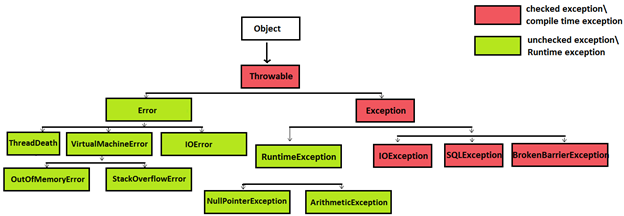
1. **Что такое исключение.**

Исключение в Java — это объект, который описывает исключительное состояние, воз­никшее в каком-либо участке программного кода. Когда возникает ис­ключительное состояние, создается объект класса Exception. Этот объект пересылается в метод, обрабатывающий данный тип исключительной ситуации. Исключения могут возбуждаться и «вруч­ную» для того, чтобы сообщить о некоторых нештатных ситуациях.

1. **Иерархия исключений.**



Исключения делятся на несколько классов, но все они имеют общего предка — класс Throwable. Его потомками являются подклассы Exception и Error.

Исключения (Exceptions) являются результатом проблем в программе, которые в принципе решаемые и предсказуемые. Например, произошло деление на ноль в целых числах.

Ошибки (Errors) представляют собой более серьёзные проблемы, которые, согласно спецификации Java, не следует пытаться обрабатывать в собственной программе, поскольку они связаны с проблемами уровня JVM. Например, исключения такого рода возникают, если закончилась память, доступная виртуальной машине. Программа дополнительную память всё равно не сможет обеспечить для JVM.

В Java все исключения делятся на два типа: контролируемые исключения (checked) и неконтролируемые исключения (unchecked), к которым относятся ошибки (Errors) и исключения времени выполнения (RuntimeExceptions, потомок класса Exception).

Контролируемые исключения представляют собой ошибки, которые можно и нужно обрабатывать в программе, к этому типу относятся все потомки класса Exception (но не RuntimeException).

1. **Как создать/бросить/поймать исключение.**

Чтобы создать свой класс исключений, надо унаследовать его от класса Exception.

В Java есть пять ключевых слов для работы с исключениями:

try – данное ключевое слово используется для отметки начала блока кода, который потенциально может привести к ошибке.

catch – ключевое слово для отметки начала блока кода, предназначенного для перехвата и обработки исключений.

finally – ключевое слово для отметки начала блока кода, которой является дополнительным. Этот блок помещается после последнего блока ‘catch’. Управление обычно передаётся в блок ‘finally’ в любом случае.

throw – служит для генерации исключений.

throws – ключевое слово, которое прописывается в сигнатуре метода, и обозначающее что метод потенциально может выбросить исключение с указанным типом.

1. **В чем разница между checked и unchecked исключениями.**

Все исключительные ситуации делятся на «проверяемые» (checked) и «непроверяемые» (unchecked).

* Checked исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода. Unchecked могут не обрабатываться и не быть описанными.
* Unchecked исключения в Java — наследованные от RuntimeException, checked — от Exception (не включая unchecked).

Checked исключения отличаются от Unchecked исключения в Java, тем что:

