Курс:
«Язык программирования Java»

ТЕМА: TDD «СТРОКОВЫЙ КАЛЬКУЛЯТОР»

Цель лабораторной работы:

изучить модель разработки программного обеспечения: от тестов к коду (TDD).

Необходимые инструменты:

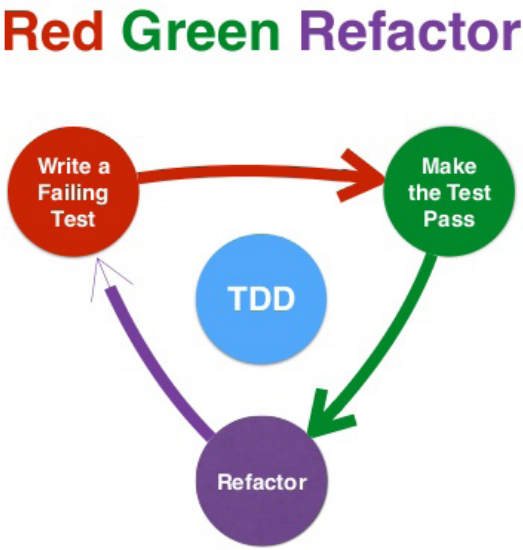
- *Eclipse*;
- Библиотека *JUnit 4*.

Разработка через тестирование (англ. *test–driven development, TDD*) – техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.

Разработка должна проходить через несколько этапов.

1. Добавление нового теста, для новой функциональности (без добавления кода тестируемой функциональности).
2. Запустить тест и убедиться, что тест не проходит. Красная зона.
3. Написать код самым простым и быстрым способом.
4. Запустить тест и убедиться, что тест проходит. Зеленая зона.

5. Рефакторинг – улучшить код, не изменяя самой функциональности.
6. Запустить тест и убедиться, что тест проходит. Зеленая зона.



7. Повторить первый шаг для новой функциональности.

Рефакторинг (англ. *refactoring*), или **реорганизация кода** – процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований. Поскольку каждое преобразование маленькое, программисту легче проследить за его правильностью, и

в то же время вся последовательность может привести к существенной перестройке программы и улучшению ее согласованности и четкости.

Цель рефакторинга – сделать код программы более легким для понимания; без этого рефакторинг нельзя считать успешным.

Рефакторинг проводят при наличии признаков плохого кода:

- дублирование кода;
- длинный метод;
- большой класс;
- длинный список параметров;
- «жадные» функции – это метод, который чрезмерно обращается к данным другого объекта;
- избыточные временные переменные;
- классы данных;
- несгруппированные данные.

Задание «String калькулятор»

Ниже приводится *TDD Kata*–упражнения в написании тестов, программировании и рефакторинге.

Прежде чем начинать:

- старайтесь не читать больше одного ***kata***, для этого – каждое новое задание будет расположено на новой странице;
- переходите к следующему заданию, только если полностью закончили текущее;
- выполняйте только одно *kata* за один раз. Хитрость заключается в том, чтобы научиться работать постепенно, выполняя задания шаг за шагом;
- убедитесь, что вы сделали тесты только для входных данных, которые указаны в задании. Нет никакой необходимости, дополнительно проверять входные данные которые отсутствуют в *kata*;
- помните, надо решать задачу как можно проще, так чтобы когда вы пишете тест, вы не задерживались на нем больше 2-5 минут.

Kata 1

Создайте класс *StringCalculatorTest* и добавьте в него первый тест, который должен тестировать статический метод *int add*(*String source*) из класса ***StringCalculator***. Обратите внимание, что на момент первого теста, класса и метода еще не должно существовать.

Метод может принимать строку, в которой может быть записано 0, 1 или 2 числа разделенных запятой. Метод должен возвращать сумму чисел, записанных в строке (для пустой строки возвращается значение 0).

Входные данные	Результат
null	0
""	0
"1"	1
" 1, 2 "	3
4,"	4

Начните с самого простого тестового случая пустой строки и переходите к следующим входным данным, только если прошли весь цикл *TDD*.

