KAZAKH BRITISH TECHNICAL UNIVERSITY

Report on the OOP project

Faculty of Information Technology

Discipline: Object-Oriented programming

Project name: University System

TEAM: «BURNT OUT»

TEAM MEMBERS:

ERDAUIT TOREKHAN

ALTAIR BISENBIEV

LARIONOVA ANASTASSIYA

ARUZHAN MEIIRKHAN

TEACHER:

SHAMOI PAKITA

Content:

Introduction

The main task is to create a university system. There are users in the system, everyone has their own roles and certain capabilities. It was also necessary to initially create a diagram using the Java programming language and the knowledge we gained in the object-oriented programming course to create a working system in the console. It was divided into 3 parts: UML diagram, full classes, and tester class. With this project, you can manage multiple university processes from different accounts, saving the process using serialization

Briefly about the project

University system can do:

- All users can log in, log out and go to the main menu.
- In the system, everyone is divided into roles with their own capabilities.
- **User**: When logging in, everyone sees the news and go back to the menu.
- **Admin**: creates and deletes users, changes password, faculty and login, sees all users.
- **Employee**: stores all data related to employees - teacher, manager and librarian, as well as write and read their messages.
- **Teacher**: set grades, view all students, add and view files, see your courses and rating.
- **Manager**: create courses, add teachers to the course, see all courses, requests, news, monitor news, registration requests, sort teachers and students.
- **Librarian**: add and delete books, see and accept requests, see a list of books, news.
- **Student**: see information about himself, register for courses and drop, see all possible courses, choose a time for lessons, log in to organizations, rate the teacher, view the certification, transcript, organizations, schedules, teacher evaluations, files and a list of books.

The main part

Diagrams

The diagrams were made in order to track what needs to be done and what will be associated with what, inherit, etc. Every time we changed something in the code, we always changed the diagrams and tried to monitor the correctness of execution.

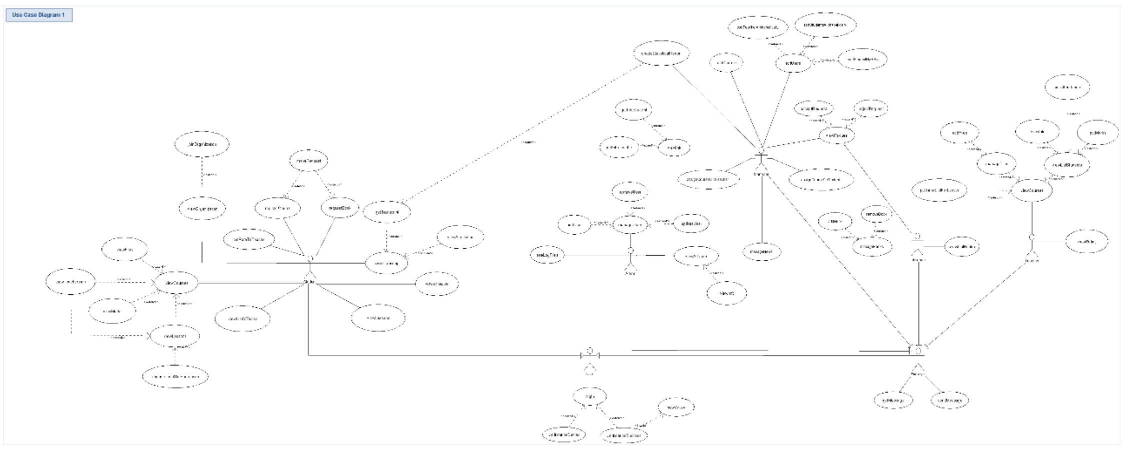
Use Case

In software development and system design, this is a description of the behavior of a system when it interacts with someone from the external environment. We have indicated all our users and what basic methods they will perform.

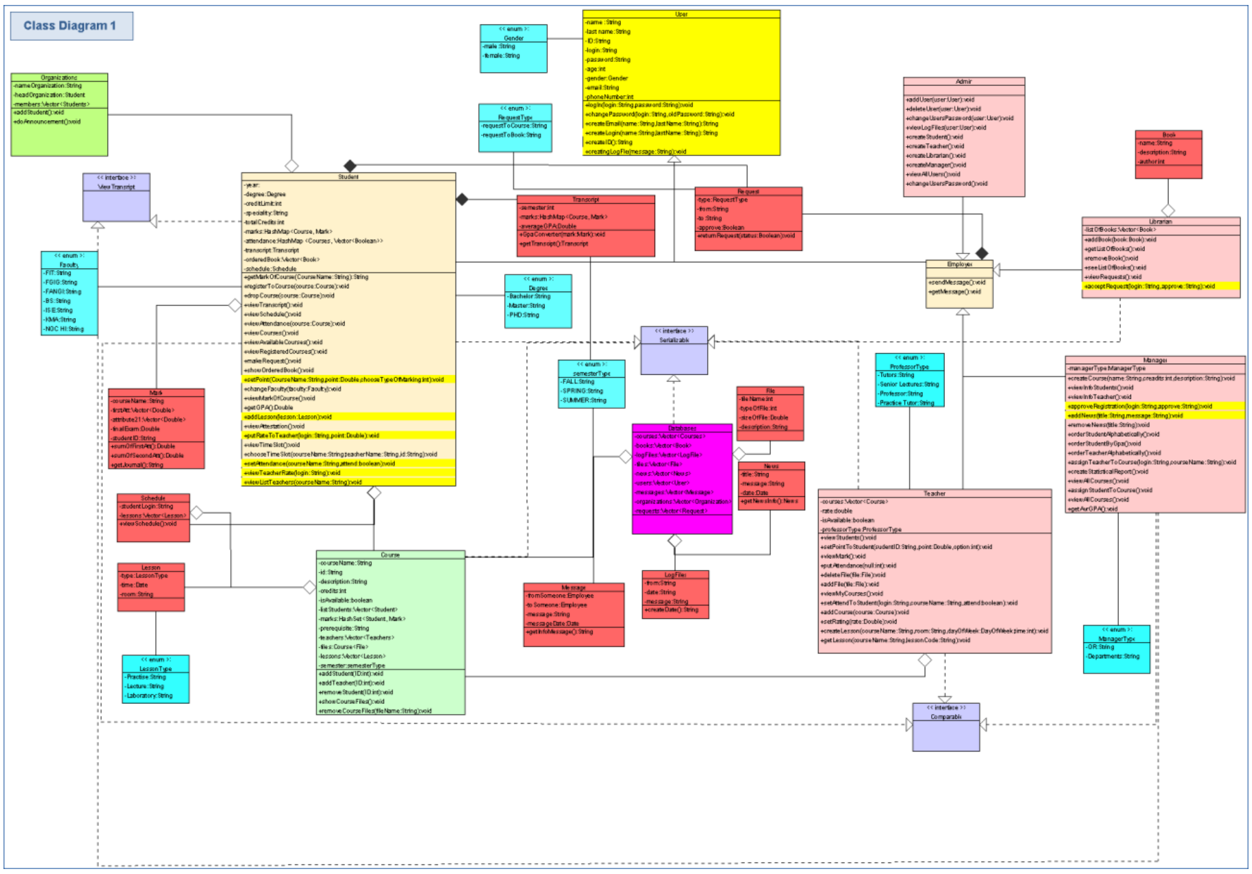
CLASS Diagrams

A graphical description language for object modeling in the field of software development, for modeling business processes, system design and displaying organizational structures. It was used to visually understand the project, where enams, connections are needed and which variables will need to be created and used.

Use Case



CLASS Diagrams **(perhaps a slightly outdated version, since we will implement the new one immediately in the document and the changes were made to the computer before the deadline)**



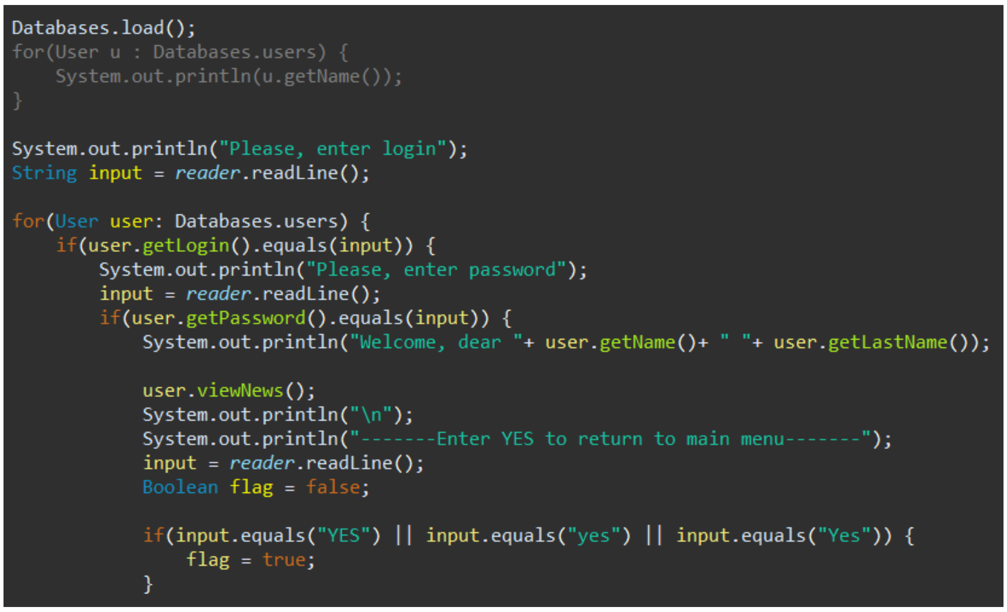
Code

The code was written by a team, everyone had their own tasks. In this report we will show the main points and features of our project.

Classes

University

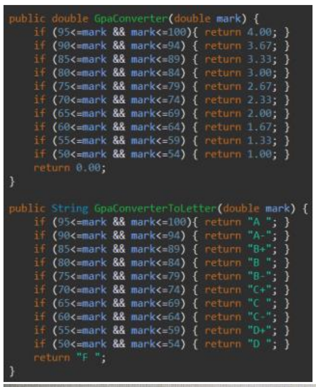
The university is a test class in which all the menus of each user are called, each user has his own personal menu where you can choose the necessary actions. This class uses a Buffered Reader. And also loads a database file where the data for the project is already registered. Through the console, we enter the login and password, after which various actions are available.



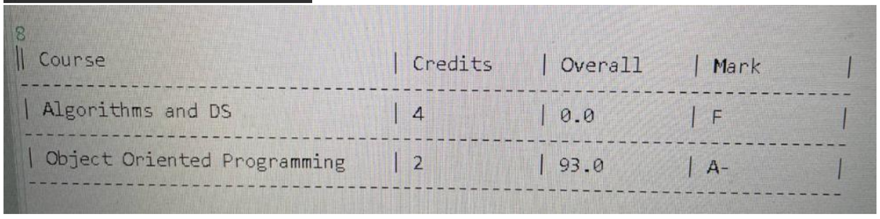
```java
Databases.load();
for(User u : Databases.users) {
    System.out.println(u.getName());
}

System.out.println("Please, enter login");
String input = reader.readLine();

for(User user: Databases.users) {
    if(user.getLogin().equals(input)) {
        System.out.println("Please, enter password");
        input = reader.readLine();
        if(user.getPassword().equals(input)) {
            System.out.println("Welcome, dear "+ user.getName()+ " "+ user.getLastName());

            user.viewNews();
            System.out.println("\n");
            System.out.println("-------Enter YES to return to main menu-------");
            input = reader.readLine();
            Boolean flag = false;

            if(input.equals("YES") || input.equals("yes") || input.equals("Yes")) {
                flag = true;
            }
```

Transcript

The transcript causes estimates and works as a converter to the letter version, as well as the GPA is calculated. Using the "View Transcript" method, a visually organized console object appears, where "Course", "Credits", "Overall", "Mark" are registered. Everything is organized for a beautiful display of results.