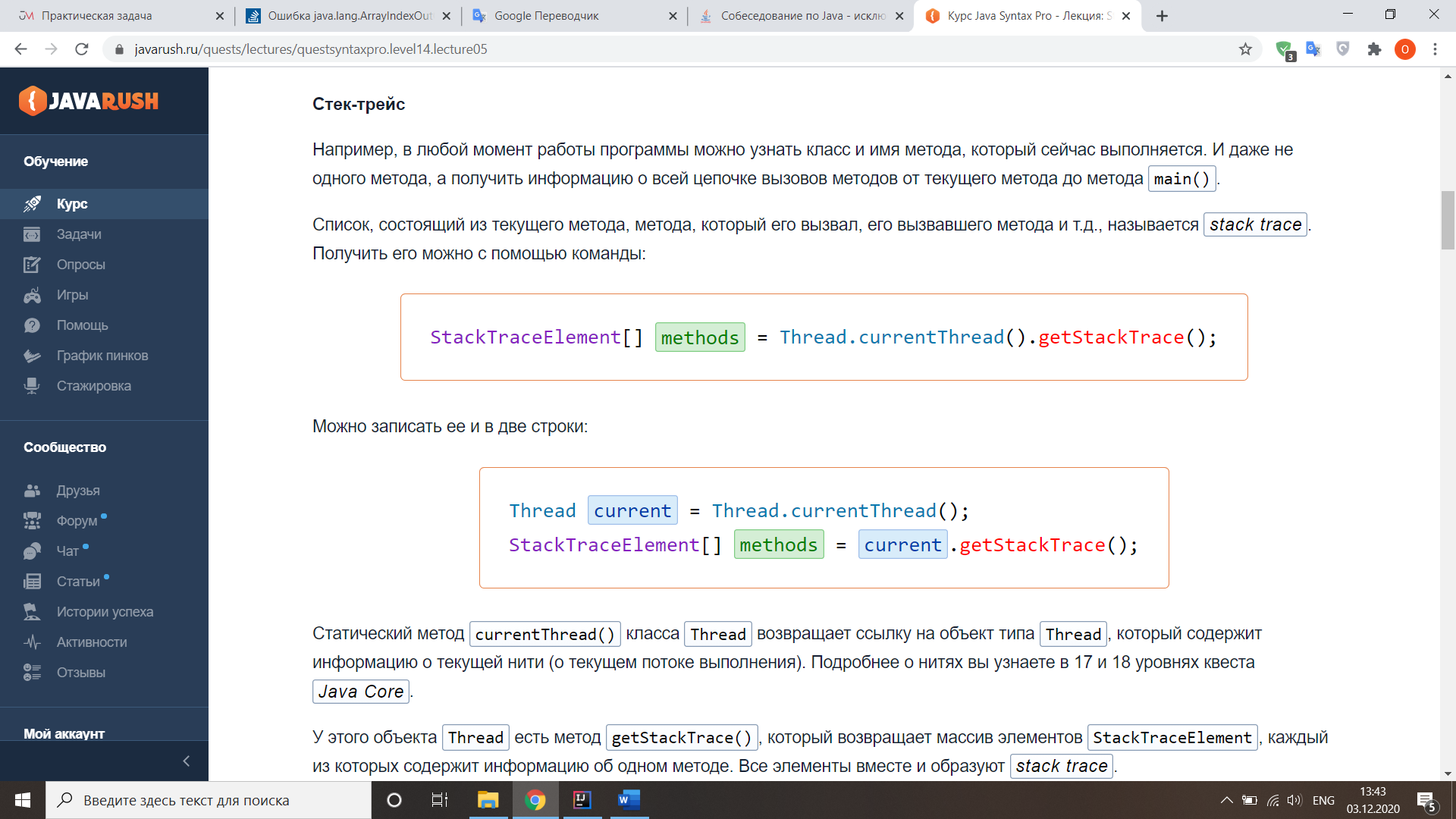
1)Наличие\обработка Checked исключения проверяются на этапе компиляции.

2) Наличие\обработка Unchecked исключения происходит на этапе выполнения.

