# RedisAdapter class

The class `nl.tudelft.sem.sem54.mainservice.redis.RedisAdapter` class forms the adapter between redis and our main microservice.

## Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO |
|---|---|---|---|---|---|
| RedisAdapter | low | medium-high | low-medium | low-medium | 11 |

For this class, the coupling was medium-high. You can also see this in the CBO (Coupling Between Objects). In this class there are 11 dependencies established through parameters, attributes being modified/read, method calls and object instantiation.

## Problem

The coupling is too high, which means that this class has the code smell coupler. The source of this problem is the following. When a message has to be processed by the RedisAdapter, on of these three classes is used:

- `nl.tudelft.sem.sem54.mainservice.service.ProcessCreditsFinished`
- `nl.tudelft.sem.sem54.mainservice.service.ProcessCreditsRemoved`
- `nl.tudelft.sem.sem54.mainservice.service.ProcessCreditsSpoiled`

Therefore, `RedisAdapter` has an instance of all of these three classes. This is not necessary.

## Solution

To solve this problem, we could extract this functionality into a new class. We put this in the new class `nl.tudelft.sem.sem54.mainservice.service.ProcessCreditsFactory`. This class has a method that takes the type of the message that the `RedisAdapter` has to process, and returns the right implementation of `ProcessCredits`. This way the coupling of `RedisAdapter` will be lower.

## Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO |
|---|---|---|---|---|---|
| RedisAdapter | low | low-medium | low | low | 8 |
| ProcessCredits Factory | low | low | low | low | 5 |

# MealController class

The class `nl.tudelft.sem.sem54.fridge.controller.MealController` is the class that handles an incoming meal request appropriately.

## Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC |
|---|---|---|---|---|---|---|
| MealController | low-medium | medium-high | low-medium | low-medium | 12 | 57 |

We can see that this class suffers from a medium-high coupling. This is also reflected in the CBO metric (Coupling Between Objects), which is 12. This means that the class depends on 12 other classes in the form of attributes or method calls.

## Problem

From the data above, we can see that the class suffered from a high coupling, which means it had a coupler code smell. The problem with this class is that, when a meal request was made, this class would handle all steps from verifying the meal request to executing the product consumptions. For example, the class depended on the class `nl.tudelft.sem.sem54.fridge.domain.User` because it was verifying users itself, and also on `nl.tudelft.sem.sem54.fridge.controller.pojo.TakeoutRequest`, because it was creating TakeoutRequest objects itself. This is not necessary.

## Solution

In order to solve this problem, functionality in the `MealController` could be extracted to other classes. For example, the `MealController` verified all users provided to check if all these users existed. This functionality has now moved to the `UserService` service, as this has actually more to do with users. The `MealController` also verified the `MealRequest`, to check if it had enough users and products. This functionality has been moved to the `MealRequest` class.

## Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC |
|---|---|---|---|---|---|---|
| MealController | low-medium | low-medium | low | low | 8 | 32 |
| MealRequest | low | low | low | low | 0 | 13 |
| UserServiceImpl | low | low | low | medium-high | 3 | 41 |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| ProductServiceImpl | low | low | low | medium-high | 2 | 35 |

Looking at these metrics, one may wonder why `UserServiceImpl` and `ProductServiceImpl` have a medium-high lack of cohesion. This is actually something we can easily explain. These classes are services for the `User` and `Product` objects respectively. What a service does is provide methods like `findById` or `findByUsername`. That means that in such classes, a lack of cohesion is to be expected.

## loginUser method

The method `loginUser()` in the mainService `UserController` class is attached to an endpoint that the user will use in the event that they want to login.

### Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO |
|---|---|---|---|---|---|
| loginUser | low | medium-high | low | low | 7 |

The method has a medium-high coupling, which suggests that it is modifying/dealing with too many objects.

### Problem

The method deals with logging in a user, and thus receives a message body in the form of a String, which should be parsed as JSON. To make sure that a new user is added to the mainService database, we extract the username from this JSON object. This was done using the GSON library within the method, which increased the coupling.

### Solution

The solution was to use the “Extract Method” refactoring technique, in which I extracted the functionality to a new method `getUsername(String jsonBody)`, which returns the extracted username from the JSON string, meaning in the `loginUser()` method all that has to be done is a call to the function.