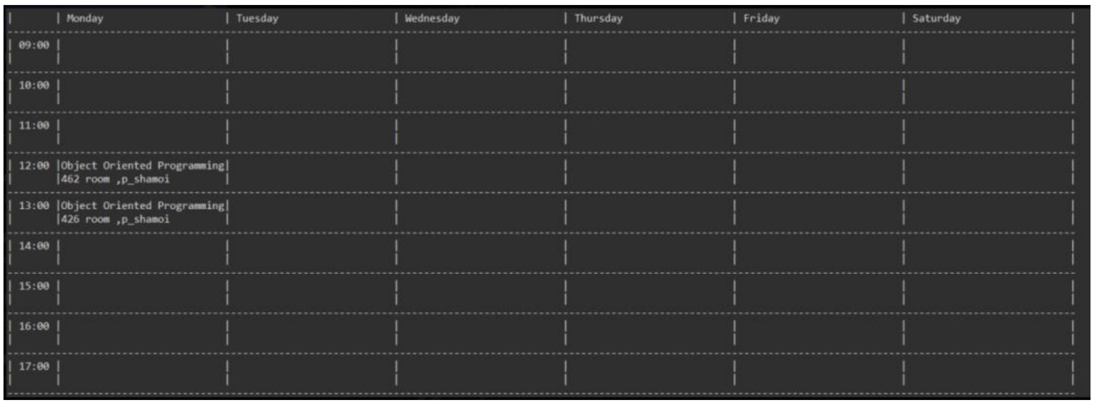
```java
public double GpaConverter(double mark) {
    if (95<=mark && mark<=100){ return 4.00; }
    if (90<=mark && mark<=94) { return 3.67; }
    if (85<=mark && mark<=89) { return 3.33; }
    if (80<=mark && mark<=84) { return 3.00; }
    if (75<=mark && mark<=79) { return 2.67; }
    if (70<=mark && mark<=74) { return 2.33; }
    if (65<=mark && mark<=69) { return 2.00; }
    if (60<=mark && mark<=64) { return 1.67; }
    if (55<=mark && mark<=59) { return 1.33; }
    if (50<=mark && mark<=54) { return 1.00; }
    return 0.00;
}

public String GpaConverterToLetter(double mark) {
    if (95<=mark && mark<=100){ return "A "; }
    if (90<=mark && mark<=94) { return "A-"; }
    if (85<=mark && mark<=89) { return "B+"; }
    if (80<=mark && mark<=84) { return "B "; }
    if (75<=mark && mark<=79) { return "B-"; }
    if (70<=mark && mark<=74) { return "C+"; }
    if (65<=mark && mark<=69) { return "C "; }
    if (60<=mark && mark<=64) { return "C-"; }
    if (55<=mark && mark<=59) { return "D+"; }
    if (50<=mark && mark<=54) { return "D "; }
    return "F ";
}
```



```
8
|| Course                          | Credits    | Overall     | Mark          |
---------------------------------------------------------------------------------
| Algorithms and DS                | 4          | 0.0         | F             |
---------------------------------------------------------------------------------
| Object Oriented Programming      | 2          | 93.0        | A-            |
---------------------------------------------------------------------------------
```

Schedule

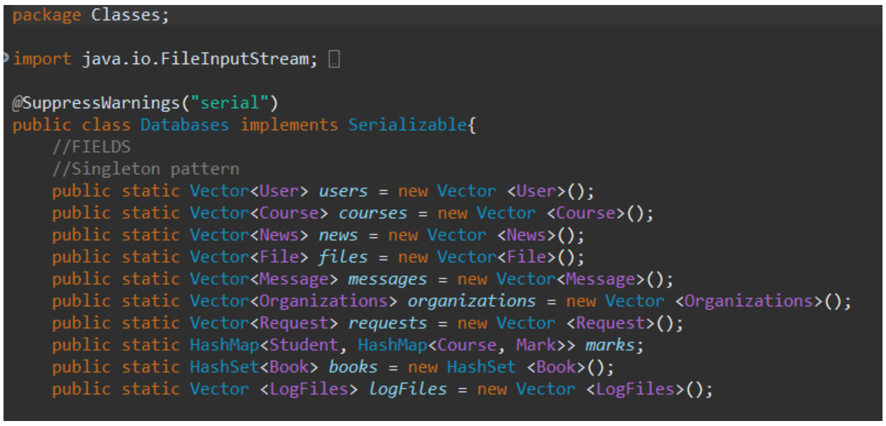
Using a special output format, we have made a schedule in the project. The subject and its teacher will be added to it. It is also divided by days of the week and time, which makes it easier to visually understand the schedule.



```
|       | Monday                    | Tuesday                   | Wednesday                 | Thursday                  | Friday                    | Saturday                  |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 09:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 10:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 11:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 12:00 |Object Oriented Programming|                           |                           |                           |                           |                           |
|       |462 room ,p_shamoi         |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 13:00 |Object Oriented Programming|                           |                           |                           |                           |                           |
|       |426 room ,p_shamoi         |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 14:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 15:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 16:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| 17:00 |                           |                           |                           |                           |                           |                           |
|       |                           |                           |                           |                           |                           |                           |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Databases

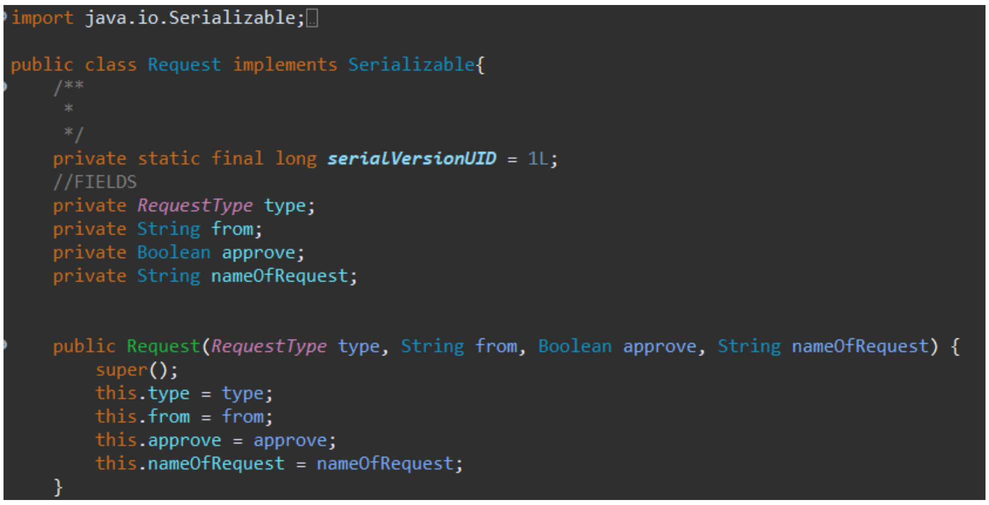
We have all the vectors and data stored in the database. Using files and serialization, everything is saved there. With methods that can be called and saved and loaded.



```java
package Classes;

import java.io.FileInputStream;

@SuppressWarnings("serial")
public class Databases implements Serializable{
    //FIELDS
    //Singleton pattern
    public static Vector<User> users = new Vector <User>();
    public static Vector<Course> courses = new Vector <Course>();
    public static Vector<News> news = new Vector <News>();
    public static Vector<File> files = new Vector<File>();
    public static Vector<Message> messages = new Vector<Message>();
    public static Vector<Organizations> organizations = new Vector <Organizations>();
    public static Vector<Request> requests = new Vector <Request>();
    public static HashMap<Student, HashMap<Course, Mark>> marks;
    public static HashSet<Book> books = new HashSet <Book>();
    public static Vector <LogFiles> logFiles = new Vector <LogFiles>();
```

Request

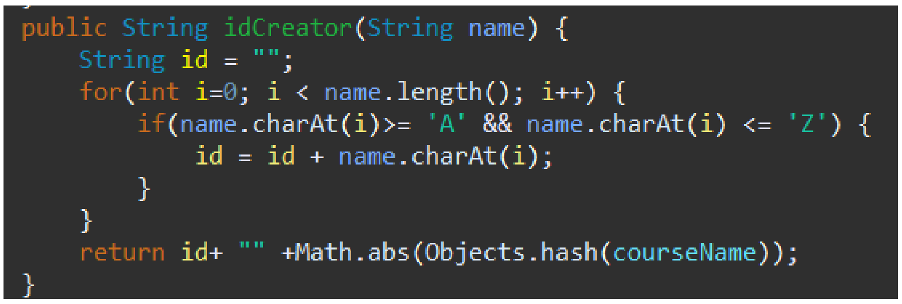
The query is executed simply, it has getters and setters. The execution function outputs the request and marks it as executed.



```java
import java.io.Serializable;

public class Request implements Serializable{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    //FIELDS
    private RequestType type;
    private String from;
    private Boolean approve;
    private String nameOfRequest;


    public Request(RequestType type, String from, Boolean approve, String nameOfRequest) {
        super();
        this.type = type;
        this.from = from;
        this.approve = approve;
        this.nameOfRequest = nameOfRequest;
    }
```

Course

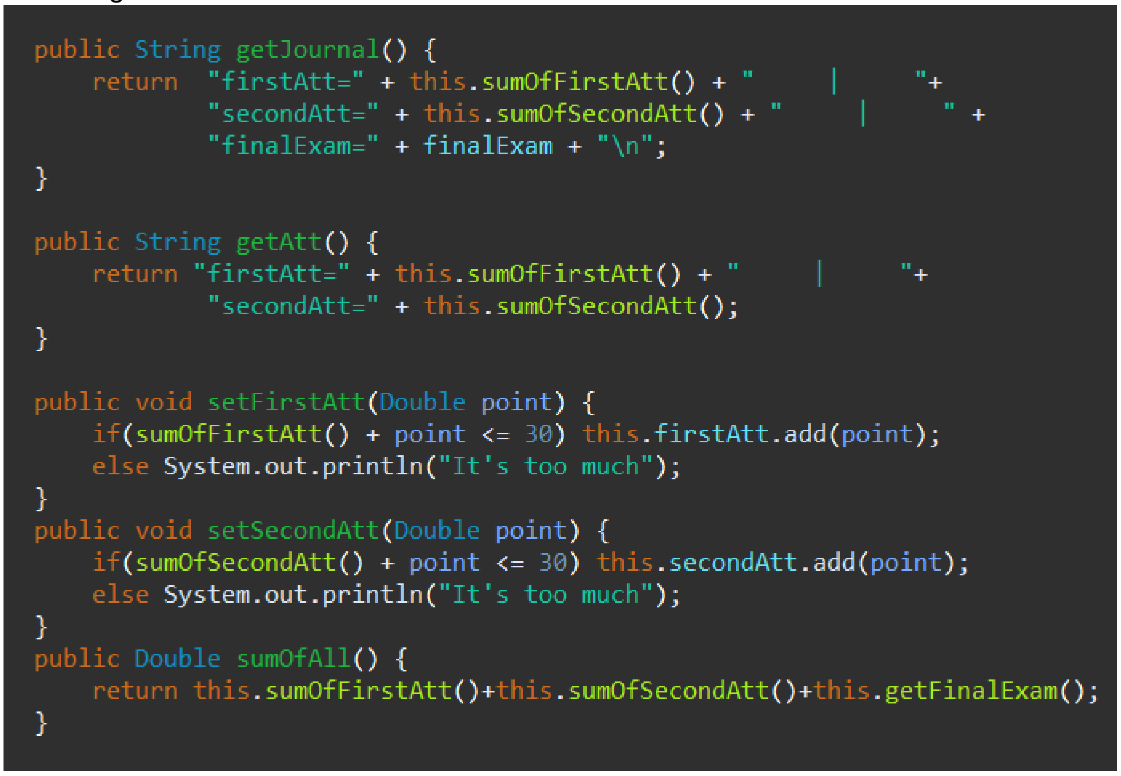
In the Course class, you can view all the information related to a particular course, as well as add a new lesson to the course. The ID creator will be checked and created automatically.