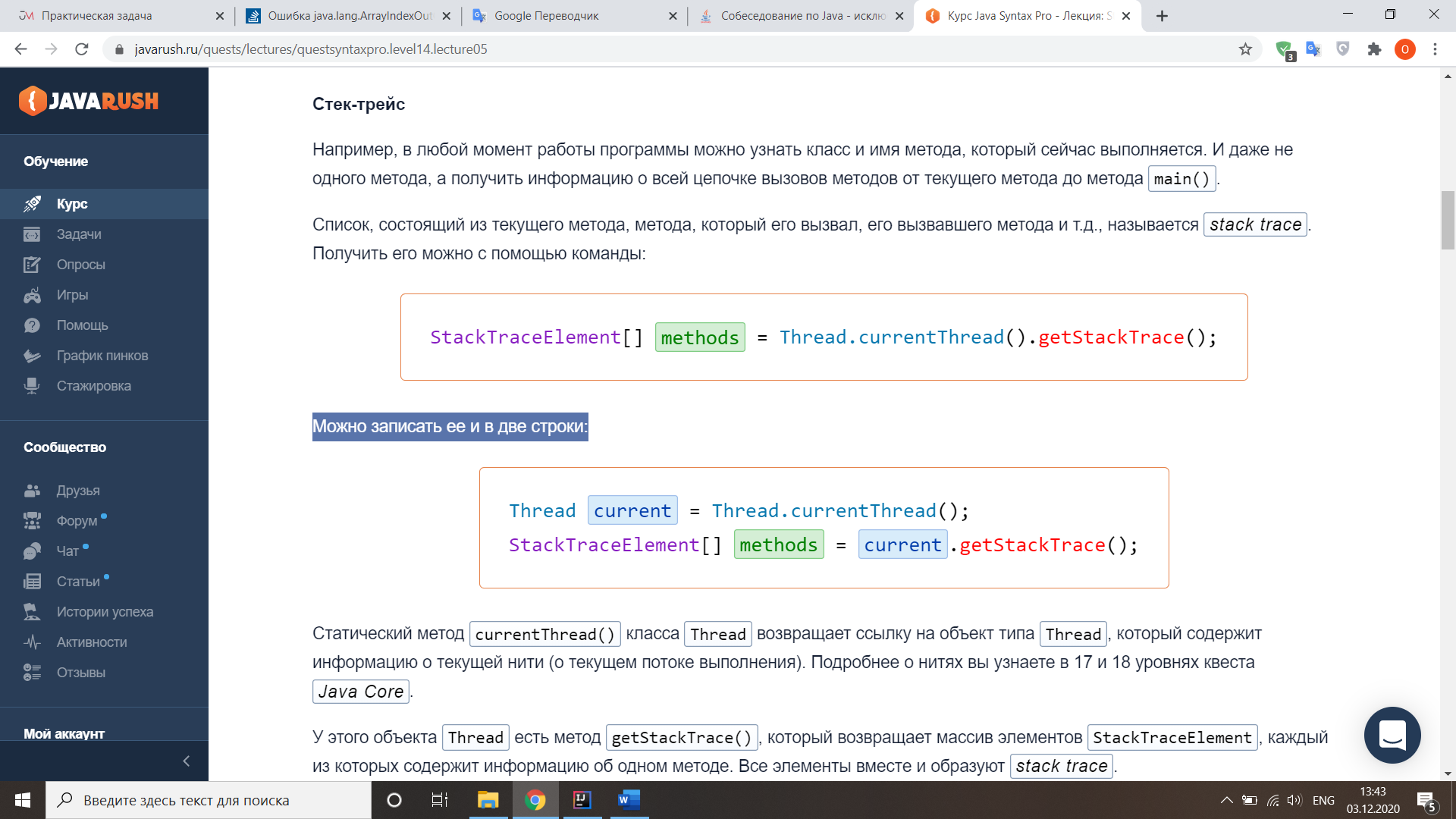
1. **Что такое стектрейс. Какую информацию из него можно получить?**

Например, в любой момент работы программы можно узнать класс и имя метода, который сейчас выполняется. И даже не одного метода, а получить информацию о всей цепочке вызовов методов от текущего метода до метода main().

Список, состоящий из текущего метода, метода, который его вызвал, его вызвавшего метода и т.д., называется stack trace. Получить его можно с помощью команды:



Можно записать ее и в две строки:

