Report: OOP Project

School of Information Technology and Engineering

Discipline: Object-Oriented programming

Project name: University System

TEAM: «W213»

TEAM MEMBERS:

Bereket Yergali

Islam Khassangaliyev

Askar Zhumabayev

Daniyal Tuzelbayev

TEACHER:

SHAMOI PAKITA

Contents

Introduction

The primary objective of the project is to develop a university platform that facilitates various processes for users. Initially, we crafted diagrams to comprehend the fundamental structure of our project and specify methods for each class. Then we moved to coding or the creation of classes, employing diverse patterns and methodologies. While coding we also made minor changes in the UML diagram.

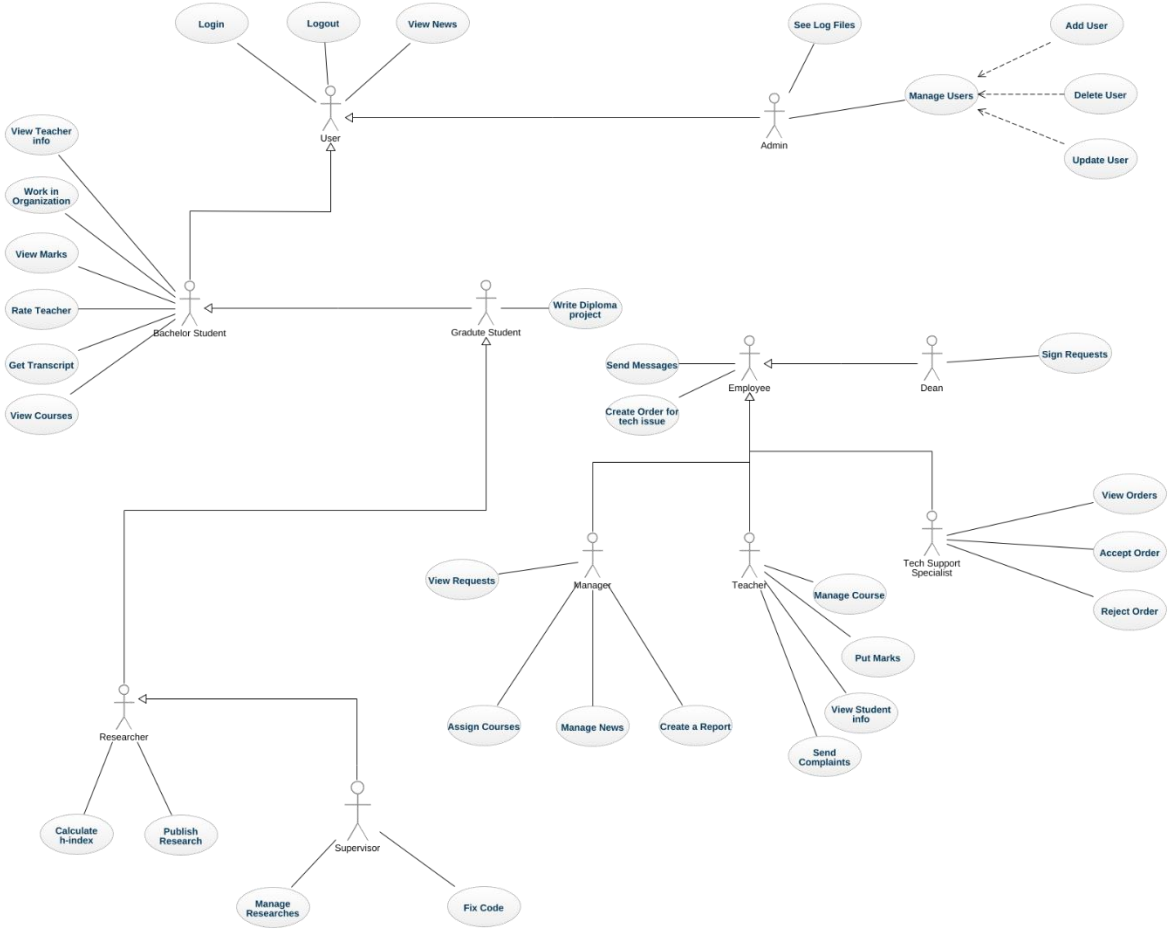
Briefly about the project

The project involves creating a comprehensive system with entities like User, Employee, Teacher, Manager, Student, etc. Key features include support for different lesson types, multilingual options, and categorization of students (bachelor, master, PhD). Teachers can send complaints, and there are constraints for Researcher supervision. The system accommodates various roles, including Teachers and Students as Researchers, with functionalities such as sorting and printing research papers, calculating h-index, and generating citations. Other features include project management, report generation, tech support, diploma projects, news, and journal subscriptions. Adherence to OOP principles, design patterns, and specific functionalities for different roles ensures a well-organized and user-friendly system.

