**Задание №2. Рефлексия**

Задание направлено на отработку навыка разработки инструментальных решений на основе рефлексии, позволяющих расширять функциональность существующего кода. К решению необходимо приложить модульные тесты.

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

* Если метод вызывается впервые после создания кэшируемого 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 @Mutator. 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 объект данного класса реализовал от какого-либо интерфейса.

Разберем задачу на примере. Допустим мы имеем интерфейс Fractionable определяющий метод получения вещественного значения у объекта.

**interface** Fractionable{  
 **double** doubleValue();  
 **void** setNum(**int** num) ;  
 **void** setDenum(**int** denum) ;  
}

Реализуется данный интерфейс классом Fraction, являющимся абстракцией для дробного значения и состоящий из числителя и знаменателя. У данного класса метод doubleValue помечен аннотацией @Cache, и, следовательно, должен кэшироваться. Оба сеттера помечены аннотацией @Mutator, и, следовательно их вызов приводит к сбрасыванию кэшей.

**public class** Fraction **implements** Fractionable{  
 **private int num**;  
 **private int denum**;  
  
 **public** Fraction(**int** num, **int** denum) {  
 **this**.**num** = num;  
 **this**.**denum** = denum;  
 }

@Mutator  
 **public void** setNum(**int** num) {  
 **this**.**num** = num;  
 }

@Mutator  
 **public void** setDenum(**int** denum) {  
 **this**.**denum** = denum;  
 }  
  
 @Override  
 @Cache  
 **public double** doubleValue() {  
 System.***out***.println(**"invoke double value"**);  
 **return** (**double**) **num**/**denum**;  
 }  
}

Далее проведем создание кэшируемой версии объекта класса Fraction и протестируем целевой метод.

**public static void** main(String... args) {  
 Fraction fr= **new** Fraction(2,3);  
 Fractionable num =Utils.*cache*(fr);  
 num.doubleValue();*// sout сработал* num.doubleValue();*// sout молчит* num.doubleValue();*// sout молчит* num.setNum(5);  
 num.doubleValue();*// sout сработал* num.doubleValue();*// sout молчит*}

Сначала создаем объект и задаем ему значение, затем, на его основе, создаем кэшируемую версию.

Первый вызов метода проходит нормально, так как еще ничего не было кэшировано.

Второй и третий вызов не приводят к выводу текста на экран, так как реальный метод расчёта не был вызван, вместо него срабатывает только возвращение кэша.

Далее мы вызываем метод мутатор, что приводит к сбросу кэшей

Четвертый вызов считается новым, и пятый вызов снова будет кэшированным.

**А**ннотация @Cache, имеет следующие характеристики:

* Целью может быть МЕТОД
* Доступна во время исполнения программы

**А**ннотация @Mutator, имеет следующие характеристики:

* Целью может быть МЕТОД
* Доступна во время исполнения программы