### Metrics after

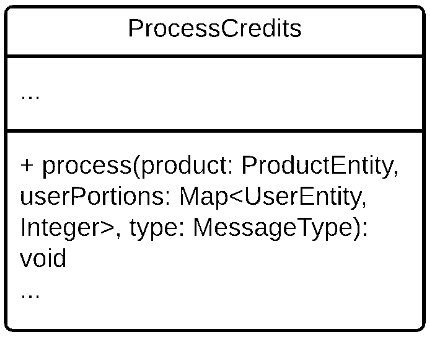
| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO |
|---|---|---|---|---|---|
| loginUser | low | low-medium | low | low | 4 |

# ProcessCredits class

This is a previous refactoring done earlier in the project. Namly on November 28, 2020. The smelly code was never actually pushed to gitlab, and therefore we can not run any statistics tools on this.

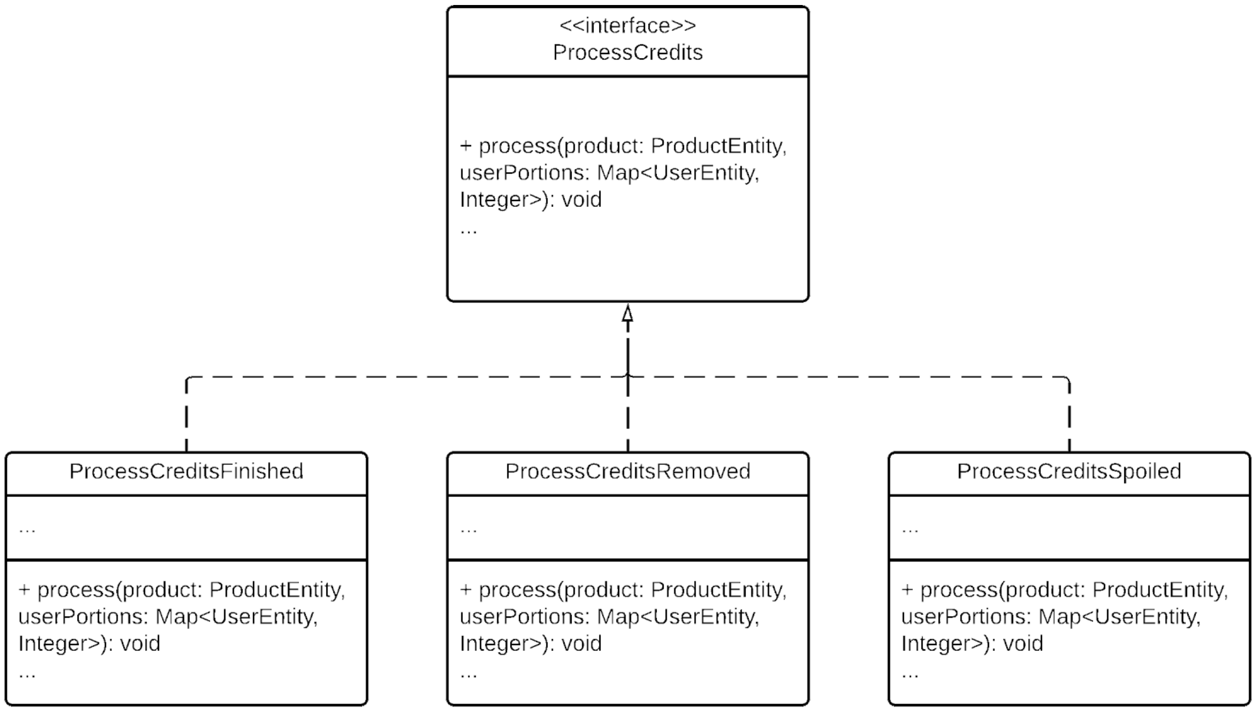
## Problem

There was an Object Oriented Abuser code smell in the form of a large switch statement used to select between the different forms of credit processing (in the event of an expiration, deletion, or a finished product).
This was done in a big class called `ProcessCredits` that handled these three cases using the large switch statement. This code is inefficient, and difficult to extend because it violates the open-closed principle. And so it was refactored.


ProcessCredits
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>, type: MessageType): void
...

