2nd Deliverable by Group 5

On

Book Reservation System (library)

Submitted to:

Prof. Eng. Wahab Hamou-Lhadj

COEN6312 – Model Driven Software Engineering

Submitted By:

Raynard Omongbale – 40150036

Alireza Sameni – 40151521

Issam Sedki – 40171941

Heyu Guan – 40113703

Ying Sun – 40115153

Yu Song – 40114330

March 05, 2021

CONTENTS

LIST OF FIGURES

1. The Revised Class Diagram

The following class diagram was generated using Papyrus.

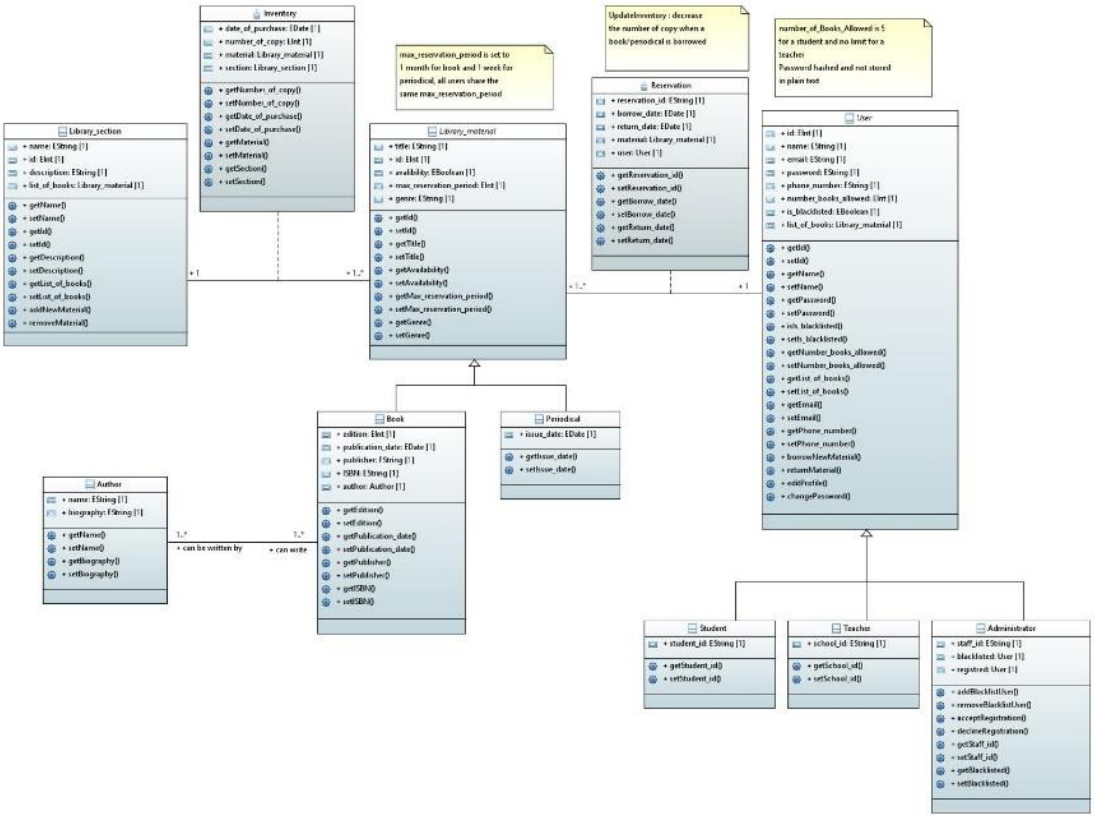


Figure 1 Revised Class Diagram of Book Reservation System

2. The OCL Constraints

2.1 Context User

The OCL constraints added to this section include the user's personal information and registration: user password, ID, and email constraints; constraints on the number of users borrowed; and constraints on the blacklist management section. The UML diagram of the user section is made more accurate.

1) Before making an appointment, users need to have their own unique ID and email for authentication.

