Report
for project

Discipline: Object-Oriented Programming

Project name: University system

Professor: Shamoi Pakita

Group: Sailau Danagul(leader),

Saliev Dias,

Zhumabayeva Nazym

KBTU, December, 2020

The goal of our project was to recreate a university system, for example: the WSP.We had to make actors, such as student, teacher; create methods for each of them with implementation, such as: putMark from the teacher - puts a certain grade to an individual student of some course; and build in this way the relationship between the different classes. We create different controllers for each user by abstract method userController and wrote its implementation in subclasses of User. Due to this, in login method we invoked user.userController, which is an example of polymorphism. Another example is updateInfo method, which is called in Admin to change specific fields of User.

```
public void putMark(Course course, Student student,  int att, double score) {
    Attestation a = null;
    for(HashMap.Entry<Course,Attestation> entry : student.viewTranscript().getJournal().entrySet()) {
        if (entry.getKey().equals(course))
            a = entry.getValue();
    }
    if (att == 1)
        a.addScoresToFirstAtt(score);
    if (att == 2)
        a.addScoresToSecondtAtt(score);
    if (att == 3)
        a.setFinalScore(score);}
```

Example for polymorphism in project:

```
public void updateUser(User user) {
    user.updateInfo();
}
```

One more example for methods:

```
 * Used to know whether the course finished or not. This method is used in the {@link registerForCourse}
public boolean isCourseFinished(Course course) {
    for(HashMap.Entry<Course, CourseStatus> entry: this.getCourses().entrySet()) {
        if (entry.getKey().equals(course.getPrerequisite()) && entry.getValue() == CourseStatus.FINISHED)
            return true;
    }
    return false;
}
 * Used to register a student for a particular course
public void registerForCourse(Course course) throws ExcessOfCreditsException{
    int numberOfCredits = 0;
    if (this.isCourseFinished(course)) {
        for(HashMap.Entry<Course, CourseStatus> entry1: this.getCourses().entrySet())
            numberOfCredits += entry1.getKey().getNumberOfCredits();
        if (numberOfCredits + course.getNumberOfCredits() <= limitOfCredits) {
            this.courses.put(course, CourseStatus.CURRENT);
            course.getStudentsOfCourse().add(this);
            Attestation newAtt = new Attestation();
            transcript.addToJournal(course, newAtt);
        } else
            throw new ExcessOfCreditsException("You have chosen more courses than you can."); }
    }
```

Also, we used pattern Singleton in Data class to create just one object of our class. We did this to address our created objects such as students, teachers, orders through one object data. However, we faced some problems during serialization, when we tried to invoke it with parameter data. But we finally solve this problem

