Admin admin can **add** and **delete users**, as well as **updating information**.
- **PERSON:**
  Person has **first name**, **last name** and **gender**:
  - male;
  - female.
- **USER:**
  User is an abstract class which extends fields from Person. Each user can log into the system using a **login** and **password**. Furthermore, they can **leave a comment** to the New added by OR Manager. When a user enters the system, the system immediately determines who the user is by the classes:
  - **Teacher;**
  - **Student;**
  - **ORManager;**
  - **Executor.**
- **EMPLOYEE:**
  Employees have a **salary** which they can **get** and **set**.
- **TEACHER:**
  Teacher has **fields** which is:
  - **Title:**
    - Lector;
    - Tutor;
    - Senior lector;
    - Professor.

- **Faculty:**
  - FIT;
  - BS;
  - ISE;
  - KMA;
  - FOGI;
  - MCM.
- **Message (array list):**
  - Topic;
  - Text;
  - Date;
  - Owner.
- **Student (hashmap)**

Also Teacher have **methods**:

- **Put Mark:**
  - **First;**
  - **Second;**
  - **Third;**
  - **To Grade.**
- **Add Course;**
- **View Course;**
- **Get Course;**
- **Add Course File;**
- **Delete Course File;**
- **Add Student;**
- **Delete Student.**

- **<u>OR MANAGER:</u>**

OR Manager has field:

- **Orders:**
  - **Topic;**
  - **Description;**
  - **Date;**
  - **Order Number;**
  - **Status;**
  - **Compare To;**
  - **Responsibility;**
  - **View Order Status.**
- And **Methods**:
  - **Add Course;**
  - **Get Student By Gpa;**
  - **Get Student By Name;**
  - **Get Teacher By Name;**
  - **Add New;**

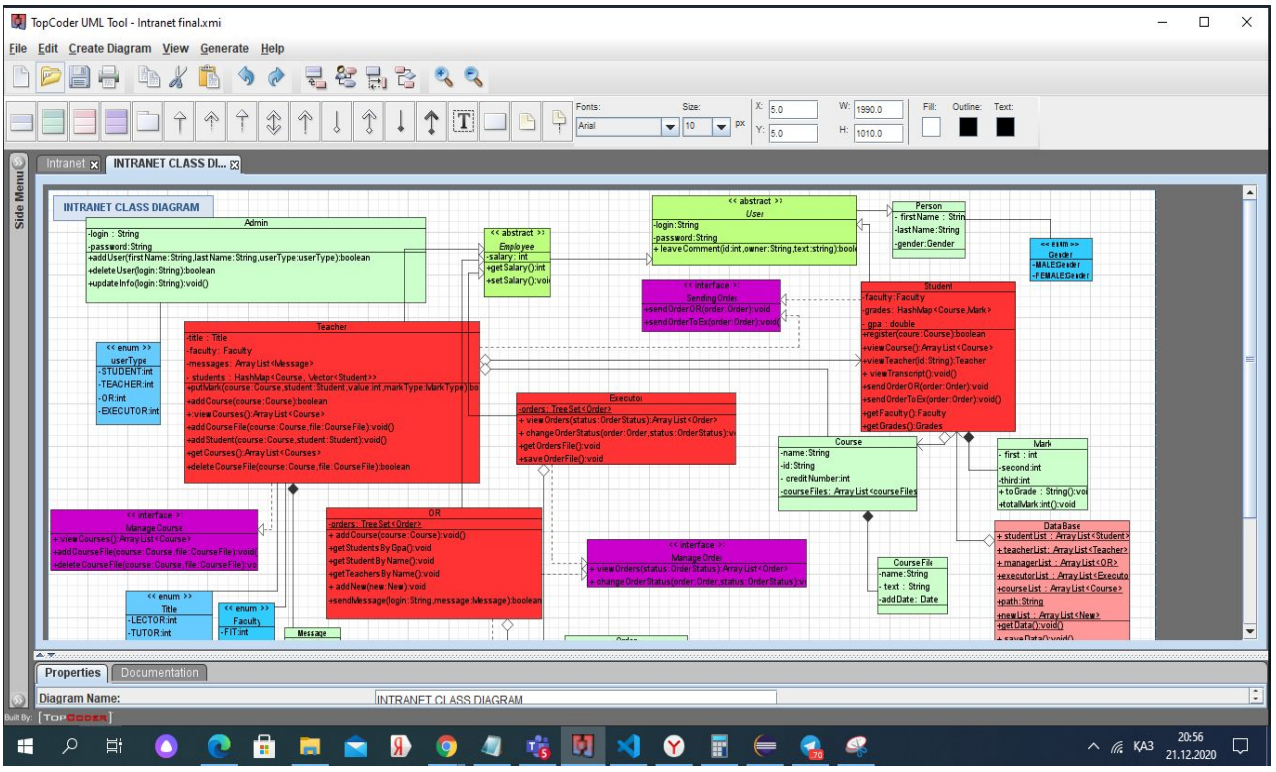
        - **Send Message.**
- **EXECUTOR:**