```java
public String idCreator(String name) {
    String id = "";
    for(int i=0; i < name.length(); i++) {
        if(name.charAt(i)>= 'A' && name.charAt(i) <= 'Z') {
            id = id + name.charAt(i);
        }
    }
    return id+ "" +Math.abs(Objects.hash(courseName));
}
```

Mark

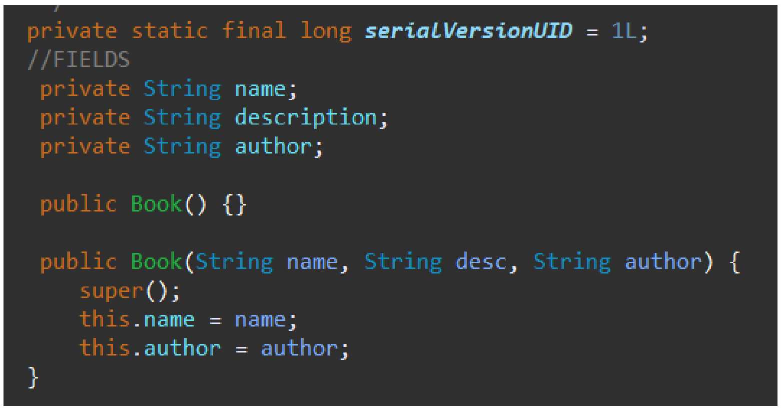
Evaluations are the most important part of the project. We can give grades to a certain student. Also by choosing which specific certification. The total amount will be calculated based on all the data. And it is also displayed in the form of a small log of student grades.



```java
public String getJournal() {
    return  "firstAtt=" + this.sumOfFirstAtt() + "     |     "+
            "secondAtt=" + this.sumOfSecondAtt() + "     |     " +
            "finalExam=" + finalExam + "\n";
}

public String getAtt() {
    return "firstAtt=" + this.sumOfFirstAtt() + "     |     "+
            "secondAtt=" + this.sumOfSecondAtt();
}

public void setFirstAtt(Double point) {
    if(sumOfFirstAtt() + point <= 30) this.firstAtt.add(point);
    else System.out.println("It's too much");
}
public void setSecondAtt(Double point) {
    if(sumOfSecondAtt() + point <= 30) this.secondAtt.add(point);
    else System.out.println("It's too much");
}
public Double sumOfAll() {
    return this.sumOfFirstAtt()+this.sumOfSecondAtt()+this.getFinalExam();
}
```

Book

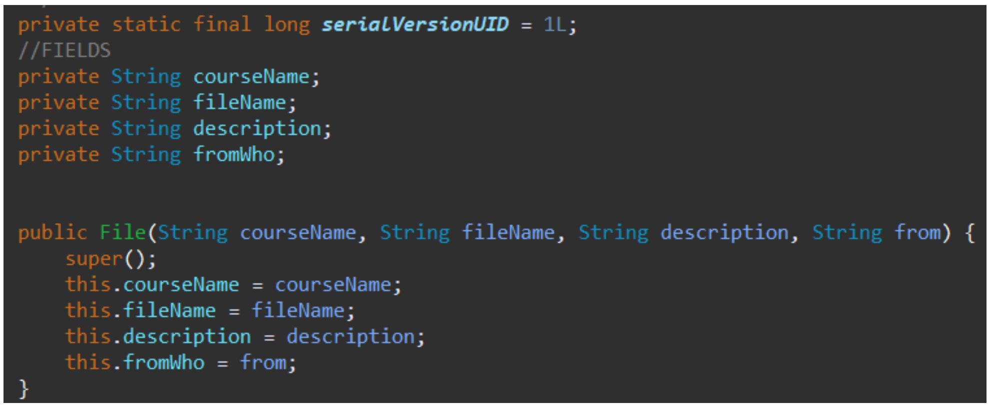
Books is a common class that stores data about the author, title and description of the book, used for a list of books for the librarian.



```java
private static final long serialVersionUID = 1L;
//FIELDS
 private String name;
 private String description;
 private String author;

 public Book() {}

 public Book(String name, String desc, String author) {
     super();
     this.name = name;
     this.author = author;
}
```

File

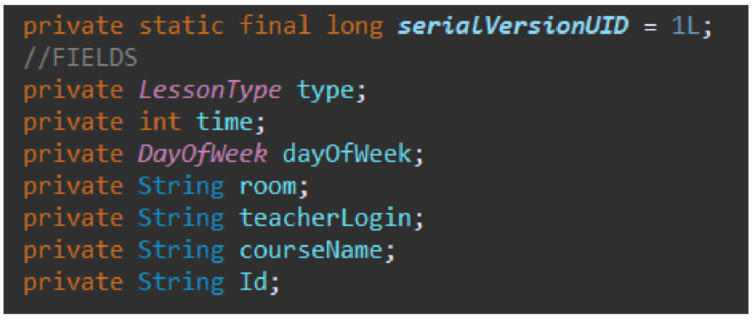
A file is a regular class that stores data about the course name, description and who, used for a list of files for the course and teachers.



```java
private static final long serialVersionUID = 1L;
//FIELDS
private String courseName;
private String fileName;
private String description;
private String fromWho;


public File(String courseName, String fileName, String description, String from) {
    super();
    this.courseName = courseName;
    this.fileName = fileName;
    this.description = description;
    this.fromWho = from;
}
```

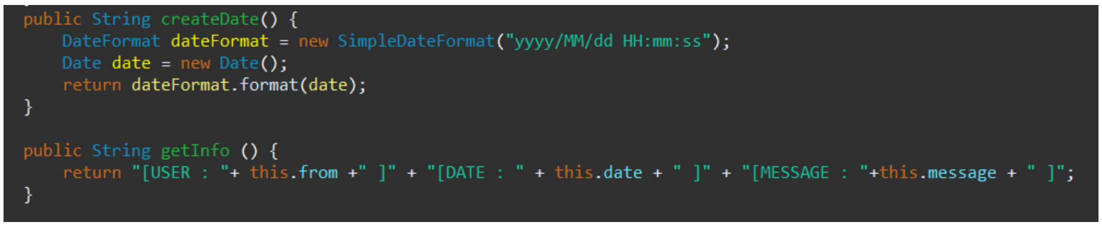
Lesson

Lessons are one of the main classes. Very important for the schedule. It stores data and transmits for the schedule, is divided by days of the week, and outputs general information at once.



```java
private static final long serialVersionUID = 1L;
//FIELDS
private LessonType type;
private int time;
private DayOfWeek dayOfWeek;
private String room;
private String teacherLogin;
private String courseName;
private String Id;
```

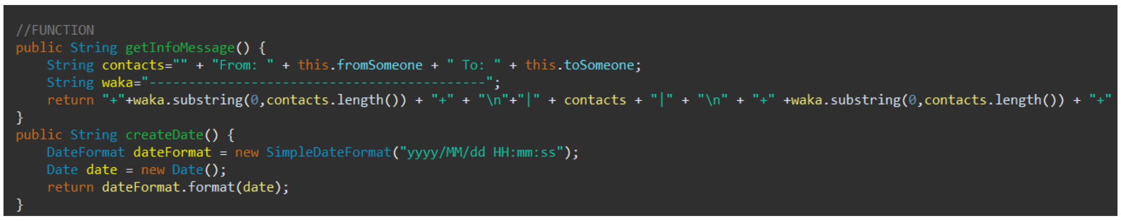
LogFiles



```java
public String createDate() {
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    return dateFormat.format(date);
}

public String getInfo () {
    return "[USER : "+ this.from +" ]" + "[DATE : " + this.date + " ]" + "[MESSAGE : "+this.message + " ]";
}
```

Message

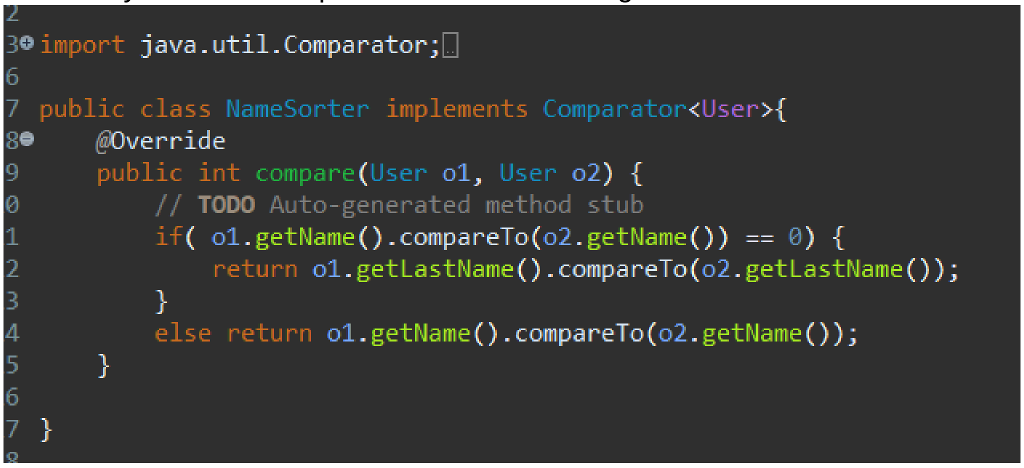
All employees can send and read messages. You can write a message and another user will be able to see the data written by you, as well as the date and from whom the letter is.



```java
//FUNCTION
public String getInfoMessage() {
    String contacts="" + "From: " + this.fromSomeone + " To: " + this.toSomeone;
    String waka="----------------------------------------";
    return "+"+waka.substring(0,contacts.length()) + "+" + "\n"+"|" + contacts + "|" + "\n" + "+" +waka.substring(0,contacts.length()) + "+"
}
public String createDate() {
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    return dateFormat.format(date);
}
```

NameSorter

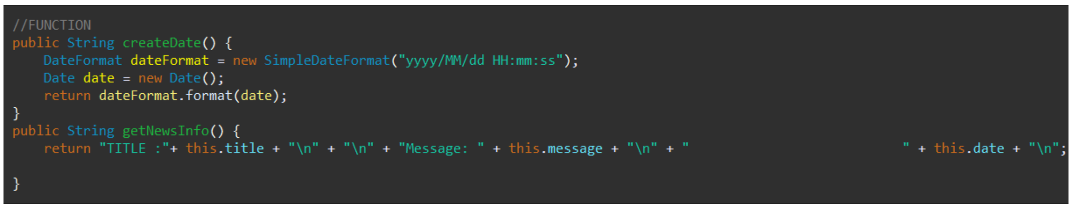
This is a comparator of users in order to be able to sort the names when output, the necessary element is implemented in the manager.



```java
import java.util.Comparator;

public class NameSorter implements Comparator<User>{
    @Override
    public int compare(User o1, User o2) {
        // TODO Auto-generated method stub
        if( o1.getName().compareTo(o2.getName()) == 0) {
            return o1.getLastName().compareTo(o2.getLastName());
        }
        else return o1.getName().compareTo(o2.getName());
    }

}
```

News

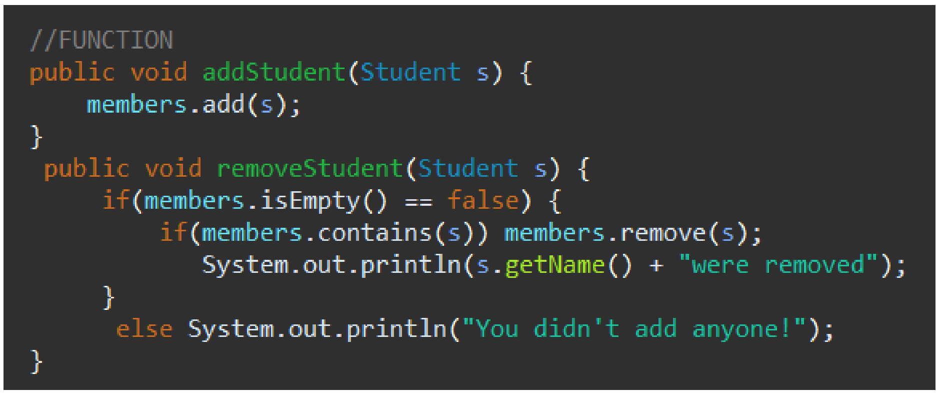
The news appears automatically when you log in. After that, the user can go to the main menu. The news stores information that the manager can control.