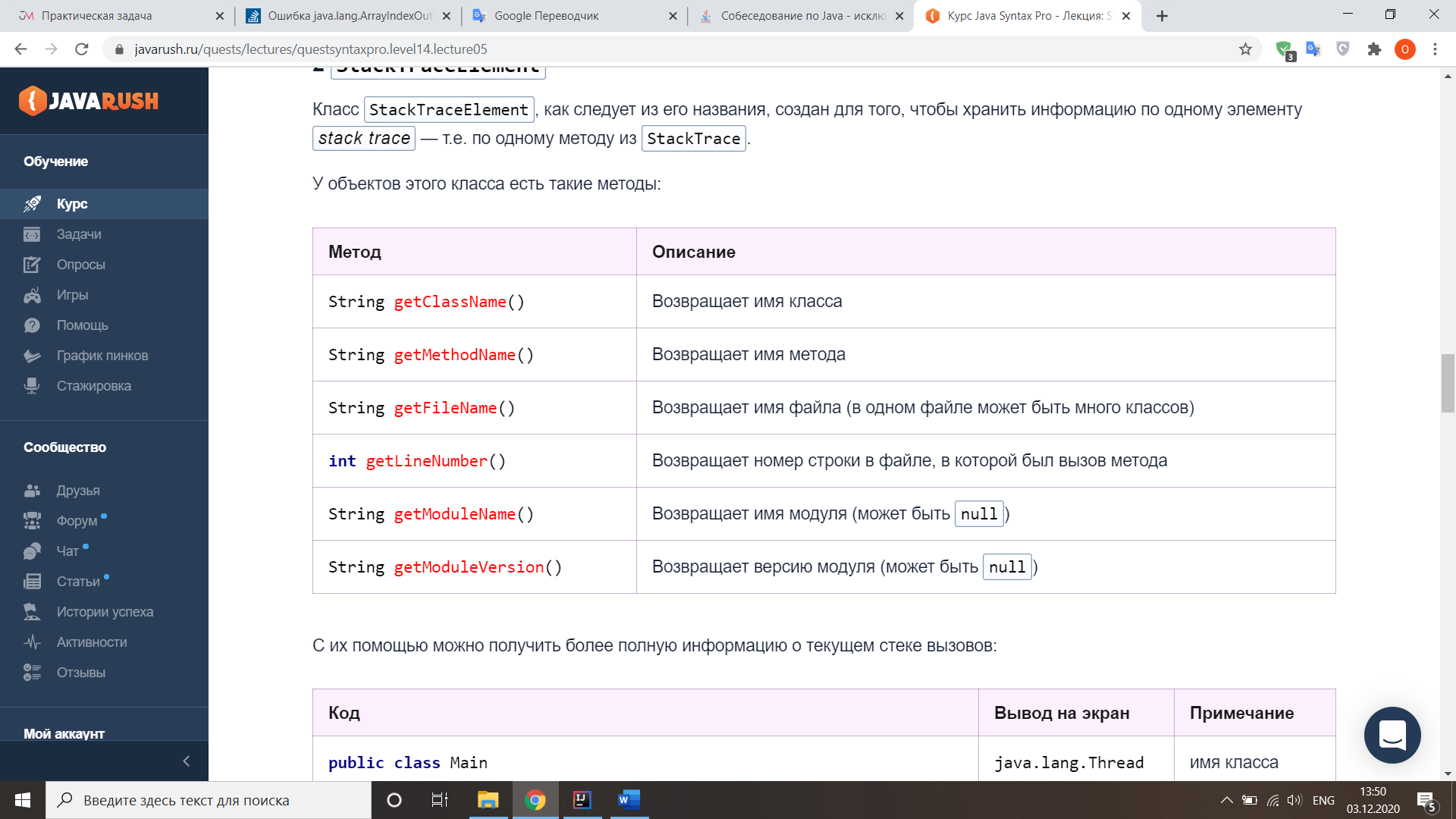


Статический метод currentThread() класса Thread возвращает ссылку на объект типа Thread, который содержит информацию о текущей нити (о текущем потоке выполнения).

У этого объекта Thread есть метод getStackTrace(), который возвращает массив элементов StackTraceElement, каждый из которых содержит информацию об одном методе. Все элементы вместе и образуют stack trace.

Класс StackTraceElement, как следует из его названия, создан для того, чтобы хранить информацию по одному элементу stack trace — т.е. по одному методу из StackTrace.

У объектов этого класса есть такие методы:



Что такое Stack Trace вы уже знаете, а что же такое сам Stack (Стек)?

Стек — это структура хранения данных, в которую можно добавлять элементы и из которой можно забирать элементы. Причем брать элементы можно только с конца: сначала последний добавленный, потом — предпоследний, и т.д.

Само название Stack переводится с английского как «стопка» и очень похоже на стопку бумаги. Если вы положите на стопку бумаги листы 1, 2 и 3, взять вы их сможете только в обратном порядке: сначала третий, затем второй, а только затем первый.

1. **блок finaly**

Когда исключение передано, выполнение метода направляется по нелинейному пути. Это может стать источником проблем. Например, при входе метод открывает файл и закрывает при выходе. Чтобы закрытие файла не было пропущено из-за обработки исключения, был предложен механизм finally.