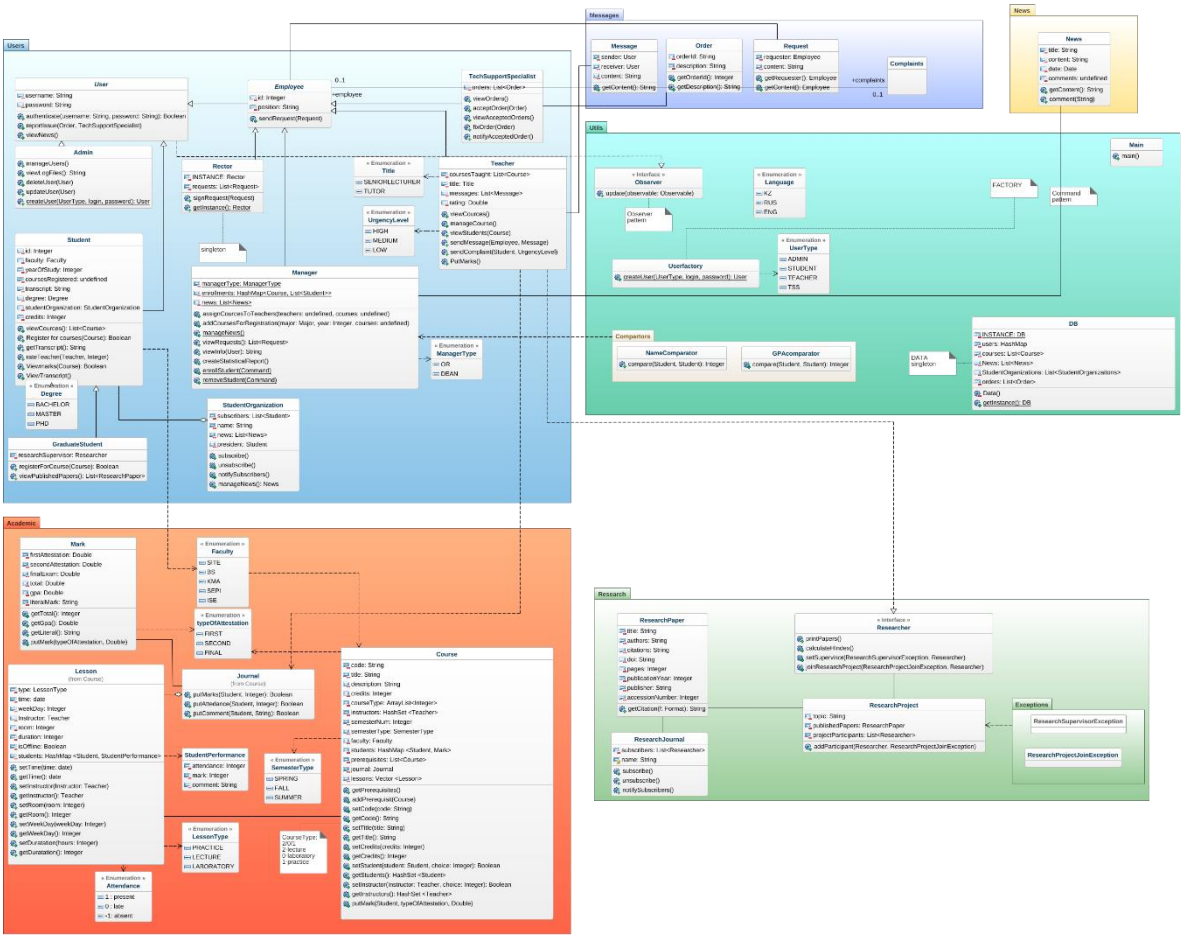
Main Part

Diagrams

Use Case Diagram



Class Diagram



Code

Academic Package

```
public class Course implements Serializable {
    private String code;
    private String title;
    private String description;
    private int credits;
    private String courseType;
    private int semesterNum;
    private SemesterType semesterType;
    private Faculty faculty;

    private HashMap<Student, Mark> students;
    private HashSet<Teacher> instructors;
    private HashSet<Course> prerequisites;
    private HashSet<Lesson> lessons;

    public Course() {
        students = new HashMap<>();
        instructors = new HashSet<>();
        prerequisites = new HashSet<>();
        lessons = new HashSet<>();

    }
```

```
    public Course(String code, String title, String description, int
credits, String courseType,
            SemesterType semesterType, Faculty faculty) {
        this();
        this.code = code;
        this.title = title;
        this.description = description;
        this.credits = credits;
        this.courseType = courseType;
        this.semesterType = semesterType;
        this.faculty = faculty;

        DB.getInstance().getCourses().add(this);
    }



public class Journal implements Serializable {
    public void putMark(Student student , Course course , int mark ,
typeOfAttestation type){
        course.putMark(student, type, mark);
    }
    public void putMark(Student student , Lesson lesson , int mark){
        lesson.putMark(student, mark); //
    }

    public void putAttendance(Student student, Lesson lesson, int att){
        lesson.putAttendance(student, att);
    }
    public void putComment(Student student, Lesson lesson, String comment){
        lesson.putComment(student, comment);
    }

    public void menu(){
        // List teacher.getCourseTaught()
        // teacher chooses one course (c)
        // teacher chooses putMark() on attestation OR
putMark/Attendance/Comment on lecture
        // case attestation: putMark(Student student , Course course , int
mark , typeOfAttestation type)
        // case lecture: List course.getLectures()
        // teachers chooses one lecture
        // teacher types a id or some identifer for student
        // teacher chooses what to do -> 1: putMark 2: putAttendance 3:
putComment
        // for each case there's a function;
    }
}
public class Lesson implements Serializable {
    private LessonType type; // lab, practice, lesson
    private LocalTime time; // 14:00
    private int weekDay; // 0: monday 1: tuesday ....
    private Teacher instructor;
    private int room;
    private int duration;  // in hours
    private boolean isOffline;
    private HashMap<Student, StudentPerformance> students; // Student:
Attendance(Enum) / Mark / Comment

    public Lesson() {
        students = new HashMap<>();
```

```
    }
    public Lesson(LessonType type, LocalTime time, int weekDay, Teacher
instructor, int room, int duration, boolean isOffline, Collection<Student>
students) {
        this();

        this.type = type;
        this.time = time;
        this.weekDay = weekDay;
        this.instructor = instructor;
        this.room = room;
        this.duration = duration;
        this.isOffline = isOffline;

        setStudents(students);
    }

    // Getters-Setters
    public LessonType getType() {
        return type;
    }
    public void setType(LessonType type) {
        this.type = type;
    }
    public LocalTime getTime() {
        return time;
    }
    public void setTime(LocalTime time) {
        this.time = time;
    }
    public int getWeekDay() {
        return weekDay;
    }
    public void setWeekDay(int weekDay) {
        if(weekDay >= 0 && weekDay <= 6) {