```java
//FUNCTION
public String createDate() {
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    return dateFormat.format(date);
}
public String getNewsInfo() {
    return "TITLE :"+ this.title + "\n" + "\n" + "Message: " + this.message + "\n" + "                                        " + this.date + "\n";

}
```

Organizations

There is a list of organizations where the names are stored. A student can join an organization or be removed from it.



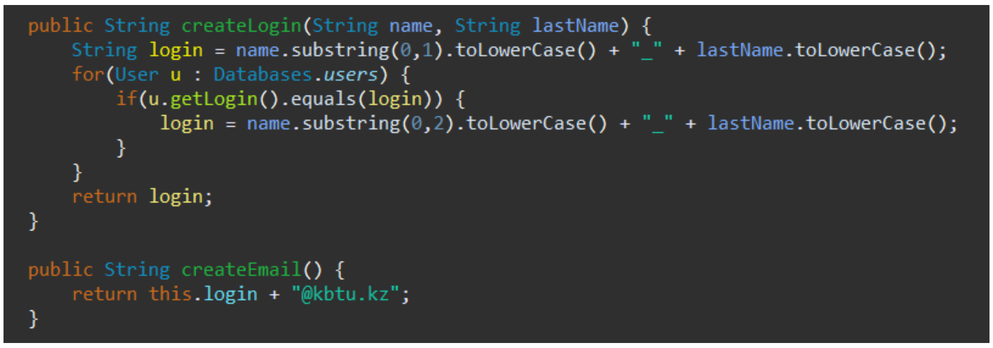
```java
//FUNCTION
public void addStudent(Student s) {
    members.add(s);
}
 public void removeStudent(Student s) {
     if(members.isEmpty() == false) {
         if(members.contains(s)) members.remove(s);
            System.out.println(s.getName() + "were removed");
     }
      else System.out.println("You didn't add anyone!");
}
```

People

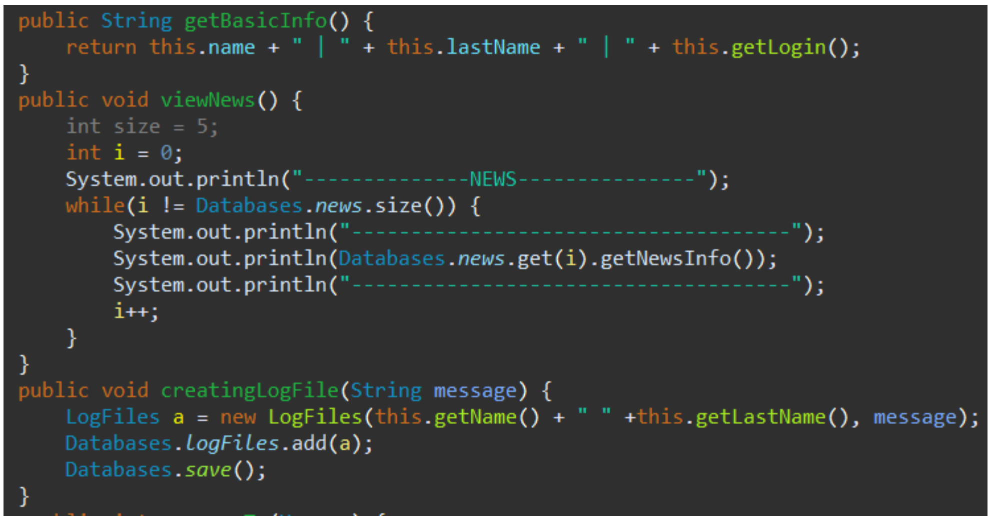
User

Every user sees the news and stores basic information about passwords, usernames, names, etc.

Feature: the method that create ID, create email, create login is enough to know the first name, the last name, it will generate this data itself. There is a method in the user that does not allow you to create duplicate logins. For example, if there are 2 Torehan's Erdaout in the system, then the first one will be with the login e_torekhan and the second one er_torekhan.



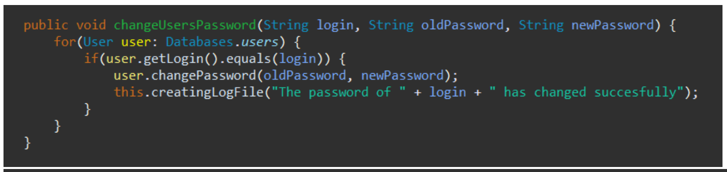
```java
public String createLogin(String name, String lastName) {
    String login = name.substring(0,1).toLowerCase() + "_" + lastName.toLowerCase();
    for(User u : Databases.users) {
        if(u.getLogin().equals(login)) {
            login = name.substring(0,2).toLowerCase() + "_" + lastName.toLowerCase();
        }
    }
    return login;
}

public String createEmail() {
    return this.login + "@kbtu.kz";
}
```



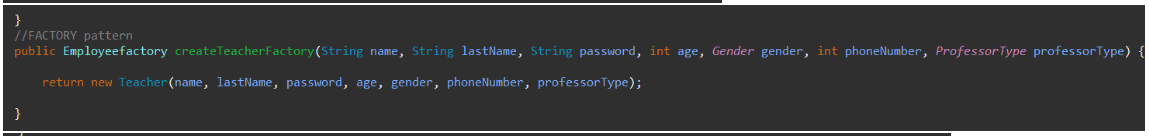
```java
public String getBasicInfo() {
    return this.name + " | " + this.lastName + " | " + this.getLogin();
}
public void viewNews() {
    int size = 5;
    int i = 0;
    System.out.println("-------------NEWS--------------");
    while(i != Databases.news.size()) {
        System.out.println("------------------------------------");
        System.out.println(Databases.news.get(i).getNewsInfo());
        System.out.println("------------------------------------");
        i++;
    }
}
public void creatingLogFile(String message) {
    LogFiles a = new LogFiles(this.getName() + " " +this.getLastName(), message);
    Databases.logFiles.add(a);
    Databases.save();
}
```

Admin

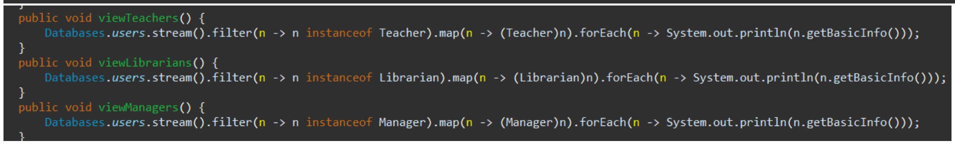
The admin has a lot of power, in terms of can add, delete all users, be it a student or a teacher. He sees all the lists of all users. Can change data and has a pattern, streams.



```java
public void changeUsersPassword(String login, String oldPassword, String newPassword) {
    for(User user: Databases.users) {
        if(user.getLogin().equals(login)) {
            user.changePassword(oldPassword, newPassword);
            this.creatingLogFile("The password of " + login + " has changed succesfully");
        }
    }
}
```



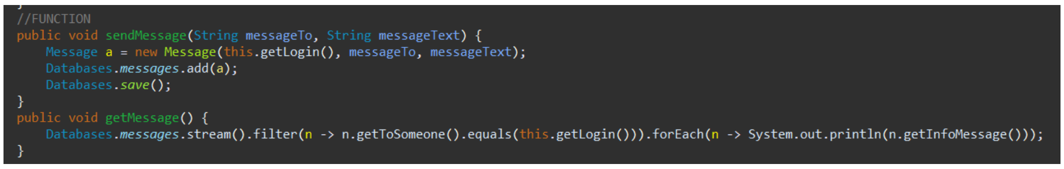
```java
}
//FACTORY pattern
public Employeefactory createTeacherFactory(String name, String lastName, String password, int age, Gender gender, int phoneNumber, ProfessorType professorType) {

    return new Teacher(name, lastName, password, age, gender, phoneNumber, professorType);

}
```



```java
public void viewTeachers() {
    Databases.users.stream().filter(n -> n instanceof Teacher).map(n -> (Teacher)n).forEach(n -> System.out.println(n.getBasicInfo()));
}
public void viewLibrarians() {
    Databases.users.stream().filter(n -> n instanceof Librarian).map(n -> (Librarian)n).forEach(n -> System.out.println(n.getBasicInfo()));
}
public void viewManagers() {
    Databases.users.stream().filter(n -> n instanceof Manager).map(n -> (Manager)n).forEach(n -> System.out.println(n.getBasicInfo()));
}
```

Employee

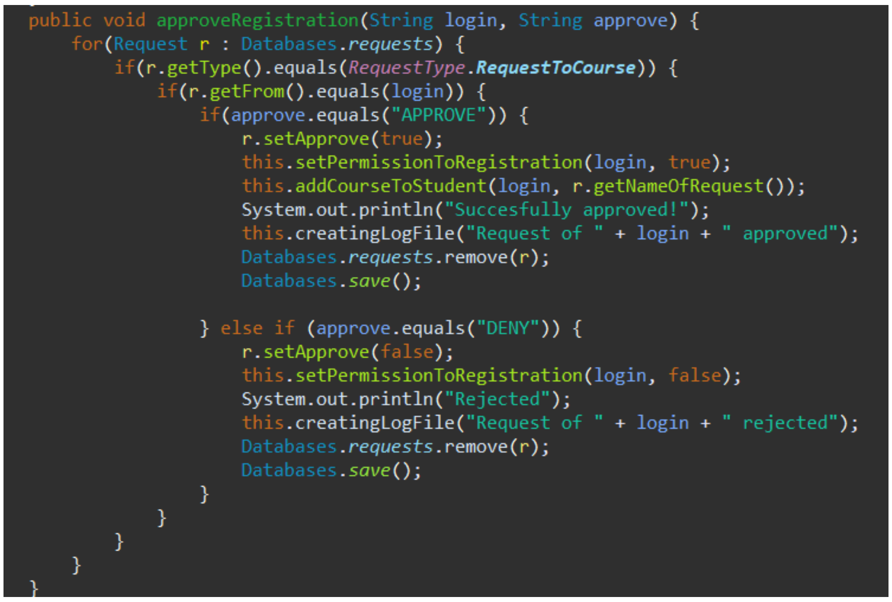
All employees inherit from the user, and the teacher, manager and librarian already inherits the employee. Everyone can send messages as well as read them.



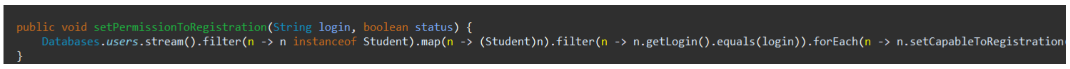
```java
//FUNCTION
public void sendMessage(String messageTo, String messageText) {
    Message a = new Message(this.getLogin(), messageTo, messageText);
    Databases.messages.add(a);
    Databases.save();
}
public void getMessage() {
    Databases.messages.stream().filter(n -> n.getToSomeone().equals(this.getLogin())).forEach(n -> System.out.println(n.getInfoMessage()));
}
```

Manager

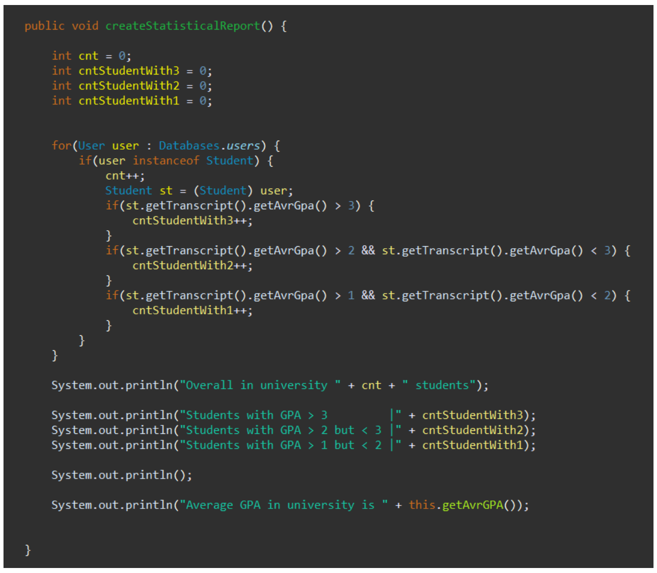
The manager also has a lot of power in this system. He can create courses, see requests, and most importantly register students for disciplines based on these requests. See news, delete news, and add students to courses. To see information about students and teachers, it will sort alphabetically or students by GPA. Can add a teacher to a course, add a lesson to a student. And among the main things, also output the report and the average value of the GPA in it, while additionally seeing the transcript.



