Министерство науки и высшего образования РФ

ФГАОУ ВО «Уральский федеральный университет

имени первого Президента России Б. Н. Ельцина»

ИРИТ-РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ N 4**

по дисциплине «Конструирование программного обеспечения»

**Тема:** Статический анализ кода

|  |  |
| --- | --- |
| Студент группы РИЗ-490028у: | А.А. Трифонов  Д.А. Житников  Д.В. Федяков  В.Н. Жирнова |
| Преподаватель: | С.И. Тимошенко,  доц., к.т.н |

Екатеринбург 2022

**1 Постановка задачи**

Разработать программы на java для каждого из разобранных примеров.

Проверить их с помощью выбранного статического анализатора кода. Тексты программ, результаты работы и пояснения проблем включить в отчет.

**2 Анализ поставленной задачи**

Для выполнения данной лабораторной работы требуется установить плагин SpotBugs.

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

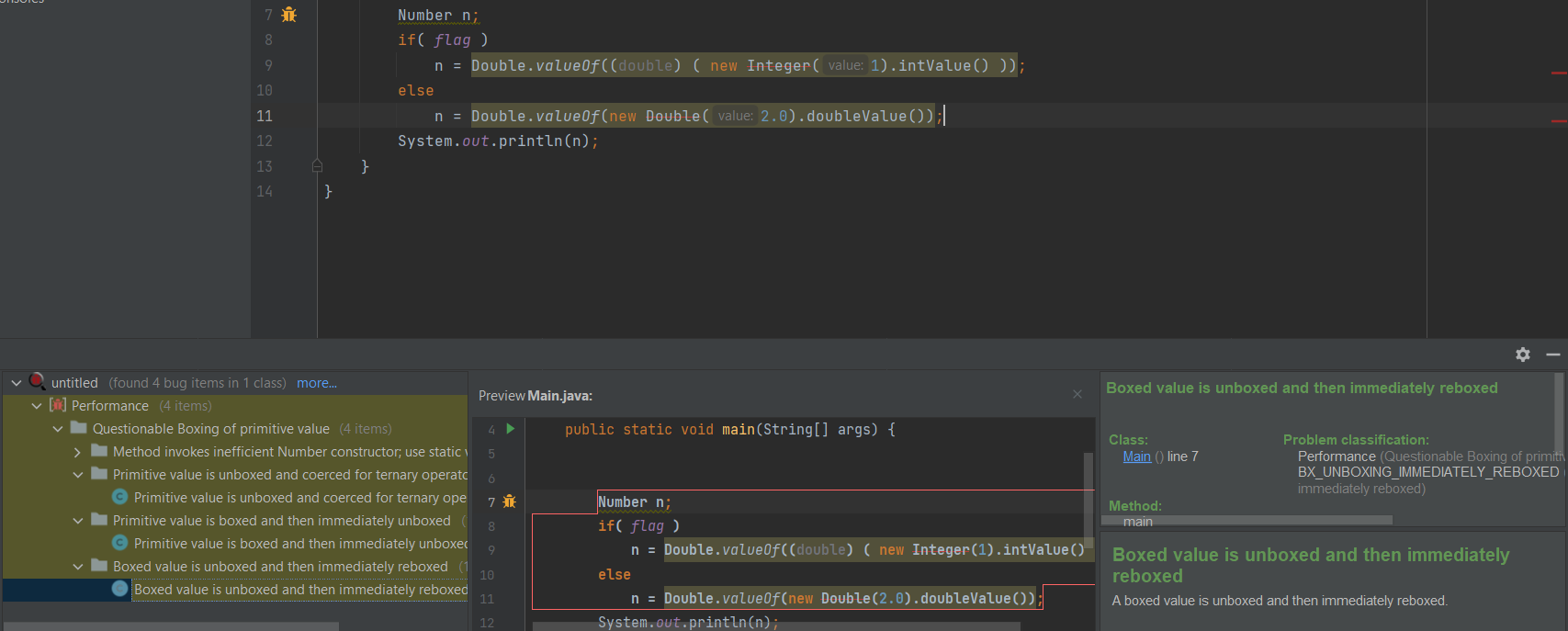


Рисунок 1 – работа SpotBugs с 1 примером.

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

* 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.

* 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.

* 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.

* Boxed value is unboxed and then immediately reboxed.

Упакованное значение распаковывается, а затем сразу же повторно упаковывается.