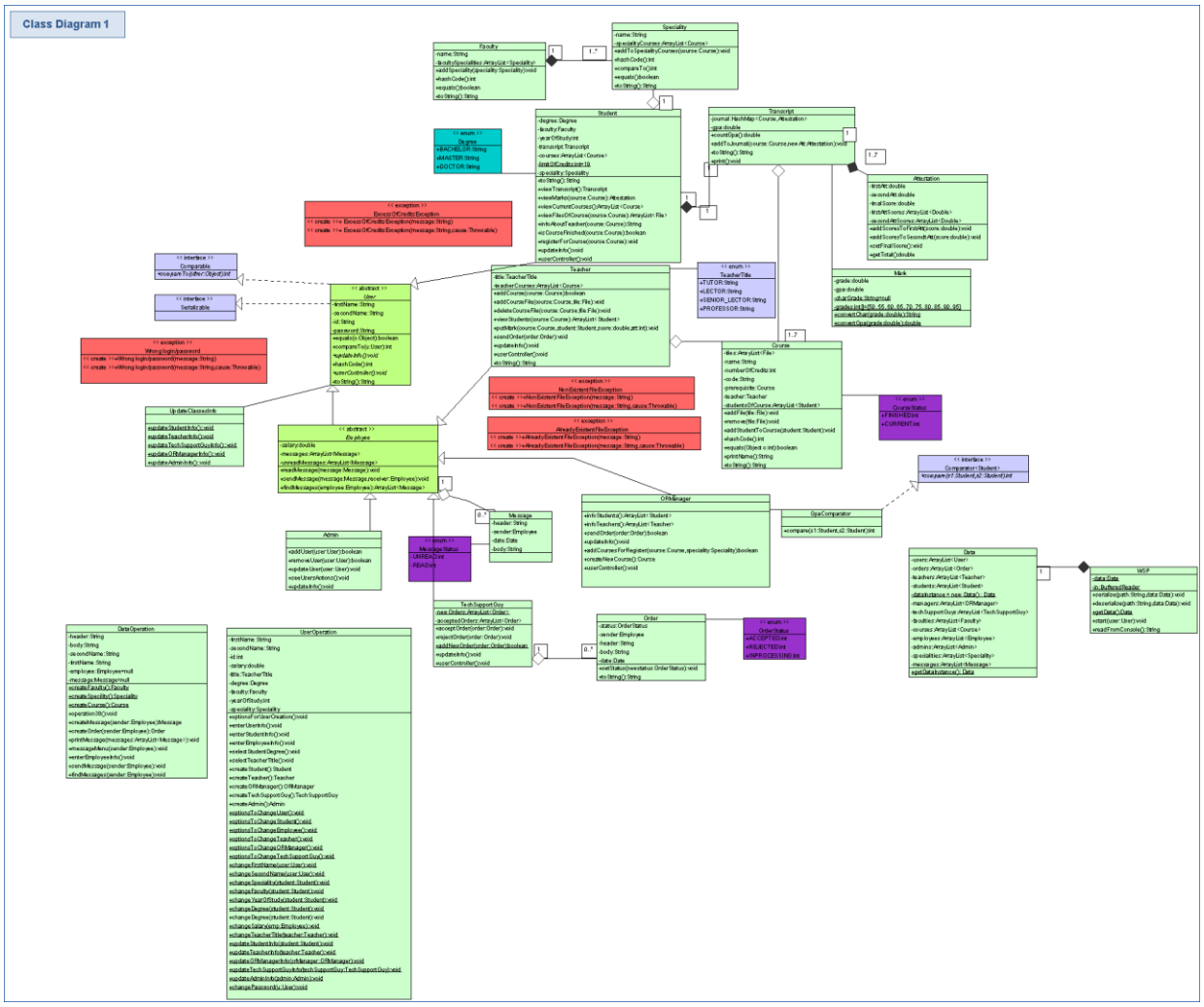
```
public static boolean serialize(Data data, String fileName) throws IOException {
    File file = new File(fileName);
    if (!file.exists())
        file.createNewFile();
    try(FileOutputStream fileStream = new FileOutputStream(fileName);
    ObjectOutputStream objectStream = new ObjectOutputStream(fileStream))
    {
        objectStream.writeObject(data);
    }catch(IOException e){
        System.out.println("Exception Occurred while Serializing");
        return false;
    }
    return true;
}
```

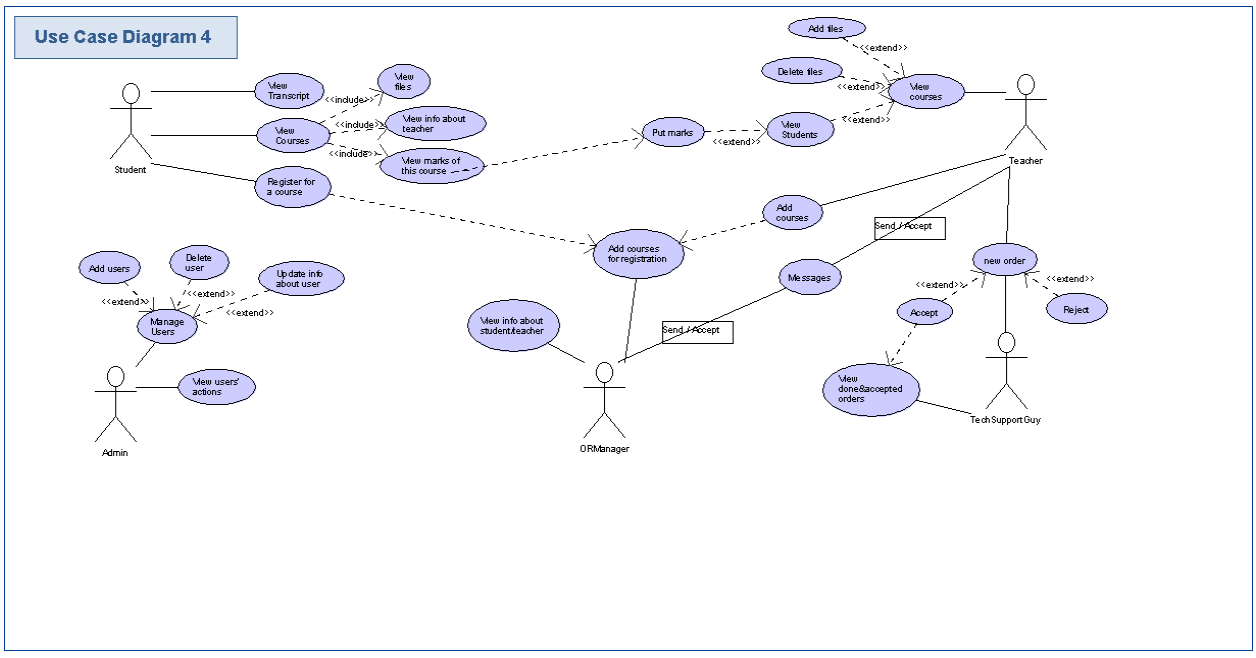
Moreover, we used Factory to create objects of serializable classes. Examples are in UserOperation and DataOperation classes.

```
public static Course createCourse() {
    System.out.println("Enter name of Course");
    String name = WSP.readFromConsole();
    System.out.println("Enter code of Course");
    String code = WSP.readFromConsole();
    System.out.println("Enter number of Credits");
    int numOfCredits = Integer.parseInt(WSP.readFromConsole());
    System.out.println("Does the course have prerequisite? \n 1. Enter 1 if YES \n 2. Enter 2 if NO");
    int option = Integer.parseInt(WSP.readFromConsole());
    Course course = null;
    if (option == 2)
        course = new Course(name, code,numOfCredits);
    if (option == 1) {
        String prereq = WSP.readFromConsole();
        for (Course c: WSP.getData().getCourses()) {
            if (c.getName().equals(prereq))
                course = new Course(name, code, numOfCredits, c);}
    }
    return course;
}
```

