**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 «Проектирование информационных систем»

**Тема: Статический анализ кода**

|  |  |
| --- | --- |
| Студенты группы РИВ-410938у: | Е.В. Москвина  А.А. Калугина  А.А. Сентябова  Р.С. Ионов  А.С. Кибардин |
| Преподаватель: | С.И. Тимошенко доц., к.т.н. |

**Екатеринбург 2024**

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc177727734)

[2 Анализ поставленной задачи 4](#_Toc177727735)

[3 Описание результатов 5](#_Toc177727736)

[3.1 Задание 1 5](#_Toc177727737)

[3.1.1 Примеры с Integer и Double 5](#_Toc177727738)

[3.1.2 Пример с DateFormat 9](#_Toc177727739)

[3.1.3 Примеры с BigDecimal 11](#_Toc177727740)

[3.1.4 Пример с переводом строк и printf 12](#_Toc177727741)

[3.2 Задание 2 13](#_Toc177727742)

[3.3 Задание 3 16](#_Toc177727743)

[4 Анализ полученных результатов 23](#_Toc177727744)

[4.1 Анализ 23](#_Toc177727745)

[4.2 Вывод 23](#_Toc177727746)

1. Постановка задачи

Выбрать и установить один из статических анализаторов кода для языка Java (встроенный в среду программирования, SpotBugs, PMD, PVS-Studio, SonarQube, Solar inCode или другой), который поддерживается в IntelliJ IDEA. Использовать выбранный статический анализатор кода для выполнения заданий №1 – №3.

Задание 1. Прочитать файл «FindBugs помогает узнать Java лучше.mhtml». Разработать программы на java для каждого из разобранных там примеров. Проверить их с помощью выбранного статического анализатора кода.

Задание 2. Проверить с помощью выбранного статического анализатора кода наличие проблем у проекта library.

Задание 3. Проверить с помощью выбранного статического анализатора кода наличие проблем у библиотеки colt. При обнаружении более десяти проблем, описать только наиболее критичные.

1. Анализ поставленной задачи

Для выполнения лабораторной работы необходимо изучить понятия статического анализа кода и ознакомиться с существующими инструментами для статического анализа кода.

В качестве статического анализатора будет использована программа PVS-Studio.

PVS-Studio — это статический анализатор кода для языков C, C++, C#, Java. Входит в реестр отечественного ПО: N 9837., PVS-Studio разрабатывается с учётом требований, предъявляемых к статическим анализаторам в ГОСТ Р 71207–2024. PVS-Studio помогает улучшать код по трём направлениям: качество, надёжность и защищённость (безопасность).

В качестве IDE будет использована IntelliJ IDEA 2023.1.3 (Ultimate Edition).

1. Описание результатов

На сайте производителя скачиваем программу PVS-Studio с расширением .exe, производим установку, представленную на рисунке 1. Предварительно закрываем все открытые файлы IDE. Программа автоматически устанавливается на все IDE, которые были выбраны в меню установки.

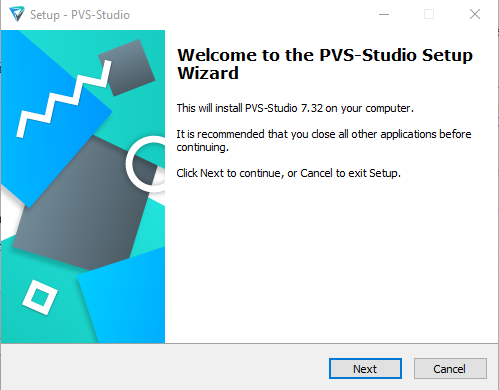


Рисунок 1 – Установка PVS-Studio

В IntelliJ IDEA 2023.1.3 (Ultimate Edition) программа расположена в нижнем меню, на рисунке 2.

Рисунок 2 – Создание проекта в PVS-Studio

* 1. Задание 1

Разработаем программы для каждого из разобранных примеров в статье «FindBugs помогает узнать Java лучше».

* + 1. Примеры с Integer и Double

Разработаем программу для демонстрации примеров с типами Integer и Double (листинг 1). Добавим перехват исключений и запуск в цикле для второго примера, чтобы гарантированно получить исключение.

Листинг . Программа для примеров с Integer и Double



Добавим вызов разработанных методов в класс Main,листинг 2.

