**­­­­Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

по дисциплине «Конструирование программного обеспечения»

**Тема:** **Статический анализ кода**

Студенты гр. РИВ-400027у Л.А. Кайгородова

|  |
| --- |
| О.В. Дрон  Д.И. Кудинов  В.И. Пинтак  Я.В. Козлов |

Преподаватель С.И. Тимошенко,

доц., к.т.н

**Екатеринбург 2023**

Содержание

[1 Постановка задачи 3](#_Toc146373857)

[2 Анализ поставленной задачи 4](#_Toc146373858)

[2.1 Задание 1 4](#_Toc146373859)

[2.2 Задание 2 9](#_Toc146373860)

[2.3 Задание 3 11](#_Toc146373861)

[3 Анализ полученных результатов 14](#_Toc146373862)

[ПРИЛОЖЕНИЕ А (справочное) А1 Исходный текст программы 1 15](#_Toc146373863)

[А2 Исходный текст программы 2 15](#_Toc146373864)

[А3 Исходный текст программы 3 15](#_Toc146373865)

[А4 Исходный текст программы 4 15](#_Toc146373866)

[А5 Исходный текст программы 5 15](#_Toc146373867)

# 1 Постановка задачи

Для задания 1 необходимо, разработать программы на java для каждого из разобранных там примеров. Проверить их с помощью выбранного статического анализатора кода. Тексты программ, результаты работы и пояснения проблем включить в отчет.

Для задания 2 необходимо, проверить с помощью выбранного статического анализатора кода наличие проблем у проекта library (из папки «Материал для лабораторной N 4»). Проект требует подключения библиотеки jsr305-2.0.0.jar из той же папки. Результаты работы с пояснениями проблем включить в отчет.

Для задания 3 необходимо, проверить с помощью выбранного статического анализатора кода наличие проблем у библиотеки colt (из папки «Материал для лабораторной N 4». Результаты работы с пояснениями проблем включить в отчет. При обнаружении более десяти проблем, описать только наиболее критичные.

# 2 Анализ поставленной задачи

## 2.1 Задание 1

Для выполнения данной работы был установлен плагин SpotBugs.

После установки были разработаны программы примеров и проверены с помощью SpotBugs, результаты проделанных работ представлены на рисунках 1-5.

Исходные тексты программы приведены в приложение А.