            this.weekDay = weekDay;
        }
        else {
            System.out.println("\nDebug: input exceed range of weekDays\n");
        }
    }
    public int getRoom() {
        return room;
    }
    public void setRoom(int room) {
        this.room = room;
    }
    public int getDuration() {
        return duration;
    }
    public void setDuration(int duration) {
        this.duration = duration;
    }
    public boolean isOffline() {
        return isOffline;
    }
    public void setOffline(boolean isOffline) {
        this.isOffline = isOffline;
    }
    public HashMap<Student,StudentPerformance> getStudents() {
        return students;
```

```
    }
    public void setStudents(Collection<Student> students) {
       for(Student student: students){
          this.students.put(student, new StudentPerformance());
       }
    }
    public Teacher getInstructor() {
       return instructor;
    }
    public void setInstructor(Teacher instructor) {
       this.instructor = instructor;
    }
```

Messages

```
public class Request implements Serializable {
    private Employee requester;
    private String content;
    public Request(Employee requester, String content) {
        this.requester = requester;
        this.content = content;
    }
    public Employee getRequester() {
        return requester;
    }
    public void setRequester(Employee requester) {
        this.requester = requester;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
public class Order implements Serializable {
    private static int num;
    private int orderId;

    private String description;

    public Order(String description) {
        this.description = description;
        orderId = num;
        num++;
    }

    public static int getNum() {
        return num;
    }

    public static void setNum(int num) {
        Order.num = num;
    }

    public int getOrderId() {
        return orderId;
    }
```

```
    public void setOrderId(int orderId) {
        this.orderId = orderId;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Order order = (Order) o;
        return getOrderId() == order.getOrderId();
    }

    @Override
    public int hashCode() {
        return Objects.hash(getOrderId());
    }
}
public class Message implements Serializable {

    private User sender;
    private User receiver;
    private String content;

    public Message(User sender, User receiver, String content) {
        this.sender = sender;
        this.receiver = receiver;
        this.content = content;
    }

    public User getSender() {
        return sender;
    }

    public void setSender(User sender) {
        this.sender = sender;
    }

    public User getReceiver() {
        return receiver;
    }

    public void setReceiver(User receiver) {
        this.receiver = receiver;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
```

```
    @Override
    public String toString() {
        return "Message{" +
                "Sender: " + sender + + '\n' +
                "Content: " + content + '\'' +
                '}';
    }
}
public class Complaint implements Serializable {
    private Student student;
    private String description;
    private UrgencyLevel urgencyLevel;
    private Teacher complainant;
    //
    public Complaint(Student student,String description, UrgencyLevel
urgencyLevel, Teacher complainant) {
        this.student = student;
        this.description = description;
        this.urgencyLevel = urgencyLevel;
        this.complainant = complainant;
    }

    public String getDescription() {
        return description;
    }

    public UrgencyLevel getUrgencyLevel() {
        return urgencyLevel;
    }

    public Teacher getComplainant() {
        return complainant;
    }

}
```