Additionally we used GpaComparator to sort list of students.

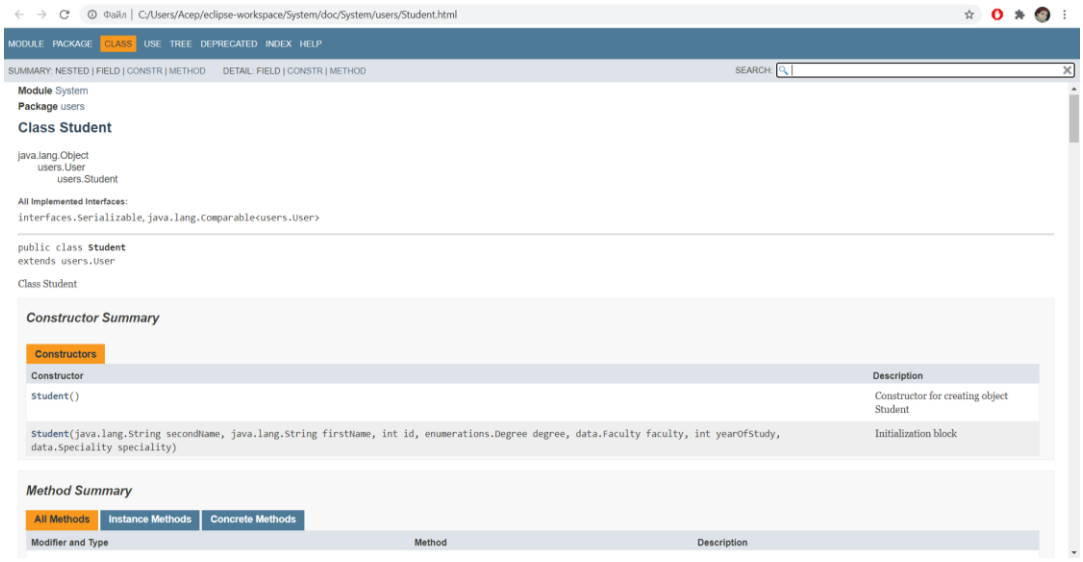
Class and use case diagram:





Documentation

Documentation for class Student



MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

Module System
Package users

Class Student

java.lang.Object
  users.User
    users.Student

All Implemented Interfaces:
interfaces.Serializable, java.lang.Comparable<users.User>

```
public class Student
extends users.User
```