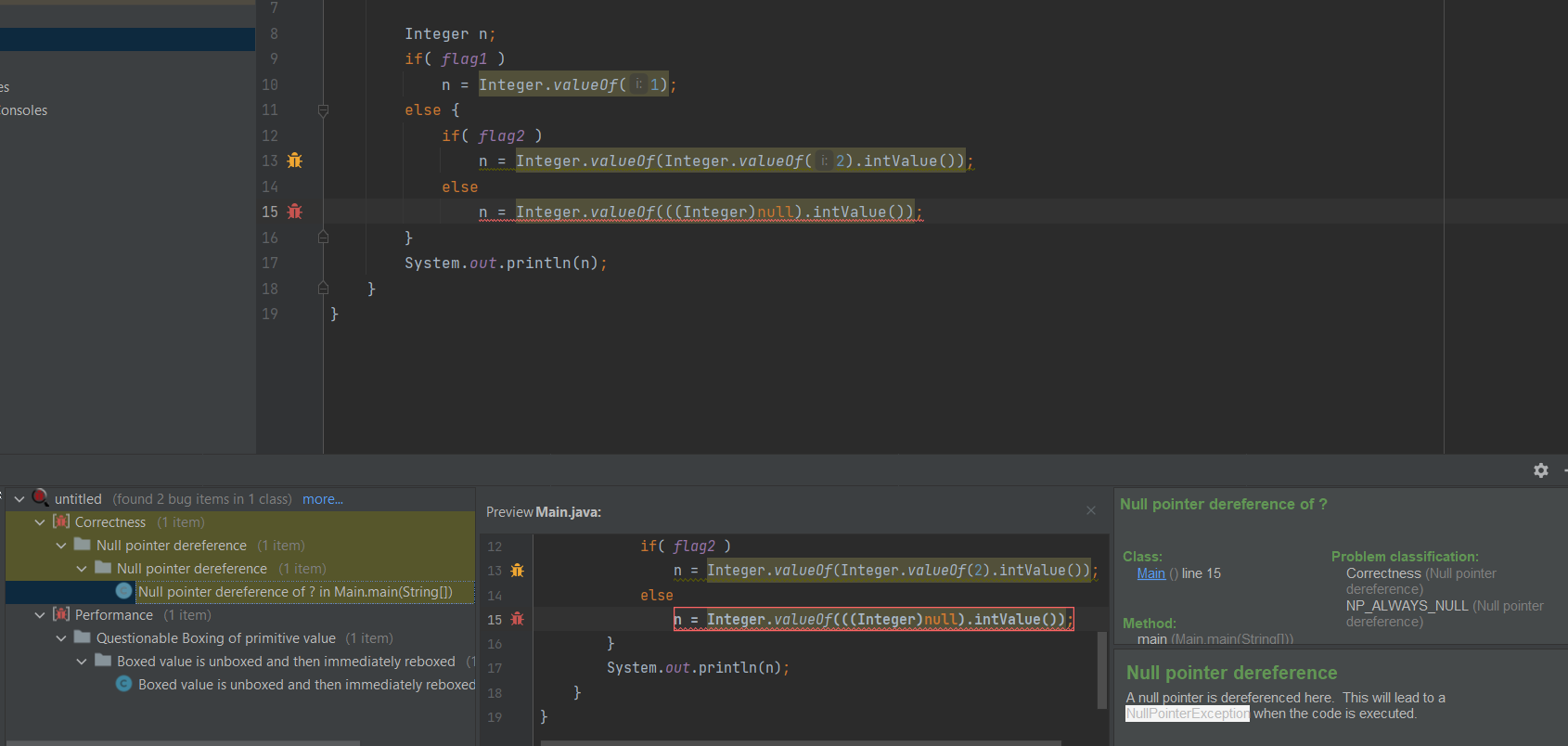


Рисунок 2 – работа SpotBugs с 2 примером.

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

* BX\_UNBOXING\_IMMEDIATELY\_REBOXED;

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

* NP\_NULL\_ON\_SOME\_PATH.