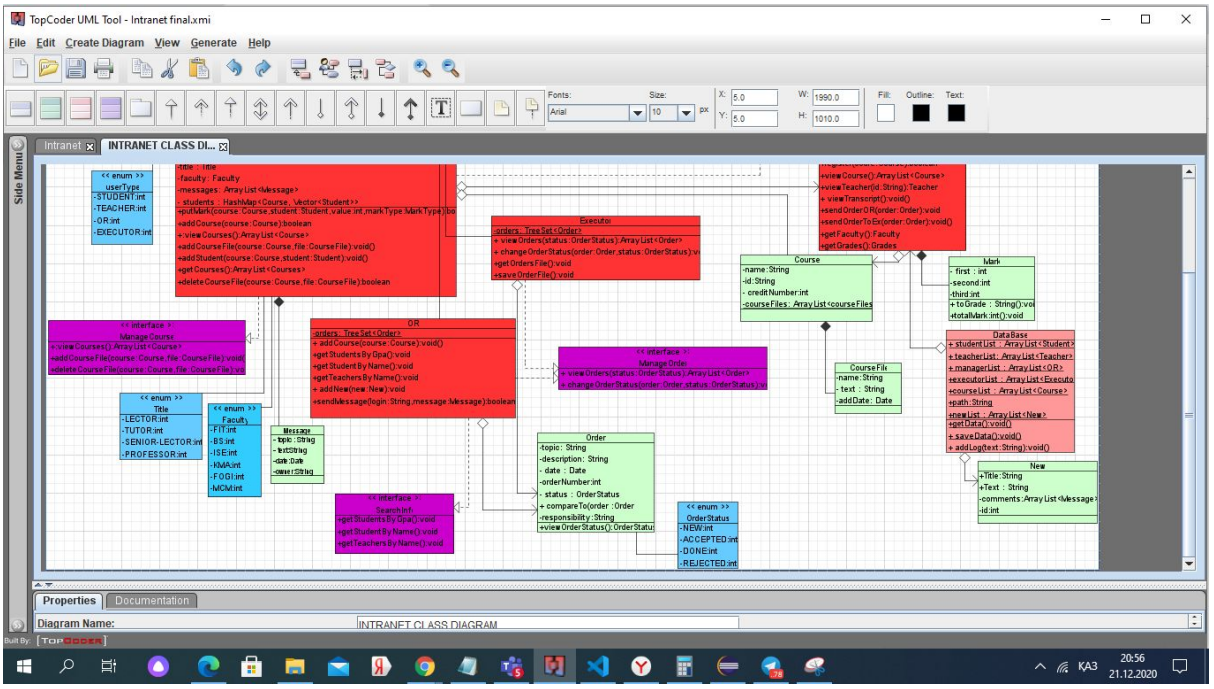
  As the field:
    - **Orders.**
        - **Order Status**
            - **NEW;**
            - **ACCEPTED;**
            - **DONE;**
            - **REJECTED.**

  As the Methods:
    - **View Orders;**
    - **Change Order Status;**
    - **Get Orders File;**
    - **Save Order File.**
- **STUDENT:**

  As the Fields:
    - **Faculty;**
    - **Grades;**
    - **GPA.**

  As the Methods:
    - **Register;**
    - **View Course:**
        - **Name;**
        - **ID;**
        - **Credit Number;**
        - **Course Files:**
            - **Name;**
            - **Text;**
            - **Add Date.**
    - **View Teacher;**
    - **View Transcript;**
    - **Send Order OR Manager;**
    - **Send Order to Executor;**
    - **Get Faculty;**
    - **Get Grades.**