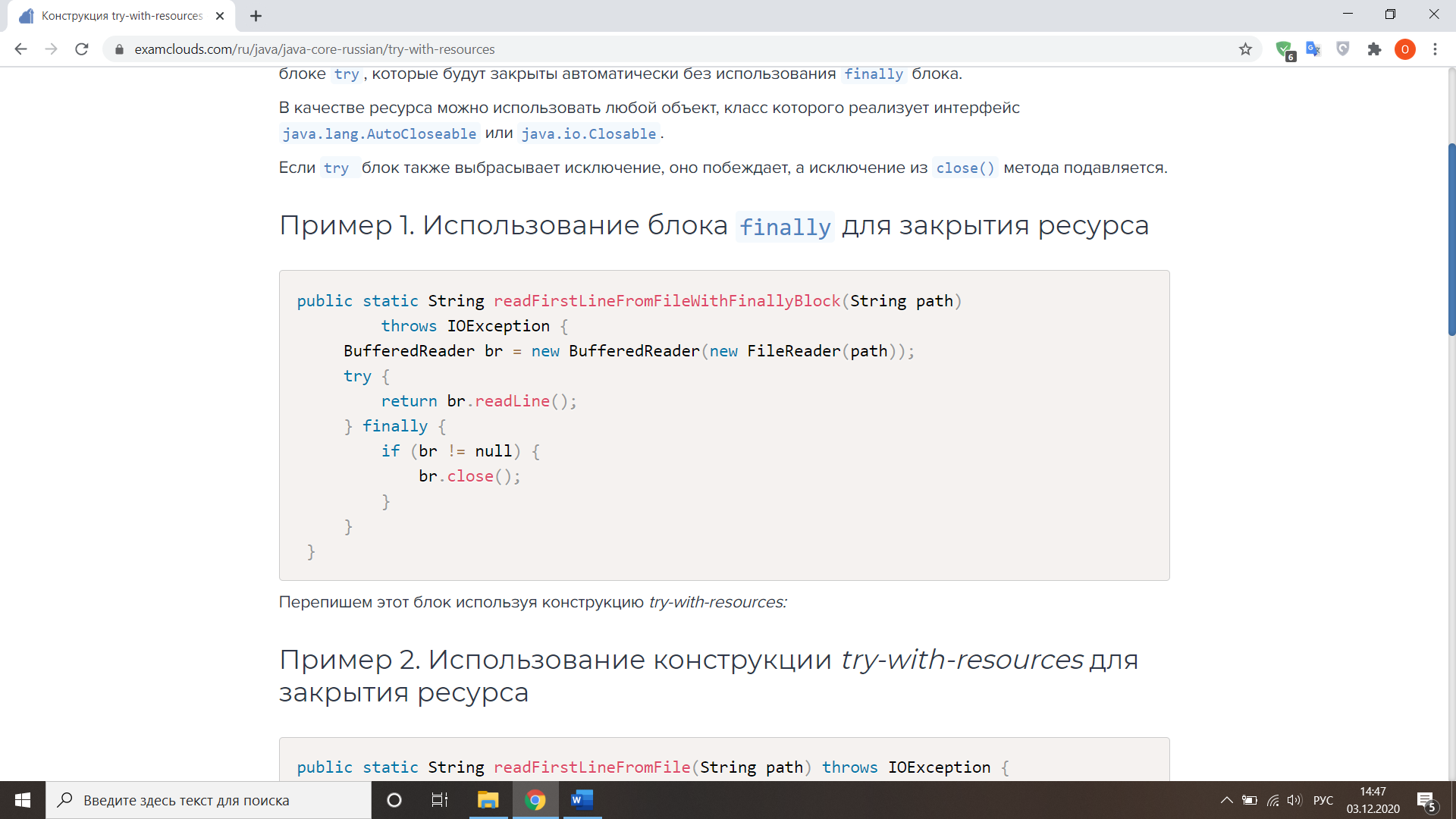
Ключевое слово finally создаёт блок кода, который будет выполнен после завершения блока try/catch, но перед кодом, следующим за ним. Блок будет выполнен, независимо от того, передано исключение или нет. Оператор finally не обязателен, однако каждый оператор try требует наличия либо catch, либо finally. Код в блоке finally будет выполнен всегда.