Context User

Inv:

allInstances()->forAll(u1, u2:User|u1<>u2 implies u1.id<>u2.id AND u1.email<>u2.email)

2) If a user is blacklisted, then that user can no longer reserve books.

Context User

Inv:

self.Library_material->notEmpty implies not self.is_blacklisted

3) To prevent users from borrowing too many books and affecting other users borrowing materials, each user cannot borrow more than the allowed number of books.

Context User

Inv:

self.Library_material->size()<=self. number_of_books_allowed

4) Calling the addBlacklistUser() method of the Administrator class, makes a user blacklisted.

Context Administrator::addBlacklistUser(u : user): void

pre: u.is_blacklisted = false

post: u.is_blacklisted = true

5) Calling the removeBlacklistUser() method of the Administrator class, removes a blacklisted user out of being backlisted.

Context Administrator::removeBlacklistUser (u : user): void

pre: u.is_blacklisted = true

post: u.is_blacklisted = false

6) The user's password should be at least 8 characters long.

   Context User

   Inv:

   allInstances()->forAll(u : User | u.password.size() >= 8)

7) Calling the borrowNewMaterial() method of the class User, borrows a new library material for the User. The material is only borrowed if it does not exist on the User's list_of_books and if it is available. After the user borrows the material, it must be unavailable.

   Context User:: borrowNewMaterial (lm1 : Library_material): void

   pre: self.list_of _books ->excludes(lm1) AND lm1.availability=true

   post: self. list_of _books ->includes(lm1) AND lm1.availability=false

8) Calling the returnMaterial() method of the class User returns a library material for the User. The library material is only returned if it exists on the User's list_of_books. Initially the material must be unavailable but after the User returns the material, it must be available again

   Context User:: returnMaterial (lm1 : Library_material): void

   pre: self.list_of _books ->includes(lm1) AND lm1.availability=false

   post: self.list_of _books ->excludes(lm1) AND lm1.availability=true

2.2 Context Library material

The OCL constraints added in this section are about the library material section, and the constraints already given in the UML diagram will not be written out in this section. The constraints added are mainly about the reservation rules for library materials and materials management.

1) One Library_material should not be in more than one section.

   Context Library_material

   Inv:

   self.Library_section-> size() <=1

2) A library material can only be reserved by one person at a time.

   Context Library_material

   Inv:

   self.User-> size() <=1

3) To facilitate the management of books, each library material should have its own ID.

   Context Library_material

   Inv:

   allInstances()->forAll (lm1,lm2:Library material|lm1<>lm2 implies lm1.id<>lm2.id)

4) Maximum reservation period is 30 days for all of the books and 7 days for all of the periodicals.

   Context Book

   Inv:

   allInstances()->forAll (b:Book | b.max_reservation_period = 30)

   Context Periodical

   Inv:

   allInstances()->forAll (p:Periodical | p.max_reservation_period = 7)

2.3 Context Library_section

The OCL constraint added in this section is the part about library section. It is mainly about the operational constraints for managing library section.

1) Each library section must have a unique id.

   Context Library_section

   Inv:

   allInstances()->forAll (ls1, ls2:Library_section|ls1<>ls2 implies ls1.id<>ls2.id)

2) Description for each library section should be at least 20 characters but no mare than 200 characters.

   Context Library_section

   Inv:

   allInstances()->forAll(ls:Library_section|ls.description.size() >=20 AND ls.description.size()<=200)

3) Calling the addNewMaterial() method of the class Library_section adds a new library material to the library section. The library material is only added if it does not already exist in the section.

   Context Library_section:: addNewMaterial (lm1: Library_material): void

   pre: self.list_of_books->excludes(lm1)

   post: self.list_of_books->includes(lm1)

4) Calling the removeMaterial() method of the class Library_section removes a library material from the library section. The library material is only removed if it already exists in the section.

   Context Library_section:: removeMaterial (lm1: Library_material): void

pre: self.list_of_books-> includes(lm1)

post: self.list_of_books-> excludes(lm1)

3. Description of the system

This section is about the description of the system programmed in java implements functions, behaviors and structures the model from the UML diagram as shown below.