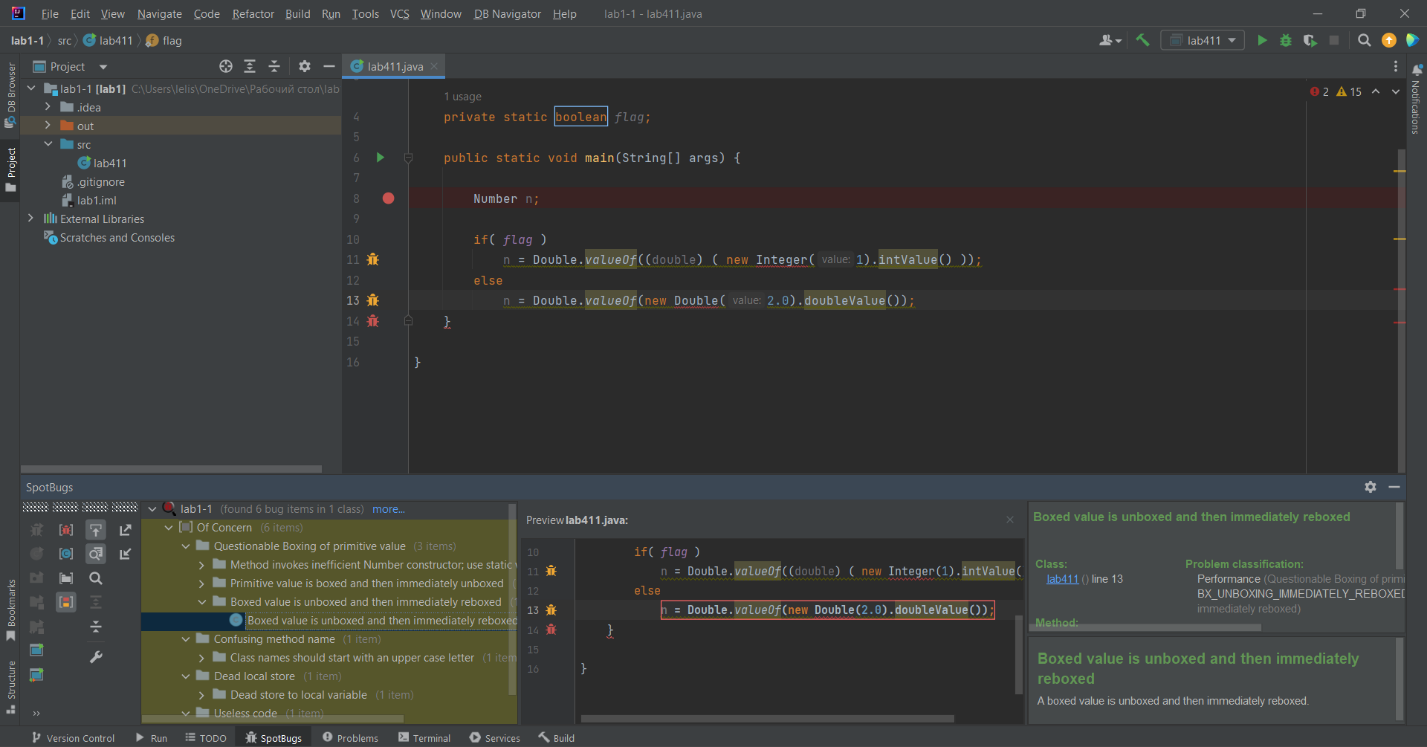


Рисунок 1 – работа SpotBugs с 1 примером.

Компилятора код не содержит проблем и прекрасно компилируется. Но StopBugs выдаёт предупреждение:

1. main(String[]) invokes inefficient new Integer(int) constructor; use Integer.valueOf(int) instead;

Метод вызывает неэффективный конструктор Number; вместо этого используйте статическое valueOf.

Использование new Integer(int) всегда гарантированно приводит к созданию нового объекта, тогда как Integer.valueOf(int) позволяет кэшировать значения компилятором, библиотекой классов или JVM. Использование кэшированных значений позволяет избежать выделения объектов, и код будет работать быстрее.

Значения от -128 до 127 гарантированно имеют соответствующие кэшированные экземпляры, а использование valueOf примерно в 3,5 раза быстрее, чем использование конструктора. Для значений вне постоянного диапазона производительность обоих стилей одинакова.

Если класс не должен быть совместим с JVM, предшествующими Java 1.5, используйте либо автоупаковку, либо метод valueOf() при создании экземпляров Long, Integer, Short, Character и Byte.

1. Primitive value is unboxed and coerced for ternary operator**.**

Примитивное значение распаковывается и приводится для тернарного оператора.

Обернутое примитивное значение распаковывается и преобразуется в другой примитивный тип как часть оценки условного тернарного оператора (оператор b ? e1 : e2). Семантика Java предписывает, что если e1 и e2 являются обернутыми числовыми значениями, значения распаковываются и преобразуются/приводятся к их общему типу (например, если e1 имеет тип Integer, а e2 имеет тип Float, то e1 распаковывается, преобразуется в значение с плавающей запятой и заключено в выборку.

1. Primitive value is boxed and then immediately unboxed.

Примитивное значение упаковывается, а затем сразу же распаковывается

Это, вероятно, связано с ручным упаковыванием в месте, где требуется неупакованное значение, что заставляет компилятор немедленно отменить работу упаковки.

1. Boxed value is unboxed and then immediately reboxed.

Упакованное значение распаковывается, а затем сразу же повторно упаковывается.