```java
public void approveRegistration(String login, String approve) {
    for(Request r : Databases.requests) {
        if(r.getType().equals(RequestType.RequestToCourse)) {
            if(r.getFrom().equals(login)) {
                if(approve.equals("APPROVE")) {
                    r.setApprove(true);
                    this.setPermissionToRegistration(login, true);
                    this.addCourseToStudent(login, r.getNameOfRequest());
                    System.out.println("Succesfully approved!");
                    this.creatingLogFile("Request of " + login + " approved");
                    Databases.requests.remove(r);
                    Databases.save();

                } else if (approve.equals("DENY")) {
                    r.setApprove(false);
                    this.setPermissionToRegistration(login, false);
                    System.out.println("Rejected");
                    this.creatingLogFile("Request of " + login + " rejected");
                    Databases.requests.remove(r);
                    Databases.save();
                }
            }
        }
    }
}
```



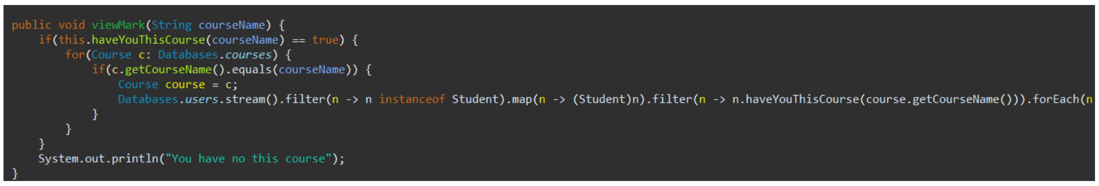
```java
public void setPermissionToRegistration(String login, boolean status) {
    Databases.users.stream().filter(n -> n instanceof Student).map(n -> (Student)n).filter(n -> n.getLogin().equals(login)).forEach(n -> n.setCapableToRegistration
}
```



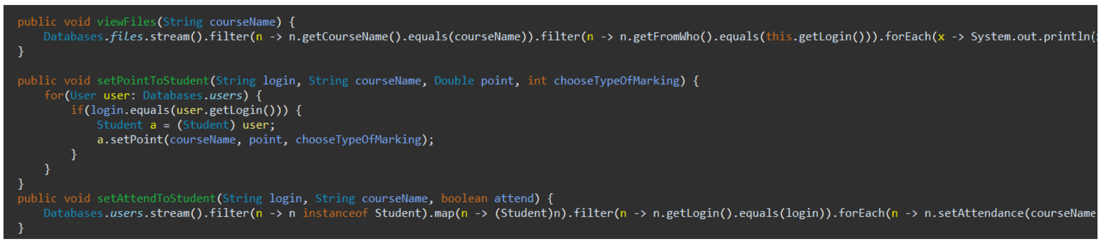
```java
public void createStatisticalReport() {

    int cnt = 0;
    int cntStudentWith3 = 0;
    int cntStudentWith2 = 0;
    int cntStudentWith1 = 0;


    for(User user : Databases.users) {
        if(user instanceof Student) {
            cnt++;
            Student st = (Student) user;
            if(st.getTranscript().getAvrGpa() > 3) {
                cntStudentWith3++;
            }
            if(st.getTranscript().getAvrGpa() > 2 && st.getTranscript().getAvrGpa() < 3) {
                cntStudentWith2++;
            }
            if(st.getTranscript().getAvrGpa() > 1 && st.getTranscript().getAvrGpa() < 2) {
                cntStudentWith1++;
            }
        }
    }

    System.out.println("Overall in university " + cnt + " students");

    System.out.println("Students with GPA > 3         |" + cntStudentWith3);
    System.out.println("Students with GPA > 2 but < 3 |" + cntStudentWith2);
    System.out.println("Students with GPA > 1 but < 2 |" + cntStudentWith1);

    System.out.println();

    System.out.println("Average GPA in university is " + this.getAvrGPA());


}
```

Teacher

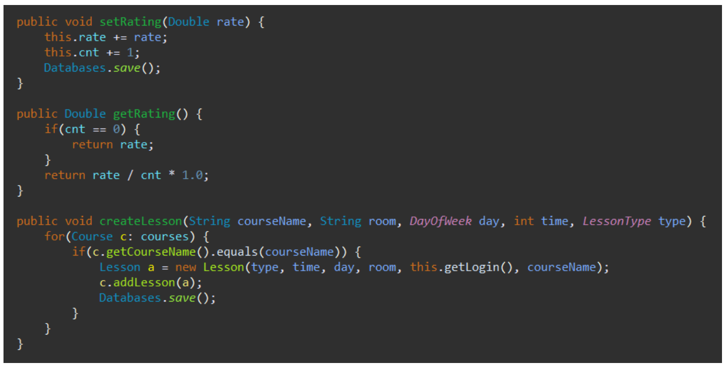
A teacher among the basic required classes. He can watch all his students and give grades to the students of this course. Also has access to files, can delete them, add them and see everything. A teacher can have several courses, he can view all of them in a separate window. Can mark a student, an attachment, Add a course, create a lesson, get lesson data.



```java
public void viewMark(String courseName) {
    if(this.haveYouThisCourse(courseName) == true) {
        for(Course c: Databases.courses) {
            if(c.getCourseName().equals(courseName)) {
                Course course = c;
                Databases.users.stream().filter(n -> n instanceof Student).map(n -> (Student)n).filter(n -> n.haveYouThisCourse(course.getCourseName())).forEach(n
            }
        }
    }
    System.out.println("You have no this course");
}
```



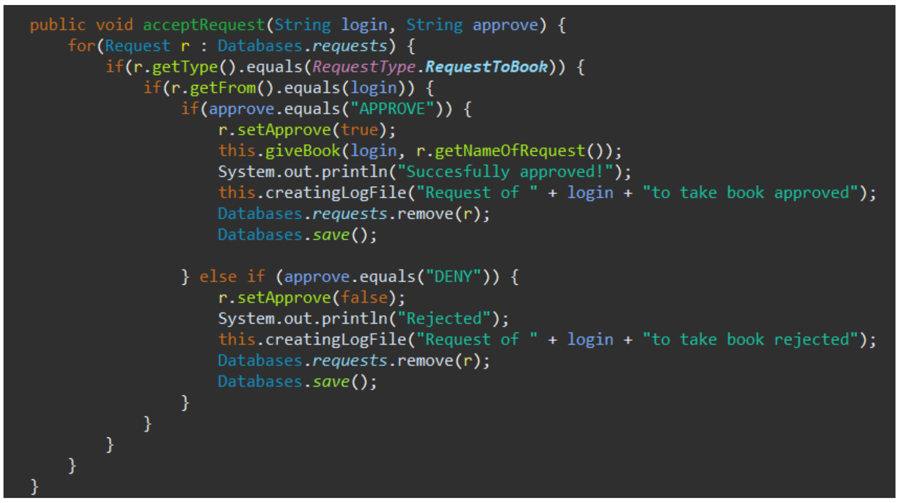
```java
public void viewFiles(String courseName) {
    Databases.files.stream().filter(n -> n.getCourseName().equals(courseName)).filter(n -> n.getFromWho().equals(this.getLogin())).forEach(x -> System.out.println(
}

public void setPointToStudent(String login, String courseName, Double point, int chooseTypeOfMarking) {
    for(User user: Databases.users) {
        if(login.equals(user.getLogin())) {
            Student a = (Student) user;
            a.setPoint(courseName, point, chooseTypeOfMarking);
        }
    }
}
public void setAttendToStudent(String login, String courseName, boolean attend) {
    Databases.users.stream().filter(n -> n instanceof Student).map(n -> (Student)n).filter(n -> n.getLogin().equals(login)).forEach(n -> n.setAttendance(courseName
}
```



```java
public void setRating(Double rate) {
    this.rate += rate;
    this.cnt += 1;
    Databases.save();
}

public Double getRating() {
    if(cnt == 0) {
        return rate;
    }
    return rate / cnt * 1.0;
}

public void createLesson(String courseName, String room, DayOfWeek day, int time, LessonType type) {
    for(Course c: courses) {
        if(c.getCourseName().equals(courseName)) {
            Lesson a = new Lesson(type, time, day, room, this.getLogin(), courseName);
            c.addLesson(a);
            Databases.save();
        }
    }
}
```

Librarian

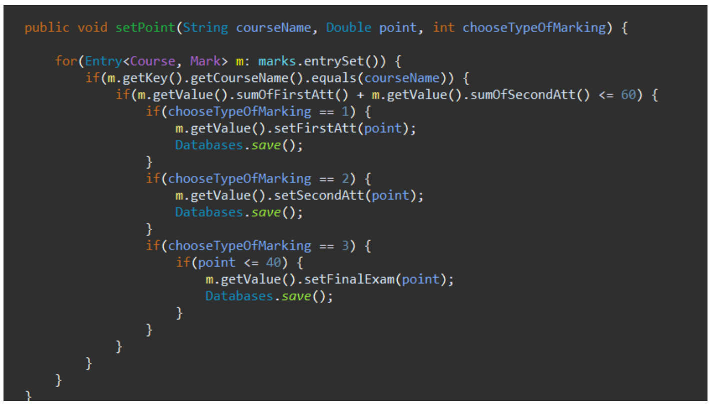
The librarian keeps books and has access to everything related to them. He can add and delete books, see the entire list of books using streams. Give the book to the student, see the request. The most important and difficult method is to respond to the request.



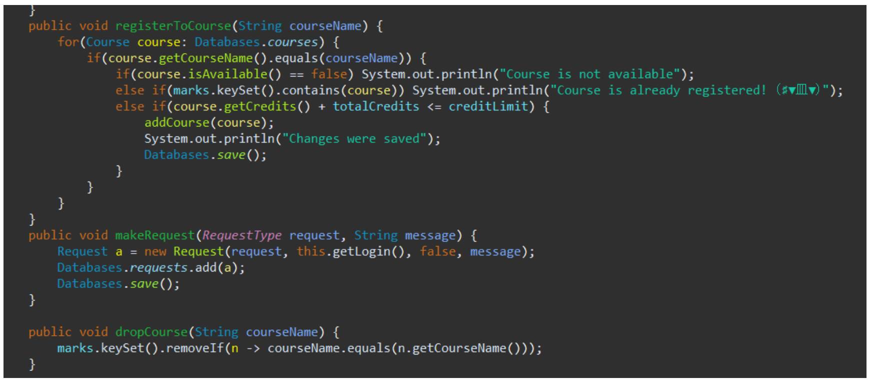
```java
public void acceptRequest(String login, String approve) {
    for(Request r : Databases.requests) {
        if(r.getType().equals(RequestType.RequestToBook)) {
            if(r.getFrom().equals(login)) {
                if(approve.equals("APPROVE")) {
                    r.setApprove(true);
                    this.giveBook(login, r.getNameOfRequest());
                    System.out.println("Succesfully approved!");
                    this.creatingLogFile("Request of " + login + "to take book approved");
                    Databases.requests.remove(r);
                    Databases.save();

                } else if (approve.equals("DENY")) {
                    r.setApprove(false);
                    System.out.println("Rejected");
                    this.creatingLogFile("Request of " + login + "to take book rejected");
                    Databases.requests.remove(r);
                    Databases.save();
                }
            }
        }
    }
}
```

Student

The student is one of the main classes in this project. He has the possibilities: to look at the list of books taken, to take a book for himself, to look at information about himself. View your grades, send a request and register for the course, skip the discipline, see your transcript and GPA, add yourself to the organization, most importantly, see your schedule, . The methods of grading, faculty replacement, add and delete a lesson, view the attendance, files and certification are prescribed in the class. He can see his organizations and registered courses, rate the teacher, and see the teacher's assessment, choose the time for the registration course, and also see the attendance.



