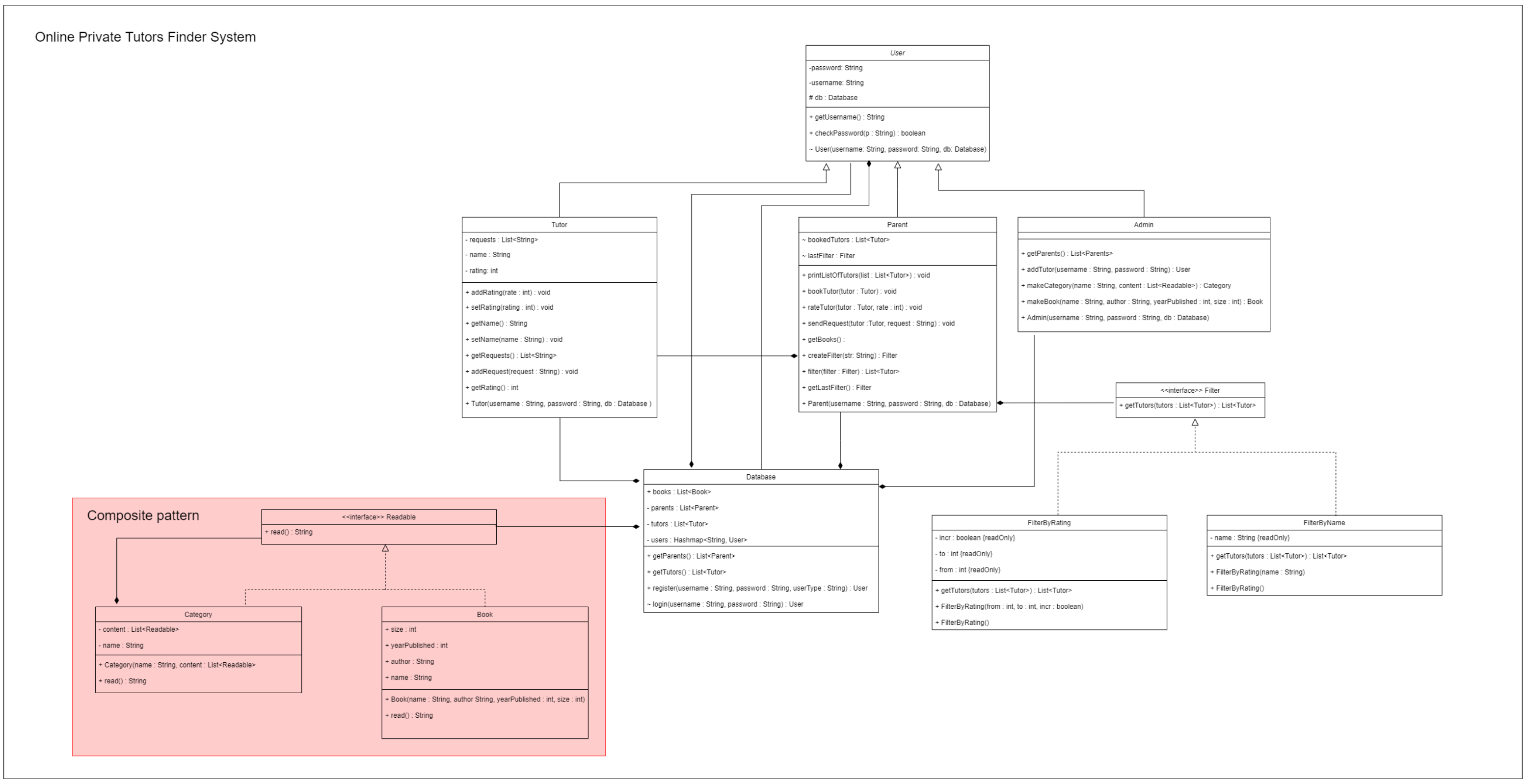
README.md

Online Private Tutors Finder System

UML diagram of the project


Online Private Tutors Finder System
User
-password: String
-username: String
# db : Database
+ getUsername() : String
+ checkPassword(p : String) : boolean
~ User(username: String, password: String, db: Database)
Tutor
- requests : List<String>
- name : String
- rating: int
+ addRating(rate : int) : void
+ setRating(rating : int) : void
+ getName() : String
+ setName(name : String) : void
+ getRequests() : List<String>
+ addRequest(request : String) : void
+ getRating() : int
+ Tutor(username : String, password : String, db : Database)
Parent
~ bookedTutors : List<Tutor>
~ lastFilter : Filter
+ printListOfTutors(list : List<Tutor>) : void
+ bookTutor(tutor : Tutor) : void
+ rateTutor(tutor : Tutor, rate : int) : void
+ sendRequest(tutor :Tutor, request : String) : void
+ getBooks() :
+ createFilter(str: String) : Filter
+ filter(filter : Filter) : List<Tutor>
+ getLastFilter() : Filter
+ Parent(username : String, password : String, db : Database)
Admin
+ getParents() : List<Parents>
+ addTutor(username : String, password : String) : User
+ makeCategory(name : String, content : List<Readable>) : Category
+ makeBook(name : String, author : String, yearPublished : int, size : int) : Book
+ Admin(username : String, password : String, db : Database)
<<interface>> Filter
+ getTutors(tutors : List<Tutor>) : List<Tutor>
Database
+ books : List<Book>
- parents : List<Parent>
- tutors : List<Tutor>
- users : Hashmap<String, User>
+ getParents() : List<Parent>
+ getTutors() : List<Tutor>
+ register(username : String, password : String, userType : String) : User
~ login(username : String, password : String) : User
FilterByRating
- incr : boolean {readOnly}
- to : int {readOnly}
- from : int {readOnly}
+ getTutors(tutors : List<Tutor>) : List<Tutor>
+ FilterByRating(from : int, to : int, incr : boolean)
+ FilterByRating()
FilterByName
- name : String {readOnly}
+ getTutors(tutors : List<Tutor>) : List<Tutor>
+ FilterByRating(name : String)
+ FilterByRating()
Composite pattern
<<interface>> Readable
+ read() : String
Category
- content : List<Readable>
- name : String
+ Category(name : String, content : List<Readable>
+ read() : String
Book
+ size : int
+ yearPublished : int
+ author : String
+ name : String
+ Book(name : String, author String, yearPublished : int, size : int)
+ read() : String

Report

Description of the work done

Our team implemented **books** and **categories** in the system. The user can read a **book** or **category** of books. A **category** can be read by reading all the books contained in the category. Also, **category** can have **nested categories**, and **nested categories** should be read when the parent **category** read as well.

WHAT pattern did we choose to implement the feature?

Out of 5 patterns given:

- Adapter
- Bridge
- Composite
- Decorator
- Flyweight

we chose **COMPOSITE** pattern for implementing our feature.

WHY did we choose composite pattern?