Листинг . Выполнение программы для примеров с Integer и Double



Запустим программу и получим вывод, представленный на рисунке 3. Примеры 2 и 3 действительно вызывают исключения.

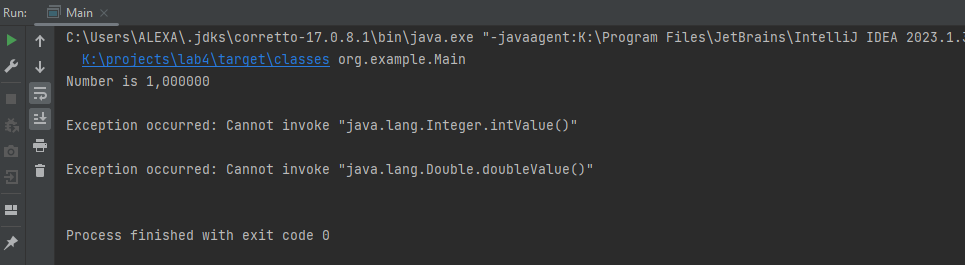


Рисунок 3 – Результат выполнения программы IntelliJ IDEA

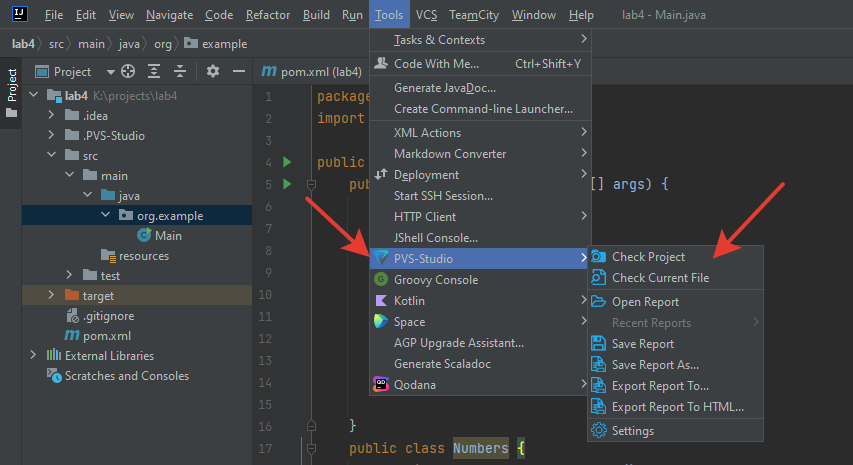
Запуск программы в IntelliJ IDEA, рисунок 4. Обратимся к результатам статического анализа программы для примеров с Integer и Double, рисунок 5. В отчете PVS-Studio мы видим все найденные ошибки, о которых говорилось в статье.

Рисунок 4 – Запуск плагина PVS-Studio в IntelliJ IDEA

V6088 Результат этого выражения будет неявно преобразован в 'double'. Проверьте, правильно ли это обрабатывается логикой программы;

V6093 Автоматическая распаковка литерала 'null' приведет к исключению NullPointerException;

V6046 Неверный формат. Несовместимые типы для аргумента 'n': требуется '%f';

V6093 Автоматическая распаковка литерала 'null' приведет к исключению NullPointerException.

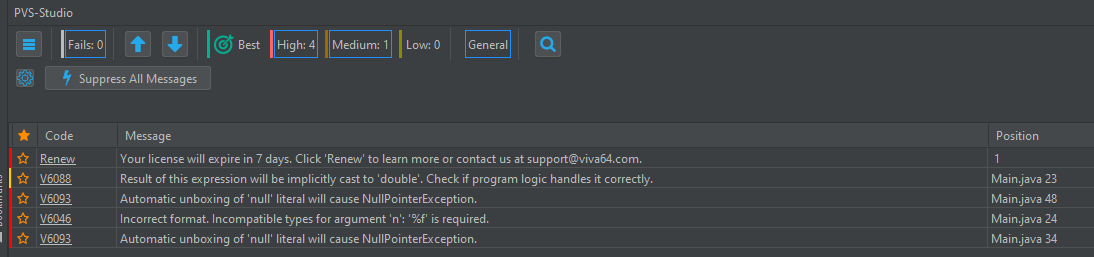


Рисунок 5 – Результат работы анализатора PVS-Studio для примеров с Integer и Double

* + 1. Пример с DateFormat

Разработаем программу для демонстрации примеров с DateFormat. Добавим перехват исключений для вывода сообщения об ошибке в консоль. Добавим также конкурентный вызов метода, чтобы убедиться в отсутствии потокобезопасности экземпляра SimpleDateFormat (листинг 4).

