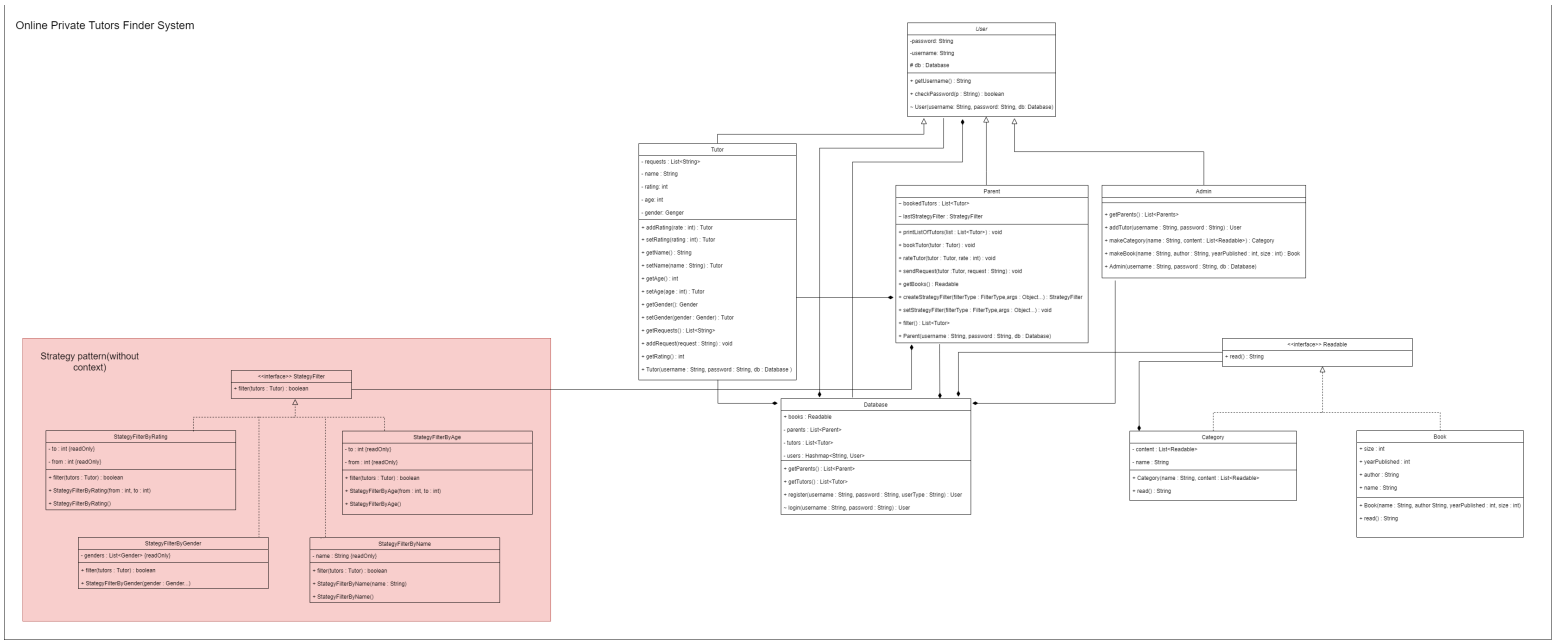
README.md

Online Private Tutors Finder System

UML diagram of the project



Report

Initial task

This private tutor system will help to find tuition teachers from nearby locations. Teachers can also get a student just by logging onto the website and setting up the profile. In the personal tutor finder system, there are three entities namely, Admin, Parents, and Tutor. Admin can login, manage tutor by adding new teachers and update their profiles. Admin can also manage E-books by adding new books to the library. Admin can also check for the registered parents. Admin will register tutors and credentials will be shared with tutors by Email. Parents can register and login, tutors can be viewed by parents. Parents can filter and select the tutor and after selecting parents will raise the request of the demo lecture. After attending the lecture, they can book the tutor online, rate the tutor and view the E-Books. The tutor can login by using credentials that will be provided by mail. They can check for the request for a demo lecture and accept the request. They can also check the booking done. They need to set their profile. This private tuition system can help the tutors to get students and parents to find the best tutors for their children.

Description of the work done

Our team implemented **filters** to help parents filter lists of tutors and find their best-fit for their purposes. **Filters** can filter tutors by such criteria as rating, age, gender, and name.