News

```
public class News implements Comparable<News>, Serializable {

    private String title;
    private String content;
    private Date date;
    private Vector<String> comments = new Vector<String>();

    //CONSTRUCTORS
    public News(String title, String content) {
        this.title = title;
        this.content = content;
        date = new Date();
    }
    //GETTER AND SETTER


    public String getTitle() {
        return title;
    }

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

```
    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
    public void addComment(String comment){
        comments.add(comment);
    }

    public Date getDate() {
        return date;
    }

    public Vector<String> getComments() {
        return comments;
    }

    @Override
    public String toString() {
        return
                "Title:'" + title + '\n' +
                "Content:'" + content + '\n' +
                "Date=" + date + '\n' +
                "Comments=" + comments
                ;
    }

    @Override
    public int compareTo(News o) {
        return this.date.compareTo(o.date);
    }
}
```

Research

```
public interface Researcher {
    /**
     * Prints the research papers of the researcher.
     *
     * @param comparator The comparator to determine the sorting order of
the papers.
     */
    void printPapers(Comparator<ResearchPaper> comparator);

    /**
     * Calculates the H-index of the researcher based on their research
papers.
     *
     * @return The calculated H-index.
     */
    int calculateHIndex();

    /**
     * Gets the name of the researcher.
     *
     * @return The name of the researcher.
     */
    String getResearcherName();

    /**
     * Sets the supervisor for the researcher.
```

```
     *
     * @param supervisor The supervisor to be set for the researcher.
     * @throws Research.Exceptions.ResearchSupervisorException If an error
occurs while setting the supervisor.
     */
    void setSupervisor(Researcher supervisor) throws
ResearchSupervisorException;
}
```

Users

```
public abstract class User implements Observer, Serializable, Researcher {

    private String username;
    private String password;
    protected UserType userType;
    protected Language language = ENG;
    protected static Scanner in = new Scanner(System.in);
    private boolean isResearcher;
    private List<ResearchPaper> allResearchPapers =
ResearchPaper.loadAllResearchPapers();
    private static final String RESEARCHER_FILE_PATH = "researchersDB.dat";

    /**
     * Constructs a User with specified username, password, and user type.
     * Adds the user to the database and initializes the researcher status
based on the user type.
     *
     * @param username the username for the user.
     * @param password the password for the user.
     * @param ut the type of the user (e.g., STUDENT, TEACHER).
     */
    public User(String username, String password, UserType ut) {
        this.username = username;
        this.password = password;
        this.userType = ut;
        DB.getInstance().addUser(this, UserType.USER);

        if (this instanceof GraduateStudent) {
            isResearcher = true;
        }
        isResearcher = false;
    }

    public User() {
    }
    {

    }
    /**
     * Reports an issue to the system.
     *
     * @param description a description of the issue to be reported.
     */
    public void reportIssue(String description) {
        DB.getInstance().addOrder(new Order(description));
    }

    //getter and setter
    public UserType getUserType() {
        return userType;
    }
```

```java
    protected void setUserType(UserType userType) {
        this.userType = userType;
    }
    public String getUsername() {
        return username;
    }

    protected void setUsername(String username) {
        this.username = username;
    }

    protected String getPassword() {
        return password;
    }

    protected void setPassword(String password) {
        this.password = password;
    }

    protected Language getLanguage() {
        return language;
    }

    protected void setLanguage(Language language) {
        this.language = language;
    }