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

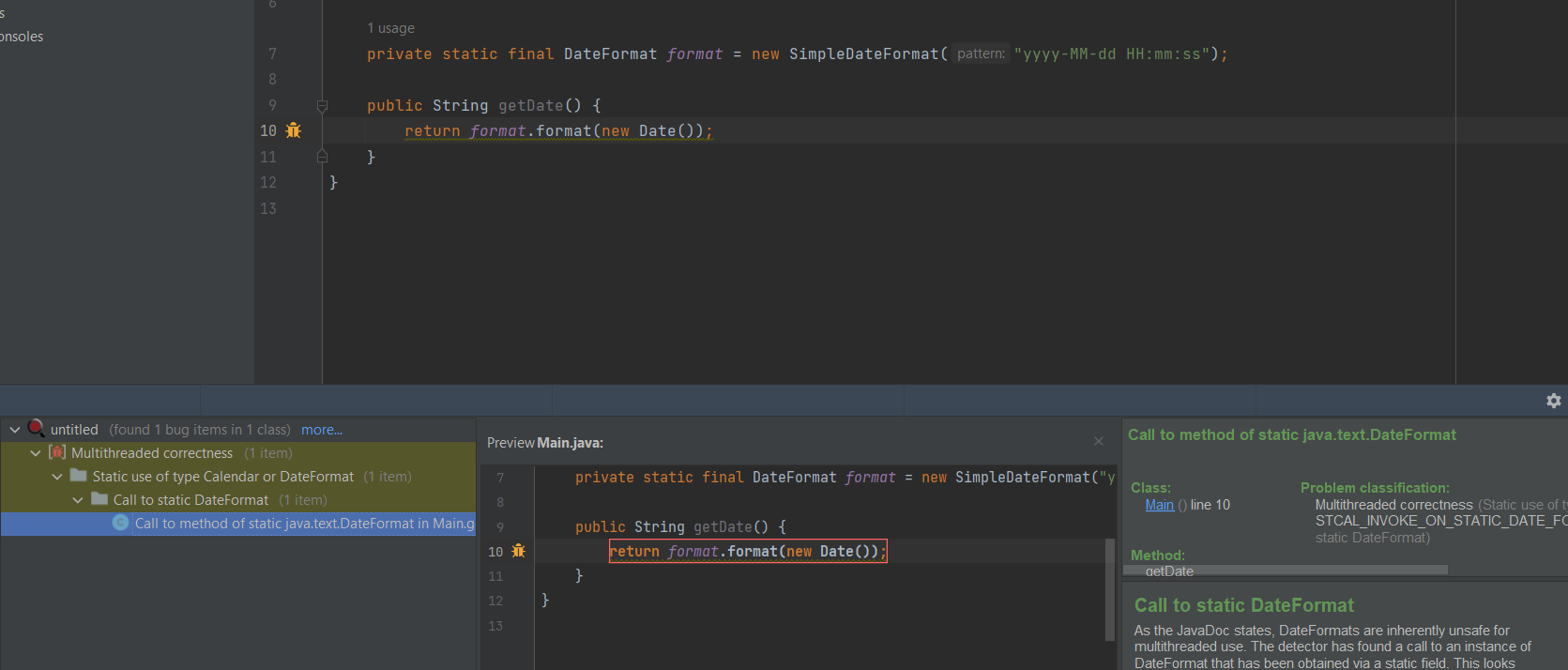


Рисунок 3 – работа SpotBugs с 3 примером.

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:

* 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.

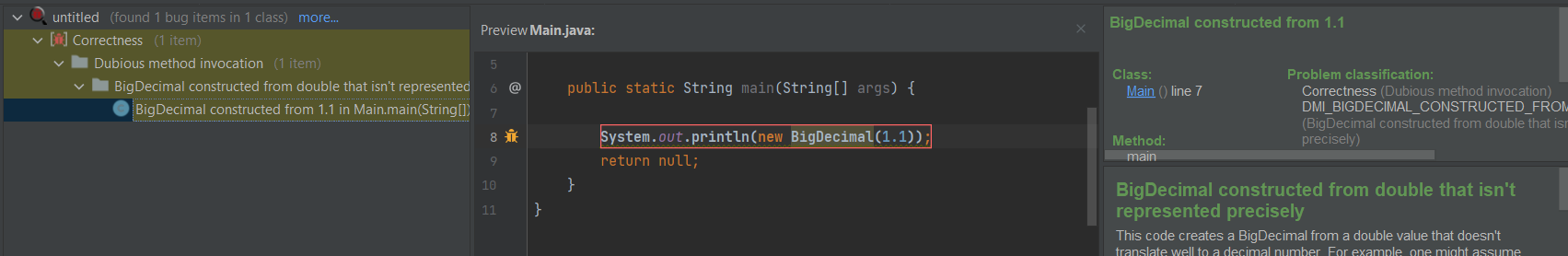


Рисунок 4 – работа SpotBugs с 4 примером.

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:

* 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).

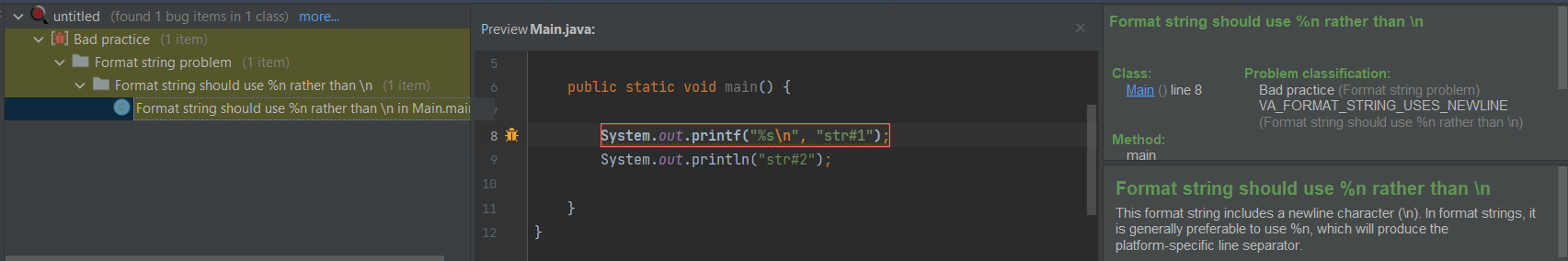


Рисунок 5 – работа SpotBugs с 5 примером.

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:

* 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.

**3 Анализ результатов**

В данной работе были разработать программы на java для каждого из разобранных примеров.

Также были проверены с помощью выбранного статического анализатора кода SpotBugs.

Данная работа позволила изучить работу с анализаторами кода и применить знания на практических примерах.