```java
public void setPoint(String courseName, Double point, int chooseTypeOfMarking) {

    for(Entry<Course, Mark> m: marks.entrySet()) {
        if(m.getKey().getCourseName().equals(courseName)) {
            if(m.getValue().sumOfFirstAtt() + m.getValue().sumOfSecondAtt() <= 60) {
                if(chooseTypeOfMarking == 1) {
                    m.getValue().setFirstAtt(point);
                    Databases.save();
                }
                if(chooseTypeOfMarking == 2) {
                    m.getValue().setSecondAtt(point);
                    Databases.save();
                }
                if(chooseTypeOfMarking == 3) {
                    if(point <= 40) {
                        m.getValue().setFinalExam(point);
                        Databases.save();
                    }
                }
            }
        }
    }
}
```



```java
}
public void registerToCourse(String courseName) {
    for(Course course: Databases.courses) {
        if(course.getCourseName().equals(courseName)) {
            if(course.isAvailable() == false) System.out.println("Course is not available");
            else if(marks.keySet().contains(course)) System.out.println("Course is already registered! (♯▼皿▼)");
            else if(course.getCredits() + totalCredits <= creditLimit) {
                addCourse(course);
                System.out.println("Changes were saved");
                Databases.save();
            }
        }
    }
}
public void makeRequest(RequestType request, String message) {
    Request a = new Request(request, this.getLogin(), false, message);
    Databases.requests.add(a);
    Databases.save();
}

public void dropCourse(String courseName) {
    marks.keySet().removeIf(n -> courseName.equals(n.getCourseName()));
}
```

Enums and Interface

View Transcript - `void viewTranscript();`

Semester Type - `FALL, SPRING, SUMMER;`

Request Type - `RequestToCourse, RequestToBook;`

Professor Type - `Tutors, SeniorLectures, Professor, PracticeTutor;`

Manager Type - `OR, Departments;`

Lesson Type - `Practise, Lecture, laboratory;`

Gender - `MALE, FEMALE;`

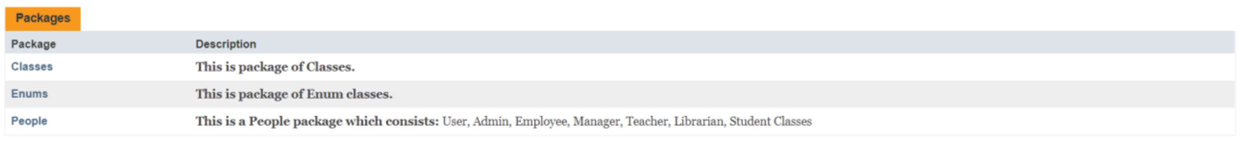
Faculty - `FIT, FGIG, FANGI, BS, ISE, KMA, NOCHI, BASIC;`

Degree - `Bachelor, Master, PHD;`

Day of Week - `MON, TUE, WED, THU, FRI, SAT, SUN;`

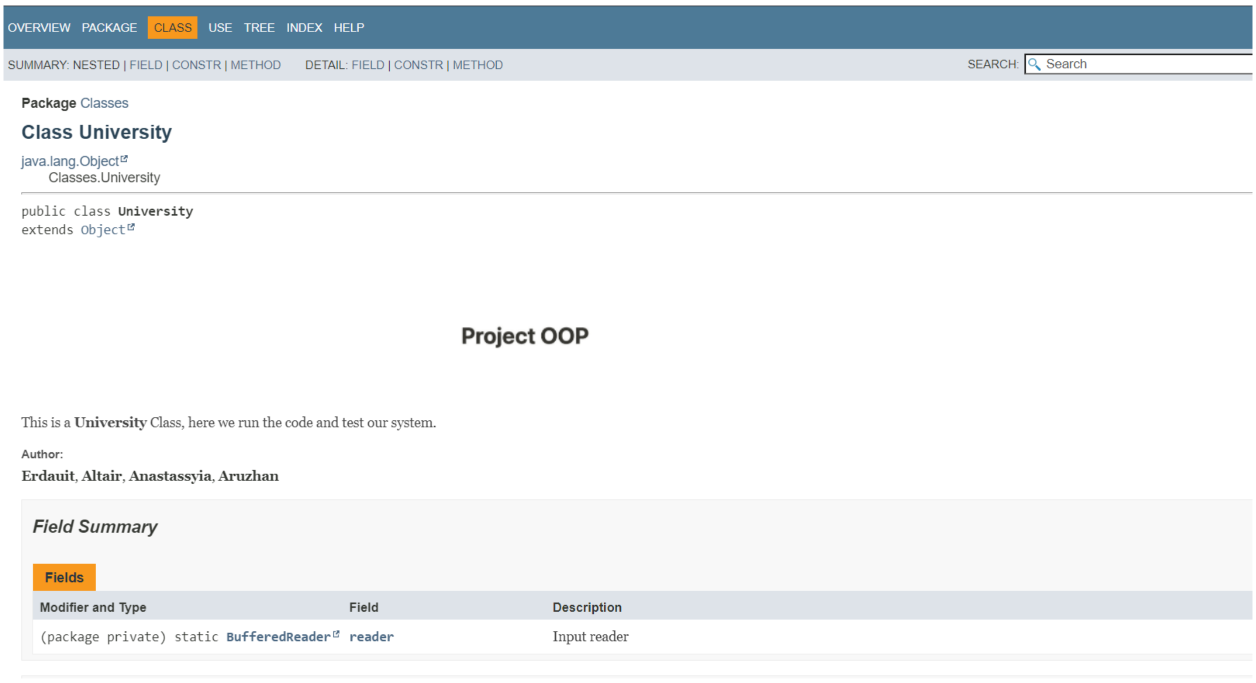
Documentation

Documentation was carried out by Aruzhan under the leadership of Captain Erdaut. With the help of comments in the code, class methods were described. We used «JavaDocs» to implement a nice interface and a clear description.



Packages

Package	Description
Classes	This is package of Classes.
Enums	This is package of Enum classes.
People	This is a People package which consists: User, Admin, Employee, Manager, Teacher, Librarian, Student Classes



OVERVIEW PACKAGE CLASS USE TREE INDEX HELP
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH: Search

Package Classes

Class University

java.lang.Object
Classes.University

```
public class University
extends Object
```

Project OOP

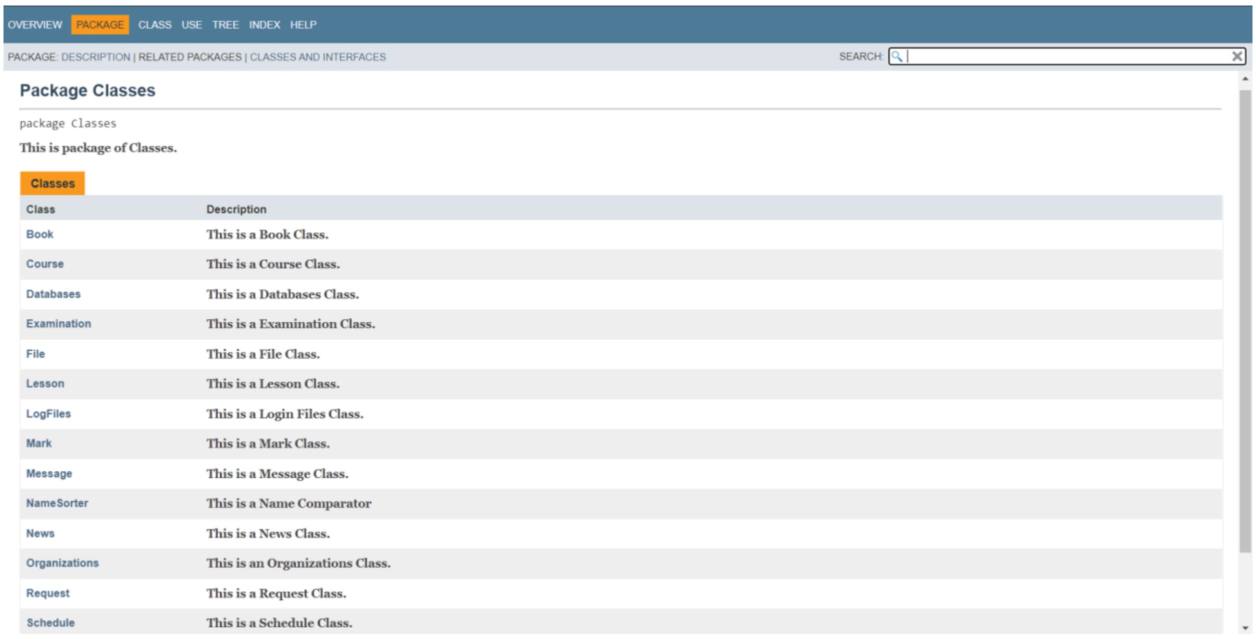
This is a **University** Class, here we run the code and test our system.

Author:
Erdauit, Altair, Anastassyia, Aruzhan

Field Summary

Fields

Modifier and Type	Field	Description
(package private) static BufferedReader	reader	Input reader



OVERVIEW PACKAGE CLASS USE TREE INDEX HELP
PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES
SEARCH:

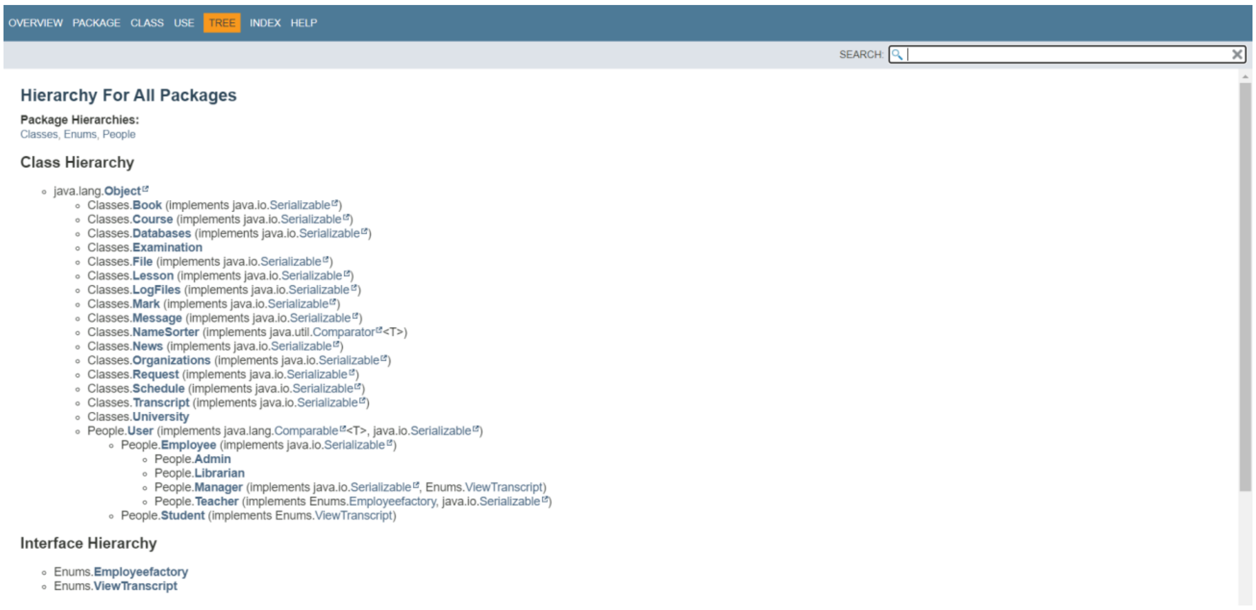
Package Classes