    //MENU METHODS
    /**
     * MENU methodtars
     *
     * @throws IOException if an I/O error occurs.
     */
    public abstract void run() throws IOException;
    /**
     * Displays the menu based on the user's selected language.
     */
    protected void displayMenu() {
        if (language == ENG) displayEnglishMenu();
        else if (language == KZ) displayKazakhMenu();
        else if (language == RUS) displayRussianMenu();
    }

    protected abstract void displayRussianMenu();

    protected abstract void displayKazakhMenu();

    protected abstract void displayEnglishMenu();
    /**
     * Authenticates a user based on username and password.
     *
     * @return the authenticated User object.
     * @throws UserNotFound if no user is found with the given credentials.
     */
    public static User authenticate() throws UserNotFound {
        System.out.println("Enter username: ");
        String username = in.nextLine();

        System.out.println("Enter password: ");
        String password = in.nextLine();
```

```
        List<User> users =
DB.getInstance().getUsersByUserType(UserType.USER);
        for (User user: users){
            if(user.getUsername().equals(username) &&
user.getPassword().equals(password)){
                return user;
            }
        }
        throw new UserNotFound();
    }
    /**
     * Saves the current state of the application.
     *
     * @throws IOException if an I/O error occurs during saving.
     */
    protected void save() throws IOException {
        DB.serializeAll();
    }

    protected void exit() {
        if(language == ENG) System.out.println("Bye bye!");
        else if(language == KZ) System.out.println("Сауболыңыз!");
        else System.out.println("До свидания!");
        try {
            save();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * Handles exceptions by printing an error message and saving the state
of the application.
     *
     * @param e the exception to be handled.
     * @throws IOException if an I/O error occurs during saving.
     */

    protected void handleError(Exception e) throws IOException {
        if (language == KZ) System.out.println("Ойбай, қате...");
        else if (language == RUS) System.out.println("Ошибка....");
        else System.out.println("Error... ");
        e.printStackTrace();
        save();
    }
    /**
     * Displays a welcome message based on the user's selected language.
     */

    protected void getWelcomeMessage(){
        if(language == KZ) System.out.println("Қош келдіңіз!");
        else if(language == RUS) System.out.println("Добро пожаловать!");
        else System.out.println("Welcome!");
    }

    //NEWS
    /**
     * Displays a list of all news articles available in the system and
allows the user to view a selected article.
     * After displaying the list, the user can select a news article by its
index to view it in detail.
     * The user also has an option to add a comment to the selected news
```

```
article.
     */
    public void viewAllNews() {
        List<News> newsList = DB.getInstance().getNews();

        if (newsList.isEmpty()) {
            System.out.println("No news articles available.");
            return;
        }

        System.out.println("List of News Articles:");
        for (int i = 0; i < newsList.size(); i++) {
            News news = newsList.get(i);
            System.out.println("Index " + i + ": " + news.getTitle());
        }

        System.out.print("Enter the index of the news article to view (or -
1 to exit): ");
        int selectedIndex = in.nextInt();

        if (selectedIndex >= 0 && selectedIndex < newsList.size()) {
            News selectedNews = newsList.get(selectedIndex);
            System.out.println(selectedNews);

            System.out.print("Would you like to add a comment (yes/no)? ");
            in.nextLine();
            String response = in.nextLine().toLowerCase();

            if (response.equals("yes")) {
                System.out.print("Enter your comment: ");
                String comment = in.nextLine();
                selectedNews.addComment(comment);
                System.out.println("Comment added successfully.");
            } else {
                System.out.println("No comment added.");
            }
        } else if (selectedIndex == -1) {
            System.out.println("Exiting.");
        } else {
            System.out.println("Invalid index. Please try again.");
        }
    }

    //LANGUAGE
    /**
     * Provides an interface for the user to change their preferred
language.
     * The method displays language options (Kazakh, Russian, English) and
sets the user's language based on their choice.
     * The user must enter a valid choice number to change the language
setting.
     */
    protected void changeLanguage(){
        System.out.println("1. Қазақша \n 2. Руский \n 3. English");

        int choice;
        do {
            System.out.print("Enter your choice: ");
            choice = in.nextInt();

            if (choice < 1 || choice > 3) {
```

```
                System.out.println("Invalid choice. Please enter a number
between 1 and 3.");
            }
        } while (choice < 1 || choice > 3);

        if (choice == 1) {
            setLanguage(Language.KZ);
        } else if (choice == 2) {
            setLanguage(Language.RUS);
        } else if (choice == 3) {
            setLanguage(Language.ENG);
        }
    }

    /**
     * Notifies the researcher about the publication of a new scientific
work.
     *
     * @param journalName The name of the journal where the paper was
published.
     * @param paperTitle  The title of the published scientific work.
     */
    @Override
    public void update(String journalName, String paperTitle) {
        System.out.println(username + "!\nA new scientific work entitled "
+ paperTitle + "  was published in the journal " + journalName);
    };