Листинг . Программа для примера с DateFormat



Запустим программу (листинг 5) и получим вывод, представленный на рисунке 6.

Листинг . Выполнение программы для примера с DateFormat



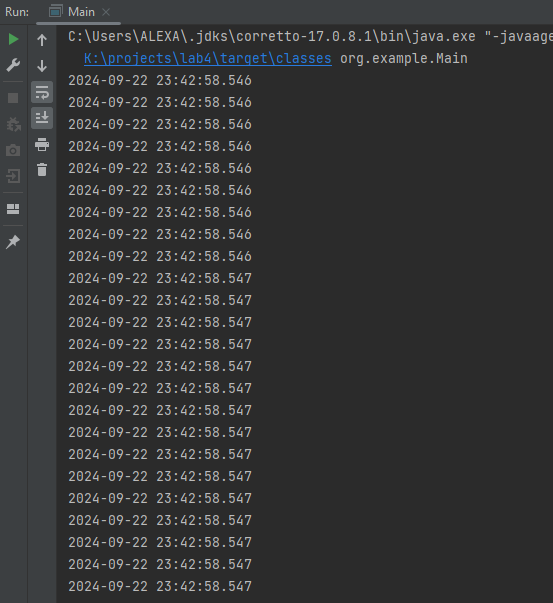


Рисунок 6 – Результат выполнения программы для примеров с DateFormat

Для исходной версии программы не удалось получить ошибку конкурентного доступа, однако это удалось сделать после создания экземпляра Date из строки вместо использования конструктора (листинг 6).

Листинг . Доработанный метод getDateInternal



После запуска доработанной программы мы видим в консоли ошибки с нечитаемыми символами и неправильное чтение даты из строки, рисунок 7, что явно свидетельствует об ошибке конкурентного доступа.

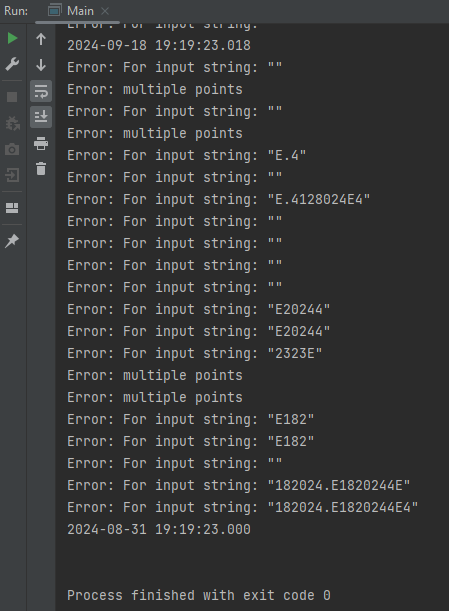


Рисунок 7 – Результат выполнения программы для примеров с DateFormat с ошибками

Обратимся к результатам статического анализа программы для примеров с DateFormat, рисунок 7. В отчете PVS-Studio мы не видим ошибок, предупреждающих о том, что экземпляр DateFormat не является потокобезопасным, рисунок 8.

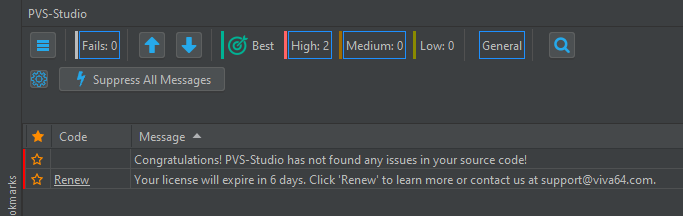


Рисунок 8– Результат работы анализатора PVS-Studio для примеров с DateFormat

Сообщили о проблеме производителю, заполнив форму обратной связи.

* + 1. Примеры с BigDecimal

Разработаем и запустим программу для демонстрации примеров с BigDecimal (листинг 7, листинг 8).

Листинг . Программа для примеров с BigDecimal



Листинг . Выполнение программы для примеров с BigDecimal



После запуска программы мы получим вывод, представленный на рисунке 9.