WHAT pattern did we choose to implement the feature?

Out of 10 patterns given:

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer

- State
- Strategy
- Template Method
- Visitor

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

WHY did we choose strategy pattern?

In case of filters, we should have variation of, basically, the same algorithm but with different filter criteria which could be switched in runtime. Strategy pattern solves the problem by providing a possibility to variate the behaviour of the filter object by substituting instances of filter classes.

HOW did we implement the feature?

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

We implemented 5 classes for our feature: `StrategyFilter`, `StrategyFilterByRating`, `StrategyFilterByAge`, `StrategyFilterByGender`, and `StrategyFilterByName`. Also, the `StrategyFilter lastStrategyFilter` field in `Parent` class was added. That field will be used by Parent's `filter()` method.

StrategyFilter

`StrategyFilter` is an interface with one method - `filter(tutor)`, this method returns boolean value(does certain tutor fit or not). The method checks `tutor` by condition defined in derived classes. The class is located in `src/filter/StrategyFilter.java`.

StrategyFilterByRating

`StrategyFilterByRating` implements `StrategyFilter` interface. It filters by rating in range [`from` ; `to`]. The class is located in `src/filter/StrategyFilterByRating.java`.

Source code for filter(tutor) method in the StrategyFilterByRating class

```
@Override
public boolean filter(Tutor tutor) {
        return from <= tutor.getRating() && tutor.getRating() <= to;
}
```

StrategyFilterByAge

`StrategyFilterByAge` implements `StrategyFilter` interface. It filters by age in range [`from` ; `to`]. The class is located in `src/filter/StrategyFilterByAge.java`.

Source code for filter(tutor) method in the StrategyFilterByAge class

```
@Override
public boolean filter(Tutor tutor) {
        return from <= tutor.getAge() && tutor.getAge() <= to;
}
```

StrategyFilterByGender

`StrategyFilterByGender` implements `StrategyFilter` interface. It filters by gender checking that tutor's gender is in the list of genders. The class is located in `src/filter/StrategyFilterByGender.java`.

Source code for filter(tutor) method in the StrategyFilterByGender class

```
@Override
public boolean filter(Tutor tutor) {
```

```
        return genders.contains(tutor.getGender());
}
```

StrategyFilterByName

`StrategyFilterByName` implements `StrategyFilter` interface. It filters by name *lexicographically*. The class is located in `src/filter/StrategyFilterByName.java` .

Source code for filter(tutor) method in the StrategyFilterByName class

```
@Override
public boolean filter(Tutor tutor) {
        return name.compareToIgnoreCase(tutor.getName()) <= 0 &&
                (name + "zzz").compareToIgnoreCase(tutor.getName()) > 0;
}
```

filter() in Parent class

This method filters tutors with `StrategyFilter lastStrategyFilter` which is set by `setStrategyFilter(...)` .

Source code for filter() method in the Parent class

```
// Uses filter method for all tutors according to the chosen STRATEGY
    public List<Tutor> filter() {
        List<Tutor> tutors = db.getTutors();
        List<Tutor> result = new ArrayList<>();
        for (var t : tutors) {
            if (lastStrategyFilter.filter(t))
                result.add(t);
        }
        return result;
    }
```

Feature usage

The `Parent` class has `setStrategyFilter(...)` method which allows parents to set their own filter by criterion. Then, Parent's `filter()` method uses field `lastStrategyFilter` to filter list of tutors. The test cases are provided in `src/Main.java`

Part of the Main.java where the feature is used(lines 30-45)

```
System.out.println("\n\tTesting StrategyFilterByAge: default from 0 to 100:");
par.setStrategyFilter(FilterType.AGE);
List<Tutor> res = par.filter();
par.printListOfTutors(res);

System.out.println("\n\tTesting StrategyFilterByRating: from 5 to 7");
par.setStrategyFilter(FilterType.RATING, 5, 7);
par.printListOfTutors(par.filter());

System.out.println("\n\tTesting StrategyFilterByName: starts with \"bo\"");
par.setStrategyFilter(FilterType.NAME, "bo");
par.printListOfTutors(par.filter());

System.out.println("\n\tTesting StrategyFilterByGender: trans or female");
par.setStrategyFilter(FilterType.GENDER, Gender.TRANS, Gender.FEMALE);
par.printListOfTutors(par.filter());
```