Kata 2

Сделать так, чтобы метод *add* обрабатывал любое количество чисел в исходной строке. Лишние пробелы в строке должны игнорироваться.

Входные данные	Результат
"1,2,3"	6
"1,1,1,1"	4
" 1,2 "	3
"11,22,33"	66

Помните о рефакторинге кода после каждого прохождения теста.

Kata 3

Сделать так, чтобы метод *add* мог обрабатывать новые линии между числами (дополнительно к запятой). \n – дополнительный разделитель.

Входные данные	Результат
"1 \n2,3"	6
"4\n5\n6"	15
"77\n"	77

Kata 4

Числа больше, чем 1000 в исходной строке должны игнорироваться.

Входные данные	Результат
"1, 1001"	1
"1002 \n 2"	2
"1, 1000"	1001
"1000, 1000"	2000
"1001\n1001"	0

Kata 5

Добавить поддержку различных разделителей. Начало строки будет содержать подстроку, которая выглядит следующим образом: "// [разделитель] \n [числа ...]". Разделитель не является обязательным. В качестве разделителя не должны использоваться цифры. В случае если в строке встречается символ, который не является цифрой или разделителем, вывести сообщение об ошибке. Все существующие до этого сценарии должны оставаться рабочими.

Входные данные	Результат
"//;\n1; 2"	3
"//* \n2,3"	5
"//#\n3# 4"	7
"//1\n1 1 1"	throw SpliterFormatException
"//;\n1#2"	throw SpliterFormatException

Kata 6

Необходимо сделать, чтобы разделители между числами могли бы быть любой длины в следующем формате:

"// [разделитель] *n*".

Входные данные	Результат
"//[***]\n1 *** 2 *** 3"	6
"//[xy]\n3xy4xy5xy8"	20

Kata 7

Добавить возможность добавлять несколько разделителей.

По шаблону: "// [*delim*1] [*delim*2] *n*".

Убедитесь, что программа может работать с несколькими разделителями разной длины (более одного символа).

Входные данные	Результат
"// [*] [%] \n1*2%3"	6
"// [&&] [||||] \ n3|||2&&3"	8

Kata 8

Добавить возможность передавать в метод *add* множество строк с исходными данными. Каждая строка может иметь свои разделители и числа. Метод должен суммировать числа из всех строк.

Пример:

StringCalculator.add(строка1, строка2);

StringCalculator.add(строка1, строка2, строка3);

Входные данные	Результат
"//[#$]\n1#$2 #$ 3", "// [&&] [||| \n3 |||2 &&3"	14
"1", "1,2", "1\n2,3"	10

Kata 9

При наличии отрицательных чисел в строке, должно генерироваться исключение. Если есть несколько отрицательных чисел, показать все отрицательные числа в сообщение об исключении.

Входные данные	Результат
"1, – 1"	throw NumberNegativException
" – 1, 1"	throw NumberNegativException
" – 1, – 1"	throw NumberNegativException
" – 1,1\n – 1"	throw NumberNegativException

Kata 10

Добавить проверку времени работы суммирования, для десяти строк. Время работы калькулятора не должно превышать 30 мс.

Входные данные	Результат
"// [*] [%] \n1 * 2% 3"	6
"// [&&] [||||]\n3 |||2 &&3"	8
"// [//] \n 1//2//4"	7
"999,999"	1998
"3,1001"	3
"// [*] [%][] \n1 * 2% 3"	6

Kata 11

Добавить возможность возведения чисел в строке в квадрат. Для указания степени используется символ ^.

Входные данные	Результат
"2^, 2"	6
"3^\n1"	10
"2^2, 2^2"	8

Kata 12

Добавить возможность возведения чисел в строке в указанную степень. Для указания степени используется символ ^.

Входные данные	Результат
"2^3, 2"	10
"3^3\n2^2"	31
"//;\n1; 2^8"	257

***Добавьте тесты, которые вы считаете необходимым добавить еще, для вышеперечисленных* kata.**