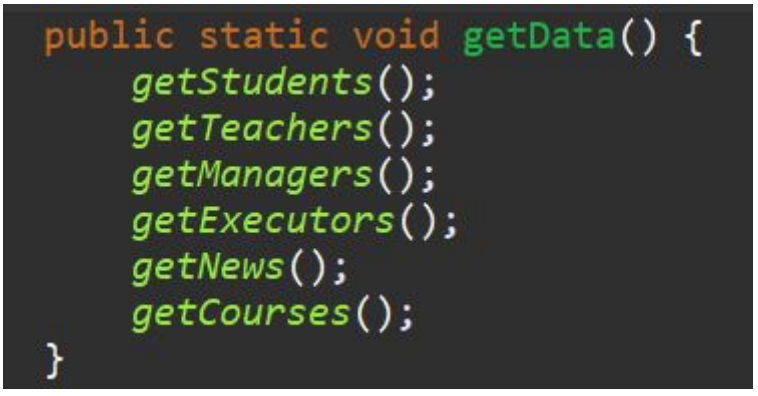

TopCoder UML Tool - Intranet final.xmi
File Edit Create Diagram View Generate Help
Intranet
INTRANET CLASS DI...
INTRANET CLASS DIAGRAM
Admin
Teacher
Executor
Course
Student
Person
Mark
DataBase
Properties
Documentation
Diagram Name:
INTRANET CLASS DIAGRAM
Built By:
20:56
21.12.2020


TopCoder UML Tool - Intranet final.xmi
File Edit Create Diagram View Generate Help
Intranet
INTRANET CLASS DI...
Properties
Documentation
Diagram Name:
INTRANET CLASS DIAGRAM
Built By:
20:56
21.12.2020

CODE IMPLEMENTATION

For the coding part we decided to choose **Java language** and as editor we use **Eclipse IDE** tool. In this block, we will represent you some significant fragments of our code.

The way of **saving data**: We have a class named **DataBase**, where all data related to each course, student, teacher and etc are stored in. When we run our **Controller,** objects from .out files are deserialized and we work with received data. After doing some work the controller fixs our edited data and **serializes** it. For doing these operations we have methods like **saveData()** and **getData()**.



Deserialization of students:

```
private static void getStudents() {
    try {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(PATH + STUDENTS));

        studentList = (ArrayList<Student>) ois.readObject();

        ois.close();

    }
    catch (ClassNotFoundException e) {
        System.out.println(STUDENTS + ": " + EXCEPT_CLASS);
    }
    catch (FileNotFoundException e) {
        System.out.println(STUDENTS + ": " + EXCEPT_FILE);
    }
    catch (IOException e) {
        System.out.println(STUDENTS + ": " + EXCEPT_IO);
    }

}
```

At the same time, DataBase uses a singleton pattern in case of keeping it in exemplar.

```
private DataBase() {

}

public static DataBase getInstance() {
    return INSTANCE;
}
```

In Controller the main menu is a method named **begin().** Depending on one or another choice the system gets information about the type of User and open corresponding menu.

```
public void begin() {
    System.out.println("Are you admin or user?");

    String ans = sc.nextLine().toLowerCase();

    if (!(ans.equals("user") || ans.equals("admin")))
        return;

    System.out.println("Enter your login and password");

    String login = sc.nextLine().toLowerCase();
    String password = sc.nextLine();

    switch (ans) {
        case "admin":
            workAsAdmin(login, password);
            break;
        case "user":
            workAsUser(login, password);
            break;
    }
    DataBase.saveData();

}
```

Admin is a user, which is different and separate from other users and which has only a login and password. Moreover, admin has the right to **create** new users and **delete** them. Also, administrators can monitor the actions of users by using a corresponding **log file** where every chosen option of users is fixed and can be outputted.
The fragment of code with an implemented method of adding new users.

```
public boolean addUser(String firstName, String lastName, UserType userType) {
    switch(userType){
        case STUDENT:
            if(addStudent(firstName,lastName))
                return true;
            break;
        case TEACHER:
            if(addTeacher(firstName,lastName))
                return true;
            break;
        case OR:
            if(addManager(firstName,lastName))
                return true;
            break;
        case EXECUTOR:
            if(addExecutor(firstName,lastName))
                return true;
            break;
    }
            return false;
}
```

LogFile is a .txt file which keeps information about any actions and its time. In this case, we use **BufferedWriter** with the path and its action.

```
public static void addLog(String text) {
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(PATH + Log, true));

        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd.MM.yy HH:mm");

        bw.write(dtf.format(LocalDateTime.now())+ " - " + text + "\n");

        bw.flush();
        bw.close();
    }
    catch (IOException e) {
        System.out.println(EXCEPT_IO);
    }
}
```

User is an abstract person/class who has its auto-generated **login** (name_firstName) and default **password**. Every User can **read news** and **leave comments** under them. **New** is a separate class which has its own ArrayList of messages which is comments. And Message is also a separate class.