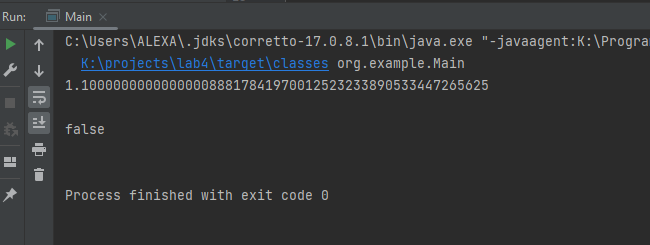


Рисунок 9 – Результат выполнения программы для примеров с BigDecimal

Мы видим, что, как и говорилось в статье, число из примера 1 не равно 1.1, а два числа из примера 2 не являются равными друг другу. Как видно на рисунке 10 , PVS-Studio по результатам статического анализа предупреждает нас о первой ошибке, однако была пропущена ошибка сравнения чисел.

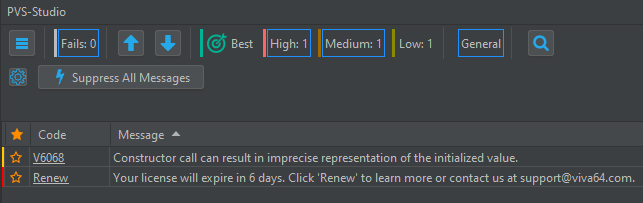


Рисунок 10 – Результат работы анализатора PVS-Studio для примера с BigDecimal

V6068 может привести к неточному представлению инициализированного значения.

* + 1. Пример с переводом строк и printf

Разработаем и запустим программу для демонстрации примеров с переводом строк и printf. Исходный код представлен на листинге 9.

Листинг . Программа для примера с переводом строк и printf



Направим результат выполнения программы в файл, рисунок 11 и проанализируем его содержимое.



Рисунок 11 – Направление результата выполнения программы в файл

Мы видим, что при выполнении программы в файл был записан корректный перенос строки, рисунок 12. PVS-Studio не предупреждает, что правильно использовать специальный символ форматирования "%n", рисунок 13, рисунок 14.

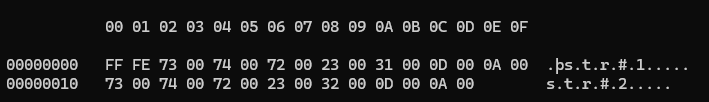


Рисунок 12 – Результат выполнения программы для примера с переводом строк и printf

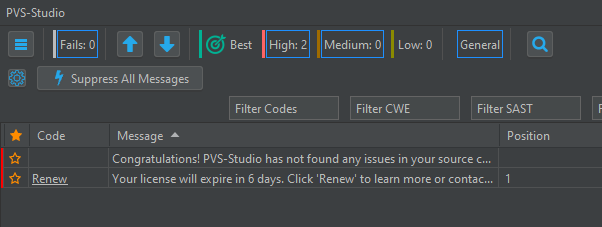


Рисунок 13 – Результат работы анализатора PVS-Studio для примера с printf

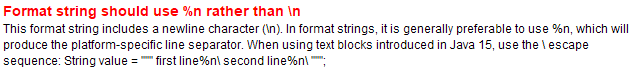


Рисунок 15. Описание ошибки для примера с printf

* 1. Задание 2

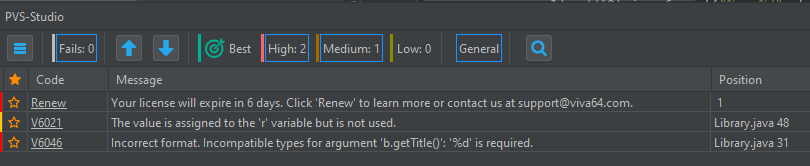
Проверим с помощью статического анализатора кода PVS-Studio наличие проблем у проекта library. Результат статического анализа представлен на рисунке 16. Найдено 2 ошибки, рассмотрим их подробнее.

Рисунок 16 – Результат статического анализа проекта library

V6021 Значение присвоено переменной 'r', но не используется;

V6046 Неверный формат. Несовместимые типы для аргумента 'b.getTitle()': требуется '%d'.