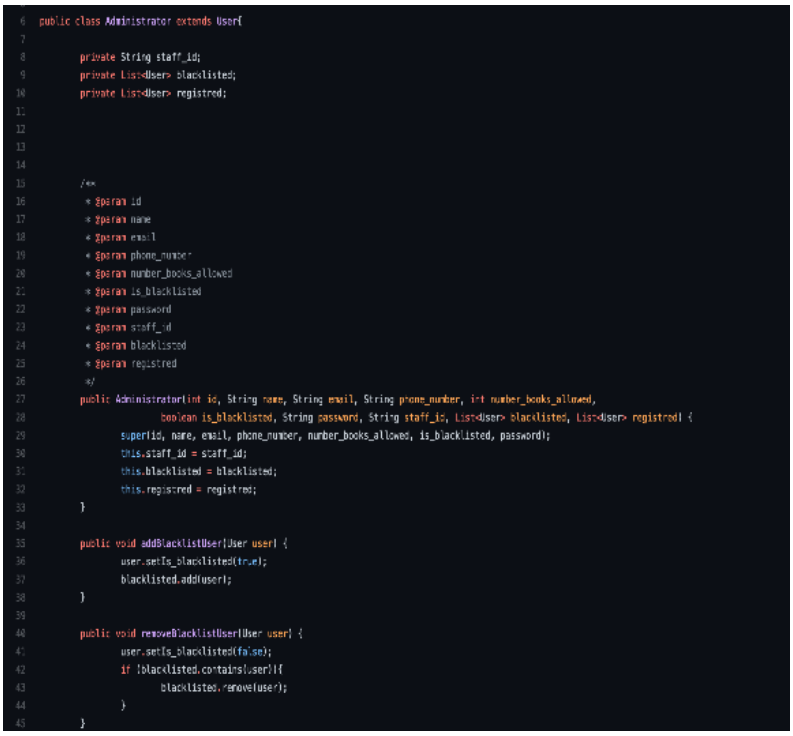
3.1 The User Class:

The user class with is an abstract of the user object, which is the base class for every person object created in the system. It consists of multiple attributed such as name, email, number_books_allowed phone_number, password, list_of_books and member functions/ behavior such as a constructor to initialize the data members and a few others as listed below.

```
public abstract class User {

	/**
	 * @param id
	 * @param name
	 * @param email
	 * @param phone_number
	 * @param number_books_allowed
	 * @param is_blacklisted
	 * @param password
	 * @param list_of_books
	 */
	public User(int id, String name, String email, String phone_number, int number_books_allowe
			boolean is_blacklisted, String password) {
		super();
		this.id = id;
		this.name = name;
		this.email = email;
		this.phone_number = phone_number;
		this.number_books_allowed = number_books_allowed;
		this.is_blacklisted = is_blacklisted;
		this.password = password;
		this.list_of_books = new ArrayList<Library_material>();

	}

	@Override
	public String toString() {
		return " [name=" + name + ", email=" + email + ", phone_number=" + phone_number
				+ "]";
	}

	/**
```

Figure 2 User class java implementation

The derived classes which inherit the user class includes the Student, Teacher and Administrator class. This classes have independent constructors since classes can't inherit their base class constructors and destructors.



```
public class Administrator extends User{

        private String staff_id;
        private List<User> blacklisted;
        private List<User> registred;




        /**
         * @param id
         * @param name
         * @param email
         * @param phone_number
         * @param number_books_allowed
         * @param is_blacklisted
         * @param password
         * @param staff_id
         * @param blacklisted
         * @param registred
         */
        public Administrator(int id, String name, String email, String phone_number, int number_books_allowed,
                        boolean is_blacklisted, String password, String staff_id, List<User> blacklisted, List<User> registred) {
                super(id, name, email, phone_number, number_books_allowed, is_blacklisted, password);
                this.staff_id = staff_id;
                this.blacklisted = blacklisted;
                this.registred = registred;
        }

        public void addBlacklistUser(User user) {
                user.setIs_blacklisted(true);
                blacklisted.add(user);
        }

        public void removeBlacklistUser(User user) {
                user.setIs_blacklisted(false);
                if !blacklisted.contains(user)){
                        blacklisted.remove(user);
                }
        }
```