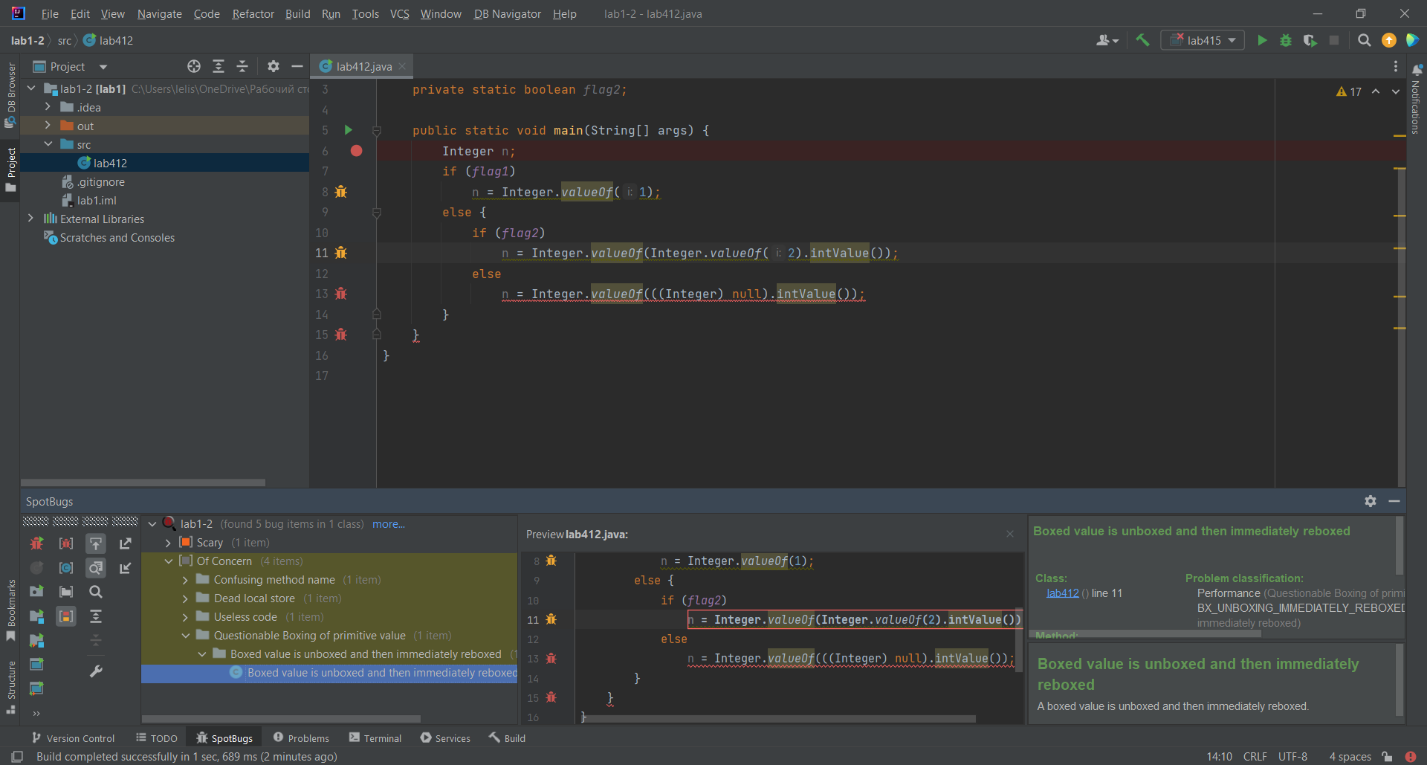


Рисунок 2 – работа SpotBugs с 2 примером.

Здесь FindBugs выдаёт два сообщения, которых достаточно, чтобы заподозрить ошибку:

1. BX\_UNBOXING\_IMMEDIATELY\_REBOXED

Упакованное значение распаковывается, а затем сразу же повторно упаковывается в TestTernary.main(String[])

1. NP\_NULL\_ON\_SOME\_PATH.

Здесь переименовывается нулевой указатель. Это приведет к исключению NullPointerException при выполнении кода.