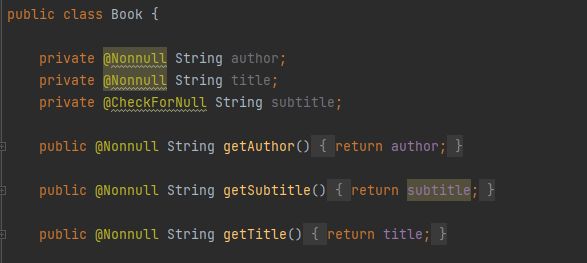


Рисунок 19 – Анализ проекта library, ошибка 1, исходный код

Обратившись к подсветке ошибки в исходном коде на рисунке 19, видим, что поля author и title помечены аннотацией @Nonnull, а значит должны быть проинициализированы при создании экземпляра Book, но этого не происходит, т.к. у класса Book есть только конструктор без параметров по умолчанию.

Ошибка «Dead store to r in bookstore.Library$ComparatorImpl.compare(Book, Book)», рисунок 20.

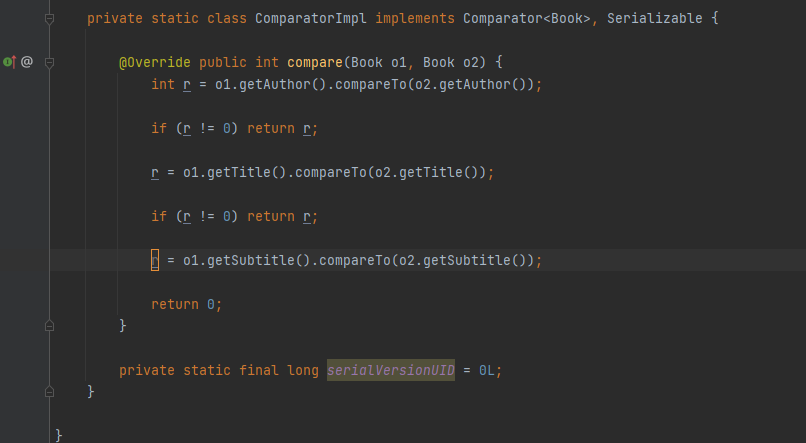


Рисунок 20 – Анализ проекта library, ошибка 2

Обратившись к подсветке ошибки в исходном коде на рисунке 20, видим, что переменной r присваивается значение результата сравнения o1.getSubtitle и o2.getSubtitle, однако эта переменная нигде далее не используется, из метода просто возвращается 0. Это потенциально может привести к ошибкам в ходе выполнения программы.

* 1. Задание 3

Проверим с помощью статического анализатора кода PVS-Studio наличие проблем у проекта colt. Результат статического анализа представлен на рисунке 21. Найдена 63 ошибки.

