Домашнее задание №2. Рефлексия

Задание направлено на отработку навыка разработки инструментальных решений на основе рефлексии, позволяющих расширять функциональность существующего кода. Задание состоит из двух частей: базовой, с оценкой в два балла, и продвинутой, также с оценкой в два балла. Кроме того, есть дополнительное условие на один балл. К решенным заданиям следует предоставить **main** метод, запуск которого продемонстрирует выполнение всех условий задачи.

В процессе оценки сначала запускается приложенный main метод и набор тестов (см. бонусное задание). После этого производится код ревью представленного решения на выполнение всех условий.

Итого, за работу можно получить 4 основных и один бонусный балл.

Базовое задание (2 балла).

Разработайте обобщенный утилитный метод **cache**. Метод принимает объект и возвращает такую версию данного объекта, у которого вызовы всех методов, помеченных аннотацией **@Cache**, кэшированы. При вызове кэшированного метода необходимо проанализировать состояние объекта:

- Если метод вызывается впервые после создания кэшированного объекта - то он вызывается обычным образом.
- Если метод вызывается повторно и с момента прошлого вызова внесены изменения в состояние объекта – то он вызывается обычным образом.
- Если метод вызывается повторно и объект не был изменен с момента прошлого вызова – то вместо вызова метода необходимо вернуть то же значение, что возвращал метод при предыдущем вызове.

Для определения того, был ли изменено состояние объекта способом влияющим на сброс кэширования используйте аннотацию **@Setter**. Данная аннотация вешается на методы, которые изменяют состояние объекта значимым образом.

Обратите внимание, что метод принимает объект обобщенного типа **T** и возвращает объект того же типа.

С целью упрощения кода, в данном задании можно считать, что кэшировать достаточно только такие методы, которые данный объект реализовал от какого-либо интерфейса.

На рисунке 1 дан пример результата кэширования для объекта класса **A**. В 18 строчке текст “original method” не выведен на экран, так как с момента последнего вызова метода **method** объект не изменился, и следовательно возвращается кэшированное значение без вызова оригинального метода.

```java
public static void main(String[] args) {
    Able a=Utils.cache(new A());
    a.method(); //original method
    a.method(); //ничего, т.к. оригинальный метод не вызван
    a.setValue(); //seter method, происходит изменение состояния объекта
    a.method(); //original method
}
```

```java
class A implements Able{
    @Cache
    @Override
    public void method(){
        System.out.println( x:"original method");
    }
    @Setter
    @Override
    public void setValue(){
        System.out.println( x:"seter method");
    }
}
```

Рисунок 1. Пример кэширования

Аннотация **@Cache**, имеет следующие характеристики:

- Целью может быть МЕТОД
- Доступна во время исполнения программы

Аннотация **@ Setter**, имеет следующие характеристики:

- Целью может быть МЕТОД
- Доступна во время исполнения программы

Продвинутое задание (2 балла).

В данной вариации задания необходимо уметь кэшировать любые варианты методов: реализованные от интерфейсов, унаследованные от других классов и оригинальные методы, впервые представленные в данном классе. Обратите особое внимание на необходимость учитывать наличие метода, помеченного разработанными нами аннотациями, по всей иерархии класса. Считайте, что, если потомок переопределил метод, но не стал вешать аннотацию, значит метод не является сеттером или не требует кэширования (в соответствии с удаленной аннотацией).

Бонусное задание: тесты (1 балл)

Предоставьте в репозитории набор Junit тестов, которые тестируют корректность реализации кода для реализованного решения (только базовое или с продвинутым).

Критерии оценивания:

1. Тесты запускаются Maven или Gradle скриптом

2. Code Coverage 100% для метода **cache** и для метода обработки вызова кэшированного метода.