# Введение

Данный документ содержит комментарии к решению тестового задания.

# Проблемы исходной реализации

В данном разделе описаны проблемы исходной реализации.

## Архитектурные

В данном разделе описаны проблемы, не влияющие на функционирование, но усложняющие дальнейшую эксплуатацию.

### Жестко зашитая реализация кэширования

Во время эксплуатации может возникнуть потребность кэшировать данные не только в памяти приложения (например, класть их в БД). Также может потребоваться особая логика хранения кэша в памяти.

**Предложенное решение.** Создание интерфейса «Cache» (Приложение 1), его использование.

### **Обязательное использование кэша и синхронизации**

Исходная реализация предполагает, что пользователю точно понадобится кэш, при том кэш будет непотокобезопасен. Исходить из этого не совсем правильно.

**Предложенное решение.** Создание интерфейса «Processor». Имплементация его исходным решением. При взаимодействии с решением взаимодействовать через использование интерфейса, а не класса. Листинг интерфейса - Приложение 2.

### Выполняемое действие жестко привязано к исполнителю

В рамках доработки системы может потребоваться выполнение одного и того же функционала разными ее частями (пример: 2 «процессора» с кэшем и без выполняют одно и то же действие). Исходная реализация системы предполагает, что такая проблема будет решаться copy-paste’ом.

**Предложенное решение.** Создание интерфейса «Process», вынесение логики в его имплементацию. Листинг интерфейса - Приложение 3.

### Непрозрачность взаимодействия

Исходная реализация принимает и возвращает массив байт. Конечному пользователю было бы удобнее оперировать объектами классов. Предложенная реализация **не содержит** решения этой проблемы.

## Функциональные

В данном разделе описаны проблемы, приводящие к некорректному функционированию системы.

### Кэш заполняется, но не используется

Исходная реализация механизма кэширования содержит логическую ошибку. В JAVA нельзя сравнивать массивы через equals, потому их нельзя использовать как ключи HashMap. При использовании текущей реализации кэш будет постоянно увеличиваться, но данные из него использовать будет нельзя.

**Предложенная реализация.** Смотри Приложение 4.

### Неограниченный кэш

Текущая реализация системы не ограничивает и не очищает кэш. При длительной эксплуатации это приведет к сильному увеличению использования памяти.

**Предложенное решение.** Ограничить кэш по размеру и очищать его. Пример решения - Приложение 5.

### Монопольный режим

В рамках исходной реализации при ненахождении данных в кэше (что является частой ситуацией, особенно в первое время функционирования, когда кэш не заполнен) метод блокируется по инстансу класса (а не по входным данным). Это приводит к сильному снижению производительности при многопоточном режиме.

**Предложенное решение.** Если синхронизация все-таки необходима, блокировать доступ к операции по данным, а не по классу. Пример - Приложение 6.

# Решение

Решение доступно по URL:  
<https://github.com/androsovgr/sberbank-exam>

## Структура проектов

Ниже изображена структура проекта. Подпроекты:

* **sberbank-exam-original** – исходная реализация;
* **sberbank-exam-mine** – предложенная реализация. Содерждит также исходную реализацию, но со скорректированной архитектурой;
* **sberbank-exam-util** – содержит вспомогательные утилиты (генерацию входных данных).

## Ограничения и допущения

Написанные тесты не покрывают решение в полной мере. Тесты писать только учусь, книгу еще не дочитал, потому mock’и и spy-функционал не использовал.

1. Интерфейс «Cache»

**package** ru.sberbank.exam.mine.cache;

**public** **interface** Cache {

**public** **byte**[] get(**byte**[] input);

**public** **void** put(**byte**[] key, **byte**[] value);

1. Интерфейс «Processor»

**package** ru.sberbank.exam.mine.processor;

**import** ru.sberbank.exam.mine.process.Process;

**public** **interface** Processor {

**public** **byte**[] process(**byte**[] input);

**public** Process getProcess();

}

1. Интерфейс «Process»

**package** ru.sberbank.exam.mine.process;

**public** **interface** Process {

**public** **byte**[] process(**byte**[] input);

}

1. Реализация кэша на HashMap

**package** ru.sberbank.exam.mine.cache;

**import** java.util.HashMap;

**public** **class** HashMapCache **implements** Cache {

**private** HashMap<String, **byte**[]> cache = **new** HashMap<String, **byte**[]>();

@Override

**public** **byte**[] get(**byte**[] input) {

**return** cache.get(**new** String(input));

}

@Override

**public** **void** put(**byte**[] key, **byte**[] value) {

cache.put(**new** String(key), value);

}

}

1. Пример самоочищающегося кэша

**package** ru.sberbank.exam.mine.cache;

**import** org.apache.commons.collections.map.LRUMap;

**public** **class** LRUMapCache **implements** Cache {

**private** LRUMap cache;

**public** LRUMapCache(**int** cacheSize) {

cache = **new** LRUMap(cacheSize);

}

@Override

**public** **byte**[] get(**byte**[] input) {

**return** (**byte**[]) cache.get(**new** String(input));

}

@Override

**public** **void** put(**byte**[] key, **byte**[] value) {

cache.put(**new** String(key), value);

}

}

1. Обработчик с защитой в режиме многопоточности

**package** ru.sberbank.exam.mine.processor;

**import** java.util.concurrent.ConcurrentHashMap;

**import** java.util.concurrent.ConcurrentMap;

**import** ru.sberbank.exam.mine.cache.Cache;

**import** ru.sberbank.exam.mine.process.Process;

**public** **class** SynchronizedCachedProcessor **implements** Processor {

**private** Process process;

**private** Cache cache;

**private** ConcurrentMap<String, String> locks = **new** ConcurrentHashMap<String, String>();

**public** SynchronizedCachedProcessor(Process process, Cache cache) {

**super**();

**if** (process == **null** || cache == **null**) {

**throw** **new** IllegalArgumentException("Arguments can't be null");

}

**this**.process = process;

**this**.cache = cache;

}

@Override

**public** **byte**[] process(**byte**[] input) {

**byte**[] result = cache.get(input);

**if** (result == **null**) {

**synchronized** (getCacheSyncObject(**new** String(input))) {

result = cache.get(input);

**if** (result == **null**) {

result = process.process(input);

cache.put(input, result);

}

}

}

**return** result;

}

**private** Object getCacheSyncObject(**final** String id) {

locks.putIfAbsent(id, id);

**return** locks.get(id);

}

@Override

**public** Process getProcess() {

**return** process;

}

}