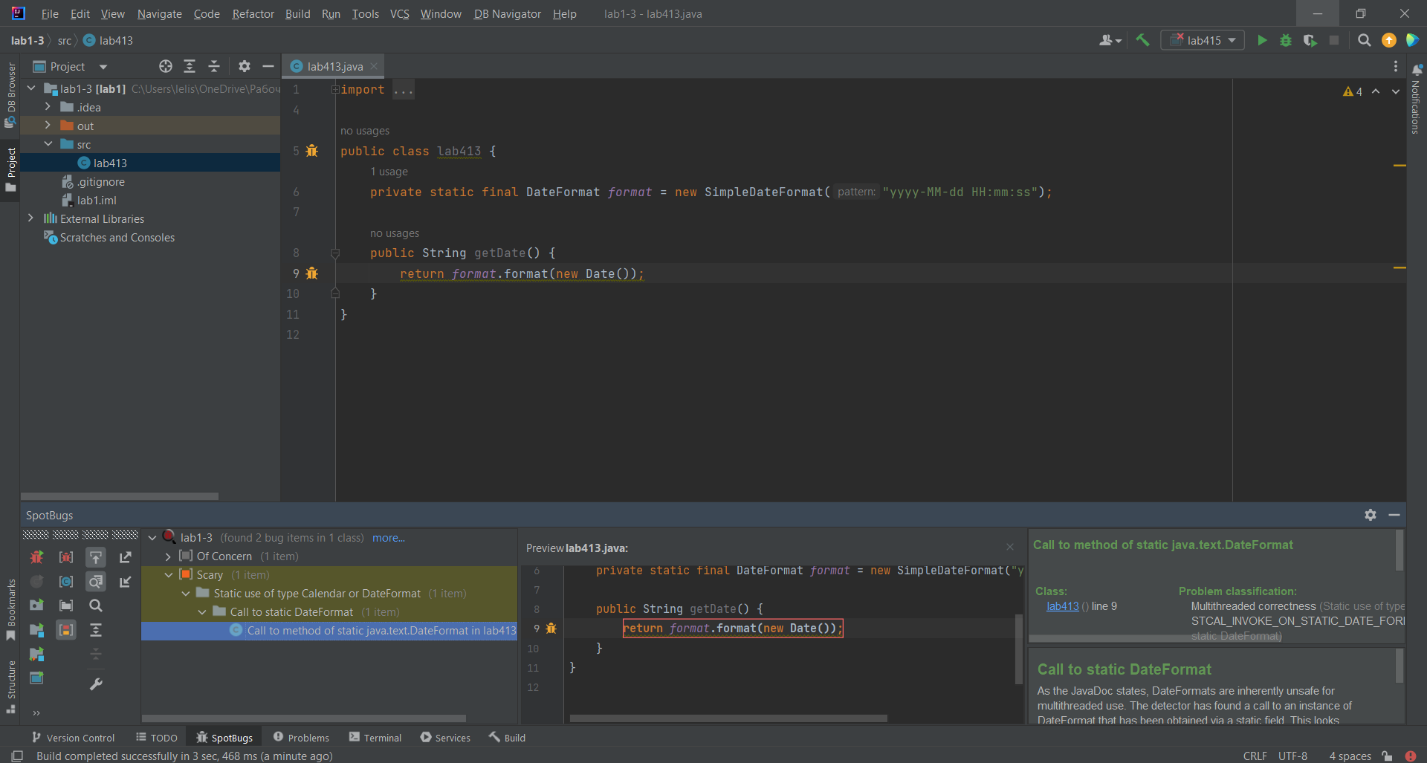
****

Рисунок 3 – работа SpotBugs с 3 примером.

В процессе выполнения метода format() объект пишет в поля класса, поэтому одновременное использование SimpleDateFormat из двух потоков приведёт с некоторой вероятностью к неправильному результату. Здесь FindBugs выдаёт 1 сообщения:

1. Call to method of static java.text.DateFormat.

Форматы DateFormat небезопасны для многопоточного использования. Детектор обнаружил обращение к экземпляру DateFormat, полученному через статическое поле. Это выглядит подозрительно.