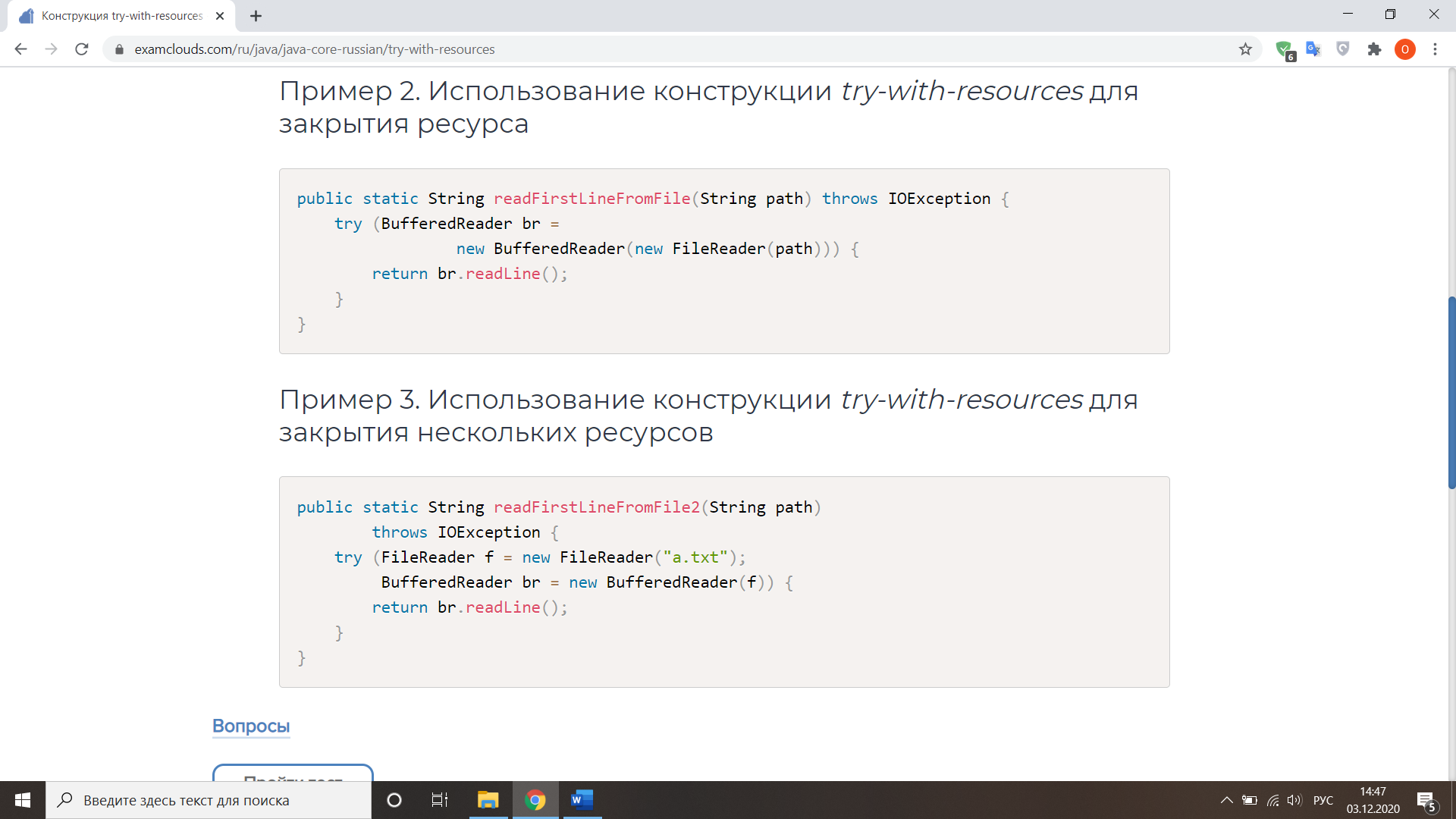
1. **Конструкция try-catch-with-resource.**

Конструкцию try-with-resources ввели в Java 7. Она дает возможность объявлять один или несколько ресурсов в блоке try, которые будут закрыты автоматически без использования finally блока.

В качестве ресурса можно использовать любой объект, класс которого реализует интерфейс java.lang.AutoCloseable или java.io.Closable.

Если try блок также выбрасывает исключение, оно побеждает, а исключение из close() метода подавляется.