Figure 3 Administrator class extension of the user class java implementation

```
public class Student extends User {
        private String student_id;

        /**
         * @return the student_id
         */
        public String getStudent_id() {
                return student_id;
        }

        /**
         * @param student_id the student_id to set
         */
        public void setStudent_id(String student_id) {
                this.student_id = student_id;
        }

        /**
         * @param id
         * @param name
         * @param email
         * @param phone_number
         * @param number_books_allowed
         * @param is_blacklisted
         * @param password
         * @param list_of_books
         * @param student_id
         */
        public Student(int id, String name, String email, String
                        boolean is_blacklisted, String password,
                super(id, name, email, phone_number, number_books_
                this.student_id = student_id;
        }
```

Figure 4 Student class extension of the user class java implementation

```java
public class Teacher extends User {
	private String school_id;

	/**
	 * @return the student_id
	 */
	public String getSchool_id() {
		return school_id;
	}

	/**
	 * @param student_id the student_id to set
	 */
	public void setStudent_id(String school_id) {
		this.school_id = school_id;
	}

	/**
	 * @param id
	 * @param name
	 * @param email
	 * @param phone_number
	 * @param number_books_allowed
	 * @param is_blacklisted
	 * @param password
	 * @param list_of_books
	 * @param school_id
	 */
	public Teacher(int id, String name, String email, String phone_number, int number_books_allowed,
			boolean is_blacklisted, String password,  String school_id) {
		super(id, name, email, phone_number, number_books_allowed, is_blacklisted, password);
		this.school_id = school_id;
	}
```

Figure 5 Teacher class extension of the user class java implementation

3.2 The Reservation Class:

The reservation association class represents the relationship between a library material and a user object. This association multiplicity between a library material and user class and vice versa is in a “one to many” and “one to “respectively (1 user can have one or many library material” and a material can be reserved by one user).

The reservation class holds attributes that cannot logically be induced in either the user or the library material class, these attributes include reservation id, borrow date, return date.

```java
import java.util.Date;

public class Reservation {
	private String reservation_id;
	private Date borrow_date;
	private Date return_date;
	public Library_material material;
	public User user;
	/**
	 * @return the reservation_id
	 */
	public String getReservation_id() {
		return reservation_id;
	}
	/**
	 * @param reservation_id the reservation_id to set
	 */
	public void setReservation_id(String reservation_id) {
		this.reservation_id = reservation_id;
	}
	/**
	 * @return the borrow_date
	 */
	public Date getBorrow_date() {
		return borrow_date;
	}
	/**
	 * @param borrow_date the borrow_date to set
	 */
	public void setBorrow_date(Date borrow_date) {
```

Figure 6 Reservation association class java implementation

3.3 The Library Material

The library material class which is an abstract of the library materials object, which is the base class for every type of book object created in the system. It consists of multiple attributed such as title, id, availability, max_reservation_period, password, list_of_books and member functions/ behavior such as a constructor to initialize the data members and a few others as such as “getId()”, “getAvailabilty()”, “getMax_reservation_period()”. The library material class is inherited by two other classes.

```java
public abstract class Library_material {
	public int id;
	public String title;
	public Boolean availability;
	private int max_reservation_period;
	private String genre;
	//library_section should be removed from Libray_material


	/**
	 * @return the id
	 */
	public int getId() {
		return id;
	}
	/**
	 * @param id
	 * @param title
	 * @param availability
	 * @param max_reservation_period
	 * @param genre
	 */
	public Library_material(int id, String title, Boolean availability, int max_reservation_period, String genre) {
		super();
		this.id = id;
		this.title = title;
		this.availability = availability;
		this.max_reservation_period = max_reservation_period;
		this.genre = genre;
	}
```