    /**
     * Sets the researcher status for the user.
     * Throws CannotBecomeResearcherException if the user is not eligible
to be a researcher.
     *
     * @throws CannotBecomeResearcherException If the user cannot become a
researcher.
     */
    public void setIsResearcher() throws CannotBecomeResearcherException {
        if (this instanceof Student || this instanceof Teacher || (this
instanceof Employee && this.getClass()==Employee.class)) {
            this.isResearcher = true;
        } else {
            throw new CannotBecomeResearcherException("This User cannot be
a Researcher!");
        }
    }
    /**
     * Checks if User is Researcher.
     *
     * @return True if Researcher, otherwise False.
     */
    public boolean isUserResearcher() {
        return isResearcher;
    }

    /**
     * Prints the research papers for the researcher, sorted based on the
provided comparator.
     *
     * @param comparator The comparator to determine the sorting order of
research papers.
     */
```

```java
    @Override
    public void printPapers(Comparator<ResearchPaper> comparator) {
        allResearchPapers.sort(comparator);

        System.out.println("Research Papers for " + getResearcherName() +
":");
        for (ResearchPaper paper : allResearchPapers) {
            System.out.println(paper);
        }
    }

    /**
     * Gets the list of all research papers associated with the researcher.
     *
     * @return The list of all research papers.
     */
    public List<ResearchPaper> getAllResearchPapers() {
        return allResearchPapers;
    }

    /**
     * Calculates the H-index of the researcher based on their authored
papers and citations.
     *
     * @return The calculated H-index.
     */
    @Override
    public int calculateHIndex() {
        if (!isResearcher) {
            System.out.println("User is not a researcher. Error");
            return 404;
        }

        List<Integer> citationsList = allResearchPapers.stream()
                .filter(paper -> paper.getAuthors().contains(this))
                .map(paper -> paper.getCitations().size())
                .sorted(Comparator.reverseOrder())
                .toList();

        int hIndex = 0;
        for (int i = 0; i < citationsList.size(); i++) {
            int citations = citationsList.get(i);
            if (citations >= i + 1) {
                hIndex = i + 1;
            } else {
                break;
            }
        }

        return hIndex;
    }

    /**
     * Gets the name of the researcher.
     *
     * @return The name of the researcher or a message indicating that the
user is not a researcher.
     */
    @Override
    public String getResearcherName() {
        if (isResearcher) {
```

```
            return username;
        }
        return "This User is not a Researcher";
    }
    /**
     * Sets the supervisor for the graduate student.
     * Throws ResearchSupervisorException if the supervisor's H-index is
less than 3.
     *
     * @param supervisor The supervisor to be set.
     * @throws ResearchSupervisorException If the supervisor's H-index is
less than 3.
     */
    @Override
    public void setSupervisor(Researcher supervisor) throws
ResearchSupervisorException {
        if (isResearcher && this instanceof GraduateStudent
graduateStudent) {

            if (supervisor.calculateHIndex() < 3) {
                throw new
Research.Exceptions.ResearchSupervisorException("Supervisor must have an h-
index of 3 or higher");
            }

            graduateStudent.researchSupervisor = supervisor;
        }
    }


}
public class Student extends User implements Serializable {

    // Attributes specific to the Student class
    protected Integer id;
    protected Faculty faculty;
    protected Integer yearOfStudy = 1;
    protected Vector<Course> coursesRegistered = new Vector<>();
    protected Degree degree;
    protected StudentOrganization studentOrganization = null;
    protected Integer credits = 0;



    public Student(String username, String password, UserType userType) {
        super(username, password, userType);
    }
    /**
     * Constructor for creating a new Student with the specified username,
password, faculty, and degree.
     *
     * @param username The username of the student.
     * @param password The password of the student.
     * @param faculty  The faculty to which the student belongs.
     * @param degree   The degree level of the student.
     */
    public Student(String username, String password, Faculty faculty,
Degree degree) {
        super(username, password, UserType.STUDENT);
        DB.getInstance();
        this.id = DB.users.get(UserType.STUDENT).size() +1;
```

```
        this.faculty = faculty;
        this.degree = degree;

    }

    public double getGPA() {
        return 0.0;
    }//null

    protected List<Course> getCoursesRegistered(){
        return coursesRegistered;
    }