We should have a common interface for both books and categories. Composite pattern satisfies our need because we can treat both book and category as one object of type `Readable` which allows us to read the object. Even if the category has nested books and categories, the composite pattern provides us with the possibility to read the category as it was one object.

HOW did we implement the feature?

The UML diagram for the composite pattern is provided on the general UML diagram of the project and highlighted with red color.

We implemented 3 classes for our feature: `Readable` , `Category` , and `Book` .

Readable

`Readable` is an interface with one method - `read()` , this method returns all the titles in the readable object as a String. The class is located in `src/book/Readable.java` .

Category

`Category` implements `Readable` interface, so it has a method `read()` , also it has a name and collection `content` of readable objects. When `read()` is called, it inserts the name of the category in the returned String and the result of read() for each object in the `content` collection. The class is located in `src/book/Category.java` .

Source code for read() method in the Category class

```
public String read() {
        StringBuilder result = new StringBuilder();
        for (Readable r : content) {
            result.append(r.read()).append(",\n");
        }
        return "(" + name + ":\n" + result.toString() + ")";
    }
```

Book

`Book` also implements `Readable` interface, it has `read()` and fields containing information about the book. `read()` method of the `Book` class returns String with name and author of the book. The class is located in `src/book/Book.java` .

Source code for read() method in the Book class

```
public String read() {
        return name + " " + author;
    }
```

Feature usage

Readable objects stored in the `Database` class. Finally, the functionality for reading is used in Main.java. It adds books and outputs the result of the `read()` method in the console.

Part of the Main.java where the feature is used(lines 52-68)

```
//        testing of added COMPOSITE pattern
//        making books and categories
        Book book1 = admin.makeBook("The History of Science", "Merlin H.", 2013, 300);
        Book book2 = admin.makeBook("Design Patterns", "Erich Gamma", 1994, 395);
        Book book3 = admin.makeBook("Design Patterns (Java)", "Erich Gamma", 2003, 455);
        Category SSADCategory = admin.makeCategory("SSAD", Arrays.asList(book2, book3));
        Book book4 = admin.makeBook("book1", "author1", 2000, 200);
        db.books = new Category("all books", Arrays.asList(book1, SSADCategory, book4));

//        printing all books
        System.out.println("\n\tTesting get all books function with COMPOSITE pattern:\n");
        System.out.println(db.books.read());

//        printing books from a category
        System.out.println("\n\tTesting get books from a category function:\n");
        System.out.println(SSADCategory.read());
//        end of COMPOSITE pattern testing
```