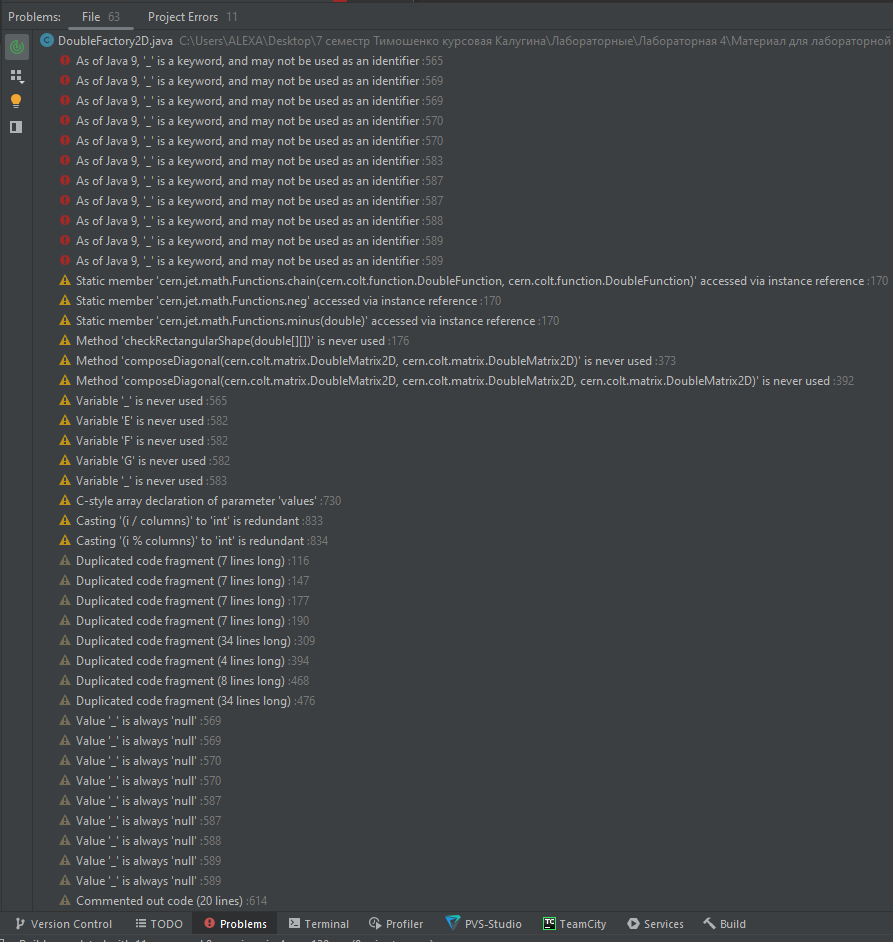


Рисунок 21 – Анализ проекта colt, все ошибки IntelliJ IDEA

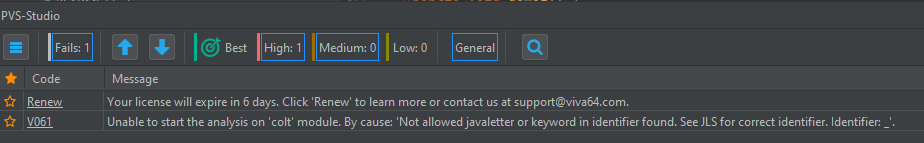


Рисунок 27 – Анализ проекта colt, PVS-Studio

V061 Не удается запустить анализ в модуле colt. Причина: "В идентификаторе не найдена допустимая буква java или ключевое слово. Правильный идентификатор указан в JLS. Идентификатор: \_".

Обратившись к подсветке ошибки в исходном коде на рисунке 30, мы видим, что при выполнении условия lambda <= 0.0 Exception просто создается и не выбрасывается. Это может привести к пропуску ошибок и неверному результату выполнения программы.

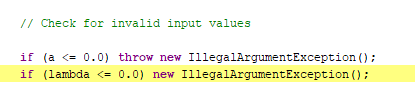


Рисунок 30 – Анализ проекта colt, ошибка 1, исходный код

1. Ошибка 2, 3 «Useless increment in return from cern.jet.math.IntFunctions$2.apply(int)» (рисунок 31, рисунок 32).

Обратившись к подсветке ошибки в исходном коде на рисунках 33-34, мы видим, что в инструкции return в качестве параметра используются операторы «а++» и «а--». Это приводит к тому, что из метода возвращается исходное значение переменной «а», а не обновленное: таков принцип работы постфиксного инкремента/декремента.



Рисунок 33 – Анализ проекта colt, ошибка 2, исходный код

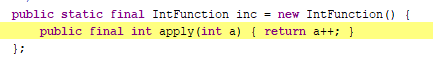


Рисунок 34 – Анализ проекта colt, ошибка 3, исходный код

1. Ошибка 4 «Repeated conditional test in cern.colt.bitvector.BitMatrix.elements(long[], int, int)» (рисунок 35, рисунок 36).

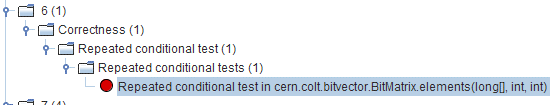


Рисунок 35 – Анализ проекта colt, ошибка 4

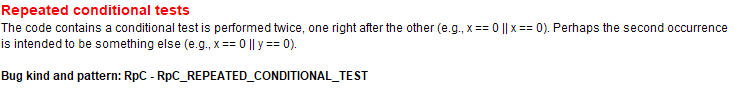


Рисунок 36 – Анализ проекта colt, ошибка 4, описание

Обратившись к подсветке ошибки в исходном коде на рисунке 37, мы видим, что дважды выполняется проверка «columns<0». Вероятно, это ошибка типа copy-paste, и вместо второго повторяющегося условия должно быть другое, которые пропущено. Это может привести к неверному ветвлению во время выполнения программы.