Class Student

Constructor Summary

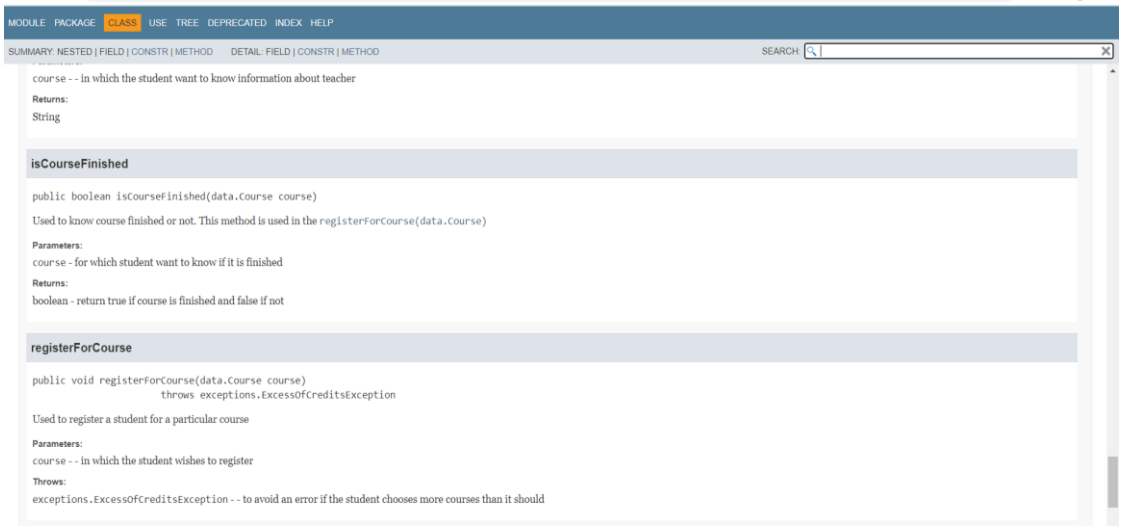
Constructors

Constructor	Description
Student()	Constructor for creating object Student
Student(java.lang.String secondName, java.lang.String firstName, int id, enumerations.Degree degree, data.Faculty faculty, int yearOfStudy, data.Speciality speciality)	Initialization block

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
java.util.HashMap<data.Course,enumerations.CourseStatus>	getCourses()	Returns courses of student with their statuses
enumerations.Degree	getDegree()	Returns degree of student
data.Faculty	getFaculty()	Returns faculty of students
data.Speciality	getSpeciality()	Returns specility of student
int	getYearOfStudy()	Returns year of study of student
java.lang.String	infoAboutTeacher(data.Course course)	Used to view information about teacher of a particular course
boolean	isCourseFinished(data.Course course)	Used to know course finished or not.
void	registerForCourse(data.Course course)	Used to register a student for a particular course
void	setDegree(enumerations.Degree degree)	Update the value of the field degree
void	setFaculty(data.Faculty faculty)	Update the value of the field faculty
void	setSpeciality(data.Speciality speciality)	Update the value of the field speciality
void	setYearOfStudy(int yearOfStudy)	Update the value of the field year of study
java.lang.String	toString()	Returns information about student
void	updateInfo()	Used to update information about student
void	userController()	Menu with different options for student
java.util.ArrayList<data.Course>	viewCurrentCourses()	Used to view all current courses of student
java.util.ArrayList<java.io.File>	viewFilesOfCourse(data.Course course)	Used to view the files of a particular course



MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH:

course - - in which the student want to know information about teacher

Returns:
String

isCourseFinished

```
public boolean isCourseFinished(data.Course course)
```

Used to know course finished or not. This method is used in the `registerForCourse(data.Course)`