    // MENU METHODS
    @Override
    public void update() {

    }

    public void viewTranscript(){
        for (Course c : coursesRegistered){
            System.out.println(c.getStudentMark(this));
        }
    }

    public void viewInfoAbTeacher(){
        int i = 1;
        System.out.println("Choose course which teachers you are interested
in");
        for (Course c : DB.instance.getCourses()){
            System.out.println(i + " " + c.getTitle());
        }
        int choice = in.nextInt();
        in.nextLine();

        for(Teacher t : DB.instance.getCourses().get(i-
1).getInstructors()){
            System.out.println(t.toString());
        }
    }

    public void viewMarks(){
        Course c = new Course();
        System.out.println(c.getStudentMark(this));
    }

    public void viewCourses(){
        for (Course element : this.coursesRegistered) {
            System.out.println(element.toString());
        }
    }

    public void rateTeacher(){
        System.out.println("Choose teacher");
        int i = 1;

        Vector<Teacher> studentsTeachers = new Vector<>();
        for (Course element : this.coursesRegistered) {
            for (Teacher t:element.getTeachers(this)){
                studentsTeachers.add(t);
```

```
                System.out.println(i + " " + t.getUsername());
            }
        }
        int choice = in.nextInt();
        in.nextLine();
        System.out.println(studentsTeachers.elementAt(choice) + " is
chosen, give rating");
        int rating = in.nextInt();
        in.nextLine();

        studentsTeachers.elementAt(choice).setRating(rating);
        System.out.println(studentsTeachers.elementAt(choice).getRating());
    }

    public void getTranscript(){
        for (Course c : coursesRegistered){
            System.out.println(c.getStudentMark(this));
        }
    }

    protected void studentOrganizations(){
        for(StudentOrganization sd: DB.getInstance().getOrganizations()){
            System.out.println(sd.toString());
        }
    }

    // Other methods and menu-related functionality...

    /**
     * Checks if a student is registered for a given course based on its
prerequisites.
     *
     * @param c The course to check for registration.
     * @return True if the student meets all prerequisites; false
otherwise.
     */
    protected boolean isIn(Course c) {
        if (c.getPrerequisites() == null) {
            return true; // No prerequisites, so it's always considered as
"in"
        }

        for (Course prereq : c.getPrerequisites()) {
            boolean prereqMet = false;

            for (Course studC : this.coursesRegistered) {
                if (prereq.equals(studC)) {
                    prereqMet = true;
                    break; // Found a match for this prerequisite, no need
to continue searching
                }
            }

            if (!prereqMet) {
                return false; // At least one prerequisite is not in
coursesRegistered
            }
        }

        return true; // All prerequisites are in coursesRegistered
    }
```

```
    protected boolean isNotRegistered(Course c){
        for(Course element : this.coursesRegistered){
            if (element==c){
                return false;
            }
        }

        return true;
    }

    protected void registerCourse(){
        //get set of available courses by name
        Vector<Course> coursesAvailable = new Vector<Course>();

        for (Course ele : DB.getInstance().getCourses()){
            if (isIn(ele) && isNotRegistered(ele)){
                coursesAvailable.add(ele);
            }
        }
        //student enters indeces of course
        if (!coursesAvailable.isEmpty()){
            int i = 1;
            System.out.println(i + "Enter your choice by int or 0 to go
back");
            for (Course avCourse : coursesAvailable){
                System.out.println(i +" "+ avCourse.getTitle());
            }

            int sumcredits = 0;
            while (true){
                int choice = in.nextInt();
                in.nextLine();
                if (choice == 0){
                    break;
                }
                if(coursesAvailable.elementAt(choice-1).getCredits() +
sumcredits < 21 ){
                    sumcredits = sumcredits +
coursesAvailable.elementAt(choice-1).getCredits();

coursesRegistered.add(coursesAvailable.elementAt(choice-1));
                    System.out.println(coursesAvailable.elementAt(choice-
1).getTitle()+" is added succesfully");
                }
                else {
                    System.out.println("Sum of credits would be more than
21");
                }
            }
        }
        else{
            System.out.println("No available courses");
        }
        //check if credit <21
    }

    public void run() throws IOException {
        try {
            getWelcomeMessage();
            menu:
```

```
            while (true) {
                displayMenu();
                int choice = in.nextInt();
                in.nextLine();
                switch (choice) {
                    case 1:
                        viewCourses();
                        break;
                    case 2:
                        viewInfoAbTeacher();
                        break;
                    case 3:
                        viewMarks();
                        break;

                    case 4:
                        viewTranscript();
                        break;
                    case 5:
                        rateTeacher();
                        break;
                    case 6:
                        getTranscript();
                        break;
                    case 7:
                        studentOrganizations();
                    case 8:
                        changeLanguage();
                        break;
                    case 9:
                        registerCourse();
                        break;
                    case 0:
                        exit();
                        break menu;
                    default:
                        throw new IllegalStateException("Unexpected value:
" + choice);
                }

            }
        } catch (Exception e) {
            handleError(e);
        }
    }

    protected void changeLanguage(){
        System.out.println("Choose language:\n" +
                "1. ENG\n" +
                "2. RUS\n" +
                "3. KAZ");

        System.out.print("Enter your choice: ");
        int choice = in.nextInt();
        in.nextLine(); // Consume the newline left-over

        switch (choice){
            case 1:
                displayEnglishMenu();
                break;
            case 2:
```

```
                displayRussianMenu();
                break;
            case 3:
                displayKazakhMenu();
                break;
        }
    }

    // Other methods and menu-related functionality...