1. **Что такое логгирование и для чего используется.**

Логирование это процесс записи каких-либо событий, которые происходят в коде. Это твоя обязанность как программиста — записать, что сделал твой код, потому что потом тебе же эти логи и дадут для разбора. Если все сделать хорошо, тогда любая бага будет очень быстро разобрана и устранена.

Тут, наверное, не буду углубляться в то, какие логеры есть. В данной статье ограничимся простым java.util.Logger: его более чем достаточно для знакомства.

Каждая запись лога содержит дату-время, уровень события, сообщение. Дата-время проставляется автоматом. Уровень события выбирает автор сообщения. Уровней есть несколько. Основные — это info, debug, error.



Дальше — настройки. Логерам можно указать, куда писать (в консоль, файл, jms или ещё куда-либо), указать уровень (info, error, debug...).

**2. Что если в конструкции try finally вылетело исключение сначала в try а потом в finally? Какое исключение вылетит? Что будет с другим?**

Оператор try с блоком finally выполняется, сначала выполнив блок try. Тогда есть выбор:

Если выполнение блока try завершается нормально, [...]

Если выполнение блока try завершается внезапно из-за throw значения V, [...]

Если выполнение блока try завершается внезапно по любой другой причине R, выполняется блок finally. Тогда есть выбор:

Если блок finally завершается нормально, то оператор try завершается внезапно по причине R.

Если блок finally завершается внезапно для разума S, то оператор try совершает внезапно по причине S (и причина R отбрасывается).

**3. Когда происходит закрытие ресурса в конструкции try-with-resources если в try возникло исключение: до перехода в catch или после того как catch отработает?**

 блоки catch и явный finally выполняются уже после того, как закрываются ресурсы

**6. Когда будет выполнен finally? Когда не будет выполнен? Будет ли выполнен finally при Error?**

Случаи, когда finally не будет вызван, это:

1. Если вы вызовете System.exit()

2. Если JVM вылетает первым

3. Если JVM достигает бесконечного цикла (или какого - либо другого не прерываемого, не завершающего оператора) в блоке try или catch

4. Если OS принудительно завершает процесс JVM; например, kill -9 <pid> на UNIX

5. Если хост-система умирает; например, сбой питания, аппаратная ошибка, паника OS и т. д.

6. Если блок finally будет выполняться потоком-демоном, а все остальные потоки, не являющиеся демонами, завершатся до вызова finally

**7. Что такое подавленные исключения?**

Подавленные исключения -это дополнительные исключения, возникающие в операторе try-with-resources (введенном в Java 7) при закрытии ресурсов AutoCloseable . Поскольку при закрытии ресурсов AutoCloseable может возникать несколько исключений, дополнительные исключения присоединяются к основному исключению как подавленные исключения .

**8. Может ли main выбрасывать исключения, что будет происходить?**

Может и оно будет передано в виртуальную машину Java (JVM).

**3. Можно ли в сигнатуре метода в throws указать непроверяемое исключение?**

можно

**4. Какие есть уровни логирования и для чего они нужны?**

Для того, чтоб как-то ранжировать логи, нужно было дать определенные обозначения и разграничения. Для этого ввели уровни логирования.

**5. Обязательно ли передавать в метод getLogger() имя класса? Почему так принято?**

Вы всегда можете использовать любую строку как имя журнала, кроме типа класса. Это определенно нормально.

Причина, по которой многие люди используют тип класса, я думаю:

* Прост в использовании. Вам не нужно беспокоиться о дублировании имен журналов в сложном приложении Java EE. Если другие пользователи также используют ваше имя регистратора, у вас может быть файл журнала, включающий не только вывод вашего класса, но
* Легко проверить класс ведения журнала, так как имя журнала будет отображаться в файле журнала. Вы можете быстро перейти к определенному классу;
* Когда вы распространяете свой класс, люди могут захотеть перенаправить ведение журнала из вашего класса в конкретный файл или в другое место. В этом случае, если вы используете специальное имя регистратора, нам может потребоваться проверить исходный код или сделать невозможным его, если souce недоступен.