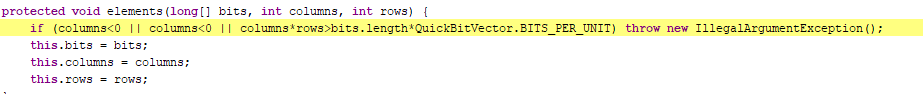


Рисунок 37 – Анализ проекта colt, ошибка 4, исходный код

1. Ошибка 5 «Bad comparison of nonnegative value with 0 in cern.colt.bitvector.BitMatrix.elements(long[], int, int)» (рисунок 38, рисунок 39).

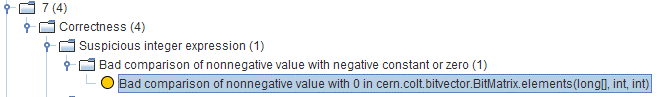


Рисунок 38 – Анализ проекта colt, ошибка 5

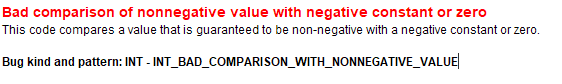


Рисунок 39 – Анализ проекта colt, ошибка 5, описание

Обратившись к подсветке ошибки в исходном коде на рисунке 40, мы видим, что анализатор предупреждает о том, что осуществляется проверка «<0» для значения уже заведомо больше или равного нулю. Эта ошибка находится в той же части кода, что и ошибка в п.3, но она получена после применения другого правила анализа.

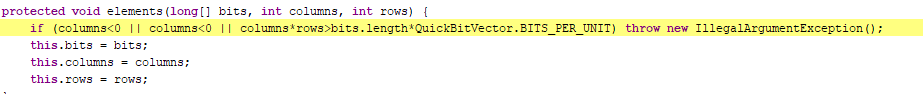


Рисунок 40 – Анализ проекта colt, ошибка 5, исходный код

1. Ошибка 6, 7, 8 «Using floating-point loop counters can lead to unexpected behavior» (рисунок 41, рисунок 42).

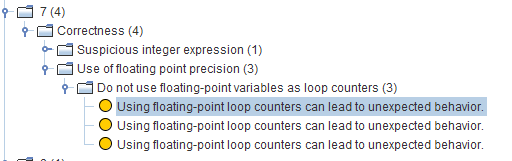


Рисунок 41 – Анализ проекта colt, ошибка 6, 7, 8

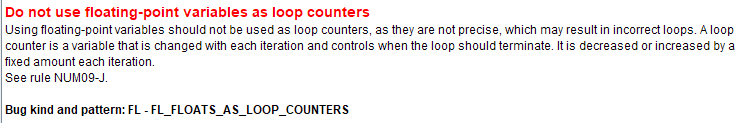


Рисунок 42 – Анализ проекта colt, ошибка 6, 7, 8, описание

Обратившись к подсветке ошибки в исходном коде на рисунках 43-45, мы видим, что проверка условия в циклах осуществляется для значений с плавающей точкой, что может привести к неверному ветвлению кода, т.к. значения с плавающей точкой не обладают необходимой точностью.

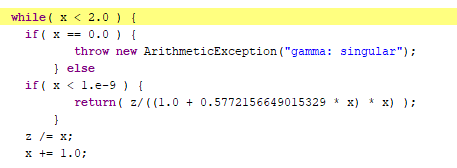


Рисунок 43 – Анализ проекта colt, ошибка 6, исходный код



Рисунок 44 – Анализ проекта colt, ошибка 7, исходный код

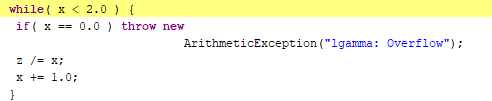


Рисунок 45 – Анализ проекта colt, ошибка 8, исходный код

1. Ошибка 9 «Field SimpleLongArrayList.size masks field in superclass cern.colt.list.AbstractLongList» (рисунок 46, рисунок 47).