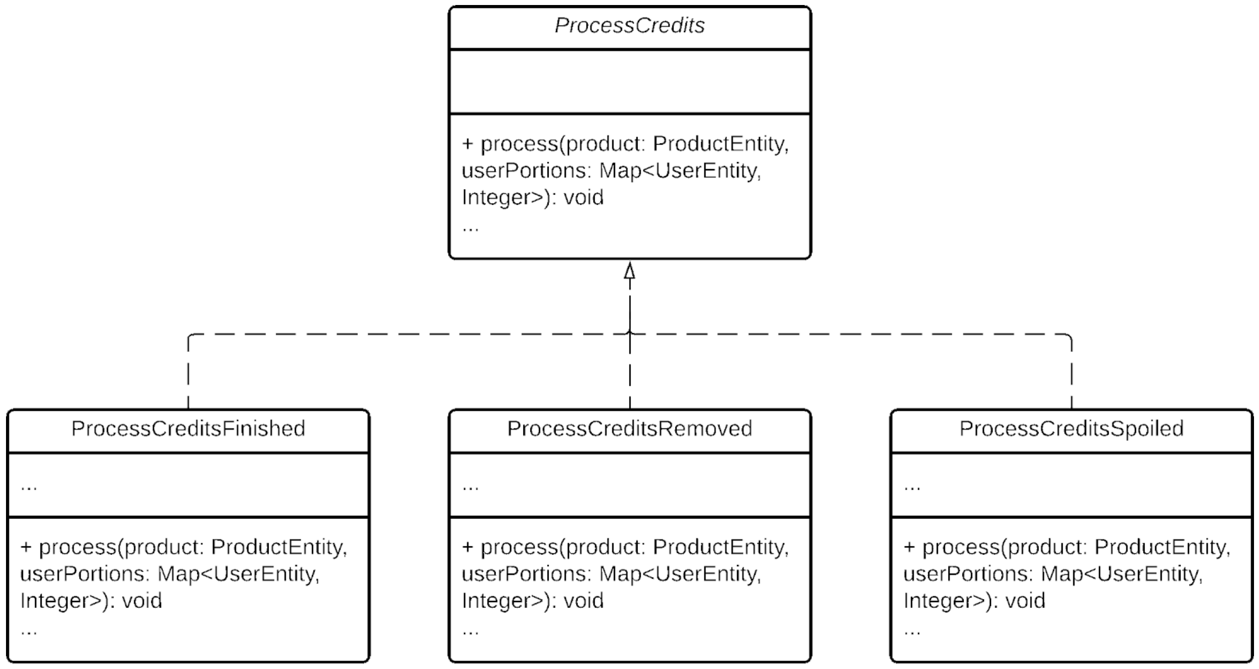
## Solution

The first solution was to make `ProcessCredits` an interface which was then concretely implemented three times into the classes `ProcessCreditsFinished`, `ProcessCreditsRemoved`, `ProcessCreditsSpoiled.`


<<interface>>
ProcessCredits
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsFinished
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsRemoved
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsSpoiled
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...

But this resulted in duplicated code between the three implementations. Namely in every implementation credits would be distributed over the users who used the product. The only difference was in the way it handled the remaining credits.

The proper solution that we went with was instead to make the `ProcessCredits` class abstract with the “duplicated” code inside, then the three child classes could simply call the super method and make their modifications as needed. In effect replacing the conditional with polymorphism. This removed the Object Oriented Abuser code smell.


*ProcessCredits*
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsFinished
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsRemoved
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...
ProcessCreditsSpoiled
...
+ process(product: ProductEntity, userPortions: Map<UserEntity, Integer>): void
...

## processMeal method

This is a refactoring job that happened earlier on in the project, namely on December 12, 2020

### Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| processMeal | low | medium-high | low-medium | low | 33 |

As we can see in the table, this method was quite large, at 33 lines. When this method was originally implemented, a merge request was made on which certain team members commented about this method length. That’s why it got refactored.

### Problem

As previously mentioned, the problem with this method was its size. At 33 lines, it was not that easy to read, nor was it easy to understand what different parts did. We can identify this as a “large method” code smell.

### Solution

The solution to this problem was fairly straight forward. We were able to apply the “extract method” refactoring technique, to extract certain functionalities from the method into different smaller methods. For example, the method `processMeal` verified the different parameters present in the given `MealRequest`. This functionality was then extracted into smaller methods in the same class. Later on in the project, we further refactored this code (see MealController Class).

### Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| processMeal | low | low-medium | low | low | 12 |
| parseUsedProducts | low | low | low | low | 11 |
| parseParticipatingUsers | low | low | low | low | 10 |

## FridgeController class

The class `nl.tudelft.sem.sem54.fridge.controller.FridgeController` is the class that handles requests for taking portions, undo and CRUD operations with products.

### Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC | LCOM |
|---|---|---|---|---|---|---|---|
| FridgeController | low-medium | medium-high | low-medium | medium-high | 20 | 164 | 0.595 |

We can see that this class suffers from a medium-high coupling and medium-high lack of cohesion. This is also reflected in the CBO metric (Coupling Between Objects), which is 20. This means that the class depends on 20 other classes in the form of attributes or method calls.

### Problem

From the data above, we can see that the class suffered from a lack of cohesion. The problem with this class is that it is responsible for too many different types of requests. It handles all the CRUD operations for products and also `takeportions` and `undo` requests. This is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.

### Solution

In order to solve this problem, functionality in the `FridgeController` could be extracted to two separate classes. For example, one class will be responsible for the CRUD operations and the other class will handle the request for taking portions and undo. Therefore, we split it and `ProductController` handles all the requests for operations with products and `FridgeController` now is only responsible for taking portion and undo.

### Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC | LCOM |
|---|---|---|---|---|---|---|---|
| FridgeController | low | low-medium | low-medium | low-medium | 16 | 117 | 0.542 |
| ProductController | low | low-medium | low-medium | low-medium | 14 | 78 | 0.55 |

## takeProduct method

This is the method responsible for doing the necessary business logic when a user requests taking a product.

### Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| takeProduct | low | very-high | medium-high | low | 93 |

As we can see, the method had some serious coupling problems and a large size, along with other members of the same class, `FridgeController` .

### Problem

The method had 2 main sources of coupling and a high size. First of all it did all of the verification of a request and had many branches for throwing different Exception classes. Then it did a permission check which employs 3 additional CRUD operations, which brings in a lot of extra classes and code.

### Solution

The solution to this problem was fairly straight forward. First-off we extract all permission checks in a new class, `PermissionService`. The permission service is something we should've thought of earlier, as it greatly improves the quality of our code. Then we extracted all logical branches of the method into separate methods, to improve readability and reduce size..

### Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| takeProduct | low | medium-high | low | low | 39 |

## undoLastTransaction method

This is the method responsible for undoing the last valid transaction of a user.

### Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| takeProduct | low | very-high | low-medium | low | 50 |

As we can see, the method had some serious coupling problems.

### Problem

The main reason that the method had a very-high sign of coupling is that it does a lot of checks. Considering we need to be sure that we can only undo a valid transaction and all of this should be saved and updated correctly to the database. It

### Solution

The solution to this problem was fairly straight forward. We had a check if the productId in the request is smaller than 0 and in that case we throw a custom exception. This was easily delegated to `ProductService` by adding this check to the method findByProductId. The other one was completely useless because it was checking if the product exists but that is already implemented in the method `ProductService.findById`. By removing it we now have less redundant and more readable code.

### Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| takeProduct | low | medium-high | low | low | 45 |

# editProduct method

This is the method responsible for editing existing products.

## Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| editProduct | low-medium | very-high | low-medium | low | 50 |

As we can see, the method had some serious coupling problems.

## Problem

The method had 2 main sources of coupling and a high size. First of all it did all of the verification of a request and had many branches for throwing different Exception classes. Then it checks whether portions of the product have been taken and whether the owner of the product is changing it.

## Solution

The solution to the problem was to extract some of the checks in other classes and services. For example we created a static method in the `ProductEntryRequest` that takes a request for an argument and does all the checks. Secondly, we delegated the check whether portions have been taken from a product to the `ProductService.`

## Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | LOC |
|---|---|---|---|---|---|
| takeProduct | low | medium-high | low | low | 35 |

# Product class

The class `nl.tudelft.sem.sem54.fridge.domain.Product` is the product entity.

## Metrics before

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC |
|---|---|---|---|---|---|---|
| Product | low-medium | low | low-medium | high | 3 | 113 |

We can see that this class suffers from a high lack of cohesion.

## Problem

From the data above, we can see that the class suffered from a lack of cohesion. The problem is that this class should be an entity with fields with getters and setters and methods inherited from `Object`. In our case we have other methods like addTransaction and getUserPortionMap which have some logic and should not be there.

## Solution

In order to solve this problem we should get rid of these 2 methods in that class and delegate them to some kind of service. By inspecting our code base we see we do not use addTransaction anywhere so that leaves us only with getUserPortionMap. We created a `PortionServiceImpl` that contains only the method getUserPortionMap and refactored the places where the method was called.

## Metrics after

| Name | Complexity | Coupling | Size | Lack of Cohesion | CBO | LOC |
|---|---|---|---|---|---|---|
| Product | low | low | low-medium | medium-high | 2 | 110 |