Технологии программирования

Лабораторная работа №4

**Модульное тестирование**

В данной лабораторной работе вы познакомитесь с основами модульного тестирования посредством фреймворка xUnit.

Работа выполняется группами из двух студентов. Работа может выполняться с использованием следующих языков и инструментальных средств:

* Java, фреймворк JUnit;
* C#/VB.NET, фреймворк NUnit.

Задание на лабораторную работу

Часть 1 (максимум 2 балла)

Реализуйте комплект модульных тестов для класса BigInteger (большие целые числа) из стандартной библиотеки классов. Необходимо написать тесты для как минимум следующих методов:

|  |  |  |
| --- | --- | --- |
| Java | .NET | Назначение метода |
| add() | Add() | Сложение |
| subtract() | Subtract() | Вычитание |
| multiply() | Multiply() | Умножение |
| divide() | Divide() | Деление |
| max() | Max() | Максимум |
| min() | Min() | Минимум |
| compareTo() | CompareTo() | Сравнение |
| gcd() | GreatestCommonDivisor() | Наибольшие общий делитель |
| toString() | ToString() | Преобразование в строку |

Должны быть протестированы стандартное выполнение операций, граничные значения и исключительные ситуации.

Часть 2 (максимум 3 балла)

Используя подход Test-Driven Development , при котором сначала пишутся тесты, а потом код, проходящий их (см. лекцию), реализуйте нижеописанный класс. При реализации используйте подход, совмещающий TDD и парное программирование, который называется TDD Ping Pong: один из программистов в паре пишет тест, который должен не выполниться. Второй программист реализует функционал, необходимый для выполнения этого теста и, при необходимости, выполняет рефакторинг (улучшает качество кода). После этого программисты меняются ролями, второй пишет тест, первый делает так, чтобы тест выполнился, и т.д. Процесс продолжается до тех пор, пока ни один из программистов не может написать тест, который не будет выполняться, в этом случае считается, что работа над классом завершена.

Пример подобного подхода: <http://szelenin.blogspot.ru/2011/12/tdd-prime-factors.html>

Для выполнения задания используйте целые числа, при возникновении дробных частей используйте округление до целых.

Реализуйте класс Wallet (кошелек) со следующей функциональностью. Изначально созданный кошелек является пустым. Должен иметься метод добавления определенного количества указанной валюты в кошелек, например, addMoney(“RUB”, 500) добавит в кошелек 500 рублей. Также должен иметься метод, извлекающий из кошелька определенное количество указанной валюты, например, removeMoney(“RUB”, 300). При добавлении валюты ее значение в кошельке увеличивается на соответствующее значение, при извлечении – уменьшается. Последовательное выполнение метода добавления одной и той же валюты должно приводить к последовательному увеличению этой валюты в кошельке. При попытке извлечь сумму валюты больше, чем имеется в кошельке, извлечение происходить не должно и должно генерироваться исключение.

В кошельке одновременно может находиться произвольное количество различных валют (RUB, USD, GBP и др.). Количество типов валют не ограничивается. Например, последовательное выполнение метода addMoney(“RUB”, 300) и addMoney(“USD ”, 100) добавит в кошелек 300 рублей и 100 долларов. Каждую из валют можно извлекать по-отдельности.

В классе Wallet должен быть реализован метод, возвращающий количество заданной валюты в кошельке, например, getMoney(“RUB”). В случае отсутствия данной валюты в кошельке метод должен возвращать 0.

В классе Wallet должен быть реализован метод, возвращающий количество различных видов валют, имеющихся в кошельке. Если какая-либо из валют в процессе работы была извлечена полностью, она не должны учитываться данным методом.