Figure 7 Library_material class java implementation

3.4 The Periodical Class

A periodical is a type of library material that is released weekly or biweekly, The periodical class inherits the library material base class, they consist of an issuer date attribute.

```java
public class Periodical extends Library_material {

	private Date issue_date;

	/**
	 * @return the issue_date
	 */
	public Date getIssue_date() {
		return issue_date;
	}

	/**
	 * @param issue_date the issue_date to set
	 */
	public void setIssue_date(Date issue_date) {
		this.issue_date = issue_date;
	}

	public Periodical(int id, String title, Boolean availability, int max_reservation_period, String genre, Date issue_date) {
		super(id, title, availability, max_reservation_period, genre);
		this.issue_date = issue_date;
	}

}
```

Figure 8 Periodical class extension of the libarary_material class java implementation

3.5 The Book Class

The book class inherits the library material base class, and share a multiplicity associate of “1..*” and “1..*” between the Author class which means an author “can write” one or more books while a book “can be written” by one or more author.

```
public class Book extends Library_material {
        public int edition;
        public Date publication_date;
        public String publisher;
        public String ISBN;
        public Author author;

        /**
         * @param id
         * @param title
         * @param availability
         * @param max_reservation_period
         * @param genre
         * @param edition
         * @param publication_date
         * @param publisher
         * @param iSBN
         */
        public Book(int id, String title, Boolean availability, int max_reservation_period, String genre, int edition,
                        Date publication_date, String publisher, String iSBN, Author author) {
                super(id, title, availability, max_reservation_period, genre);
                this.edition = edition;
                this.publication_date = publication_date;
                this.publisher = publisher;
                this.ISBN = iSBN;
                this.author = author;
        }
        @Override
        public String toString() {
                return "[name=" + super.title + ", publication_date=" + publication_date + ", publisher=" + publisher
                                + "]";
        }
        /**
         * @return the edition
         */
        public int getEdition() {
                return edition;
```

Figure 9 Book class extension of the libarary_material class java implementation

3.6 The Library Section Class

The library section stands as a storage for library material. An associate class (inventory class) hold the relationship between every library material and the library section, The library Section class consists of multiple attributed such as name, id, description, list of book and member functions/ behavior such as a constructor to initialize the data members and a few others as such as “getName()”, “setName()”, “getID()” etc.

```
public class Library_section {
        private String name;
        private int id;
        private String description;
        private List<Library_material> list_of_books;


        /**
         * @param name
         * @param id
         * @param description
         * @param list_of_books
         */
        public Library_section(String name, int id, String description) {
                super();
                this.name = name;
                this.id = id;
                this.description = description;
                this.list_of_books = new ArrayList<Library_material>();
        }
        /**
         * @return the name
         */
        public String getName() {
                return name;
        }
        /**
         * @param name the name to set
         */
        public void setName(String name) {
                this.name = name;
        }
        /**
         * @return the id
         */
        public int getId() {
                return id;
        }
        /**
```

Figure 10 Libarary_section class java implementation

3.7 The Inventory Class

The inventory associate class hold the relationship between every book and how they are stored in the library (sections).

This association multiplicity between the library section and book class and vice versa is in a “one to one” and “one to “respectively (1 library material can exist only in a library section and a library section can have one or more library material).

The inventory class holds attributes that cannot logically be induced in either the library section class or the library material class, these attributes include date of purchase, numbers of copies.

```
public class Inventory {

public Date date_of_purchase;
public int number_of_copy;
public Library_material material;
public Library_section section;

/**
        * @param date_of_purchase
        * @param number_of_copy
        * @param material
        * @param section
        */
public Inventory(Date date_of_purchase, int number_of_copy, Library_material material, Library_section section) {
super();
this.date_of_purchase = date_of_purchase;
this.number_of_copy = number_of_copy;
this.material = material;
this.section = section;
        }

/**
        * @return the number_of_copy
        */
public int getNumber_of_copy() {
return number_of_copy;
        }

/**
        * @param number_of_copy the number_of_copy to set
        */
public void setNumber_of_copy(int number_of_copy) {
this.number_of_copy = number_of_copy;
        }

/**
        * @return the date_of_purchase
```

Figure 11 Inventory association class java implementation

4. Driver test:

The test we performed is supposed to prove that the implementation of the UML diagram is working overall.