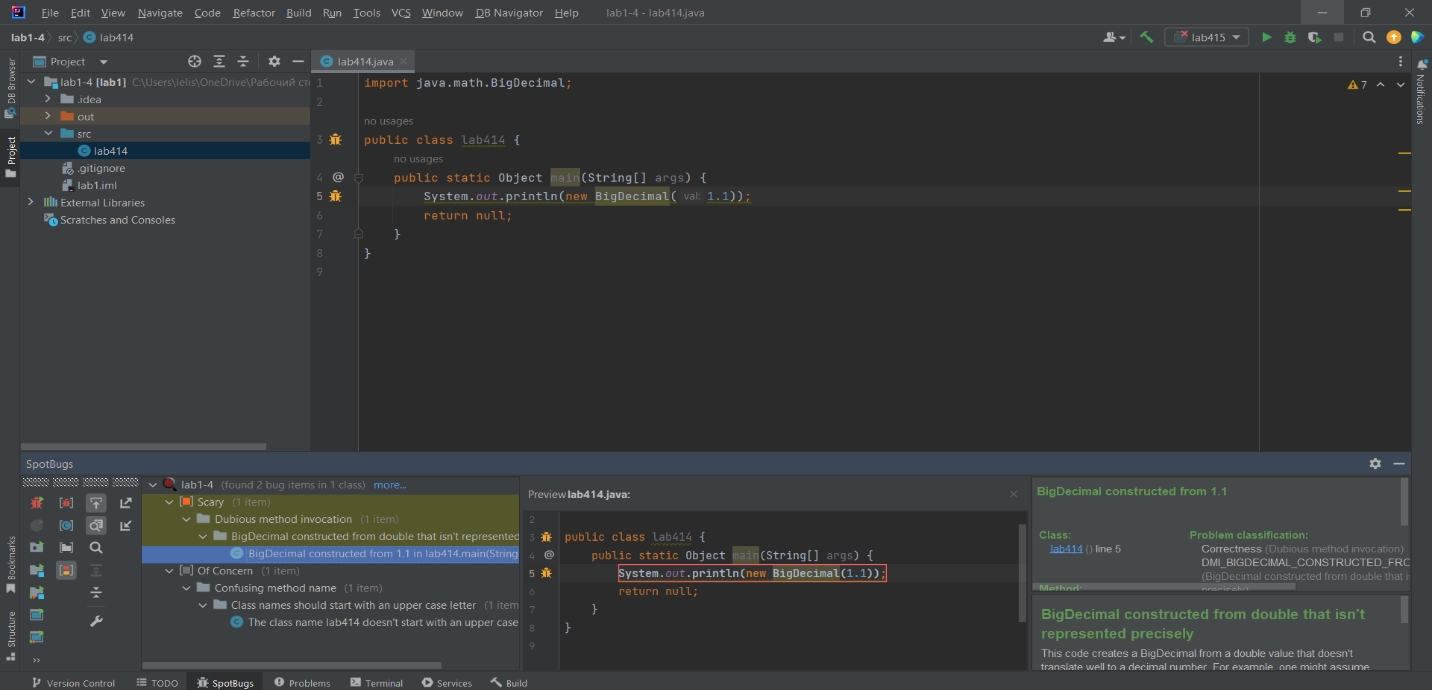


Рисунок 4 – работа SpotBugs с 4 примером.

Результат может показаться неожиданным: 1.100000000000000088817841970012523233890533447265625. Так происходит, потому что примитивный double хранится в формате IEEE754, в котором невозможно представить 1.1 идеально точно. Поэтому там хранится максимально близкое значение к 1.1. А FindBugs выдаёт 1 сообщения выдает следующее сообщение:

1. BigDecimal constructed from 1.1.

Этот код создает BigDecimal из двойного значения, которое плохо преобразуется в десятичное число. Например, можно предположить, что запись new BigDecimal(0.1) в Java создает BigDecimal, который точно равен 0,1 (немасштабированное значение 1 с масштабом 1), но на самом деле он равен 0,100000000000000000555111151231257827021181583404541015625. Вероятно, вы захотите использовать метод BigDecimal.valueOf(double d), который использует строковое представление числа double для создания BigDecimal (например, BigDecimal.valueOf(0.1) дает 0,1).