В классе Wallet должен быть реализован метод toString, возвращающий строку, описывающую содержание кошелька в виде “{ 300 RUB, 100 USD }”. Для пустого кошелька метод должен возвращать “{ }”

Часть 3 (максимум 4 балла) (продолжение части 2)

В классе Wallet должен быть реализован метод, возвращающий общую сумму денег в кошельке, приведенную к указанной валюте, например, getTotalMoney(“RUB”). Все валюты в кошельке при выполнении метода должны пересчитываться к указанной по текущему курсу. Чтобы определить курсы валют, метод должен обратиться к объекту класса Bank (банк), который также необходимо реализовать. В классе Bank необходимо реализовать метод, выполняющий перевод указанного значения денег из одной валюты в другую, например, convert(200, “RUB”, “USD”) должен вернуть сумму в долларах, эквивалентную 200 рублям.

Экземпляр класса Bank, с которым в дальнейшем будет использоваться кошелек, должен задаваться при создании объекта класса Wallet.

В класс Wallet необходимо добавить функциональность журналирования всех операций с деньгами (добавления и удаления). Журналирование должно производиться посредством класса MoneyLogger, в котором должен быть реализован метод log(String message), в простейшей реализации выводящий сообщение message на экран (вы можете реализовать более сложный вариант данного класса, выполняющий журналирование в файл). Экземпляр класса MoneyLogger, используемый операциями кошелька для журналирования, должен задаваться при создании объекта класса Wallet.

Для выполнения заданий этой части вам потребуется создать классы Bank и MoneyLogger. Для их разработки также применяйте TDD.

Обратите внимание, что модульные тесты должны тестировать классы в изоляции. В данной части же, например, метод getTotalMoney() будет содержать в себе обращение к объекту класса Bank. Может оказаться так, что тест, написанный для этого метода, будет по факту тестировать не просто метод класса Wallet, а еще метод класса Bank. Такие тесты не являются модульными по определению, и в данном задании их необходимо избегать.

Для изоляции класса Wallet при модульном тестировании используют подход, называемый Test Double (двойник для теста). Двойник – вариант класса, используемый только для тестирования другого класса. Существует два варианта двойников: stub (заглушка) и mock (имитация). Заглушки позволяют выполнять тестирование по состоянию, имитации – по поведению. Поскольку mock-тестирование более сложное и требует, как правило, использования дополнительных библиотек, в данном задании рекомендуется использовать тестирование с помощью заглушек.

Класс-заглушка содержит ту же операцию (или набор операций), что и класс, для которого заглушка создается, только эти операции не выполняются в реальности, а выполняют действия, необходимые для тестирования поведения основного тестируемого класса.

Рассмотрим пример. Пусть имеется метод класса calc, выполняющий сложение двух чисел и сохраняющий результат в файл:

class Calc {

…

pubic int sum(int a, int b, File f) {

int result = a + b;

f.write(result);

return a + b;

}

}

Для тестирования этого метода можно написать следующий тест (с учетом того, что File – это интерфейс, а TextFile – класс, его реализующий):

@Test

void testSum() {

Calc calc = new Calc();

File f = new TextFile(“c:/res.txt”);

assertEquals(5, calc.sum(2, 3, f));

}

Данный метод проверит правильность выполнения операции сложения, но, во-первых, при каждом выполнении теста будет выполняться запись в реально существующий файл на диске, а во-вторых, при ошибке в методе write() класса TextFile тест не выполнится, хотя тестируется класс Calc. Кроме того, необходимо еще написать тест, проверяющий, что метод add() выполняет операцию записи в файл.

Для устранения указанных проблем можно создать класс-заглушку, реализующий интерфейс File, и использовать его в тестовом методе:

class FileStub implements File {

int res = 0;

public void write(int a) {

res = a;

}

}

Тест будет выглядеть следующим образом:

@Test

void testSum() {

Calc calc = new Calc();

File f = new FileStub();

assertEquals(5, calc.sum(2, 3, f));

assertEquals(5, f.res);

}

Теперь при проверке операции сложения запись в файл не выполняется, вместо этого класс-заглушка сохраняет результат записи внутри себя и позволяет проверить, что операция записи была вызвана с правильным параметром.

Более подробно про реализацию заглушек и их использование можете прочитать по следующим ссылкам:

<http://www.nestor.minsk.by/kg/2008/05/kg80510.html>

<http://itvdn.com/ru/video/test-driven-development/stub-objects>