```
package Classes
```

This is package of Classes.

Classes

Class	Description
Book	This is a Book Class.
Course	This is a Course Class.
Databases	This is a Databases Class.
Examination	This is a Examination Class.
File	This is a File Class.
Lesson	This is a Lesson Class.
LogFiles	This is a Login Files Class.
Mark	This is a Mark Class.
Message	This is a Message Class.
NameSorter	This is a Name Comparator
News	This is a News Class.
Organizations	This is an Organizations Class.
Request	This is a Request Class.
Schedule	This is a Schedule Class.



Hierarchy For All Packages

Package Hierarchies:
Classes, Enums, People

Class Hierarchy

- java.lang.**Object**
  - Classes.**Book** (implements java.io.Serializable)
  - Classes.**Course** (implements java.io.Serializable)
  - Classes.**Databases** (implements java.io.Serializable)
  - Classes.**Examination**
  - Classes.**File** (implements java.io.Serializable)
  - Classes.**Lesson** (implements java.io.Serializable)
  - Classes.**LogFiles** (implements java.io.Serializable)
  - Classes.**Mark** (implements java.io.Serializable)
  - Classes.**Message** (implements java.io.Serializable)
  - Classes.**NameSorter** (implements java.util.Comparator<T>)
  - Classes.**News** (implements java.io.Serializable)
  - Classes.**Organizations** (implements java.io.Serializable)
  - Classes.**Request** (implements java.io.Serializable)
  - Classes.**Schedule** (implements java.io.Serializable)
  - Classes.**Transcript** (implements java.io.Serializable)
  - Classes.**University**
  - People.**User** (implements java.lang.Comparable<T>, java.io.Serializable)
    - People.**Employee** (implements java.io.Serializable)
      - People.**Admin**
      - People.**Librarian**
      - People.**Manager** (implements java.io.Serializable, Enums.ViewTranscript)
      - People.**Teacher** (implements Enums.Employeefactory, java.io.Serializable)
    - People.**Student** (implements Enums.ViewTranscript)

Interface Hierarchy

- Enums.**Employeefactory**
- Enums.**ViewTranscript**

The work process

Groups

We have created two teams. In Microsoft Tims and Telegram. At Tims, we called up and discussed important issues online. In the telegram, we were always in touch, sent each other all the files, monitored the changes and solved the issues of project implementation.

Platforms and methods used

Java is the #1 programming language and development platform that reduces cost, shortens development timeframes and improves application services. We have used this language in our project.

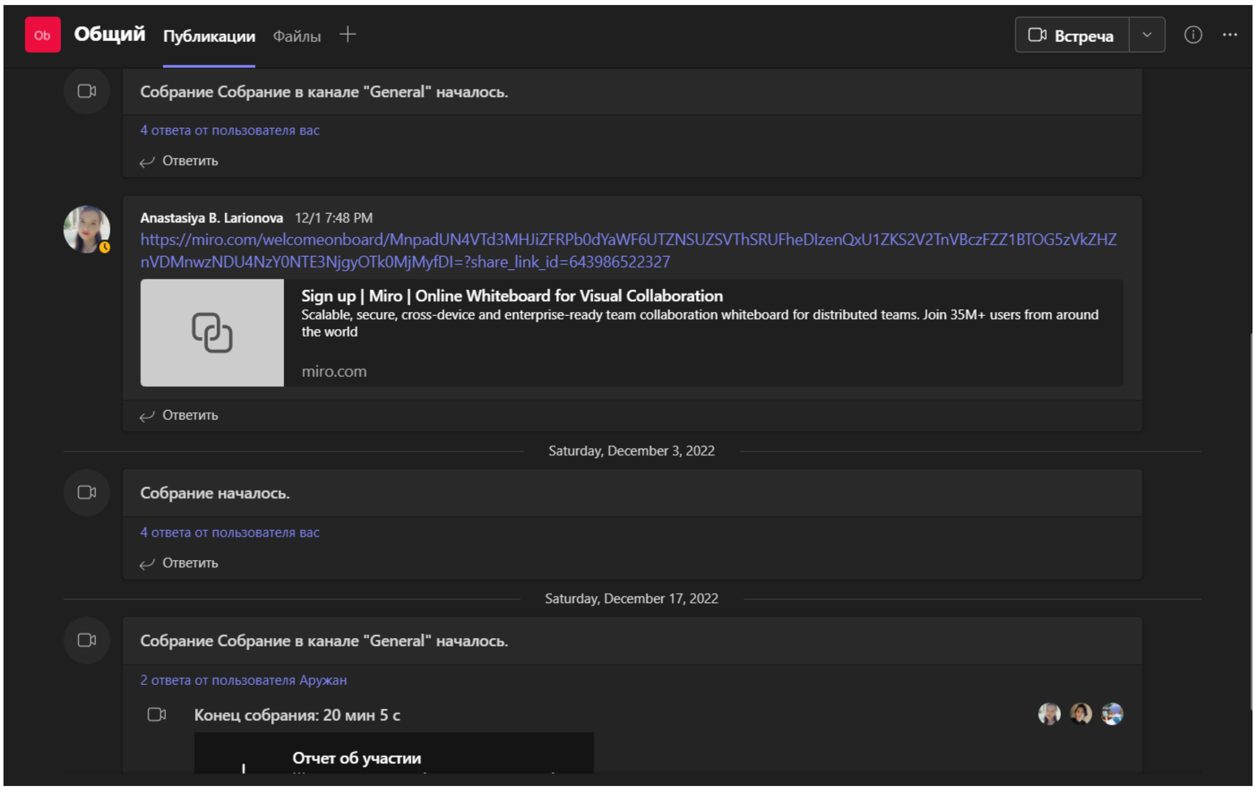
Top Coder – “Managing the UML Tool was an eye opening experience in just how effective the TopCoder software development model really is” - used to create a diagram and a case. Thanks to the clear and approximately clear layout of the buttons, the diagrams were thought out quickly and accurately.

We used **Google Excel** documents by creating a table of responsibilities, this was done in order to mentor all the creation processes, observe what tasks need to be completed and what has already been done, as well as thanks to deadlines and a description, there were minimal questions and embarrassing situations.

Miro board - is an online whiteboard for group work. In order to always be aware of visual progress, assistance, and changes, we used this service, as it allows you to see changes in real time for all participants.

Teams - we called up with the team, since we did not always have offline meetings, and the issues needed to be resolved, or to show progress, then we kept a record of the meeting, for those from the team who could not join.

Screenshots



Собрание Собрание в канале "General" началось.

4 ответа от пользователя вас

Ответить

Anastasiya B. Larionova 12/1 7:48 PM

https://miro.com/welcomeonboard/MnpadUN4VTd3MHJiZFRPb0dYaWF6UTZNSUZSVThSRUFheDIzenQxU1ZKS2V2TnVBczFZZ1BTOG5zVkZHZnVDMnwzNDU4NzY0NTE3NjgyOTk0MjMyfDI=?share_link_id=643986522327

Sign up | Miro | Online Whiteboard for Visual Collaboration

Scalable, secure, cross-device and enterprise-ready team collaboration whiteboard for distributed teams. Join 35M+ users from around the world

miro.com

Ответить

Saturday, December 3, 2022

Собрание началось.

4 ответа от пользователя вас

Ответить

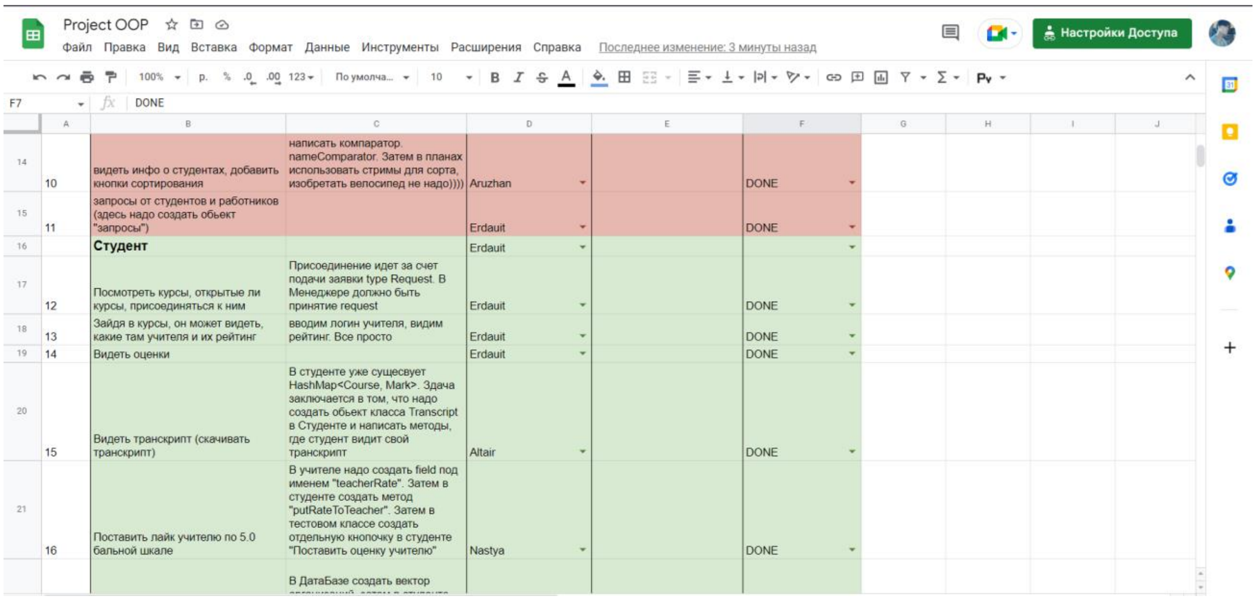
Saturday, December 17, 2022

Собрание Собрание в канале "General" началось.

2 ответа от пользователя Аружан

Конец собрания: 20 мин 5 с

Отчет об участии

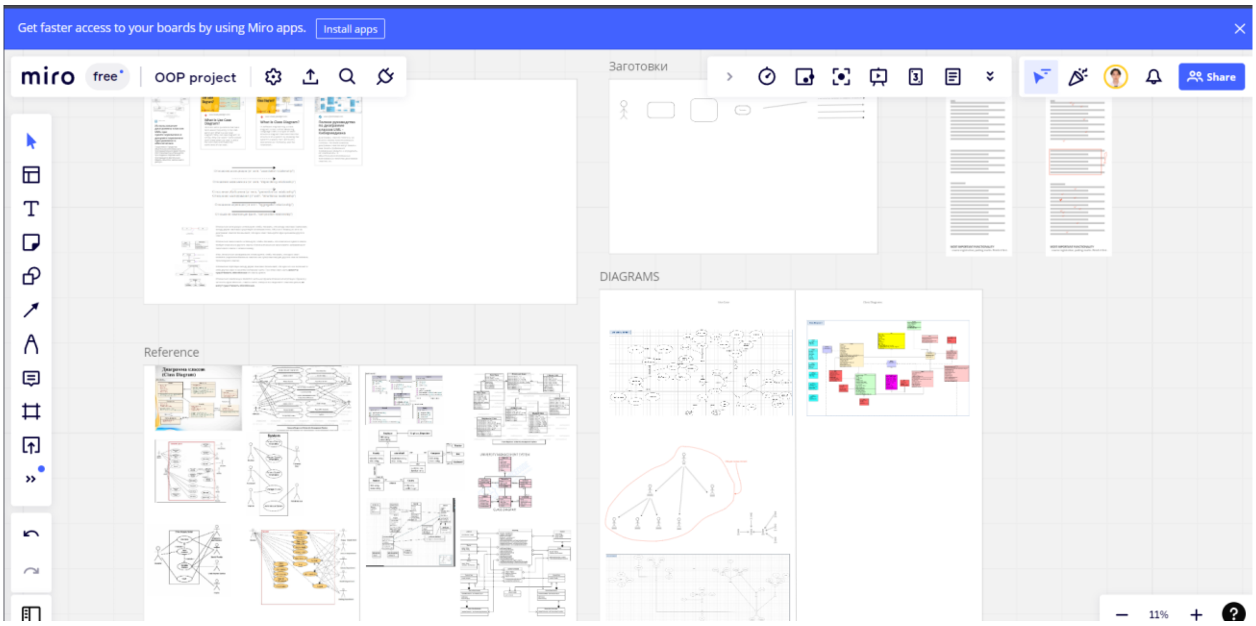


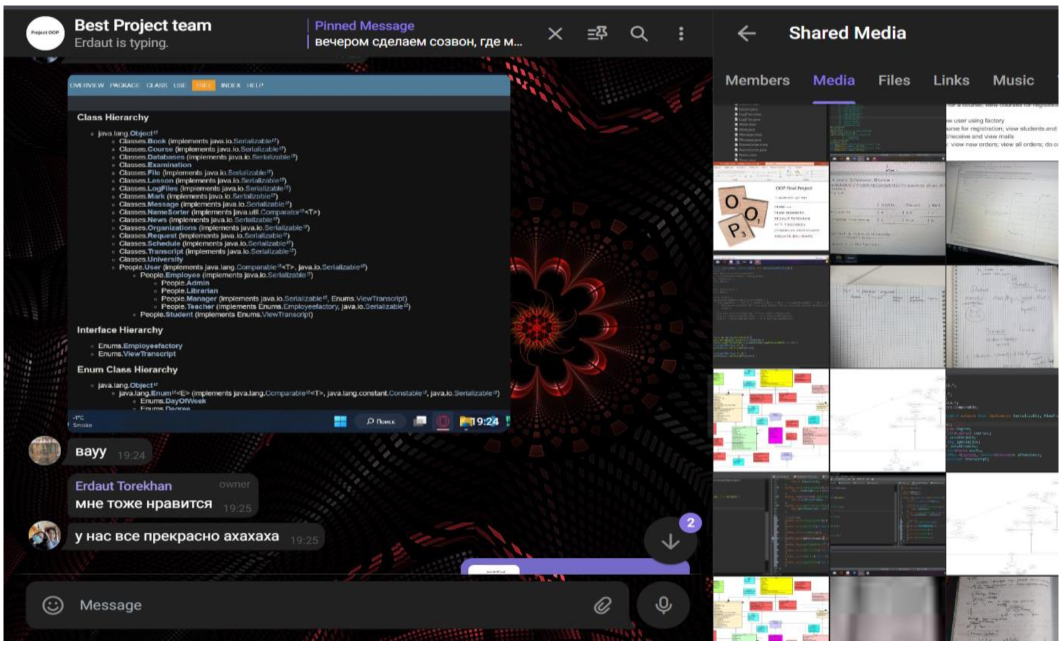
Project OOP
Файл Правка Вид Вставка Формат Данные Инструменты Расширения Справка
Последнее изменение: 3 минуты назад
Настройки Доступа
100%
По умолча...
10

F7 DONE

	A	B	C	D	E	F
14	10	видеть инфо о студентах, добавить кнопки сортирования	написать компаратор. nameComparator. Затем в планах использовать стримы для сорта, изобретать велосипед не надо))))	Aruzhan		DONE
15	11	запросы от студентов и работников (здесь надо создать обьект "запросы")		Erdauit		DONE
16		**Студент**		Erdauit		
17	12	Посмотреть курсы, открытые ли курсы, присоединяться к ним	Присоединение идет за счет подачи заявки type Request. В Менеджере должно быть принятие request	Erdauit		DONE
18	13	Зайдя в курсы, он может видеть, какие там учителя и их рейтинг	вводим логин учителя, видим рейтинг. Все просто	Erdauit		DONE
19	14	Видеть оценки		Erdauit		DONE
20	15	Видеть транскрипт (скачивать транскрипт)	В студенте уже сущесвует HashMap<Course, Mark>. Здача заключается в том, что надо создать обьект класса Transcript в Студенте и написать методы, где студент видит свой транскрипт	Altair		DONE
21	16	Поставить лайк учителю по 5.0 бальной шкале	В учителе надо создать field под именем "teacherRate". Затем в студенте создать метод "putRateToTeacher". Затем в тестовом классе создать отдельную кнопочку в студенте "Поставить оценку учителю"	Nastya		DONE
			В ДатаБазе создать вектор			

Link - https://docs.google.com/spreadsheets/d/1e9b_GCv8ItNmRPMJVyvbyEO084k-IwykEDZgX6DZR9o/edit#gid=0