Parameters:
course - for which student want to know if it is finished

Returns:
boolean - return true if course is finished and false if not

registerForCourse

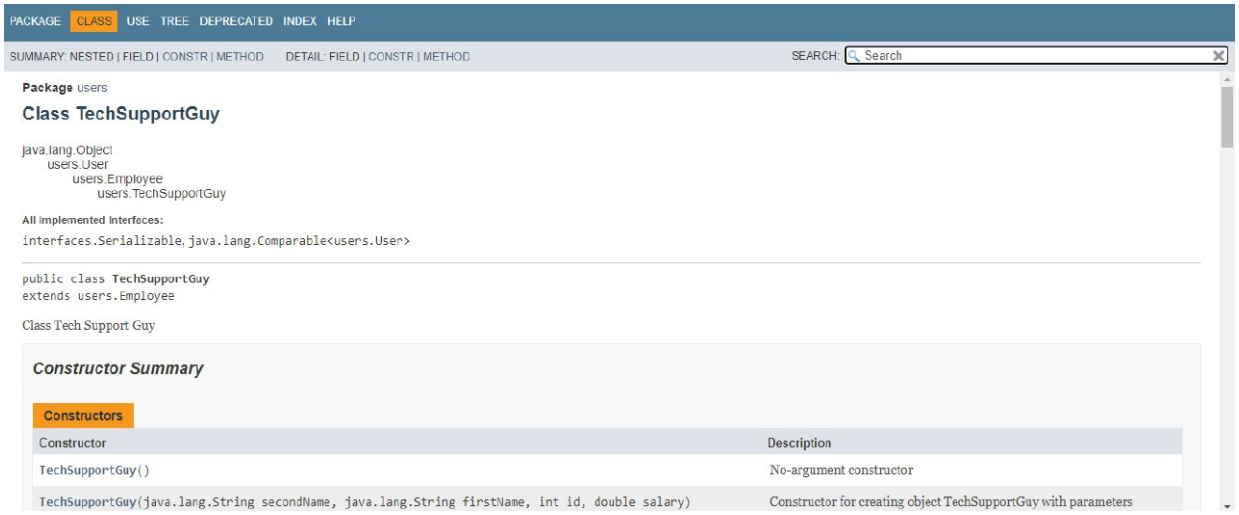
```
public void registerForCourse(data.Course course)
                       throws exceptions.ExcessOfCreditsException
```

Used to register a student for a particular course

Parameters:
course - - in which the student wishes to register

Throws:
exceptions.ExcessOfCreditsException - - to avoid an error if the student chooses more courses than it should

Documentation for class TechSupportGuy



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package users

Class TechSupportGuy

java.lang.Object
  users.User
    users.Employee
      users.TechSupportGuy

All implemented Interfaces:

`interfaces.Serializable, java.lang.Comparable<users.User>`

```
public class TechSupportGuy
extends users.Employee
```

Class Tech Support Guy

Constructor Summary

Constructors

Constructor	Description
`TechSupportGuy()`	No-argument constructor
`TechSupportGuy(java.lang.String secondName, java.lang.String firstName, int id, double salary)`	Constructor for creating object TechSupportGuy with parameters

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`void`	`acceptOrder (data.Order order)`	This method is used to add order to static ArrayList "accepted orders" and remove it from "new orders" static ArrayList.
`static boolean`	`addNewOrder (data.Order order)`	This method is used to add new order to static "newOrders" ArrayList
`static java.util.ArrayList<data.Order>`	`getAcceptedOrders()`	Returns ArrayList of accepted order
`static java.util.ArrayList<data.Order>`	`getNewOrders()`	Returns ArrayList of new orders
`void`	`rejectOrder (data.Order order)`	This method is used to remove order from "new orders" static ArrayList.
`void`	`updateInfo()`	This method is used to update particular fields of TechSupportGuy
`void`	`userController()`	Menu of Tech Support Guy with different options

Methods inherited from class users.Employee

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

acceptOrder

```
public void acceptOrder(data.Order order)
```

This method is used to add order to static ArrayList "accepted orders" and remove it from "new orders" static ArrayList. The status of order is changed to ACCEPT

Parameters:

`order` - the order that Tech Support Guy wants to accept;

rejectOrder

```
public void rejectOrder(data.Order order)
```

This method is used to remove order from "new orders" static ArrayList. The status of order is changed to REJECT

Parameters:

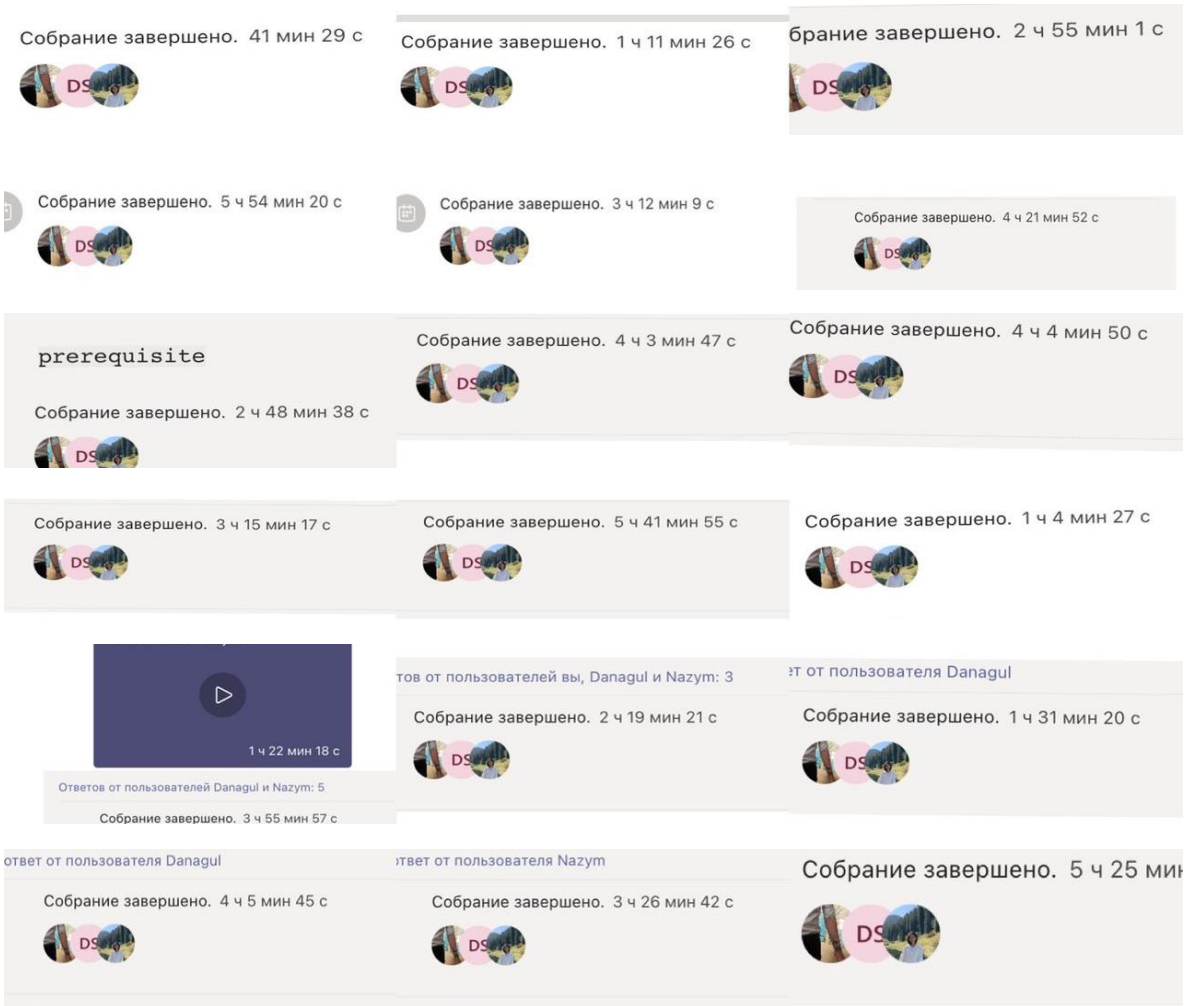
`order` - the order Tech Support Guy wants to reject;

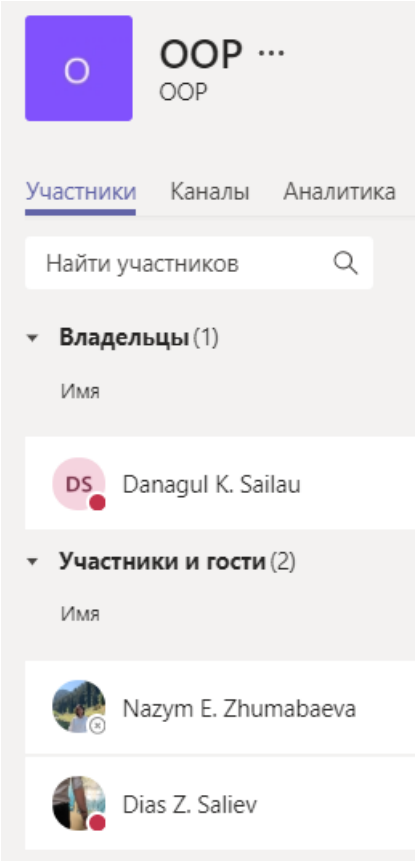
Project management

Teamwork

We are all from three different cities, so we had to arrange meetings in Microsoft Teams.

We held a total of 80 hours of meetings:





So we also delegated the work of.

Danagul create: Teacher, Course, Data, DataOperation, WSP, TeacherController

Nazym create: Student, Attestation, Mark, Transcript, Faculty, Speciality, Student and Admin controllers

Dias create: ORManager, TechSupportGuy, Employee, Admin and controllers for both, Order

Other classes, interfaces, exceptions we created together