    /**
     * Displays the menu in English for the student user.
     */
    @Override
    protected void displayEnglishMenu() {
        System.out.println("User student:\n" +
                "1. View Courses\n" +
                "2. View information about teacher of a specific course\n"
+
                "3. View marks\n" +
                "4. View Transcript\n" +
                "5. Rate teacher on scale 1-10 (first enter id)\n" +
                "6. Get Transcript\n" +
                "7. Student organizations\n" +
                "8. Change language"+
                "9. Register for courses"+
                "0. Back to Main Menu");


    }
    @Override
    protected void displayRussianMenu() {
        System.out.println("User student:\n" +
                "1. Просмотр курсы\n" +
                "2. Просмотр информацию про педагога определенного
предмета\n" +
                "3. Просмотр оценки\n" +
                "4. Просмотр Транскрипта\n" +
                "5. Оценить педагога\n" +
                "6. Получить транскрипт\n" +
                "7. Студенческие организации\n" +
                "0. Вернуться в Главное Меню");

    }

    @Override
    protected void displayKazakhMenu() {
        System.out.println("User student:\n" +
                "1. Курстарды көру\n" +
                "2. Белгілі бір пәннің мұғалімі туралы ақпаратты көру\n" +
                "3. Бағаны көру\n" +
                "4. Транскрипты көру\n" +
                "5. Мұғалімді бағалау\n" +
                "6. Транскриптты алу\n" +
                "7. Студенттік организациялар\n" +
                "0. Негізгі мәзірге қайту");

    }

    // Other methods and menu-related functionality...
```

```
    /**
     * Returns a string representation of the Student object.
     *
     * @return A string representation of the Student object.
     */
    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", faculty=" + faculty +
                ", yearOfStudy=" + yearOfStudy +
                ", coursesRegistered=" + coursesRegistered +
                ", degree=" + degree +
                ", studentOrganization=" + studentOrganization +
                ", credits=" + credits +
                '}';
    }

    public Faculty getFaculty() {
        return faculty;
    }
}
```

The work process

In our effort to streamline collaboration, we transitioned to Telegram for real-time communication. Simultaneously, we established a GitHub repository for centralized version control, facilitating seamless collaboration and maintaining a clear history of changes. The project was organized into packages to enhance code readability and maintainability, enabling focused development. Regular coordination meetings, whether virtual or in-person, ensured alignment on progress and goals. Occasional on-site university gatherings strengthened team bonds and fostered a collaborative atmosphere. These changes significantly optimized our work process, improving productivity and team cohesion.

Used platforms

Communication: Telegram

Programming Language: Java

Version Control System: GitHub

Development Environment: IntelliJ

UML Diagrams: Genmymodel

Problems & Solutions

1. Git Challenges in Eclipse. The use of Git in Eclipse initially posed some challenges. Despite the numerous tools provided by Eclipse for Git integration, many team members faced difficulties in navigation and performing basic operations. This impacted the efficiency of teamwork and version control management.

Solution: Transition to IntelliJ IDEA. To enhance the Git experience, we decided to transition to IntelliJ IDEA. This integrated development environment offers a more intuitive and user-friendly interface for version control systems, including Git. The switch to IntelliJ improved team efficiency and streamlined the version control process.

2. Database and Serialization Issues. Another set of challenges arose in dealing with databases and serialization. Retrieving specific data required traversing through extensive datasets, leading to inefficiencies in data access and processing.

Solution: Streamlining Data Access. To address these challenges, we implemented measures to streamline data access. This involved optimizing database queries and adopting more efficient serialization techniques. These improvements not only enhanced the speed of data retrieval but also contributed to overall system performance.

In summary, our transition to IntelliJ IDEA for Git management and the implementation of optimized data access strategies significantly improved our development process, fostering a more efficient and streamlined workflow.

Conclusion

As a team, we learned effective collaboration. Bereket excelled as a team lead, skillfully coordinating efforts. Danial mastered Git and IntelliJ, Islam honed negotiation skills and GitHub usage, Askar became adept at version control systems, contributing significantly to coordination and tackling complex code. While our work isn't flawless, we're satisfied with the outcome, having nearly accomplished all our intended goals.