The test is composed of the following instances:

- 2 users (class student)
- 1 author
- 1 book
- 1 periodical
- 2 library sections

First the library material are added to the appropriate section through the inventory association.

This association is keeping track of the date where these resources have been added to the section.

Once the books are in the section within the library, the students can make the reservation (through the association).

The test checks that when a student is blacklisted, he/she can not make any reservation.

```
public static void main(String[] args) {

    /**  creation of a new student **/
    //the two student are not blacklisted initially
    User alireza = new Student(1,"Alireza","alireza@gmail.com","5147835678", 3, false, "xxxx12345@", "encs1");
    User issam = new Student(2,"issam","issam@gmail.com","51478357654", 3, false, "xxxx6789!", "encs2");

    /** creation of an author named John Doe **/
    Author john_doe = new Author("JOhn Doe", "Great science author living in Canada with his wife and 5 kids");

    /** creation of a book **/
    //this book can be borrowed for up to 7 days
    //genre = science
    //edition = 3
    //publisher = Gilbert Edition
    //publication date = March 2020
    //ISBN = IS123456
    //Author = John Doe
    //(int id, String title, Boolean availability, int max_reservation_period, String genre, int edition,
    //Date publication_date, String publisher, String iSBN, Author author)
    Library_material book_science_grade3 = new Book(1,"book of science for grade 3",true,7,"science",3,new Date("01/03/2020"),"Gilbert Edition","IS123456", john_d


    /**  creation of a periodical instance **/
    //this book can be borrowed for up to 10 days
    //genre = nature
    //publication date = 15/02/2021

    Library_material magazine_of_nature = new Periodical(1,"Magazine of Nature and Wildlife",true, 10, "nature", new Date("15/02/2020"));

    /** creation of two library sections **/
    Library_section science = new Library_section("science",1,"section for sceince books and periodicals");
    Library_section nature = new Library_section("nature",1,"section for nature and wildlife books and periodicals");

    /** adding the material to the appropriate section
     * book material where bought and added the same date to different library sections
     */

    Inventory inv1= new Inventory(new Date("28/01/2021"),1,book_science_grade3,science);
    science.addNewMaterial(book_science_grade3);
    Inventory inv2= new Inventory(new Date("28/01/2021"),1,magazine_of_nature,nature);
    nature.addNewMaterial(magazine_of_nature);

    /** a student is reserving a book **/

    Reservation resa1= new Reservation ("1", new Date("28/02/2021"), new Date("05/03/2021"),book_science_grade3,issam);
    issam.borrowNewMaterial(book_science_grade3);
    System.out.println("display of books borrowed by"+issam);
    System.out.println(issam.getList_of_books());
    issam.returnMaterial(book_science_grade3);
    System.out.println("display of books borrowed by"+issam);
    System.out.println(issam.getList_of_books());

    //we are performing the same test for alireza who is blacklisted
    //the outcome should be that he can not borrow any material
    alireza.setIs_blacklisted(true);
    alireza.borrowNewMaterial(magazine_of_nature);
    System.out.println("display of books borrowed by"+alireza);
    System.out.println(alireza.getList_of_books());
```

Figure 12 Drivers for testing

Versioning

Github repository: https://github.com/raynard2/6312_LibraryProject.git

Reference

[1] Ian Sommerville, Software Engineering, 10th ed., Pearson, March 24 2015.

[2] Roger S Pressman and Dr. Bruce R. Maxim, Software Engineering: A Practitioner's Approach, 8th Ed., Mcgraw-Hill Education, Jan 23 2014.

[3] Ian Gorton, Essential Software Architecture, 2nd Ed., May 17 2014.

[4] OMG Unified Modeling Language specification V2.5, June 2015