```
Class Hierarchy

  ◦ java.lang.Object
      ▪ Classes.Book (implements java.io.Serializable)
      ▪ Classes.Course (implements java.io.Serializable)
      ▪ Classes.Databases (implements java.io.Serializable)
      ▪ Classes.Examination
      ▪ Classes.File (implements java.io.Serializable)
      ▪ Classes.Lesson (implements java.io.Serializable)
      ▪ Classes.LogFiles (implements java.io.Serializable)
      ▪ Classes.Mark (implements java.io.Serializable)
      ▪ Classes.Message (implements java.io.Serializable)
      ▪ Classes.NameSorter (implements java.util.Comparator<T>)
      ▪ Classes.News (implements java.io.Serializable)
      ▪ Classes.Organizations (implements java.io.Serializable)
      ▪ Classes.Request (implements java.io.Serializable)
      ▪ Classes.Schedule (implements java.io.Serializable)
      ▪ Classes.Transcript (implements java.io.Serializable)
      ▪ Classes.University
      ▪ People.User (implements java.lang.Comparable<T>, java.io.Serializable)
          ▪ People.Employee (implements java.io.Serializable)
              ▪ People.Admin
              ▪ People.Librarian
              ▪ People.Manager (implements java.io.Serializable, Enums.ViewTranscript)
              ▪ People.Teacher (implements Enums.Employeefactory, java.io.Serializable)
          ▪ People.Student (implements Enums.ViewTranscript)

Interface Hierarchy

  ◦ Enums.Employeefactory
  ◦ Enums.ViewTranscript

Enum Class Hierarchy

  ◦ java.lang.Object
      ▪ java.lang.Enum<E> (implements java.lang.Comparable<T>, java.lang.constant.Constable, java.io.Serializable)
          ▪ Enums.DayOfWeek
          ▪ Enums.Degree
```

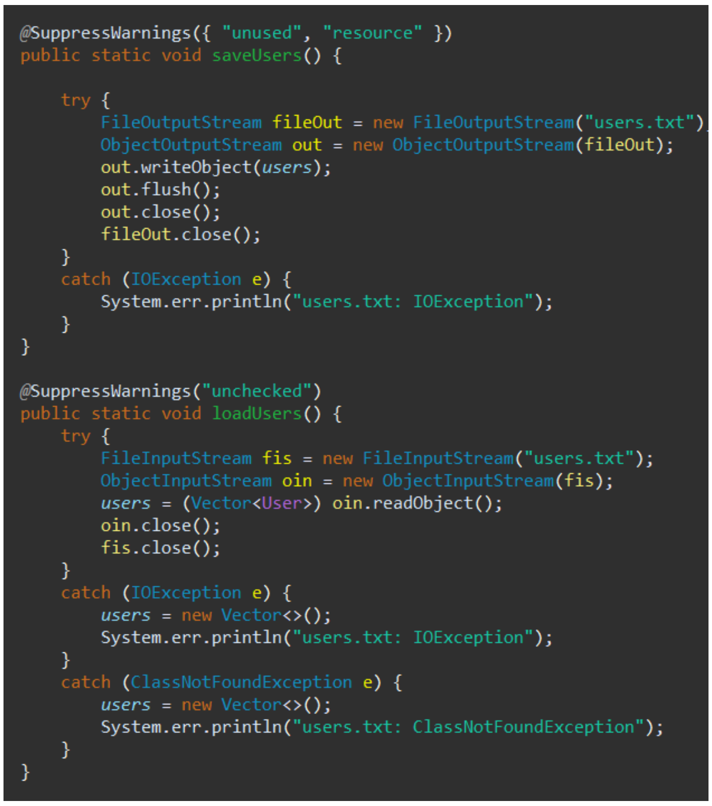
вауу 19:24

Erdaut Torekhan owner
мне тоже нравится 19:25

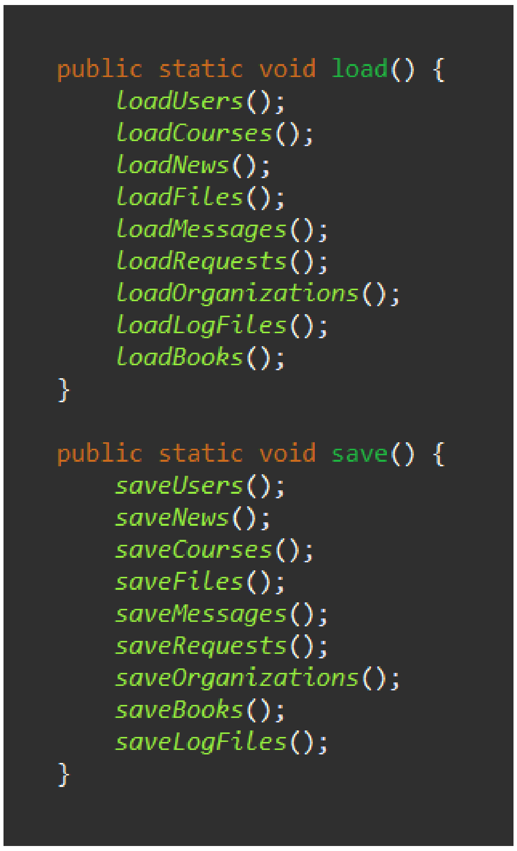
у нас все прекрасно ахахаха 19:25

Problems and solutions

There were minor problems during the implementation, but they were easily solved and quickly fixed with the help of the knowledge of other team members. The main problem was to implement Serialization. But after a long time and effort spent, everything worked and now progress is maintained. There were also difficulties with the implementation of the schedule, a solution was found among the team members, thanks to which a beautiful schedule is now displayed in the console. Serialization:



```java
@SuppressWarnings({ "unused", "resource" })
public static void saveUsers() {

    try {
        FileOutputStream fileOut = new FileOutputStream("users.txt")
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(users);
        out.flush();
        out.close();
        fileOut.close();
    }
    catch (IOException e) {
        System.err.println("users.txt: IOException");
    }
}

@SuppressWarnings("unchecked")
public static void loadUsers() {
    try {
        FileInputStream fis = new FileInputStream("users.txt");
        ObjectInputStream oin = new ObjectInputStream(fis);
        users = (Vector<User>) oin.readObject();
        oin.close();
        fis.close();
    }
    catch (IOException e) {
        users = new Vector<>();
        System.err.println("users.txt: IOException");
    }
    catch (ClassNotFoundException e) {
        users = new Vector<>();
        System.err.println("users.txt: ClassNotFoundException");
    }
}
```



```java
public static void load() {
    loadUsers();
    loadCourses();
    loadNews();
    loadFiles();
    loadMessages();
    loadRequests();
    loadOrganizations();
    loadLogFiles();
    loadBooks();
}

public static void save() {
    saveUsers();
    saveNews();
    saveCourses();
    saveFiles();
    saveMessages();
    saveRequests();
    saveOrganizations();
    saveBooks();
    saveLogFiles();
}
```

Conclusion

The main goal of this project was to create a Client - Friendly product that will not be difficult to use, and view allows us to display it in a beautiful way for the user. After analyzing the work done, you can summarize that the work was done in detail and thoughtfully.

Together with the team, we discussed each step, how best to perform and implement it. After spending a lot of time discussing, they got to work, everyone had their own responsibilities, which they fulfilled. Working on this project, we developed our hard and soft skills, learned how to work clearly and clearly in a team, used various services to work, check and study the materials necessary for the project. We repeated and consolidated the knowledge gained in the lessons and implemented serialization, polymorphism, inheritance, patterns, and methods. It was very interesting to do this work, although it was not without difficulties, but in the end everything works.

Thank you very much to the teacher and wonderful person Shamoi Pakita for giving such a wonderful project and always helping when something was difficult, giving advice and improving the quality of the project