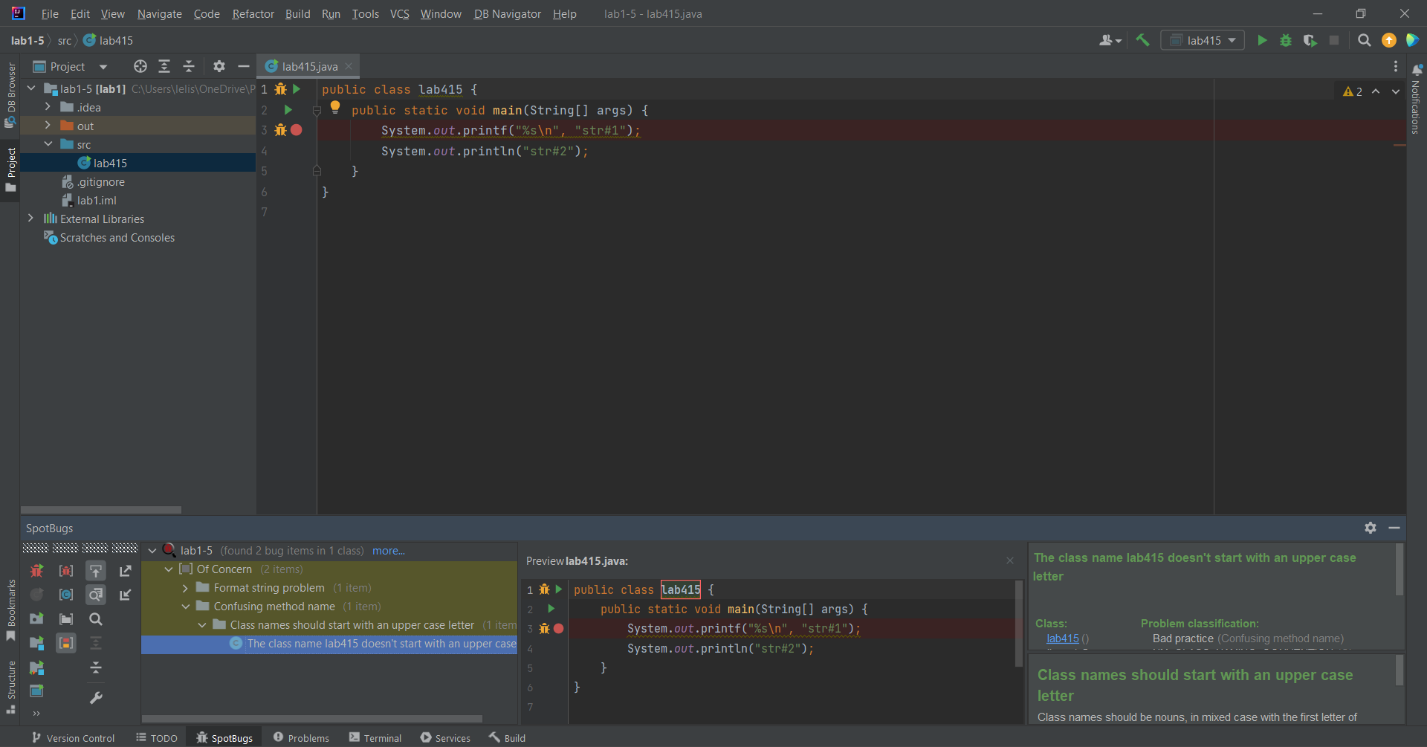


Рисунок 5 – работа SpotBugs с 5 примером.

Таким образом можно получить странную комбинацию переводов строки в одном потоке, что выглядит неаккуратно и может снести крышу какому-нибудь парсеру. Ошибку можно долго не замечать, особенно если вы преимущественно работаете на Unix-системах. Для того, чтобы вставить правильный перевод строки с помощью printf, используется специальный символ форматирования "%n". Вот что пишет FindBugs по этому поводу:

1. Format string should use %n rather than \n.

Эта строка формата включает символ новой строки (\n). В строках формата обычно предпочтительнее использовать %n, который будет создавать разделитель строк для конкретной платформы.

## 2.2 Задание 2

Для выполнения данной лабораторной работы требуется подключение библиотеки jsr305-2.0.0.jar., для подключения было достаточно положить его в папку lib директории Java.

После подключения был запущен проект library и проверен с помощью SpotBugs, результаты проделанной работы представлен на рисунке 6.