Marks:

OR Manager can add a course and register it.

```
public void  addCourse(Course course) {
    if(!DataBase.courseList.contains(course))
    DataBase.courseList.add(course);
}
```

Every Teacher has a **HashMap<Course, Vector<Student>>**, where teachers store a list of students for every course. When a teacher registers for the course his/her presence in DataBase is checked firstly (added by OR Manager). At the beginning the vector of students is empty.

```
public boolean addCourse(String id) {
    for(Course course : DataBase.courseList) {
        if(course.getId().equals(id)) {
            students.put(course, new Vector<Student>());
            return true;
        }
    }
    return false;
}
```

Every student has its own **HashMap<Course, Mark>**. Where Course is key and Mark is value.

Students **register** for the course.

```
public boolean register(String id) {
    for(Course course : DataBase.courseList) {
        if(course.getId().equals(id)) {
            for(Teacher teacher : DataBase.teacherList) { //if there exists teacher who have this course
                if(teacher.getCourses().contains(course)) {
                    teacher.addStudent(this, course); //teacher[course].addStudent
                    grades.put(course,new Mark());  //initialize course in list of his grades
                    return true;
                }

            }
        }

    }
    return false;
```

The process of **putting marks** to the student by the teacher. Total mark=first+second+third;

```
public boolean putMark(Course course, Student student, int value , MarkType markType) {
    for(Course key : students.keySet()) {
        if(key.equals(course)) {
                    student.grades.get(course).setMark(value, markType);
                    return true;

        }
    }
    return false;
}
```

```
public class Mark implements Serializable{
    int first;  //first att
    int second;  //second att
    int third;  //final
```

Orders.

Teachers and Students can **send orders to Executors and OR Managers**. Every type of order (OR or ToEx) is implemented separately.

```
public interface SendingOrder {

    public void sendOrderOR(Order order);

    public void sendOrderToEx(Order order);
}
```

Also every order has its own status: **NEW, ACCEPTED, REJECTED, DONE**.

At the beginning orders status is NEW and no one takes responsibility for it. However, Executor(TechSupport) and OR Manager have static TreeSet<Order> which can sort orders by their date.

```
private String responsibility;

{
    this.orderNumber = numberOfOrder++;
    this.status = OrderStatus.NEW;
    setResponsibility("N/A");
}
```

```java
public int compareTo(Order o) {
    if (getDate().before(o.getDate())) {
        return 1;
    }
    if (getDate().after(o.getDate())) {
        return -1;
    }
        return 0;
}
```

Messages

OR Managers can **send messages** to the teacher and teachers' **arraylist of messages** is fulfilled.

```java
public boolean sendMesage(String login,Message message) {
    for(Teacher teacher : DataBase.teacherList) {
        if(teacher.getLogin().equals(login)) {
            teacher.receiveMessage(message);
            return true;
        }
    }
    return false;
}
```

```java
public Message(String topic, String text) {
    this.date = Calendar.getInstance().getTime();
    this.topic = topic;
    this.text= text;
    this.owner = "OR"; //owner will change when we use message as comment
}
```

News

Every User can **read and watch News** which are added only by the OR Manager. Furthermore, Users can leave comments to every single new in this block.

Problems:

In general, due to the great management job represented by our team leader Dariga the process of work was really easy and without huge failures. However, several unexpected troubles happened during the whole process.

The first problem was connected with responsibility and task distribution. Because none of us knows at the beginning our levels of programming ability.
The next problem was connected to the coding part. The problem was that we could not easily split a single class into one person, because almost every class is connected with each other and that was quite hard for us to assign responsibilities.
The last problem appeared suddenly and unexpectedly. The thing is that our teammate Zhanym gets low points for the end term exam and does not pass the Final exam, and we have structured our project management including her abilities and this situation is a little bit confusing for us.

Conclusion:

Work distribution:
Dariga did: optimated realization of methods, putMark and view them, transcript, main idea of controller, Admin, OR Manager, Teacher, User, Executor, Employee, Mark, Order, Interfaces
Abylaikhan did: connections between groups, Student, DataBase classes implementation, the idea of making News block, Person, Course, Message, CourseFile

Taking everything into account, We can truly say that our Team really enjoyed the process of doing this group project together. Despite the fact that we lost our teammate Zhanym, because of lack of grades to pass the Final exam, we tried to stay close to each other and complete the started project. The project was successfully done technically and it was a really captivating process which helps us to use our knowledge in real-life project's practice. We open a lot of new methods during information exchange between each other and this knowledge is extremely helpful for each group member in the further IT-specialist branch. Also, this work enriches our responsibility sense, because everyone feels that his/her work is directly related to others' grades too. Therefore, everyone tries his/her best in terms of this project.