Crossreview of team5’s project

by team2

1 Design

1.1 Violation of MVC pattern

The MVC pattern is followed pretty well; there are controllers, services, models, forms, etc. There are however some problems with the responsibilities being not so clearly assigned, but we will get into that later.

There was one way you could have used the framework more efficiently: Instead of checking wrong user input in the *SampleServiceImpl* with the use of if-clauses, you could check them in the Forms with help of *javax.validation.constraint (*see the spring guides).

1.2 Usage of helper objects between view and model

The usage of helper objects is generally appropriate. The logic is mostly assigned to the helper objects and not handled by the view/controllers.

However there are some methods in the *indexControler* which should better outsource some parts of their logic to a helper object.
For example:

- The type switch (in *getNotifications*)
- The save search (in *index*)
- The picture-handling (in *addUpdate*)
- The file creation (in *createAd*)

Furthermore almost all helper methods are in one class, which makes it difficult to get a clear overview.

1.3 Rich OO domain model

The implementation of OO design is there to some point but could be more distinct.

A example for good usage of OO design is the *Bookmark*. It has a *User* and an *Advert*. Which in turn has an *Address*, *User* and so on. But there are also two huge classes the *SampleServiceImpl* and the *IndexController*. This makes designing for low coupling/ high cohesion a bit difficult.

1.4 Clear responsibilities

Responsibilities are quite clear throughout the majority of the project. Except the two classes mentioned above (*SampleServiceImpl* and the *IndexController).*They are massively overloaded (which is in our humble opinion the only major problem in the whole system). The following methods from the *IndexController* could probably be assigned to specialized classes:

- *getNotifications*
- *setread*

- *adcreation*
- *addediting*
- *addUpdate*
- *showad*
- *createAd*

It could be wise to add an *AdController* and a *NotificationController*.

Concerning the *SampleServiceImpl* it would probably be best to just split up the methods into more specialised classes and rename/delete the *SampleServiceImpl*.

1.5 Sound invariants
We were unable to find any invariants in the project.

1.6 Overall code organization & reuse, e.g. views
We were only able to find one duplicated method: *loadUserByUsername(...)* (in *SampleServiceImpl* and *CustomUserDetailsService*)

2 Coding style

2.1 Consistency
The code overall is very consistent. There were some minor inconsistencies concerning the naming.

For Example all the files under *pojos* are of the format *****Form* except for *SignupUser* and sometimes there are names that do not comply with conventions (e.g. *setread* instead of *setRead*).

2.2 Intention-revealing names
Overall the naming is superb. They give a good idea over the function of a method/class/variable.

There are only two classes with unfortunate names. *SampleService* which should be split up into different services and *Enquiry* which probably just needs some extension like ‘NotificationEnquiry.java’.

2.3 Do not repeat yourself
As mentioned above there was one case of a duplicated method. Everything else looks fine.

2.4 Exception, testing null values
There are a lot of checks for null values, which is good.

However we found some try-catch clauses which didn’t look quite complete to us. Like the following in the *IndexController:*

```
try { … }
catch ( Exception e ) { }
```

2.5 Encapsulation

Encapsulation is looking good. Nice choice of visibility modifiers.

2.6 Assertion, contracts, invariant checks

We could not find any assertions apart from the ones found in tests.

There are no contracts specified in any class or method.

2.7 Utility methods

There are very few present at the moment, but refactoring the longer methods in *SampleService* will provide quite a few.

3 Documentation

3.1 Understandable

The documentation is sound! Almost every method is documented and most of the classes too.

3.2 Intention-revealing

The documentation explains the functionality quite well. There is very few redundant documentation and it is held to the minimum (no story telling). In most important cases even parameters, return and throwable exceptions are noted.

3.3 Describe responsibilities

Responsibilities are not described in any class or method.

They could be noted like in this example from another project (*miniSettlersOfCatan,* P2):

```
/**
 * Assign a structure to this Knot
 * @param _structure the structure to be placed, musst not be null
 */
public void setStructure ( Structure _structure ) {
        assert ( null != _structure ); ...
```

3.4 Match a consistent domain vocabulary

We could not find any major inconsistencies.

4 Testing

4.1 Clear and distinct test cases

There are quite a few test methods which check for five or more different values. A test should ideally check for one single value in order to allow faster location of an occurring problem.

4.2 Number/coverage of test cases

There are 9 different tests, each containing a bunch of test cases. Meaning you ranked well above the average programmer (and our project)! Very Nice! :D

4.3 Easy to understand the case that is tested

Due to the good choice of method names and the organized structure of the tests, it is easily understandable what is being tested.

4.4 Well crafted set of test data

The tests cover the essentials of the application.

4.5 Readability

Test methods are very distinctively named. In order to further improve readability consider the following helper methods.

Instead of writing:

```
@Before
public void init() {
        user = new User();
        user.setUsername(username);
        user.setPassword(password);
        user.setEmail(username);
        user = userDao.save(user);

        advert.setTitle(advertTitle);
        advert.setUser(user);
        advert = adDao.save(advert);

        bookmark = new Bookmark();
        bookmark.setAd(advert);
        bookmark.setUser(user);

        bookmark = bookmarkdao.save(bookmark);
}
```

Write it like this:

```
@Before
public void init() {
        this.initializeUser();
        this.initializeAdvert();
        this.initializeBookmark();
}
```

5 Code Analysis

5.1 IndexController.java

The IndexController has quite a few responsibilities. Actually it has a lot. According to the name, only mappings which correspond to the Index page, should be inside the indexController.
You should extract some methods into other controllers. This would mean to move methods like:

- getNotifications()
- setread()
- sendenquiry()
- adcreation()
- addUpdate()
- addediting()
- showad()
- createad()

The coding style in general is really good and very nice to read. Except for the methods which could use refactoring (mentioned earlier).