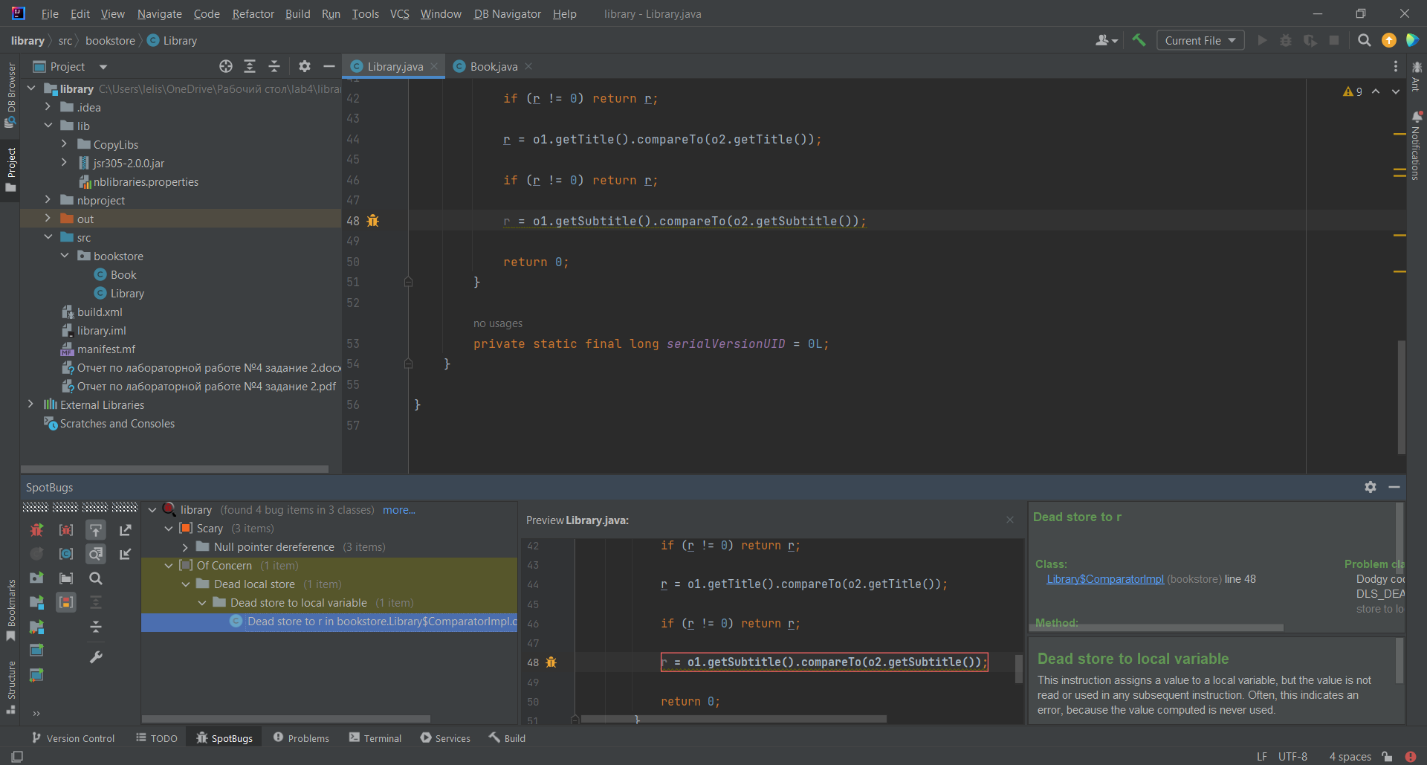


Рисунок 6 – работа SpotBugs с проектом library.

StopBugs успешно проверил данный проект, найдя несколько несущественных ошибок:

1. Dead store to r;

Эта инструкция присваивает значение локальной переменной, но это значение не считывается и не используется ни в одной последующей инструкции. Часто это указывает на ошибку, поскольку вычисленное значение никогда не используется.

Компилятор Sun javac часто создает мертвые хранилища для конечных локальных переменных. Поскольку SpotBugs — это инструмент на основе байт-кода, не существует простого способа устранить эти ложные срабатывания.

Возможно, использовать конструктор, который принимает начальный размер, и это будет намного лучше, но из-за loadFactor карт и наборов даже это не будет правильной оценкой.

Если использовать Guava, его методы, которые выделяют карты и наборы с заранее определенным размером, чтобы получить наилучшие шансы на отсутствие перераспределения, например:

* Sets.newHashSetWithExpectedSize (целое число)
* Maps.newHashMapWithExpectedSize(int)

Если нет, хорошей оценкой будет ожидаемый размер / {LOADING\_FACTOR}, который по умолчанию равен 0,75.

## 2.3 Задание 3

Для выполнения данной лабораторной работы требуется подключение библиотеки colt.

После подключения был запущен проект colt и проверен с помощью SpotBugs, результаты проделанной работы представлен на рисунке 7.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 7 – работа SpotBugs с проектом colt

StopBugs проверил данный проект, на ряд ошибок:

1. Check for oddness that won't work for negative numbers;

Проверка на нечетность, которая не будет работать для отрицательных чисел. Код использует x % 2 == 1, чтобы проверить, является ли значение нечетным, но это не будет работать для отрицательных чисел (например, (-5) % 2 == -1). Если этот код предназначен для проверки на нечетность, рассмотрите возможность использования x & 1 == 1 или x % 2 != 0.