**6. Сообщения каких уровней мы увидим, задав уровень INFO?**

**8. Try с ресурсами: что если исключение вылетело сначала в try, а потом в close в конструкции try-with-recources? Какое исключение вылетит? Что будет с другим?**

Если исключение будет выброшено в основном коде и в методе close(), то приоритетнее будет первое исключение, а второе исключение будет подавлено, но информация о нем сохранится (с помощью метода Throwable.addSuppressed(Throwable exception), который вызывается неявно Java компилятором):

**9. Как достать подавленное исключение?**

Подавленные исключения могут быть извлечены из основного объекта исключения, вызвав getSuppressedExceptions .

**11. Расскажи про информацию, которая находится внутри исключения? Как с ней работать?**

* getMessage(). Этот метод возвращает строку, которая была первым параметром при создании исключения;
* getCause() возвращает исключение, которое стало причиной текущего исключения;
* printStackTrace() печатает stack trace, который содержит информацию, с помощью которой можно определить причину исключения и место, где оно было брошено.

**12. Как создать свое исключение? Зачем создавать свои?**

Хотя имеющиеся в стандартной библиотеке классов Java классы исключений описывают большинство исключительных ситуаций, которые могут возникнуть при выполнении программы, все таки иногда требуется создать свои собственные классы исключений со своей логикой.

Чтобы создать свой класс исключений, надо унаследовать его от класса Exception. Например, у нас есть класс, вычисляющий факториал, и нам надо выбрасывать специальное исключение, если число, передаваемое в метод, меньше 1

**13. В чем разница между AutoCloseable и Closeable в try witch resourses?**

До Java 7 уже существовал похожий интерфейс – Closeable. Смысл его точно такой же. Он всё еще доступен в стандартной библиотеке для обратной совместимости, но в новом коде рекомендуется использовать AutoCloseable. Чтобы экземпляры старого Closeable тоже можно было использовать в try-with-resource, новый интерфейс был добавлен его родителем.  
  
Проблема старого интерфейса Closeable была в узости типа исключений, которые может выбрасывать close(). [Ковариантность](https://itsobes.ru/JavaSobes/chto-takoe-kovariantnost-i-kontravariantnost) позволила расширить тип в базовом интерфейсе AutoCloseable с IOException до Exception.  
  
Еще одно отличие – контракт метода close(). Старый Closeable требует его [идемпотентности](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BC%D0%BF%D0%BE%D1%82%D0%B5%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C), тогда как новый AutoCloseable разрешает методу иметь побочные эффекты.

Идемпоте́нтность — свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом.

**14. Можно ли выбросить (throw) исключение в конструкторе? Если да или нет, то почему."**

Выбрасывание исключений в конструкторе - неплохая практика. Фактически, это единственный разумный способ для конструктора указать, что есть проблема; например что параметры недействительны.

Однако явно декларирование или метание java.lang.Exception почти всегда плохое.

Вы должны выбрать класс исключений, который соответствует исключительному состоянию, которое произошло. Если вы выбрали Exception, вызывающему пользователю сложно отделить это исключение от любого количества других возможных объявленных и необъявленных исключений. Это затрудняет восстановление ошибок, и если вызывающий абонент выбирает распространение Исключения, проблема просто распространяется.

Закроются ли ресурсы в try с ресурсами, если во время закрытия произойдет исключение?

Могут не закрыться

1. **Можно ли так написать**  
try {  
   throw new Object();  
}

нельзя

**4. Можно ли обработать Error, брошенный вручную?**

нельзя **5. Расскажи, как правильно ловить исключения? Иерархия catch блоков**

Сначала более точные а потом общие **6. Можно ли одном catch обработать несколько исключений?**

Однако помните, что если все исключения относятся к одной и той же иерархии классов, вы можете просто поймать этот базовый тип исключения.

Также обратите внимание, что вы не можете поймать оба ExceptionA и ExceptionB в одном блоке, если ExceptionB наследуется, прямо или косвенно, от ExceptionA. **7. Try с ресурсами: что если исключение вылетело сначала в try, а потом в close в конструкции try-with-recources? Какое исключение вылетит? Что будет с другим?**

Если исключение будет выброшено в основном коде и в методе close(), то приоритетнее будет первое исключение, а второе исключение будет подавлено, но информация о нем сохранится (с помощью метода Throwable.addSuppressed(Throwable exception), который вызывается неявно Java компилятором):