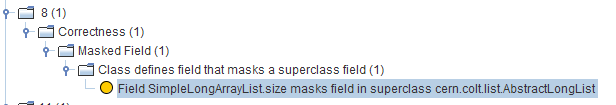


Рисунок 46 – Анализ проекта colt, ошибка 9

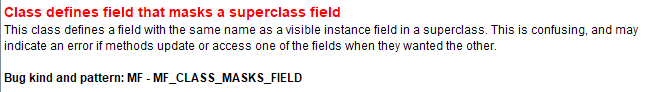


Рисунок 47 – Анализ проекта colt, ошибка 9, описание

Обратившись к подсветке ошибки в исходном коде на рисунке 48, мы видим, что в классе SimpleLongArrayList определено поле size с таким же названием и областью видимости, как и в родительском классе AbstractLongList. Это может привести к неожидаемому поведению программы, т.к. обращение к полю size не будет полиморфным: при обращении к полю size экземпляра SimpleLongArrayList через SimpleLongArrayList будет возвращено одно значение, а при обращении через AbstractLongList – другое. Возможно, по задумке автора такое поведение ожидаемое, но часто это может быть ошибкой.

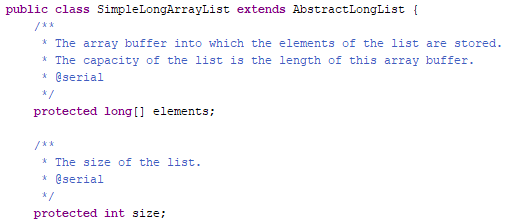


Рисунок 48 – Анализ проекта colt, ошибка 9, исходный код

1. Ошибка 10 «Field DelegateDoubleMatrix1D.content masks field in superclass cern.colt.matrix.impl.WrapperDoubleMatrix1D» (рисунок 49, рисунок 50).

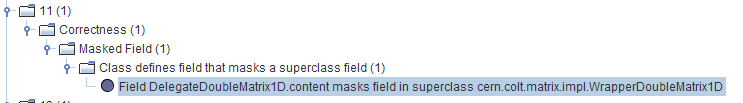


Рисунок 49 – Анализ проекта colt, ошибка 10

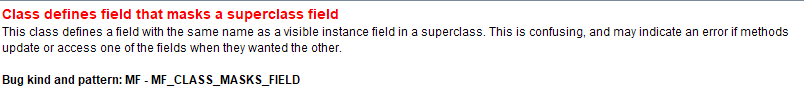


Рисунок 50 – Анализ проекта colt, ошибка 10, описание

Обратившись к подсветке ошибки в исходном коде на рисунке 51, мы видим ошибку аналогичную п. 9, но в другом классе. Возможно, по задумке автора такое поведение ожидаемое, но часто это может быть ошибкой.

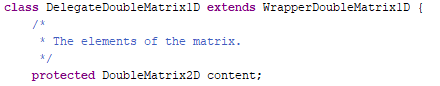


Рисунок 51 – Анализ проекта colt, ошибка 10, исходный код

1. Анализ полученных результатов
   1. Анализ

В первом задании мы реализовали примеры из материала «FindBugs помогает узнать Java лучше» и проверили их с помощью статического анализатора SpotBugs. Анализ выявил следующие проблемы:

* неэффективный боксинг и анбоксинг;
* потенциальные NullPointerException;
* потокобезопасность SimpleDateFormat;
* инициализация BigDecimal;
* портируемость форматирования строк: использование символа \n для переноса строки может вызвать проблемы на разных платформах.

При анализе проекта library с помощью SpotBugs были обнаружены следующие недостатки:

* неинициализированные поля с аннотацией @Nonnull;
* методы, возвращающие null при аннотации @Nonnull;
* неиспользуемый код.

Анализ библиотеки colt выявил значительное количество потенциальных дефектов:

* общее число обнаруженных проблем: 781;
* основные типы проблем:
  + логические ошибки в условиях и сравнениях;
  + скрытие полей суперклассов: может приводить к непредвиденному поведению при наследовании;
  + некорректная обработка исключений: исключения создаются, но не выбрасываются, что может скрыть ошибки.
  1. Вывод

Проведенный анализ с использованием PVS-Studio показал не эффективность статического анализа кода для выявления скрытых ошибок и потенциальных проблем в программах на Java. Инструмент не позволил обнаружить недостатки как в небольших учебных примерах, так и в крупных проектах, таких как colt.

Использование статического анализа способствует повышению качества программного обеспечения, позволяя выявлять и устранять ошибки на ранних этапах разработки. Это повышает надежность и эффективность приложений, а также облегчает их дальнейшую поддержку и развитие.