1. Random object created and used only once;

Случайный объект, созданный и использованный только один раз. Этот код создает объект java.util.Random, использует его для генерации одного случайного числа, а затем отбрасывает объект Random. Это создает случайные числа посредственного качества и неэффективно. Если возможно, перепишите код так, чтобы объект Random создавался один раз и сохранялся, а каждый раз, когда требуется новое случайное число, вызывайте метод существующего объекта Random для его получения.

Если важно, чтобы сгенерированные случайные числа нельзя было угадать, вы не должны создавать новый случайный номер для каждого случайного числа; значения слишком легко угадываются. Вместо этого вам следует настоятельно рассмотреть возможность использования java.security.SecureRandom (и избегать выделения нового SecureRandom для каждого необходимого случайного числа).

1. new IllegalArgumentException() not thrown;

Исключение создано и удалено, а не выброшено. Этот код создает объект исключения (или ошибки), но ничего с ним не делает.

1. Field SimpleLongArrayList.size masks field in superclass cern.colt.list.AbstractLongList;

Класс определяет поле, которое маскирует поле суперкласса. Этот класс определяет поле с тем же именем, что и видимое поле экземпляра в суперклассе. Это сбивает с толку и может указывать на ошибку, если методы обновляют или обращаются к одному из полей, когда им нужно другое.

1. Bad comparison of nonnegative value with 0;

Плохое сравнение неотрицательного значения с отрицательной константой или нулем. Этот код сравнивает значение, которое гарантированно неотрицательно, с отрицательной константой или нулем.

1. Repeated conditional test;

Повторные условные тесты. Код содержит условную проверку, выполняемую дважды, одну сразу после другой (например, x == 0 || x == 0). Возможно, второе вхождение должно быть чем-то другим (например, x == 0 || y == 0).

1. Useless increment in return from apply(int);

Бесполезное увеличение в операторе возврата. Этот оператор имеет возврат, такой как return x++; / вернуть х--;. Увеличение/уменьшение постфикса не влияет на значение выражения, поэтому это увеличение/уменьшение не имеет никакого эффекта. Убедитесь, что это утверждение работает правильно.

1. Found reliance on default encoding: new java.io.FileReader(String).

Использование кодировки по умолчанию. Обнаружен вызов метода, который выполняет преобразование байта в строку (или строки в байт) и предполагает, что кодировка платформы по умолчанию подходит. Это приведет к тому, что поведение приложения будет отличаться на разных платформах. Используйте альтернативный API и явно укажите имя набора символов или объект набора символов.

# 3 Анализ полученных результатов

Для выполнения каждого задания лабораторной работы сделано:

1. Для задания 1, были разработаны программы на Java для каждого из разобранных примеров. Также разработанные программы были проверены с помощью выбранного статического анализатора кода SpotBugs.
2. Для задания 2, была осуществлена проверка проекта library на наличие проблем у проекта, были обнаружены незначительные ошибки в коде. Также была подключения библиотеки jsr305-2.0.0.jar.
3. Для задания 3, была осуществлена проверка проекта colt на наличие проблем у проекта colt. SpotBugs нашел ряд ошибок.

Данная лабораторная работа позволила изучить работу с анализаторами кода и применить знания на практических примерах, также позволила изучить работу с ошибками, на которые указывает анализатор когда.

ПРИЛОЖЕНИЕ А  
(справочное)  
А1 Исходный текст программы 1

Number n;

if( flag )

n = Double.valueOf((double) ( new Integer(1).intValue() ));

else

n = Double.valueOf(new Double(2.0).doubleValue());

А2 Исходный текст программы 2

Integer n;

if( flag1 )

n = Integer.valueOf(1);

else {

if( flag2 )

n = Integer.valueOf(Integer.valueOf(2).intValue());

else

n = Integer.valueOf(((Integer)null).intValue());

}

А3 Исходный текст программы 3

private static final DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public String getDate() {

return format.format(new Date());

}

А4 Исходный текст программы 4

System.out.println(new BigDecimal(1.1));

return null;

А5 Исходный текст программы 5

System.out.printf("%s\n", "str#1");

System.out.println("str#2");